

Manager

JOURNEY MANAGER

TEMENOS

Information in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form or by any means, for any purpose, without the express written permission of TEMENOS HEADQUARTERS SA. © 2021 Temenos

TOC

Journey Manager Overview **46**

Journey Manager Features **50**

Main Features 50

Features by Release 50

22.04 51

21.11 51

21.05 51

20.11 52

20.05 52

19.11 53

19.05 54

18.11 54

18.05 56

17.10 56

5.1 57

5.0 59

4.3 59

Unified Application Data Document Model **61**

When to use Unified App Data 62

When to use Transaction Properties 62

Save Challenge **64**

Journey Platform Integration **65**

Integration Areas 65

Transactions Discovery 65

Data Pre-population 66

Data Entry Streamlining 66
Execution 67
Save and Resume 67
Delivery 68
Authenticated and Unauthenticated Transactions 68
Extension Points 69
Calling External Services and API 73
Web Site Integration and Single Signon 74
Form Level Integration 75
Integrate Manager with SharePoint 76
Five Journey Platform Integration Areas 78
Delivery 78
Form Discovery 81
3 Pre-Population - Accelerating Data Entry 82
Streamlining Data Entry 82
Execution 83
The Economics of Integration 85
Why do we do integration? 85
Assess the Value of Integration 86
Consider Deferring Integration 86
Consider Leveraging Existing Paper/image-based Processes 86
Hidden Costs of Process Re-engineering 87
Google Analytics Virtual Page Integration 88
Add GA Code to Head of Page 88

Create Business Rule to call the ga method on page change 88

Add Google Tag Manager Support 92

Importing the Container into Google Tag Manager 93

Configure Manager to Enable Analytics 95

Enable Analytics for each Form 96

Add GTM container script to the form 96

Additional Forms Pages 97

Configure Google Tag Manager 99

Further Information 100

Navigate and Locate Items in GTM 100

General Functional Overview 101

In-form Page Views 101

Other Page Views 101

Cloud hosted Journey Platform VDC Editions 103

Journey Manager Home Dashboard 106

Recent Forms 107

Recent Services 107

Latest Transactions 107

Completed Transactions By Form 108

Transaction Trend 108

Journey Manager Modules 110

Journey Manager 3rd Party Libraries 112

Licenses 112

Libraries 112

Installation Overview **123**

Download the JM Installation Guide PDF 123

Plan the Installation 124

Organizations Overview **125**

View Organizations **127**

Create an Organization **129**

Copy an Organization **131**

Remove an Organization **133**

Export an Organization **134**

Import an Organization **135**

Migrate an Organization **138**

Configure an Organization **139**

Assign Form Spaces to an Organization **140**

Configure an Organization Payment Gateway **142**

Configure Organization Properties **144**

Configure Organization Metadata Tags **145**

Configure Organization Reference Data **146**

Test Organization Reference Data **149**

Query Syntax 150

Configure Organization Form Categories **152**

Create an Application Package **153**

Configure an Application Package for an Organization **155**

Export an Application Package **158**

Import an Application Package **159**

Configure Organization Security **162**

Form Spaces Overview **164**

Structure of Form Space WAR File 165

Default Form Spaces 165

Salesforce 165

TransactField App 166

Web Plug-in 166

Work Space 166

Pre-configured Form Spaces 166

Work Spaces 166

Maguire 166

View Form Spaces **167**

Create a Form Space **169**

Configure a Form Space **171**

Configure a Form Space Status **174**

Assign Organizations to a Form Space **177**

Assign Form Spaces to a Form **179**

Remove a Form Space **181**

Export Form Spaces **182**

Import Form Spaces **183**

Migrate Form Spaces **186**

Export a Form Space as a WAR Archive **188**

Import a Form Space as a WAR archive **189**

Standard Form Space Properties **191**

Cache Headers 191
CORS Headers 191
Security Headers 192
System Wide Settings 193
Organization Level Settings 193
LinkedIn Example 193
X-Frame-Options 194
Create Form Space Properties 195
Configure Form Space Properties 196
Configure Workspaces Properties 201
Configure a Form Space Email Template 203
Configure Form Space Pages 205
View Form Space Pages 207
Create a Form Space Resource 209
Configure a Form Space Resource 211
View Form Space Resources 214
View Forms Hosted on a Form Space 217
Form Space Hot Deployment 218
Web Plug-In Form Space 219
Host an Embedded Form 221
How does it work? 221
Step-by-step guide 221
Explanation of Form Page Content 223
Map a Template to a Model 225

How to Unlock a Transaction **227**

Upgrade a Form Space **228**

Rebuilding Spaces 228

Content Only Spaces 228

Custom Java Spaces 228

Important Points when re-building and deploying a Space: 228

Styling and Mark-up 229

Work Space **231**

Work Space Login **233**

Work Space Change Password **236**

Work Space 2 Factor Authentication **238**

Work Space Home **240**

Work Space Forms **242**

Work Space Tasks **243**

Work Space History **244**

Work Space Reviews **245**

Work Space Help Desk **246**

Anonymous Users Only 246

Anonymous and Authenticated Users 247

Work Space Account **249**

Configure Work Space Properties **253**

Configure Work Space Page Markup **255**

Form Dashboard **259**

Form Details 259

Form Versions 259
Form Spaces 260
Latest Transactions 261
Deploy Forms **262**
View Forms **263**
Configure a Form **266**
Configure Form Details **267**
Configure Form Abandonment **270**
Configure Form Email Verification **272**
Configure Form Flow **274**
Configure Form Group Access **277**
Configure Form Page Tracking **279**
Configure Form Test Mode **281**
Render a Form **285**
Test a Form **286**
Import Forms **287**
Import a Form Application **292**
Export Forms **294**
Remove a Form **295**
Make a Form Inactive 295
Remove a Form 296
Migrate Forms **298**
Rollback Forms **299**
Save Challenge **301**

Maestro Form Static Content Externalization **302**

Server Monitor Test Form **303**

Integrate Forms to Existing Websites **304**

iFrame Embedding 304

Pros 304

Cons 304

<DIV> Embedding 304

Pros 304

Cons 304

Launch a new tab or window 305

Pros 305

Cons 305

Configure the Form Display Mode 305

Form URLs 305

Advanced Debugging of Maestro Forms **307**

Recommended Skill Set 307

Debugging Tools 307

Modern Browser 307

Developer Tools Enabled 307

AngularJS Browser Plugin 307

Maestro Form Architecture 307

Form View 308

Form Items 308

Form Data 309

Publishing Forms for Debug 309
Activating a Debug Session 310
Shortcuts for Republishing Forms 311
Inspecting the Form Object 311
Logging an Item to the Console 313
Locating a JavaScript Rule for Debug 313
Rule Types and JavaScript Function Naming 315
Examples 316
Change the Maestro Form Description 317
Change a Maestro Form Version Description 318
Method 1 318
Method 2 319
Maestro Data Models 321
Data Mapper 324
Data XML Location 326
Component XML Name and Entity XML Path 328
ID without an Entity 328
ID with an Entity 329
Form XML Data Payload 330
Submission Data Extract Overview 332
View Form Submission Data Extract 334
Configure Component Data Extract 336
Form Data Configuration Overview 338
View Form Data Configuration 339

Form Configuration Mapping **340**

Form XML Data **344**

Form Seed **348**

Form Data Extract Mappings **351**

Collaboration Job Contex 354

AvailableRoutes 354

ReferenceNumber 355

StepName 355

RouteName 355

Test a Form Data Extract Mapping **357**

Form Prefill Options **358**

Property prefill mappings: client, form, user properties 358

XML prefill mappings: external XML file containing prefill data 358

Parameter prefill mappings: HTTP request parameters, session attributes or cookies 358

Pros: 358

Cons: 358

Form Input XML Prefill Mapping **360**

Form Input XML Prefill Mapping Using Drag and Drop **363**

Form Input XML Prefill Mapping Using XPath **365**

Form Property Prefill Mapping **367**

Form Property Prefill Mapping Using Drag and Drop **370**

Form Property Prefill Mapping Using XPath **372**

Form Request Parameter Prefill Mapping **375**

GET vs POST for Request Parameter Prefill Mapping 377

GET Requests 377
POST Requests 378
Prefilling Product Information 378
Prefilling Customer Information 378
Add Repeating Content from XML Prefill Mapping Dynamically 380
Form Prefill for Anonymous Users 382
URL Parameter passing on a GET request 382
Pros 382
Cons 383
HTTP request headers on a POST request 383
Pros 383
Cons 383
Encrypted Cookies 384
Pros 384
Cons 384
Short life tokens (SLT) 384
Pros 386
Cons 386
Encrypted String 386
Pros 388
Cons 388
Launch a Form From Another Form 389
Technique 389
1. Search form 389

Main form 390
Alternatives 390
Restrictions 390
Prefill a Cascading Dropdown in Composer **391**
Form Versions Overview **395**
View Form Versions **396**
Configure a Form Version **398**
Configure a Form Version Metadata **402**
Form Version Services **403**
Form Version Properties **405**
Form Categories **410**
Form Design Info **412**
Job Properties of a Form Version **414**
Job Information of a Form Version **416**
Delete a Form Version **417**
Delivery Channels Overview **418**
Delivery Process 418
Email 419
Email Secure 419
REST Service 419
Web Service 419
Trash Can 420
Create a Delivery Channel **421**
Create an Email Secure Delivery Channel **427**

Create a Web Service Delivery Channel 429
Create a REST Delivery Channel 430
View Delivery Channels 431
Associate a Delivery Channel With a Form 432
Remove a Delivery Channel 435
View Transactions Delivery 436
Service Connection Overview 438
Service Connection Types 440
View Service Connections 441
Create a Service Connection 443
Configure a Service Connection 445
Remove a Service Connection 446
Migrate Service Connections 447
Export Service Connections 448
Import Service Connections 449
AWS KMS Service Connection 451
AWS S3 Service Connection 453
AWS SQS Service Connection 455
AWS STS Service Connection 457
Azure Service Connection 459
ClamAV Virus Scan Service Connection 461
Symantec Virus Scan Service Connection 462
HTTP Endpoint Service Connection 463
DocuSign Service Connection 465

Integration Gateway Service Connection **467**

Salesforce Service Connection **468**

Services Overview **470**

Service Types **473**

View Services **484**

Create a Service **487**

Configure a Service **489**

Service Types **491**

Configure Service Parameters **502**

Configure Service Parameters Edit **504**

Service Usage **506**

Copy a Service **507**

Remove a Service **509**

Deactivate a Service 509

Remove a Service 509

Migrate Services **511**

Export Services **512**

Import Services **513**

Core Global Services Overview **516**

View Core Global Services **517**

A/B Testing Form Version Selector Service **519**

Database Submission Data Storage Service **521**

File System Submission Data Storage Service **524**

Amazon Submission Data Storage Service **527**

Azure Submission Data Storage Service **531**

System Event Publisher Service **535**

AWS SQS System Event Publisher Service **540**

Azure System Event Publisher Service **541**

Chromium Puppeteer Receipt Service **542**

Composer Render HTML Form Service **545**

Current Version Selector Service **547**

Data Retention Management Service **548**

Dynamic PDF Receipt Service **552**

Dynamic PDF Receipt 2 Service **555**

Form Submission Access Controller Service **558**

Payment Reminder Service **561**

Random Tracking Number Service **562**

Status Management Service **563**

Submission Receipt Render Service **565**

Transaction Processor Service **567**

Attachment Virus Scanning Service **571**

ClamAV Attachment Virus Scanning Service **572**

Symantec Attachment Virus Scanning Service **574**

Fluent Attachment Virus Scanning Service **576**

Dynamic Data Service **578**

Transact Fluent API 579

Core Transact API 579

Transact Fluent API 580

Core Transact API 580

Submission Completed Processor Service 582

Task Cancellation Processor Service 582

Job Services Overview 587

Job Definitions 588

jobDetails 588

jobGroups 589

steps 589

actions 591

properties 592

routes 594

View Job Services 596

Configure Job Definition 598

Job Definitions 600

jobDetails 600

jobGroups 601

steps 601

actions 603

properties 604

routes 606

View Job Info 608

Job Controller Service 609

View Job Controller Services 612

Job Action Services 613

Job Service Templates **618**

Job Task Wait Action Services **620**

Open the TaskUserWebService in SOAP UI **625**

Scheduled Jobs Overview **630**

Trigger Types 630

Simple 630

Cron 630

Create a Scheduled Job **631**

Standard Scheduled Jobs **634**

Manage Scheduled Jobs **636**

View Scheduled Job History **639**

Collaboration Job Controller **642**

Receipts Overview **644**

Configure Receipt Numbers and Tracking Codes **646**

Configure Receipts Delivery Mode **648**

Configure a Receipt Server Node **649**

Receipt Headers and Footers Overview **650**

Header and Footer Templates 650

Custom Headers and Footers In a Form 651

Receipt Headers and Footers Customization **655**

Dynamic PDF Receipt 2 655

Receipt Properties 655

Customizing Receipts from the Form 659

Using Header and Footer Templates **663**

How to Control Custom Headers and Footers From a Form **664**

Understanding How Headers and Footers are Rendered **667**

Field Overlay Receipts Using AcroForms **668**

Getting Started 668

Step 1: Add Data Extracts to form fields 669

Step 2: Export the form from Maestro 670

Step 3: Publish the form to Manager 670

Step 4: Create a fillable PDF with Adobe Acrobat Pro DC 671

Step 4.1: Insert a checkbox 672

Step 4.2: Insert a radio button 673

Step 5: Upload the fillable PDF to Manager 675

Step 6: Configure the Receipt Render Service 676

Step 7: Verify the results 677

Test a Receipt with Receipt Test Harness **679**

PDF Receipt Options **682**

Manager PDF Receipts 682

Pros 682

Cons 683

XFA Receipts 683

Pros 683

Cons 683

Field Overlay Receipts 684

Pros 684

Cons 684

Field Overlay Receipts Using Adobe Acrobat 684
Pros 685
Cons 685
Field Overlay Receipts Using Adobe LiveCycle Designer 685
Pros 685
Cons 685
Field Overlay Receipts Using Texcel 685
Pros 686
Cons 686
Custom Receipt Generation 686
Pros 686
Cons 686
Generate Multiple PDF Receipts from a Single Submission 687
Storing multiple PDF receipts 689
Custom Rendering Service 689
Import an Acrobat PDF Form as a Receipt Template (AcroForms) 692
What is an AcroForm? 692
High Level Overview of steps required 692
Step 2: Creating AcroForms 694
Adobe Acrobat Pro DC (2015 Release) 694
Acrobat XI and below 695
Possible Issues & workarounds 696
Step 3: Correcting automatic conversion issues 696
Automatically Detected Fields 696

Missed Fields 697
Dealing with character-separated fields (Dates / phone numbers with area code) 698
Re-factoring the format of field values 698
Check Boxes 698
Radio Buttons 700
Step 4: Composer Configuration 702
Re-factoring the format of field values 703
Radio Buttons & Check Boxes 704
Step 5: Manager Configuration 705
Extending this solution 706
Insert a Watermark in a PDF Receipt 707
Regenerate a Receipt 709
Collaboration Jobs Overview 710
Collaboration Job Modes 713
Standard Mode 713
Process Immediate Mode 713
Job Services Overview 715
Job Definitions 716
jobDetails 716
jobGroups 717
steps 717
actions 719
properties 720
routes 722

View Job Services **724**

Configure Job Definition **726**

Job Definitions **728**

jobDetails 728

jobGroups 729

steps 729

actions 731

properties 732

routes 734

View Job Info **736**

Job Controller Service **737**

View Job Controller Services **740**

Job Action Services **741**

Job Service Templates **746**

Job Task Wait Action Services **748**

Open the TaskUserWebService in SOAP UI **753**

Steps **758**

Actions **766**

Simple Actions 767

Normal Case 767

Error Case 767

Polling Case 767

Expiry Case 767

Auto Retry Actions 767

Wait Actions 768
Example 1 - Task Step - Standard Mode Processing 768
Job Thread 1 768
Assigns User Action 1 - Task Submission Thread 769
Job Thread 2 769
Example 2 - Multiple Tasks Step - Standard Mode Processing 769
Job Thread 1 769
Assigned User Action 1 - Task Submission Thread 769
Job Thread 2 770
Example 3 - Form Bundle - Process Immediate Mode 770
User Thread 1 - User Submits Initial Form 770
User Thread 2 - Same User Submits Credit Card: 770
Action Properties **772**
Task Types 777
Job Functions 782
Collaboration Job Basics **783**
View Collaboration Jobs **785**
Collaboration Job Details **789**
View Collaboration Job Steps **791**
View Collaboration Job Actions **793**
View Collaboration Job Submissions **797**
View Collaboration Job Events **799**
View Collaboration Job Errors **801**
View Collaboration Job Status History **803**

View Collaboration Job History **805**

Collaboration Job Examples **808**

Configure Forms for Use with Collaboration Jobs **814**

Configure Maestro Forms for Use with Collaboration Jobs **821**

Data Extracts 821

Collaboration Job Components 822

Form Bundle Job Tasks 823

Dropdown Bundle Navigation 823

Job Details Block 824

Job Number 824

Step Number 824

Preview Test Block 824

Route Select 825

Form Rules 825

Using a Radio Button Group for the Reviewer and Manager Decision 829

Change Rule 831

Editability Rule 832

Change Rule 833

Visibility Rule 833

Configure Composer Forms for Use with Collaboration Jobs **836**

Data Extracts 836

Form Data Mappings 837

Collaboration Widgets and Components 838

Collaboration Reference Number and Collaboration Step Name 838

Collaboration Routes Dropdown List 839
Collaboration Submit Button 840
Collaboration Test Block 841
Routing Using the Standard Radio Button Group 842
Visibility and Editability Rules 845
Attachments 847
Configure Routes with Business Rules 849
Customer Onboarding Collaboration Jobs 850
The Workflow 850
Dynamic Customer Onboarding Collaboration Jobs 860
Maestro Customer Onboarding Collaboration Jobs 861
Multi-Step Group Review Collaboration Jobs 862
Configure a Form 862
Configure Manager 864
Configure Users and Groups 871
Route Options 871
Form - Anne 873
Initial Review - Roger 875
Applicant Update - Anne 879
Initial Review - Roger 881
Additional Review - Mike 883
One Step Review Collaboration Job 885
Configure a Form 885
Configure Manager 887

Configure Users and Groups 889
Form - Anne 890
Manager - Administrator 892
Form Space - Anne 893
Form Space - Roger 894
Manager - Administrator 896
Form Space - Anne 897
One Step Anonymous Review Collaboration Jobs 899
Cancel an Anonymous Task in Maestro Forms 903
Step Expiry 908
Configure a Form 909
Configure a Job Controller 909
Configure a Job Expiry Service 911
Form - Start 914
Manager - Action Review Expiry 914
Form Space - Application Escalated 915
Task Assign Repeat Action 917
Assignee 917
AssignRepeatIndex 918
AssignRepeatItem 918
Collaboration Jobs Reuse 920
Form Bundles Overview 921
Sharing Data Between Tasks 924
No Sharing 924

Shared XML Data Model 925
Share Extract Data 925
Show Previous Forms 925
Forms in the Same Step Editable 925
Preconditions 925
Dynamic Preconditions 926
Dynamic Precondition and Action Order 926
User Authentication 927
Form Bundles Context 928
Configure Form Bundles 932
Step Attributes For Form Bundles with More Than Two Tasks 932
Step Attributes For Form Bundles with One or More Tasks 932
Step Properties 932
Action Properties 932
Anonymous Form Bundles 934
Configure a Form 934
Job Definition - Additional Products 935
Example 1 937
Example 2 937
Authenticated Form Bundles 939
User Task 939
Non Claimable Tasks 939
Claimable Group Tasks 939
Job Definition - Additional Products 940

Example 1 942
Example 2 942
Review Task 942
Example 3 942
Form Bundle Flow Example 944
The Flow 944
Form Chaining Overview 955
Change of User 957
Legacy Mode 958
Transact Functions Overview 959
Fluent Functions 959
Delivery Functions 959
How It Works 959
View Transact Functions 962
Create a Transact Function 964
Copy a Transact Function 966
Import a Transact Function 967
Export a Transact Function 969
Backup a Transact Function 970
Rollback a Transact Function 971
Remove a Transact Function 972
Run a Transact Function Unit Test 973
Configure Transact Function Service Definition 976
Edit a Transact Function Groovy Script 979

Configure Transact Function Parameters **983**

Configure Transact Function Parameters Edit **986**

Configure a Transact Function Unit Test **991**

Configure Transact Function Unit Test Parameters **993**

Groovy Console **995**

Groovy Service Log **1000**

Configure Transact Function Triggers **1004**

Transact Function Help Doc **1006**

Server-Side Persistence Pattern **1008**

Storing data 1008

Retrieving data 1008

Some notes 1009

Handling Form Function Exceptions in Maestro Applications **1010**

Journey Manager - the Server Side 1010

Maestro Application - the Client Browser Side 1011

Code Examples 1011

Success 1015

DataError 1016

SystemError 1016

Exception 1016

Manager Error 1016

Conclusion 1016

Validate Form Data Against a Schema **1018**

User Builder API Overview **1019**

Create New User 1019
Update an Existing User 1019
Use the Groovy Console 1020
Conclusion 1020
Access Form Properties with a Groovy Service 1021
Access Service Parameters of Another Groovy Service 1022
Create Anonymous Tasks with a Groovy Service 1023
Create Form Tasks with a Groovy Delivery Service 1027
Mix Standard Delivery Functions with Custom Groovy Processing 1032
Invoking Dynamic Data Service from a Groovy Service 1036
Use Case 1036
Solution 1036
Dynamic Data Lookup Service 1037
Introduction to Dynamic Data 1037
Handling Data 1037
Building the Composer Form 1037
Input Data 1037
Output Data 1038
The Lookup Widget 1038
Testing the Lookup 1038
Building the Transaction Manager Dynamic Data Service. 1039
Copying an existing Dynamic Service Definition 1039
Modifying the Dynamic Service Definition 1039
Linking the Dynamic Data service in Transaction Manager to the Composer form. 1041

A Final Word - Testing Testing Testing 1041

Bonus Points - Repeating blocks of results 1041

Conclusion 1042

Check a Form Request IP Address 1043

Groovy Service Tips and Tricks 1045

Get the version of Groovy 1045

Get Date widget into a W3C dateTime 1045

Manager included libraries 1046

Groovy script body building 1046

Capturing Groovy Script Output 1047

Use a Builder and a Map to Conveniently Return Dynamic Data 1048

Mocking up Web Service SOAP Return Data 1048

Typed Objects 1050

JSON 1050

Collections and JSON 1051

Create an Object Based on a Collection 1051

HTTP Session Object Pattern 1053

Invoke Core Services Pattern 1054

Step 1 1054

Step 2 1055

Step 3 1056

Step 4: 1056

Step 5 1058

Step 6 1059

Benefits 1059

Long Running Service Pattern 1060

Progress Polling 1061

Blind State Machine 1064

Enterprise Service Bus 1066

Conclusion 1066

Multi-Step Groovy Delivery Process 1067

Basic Groovy Delivery Process 1067

Dual Delivery 1069

Restore Undeliverable Transactions 1072

Reload Form Submission Data Extracts from the Groovy Console 1074

Server-Side Persistence Pattern 1077

Storing data 1077

Retrieving data 1077

Some notes 1078

Tasks Overview 1079

View Tasks 1080

Create a Task 1083

Configure a Task 1086

View Task Transaction Details 1088

Abandon a Task 1089

System Event Publisher Overview 1090

A Single Queue 1091

Multiple Queues 1091

Configure System Event Publisher **1092**

System Event Publisher Events **1093**

System Event Publisher Audit Log Events **1095**

System Event Publisher Job Events **1096**

System Event Publisher Submission Events **1098**

System Event Publisher System Health Events **1103**

System Event Publisher User Authentication Events **1105**

Security Managers Overview **1107**

SSO Flow 1109

Login Flow 1109

Download the Security Managers PDF 1110

User Accounts Overview **1111**

Authenticated Users 1112

Anonymous Users 1112

View User Accounts **1115**

Create a User Account **1117**

Configure a User Account **1121**

Assign Roles to a User Account **1124**

Assign Organizations to a User Account **1127**

Assign Form Spaces to a User Account **1130**

Provision a User Account **1131**

Provision a User Account for Accessing a Maestro Shared Environment Only 1131

Provision a User Account with the Organization User Manager Role 1132

Troubleshooting 1132

General Cause 1 1133
General Cause 2 1133
General Cause 3 1134
User Profiles Overview **1135**
Create a User Profile **1136**
Configure a User Profile **1138**
User Property Types **1139**
Remove a User Account **1140**
Deactivate a User Account 1140
Remove a User Account 1141
Bulk Import of User Accounts **1143**
Migrate User Accounts **1146**
View User Login History **1147**
User Visibility Rules for Restricted Content **1149**
User Visibility Rules for Collaboration Jobs **1150**
User Visibility Rules for Transactions **1153**
Roles Overview **1156**
Root Administrator 1156
Developer / System Manager 1156
Operations 1156
Organization User Manager 1156
System Support 1156
Transaction Data Access 1157
Standard Roles **1158**

View Roles **1163**

Create a Role **1165**

Manage Role Members **1167**

Configure Role Expiry **1170**

Remove a Role **1172**

Deactivate a Role 1172

Remove a Role 1172

Export Roles **1174**

Import Roles **1175**

Migrate Roles **1177**

Groups Overview **1178**

View Groups **1179**

Alert Groups **1181**

Receive Delivery Escalation Alerts 1181

Receive Outstanding Payment Alerts 1181

Receive Promotion Alerts 1181

Receive System Alerts 1181

Receive Submission Updates 1181

Receive System Alerts - Delivery 1182

Receive System Alerts - Receipts 1182

Receive User Account Creation Alerts 1182

Custom Alerts 1182

Create a Group **1183**

Assign Users to a Group **1186**

Assign Groups to a User **1189**

Configure Group Expiry for a User Account **1191**

View Forms Assigned to a Group **1193**

Migrate Groups **1194**

Export Groups **1195**

Import Groups **1196**

Permissions Overview **1199**

Core Permissions **1200**

Maestro Permissions **1229**

Exchange Permissions **1232**

Journey Analytics Permissions **1233**

Workspaces Permissions **1234**

Assign Permissions to a Role **1236**

View User Login History **1239**

Log in to Manager **1241**

Two-factor Authentication 1241

Change a Password **1243**

Log out of Manager **1244**

Operations Overview **1246**

Transactions Overview **1247**

Submission Life Cycle **1248**

User flow - Form Transaction Experiences (FTX) 1248

User flow -Traditional (None FTX) 1249

Delivery flow 1249

User Actions 1249
Manager Actions 1251
Submission Categories **1253**
Completed Submissions **1254**
Collaboration Job Submission **1256**
Abandoned Submissions **1258**
Authenticated vs Unauthenticated Transactions **1261**
Form Pre-fill 1261
Form Save & Resume 1261
Transaction History 1261
Tasks 1262
Best Practices 1262
Configure Form Abandonment **1264**
Abandon a Transaction **1266**
Transactions and Tasks Abandonment **1267**
Automatic Abandonment 1267
Transaction Abandonment 1267
Task Abandonment 1268
Transaction and Task Purging 1269
Manual Abandonment 1269
Transaction Details **1270**
Transaction Status **1273**
Change Transaction Status **1279**
Method 1 1279

Method 2 1280

Form Sessions **1281**

Transaction History **1282**

Transaction History Details **1285**

Form XML Data **1287**

Form Data Extract **1289**

Attachments **1290**

Transaction Groovy Log **1292**

Transaction Events **1294**

Transaction Properties **1295**

Transaction Timeline **1299**

Transaction Support Search **1300**

Track Transactions **1307**

View Form Transactions **1308**

Form Data Extract **1310**

View Abandoned Transactions **1311**

View Saved Transactions **1313**

View Payment Transactions **1315**

View Form Submission Data **1316**

View Form Requests **1317**

View Form Rendering Metrics **1320**

View Transactions History **1322**

Debugging Maestro Forms with Chrome DevTools **1324**

Prerequisites 1324

Building a Form 1324
Opening Chrome DevTools 1325
Inspecting the Form Object 1326
data 1327
items 1328
view 1328
Debugging JavaScript Rules 1329
Using Breakpoints 1330
Rule Types and JavaScript Function Naming 1333
Rule Type Codes 1334
Action Rules 1335
Dynamic Data Rules 1335
Change Data Format in Form XML Data 1336
Linux 1337
Windows 1337
System Logs 1338
Dos: 1338
Don'ts: 1338
Error Log 1339
Error Log Trend 1342
Event Log 1343
Database Version Log 1346
Import Log 1348
Audit Log 1350

Groovy Service Log **1352**

Groovy Log to Splunk Integration **1356**

Apache Log **1358**

T.Field Device Log **1359**

T.Field Sync Log **1360**

Journey Manager Analytics **1361**

Analyze Transaction Trend **1362**

Transaction Volume 1363

Completion Rate (%) 1364

Conversion Rate (%) 1364

Form Abandonment Rate (%) 1364

Bounce Rate (%) 1364

Median Form Completion Time (mins) 1365

Form A/B Testing Overview **1366**

Configure Form A/B Testing **1367**

Elect Form A/B Testing **1369**

Analyze Form A/B Testing **1372**

Analyze Forms Activity **1374**

Analyze Device Type Submissions **1376**

Analyze Form Submissions **1377**

Analyze Submission Trend **1379**

Analyze Referer Submissions **1380**

Analyze Browser Submissions **1382**

Analyze OS Submissions **1383**

Analyze Reader Versions **1384**

Maestro and Composer Forms A/B Testing **1385**

Maestro 1385

Composer 1386

Manager 1387

Performing the Tests, Reviewing Results 1388

Reports Overview **1390**

Report Configurations **1391**

System Report Schedules **1394**

View Transaction Licensing Report **1397**

View Job Licensing Report **1400**

Create a Custom Report **1402**

Form 1402

Groovy Script 1406

Outgoing Email Service Overview **1414**

Configure a SMTP Server **1416**

Create an Email Queue Item **1418**

Edit an Email Queue Item **1419**

View the Email Queue **1421**

Data Retention Management Overview **1423**

Data Retention Configuration - Global Level 1424

Data Retention Configuration - Organization Level 1424

Configure Data Retention **1425**

Configure Organization Data Retention **1429**

Configure Form Data Retention **1431**

Data Retention Policies **1433**

Saved Transactions 1434

Finished Transactions with PII Data 1434

Finished Transactions 1434

Finished Collaboration Jobs 1435

Submission Data Extracts 1435

Submission Properties 1435

Data Retention Schedule **1437**

Manual Data Purge **1440**

Data Retention Summary **1441**

Data Retention Summary: Transactions **1442**

Data Retention Summary: Job Transactions **1444**

Data Retention Summary : Collaboration Jobs **1446**

Data Retention Summary: System Logs **1448**

Data Retention Summary: Top Table Sizes **1449**

Server Nodes Overview **1451**

View Server Nodes **1452**

Create a Server Node **1454**

Configure a Server Node **1456**

Monitor Server Node Performance **1459**

Remove a Server Node **1461**

System Info **1462**

Runtime Info 1462

Build Properties 1462
System Properties 1463
System Health **1464**
System User Preferences **1466**
System Licensing **1467**
Deployment Properties **1469**
Default Deployment Properties 1470
Important Deployment Properties 1475
Directories 1475
SMTP Server 1475
Duplicate Form Submissions 1476
Journey Analytics Deployment Properties 1476
Enable Deprecated Features **1477**
About **1479**
Multi-Environment Support **1480**
Multi-Environment Migration **1481**
Manager Deployment and Rollback Procedure **1482**
Deployment 1482
Security Managers 1482
Organisations 1482
Form Spaces 1482
Organisation Applications 1483
Service Connections 1483
Verification 1483

Rollback 1483
Security Managers 1483
Organisations 1483
Form Spaces 1483
Organisation Applications 1483
Service Connections 1484
Verification 1484
Appendix - SSL Keystore Generation 1484
Appendix - Organizations Environment 1484
Appendix - Form Spaces Environment 1484
Appendix - Organization Applications Environment 1484
Appendix - Service Connections Environment 1484
Appendix - Verification 1484
Rollback Overview 1485
Rollback Configuration 1486
Manager FAQ 1487
General 1487
How to verify 2FA PIN from Amazon SNS was sent using SMS? 1487
How to redirect a form user to an external URL instead of a Journey Manager form? 1487
How to add a Java library to Journey Manager? 1487
How to turn off the auto-confirmation email? 1488

Journey Manager Overview

Journey Manager (JM) | System Manager / DevOps | 21.05

Journey Manager is the heart of the Journey platform. It is a system of engagement, purpose-built to manage customer **onboarding**[1] and customer journeys, such as account opening and servicing.

Manager productizes the technology needed for a best-in-class digital sales experience in a pre-built platform. This experience enables financial and government institutions, large or small, to create their user journeys without having to engineer complex features such as security, data capture, resiliency, convenience, and integration.

Manager is a complex software system, consisting of many interconnected components. It's designed to:

- Host application forms and associated resources, and provide customers with secure access to them.
- Manage forms' transaction flow.
- Create highly configurable services to implement the desired **customer experience**[2].
- Enable integration with various **FinTech**[3] services and systems.
- Analyze actual customer behavior to continuously improve the customer experience.

Manager provides the following core capabilities:

---

[1]The steps required to get a new customer integrated into a new program. These steps may vary business to business.

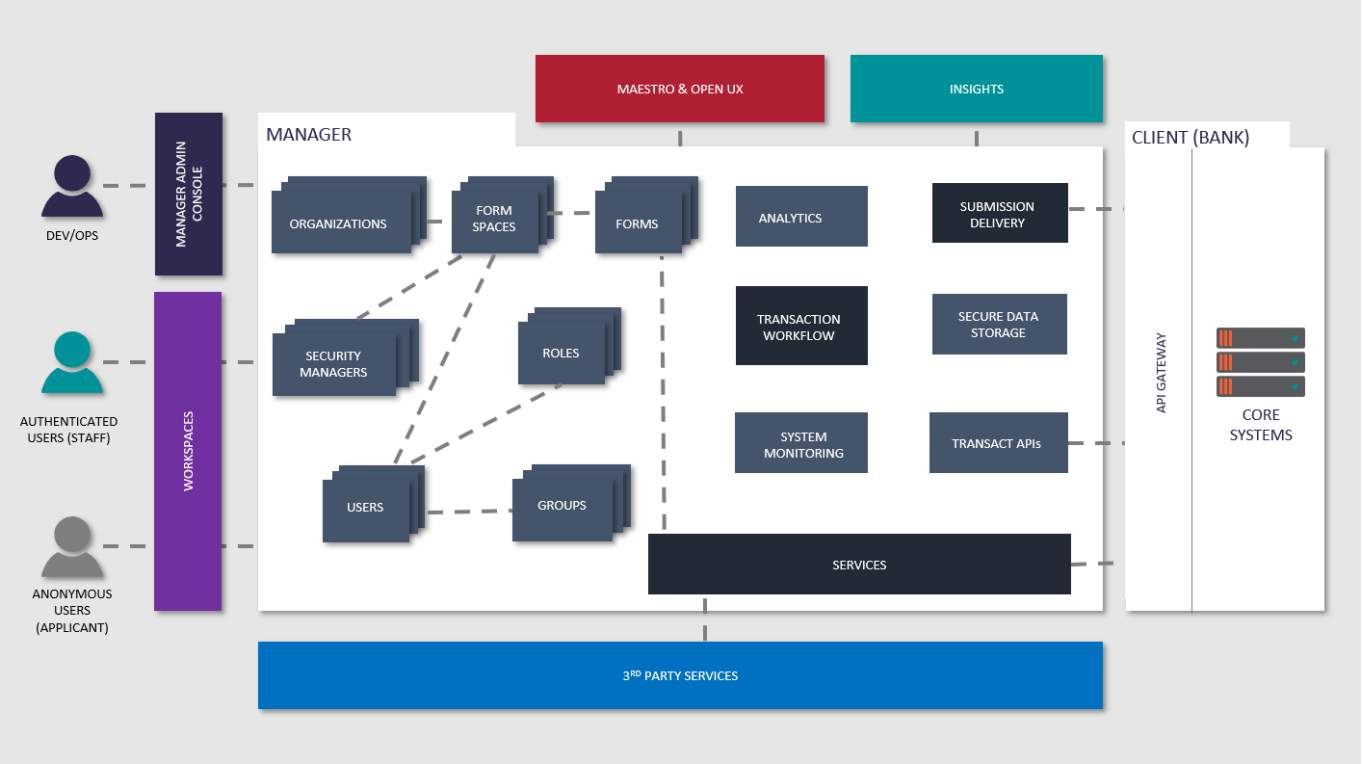
[2]Customer Experience (CX) is the product of an interaction between an organization and a customer over the duration of their relationship. This interaction is made up of three parts: the customer journey, the brand touchpoints the customer interacts with, and the environments the customer experiences (including digital environment) during their experience. A good customer experience means that the individual's experience during all points of contact matches the individual's expectations.

[3]Financial Technology, also known as FinTech, is a new industry that uses new technology and innovation to improve activities in finance. It aims to compete with traditional financial methods in the delivery of financial services to customers by improving customer experience, reducing time-to-market, decreasing cost to mention a few.

- Form hosting - multiple forms can be hosted centrally and be made accessible to applicants by embedding links to the forms into organization's web pages or by using form search engines. You can also manage forms and process submissions made by users.
- Data pre-population - users can quickly complete forms that are pre-populated with their information, which comes from one or more user's profiles or organization provided data retrieved via web services or APIs.
- Receipts - users are provided with an electronic receipt that includes the submitted form, payment and submission information. Receipts can be viewed online, emailed, or downloaded and then printed.
- Payments - if a submission needs to be accompanied by a payment, Manager can be configured to read the correct payment amount from the form and process the payment using a secure credit card payment process.
- Supporting documentation - submissions can be supported by additional documents that may be uploaded electronically or submitted in paper form. Business logic in the forms can be applied to supporting documents.
- Submission delivery - completed submissions can be delivered by email to one or more user's addresses. The delivery process can be configured separately for each form.
- Form configuration and version control - forms provide fine-grained control over behavior, including form properties, form metadata, and custom rendering orchestrations. Forms also support version control enabling parallel development and multiple versions of the same form, so you can promote forms from development, to test, and finally production on separate servers.
- Task assignment - creates a task for a user or a group of users to work with forms to complete, review or reject then.
- Submission tracking - all submissions can be tracked from form download to submission delivery. Customer support can view forms and attachments to provide user assistance during the submission process.
- Reports - provides multiple charts and standard reports for reporting on submissions. Additional reports can be configured using advanced reporting modules. Reports can be displayed on a screen or can be automatically sent out via email.

For the complete list of all product capabilities, see Manager features.

The diagram below outlines the main components, with the submission delivery, transaction workflow and services being of the high significance.



Manager provides a well-defined structure to enable authenticated (staff) and anonymous (applicants) users' interaction with forms. An organization sits at the top of that structure, followed by form spaces and finally forms. A client can have multiple organizations, which each organization using several form spaces. Each form space can provide access to different forms and a form itself can be associated with several form spaces.

Manager is a secure system that allows you to enable fine grained user access to forms by using preconfigured security managers or and customized groups and roles. The platform developers or system admins can have dedicated access to configure, maintain and manage your Manager environment. Staff and applicants can access forms directly or indirectly through Journey Workspaces.

Manager provides rich and fine grained framework for developing, testing and deploying services that implement client's business logic, interact with FinTech services and external systems. Manager comes with many preconfigured core services, covering most of common client's requirements, but you can easily create new services using Transact Fluent API and Transact functions as they are based on the standard Groovy framework.

Manager includes pre-built connectors for best in class services such as fraud detection, ID verification, digital signatures, and over 30 different connection partners. The integration has already been done, so each connector can be added to the user experience as needed when designing the application.

Every user interaction with a form is a transaction that is created and managed in Manager using the submission delivery mechanism. You have a wide of range of tools to view and manage these transactions. Manager comes with analysis tools including Insights to drill down to transactions, analyze user tends so you can make an educated decision how to improve a customer journey.

Manager stores all users' transactions in a secure data storage. You can configure how long to keep and when to purge the data by customizing data retention policy. This configuration is important to maintain optimal performance of your server. At any given time, system administrators can check Manager system info and health to see if any additional tuning is required such as adding a new server node.

Manager can be deployed in the **cloud environment**[1], or on-premise depending on customer requirements and available infrastructure.

Manager is implemented using a broad range of open source 3rd party libraries.

Next, learn how to create an organization.

[1]Cloud computing is shared pools of configurable computer system resources and higher-level services that can be rapidly provisioned with minimal management effort, often over the Internet. Cloud computing relies on sharing of resources to achieve coherence and economies of scale, similar to a public utility.

Journey Manager Features

Journey Manager (JM) | System Manager / DevOps | 21.11

Journey Manager comes packed with the wide range of features empowering you to create complex customer onboarding and user journey solutions. Some of the main product features have now become its core functionality. Click on each feature to learn more.

Main Features

- Forms for both anonymous and authenticated users
- Form prefill with user data
- Form save and resume
- Form delivery and provisioning
- Receipts and attachment management
- Services and service connections to various 3rd party services
- Form data access using the Unified Application Data Document Model and transaction properties
- Digital signature integration
- Collaboration workflows and product bundling orchestration
- Form and process analytics to improve customer journey
- Form testing, including A/B form testing
- Security management, including **Single sign-on**[1] and a user identity verification
- Management of in-flight transactions and transient data
- Error handling, monitoring, and logging for auditability and compliance regulations
- Integration with systems of records
- Configurable support of full redundancy, quality of service and disaster recovery
- Encryption of in-flight data for enhanced security
- Management of complex **PII**[2] data and encryption keys
- Email notifications for various form's events

Features by Release

Each release of Manager is shipped with new features and bug fixes to provide you with the best-in-class software and to accelerate business agility and improvements across the board.

---

[1]Single sign-on (SSO) is a property of access control of multiple related, yet independent, software systems. With this property, a user logs in with a single ID and password to gain access to any of several related systems.
[2]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

The key features for each major Manager release are listed below. Click on each release number to view the complete release notes and learn about all features and bug fixes.

22.04

Release Key Features	Description
?	?
?	?
?	?

For more information see Manager 22.04 Release Notes.

21.11

Release Key Features	Description
New Statuses added for organizations, form spaces, form versions and forms	Introduced a new set of statuses for organizations, form spaces, forms and form versions, so you have fine control over each of the entities.
New ServiceUser role	Scheduled Groovy services now run using the ServiceUser role, which defaults to a admin level role. This role can be limited to enhance the security around these services, if required.
New CSRF Protection option	Added the CSRF Protection option to the Workspaces, Workspace and Web-plug spaces.

For more information see Manager 21.11 Release Notes.

21.05

Release Key Features	Description
A new REST service - Proof of Concept	A new service which exposes some of Manager's Fluent API via REST[1] end points.
Fluent API	We've further expanded the Fluent API to expose more features to help reduce the dependency on the deprecated Core Groovy API:

[1] REST or RESTful API design (Representational State Transfer) is designed to take advantage of existing protocols. While REST can be used over nearly any protocol, it usually takes advantage of HTTP when used for Web APIs.

	• Form VO - Included the form type through the **getFormType()** method. • **TxnUpdater**, **JobUpdater** and **TxnQuery** - Included feature to view and manager comments against Transactions and Jobs • **TxnUrlBuilder** - Included the capability to generate secure URL for forms/transactions through the **setSecured()** accessor method. • **TxnUpdator** - Included the capability to lock and unlock the save Challenge lock state for a transaction
Manager installer	We've enhanced the installer so that for AWS installs using MySQL SSL is enabled by default to improve server security. For more information, see the SSL JDBC Connection Enforcement for MySQL section of the installation guide. Due to components requiring commercial licences, iText, Oracle and Luna dependencies are not included with installs by default. These will need to be deployed separately. For more information, see the Manual Libraries Deployment section of the installation guide.
Reports	Remove the PDF export option from the Transaction and Job Licensing reports because it uses iText, which requires a commercial license.

For more information see Manager 21.05 Release Notes.

20.11

Release Key Features	Description
Core services	Added support for: • A new magic byte scanning service • A new Attachment File Name Regex deployment property

For more information see Manager 20.11 Release Notes.

20.05

Release Key Features	Description
Azure Cloud support	Added support for: • Encryption key management using Azure Key Vault • Attachment and Receipt storage using Azure Blob Storage • System Event Publisher using Azure Queue

Automatic deployment and testing	Added support for: • The ExampleImportService, which was used to import example artifacts into Manager on initial startup, now supports all archive types. This is useful to import test artifacts for automation testing. • The bulk user import spreadsheet is expanded to support Groups, Roles and Organization. This is also incorporated into the ExampleImportService.

For more information see Manager 20.05 Release Notes.

19.11

Release Key Features	Description
General Fluent API enhancements	Added support for: • TxnUpdater, TxnQuery, TxnBuilder and Txn VO to improve the transaction level capability of the Fluent API • Emailer and UserEmailer • Added a new Job Builder class
Fluent API receipting enhancements	Added support for: • Calling a named receipt render service from within a Fluent Receipt Service to enable receipt merging while still generating a receipt for the associated transaction. • Generating a Web and Email received from a Fluent Receipt Service. • Externalization of the Core PDFUtils methods to merge receipt in Fluent.
Fluent Security Managers support added	Added support for: • A Fluent User Authentication Provider, with a template for a skeleton Fluent Groovy User Auth Provider. • A Fluent Security Manager, which includes a template for a skeleton SSO Security Manager and an out-of-the-box ADFS Security Manager. • Fluent Security Managers implementation has resulted in a number of new Fluent API classes to support the scripting within a Security Manager.
Form version import enhancements	Form version import options include Form Function now.
Removed features	Removed:

	• Adobe Livecycle support removed • Card Payment page removed from the Workspace

For more information see Manager 19.11 Release Notes.

19.05

Release Key Features	Description
Workspace API	Enhanced to support the development of the new help desk persona for the Workspaces, including: • Transaction search using multiple criteria • Additional updates to Txn and Job Queries • Security enhancements
Fluent API	Enhanced to reduce the dependency on the Core API and improve upgrade supportability, including: • New PaymentLog Query class • Improved TxnQuery transaction search capabilities • New feature to support collaboration jobs • Added form expiry information to the Form object
Security enhancements	Improved CORS support for form spaces
Chromium puppeteer receipt service	Added a new Chromium puppeteer receipt service to render forms as PDF receipts
Library upgrades	Upgraded the dependent libraries, including: • Apache Cayenne 4.0 • Spring Security 4 • Wildfly 16
Removed features	Removed: • Birt and PayPal Payment gateway. It is replaced with the Transaction and Job Licensing screens for transaction usage information. • PayPal Payment gateway.

For more information see Manager 19.05 Release Notes.

18.11

Release Key Features	Description

Workspace API	Introduced new REST API to support the development of rich UI experiences for 'Back of House' staff who support complex workflows and provide help desk functions to end users. The API utilizes Swagger and Spring Security.
Static content externalization	Added a new Maestro form rendering mode which significantly reduces the time it takes to load large forms. This is important as long application loading times can significantly increase the bounce rate with users simply closing the browser window before starting the application.
System Event Publisher	Added a new user authentication event feed to send user login events. Enhanced scheduled purge time and actual purge time for a transaction.
Security hardening	Added support for: • Added Validation of date field to the workspace portal to fix internal error when provided date is invalid. • Security around service connection password in the manager UI. Removed confirmation page and receipting from the Server Monitor Form configuration.
Library upgrades	Upgraded the dependent libraries, including: • Open JDK 11 • User Agent Utils 1.21 • Spring Framework 4.3.19 • Spring Security 43.1.7 • Bouncycastle 1.60 • SDK 3.9 • org.apache.xmlgraphics batik 1.10 Removed: • JLine
Platform support enhancements	Added support for: • Microsoft SQL Server 2017 database. • Microsoft Windows Server 2016 operating system.
Fluent API	Added new API.

Behavior changes	Enhanced management of submissions' lifetime.
Removed features	Removed: • FopRenderReceiptService receipt render service & org.apache.xmlgraphics library • Use of Flash in the Manager UI • Apache Camel Support for Groovy Services

For more information see Manager 18.11 Release Notes.

18.05

Release Key Features	Description
System event publisher	Introduced the System event publisher to connect external systems through an AWS SQS queue, in near real time, where the messages can be picked to provide up to date information about the states of form transactions in Journey Manager.
Library upgrades	Upgraded the dependent libraries, including: • Apache Groovy Lang 2.4.15 • Eclipse BIRT Runtime 4.6.0 • JDBC Database Drivers (MySql, Oracle and SQL Server) • JUnit 4.12 • Oracle Java JDK 1.8u172
Platform support enhancements	Added support for new Insights, including: • Centrally stored ECCL Script • Fluent API to send Insights Segments from server side Groovy Scripts
Fluent API	Added new API
Behavior changes	Enhanced the transaction abandonment and data purging to ensure PII data is removed accordigly.

For more information see Manager 18.05 Release Notes.

17.10

Release Key Feature	Description

Transact Functions	Introduced Fluent Functions to allow you to use functional programming within Journey Manager, so you can now write a function once and simply trigger it at whatever points or points you need.
Continuous Integration	Provided improved CI-based testing support. Created a definition of what is included in the application package and versioned in GIT or another SCM tool.
Transact SDK	Combined developer tools into a unified Transact SDK to provide a single set of tools to help developers make the most of integrating with Transact API's while using Continuous Integration.
Reference data	Introduced Transact Reference Data to decouple reference data from the form, decrease the form loading time, and enable you to deploy once and use in many forms.
Semantic versioning	Unified versioning across the platform for consistency including full semantic versioning across all forms and services. Semantic Versioning enables two teams to work in parallel on more than one form solution and permits hierarchal management of versions. Additional features include support for up to 3 levels of versions and version branching.
Security hardening	Add support for Strict Security Headers to enforce that each HTTP request within authenticated user session is checked to make sure it is the originated source that created the session. The HTTP request's IP Address and Browser User Agent are checked.
Collaboration Jobs	Enhanced collaboration jobs to support group sharing.

For more information see Manager 17.10 Release Notes.

5.1

Release Key Features	Description
Data retention policy enhancements	Retention duration time is reduced to make sure Journey Manager is used as a system of engagement and not a system of records: • Transactions - reduced from 180 days to 90 days. • Transaction history - reduced from 2 years to 180 days.
Transaction history data warehouse integration	Added the following publishing services to publish transaction history directly to a data warehouse:

	• AWS S3 • File system • Fluent Groovy
Automated deployments	Introduced a new script-able install or upgrade mode for deployment automation, which supports Continuous Integration (CI) testing. This mode is beneficial to both hosted services and IT professionals in that it helps to automate provisioning and upgrading, which in turn can improve quality by reducing operator errors.
Fine grained form space authentication policies	Enhanced forms authentication policies, which depend on form space they are in.
Fluent Groovy API	Added new Fluent service types. • Form version Selector • Receipt Render • Transaction History Publisher • Virus Scan Service
Application packages	Added new configurations for Organization Properties, Delivery Channels, and Service Connections.
Security manger logging	Introduced a new Security Manager logger that can be used in all Security Manager Groovy scripts.
Update to latest Maestro form from within Manager	Enhanced Maestro form updates from Journey Manager, so it uses the latest build of the Maestro form without exporting a Maestro Form Version Archive and then importing it Manager.
Maestro localization support	Introduced localization to Journey Manager and Maestro, which allows multiple languages to be added to a single form. This feature allows a form to change languages based on the users of the form.
Collaboration Jobs	Form and service version pinning: • The form code for Review Tasks is no longer necessary. Task Assign actions that do not specify a Form Code will by default use the same form code and version as the previous submission. • The Form Version can be explicitly set in the Job Definition. • The Job Action Version can be explicitly set in the Job Definition.
Collaboration Jobs	History logging, so the state of the job is recorded and logged at the end of each branch of processing completes.

For more information see Manager 5.1 Release Notes.

5.0

Release Key Features	Description
Fluent Groovy API	Introduced completely new and improved API, which can be used from Groovy scripts.
Maestro form version import	Added support for Maestro form version import defaults, which provides specialized import behavior if a form version archive was created in Maestro.

For more information see Manager 5.0 Release Notes.

4.3

Release Key Features	Description
Application packages	Added support for: • Form bundle and form chaining • Receipts with Groovy scripts • Dynamic task assignment • Cancellable tasks • Publish and subscribe data extract sharing • Unified navigation support
AWS KMS	Added support for AWS Key Management System (KMS) data encryption in the Cloud Service Enterprise VDC edition.
Groovy service tests	Added comprehensive Groovy Services test framework to support accelerated development and development best practices.
REST API	Added Transactions, Service Definitions and Groovy Service Invoke REST APIs.
Security hardening	Added security role and group grant leases, improved SEM logging support and Apache Mod Security configurations.

Service improvements	Added Form Security Filter, Groovy Receipt service types, service versioning support, Groovy Service Logging, improved documentation, service editing and form and service definition Organisation Application Packages.
Transact Integration Gateway	Introduced TIG, which provides a light weight on-premise integration module for performing deliveries and proxy calls to internal systems. This module replaces the TIA and TDA modules.

For more information see Manager 4.3 Release Notes.

Detailed information on Manager releases is available from the the Release Notes Sitemap in our Journey Platform resources website.

Next, learn more about deprecated features.

Unified Application Data Document Model

Journey Manager (JM) | System Manager / DevOps | 17.10

Journey Manager comes with the Unified Application Data Document Model that allows you to store integration data separately from a form data. This is the recommended approach as compared to the transaction properties.

The Journey platform is essentially based on a document oriented architecture where we provide a seamless customer experience to capture data for further processing. During the application life cycle we persist application data in XML format that can be used to deliver data to the back end for customer onboarding. During the application process we also integrate with various third party to perform operations such as lookup, prefill, validation, authentication, authorization, funding, account onboarding and list goes on. These operations are executed on server side and responses may return back to application front end to perform next steps. End user may not need to see every single detail of the operation performed by the server, however it is important to persist the state and responses on the server for making decisions based on the outcome, or deliver to back-end for detailed outcome of those operations. So persisting that information becomes as critical as persisting user entered information.

In earlier days, developers persist server side information as Manager properties and use it for making a decision. These properties are saved as database entities against the transaction and available until transaction is purged. This is basically a key-value pair that can be retrieved by querying transaction property name.

Since Manager 17.10, you can enable the Unified Application Data Document Model mechanism in a form version by selecting the **Unified App Data** checkbox.

By default all transactions use **`AvokaSmartForm`** as the root node for transaction XML. Upon enabling this feature, Manager adds an additional wrapper called **`Root`** to **`AvokaSmartForm`**. During the form render request, Manager only sends or merges nodes under "AvokaSmartForm" and the rest of the information stays on the server. This allows the developer to persist all information that is not relevant or sensitive to the client on the server itself.

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
  <AvokaSmartForm>
    <Applicant>
      <FirstName>John</FirstName>
      <LastName>Doe</LastName>
    </Applicant>
  </AvokaSmartForm>
</Root>
```

When to use Unified App Data

Both Transaction Properties and Unified App data model allow the developer to persist server side information. Since there is no right or wrong approach but Unified App model was introduced to make the developer's life easier and serves many benefits over the conventional approach to save information as transaction properties such as:

- Data Integrity - It is important that application data is not manipulated by accessing form data JSON on client side. If we completely rely on form data, then there is a possibility that if someone intentionally try's to override the application outcome. Persisting data on server side XML allows developer to validate the information or use that information as a base for further processing rather than completely relying on data passed by front end.
- Data Marshaling - Manager provides APIs to allow developers to generate and populate VOs from XML structure. This allows developers to work with POJOs rather than arbitrary values. This helps to write better unit tests against various application states.
- Data validation - Data can be validated against XML Schema validation rules.
- Data Sensitivity - Critical information that doesn't need to be exposed will still be available on server by accessing transaction XML. Developer doesn't need to adopt any additional approach to persist anything beyond transaction XML.
- Data Delivery - Delivery API can deliver transaction XML which will automatically deliver the server side information to a customer where transaction properties require additional effort.
- Encryption - A transaction XML data is always encrypted in the database where transaction properties are not. This will protect against any PII data being exposed.

When you prefill the application or hydrate it with details like created account numbers, you should consider carefully what data to include in each tag because it goes to the browser. In most cases, you should only send masked values, not the raw ones. You should keep the raw values on the server-side only.

Springboard by default has some design patterns around this with the Lock Customer function, which you can use as a best practice.

When to use Transaction Properties

Transaction Properties and Unified App Data are not mutually exclusive. The developer can still use transaction properties for scenarios such as:

- Reporting - You may require a custom report on transaction data which becomes pain by parsing form xml for every transaction. A data extract or transaction property (as long as it doesn't contain PII) can be used to save the information that you can report on.
- Workspace - As at the time of writing this article, workspace API are designed to read transaction properties to display integration outcome / high level information on the screen. This is currently not designed to read form xml. There are plans to include form xml support in future.
- Immense data - There are some use cases where developer needs to persist huge amount of information (may be a raw response). This is still debatable but I personally don't feel to record immense amount of data as server xml, specially responses returned by 3rd party vendors as we don't know what the data format would be. Saving unknown information that may include unsupported xml values can lead into issues. This kind of information can persist as Transaction Property (blob) in the database.

Next, learn about the data model.

Save Challenge

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager comes with a Save Challenge feature that allows users to resume their form applications, so they can complete or cancel them. This is an important step in the onboarding process when a user needs to save a form application to get some required information or a web browser is closed or crashed. Therefore, the Save Challenge provides a secure mechanism to return to the form.

To enable a Save Challenge:

1. Select the **Supports Save Challenge** checkbox in a form version.
2. Specify the **Save Challenge XPath** field in Form Configuration Mapping, for example, the Save Challenge XPath is set as **`//DateOfBirth`**.
3. Check the **`supportsSaveChallenge`** is set to **`true`** in the SDK **`form-def.json`** file.

   > The **`application-package-config.xml`** is updated with this option after running the **`app-package`** command.

4. Check the Save Challenge is enabled in Transaction Status.

Next, learn abut other Manager's features.

Journey Platform Integration

Journey Platform | All Personas

The Temenos Journey Platform enables digital interaction between an organization and its customers. Because of the importance of this interaction, there is often a need to integrate between the Journey platform and other systems. In some cases, the integration required can be minimal, but integration can be quite extensive.

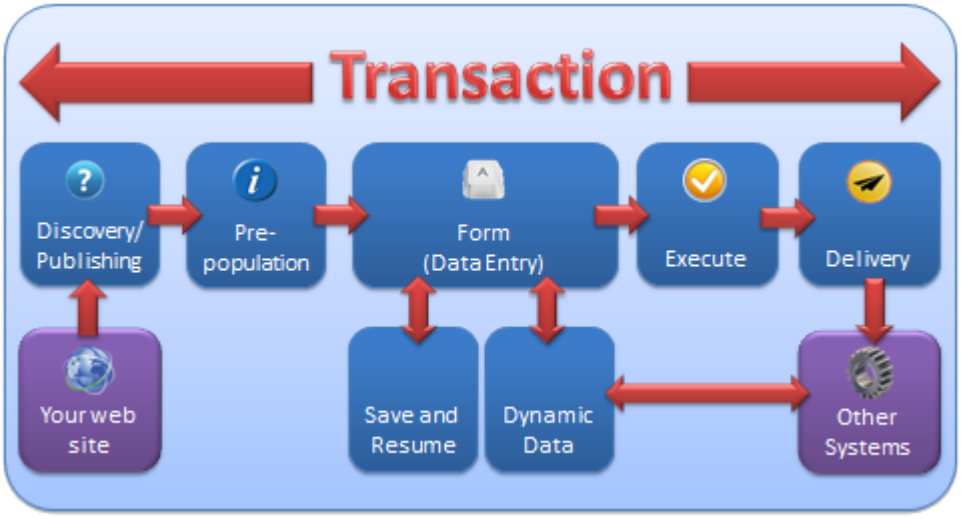
- Integration areas occur at different stages within the interaction between a user and the organization.
- Within each integration area, there are different integration implementations, to support different technologies used by different organizations.

Integration Areas

There are several key areas or stages in which integrations generally occur during transactions:

- Transactions Discovery
- Data Pre-population
- Data Entry Streamlining
- Execution
- Save and Resume
- Delivery

This is illustrated in the diagram below.



These integration stages are detailed described below.

Transactions Discovery

Your users need to be able to discover the transactions that you provide. There are a number of ways to do this. The simplest is to have a link on your web site which launches the transaction.

Things can get more sophisticated than this. For example:

- You may want to have customized links, which are different for different users. Generally this is so that you can provide custom pieces of data in the link URL, generally so that you can pre-populate some of the fields in the form from this data. This helps to make it easier for users to fill out forms (see Pre-population below).
- If your users are logged into your web site, you may want to pass their identity to the transaction. This again allows the form to pre-populate some of the fields.
- You may want to embed the form into your existing web site, rather than linking to it. This is generally not recommended in modern responsive and task-oriented designs, although it is possible to implement with Transact.
- Another option is to send the user a link to the transaction using some other mechanism, such as via an email. You can send them a link to a blank form, alternately you can also use Transaction Manager to create a custom link that contains a partially pre-filled form to make their lives easier.
- Users can also save a transaction that they have partially completed. This will generate a custom link for them, that will enable them to return to the transaction at a later date. There are also other ways for them to get back to the transaction if they lose the link. You can configure a challenge question to ensure that only the original author can access their saved transaction.

Data Pre-population

We want to make it as easy for your users to complete a transaction as possible. Transact is specifically designed to make the entire experience as intuitive and simple and engaging as possible.

You can make the data entry process even simpler by pre-populating (or pre-filling) information into the form that you already know about them, which means that they won't have to type the information themselves. Depending on what information is available, this can dramatically

We can generally provide the pre-fill data in one of the following ways:

- Front-end data passing. If your existing web site knows who the user is, you can pass specific pieces of data about that user directly to the transaction. This can be done via URL parameters, cookies, lookup tokens, and a variety of other techniques.
- Back-end look-ups. If the user has been authenticated by your web site, you can pass their identity to the Transaction Manager server. We can then use this identity to look up information in back-end business systems, or Transaction Manager's own profile database, to pre-fill the form.

Pre-population is also sometimes provided by using a previous submission's data to populate a new submission. This is useful when a transaction has to be repeated every year (or periodically). and the data doesn't change much from submission to submission.

Data Entry Streamlining

During the data entry process, we want to make it as easy as possible for users to complete the form, and get the product or service they are requesting.

There are a number of ways we can do this. For example:

- We can populate fields in the form from third party services, such as LinkedIn or Facebook. (This is similar to pre-population, but occurs once the form has already loaded, rather than prior to loading.)
- We can do calculations and lookups to provide them more information, or to validate their data entry. For example, we might verify the name of the account holder that they are trying to transfer funds to, or calculate the approximate repayments for their loan.
- We can provide them ways of speeding up their data entry. For example, instead of forcing them to type in an address laboriously, field by field, we can allow them to just start typing an address, and provide a matching set of possible candidate addresses, and then auto-complete the individual fields. (Like the address picker in Google Maps.)
- We can ask the user to upload a document, such as a photo of their license. We could use OCR technology to recognize the text on the document, and used to populate fields on the form.
- We can give the user real-time validation of their eligibility. For example, we could integrate with a credit scoring service to provide an indication about whether their loan is likely to be approved or not.

Data entry optimization is usually done using one of two techniques:

- Client-side integration. This occurs directly between the form (executing in the client's browser), and the information system. The LinkedIn pre-population widget is this type - no server-side installation or configuration is required.
- Dynamic Data. In this case, the form will invoke Transaction Manager, which will in turn invoke the actual service. Several Composer widgets are designed to work with Dynamic Data Services. Transaction Manager can provide added value, such as data and protocol transformation and security.

Execution

Execution is the process of adding additional value to the transaction after the data entry step. It can consist of a number of added-value steps, some of which require may integration with back-office or third party systems.

Some examples of added value integrations include:

- Adding electronic signatures using third-party eSignature solutions.
- Processing payments using third-party payment gateways.
- Retrieving credit scores or identity verification results using third-party services.
- Using Enterprise Document Management systems to generate documents or welcome kits to provide to the customer.
- Providing a review-and-approval step prior to delivery.

Save and Resume

Saving a form and returning to it to resume editing is an important capability. This is shown on the diagram for completeness, but is in fact implemented purely inside of Transaction Manager. There are no user-modifiable extension points.

Delivery

Delivery is the process of delivering the results of the transaction, in both human-readable and machine-readable format, to the organization.

Of all the different points of integration, delivery is the most important. The reason is obvious - unless we deliver the data that was collected (in some way) to the organization, there is no value achieved for the organization.

The range of different delivery options is highly varied. It can be as simple as an email containing a PDF as an attachment which is processed manually, or as complicated as direct integration of different parts of the data into a number of different back-end business systems. Delivery can even be performed for transactions that haven't been completed by the end user, as this abandonment information can itself be useful.

Authenticated and Unauthenticated Transactions

Manager supports the use of both authenticated or unauthenticated transactions. Authentication is usually the process of identifying a user using a user-name and password, or sometimes more sophisticated methods. An unauthenticated user is a user who hasn't logged into Transaction Manager, and doesn't have an identity in Transaction Manager. This does not necessarily mean that they are unknown, and usually some information will be known about them, often including their email address.

Similar capabilities are available for authenticated and unauthenticated users, but are implemented in slightly different ways:

Capability	Authenticated	Unauthenticated (Anonymous)
Pre-population	A user's identity can be used to pre-fill forms with information that is already known about that particular person. This assumes that the user's identity already exists in an existing system.	Pre-population can still be achieved, but is implemented without the user actually logging in. For example, pre-fill data can be passed to the transaction using URL parameters.
Save and Resume	An authenticated user can save a partially completed transaction. By logging in again later, the user can view their partially completed transactions (known as drafts), and resume editing. The user can also log in from a different device.	Save and resume is implemented via a simple reference code and challenge question (similar to an airline booking code).
Transaction History	When an authenticated user logs into Transaction Manager, they can see the history of their submissions. (Note that they will usually only be able to see a summary of the transaction, not the content, because generally data retention policies will result in the details being purged.)	Transaction history can be provided to in a number of different ways, such as a confirmation email. Historical information is often maintained in other systems that maintain a more complete history of the customer's interactions with the organization, such as CRM systems.
Tasks	A task can be assigned to a user who is authenticated in Transaction Manager, and	A task can be assigned to an unau-

appears in their task list.	thenticated user via email or any other system that can display a URL. The user will simply click on the link in order to perform the task.

A question that is often asked is whether to use authenticated or unauthenticated access. Transaction Manager supports both types extremely well. However, it is important to note the following regarding authenticated transactions.

1. Many organizations already have the concept of identity and usernames/passwords. Usually the organization will not want to introduce a new and different username and password, but rather re-use the existing mechanism.
2. Users are often reluctant to sign up with a username and password just in order to be able to save and resume a transaction later. (We have data that supports this assumption.) This can increase transaction abandonment, because users simply don't want to register. It's usually wise to allow users to complete a transaction without requiring them to register.
3. Identity can also be complicated to implement, especially the identity of your customers outside the firewall. Transaction Manager does support both common signon using LDAP, and Single Sign On (using SAML), but these can add complexity to the implementation.

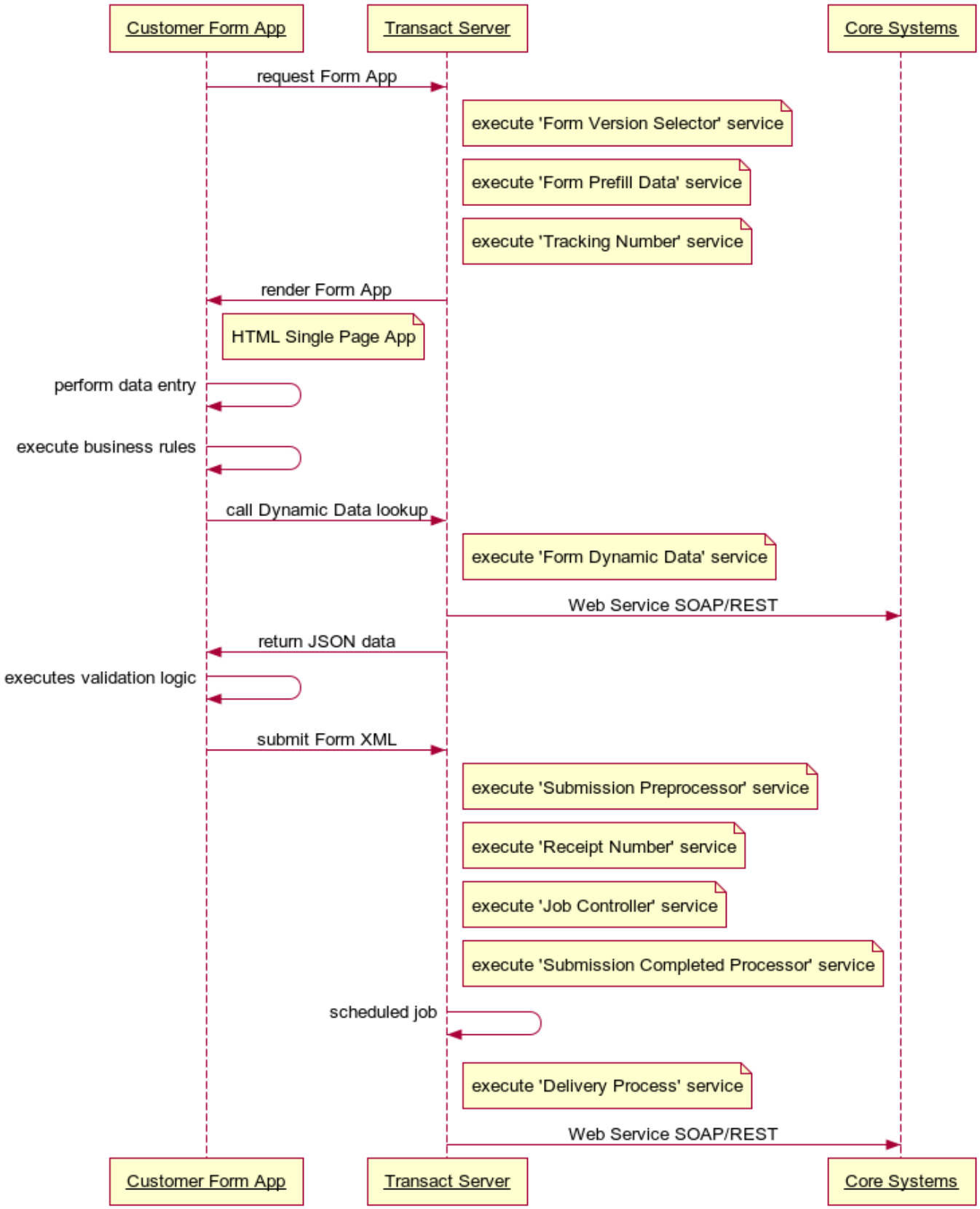
Because of the above, we generally recommend the following:

- External customers be treated as "anonymous" users, without trying to establish or confirm their identity online. This may have a minor impact on their ability to pre-fill their forms (although there are ways to achieve this), but in general, treating them as anonymous dramatically simplifies the integration requirements, the security implications, and the time to market. As outlined above, most features in Transact such as save and resume, task assignment and pre-fill have been designed to work with either anonymous or authenticated users.
- Internal users (such as help desk and call center staff) be treated as authenticated users. This is usually much simpler to implement, since all internal users are already authenticated on the network, and both LDAP and SSO are relatively simple to implement.

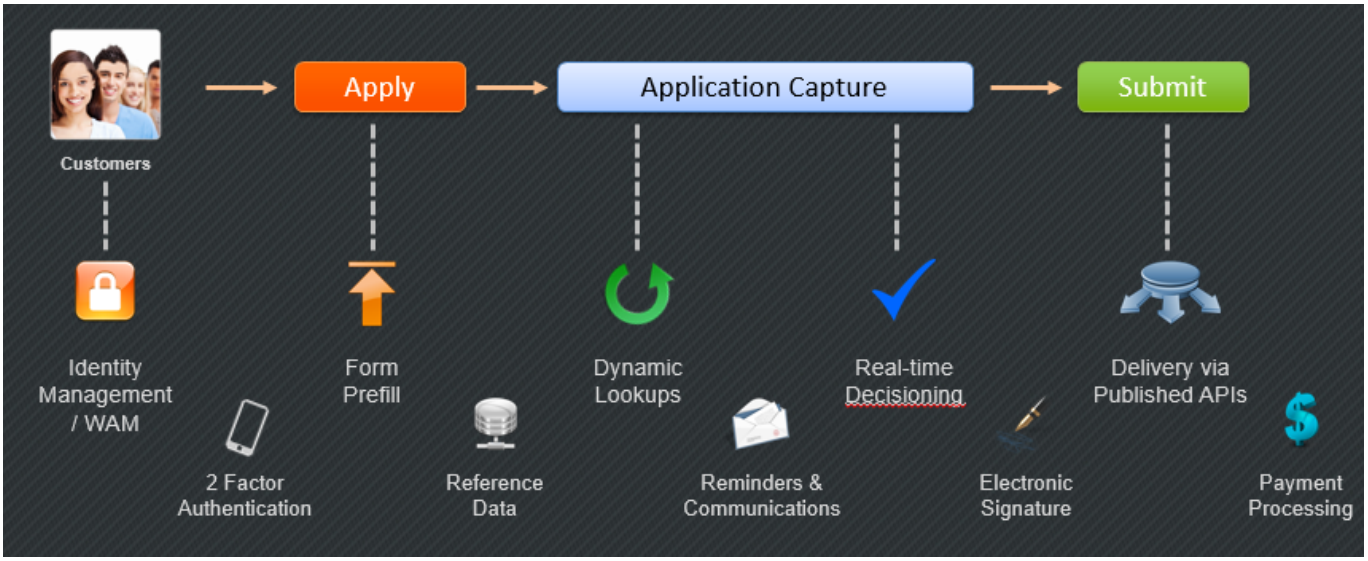
In some cases authenticated end users may be required for business reasons, and Transact does support this extremely well - but bear in mind that it may affect your time and cost of implementation.

Extension Points

Integration is performed in various areas of the transaction life-cycle though the use of defined extension points. Developers can inject code or other invocations at these defined extension points. The main points are shown in the interaction or sequence diagram below, although some extension points have been omitted for simplicity.

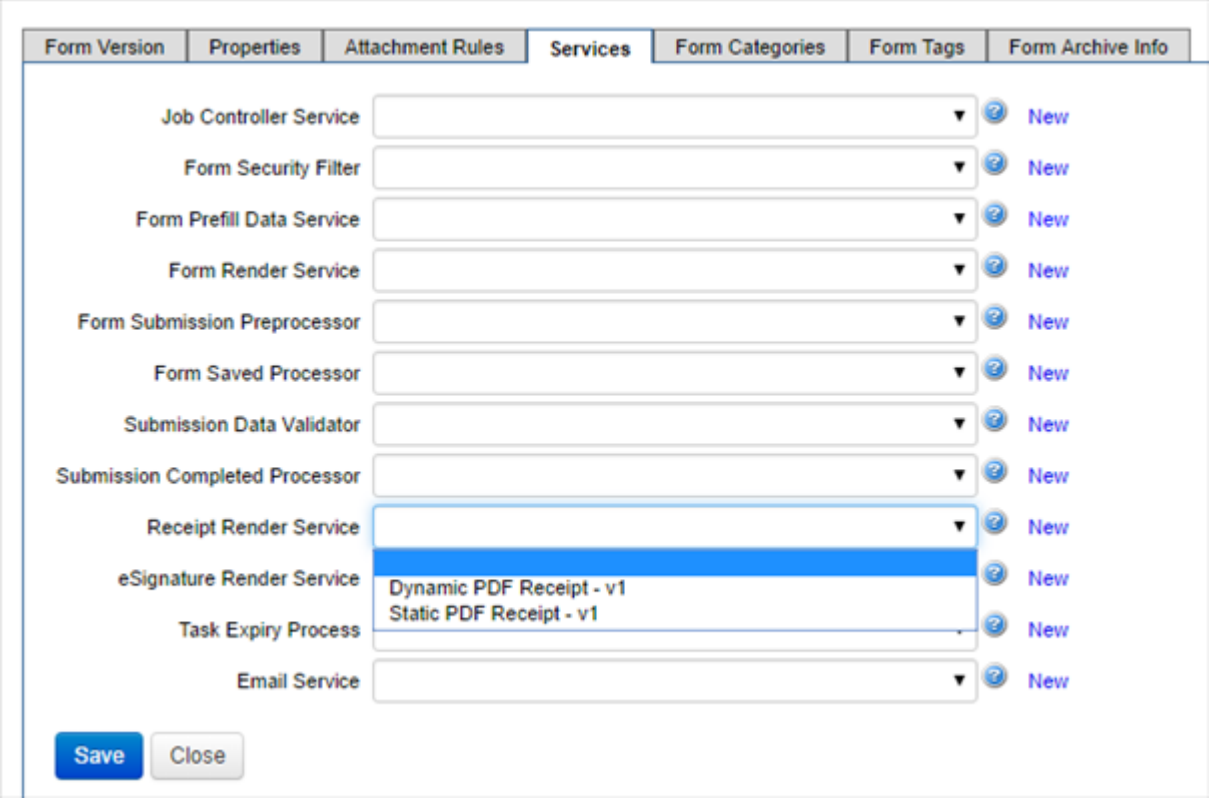

Customer Form App
Transact Server
Core Systems
request Form App
execute 'Form Version Selector' service
execute 'Form Prefill Data' service
execute 'Tracking Number' service
render Form App
HTML Single Page App
perform data entry
execute business rules
call Dynamic Data lookup
execute 'Form Dynamic Data' service
Web Service SOAP/REST
return JSON data
executes validation logic
submit Form XML
execute 'Submission Preprocessor' service
execute 'Receipt Number' service
execute 'Job Controller' service
execute 'Submission Completed Processor' service
scheduled job
execute 'Delivery Process' service
Web Service SOAP/REST
Customer Form App
Transact Server
Core Systems

There are many different points in the life-cycle of a user interacting with a form, between the time that the user first clicks on a link to open the form, and the time that the transaction is complete. We won't go into all the details, since there are over 20 different life-cycle events, but some of the key events are shown in the diagram below.



Some of these points are handled by built-in capabilities within the Journey platform, but others require integration with third-party or organization systems.

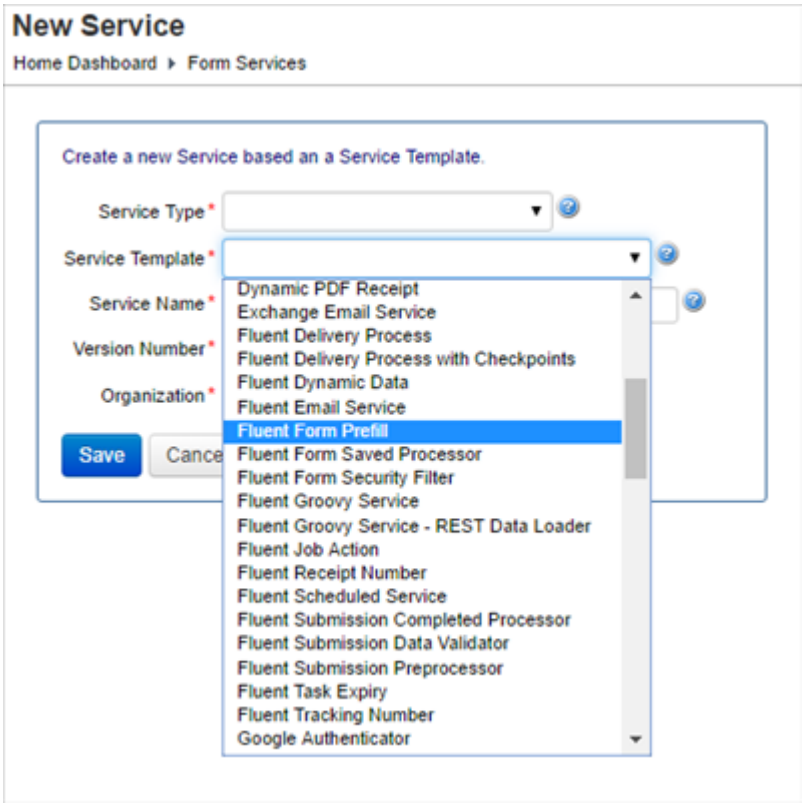
Manager provides extension points at many of these life-cycle events to allow integration between Manager and these other systems. The example of these extension points is shown below.



Each of the drop-down lists specifies a service that may be invoked at that particular point in the transaction life-cycle. You may select an existing extension point (as is shown in the Receipt Render Service in the screenshot), or you may create a new extension point by clicking the "New" button.

In order to build a new extension point, you author the extension point in a language called Groovy. Groovy is a simple yet powerful scripting environment that most developers find intuitive and productive.

When you create a new service, you are not simply presented with an empty screen and begin typing. Over the years, we have collected a series of templates from real projects that can be used as starting points for development of different types of services. A subset of the available templates is shown in the screenshot below:



The template will create a starter script which allows a programmer to simply "fill in the blanks" for their particular service.

Calling External Services and API

Once you start building a service, you will often need to invoke external services to actually perform the work. These services will usually be exposed as either SOAP or REST web services. These services may already exist, or they may be built as part of the Manager project. They can be built in a number of different ways, including hand-built using coding languages, or the use of specialized integration system or an Enterprise Service Bus.

Libraries for calling these external services are built into the Journey platform as shown in the example below.

Examples

The example below performs REST request and returns a JSON object

```
import com.avoka.tm.http.*
import com.avoka.tm.util.*

Map params = [:]
params.id = request.getParameter('customerId')
params.account = request.getParameter('accountId')

String username = svcDef.paramsMap.username
String password = svcDef.paramsMap.password

// execute GET request and return a HttpResponse object
HttpResponse response = new GetRequest('https://service.mycorp.com/secure/rest/accounts/')
      .setParams(params)
      .setBasicAuth(username, password)
      .execute()

if (response.isStatusOK()) {
    // get Path object from the response
    Path path = response.getPathContent()
    ...

} else if (response.isStatusNotFound()) {
    // object not found
    ...

} else {
    throw new RuntimeException(response.statusLine)
}
```

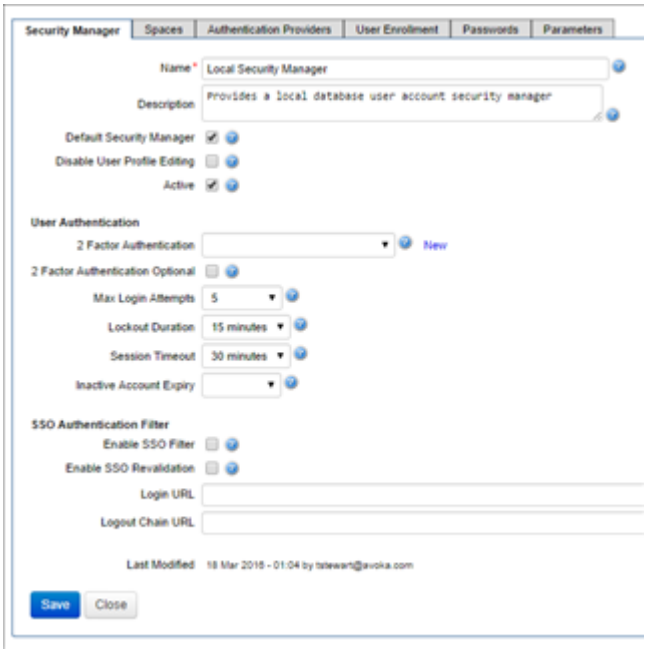
This illustrates how easy it is to invoke these external services. Of course, more sophisticated services can be developed.

It is often also necessary to invoke Transact Manager itself in order to get more information from the system, or to update the transaction. This is also available through a set of comprehensive Transact API's.

Web Site Integration and Single Signon

It is almost always required to integrate a form into an existing web site. Usually this is achieve through a simple link or button. In the past, more complex integrations were performed, such as embedding forms within existing html pages, although in modern sites this is rare, largely due to the needs of smaller screens and mobile devices.

It is also sometimes necessary to provide single-sign-on integration with existing systems. This is provided by an extensible single-sign-on mechanism that is part of the security management sub-system within Transact. SAML, oAuth and two-factor authentication are supported. A screenshot is shown below:



Form Level Integration

Sometimes it is useful to perform integration directly between a form and another service. This can be performed in two ways:

- Directly between the form and the service. This can be implemented directly using JavaScript in the usual way. Generally the integration will be packaged up into re-usable component that can be simply dropped into any form that needs it.
- Using Transact Manager as a proxy service. This mechanism is known as Transact Dynamic Data services. The form uses a standard mechanism for passing data in and out, which both programmers and non-programmers can easily use and configure. This invokes a service on Transact Manager (written in Groovy). The Groovy service has all the power of the Groovy programming language and supplied libraries, and so arbitrary complexity can be simplified to ensure that the service exposed to the form is easy to use. There are several advantages to implementing the system this way, including ease of use, centralized configuration, integration with Transact licensing audit logs, and more. This is the technique usually used.

Next, learn about the five Journey platform integration areas.

Integrate Manager with SharePoint

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager allows you to integrate with **SharePoint**[1], using a Java integration component, that supports the following SharePoint interfaces:

- **Copy.copyIntoItems**
- **Copy.getItems**
- **Lists.addAttachment**
- **Lists.getListItems**
- **Lists.getListCollection**
- **Lists.updateListItems**
- **Lists.checkInFile**
- **UserGroup.getUserInfo**
- **UserProfileService.getUserProfileByName**
- **Webs.getWeb**
- **Webs.getWebCollection**

Security of your applications is our priority, so, before using the SharePoint web services, you must ensure that a user, which you use to authenticate the connection, has the following permission: **Use Remote Interfaces - Use SOAP, Web DAV, the Client Object Model or SharePoint Designer interfaces to access the Web site** .

To call methods of SharePoint interface, you need to establish a SharePoint Connection. This can be done with a Groovy service. The **com.avoka.component.sharepoint.SPClient** class uses a Microsoft SharePoint Client class with web service factory methods.

The Groovy script example below illustrates using the **SPClient** object to create a **List SharePoint** web service object, which is then used to look up a collection, such as a **List**. In this example, configuration Service Parameters are accessed via the **params** map object .

[1] SharePoint is a web-based collaborative platform that integrates with Microsoft Office.

```
import com.avoka.component.sharepoint.SPClient
import com.avoka.component.sharepoint.service.List

// Create a SharePoint client using the service params map with the configured connection
details
def spClient = new SPClient(params.username, params.password, params.url, params.domain)

// Create a List web service
def listService = spClient.getLists()

// Get a List collection map
def collectionMap = listService.getListCollection(params.siteContext)
```

For more information, see API documentation.

Next, learn about the form dashboard.

Five Journey Platform Integration Areas

Journey Platform | All Personas

The Temenos Journey Platform comes with the following five integration areas:

1. Delivery
2. Form discovery
3. Pre-Population to accelerate data entry
4. Streamlining of data entry
5. Execution

Let's take a look at each integration in more details.

Delivery

Delivery integration is extremely important to every organization. Unless you deliver the data collected to the organization in some way, you are ultimately not gaining any value. This area is listed first, because it is almost always implemented, whereas many of the other integration areas are optional.

There are generally three types of artifacts that are delivered:

- **Receipt.** A human-readable version of the form, usually in the form of a PDF document. This is known as a receipt, and can be generated in several different styles.
- **Data.** A computer-readable version of the data, in the form of an XML document. This is suitable for integrating into other systems.
- **Attachments.** Any supporting documents, known as attachments. These may be photographs, scanned documents, or electronic documents of various types.

The Journey platform has a large number of different ways in which deliveries can be performed, ranging from simple to sophisticated.

Delivery Type	Description	Examples
Email	The PDF, XML and attachments are attached to an email, which is then emailed to an individual or group email box for processing. The person/people monitoring this email account process the PDF, XML and/or attachments manually. This approach is usually the simplest option for low volume and/or low value transactions, but it can also be used very successfully as a short term solution while more sophisticated integrations are being built.	There is really not very much to do, other than making sure Manager has access to a mail server, and identifying the correct email recipients.
Fax Gateway	If your organization already has a fax gateway, then the inbound faxes will generally be delivered as attachments to an email message, or as files in a folder. Manager can easily be	Avoka have previously integrated with the RightFax fax gateway.

	configured to do the same.	
Web Services (SOAP)	Managerhas a SOAP API that lets external systems retrieve submissions (including their PDF, XML and attachments). This API can be called by any client that is SOAP-compliant. This approach has the advantage that no firewall holes are required since this is a pull invocation.	Several of our customers have used the SOAP API in order to retrieve submissions.
TIA	The Journey platform includes a pre-built (but extensible) application known as the Integration Agent. This Java-based application uses the SOAP API to retrieve submissions from Manager, and deliver these into external systems. A number of standard integration end-points are provided, such as folder drop, or customers can customize TIA to perform any type of integration that is possible from Java.	The Australian Dept of Industry uses TIA to deliver submissions into a number of different agencies in a multi-tenanted environment. This provides a level of separation between the Dept of Industry who manage and administer the system, and the agencies who only receive submissions destined for them.
ESB / Mulesoft	MuleSoft is an Enterprise Service Bus. Avoka has developed a Mule connector which uses the SOAP API to retrieve submissions from Manager. Once a submission has been retrieved, it can be easily integrated with any of the systems that Mulesoft supports, which are listed here: https://www.mulesoft.org/connectors Mulesoft also has a sophisticated data mapping tool for mapping data from one form to another.	
REST API	Manager has a REST API that that lets external systems retrieve submissions (including their PDF, XML and attachments). This is similar to the SOAP API, but is a more modern protocol, easier to use, and supports high availability architectures.	
TM RDC	The Manager REST Deliver Client is a very simple and self-contained application built in the Go language. It uses the REST API to retrieve submissions from a Manager server. The default implementation is to store these submissions in a folder/directory. However, source code is provided, and can be customized to perform other types of integrations.	
Groovy	Manager allows delivery integrations (and other types of integrations) using the Groovy scripting language. Groovy is a simple yet powerful scripting language that has been built into Manager. Groovy gives you access to many of the internal Java API's provided by Manager, and also has cap-	Examples!!! They abound.

Java	abilities for calling third party services and systems, including REST and Web Service API's, so it is a powerful tool for building sophisticated integrations. Groovy is often used to "glue" Manager to another system, rather than being used as a programming language. A full set of the available Manager API's can be provided upon request if desired. For even more sophisticated integrations, Manager allows Java modules to be developed and uploaded into Manager, where the methods they contain can be invoked at defined extension points. This provides access to the full power of Java, plus all of the Manager Java API's.

In addition to these general purpose delivery mechanisms, several specific integrations have been developed for particular end-points.

These have been integrated into Manager as pre-built integrations, and they include:

Delivery End-point	Description
Microsoft SharePoint	This integration allows submissions to be delivered directly into a Sharepoint instance.
SalesForce	This integration allows for: 1. OAuth 2 authentication against SalesForce identities. 2. Delivery of submission information directly into SalesForce for marshalling into destination objects. 3. API level form assignment to known and anonymous users directly from SalesForce, encapsulating APIs on Manager. 4. UI objects to present a categorized list of available forms directly inside SalesForce, for either opening immediately or pushing to SalesForce contacts. These may be automatically pre-filled with data from SalesForce objects. 5. UI objects to review the active and completed forms for a particular object within SalesForce.
Marketo	Creates or updates the visitor's Marketo LeadRecord using the submitted form data.
Adobe LiveCycle	Manager can deliver submissions directly into Adobe LiveCycle Process Management. Process Management is a BPM system that allows back-office business processes to be easily developed.

Finally, Avoka staff and our customers have used some of the above capabilities to integrate with a large number of off-the-shelf and custom systems. Some of these include:

- PEGA (BPM system)
- SAP (ERP system)
- Microsoft Dynamics (CRM system)
- FTP/SFTP (file transfer protocols)
- Shift (a dealer management system)
- Carflow (a dealer management system)
- Huddle (a document management system)
- K2 (a BPM system)
- IBM Lombardi (a BPM system)
- Telco systems (several bespoke systems at a large Telco)

The Delivery Service extension point with Manager provides a number of built-in added value services, including:

- **Retry capabilities.** If a delivery fails, it can be automatically or manually retried. This can be helpful for cases when a delivery end-point is temporarily unavailable.
- **Delivery checkpoints.** If a delivery consists of multiple steps (for example, send an email, then update a database, and then store a file in a directory), then each part of this delivery can be set up as an individual checkpoint. If one of the steps fails, but a previous step has completed successfully, then when the delivery is retried, it will start on the first incomplete checkpoint, rather than from the beginning. This would ensure that, for example, only one email ever gets sent.
- **Alerts.** The delivery subsystem provides administrative alerts to notify system administrators when a delivery has failed so that the underlying cause can be corrected.

Form Discovery

Every unique form in Manager has an associated URL link that you can place on your site. Any user who clicks on the links will launch the form. You can also use the form URL in any other way such as in an email. You can also link to the form from any type of "form finder" technology, or a wizard that assists a customer to find the appropriate forms. You can also embed links to forms in outbound communications such as email or social networks.

Another way for customers to complete a form is to send a customer a link to a form that has already been partially completed on their behalf. Manager includes a Task API, that allows you to create a partially completed form, generate a custom URL, and then pass that URL to the end user. This can be used to "push" forms from CRM or other systems.

An older approach is to embed the Journey platform form directly inside the organization's web site. This approach is rarely used, mostly because of the advent of smart-phones, which lack the screen real estate to display both the existing site and the form in the same screen. There are also other issues with this approach, including the importance of removing distractions and navigation links from the task-oriented process of completing the transaction, as well as the difficulty of effectively designing responsive user interfaces in two separate web pages that effectively work together.

3 Pre-Population - Accelerating Data Entry

Manager provides many different techniques for pre-filling forms. One of the simplest ways to implement this is to pass pre-population data as URL parameters in the URL that links to the form. These parameters can be customized on a per user basis in your existing web site if required. Another way is to retrieve pre-population data from data stored in cookies.

Alternately, Manager has many extension points where calls can be made to external systems, and the results of these calls can be injected into the form's data before it is presented to the user. The sequence diagram above highlights the key extension points. The extension points can be implemented as Java or Groovy calls to external systems. These extension points also have access to many of the internal Manager API's.

Manager implements a form receipt-number generation service, to ensure that each form has a unique code. This service can be configured to auto-generate different styles of receipt number, or it can be extended to call external systems to generate the receipt number. This allows the form receipt number to match a reference number in an existing system.

Manager implements several different techniques for passing identity from an existing system to Manager, including SSO/SAML, LDAP integration, and others. If the user's identity has been established, then calls can be made to back-end system using this identity to retrieve user-specific data, which can then be injected into the form before being presented to the user.

Streamlining Data Entry

Once the user is actually within the form, we want to make their experience as streamlined as possible. One way we can do this is by designing and creating engaging, simple to use, guided experiences using Composer.

We can also streamline their experience by providing contextual data within the form that:

- Accelerates their data entry
- Validates their data entry
- Provides information to them that helps them to make decisions

Usually this type of integration is provided by custom user interface widgets within the form itself. There are generally two different mechanisms that these widgets use to communicate to external systems.

Mechanism	Explanation
Direct	The widget communicates directly to the third party service, usually (but not always) using JavaScript, REST and JSON. This is most useful for cloud-hosted third-party services that don't require usage keys, identity verification, or license/billing, and are not subject to security concerns.

	A Composer widget communicates with a standardized dynamic data service within Manager. The dynamic data service protocol is predefined, and uses JSON data as the payload. The dynamic data service then communicates to the desired back-end service to verify or retrieve data, perform a calculation, or verify data. The results are then passed back to the widget.
Dynamic Data Service	Dynamic data services are useful for several scenarios: • Licensing. Some external services require the user to specify a usage key of some sort. • Billing reconciliation. • Security (secure channel to backend services) • Data transformation or consolidation, protocol translation. Composer includes several pre-built widgets that support Dynamic Data calls, and custom widgets can also be created.
Hybrid	Occasionally it is useful to build widgets that use a combination of direct access and dynamic data services. For example, an initial call may use Dynamic Data to establish credentials with a third-party, but then use direct access to the third party service for greatest efficiency.

Some examples of pre-built services that communicate with third-party services include:

- Experian, which provides postal address lookup services.
- GreenID which provides identify verification services.
- Manager Reference Data, which allows data to be stored in Manager, and displayed in a variety of different ways.

Many our customers have used the generic Dynamic Data-aware widgets to expose their back-end web services, REST services, databases and bespoke systems in a variety of ways within the form.

Execution

Execution is largely about adding additional value to the overall transaction, after the basic data completion step has been completed. Some of these integrations are part of the process of completing the transaction, and are visible steps to the end users, whereas other types of integrations occur behind-the-scenes.

The following examples list some of the types of integrations that are possible at execution time with Transact.

Integration	Description

Manager can integrate with Electronic Signature providers in order to provide an improved level of signature compliance. (Note that Manager provides several types of electronic signature capability directly in our system, without requiring third-party services.) Manager has already been integrated with DocuSign and Silanis, but the same principles can be extended to similar electronic signature providers.

Electronic Signatures

Electronic signatures are usually implemented in one of two ways:

1. **Immediate signature.** A receipt (or document of record) is created, and submitted to the third-party signature provider, along with signatory meta-data. The signature provider processes this request, prepares the document for signing, and returns the signatory id to Manager. It then redirects the user to the signature provider web site, where the document will be presented for signature. The signature providers may also request additional identification information. Once the user has signed the document, the signature provider will redirect back to the Manager web site, so that the transaction can be completed, delivered, etc.
2. **Signature by email.** A receipt is submitted to the signature provider, along with signatory meta-data.The signature provider will prepare the document for signing, and then send an email to the signer for processing. Manager will notify the applicant to check their email address. The user will click on the link in the email, and complete the signature process in the signature provider's web site. Manager will be notified of the signature completion, and will complete the transaction, including delivery etc.

The immediate signature approach is a more seamless experience for the end user. However, the signature by email approach can have a higher level of identity for the signer, because it implicitly validates their email address. It is also generally the only way in which a document can be signed by multiple signers - this is usually supported by signature providers in either parallel or sequential flows.

Payment processing

Many transactions require an associated payment. The Avoka form will calculate the fee, and also specify what payment methods are available. When the form has been submitted, Manager will redirect the user to a payment provider's website, along with meta-data about the payment, such as the amount. This site is known as a Payment Gateway. The Payment Gateway will process the payment, and then redirect back to Transact, passing payment meta-data. Manager will inform the user of the outcome, as well as store the payment-meta-data to allow for reconciliation reports to be run.

never stores or processes credit card or banking information directly. This is a specialist service that requires specific levels of security and privacy compliance, and is always performed by a specialist payment providers. Journey platformhas built-in integrations with several providers, and other providers can be implemented as required.

Additional Verification

Additional Verification of the user's information for purposes such as credit scores or identity verification can be implemented within Manager itself. These can also be implemented post-delivery in the organization's back-end processes, but performing them within Man-

	ager provides the ability for the end-user to be informed of the result while they are still within the transaction, and therefore have an opportunity to retry the application and correct the information.
Review and Approval	Manager includes a simple but powerful review and approval capability known as Collaboration Jobs. Transactions that have been completed by end users can be progressed through several review/approval steps, by different individuals or groups. Steps can also be designed to retrieve information, perform processing, or make decisions. Review and approval is often implemented in larger and more sophisticated BPM or Workflow systems, but the Review and Approval capability in Manager provides a number of benefits: • There is no need to purchase or maintain a BPM or Workflow system. • Review and Approval is very simple to configure, and very easy to use for the reviewers.
Document Generation	Manager produces a receipt automatically, in PDF format, providing a document of record for the customer and for the organization. Manager can produce a document of record using: • Composer-generated receipt templates (automatically) • Hand-created Adobe LIveCycle Designer receipt • Hand-created Adobe Acroform receipt (also known as pixel-perfect receipt). However, in some cases, the documents that need to be created are more sophisticated or different to what can be achieved in Manager natively. There are a number of systems known as Customer Communication Management systems which are designed to produce customer communication such as offers, receipts, contracts and information packs. These systems can be used by Manager to generate the document of record.

The Economics of Integration

It is very important to remember the consider why we do integration - we don't do integration just for the sake of it, we do integration order to provide specific value to the organization. Integration can often be the most time-consuming and costly part of any implementation. It's very easy to fall into the trap of doing integration because it's possible or seems obvious, but it is wise to always step back and consider what value integration is providing to the overall business objectives relative the cost of implementation and future maintenance.

Why do we do integration?

Usually we do integration in order to achieve operational efficiencies. Integration usually saves humans from having to perform manual tasks, which in turn reduces costs. These efficiencies may be realized by the end-user (eg save typing and frustration through pre-population) or by the organization itself (save money and reduce errors by eliminating

Assess the Value of Integration

A survey performed by the Forrester group in 2015 asked "What initiatives are likely to be your organization's top business priorities over the next 12 months?"

The top three responses were:

- Grow revenues (77%)
- Improve the experience of our customers (74%)
- Reduce costs (61%)

It is interesting that improving user experience is regarded as significantly more important than reducing costs. And that growing revenues is the most important criterion in the entire survey.

- The Journey platform is all about improving customer experience, and growing revenues (although it also leads to improving efficiency).
- Integration is all about improving operational efficiencies, and reducing costs.

It is important to keep in mind that in many cases, the primary goal of implementing the Journey platform is improving user experience, and growing revenue. You should therefore consider very carefully the value of doing integration compared to the cost. It may be that you can achieve a large proportion of the value of the Journey platform without doing any integration.

We're not saying that you shouldn't do integration, just that you should be aware of the costs and the value.

Consider Deferring Integration

Another factor to consider is time to market. Manager can be implemented very quickly, and realize important benefits such as improved user experience and improved conversion rates. However, integration, because it touches back-office systems, needs to be very carefully developed and tested, and this can often take a significant amount of time.

One common approach is to implement Manager with little or simple integration, and realize the primary benefits as early as possible. More sophisticated integration can then be performed at a slower and more careful pace - and because Manager delivers the data in XML format, it's easy to add integration later. Integration can be prioritized for the most important transactions first, with lower value or lower volume transactions deferred till later.

Some of our most successful implementations had started with minimal integration at the beginning, but over time the integration has been phased in separately.

Consider Leveraging Existing Paper/image-based Processes

Ideally the goal is to completely replace paper/images with all-digital transactions. However, in reality we rarely achieve 100% success - there are almost always going to be some customers who insist on hand-writing on paper. This means that you almost always needs to maintain your existing back-office processes which currently handle paper or images of paper.

Since you're going to need to maintain these back-office processes anyway, one useful approach is to augment or plug into these back-office processes using digital artefacts, rather than trying to create a completely all-digital back-office process. In other words, deliver a digital document (generally a PDF) into your existing paper- or image-based systems. If you do this, integration is usually very simple and quick.

Hidden Costs of Process Re-engineering

Process re-engineering is costly. This is not just about technology, but about people, skills, training, management, and related costs. Overall change management is often overlooked, and can be hugely costly. You don't want to introduce a new digital channel, and at the same time, completely re-engineer your back-office processes. Instead, one option is to simply plug digital artefacts into existing back-office systems. You get the benefits of a new digital customer-experience, but you don't have to retrain anyone, build new processes and procedures and manuals, hire new staff with different skills, or put new managers in place. You have the opportunity to re-engineer and optimize your existing back office processes at a later time.

Our strong recommendation is not to combine process re-engineering with a (user-experience improvement) project, otherwise there is a possibility that both projects will fail. Process re-engineering and change management are hard and expensive, and shouldn't be embarked upon lightly.

Next, learn about Journey platform development.

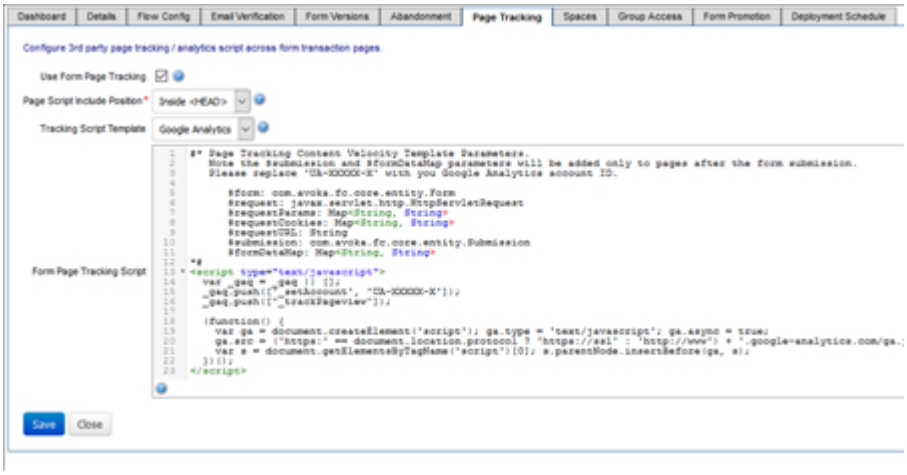
Google Analytics Virtual Page Integration

Journey Platform | All Personas

To include page tracking via Google Analytics, each wizard page will fire a "page view" with title and url.

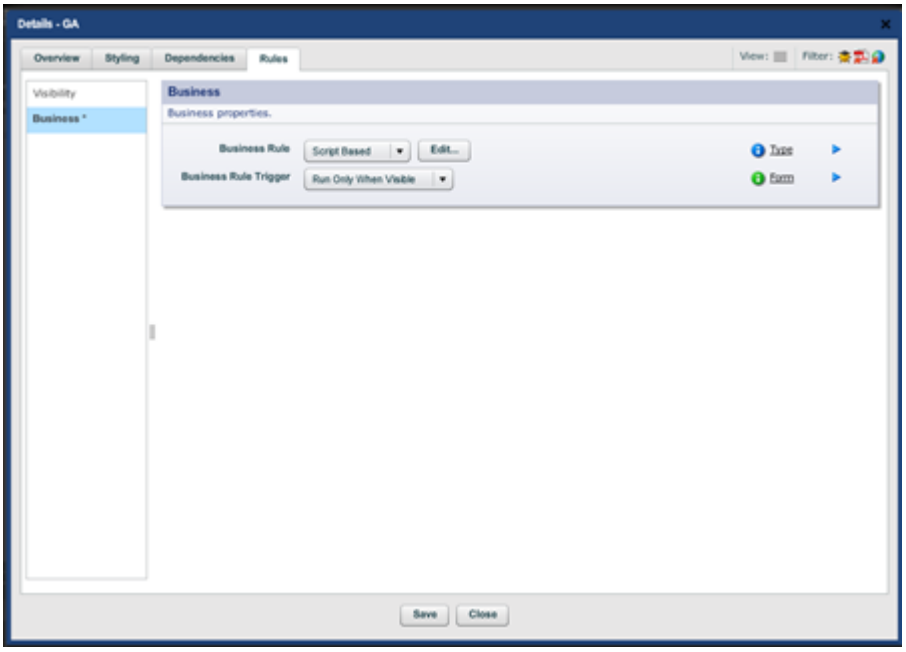
Add GA Code to Head of Page

Navigate to the "Page Tracking" tab, select the "Inside <HEAD> option" and paste in your Google Analytics code. See example below.



Create Business Rule to call the ga method on page change

Under the business rules triggering conditions check the following condition.



In the script you will need to create the virtual page views and then call the ga pageView method. See example script below.

This script uses the current page number to reference the correct object.

```
var gaObj = {
"1": {
"title": "First page",
"page": ""

},
"2": {
"title": "About you",
"page": "about_you"

},
"3": {
"title": "Your property",
"page": "your_property"

},
"4": {
"title": "Your new loan",
"page": "your_new_loan"

},
"5": {
"title": "Your details",
"page": "your_details"

},
"6": {
"title": "Your employment",
"page": "your_employment"

},
"7": {
"title": "Your finances",
"page": "your_finances"

},
"8": {
"title": "Application complete",
"page": "application_complete"

}

}
```

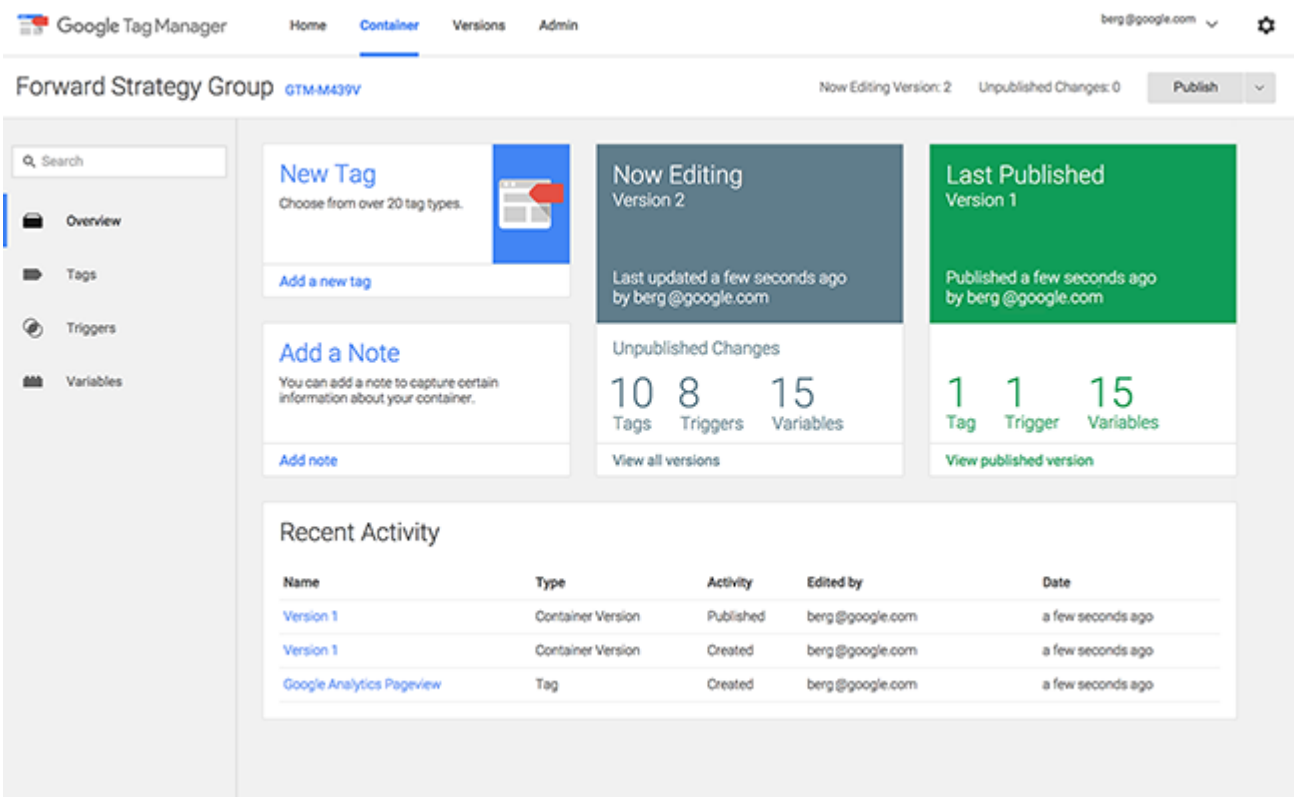
```
var baseUrl = "/macquarie/servlet/SmartForm/";
var currentPage = sfcInternal.av_wizard_findCurrentPage();

if(typeof ga != 'undefined') {
ga('send', 'pageview', {'page': baseUrl+gaObj[currentPage].page,'title': gaObj[cur-
rentPage].title});
}
```

Next, learn about Journey platform development.

Add Google Tag Manager Support

Journey Manager (JM) | Form Builder | System Manager / DevOps | 5.1 & Higher

Google Tag Manager (GTM) lets you launch new tags with just a few clicks. The result? You' enjoy faster, tighter control over your digital marketing and analytics software.



In combination with manage, GTM will allow you to collect analytics information by adding tags to any field, or action in your form without having to make any code changes. This how-to article describes how to add GTM in Transaction Manager and how to add tags to your form. We also have added a number of best practices as well as possible mistakes that you'd like to avoid when using GTM.

Online web forms created with Composer have a single-page architecture, whereby all of the content exists in a single rendered html page, but the user navigation flow is controlled through hide/show logic of different sections (or logical form pages). Whilst providing a better user experience, this means any standard Google Tag Manager (GTM) configurations that utilise page views to track web-site traffic will stop tracking the user movement once the form is opened - none of the in-form navigation will be tracked.

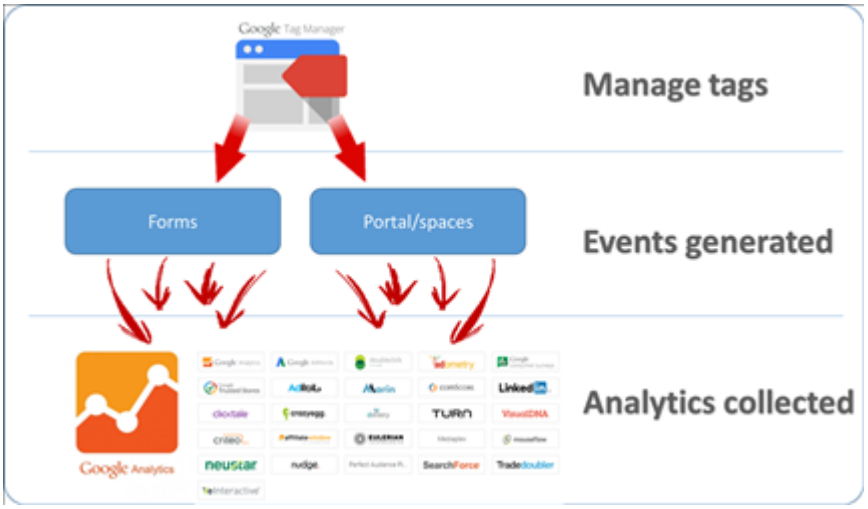
The Google Tag Manager (GTM) container discussed here provides a configurable user interface to enable web-site analytics for these forms - tracking user movement through the different sections of a form, as well as improved tracking for the separate pages that a user will be directed to after completion of the form.

This implementation includes Tags for the Google Analytics framework, however tags could be adapted or created to support any analytics engine.

The GTM container is a fully functional entity that can be applied to any Transact Composer environment. Installation and configuration can be summarised as a series of steps:

1. Import the container into a Google Tag Manager account
2. Configure Transaction Manager to enable Analytics with this Tag Manager Account

This how-to guide assume that you already have a GTM account with the "manage" permission. You will also have to download the Temenos GTM container.



These steps only need to be performed once for a Transaction Manager installation, but note that there is additional configuration required for each form that is to be added to the GTM implementation. This extra configuration will be covered below.

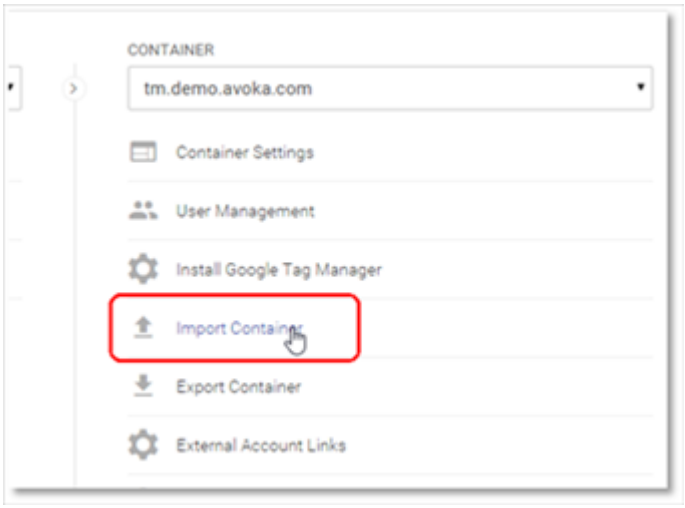
Importing the Container into Google Tag Manager

The Avoka GTM Container is designed to run in isolation - the tags pre-configured within it will only produce page views for Avoka forms and their related pages. For this reason it is safe to use either option for importing the container into GTM - either merge it into an existing container, or create a new one.

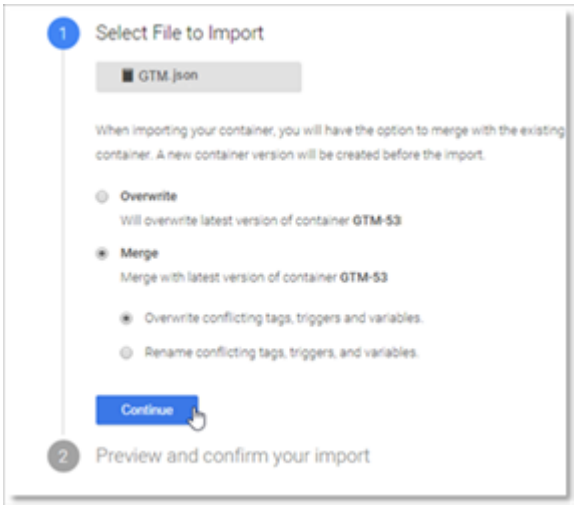
It should be noted however that if choosing to merge it into an existing container, care should be taken that pre-existing tags/triggers do not adversely affect the Avoka pages. Additionally, if new tags are being created for Avoka forms, correct use of triggers should be maintained to ensure that the tags only fire in the correct places - and not in other pages that may be connecting to the same GTM container.

Steps for importing the container:

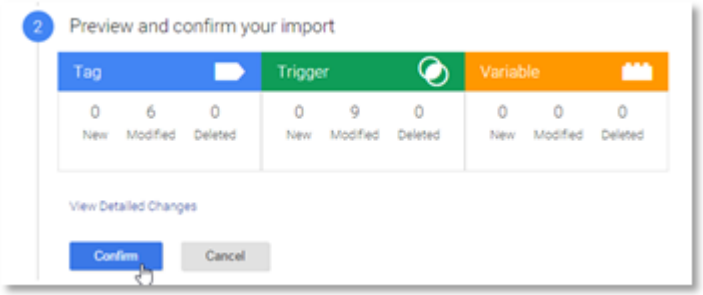
1. Login to Google Tag Manager and select the correct container to import into (Navigate to the Accounts tab and select the relevant container - or create a new container if required)
2. Navigate to the Admin tab and select Import Container from the Container menu.



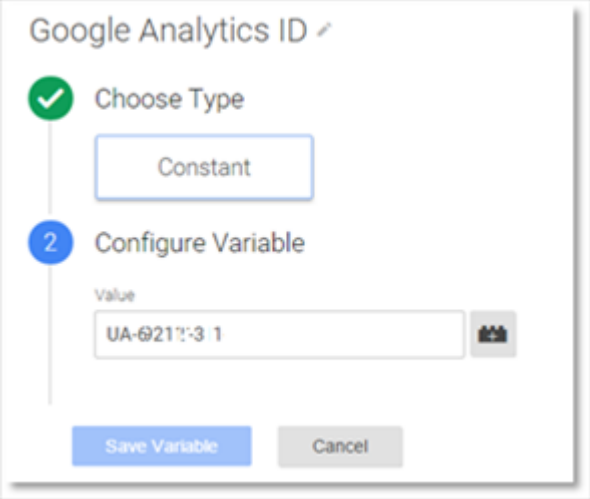
3. Under Select File to Import navigate to the Avoka container file on the file system
4. Select to either overwrite the existing container (for new containers only) or merge and press the Continue button.



5. After viewing the Preview hit Confirm to import the container.



6. Navigate to Container > Variables and select the Google Analytics Id variable
7. Inside this variable, enter your Google Analytics Tracking Id. This is important to ensure that page views are being sent to the correct Google Analytics account.



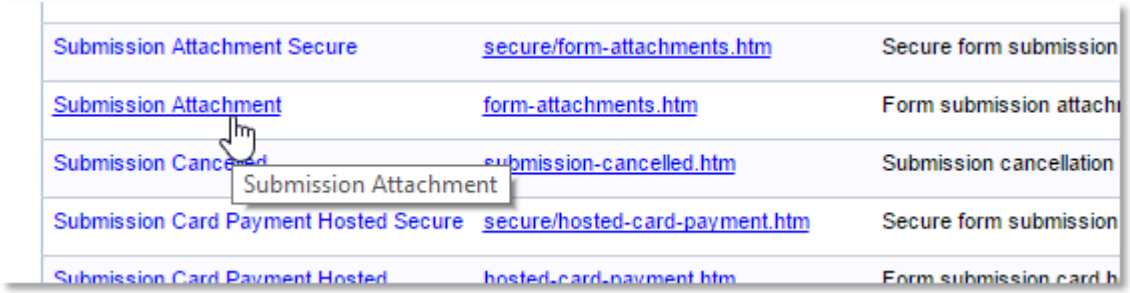
Configure Manager to Enable Analytics

As stated, a Composer form is built using a single-page architecture, where all of the content added to the form in the Composer tool is rendered in a single html page, and hidden/shown as appropriate via browser scripting. However, the actual form content only forms part of the user-experience picture.

Landing pages, Save and Resume pages, Attachments pages and Submission Complete pages that may be presented to the user before or after they have filled in the form. These pages are also critical to deriving useful data from an analytics engine, and therefore need to be included in the GTM implementation.

Avoka TM maintains global versions of these pages that are used by all forms by default, so it becomes important to be able to identify which form the page has been opened for. For the most part this can be achieved by using the formCode URL request parameter that is required to open the form, however for some of the post-form pages (Submission Complete, Attachments, Save Confirmation) this parameter is not provided. In these cases GTM has been setup to look for a form code within the html of the page that's been rendered, by finding a hidden element with the id of submission_formCode. The pages do not automatically include this hidden element, but any TM user with the Portal Edit permission can add it with a few simple steps:

1. After logging into TM, navigate to System > Form Spaces (called Portals/Modules in older versions of TM)
2. Find the space that contains the form you would like to add GTM capability to, and click on the name of the space.
3. Find the page to be edited, and click on the name entry.



4. Navigate to the Pages tab.
5. Place the following html inside the page content. The location is not critical, but ensure it is not inside any conditional logic (with markup such as **`#if ($session.emailed)`**:

```
<input type="hidden" name="formCode" id="submission_formCode" value-
e="$form.formCode"/>
```

6. Save your changes (the button at the bottom of the page). These changes will need to be applied to the all pages that you would like to track even ts on.

Enable Analytics for each Form

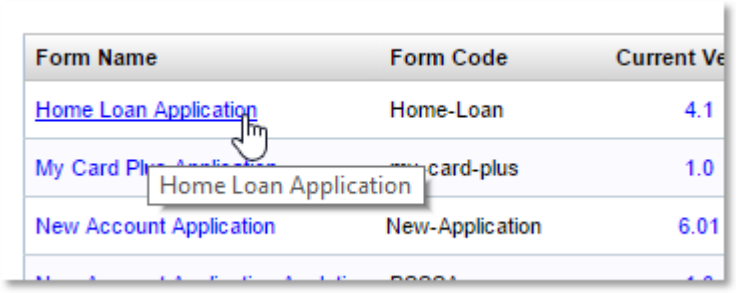
Part of the configuration exercise is to enable the creation of analytics data on a per-form basis. This allows greater control over which forms are tracked (and when) as well as tailoring the exact look of the page views that are generated. Enabling analytics for a form involves steps in both Avoka TM and GTM.

If any of these steps are not performed, no analytics data will be received for the particular Avoka form - even if it's published to an environment that already has other forms sending analytics data.

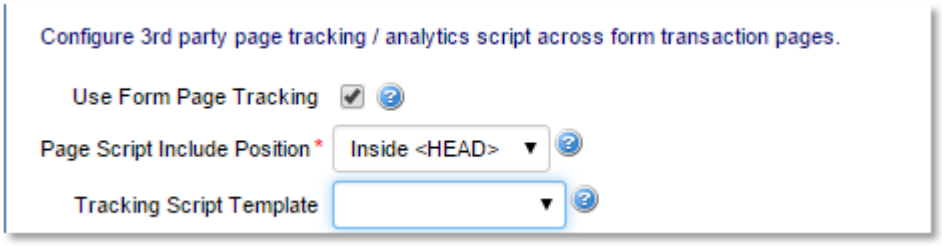
Having said that, it only needs to be done once for each form - thereafter, if changes to the form are made in Composer, and published to the TM environment, this setup does **not** need to be repeated.

Add GTM container script to the form

1. Navigate to Forms -> Forms
2. Click on the relevant form name to open the details



3. Navigate to the Page Tracking tab
4. Select the checkbox marked Use Form Page Tracking and select Inside HEAD in the Page Script Include Position dropdown



5. In the Form Page Tracking Script text box, paste the GTM script that adds your GTM container to the page. This script can be obtained from GTM by navigating to Admin > Install Google Tag Manager



6. Save your changes.

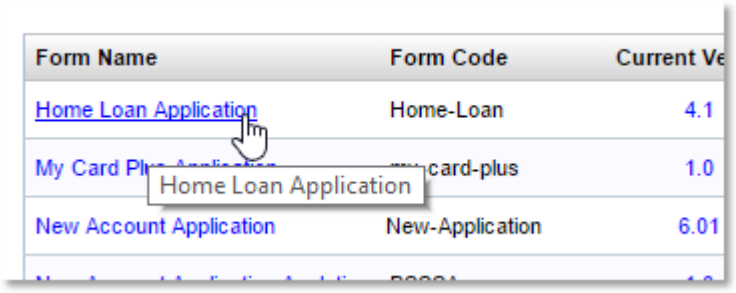
Additional Forms Pages

Manager has the ability for a form to maintain its own version of these pages - so that different content may be displayed to the user for a particular form. However, this setup is optional, and if not configured, the default pages referred to in section 2.2 are used.

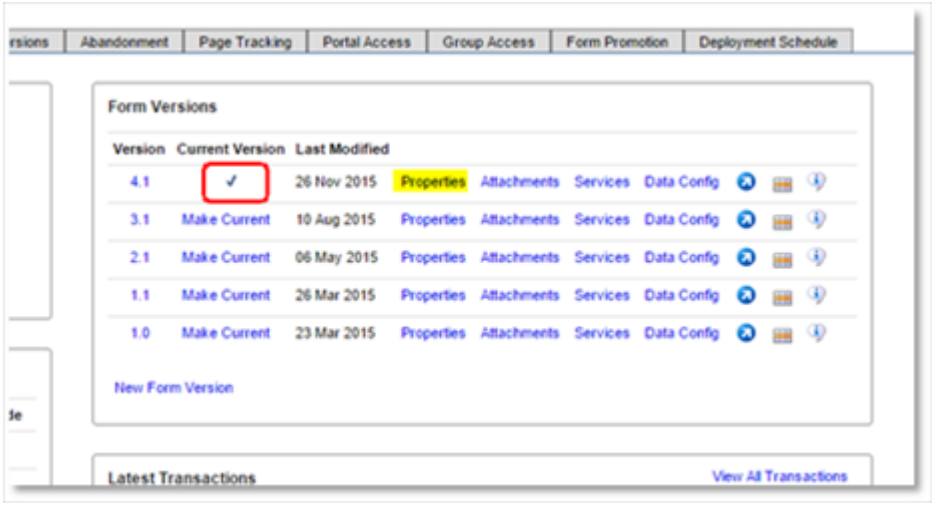
If configured, then these custom pages also require the same changes that were applied in section 2.2 - to include a hidden field with the form code that GTM can use.

Every time a form is being enabled for GTM it is therefore necessary to check whether any custom pages have been configured, and to edit these.

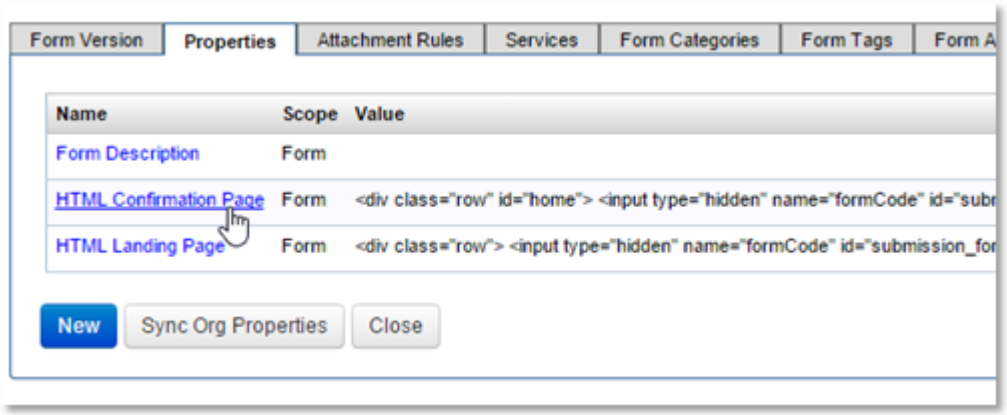
1. Navigate to **Forms -> Forms**
2. Locate the relevant form and click on the name entry



3. In the Form Versions box, locate the version that is ticked as the default, and click on the Properties link for that version



4. In the list of properties, if you see an entry for any of the relevant pages, they will need to be edited. The list of possible entries that will require editing are:
   - HTML Confirmation Page
   - HTML Saved Page
5. If present, click on the name of the relevant property



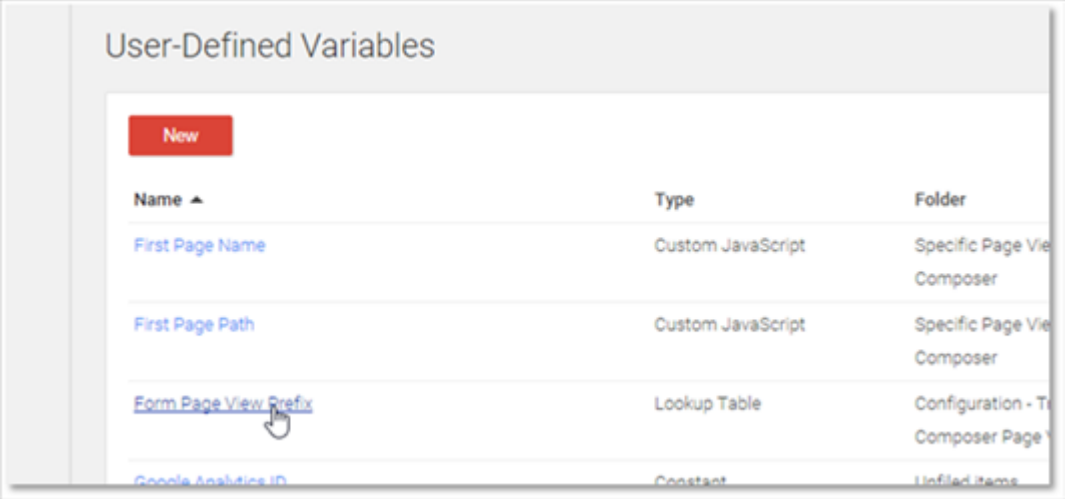
6. Place the following html inside the page content. The location is not critical, but ensure it is not inside any conditional logic (with markup such as **`#if ($session.emailed)`**:

```
<input type="hidden" name="formCode" id="submission_formCode" value-
e="$form.formCode"/>
```

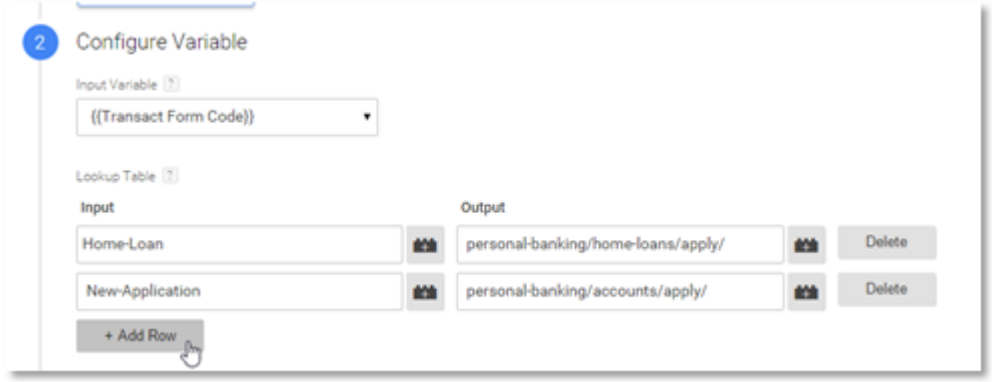
7. Save your changes (the button at the bottom of the page)

Configure Google Tag Manager

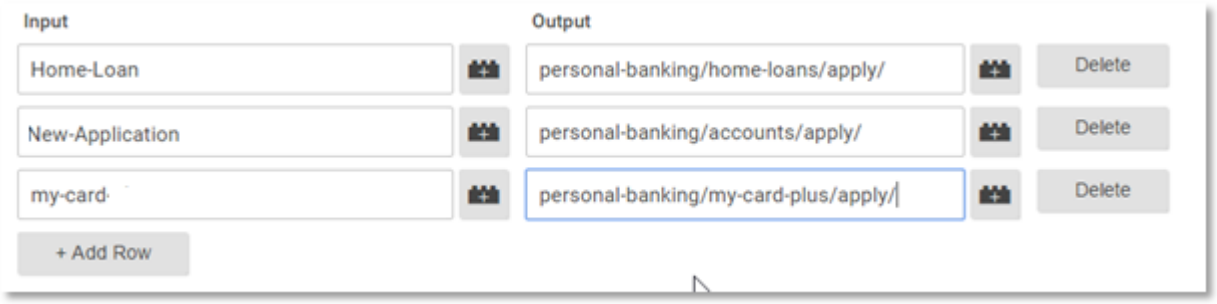
Navigate to **Container -> Variables -> Form Page View Prefix.**



Click inside the Configure Variable section to allow editing, and add a row to the Lookup Table.

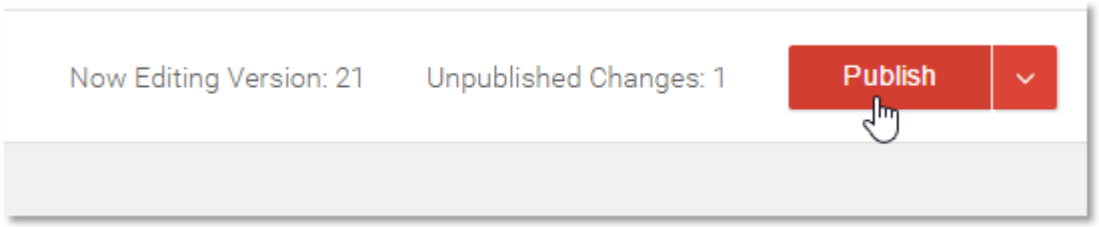


In the Input field, type the Manager Form Code and in the Output field enter the Page View prefix that will be used for this form. This prefix will be used for all Page Views generated by this form, and is a unique path that will identify the form in the Analytics engine

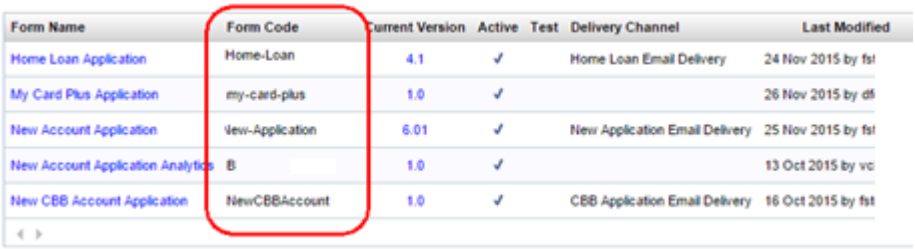


Save the Variable

Once all changes have been made, it is important to publish the changes using the Publish button in the top right corner of the GTM page. Once published, traffic on that form will generate the new page views which will be sent to Google Analytics



To find the Manager Form Code, open Avoka Transact Manager, and navigate to Forms -> Forms. The resulting list will show the form codes for all published forms in that environment:



Form Name	Form Code	Current Version	Active	Test	Delivery Channel	Last Modified
Home Loan Application	Home-Loan	4.1	✓		Home Loan Email Delivery	24 Nov 2015 by fs
My Card Plus Application	my-card-plus	1.0	✓			26 Nov 2015 by df
New Account Application	iew-Application	6.01	✓		New Application Email Delivery	25 Nov 2015 by fs
New Account Application Analytics	B	1.0	✓			13 Oct 2015 by vc
New CBB Account Application	NewCBBAccount	1.0	✓		CBB Application Email Delivery	16 Oct 2015 by fst

Further Information

The following sections include some useful information for GTM administrators that may be wishing to maintain the container, or make changes going forward.

Navigate and Locate Items in GTM

A folder structure has been setup inside the container to group relevant items together. By navigating to the Folders section (in the left-hand menu structure) items can be located a little more easily. The folder structure consists of the following folders:

- Configuration - Transact Composer Page Views - contains variables that relate to the changeable configuration. In particular, the Form Page View Prefix variable that an administrator needs to edit to add or change a form's analytics settings is located here.
- General In-form Navigation Page Views - Transact Composer - contains Variables, Triggers and Tags that are involved in generating the page views for in-form navigation - such as registering to receive custom events when a user changes sections, as well as the variables that resolve the section name from the custom event, to be used in the page view Tag.
- Specific Page Views - Transact Composer - contains Variables, Triggers and Tags for the page views that are to be generated for all pages outside of the actual Composer form - such as confirmation pages, attachments pages and landing pages.

General Functional Overview

In-form Page Views

To enable page views to be generated as a user navigates through the Composer form, a few key GTM elements have been configured.

- Register Page Change Form Events (Tag)
  - Custom HTML tag type
  - Injects some javascript that utilises the Composer libraries to detect when a user has changed pages, and sends a custom event to GTM - the custom event contains information relating to which section has been opened.
  - Fires on Document Ready, but only in the SmartForm.html page, and if the form being viewed has been configured with a page view prefix (see the Document Ready with Section Level Analytics Trigger)
- Section Level Page View (Tag)
  - Google Analytics (Universal) tag type
  - Actually creates the Google Analytics page view with the relevant prefix and section information
  - Fires on the custom page.change event that is generated by the user action

The other elements in this In-form Navigation folder in GTM are triggers and variables that support these two tags - mainly for extracting details from the generated custom event, as well as ensuring the page views only fire in relevant Composer form pages.

Other Page Views

For all other page views, a fairly simple paradigm has been followed of a single Trigger and single Tag. It should be noted though that these items also rely on other existing Variables, but going forward it is unlikely that these supporting variables would need to change.

As an example, the Submission Complete page views are configured as follows (and the other pages follow the same

- Submission Complete Page Opened (Trigger)
  - Page View Trigger
  - Fires on DOM Ready
  - Checks that the Page URL contains confirmation.htm and that a form prefix has been configured for this form - which automatically ensures the page is a TM form confirmation page, and that the form has been set to use Google Analytics
- Submission Complete Page View (Tag)
  - Google Analytics (Universal) tag type
  - Uses the configured form prefix with a hard-coded page name - i.e. {{Form Page View Pre-fix}}submission
  - Fires on the Submission Complete Page Opened trigger

Next, learn about the form dashboard.

Cloud hosted Journey Platform VDC Editions

Journey Platform | All Personas | 5.1 & Higher

When it comes to running our Temenos Journey platform, you have several options with the most popular being the Private Cloud Enterprise Edition or **Virtual Data Centre (VDC)** [1], which is suitable for organizations with the highest requirements for privacy and security. It is cloud based for maximum agility, and includes multiple servers to ensure high availability. It uses a private instance of the Amazon RDS database server that is not shared with any other organizations. It also offers hybrid cloud options for maximum flexibility in storage of **Personal Identifiable Information**[2], and **SalesForce**[3] integration.

You can choose between the VDC and VDC+ Edition, depending on your requirements. To help you decide which edition works best for you, we have listed specific features of each VDC edition below.

Features	VDC Edition	VDC+ Edition
Redundancy & Availability		
High Availability[4] Application Servers (Multi-AZ) (Traffic load balanced across application servers in separate data centers)		
Transact Application Server Nodes (Production) 2+2 = 2x /manager (admin) portal nodes + 2x /workspace portal nodes HA = High Availability (nodes deployed into separate AWS AZs)	4+ (HA) (2+2)	4+ (HA) (2+2)
High Availability RDS Database Server Replicated across separate data centers (Multi AZ)		
Non-production environments (test, UAT, etc) included	3	3
Production - Service Level Availability Target (monthly)	99.9%	99.9%
Number of countries supported (ie tenants) (You can't use TM ORGs to segregate country solutions, or deploy additional instances)	1	1
Extra countries/tenants available (POA - price on application) (In standard/enterprise edition, you need to buy a new edition for an extra country)		
RDBMS	Aurora	Aurora
Backup & Recovery		

[1]Virtual Data Centre (VDC) is an automated and scalable Infrastructure as a Service (IaaS) cloud platform providing compute, network and storage.

[2]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

[3]Salesforce is the customer relationship management (CRM) platform.

[4]High availability (HA) is a characteristic of a system which aims to ensure an agreed level of operational performance, usually uptime, for a higher than normal period.

Database Backup Frequency	Continuous	Continuous
Backup Frequency (**RPO**[1])	~ 5 minutes	~ 5 minutes
Database Recovery Time (**RTO**[2])	4 hours	4 hours
Security		
Customer Virtual Private Cloud (Solution installed in customer-specific AWS VPC with private subnets) Data Sovereignty - All data resides in same geographic region. Regions are US (Oregon) EU (Ireland) Asia Pacific (Sydney) Other AWS Regions available as required Strong AES 256-bit data encryption for data in transit and at rest Separate security subnets for production and non-production environments IP White-listing Automatic updates for o/s & security vulnerability patches Anti-virus scanning Dedicated RDS Database Server in same Virtual Private Cloud Separate administration and end user portal servers		
Encryption Data Key Management	Amazon KMS	Amazon KMS
Internal vulnerability scanning – Continuous (Qualys Cloud Agent) External vulnerability scanning - Monthly (Qualys WAS scan) Third-party penetration testing - Annually	---	
Akamai - for WAF and CDN	Available (POA)	Available (POA)
Network Connectivity Features		
AWS VPC Peering (Connect 2 AWS accounts; no service cost - AWS Doc)	Available	Available
AWS Direct Connect (Connect AWS to on premise; service cost not included - AWS Doc)	Available	Available
AWS VPN Site to Site	Available (POA)	Available (POA)
Advanced Security Features		
Web Application Firewall (WAF)	Apache ModSe-	Sophos

[1]Recovery point objective (RPO) is the age of files that must be recovered from backup storage for normal operations to resume if a computer, system, or network goes down as a result of a hardware, program, or communications failure.

[2]Recovery Time Objective (RTO) is the targeted duration of time and a service level within which a business process must be restored after a disaster (or disruption) in order to avoid unacceptable consequences associated with a break in business continuity.

	curity (fixed customized rules)	UTM
Network Protection (Deep packet inspection, Flood protection, DoS, DDoS and port scan blocking, Country blocking)	AWS Shield Standard (no DPI and Country blocking)	Sophos UTM
Advanced Threat Protection	---	Sophos UTM SandStorm
Outbound Proxy (whitelist)	Squid	Sophos UTM
Security Extras		
AWS Dedicated Instances (dedicated hardware for application servers - AWS Doc)	Available (POA)	
Additional IPS/ UTM (Sophos Unified Threat Management) layer (AWS Doc)	Available (POA)	
Platform Issue - Response Times – Production		
Level 1	30 mins	30 mins
Level 2	1 hour	1 hour
Level 3	2 hours	2 hours
Level 4	1 business day	1 business day
Platform Issue - Resolution Times – Production		
Level 1	2 hours	2 hours
Level 2	4 hours	4 hours

If you have any further questions or you need more information including pricing, contact your Customer Success Manager.

Next, learn about the Journey platform system requirements.

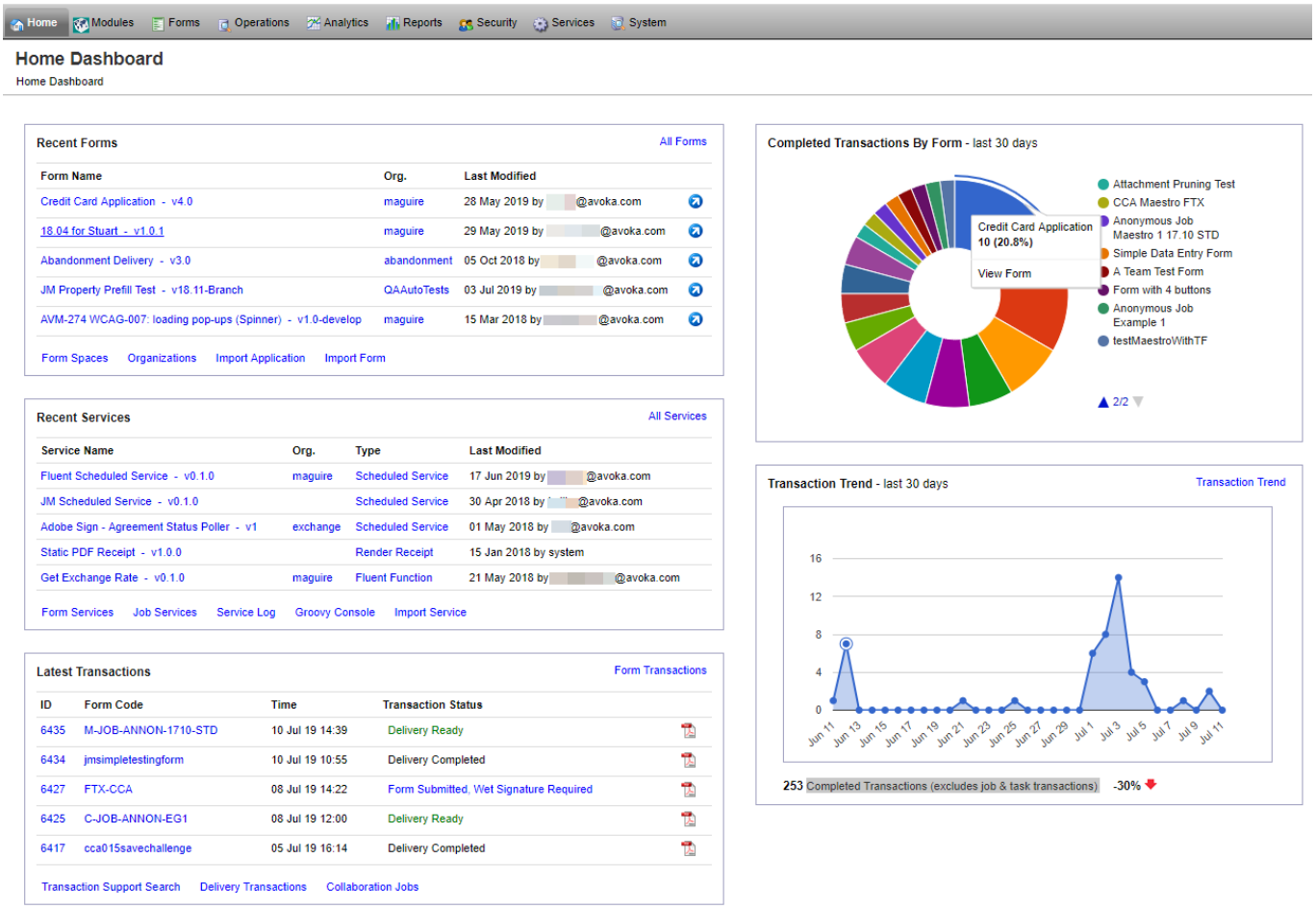
Journey Manager Home Dashboard

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager comes with the Home Dashboard where you can quickly access your recent forms, services, and transactions, as well as check the number of completed transactions and the transaction trend.

> The Manager Home Dashboard is often referred to as the Manager console.

To navigate to the Manager Home Dashboard, click **Home**.



The Dashboard consists of the following sections:

- Recent Forms
- Recent Services
- Latest Transactions
- Completed Transactions By Form
- Transaction Trend

Recent Forms

This section lists five recently viewed and modified forms ordered by last modified date and time.

You can perform the following actions from this section of the Dashboard:

- Click a form name to see its configuration in a form dashboard.
- Click **Org** to view this organization configuration.
- Click **From Test Render**  to render this form in the default form space.
- Click **Form Spaces** to view form spaces.
- Click **Organizations** to view organizations.
- Click **Import Application** to import an application package.
- Click **Import Form** to import a form.

Recent Services

This section lists five recently viewed and modified services ordered by last modified date and time.

You can perform the following actions from this section of the Dashboard:

- Click a service name to see its service definition configuration.
- Click **Org** to view this organization configuration.
- Click **Service Type** to view services of this selected type.
- Click **Form Services** to view form services.
- Click **Job Services** to view job services.
- Click **Service Log** to view Groovy service log.
- Click **Groovy Console** to open the Groovy console.
- Click **Import Service** to import a service.

Latest Transactions

This section lists five latest transactions of users' submissions.

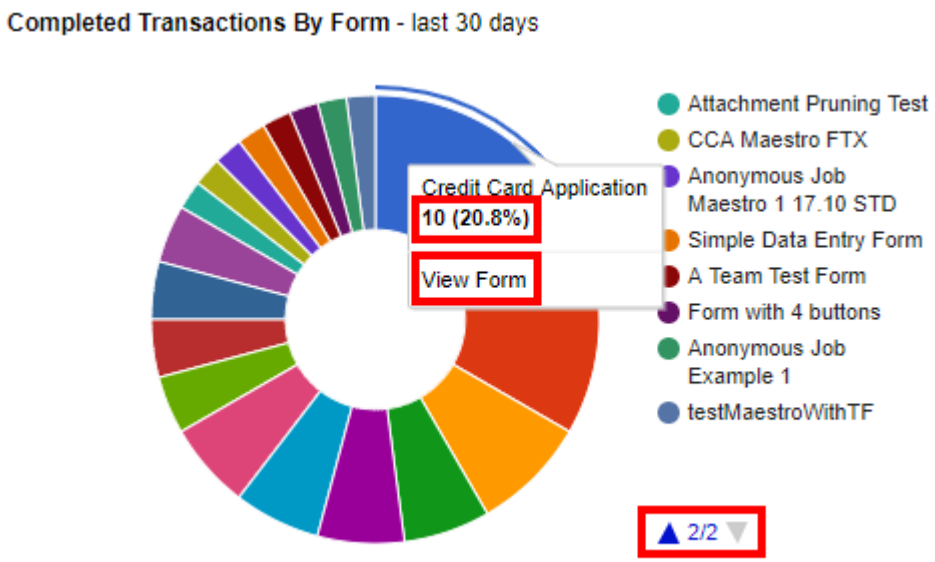
You can perform the following actions from this section of the Dashboard:

- Click **ID** or **Form Code** to see the transaction details.
- Click **Transaction Status** to view a transaction status.
- Click **View PDF Receipt** to view and download a PDF receipt of a submitted form.
- Click **Transaction Support Search** to view and search transactions.
- Click **Delivery Transactions** to view transaction delivery.
- Click **Collaboration Jobs** to view collaboration jobs.
- Click **Form Transactions** to view form transactions.

Completed Transactions By Form

This pie chart visualizes the number and the percentage distribution of completed transactions by each form for the last 30 days.

Click a sector of the pie chart to see which form it represents and the following form statistics:



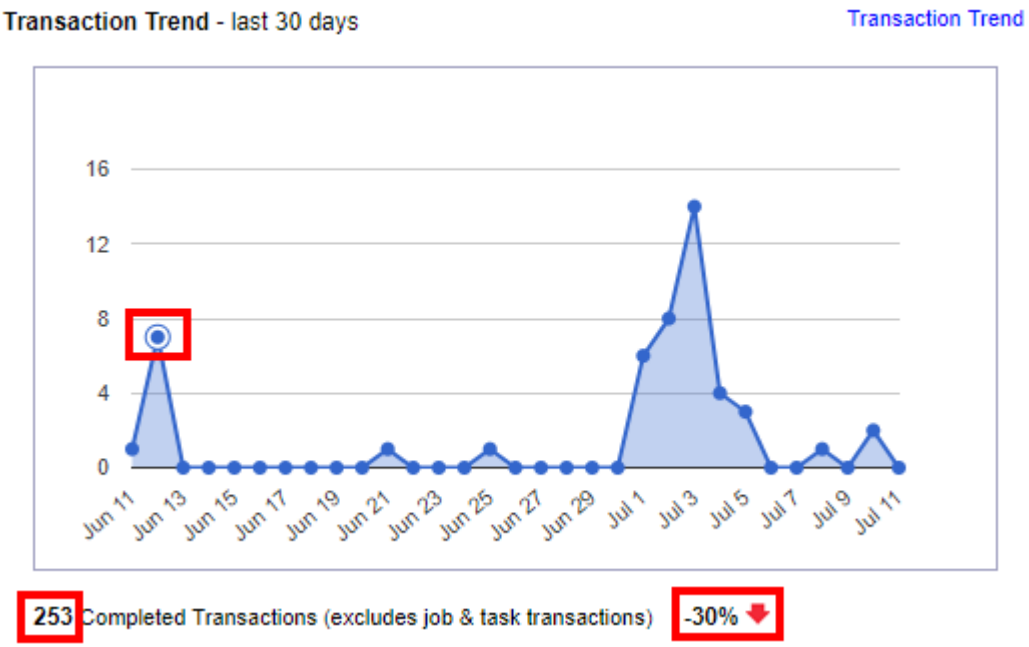
- A form name
- A number of transactions for this form
- A percentage of this form transactions as compared to the total number of transactions for the last 30 days
- Click **View Form** to open a form dashboard. to check the form configuration

Click up and down arrows to scroll the list of the forms visualized on this pie chart

Transaction Trend

This graph visualizes the transaction trend - the total number of completed transactions per day - for the last 30 days.

You can perform the following actions from this section of the Dashboard:



- Click on each node to see the number of completed transactions for that day
- Click **Transaction Trend** to analyze transaction trend
- Check the total number of completed transactions excluding collaboration job and task transactions
- Check the trend, for example 30% less transactions as compared to the previous period

Next, learn about the form dashboard.

Journey Manager Modules

Journey Manager (JM) | System Manager / DevOps | 19.05

Journey Manager supports modular architecture, so you can deploy new software modules to add or enhance the existing functionality. A module is a WAR file that encapsulates this functionality. Manager includes the following modules:

- Avoka Exchange
- Transact Insights
- Transact Maestro
- Transact Manager
- Transact Receipts

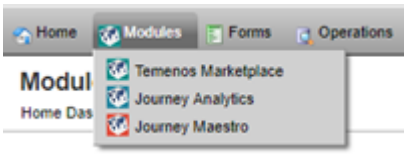
Some modules can be considered as products themselves, because of their rich functionality and the way they can be used, so you can refer to modules as products too, for example, the Maestro product.

> Our product names have changed, but the modules will maintain their original names for backward compatibility. | 19.05

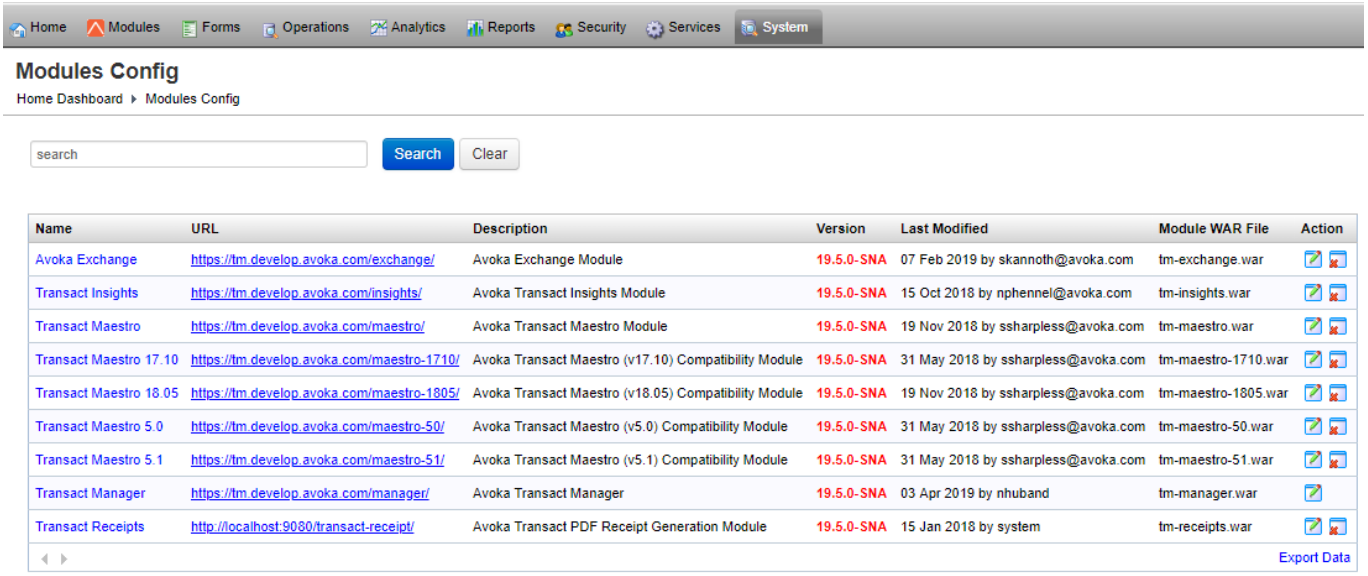
You can decide which modules to install during the Journey Manager installation or upgrade.

> You should only install the modules you need because each module incurs licensing fees and consumes both memory and processing power of a server.

You can access the most popular modules, which are also called products, from the **Modules** dropdown list, as shown below.



To view all installed modules, select **System > Modules Config.**



Next, learn how to configure modules.

Journey Manager 3rd Party Libraries

Journey Manager (JM) | System Manager / DevOps | 22.04

Journey Manager is implemented using a broad range of open source 3rd party libraries. We have carefully selected each library to make sure it's right for our purpose. We diligently update each library with new versions to ensure Manager runs as expected. If we get to know that a library contains a bug or a security vulnerability (CVE), we will update it as soon as possible, so each major release of Manager will include only updated libraries.

Access to 3rd party **Java**[1] libraries by Fluent Groovy services is restricted via a security white list. For more information, see Fluent Security Configuration.

Licenses

Manager uses the 3rd party libraries according to their term and conditions outlined in the followings licenses:

- Apache License 2.0
- MIT License
- iText Commercial License
- NetBSD (2-line) License
- BSD 3-Clause License
- Microsoft JDBC Driver 4.0 For SQL Server License
- Oracle's dual-license JDBC Driver for MySQL
- PayPal, Inc. SDK License
- OTN License Agreement
- Gemalto Commercial License
- Mozilla Public License
- BSD 3-clause

Libraries

To view the list of 3rd party libraries in Manager prior to version 19.05, select **System > About** and click **3rd Party Libraries**. | Removed in 19.05.

The 3rd party library names and their versions are listed below for each major Manager version:

---

[1] Java is a programming language that produces software for multiple platforms. When a programmer writes a Java application, the compiled code (known as bytecode) runs on most operating systems (OS), including Windows, Linux and Mac OS. Java derives much of its syntax from the C and C++ programming languages.

Library Name / Java Package	19.05	19.11	20.05	20.11	21.05	21.11	22.04
Apache Axis org.apache.axis	1.4						
Apache Camel org.apache.camel.core							
Apache Cayenne org.apache.cayenne	4.0.1	4.0.1	4.0.1	4.0.1	4.0.1	4.1	4.1.1
Apache Click org.apache.click	2.4-RC3-TM	2.4-RC5-TM	2.4-RC5-TM	2.4-RC8-TM	2.4-RC8-TM	2.4-RC8-TM	2.4-RC1-7-TM
Apache Commons Command Line Interface org.apache.commons.cli	1.2	1.2	1.3.1	1.3.1	1.3.1	1.3.1	1.3.1
Apache Commons Bean Utils org.apache.commons.beanutils	1.9.3	1.9.4	1.9.4	1.9.4	1.9.4	1.9.4	1.9.4
Apache Commons Codec org.apache.commons.codec	1.10	1.13	1.13	1.14	1.14	1.14	1.14
Apache Commons Collections	3.2.2	3.2.2	3.2.2	3.2.2	3.2.2	3.2.2	3.2.2

org.apache.commons.collections							
Apache Commons Collections4							
org.apache.commons.collection4	4.4	4.4	4.4	4.4	4.4	4.4	4.4
Apache Commons Compress							
org.apache.commons.compress		1.19	1.19	1.20	1.20	1.21	1.21
Apache Commons Discovery							
org.apache.commons.discovery	0.2	0.2	0.2	0.5	0.5	0.5	0.5
Apache Commons Email							
org.apache.commons.email	1.5	1.5	1.5	1.5	1.5	1.5	1.5
Apache Commons FileUpload							
org.apache.commons.fileupload	1.3.3	1.4	1.4	1.4	1.4	1.4	1.4

Apache Commons HttpClient org.apache.commons.httpclient	3.1						
Apache Commons IO org.apache.commons.io	2.5	2.5	2.5	2.5	2.5	2.10	2.10.0
JSON for Java com.fasterxml.jackson	2.9.9	2.10.0	2.10.1	2.11.3	2.11.3	2.11.3	2.12.1
Apache Commons Lang org.apache.commons.lang	2.6	2.6	2.6	2.6	2.6	2.6	2.6
Apache Commons Lang org.apache.commons.lang3	3.8	3.9	3.9	3.10	3.10	3.10	3.11
Apache Commons Math org.apache.commons.math	1.2	1.2	1.2	1.2	1.2	1.2	1.2
Apache Commons Net org.apache.commons.net	3.3	3.3	3.6	3.7	3.7	3.7	3.8

Apache Commons Pool org.apache.commons.pool	1.6	1.6	1.6	1.6	1.6	1.6	1.6
Apache CXF org.apache.cxf	3.2.7	3.3.3	3.3.5	3.3.7	3.3.7	3.3.7	3.3.1-0
Apache FOP org.apache.fop	2.3	2.3	2.4	2.5	2.5	2.5	2.6
Apache HTTP Components org.apache.httpcomponents	4.5.4	4.5.4	4.5.11	4.5.13	4.5.13	4.5.1-3	4.5.1-3
Apache James Mime4J org.apache.james.mime4j	0.6	0.6	0.6	0.6	0.6	0.6	0.6
Apache Neethi org.apache.neethi	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1
Apache POI org.apache.poi	3.17	4.1.1	4.1.2	4.1.2	4.1.2	4.1.2	5.0
Apache Santuario org.apache.santuario.xmlsec	2.1.2	2.1.2	2.1.2	2.1.4	2.1.4	2.1.4	2.1.7
Apache Shiro org.apache.shiro	1.3.2	1.3.2	1.5.2	1.7.0	1.7.0	1.8.0	1.8.0

Apache Velocity org.apache.velocity	1.7	1.7	1.7	1.7	1.7	1.7	2.3
Apache WS Security org.apache.ws.security	2.2.2	2.2.2	2.2.5	2.2.5	2.2.5	2.2.5	2.2.5
Apache WS XML Schema org.apache.ws.xmlschema	2.2.1	2.2.1	2.2.1	2.2.5	2.2.5	2.2.5	2.2.5
Apache Xalan org.apache.xalan	2.7.1	2.7.1	2.7.1	2.7.1	2.7.1	2.7.1	2.7.1
Apache Xerces org.apache.xerces	2.12.0.SP02	2.12.0.SP02	2.12.0.SP02	2.12.0.SP03	2.12.0.SP03	2.12.0.SP03	2.12.2
Apache XML Resolver org.apache.xml.xmlresolver	1.2	1.2	1.2	1.2	1.2	1.2	1.2
Auth0 Java JWT com.auth0.java-jwt	2.2.2	2.2.2	2.2.2	2.2.2	2.2.2	2.2.2	2.2.2
AWS SDK for Java com.amazonaws	1.11.193	1.11.193	1.11.754	1.11.865	1.11.865	1.11.865	1.11.983
Bouncy Castle org.bouncycastle	1.61	1.64	1.64	1.66	1.66	1.69	1.69
ClamAVJ	0.1	0.1	0.1	0.1	0.1	0.1	0.1

com.-philvarner.clamavj							
FuseSource Com-mon Man-agement org.-fuse-source.-commonman	1.0	1.0	1.0				
Google Gson com.google.gson	2.8.2	2.8.2	2.8.2	2.8.6	2.8.6	2.8.6	2.8.6
Google Guava com.google.guava	25.0-jre	25.0-jre	25.0-jre	25.0-jre	25.0-jre	25.0-jre	25.0-jre
Groovy Lang org.-codehaus.groovy	2.4.1-5	2.5.6	2.5.9	2.5.14	3.0.8	3.0.8	3.0.8
Groovy WS-Lite org.-codehaus.groovy	0.8.0	0.8.0	0.8.0	0.8.0	0.8.0	0.8.0	0.8.0
Groovy HTTP Builder org.-codehaus.groovy	0.7.1	0.7.1	0.7.1	0.7.1	0.7.1	0.7.1	0.7.1
Hessian Remot-ing com.-caucho.hessian	4.0.3-8	4.0.3-8	4.0.63	4.0.63	4.0.63	4.0.6-3	4.0.6-5
iText PDF com.i-textpdf.text.pdf	5.5.1-3	5.5.1-3	5.5.13.1	5.5.13.2			

Java Sysmon com.jezhumble.-javasysmon	0.3.3	0.3.3	0.3.3	0.3.3	0.3.3	0.3.3	0.3.3
Jaxen org.jaxen	1.1.6	1.1.6	1.1.6	1.1.6	1.1.6	1.1.6	1.2.0
Jayway JsonPath com.-jayway.jsonpath	2.0.0	2.0.0	2.0.0	2.4.0	2.4.0	2.4.0	2.5.0
JBoss RESTEasy JAX-RS org.jboss.resteasy.-resteasy-jaxrs	3.6.3.-Final	3.9.0.-Final	3.11.0.F-inal	3.13.2.F-inal	3.13.2.F-inal	3.15.-1.Fin-al	4.7.4-.Final
Joda Time org.joda.time	2.9.7	2.9.7	2.9.7	2.9.7	2.9.7	2.9.7	2.9.7
Jsch com.jcraft.jsch	0.1.5-4	0.1.5-4	0.1.54	0.1.55	0.1.55	0.1.5-5	0.1.5-5
Json-Lib net.-sourceforge.json	2.4	2.4	2.4	2.4	2.4	2.4	2.5.0
JSoup org.jsoup	1.8.3	1.8.3	1.8.3	1.8.3	1.8.3	1.14.-2	1.14.-2
JUnit org.junit	4.12	4.12	4.13	4.13.1	4.13.1	4.13.-1	4.13.-2
Microsoft EWS Java API com.microsoft.ews-java-api	2.0	2.0	2.0	2.0	2.0	2.0	2.0

Microsoft JDBC Driver com.-microsoft.sqlserver	6.4.0.-jre8	6.4.0.-jre8	6.4.0.jr-e8	8.4.1.jr-e8	8.4.1.jr-e8	8.4.1.-jre8	9.2.1-.jre8
Mockito org.mockito	2.26.-0	2.26.-0	3.3.3	3.5.13	3.5.13	3.5.1-3	3.8.0
MVEL org.mvel	2.4.3	2.4.3	2.4.7.Fi-nal	2.4.10.F-inal	2.4.10.F-inal	2.4.1-0.Fin-al	2.4.1-2.Fin-al
MySQL JDBC Driver com.mysql	8.0.1-6	8.0.1-7	8.0.19	8.0.21	8.0.21	8.0.2-1	8.0.2-3
PayPal Java SDK com.paypal.sdk							
QRGen net.glxn.qrgen	1.4	1.4	1.4	1.4	1.4	1.4	1.4
Oracle JDBC Driver com.oracle	10.2.-0.4.0	10.2.-0.4.0	10.2.0.4-.0	10.2.0.4-.0			
Open SAML org.opensaml	3.3.0	3.3.0	3.3.0	3.3.1	3.3.1	3.3.1	3.4.6
Quartz org.quartz	1.8.6	2.3.2	2.3.2	2.3.2	2.3.2	2.3.2	2.3.2
Recaptcha net.tane-sha.recaptcha	0.0.8	0.0.8	0.0.8	0.0.8	0.0.8	0.0.8	0.0.8
SafeNet Luna	5.3.0-11	5.3.0-11	5.3.0-11	5.3.0-11	5.3.0-11	5.3.0-11	5.3.0-11

com.safenet.luna							
SAXON net.-sourceforge.saxon	9.4.0.-6	9.4.0.-6	9.4.0.6	9.4.0.6	9.4.0.6	9.4.0.-6	9.4.0-.6
SLF4J org.slf4j	1.7.2-2	1.7.2-2	1.7.22	1.7.30	1.7.30	1.7.3-0	1.7.3-2
Spring Frame-work org.s-pring-framework.spring	4.3.2-4	4.3.2-5	5.2.4.RE-LEASE	5.3.0	5.3.0	5.3.0	5.3.1-8
Spring Security org.s-pring-frame-work.security	4.2.1-2	4.2.1-3	4.2.13.R-ELEASE	4.2.18.R-ELEASE	4.2.18.R-ELEASE	5.5.1	5.6.1
Super CSV org.supercsv	2.4.0	2.4.0	2.4.0	2.4.0	2.4.0	2.4.0	2.4.0
User-Agent-Utils eu.bit-walk-er.useragentutils	1.21	1.21	1.21	1.21	1.21	1.21	1.21
WildFly org.wildfly	16.0.-0.Fin-al	18.0.-0.Fin-al	19.0.0.F-inal	21.0.1.F-inal	23.0.1.F-inal	24.0.-1.Fin-al	26.0.-1.Fin-al

-- A library is still supported by Manager but it is not distributed with the installer. You may need to deploy it manually. For more information, see the installer guide of your Manager version.

If you have issues with the Java classloader resolving some of these libraries, you need to contact Temenos support for assistance.

Journey SDK uses the same 3rd party libraries as Manager.

If you want to run a library vulnerability check independently, you can use a library name and its version from the table above.

If you use one of End-of-Life Manager versions, you can check their 3rd party libraries here.

Next, learn about our end-of-life policy.

Installation Overview

Journey Manager (JM) | System Administrator | 21.11

Before you can start using Journey Manager, you must correctly install and configure it. This is easy to do by following our installation guide, which is designed and written for product evaluators, administrators, DevOps, as well as developers. For CentOS Linux, Oracle Linux, Red Hat Enterprise Linux, and Microsoft Windows installations, you should be familiar with your target operating system and its administration.

More often than not, Manager is installed on cloud so you don't need to do it yourself. However, we recommend getting familiar with this documentation as it helps you understand some product features and performance considerations as well as how to upgrade your product to newer versions.

Each major release of Manager comes with its own installation guide describing specific steps you must perform before, during and after installation or upgrade. The installation guides are PDF documents named **`jm_installation_guide_v<MANAGER-MAJOR-VERSION>.pdf`**, for example, **`jm_installation_guide_v21.11.pdf`**.

Download the JM Installation Guide PDF

We've replaced the Manager Installation online documentation with a PDF version. Here's how you can get a copy of the PDF.

- You can download the installation guide from the same location where the initial release of a major version of Manager downloads are hosted. For example, the installation guide for Manager 21.11 can be found in the **`21.11.0`** directory. We do not update the installation guide for minor and maintenance releases of Manager.

  Registered members of our resources website can request access to download Journey Platform products and documentation for supported Journey Platform releases.

- If you have access to the Temenos instance of Microsoft Teams, you can download the JM Installation Guide PDF from Teams. Follow these steps to download the version you need:
  1. Login to Microsoft Teams.
  2. Select the **Temenos Journey Manager (TJM)** team.
  3. Select the **Temenos Document Share** channel.
  4. Select **Files** and browse to the **`JM Installation (INTERNAL USE ONLY)`** folder.
  5. Download the JM Installation Guide PDF (**`jm_installation_guide_v21.11.pdf`**).
- Otherwise, contact us to request a copy of the JM Installation Guide PDF.

Plan the Installation

You need to plan the installation or upgrade of Manager carefully before you run installer in your environment or ask the cloud team to install it for you. There's no cover-it-all plan for every environment or configuration, but we have combined the best practices into a simple check list below. We strongly recommend reviewing it and commencing the installation only if all items of the list are satisfied.

The installation plan check list:

1. Ensure you have read and understood this installation guide. If something is unclear or your need more information, contact your Customer Service or Support.
2. Compile and build your current application packages and execute unit tests in a local IDE or on a build server using Journey SDK of the same Journey Manager version you are planning to install or upgrade to. Groovy compiler can reveal potential issues and thus provides you with more confidence to move on the next step of the upgrade.

   > If any compilation or build errors arise, resolve them before commencing the installation. Resolutions may include the product configuration, Groovy scrips corrections, or cloud hosting configuration. For more information, contact your Customer Service or Support.

3. Review all release notes of your current and target Manager versions, understand them, and analyze the impact on your existing solution. In some cases, you may need to review your current solution, for example, due to critical library updates.
4. Ensure the back up and rollback procedures for your organizations, form spaces, forms, services, service connections work as expected, so you can restore them from back-ups.
5. Plan ahead to upgrade all your environments to the same target Manager version.

   > We don't recommend deploying any application packages in to a production environment until your test, staging, UAT, and production environments run the same Manager version.

Next, learn about Journey Manager features.

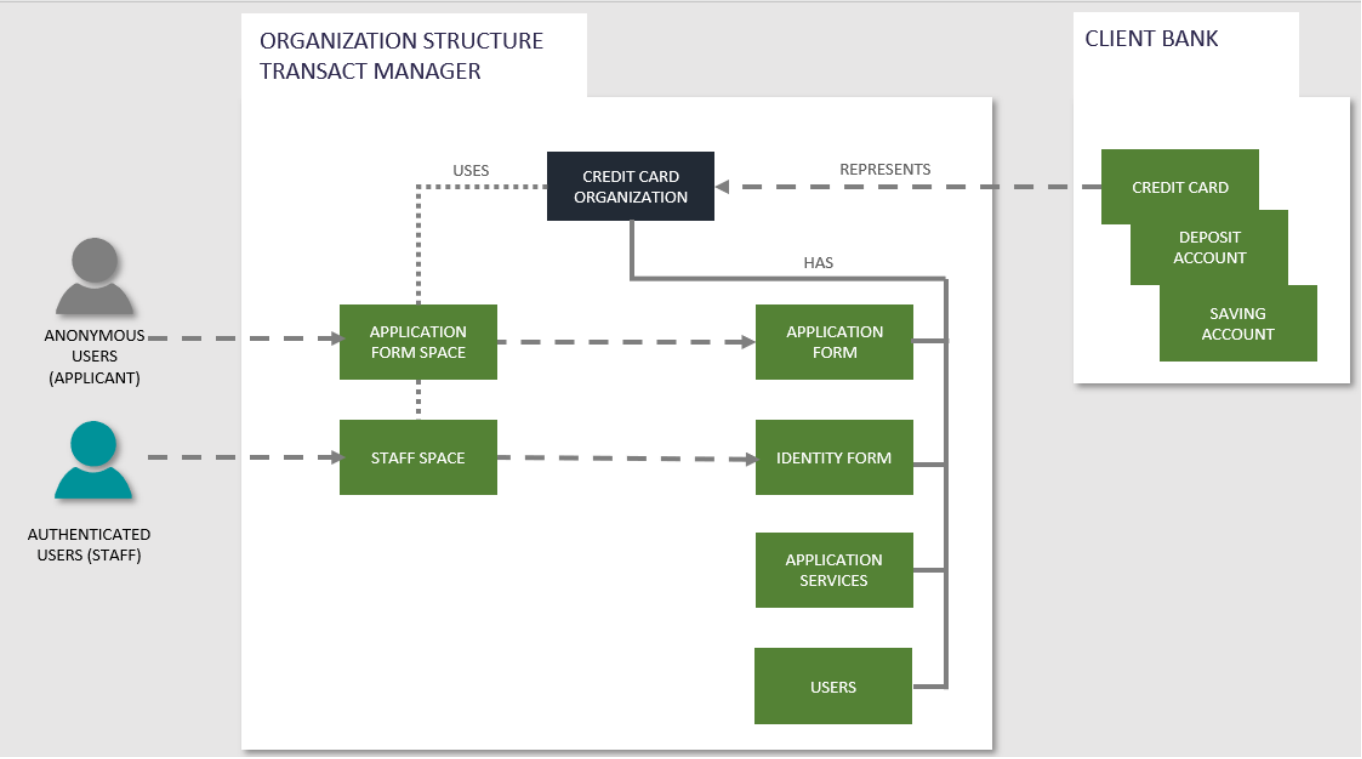
Organizations Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager uses organizations to allows you to represent your business that can be

- A company
- A department or a branch within the company
- A government organization
- A product

An organization is a logical entity within Manager that owns forms and their assets, so they can be grouped and managed as a whole. An organization can have many forms, but a form always belongs to one organization only. Each organization can use one or several form spaces to host its forms. In the diagram below, Manager represents a client bank structure with the established credit card application process.



Manager is designed to accommodate anything from a single organization with a single form to multiple organizations with many forms having multiple form versions.

> We recommend creating a separate organization in Manager for its corresponding real life counterpart.

Organizations allow:

- Developers within each organization to access, deploy, and manage forms for their own organization.
- Administrators to manage one or more organizations.
- Operators to perform a broad range of day-to-day operations for one or more organizations.

Organizations allows you to create and configure forms with standard:

- Branding
- Delivery channels
- Properties
- Services
- Reference data
- Data retention

Next, learn how to view organizations.

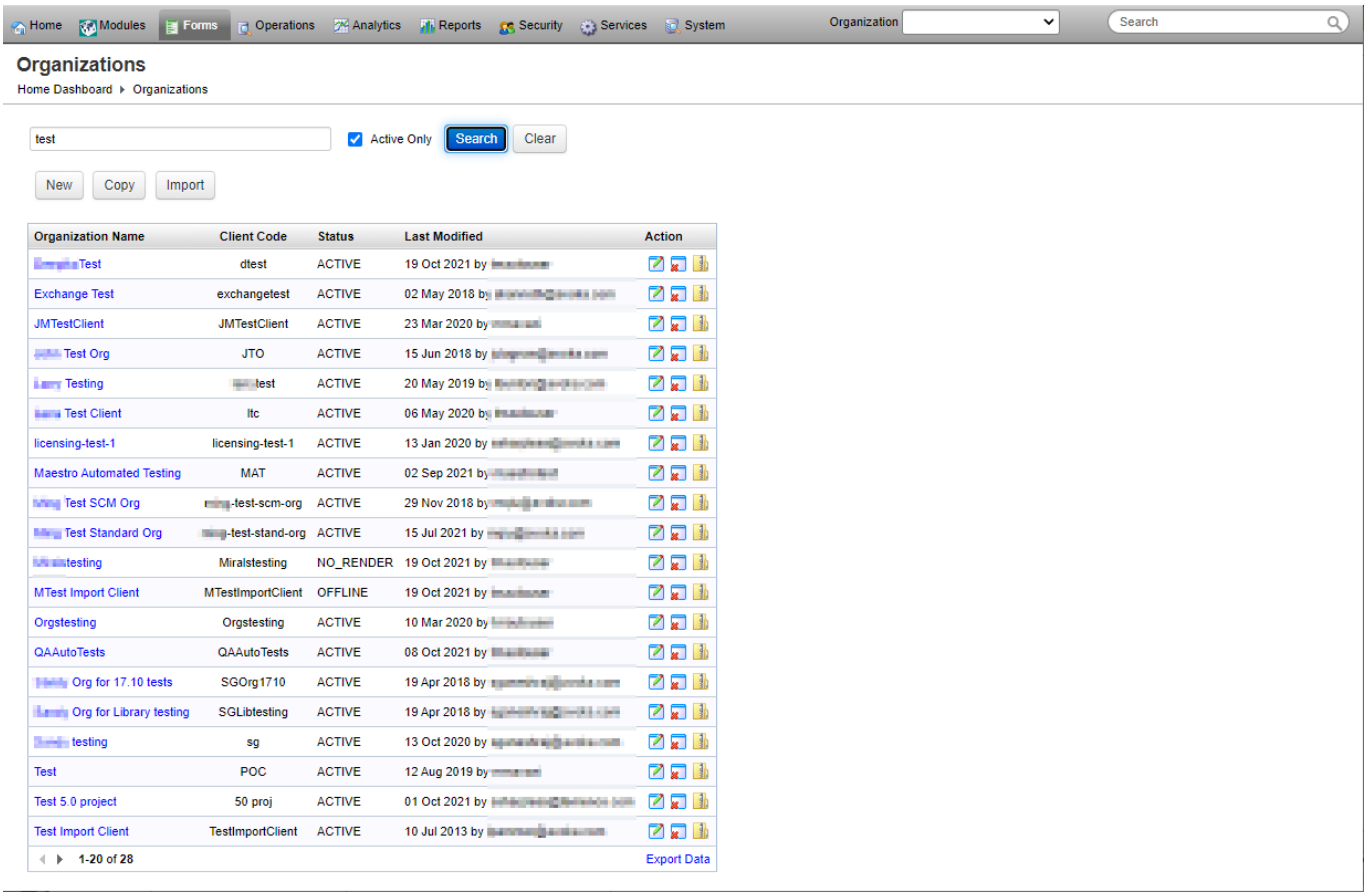
View Organizations

Journey Manager (JM) | System Manager / DevOps | 21.11

Manager allows you to view the list of available organizations, so you can manage them efficiently.

To view organizations, select **Forms > Organizations**.

Organizations are shown page by page, with the maximum number of records per page configured via the user's preferences.



To filter or search the roles by one or more criteria, specify the following settings and click **Search**:

- Organization's name or part of a name.
- **Active Only**. Only Active organizations are listed by default.
- Select an organization from the **Organization** dropdown list at the Toolbar.

Click **Clear** to reset the search criteria.

You can check the following organization details:

- Organization name is a globally unique organization name, which is often displayed on form landing pages.
- Client Code is a globally unique organization short name, which can be used in a long form URL (Form Friendly URL).
- Status determines the availability of an organization. Statuses can be one of the following: | 21.11
- Active - it is in use and readily available to all users. All functionality is allowed.
- Offline - it can't be used at all. No functionality is available, including DDS, Delivery and so on
- Inactive - similar to Offline.
- NoRender - it doesn't allow for new or saved and resumed forms, but ongoing open sessions can be completed. DDS, submits, saves, and delivery are allowed.
- Last modified is a date when the organization's configuration was updated last time.

You can also perform the following actions:

- Create an organization
- Copy an organization
- Remove an organization
- Export an organization
- Import an organization

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to create an organization.

Create an Organization

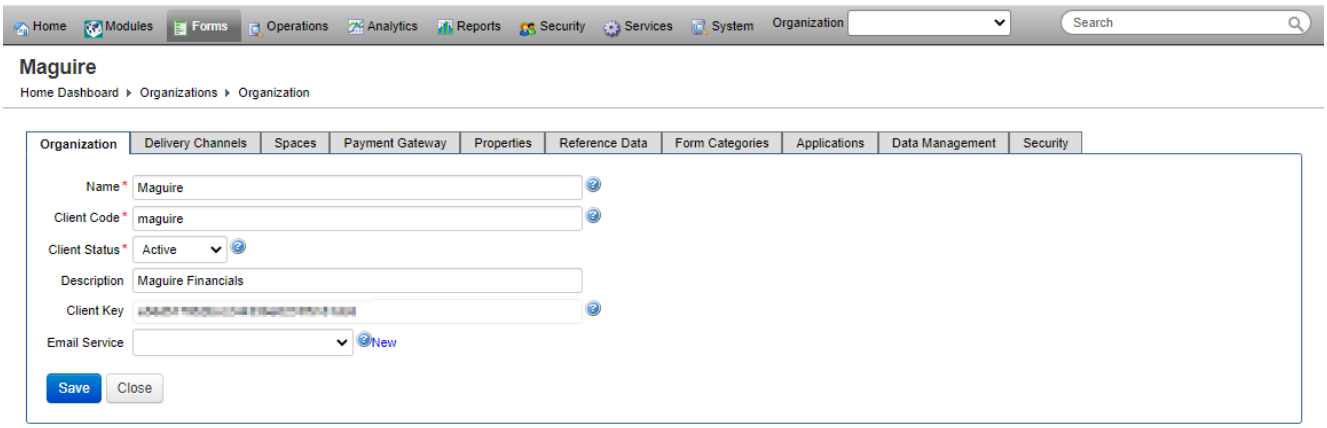
Journey Manager (JM) | System Manager / DevOps | 19.05

Manager uses organizations to encapsulate all relevant forms and resources.

> Only users with administrator privileges can create new organizations.

To create an organization:

1. Select **Forms > Organizations**.
2. Click **New**.



3. Enter a unique organization name in the **Name** field.
4. Enter a unique and abbreviated organization name in the **Client Code** field.
5. Provide a comprehensive description in the **Description** field. If a client has multiple organizations, a description may also be useful for those users who have global permissions to multiple organizations on a single Manager instance.
6. Provide an organization credential alias or client key in the **Client Key** field, which is the organization unique identifier used for subscription delivery Web Service integration.
7. Select the **Active** checkbox to allow this organization to use all facilities. Otherwise, it is inactive which means it is disabled and is not shown.

> It's not recommended to remove an organization. Instead, you should make the organization inactive.

8. Select an email service, used by this organization's forms to send submission related emails to users, from the **Email Service** dropdown list.
9. Click **Save** to update the changes.

Once the new Organization is created, it automatically takes you to the Spaces tab for space assignment. This screen will show both default form spaces, as well as any custom form spaces you have configured in your journey manager environment so far. The form spaces assignment determines what form spaces will be available to forms assigned to the Organization. The space configuration is then also configurable at the form level.

Next, learn how to configure an organization.

Copy an Organization

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

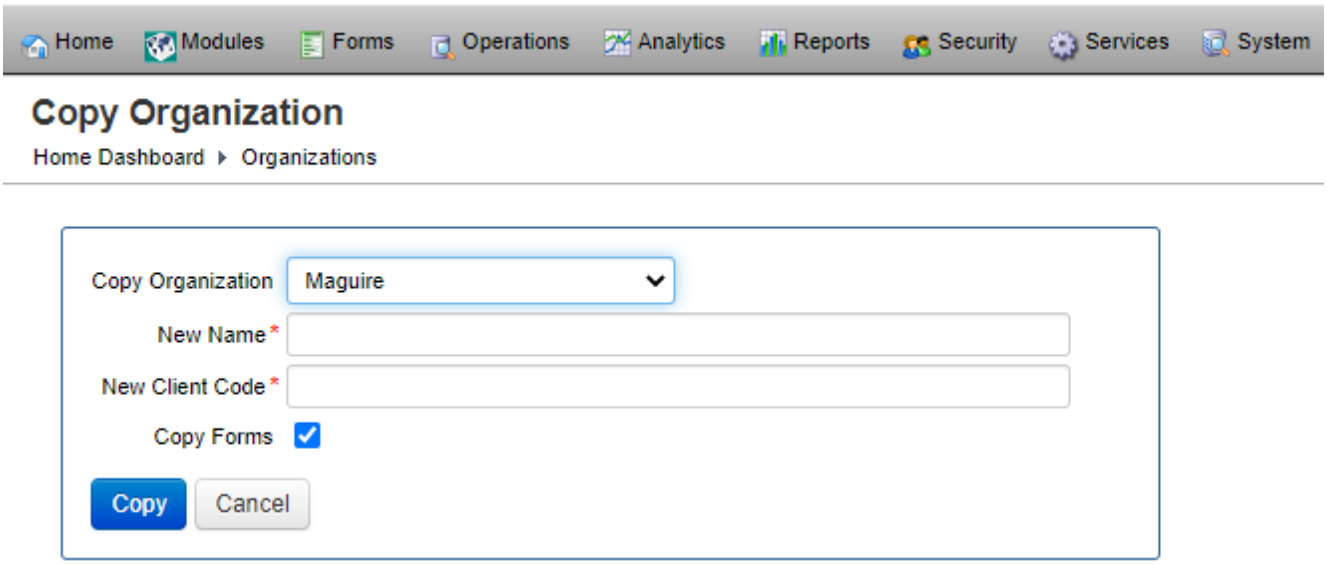
Manager allows you to copy an existing organization. You can use this functionality to:

- Test organizations
- Create multiple copies of the same organization
- Configure a new organization from an existing organization

The copied organization inherits the same configuration as the original one, which can include all of the organization's forms and resources.

To copy an existing organization:

1. Select **Forms > Organizations**.
2. Click **Copy**.



3. Select the organization you want to copy from the **Copy Organization** dropdown list.
4. Enter a unique organization name in the **New Name** field.
5. Enter a unique and abbreviated organization name in the **New Client Code** field.
6. Select the **Copy Forms** checkbox to copy all this organization's forms. If you do not want to copy all this organization's forms, clear this checkbox.
7. Click **Copy**.

Organization's payment configuration, report schedules, delivery details and form manager references are

not copied.

The newly created organization will be in **Active** state.

Next, learn how to configure an organization.

Remove an Organization

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to remove an organization and all its dependent resources, such as forms and submissions.

> It is recommended to make an organization inactive rather than remove it, especially if there have been submissions for any of its forms.

To remove an organization:

1. Select **Forms > Organizations**.
2. Click **Remove** and confirm your decision. This will permanently remove the organization and all its configurations.

Next, learn how to export an organization.

Export an Organization

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to export organizations. You can use this functionality to:

- Backup or archive organizations.
- Re-create one or several organizations in another Manager.
- Promote or migrate one or several organizations to other environments.

To export one or several organizations, select **Forms > Organizations** and click **Export Archive**.

Manager exports them into an organization archive ZIP file, which will be downloaded to your PC.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to import an organization.

Import an Organization

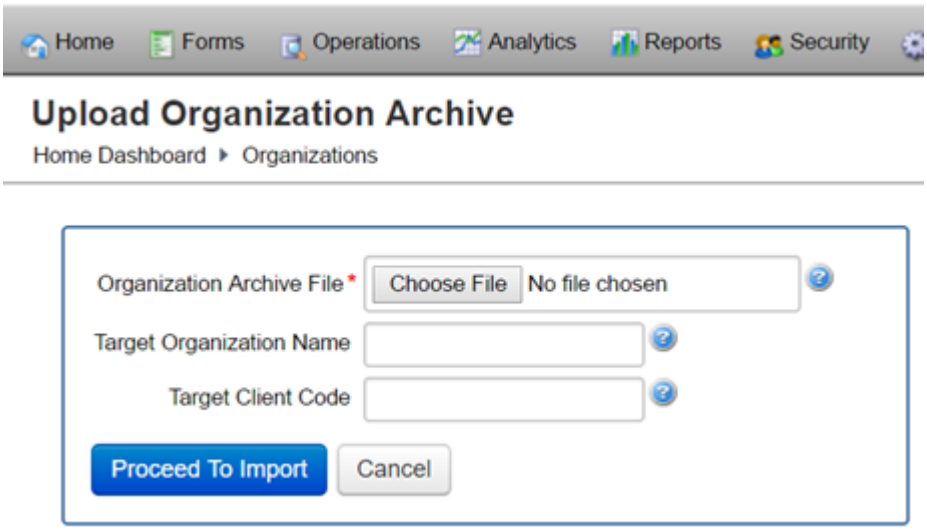
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to import organizations. You can use this functionality to:

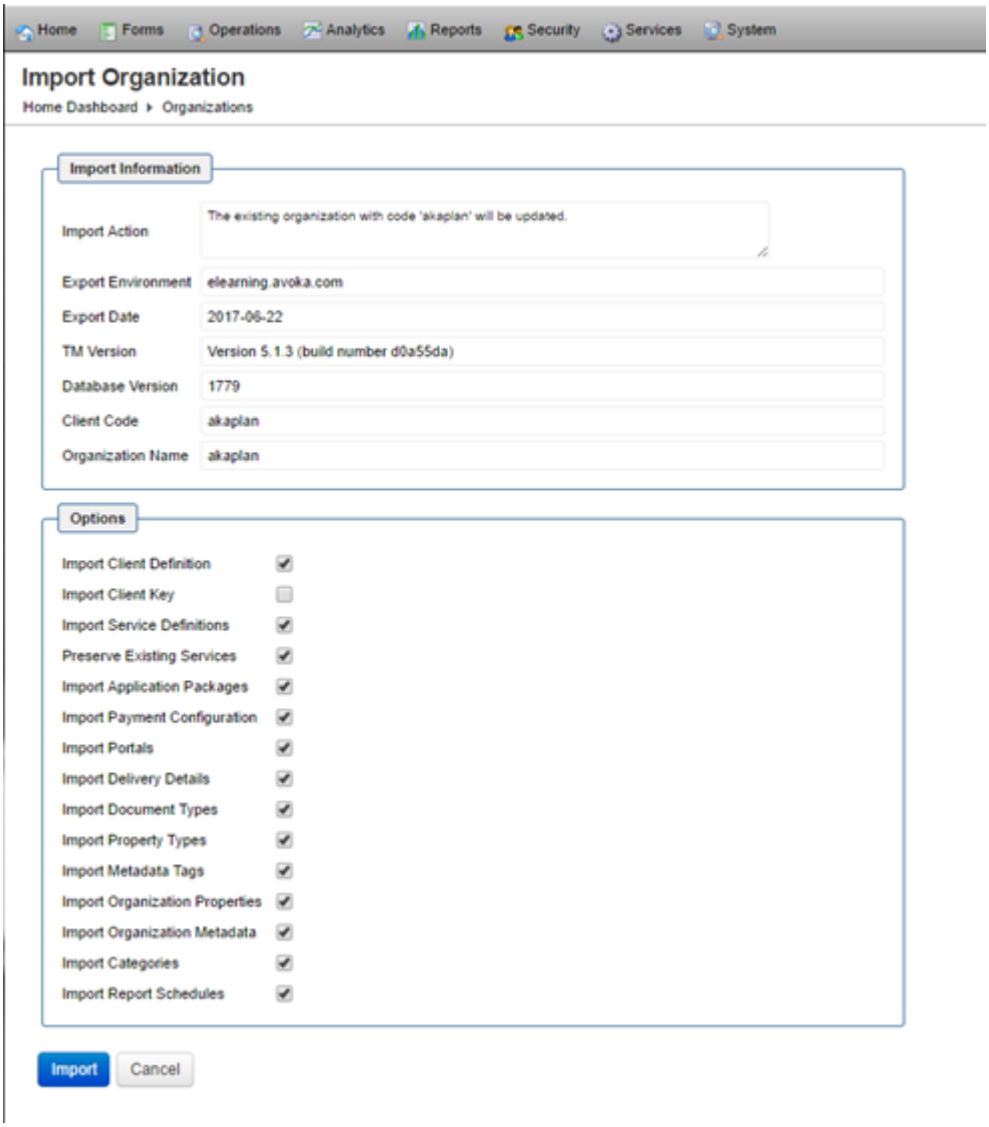
- Roll-back one or several organizations.
- Re-create one or several organizations in another Manager.
- Promote one or several organizations to other environments.

To import one or several organizations:

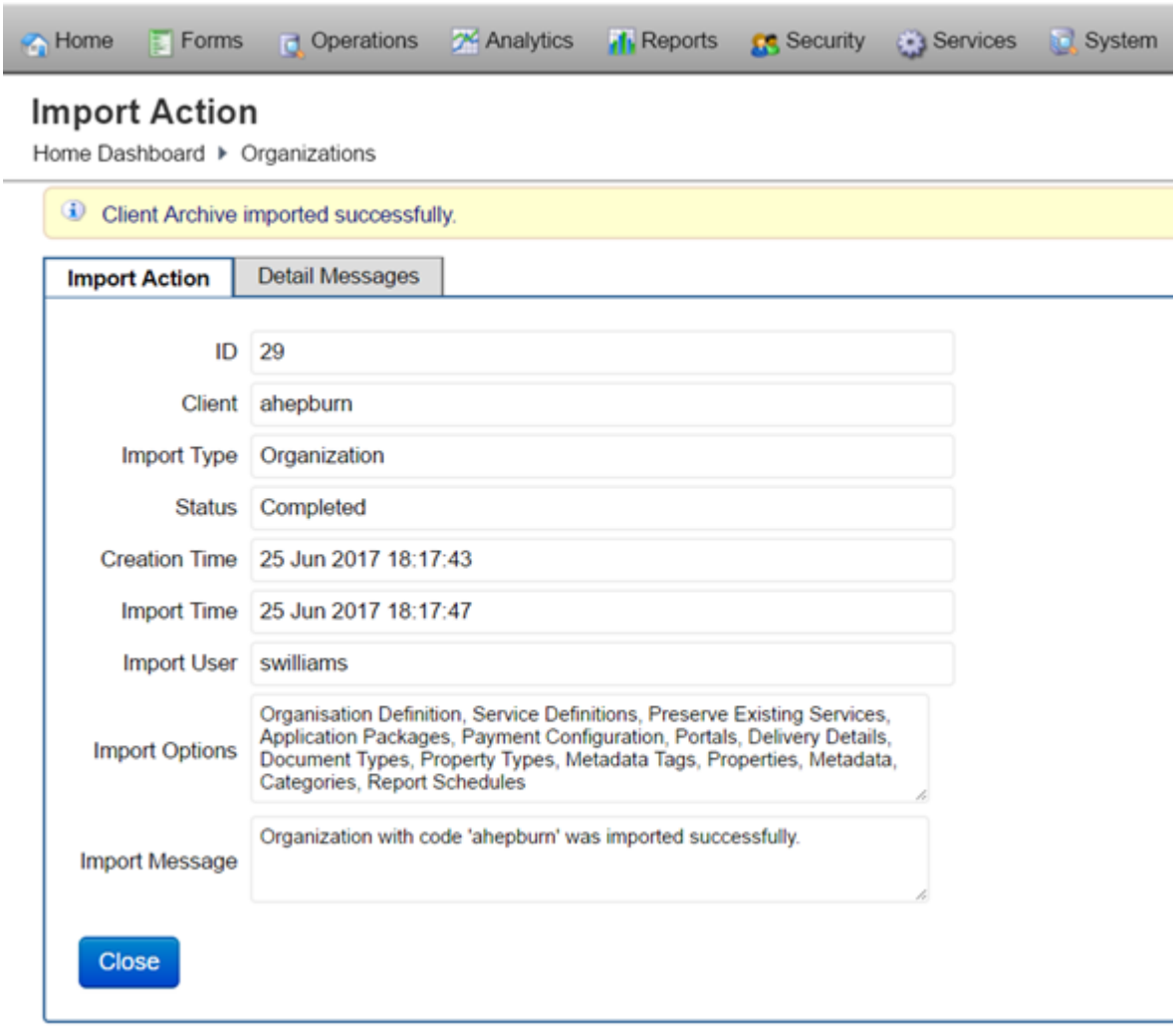
1. Select **Forms > Organizations** and click **Import**.



2. Click **Choose File** and choose an organization archive ZIP file that was created during the organization export.
3. Provide a new **Target Organization Name** and **Target Client Code** to import the organization with a different name and code. You might want to do so if an organization with the same name and code already exist in the target environment. To import the organization as a new one in another Manager environment, leave **Target Organization Name** and **Target Client Code** fields blank.
4. Click **Proceed To Import**.
5. Review the **Import Options**.



6. Select or clear the Options' checkboxes, which identify organization's configurations to be imported.
7. Click **Import** to complete the process. The expected result page looks like this.



8. Click **Close** and view organizations to verify the imported organizations have been added.

Next, learn how to migrate an organization.

Migrate an Organization

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides a mechanism to migrate organizations from one environment to another. It allows you to develop and test organizations in different environments before migrating them to the production environment.

When migrating an organization, all of its associated configurations are also migrated, except the organization administrator's entities and forms with all their data. Once the organization is migrated to a new environment, it will have configuration of the original environment, although entry IDs may change.

To migrate one or several organizations:

1. Export organizations.
2. Import organizations to another Manager environment.

> You can use the same process to perform an organization backup and rollback.

Next, learn how to configure an organization.

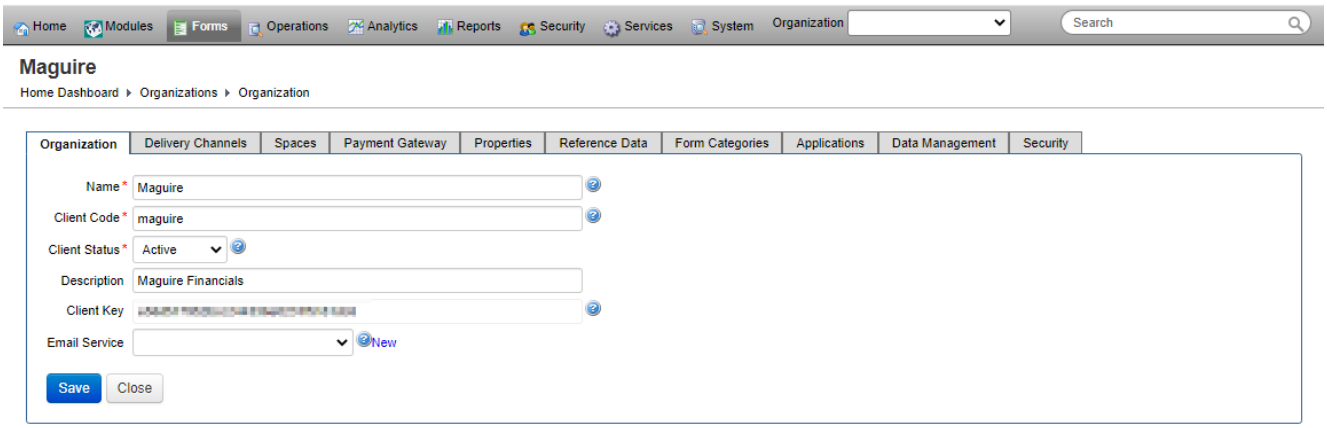
Configure an Organization

Journey Manager (JM) | System Manager / DevOps | 19.05

Manager allows you to create new and configure existing organizations.

To configure an organization:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.



3. Assign spaces that will host this organization's forms. For Maestro Manager ALL organizations MUST have "Web Plug-In" assigned in order for the **Pull From Maestro** button to work in their Client Manager Instance.
4. Create delivery channels that will be available to this organization and its forms and form versions.

   For more information on which delivery channel is used by which form and form version, see delivery channels overview.

5. Configure the payment gateway for the payment details required for any form within this organization to enable credit card payments.
6. Configure organization properties for customizing a broad range of features, such as emails.
7. Configure organization reference data for storing reference data that can be used by any form within this organization. | 17.10 .
8. Configure form categories for the ability to filter forms by categories within form spaces.
9. Configure applications for grouping organization specific services to facilitate their migration between Manager servers.
10. Configure this organization report schedules. | Removed in 19.05.
11. Configure this organization data management.
12. Configure security for general settings relating to form rendering.

Next, learn how to assign form spaces to an organization.

Assign Form Spaces to an Organization

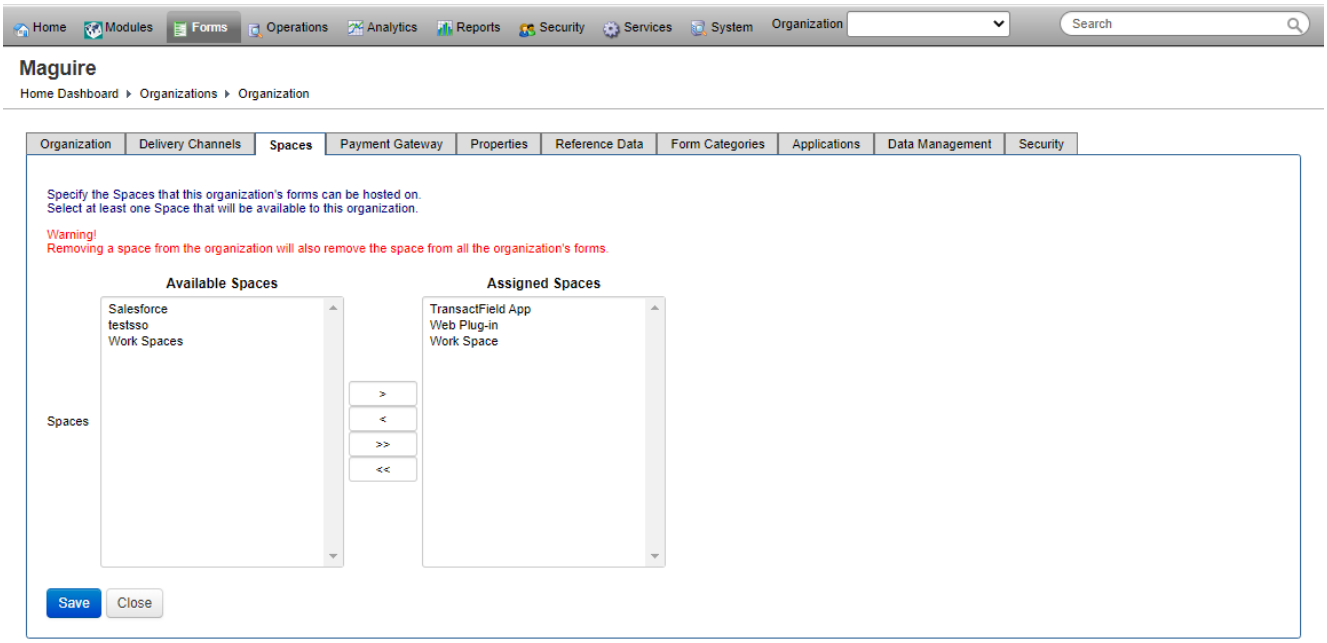
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to assign form spaces to an organization to host all organization's forms and their resources.

> Each organization must be assigned the Web Plug-In form space.

To assign form spaces to an organization:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Spaces** tab to see all assigned form spaces in the **Assigned Spaces** field.



4. Select one or more form spaces from **Available Spaces** that you want to assign to the organization.

> You can use the **Shift** and **Ctrl** keys to select multiple spaces on Windows PC.

5. Click **>** to move the selected form spaces from **Available Spaces** to **Assigned Spaces**.
6. Click **<** to revoke the selected form spaces from **Assigned Spaces** .
7. Click **>>** to assign all form spaces.
8. Click **<<** to revoke all form spaces.
9. Click **Save** to update the changes.

You can assign or re-assign form spaces to an organization at any stage.

Next, learn how to configure an organization payment gateway.

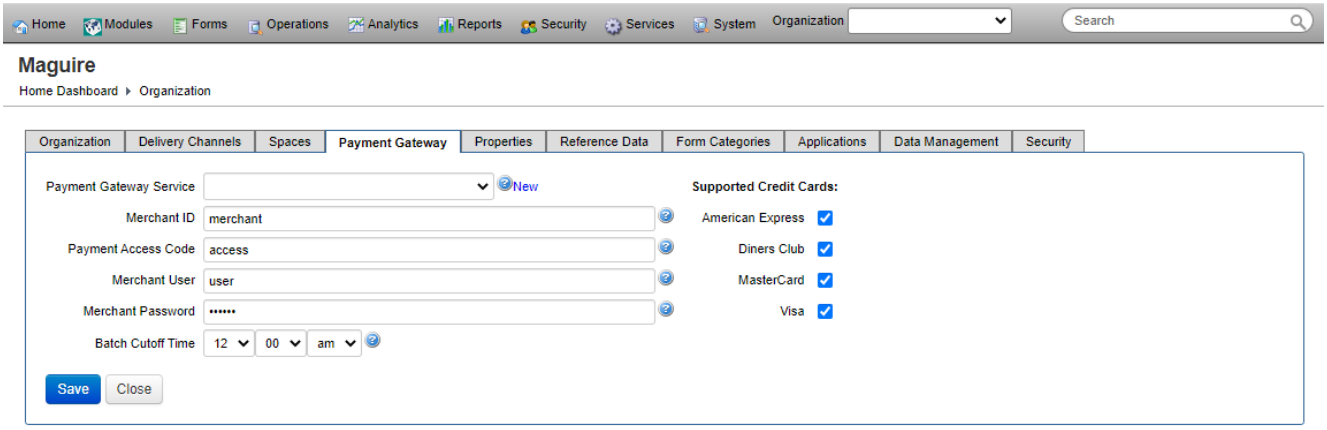
Configure an Organization Payment Gateway

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to enable and configure credit card payment gateways from any form within the selected organization. The payment gateway is necessary when a form requires additional payment details from a user as part of a successful form submission.

To configure a payment gateway:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Payment Gateway** tab.



4. Select a payment gateway service from the **Payment Gateway Service** dropdown list that this organization uses when processing credit card payments.

> An administrator must create a service for the payment gateway and make it available to this organization.

5. Enter the merchant account number into the **Merchant ID** field that specifies the merchant's account to the payment gateway.
6. Enter the merchant payment access code into the **Payment Access Code** field that specifies the payment access code used to authenticate the merchant account on the payment gateway.
7. Enter the merchant user name into the **Merchant User** field specifies the merchant's user name if it is requested by the payment gateway.
8. Enter the merchant password into the **Merchant Password** field that specifies the merchant's password for when a password is requested by the payment gateway.
9. Select the cut-off time from the **Batch Cutoff Time** dropdown lists that specifies the time of day when payments stop being transferred to the organization's bank account on the same day.
10. Select the credit cards that this payment gateway accepts in the **Supported Credit Cards** section:

- American Express
- Diners Club
- MasterCard
- Visa

11. Click **Save** to update the changes.

Next, learn how to configure organization properties.

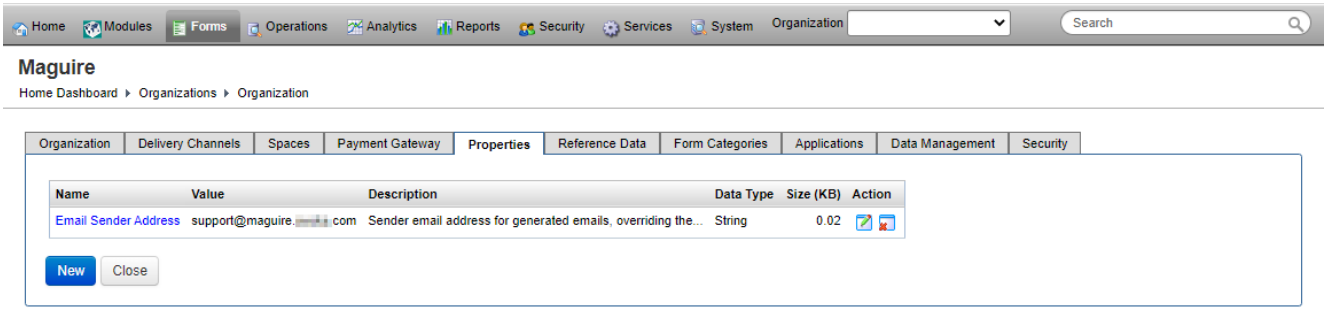
Configure Organization Properties

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to add new and configure existing organization's properties. Organization properties can be used to pre-populate forms or modify their behavior.

To configure organization properties:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Properties** tab to view, create and edit properties.



4. Select a property and click **Edit**.

Next, learn how to configure reference data.

Configure Organization Metadata Tags

Journey Manager (JM) | System Manager / DevOps | Removed in 18.05.

Manager allows you to configure metadata tag definitions for your organization. Metadata can be used to run queries on the set of forms defined in Manager using a Web Service API. If you want to define metadata for an organization, you can configure values for these metadata tags. All forms will inherit organization metadata when they are created.

To configure a metadata value:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Form Tags** tab and click **New**.
4. Choose a metadata tag to assign a value to from the **Tag Name** dropdown.

   Only metadata tags with a scope of **Client** will be displayed

5. Set **Tag Value** to the metadata value for this organization and save. You will see that the metadata value has been added to the **Form Tags** tab.
6. Click **Save** to update the changes. All forms that are set up for this organization will inherit this metadata value.

Next, learn how to configure reference data.

Configure Organization Reference Data

Journey Manager (JM) | System Manager / DevOps | 17.10

Manager allows you to view, create, configure, and test reference data for a selected organization. This is handy, because, since Maestro 17.10, reference data can be stored in Manager and accessed in Maestro forms by calling the `referenceData` function in the Transact name space.

To start using reference data in your applications, you should understand more about its structure and types. You can think of reference data as an object that:

- is defined at the organization level
- is uniquely identified in an organization by a name and locale
- supports language translation through different locales
- supports the **CSV**[1], **JSON**[2] and **XML**[3] data types
- supports optional dynamic filtering and sorting of the CSV data type
- supports server caching, with configurable refresh time-out specified for each reference data item

Manager manages reference data using the following file formats:

- CSV - supports filtering and sorting, so it has specific requirements described below. The first line of a CSV file must be a heading line containing comma-separated field names with corresponding data following on subsequent lines. Field names defined in the first line are used in filtering and sorting order options by Maestro clients. Field names in the heading line are case sensitive.
- JSON and XML - no specific format requirements and are returned as-is to the requesting Maestro client.

Every reference data item must be defined with a **locale**[4]. The locale, in conjunction with the reference data **Name** property, forms a unique combination. Both the name and locale must be specified when requesting reference data in a Maestro form.

To view organization reference data:

---

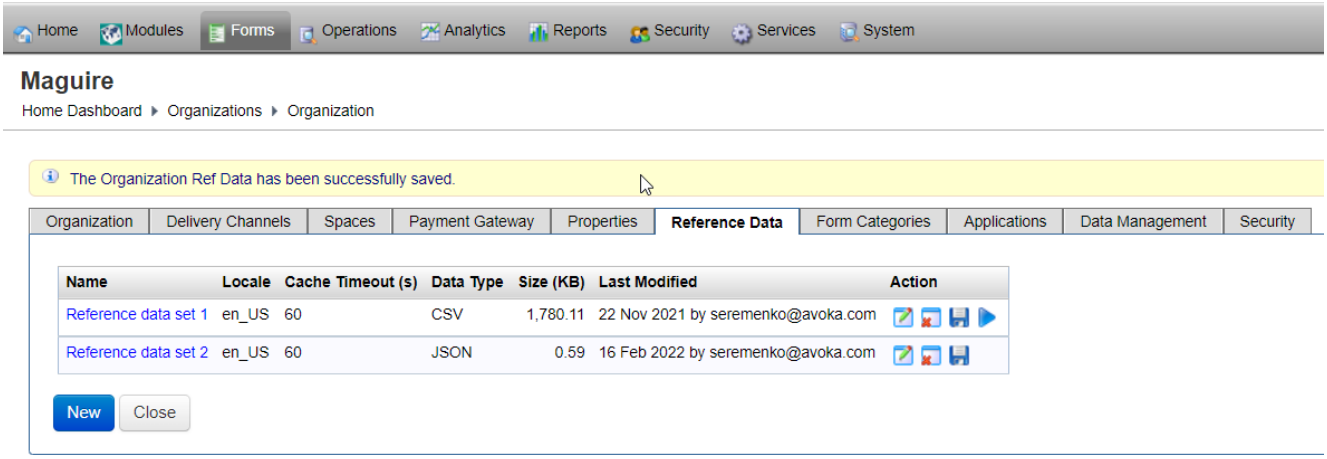
[1]CSV is a comma-separated values fil, which s a delimited text file that uses a comma to separate values. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas.

[2]JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

[3]XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

[4]Locale is a set of parameters that defines the user's language, region and any special variant preferences that the user wants to see in their user interface. Usually a locale identifier consists of at least a language code and a country/region code.

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Reference Data** tab.



You can check the following details of all configured reference data:

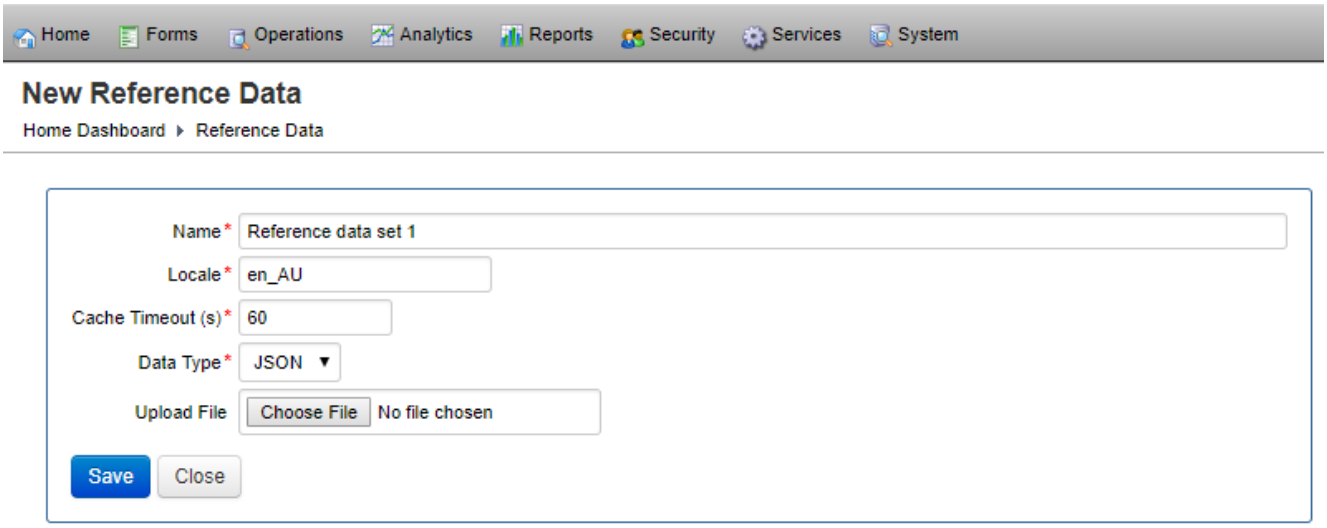
- Name of the reference data set
- Locale of the data set
- Cache timeout
- Data type
- Size of the reference data set
- Last modified

4. Click **Edit** to update the existing reference data.
5. Click **Remove** to delete a reference data.
6. Click **Download as File** to save a reference data set on your local PC.
7. Click **Query Test** to test your reference data set by running a test query against a reference data set.

> This option is only available for CSV data type.

To create and configure organization reference data:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Reference Data** tab.
4. Click **New** to create a new reference data set, or click **Edit** to update the existing reference data set.



5. Enter a unique name in the **Name** field.
6. Enter a locale in the **Locate** field, for example en_AU.
7. Provide cache timeout in seconds in the **Cache Timeout** field.
8. Select a data type from the **Data Type** dropdown list, which can be one of the following:
   - CSV
   - JSON
   - XML
9. Click **Choose File** to upload a reference data file. Select a reference data file that matches the specified **Data Type**.
10. Click **Save** to update the changes.

> We recommend checking feference data example for step-by-step instructions on how to use reference data from a Maestro form.

Next, learn how to configure organization form categories.

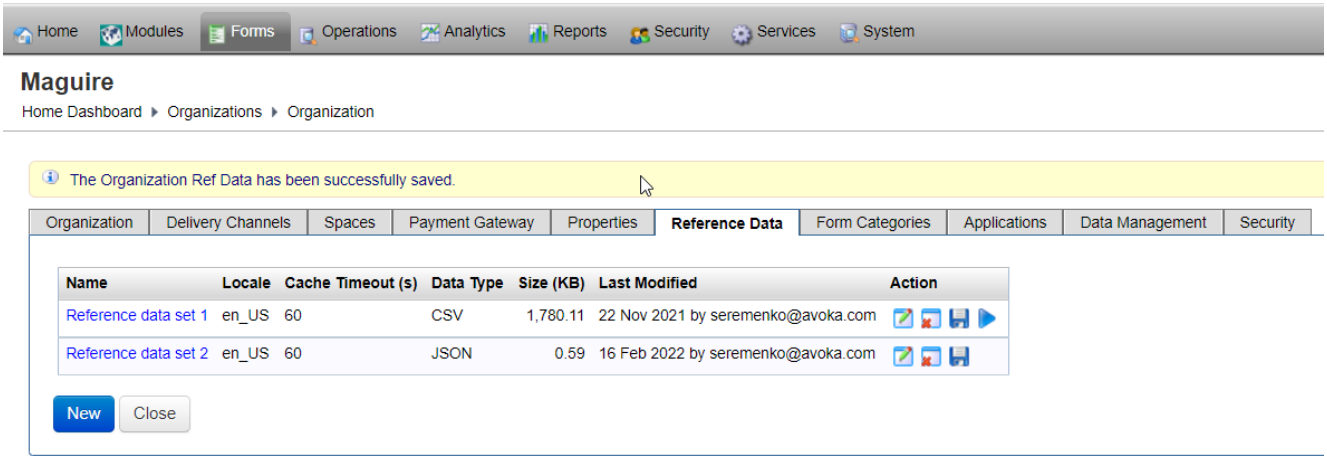
Test Organization Reference Data

Journey Manager (JM) | System Manager / DevOps | 17.10

Manager allows you to test reference data of CSV[1] data type for a selected organization, so you can quickly test filtering and sorting of CSV reference data. This is handy while working on or troubleshooting Maestro forms.

To test organization reference data:

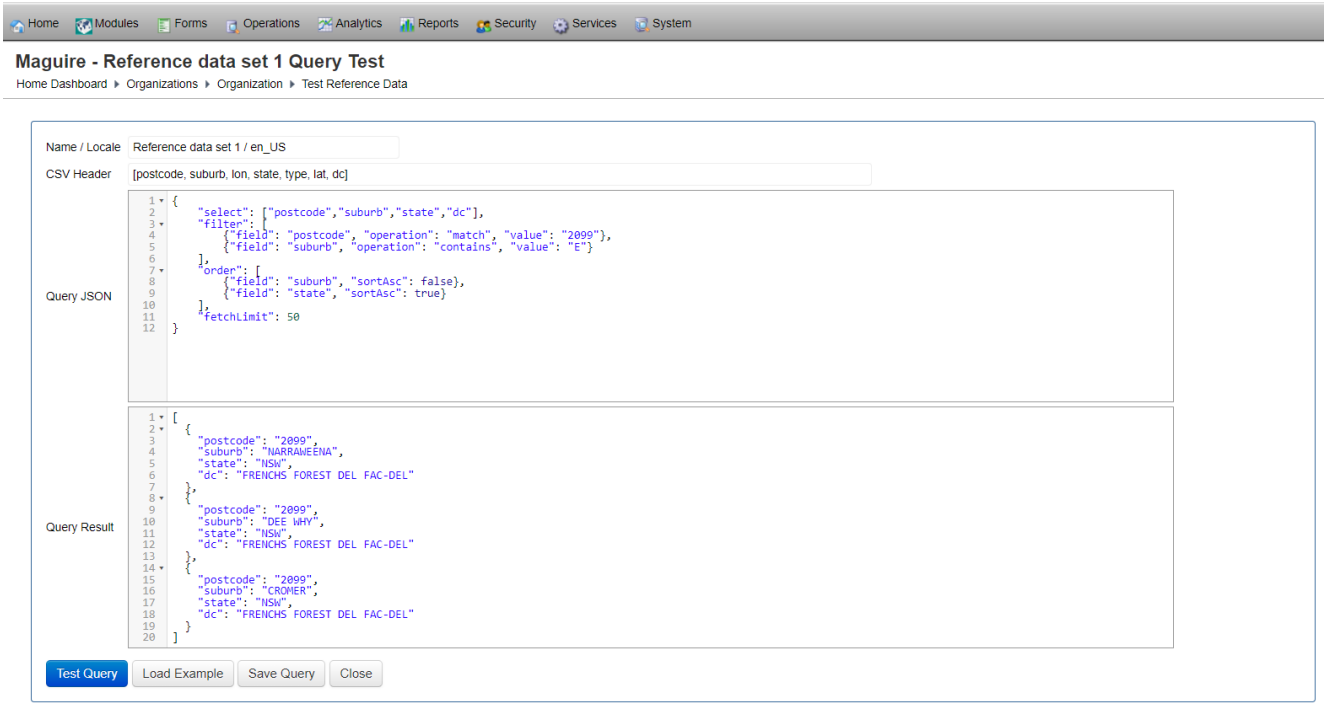
1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Reference Data** tab to see all configured reference data sets.



4. Choose reference data you want to test and click **Query Test**.

> This option is only available for CSV data type.

[1]CSV is a comma-separated values fil, which s a delimited text file that uses a comma to separate values. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas.



5. Update a query request in the **Query JSON** field and click **Test Query**. For more information, see query syntax below.
6. Check a query result in the **Query Result** field.
7. Click **Load Example** to upload a new query.
8. Click **Save Query** to save the current query for the future use.

Query Syntax

A query you write to sort and filter reference data is a **JSON**[1] object that is applied to the CSV data. You must adhere to valid JSON syntax, otherwise Manager throws an exception. For more information, see Introducing JSON.

The query object consists of four elements.

Element	Type	Description
`select`	array of string	The list of fields (described in the next table) that will be returned in the **Query Result** box.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

filter	array of objects	A list of conditions to constrain the data returned in **Query Result**. Each filter condition is an object consisting of three elements: **field**, **operation** and **value**(described in the next table). The operation is used to perform a case-insensitive test of the field against the value, returning a true result for a successful test, and a false result otherwise.
order	array of objects	A list of sorting specifications that control the order that CSV rows are returned. Each sorting specification is an object consisting of two elements: **field** and **sortAsc** (described in the following table).
fetchLimit	number	The maximum number of rows returned from the CSV file, applied after the filter and sort operations have been applied.

The **select**, **filter** and **order** array elements are constructed using the following elements.

Element	Type	Description
field	string	A case sensitive string matching a column heading in the first line of the CSV file.
operation	string	A case sensitive string, which must be one of the following: **match**, **contains**, **startsWith**, **endsWith**, **greaterThan**, **lessThan**, **greaterThanEqual**, **lessThanEqual**.
sortAsc	string	Specifies how an order specification is applied to the output; either ascending (**true**) or descending (**false**).
value	string	A case-insensitive string used by a filter condition to test against the values in the column specified by the field property.

Next, learn how to configure organization form categories.

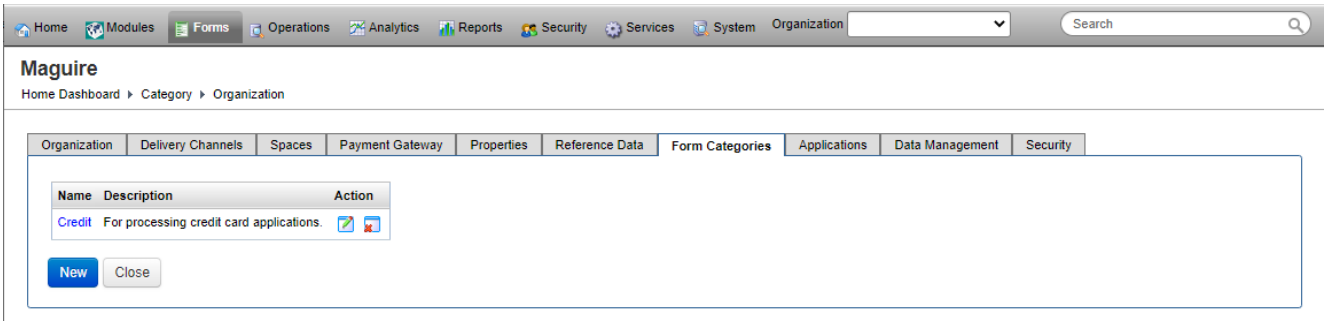
Configure Organization Form Categories

Journey Manager (JM) | System Manager / DevOps | 19.05

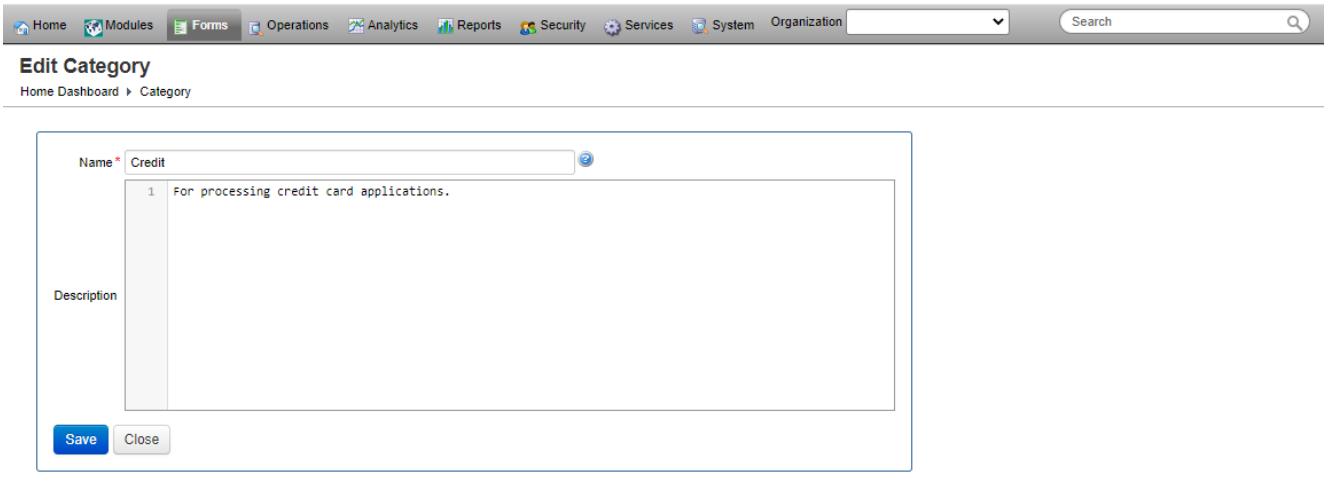
A form space hosting your organization's forms may contain many forms. Manager allows you to create form categories to assist users with locating forms they need in the **Forms** tab. The standard form spaces support the form categories filter. If your form space has been customized, verify that this filter is still supported.

To create and configure a form category for an organization:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Form Categories** tab to see all existing form categories.



4. Locate a form category and click **Edit** or click **New** to create a new form category



5. Enter a unique form category name into the **Name** field.
6. Enter a suitable description into **Description** field.
7. Click **Save** to update the changes.
8. Assign each form version to the related form categories
9. Click **Edit** to configure the existing form category.

Next, learn how to configure an application package for an organization.

Create an Application Package

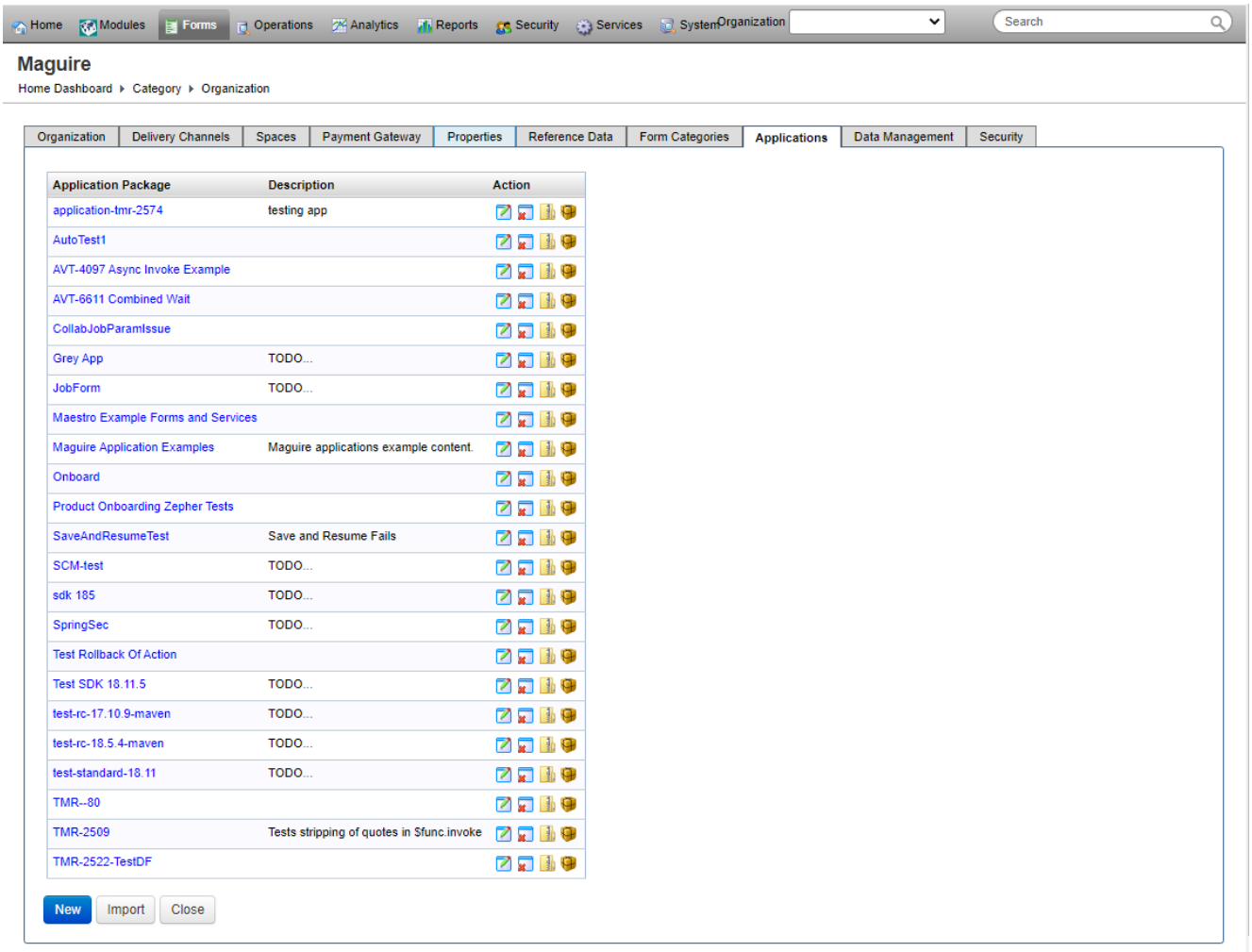
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to package forms and related services and assets into an application package, so you can easily migrate it between organizations in different Manager instances.

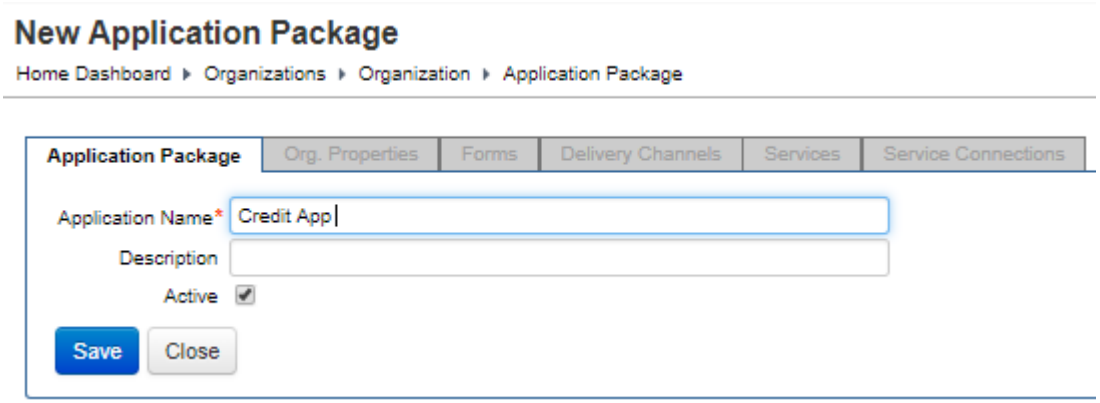
To package a form, you need to create a new application package and assign the required services and forms to it. You can also assign the required organization properties, delivery channels, and service connections, where applicable.

To create an application package:

1. Select **Forms > Organizations.**
2. Locate an organization that you want to configure or manage existing application packages.
3. Select the **Applications** tab.



4. Click **New**.



5. Enter a unique name in the **Application Name** field.
6. Enter a suitable description in the **Description** field.
7. Select the **Active** checkbox, unless you don't want the application package to be used.
8. Click **Save** to update the changes.

Next, learn how to configure an application package for an organization.

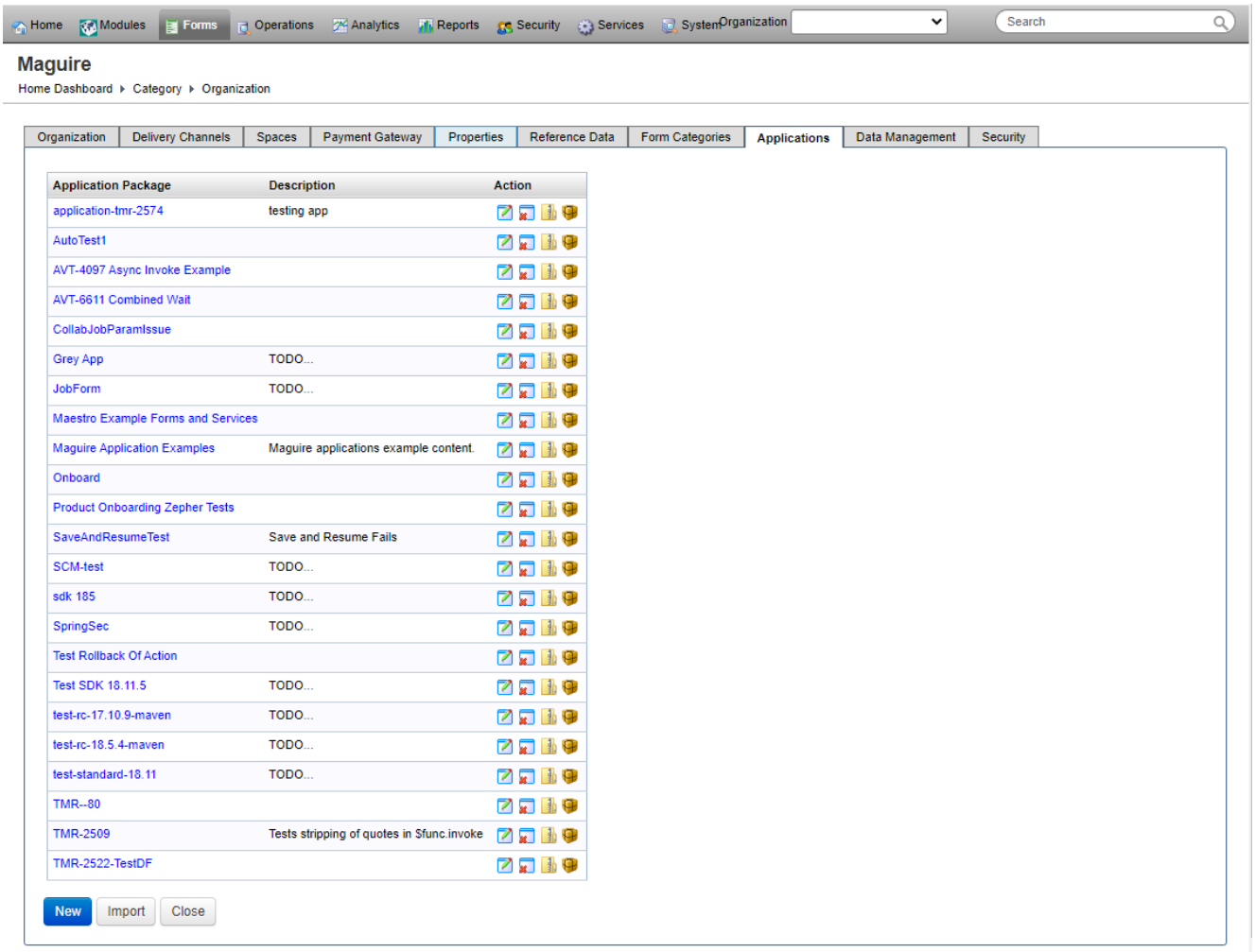
Configure an Application Package for an Organization

Journey Manager (JM) | System Manager / DevOps | 19.05

Manager allows you to view application packages, create new and configure existing ones for the selected organization.

To view application packages:

1. Select **Forms > Organizations**.
2. Locate an organization.
3. Select the **Applications** tab.



You can check the following details of all configured application packages:

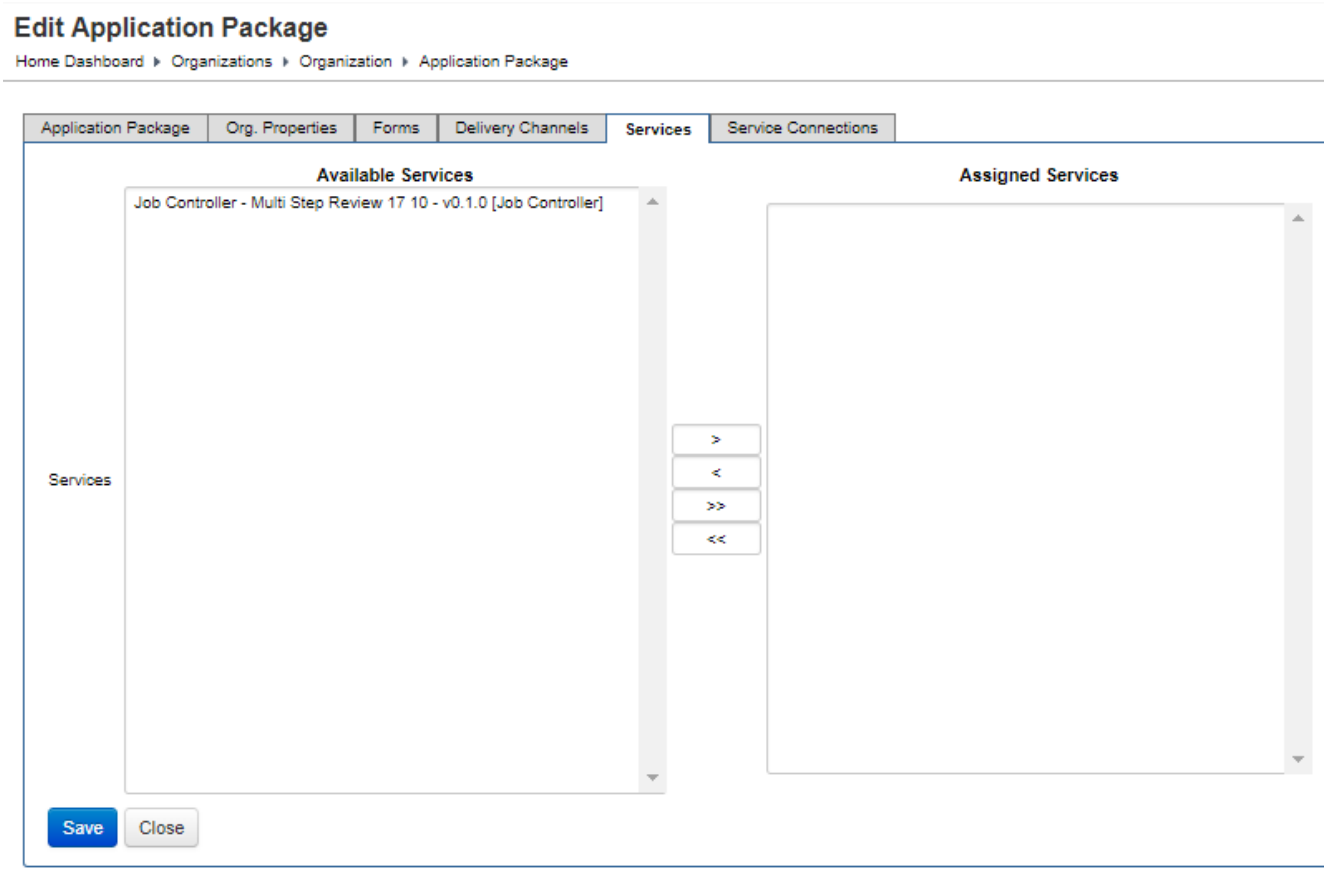
- Application package
- Application package description

4. Click **Edit** to update the existing application package.
5. Click **Remove Application Package** to delete an application package.

6. Click **Export Archive (Current Form Versions)** to export an application package including current form versions only.
7. Click **Export Archive (All Form Version)** to export an application package including all form versions.
8. Click **New** to create a new application package.
9. Click **Import** to import an application package.

To configure an application package:

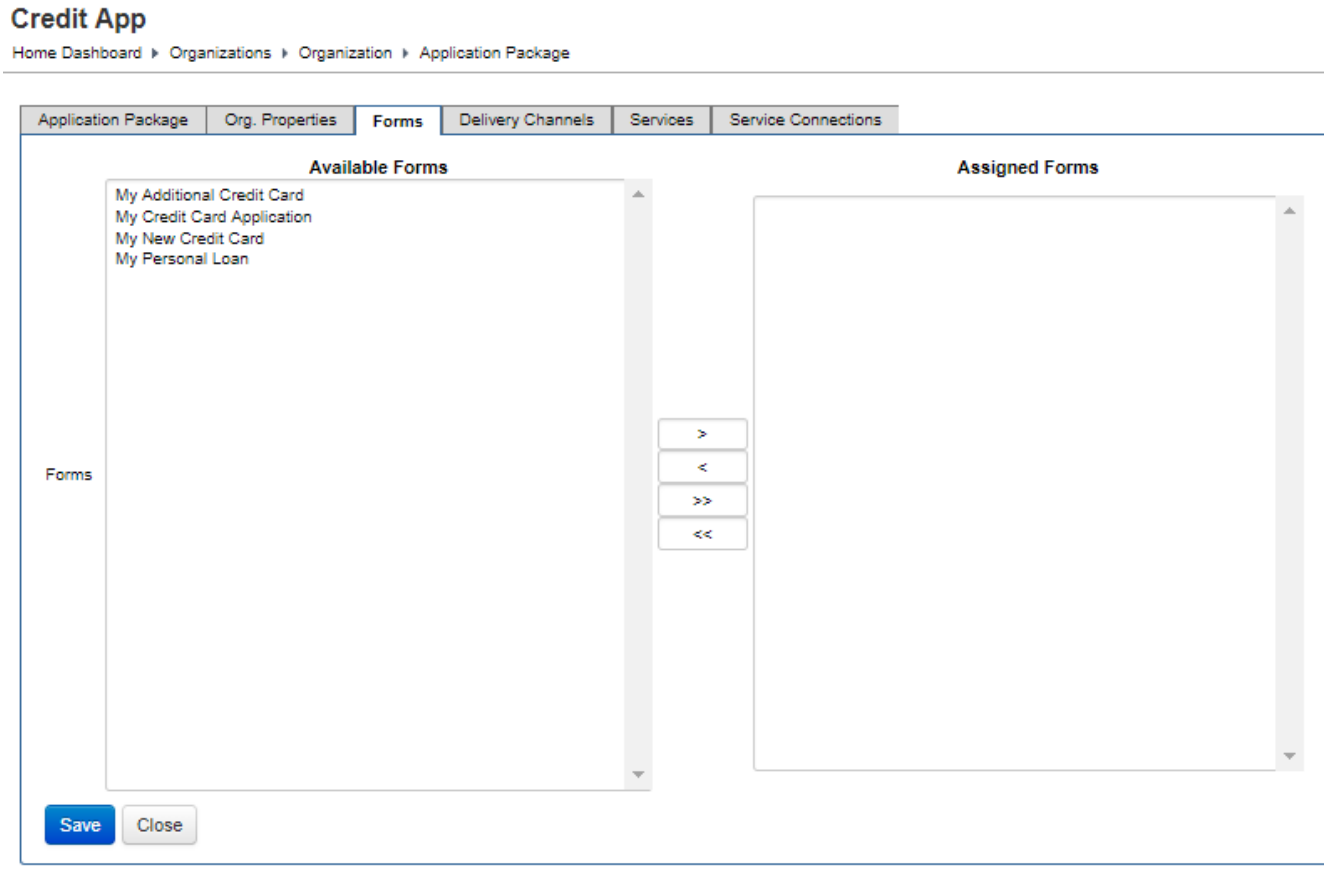
1. Select **Forms > Organizations**.
2. Locate an organization.
3. Select the **Applications** tab.
4. Locate an application package and click **Edit**.
5. Assign the services from the **Services** tab to this application package.



You can't assign global services to an application package.

6. Select one or more services from **Available Services** that you want to assign to the application package. You can use the Shift and Ctrl keys to select multiple services on Windows PC.
7. Click **>** to move the selected services from **Available Services** to **Assigned Services**.

8. Select the services from **Assigned Services** and click **<**.
9. To assign all Services, click **>>**.
10. To revoke all Services, click **<<**.
11. Click **Save** to update the changes.
12. Select the **Forms** tab to assign forms to the application package.



13. Click **Save** to update the changes.
14. Select the **Org. Properties** tab to assign organization properties to the application package.
15. Click **Save** to update the changes.
16. Select the **Delivery Channel** tab to assign delivery channels to the application package.
17. Click **Save** to update the changes.
18. Select the **Service Connections** tab to assign service connections to the application package.
19. Click **Save** to update the changes.

You can also export and import application packages so you don't need to configure them every time in a new environment.

Next, learn how to export a package.

Export an Application Package

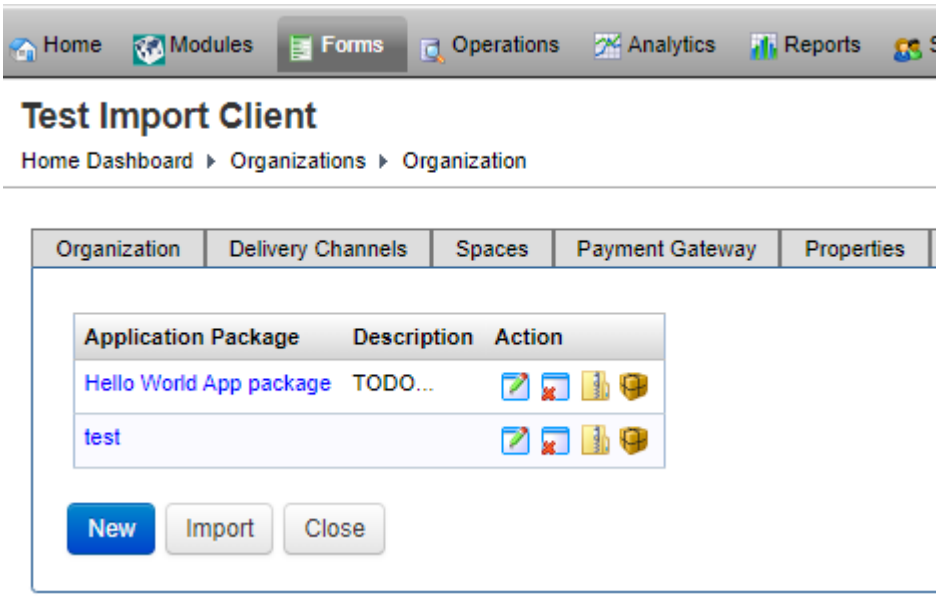
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to export an application package from an organization. You can use this functionality to:

- Backup or archive applications.
- Re-create or migrate applications in another Manager.
- Promote one or several applications to other environments.

To export an application package:

1. Select **Forms > Organizations.**
2. Locate an organization that you want to configure or manage existing application packages.
3. Select the **Applications** tab.



4. Locate your application and click **Export Archive (Current Form Versions)** or **Export Archive (All Form Versions)**. Manager exports them into an application archive ZIP file, which will be downloaded to your PC.

Next, learn how to import an application package.

Import an Application Package

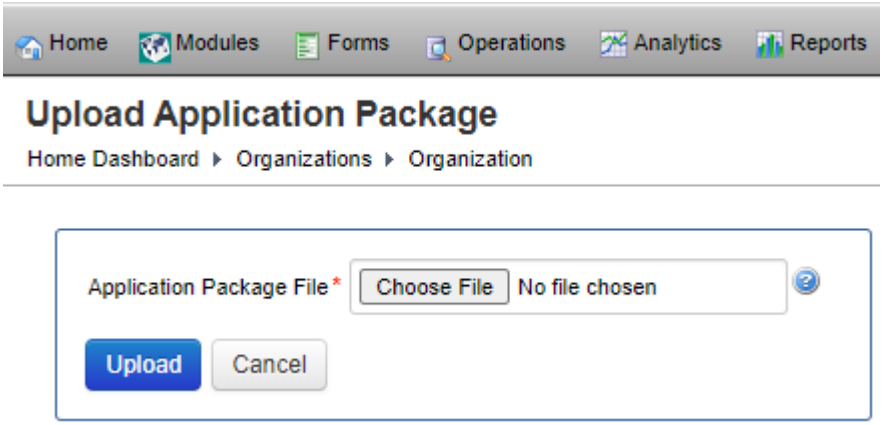
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to import an application package into an organization. You can use this functionality to:

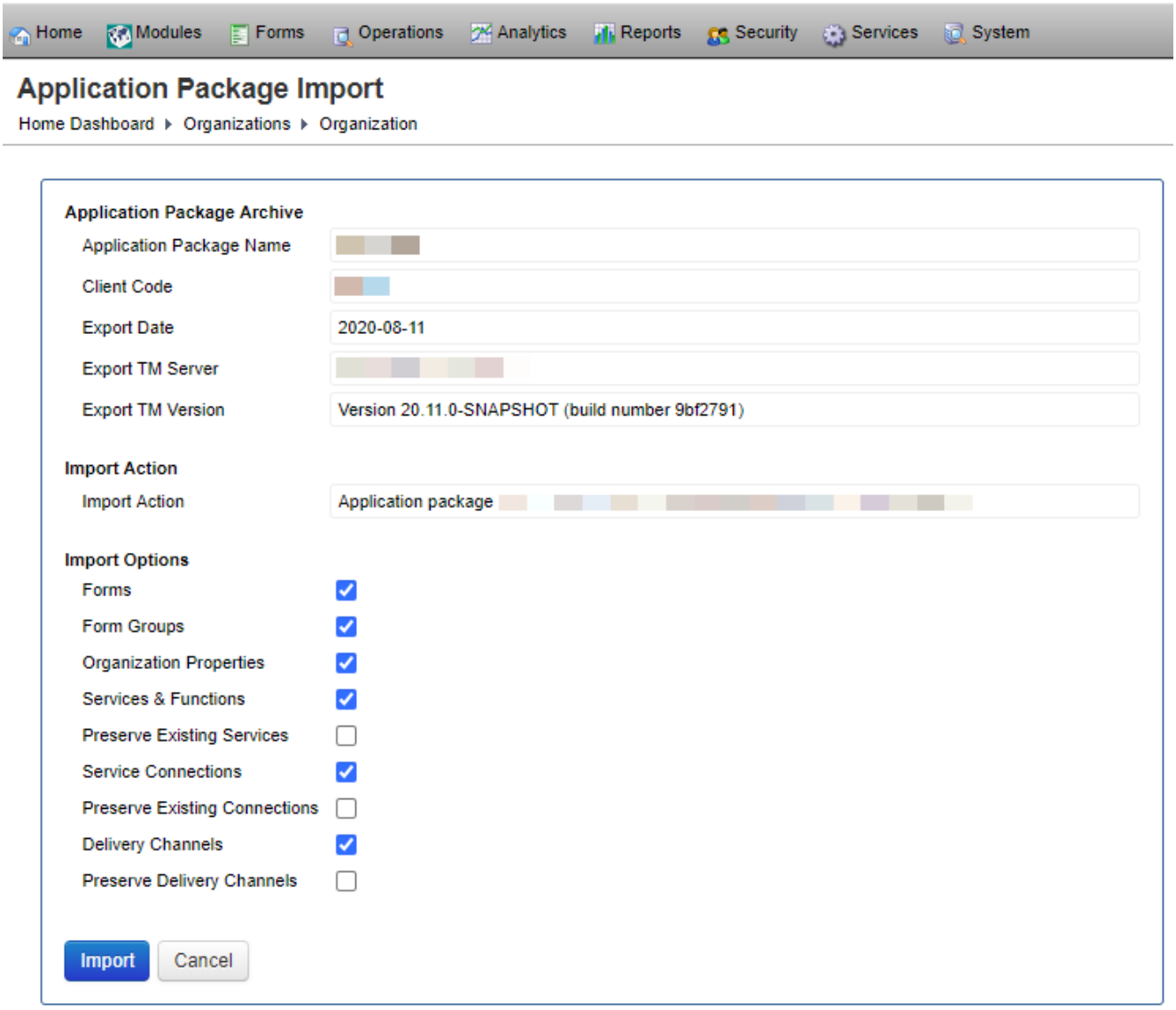
- Roll-back one or several applications.
- Re-create one or several applications in another Manager.
- Promote one or several applications to other environments.

To import an application package:

1. Select **Forms > Organizations.**
2. Locate an organization that you want to configure or manage existing application packages.
3. Select the **Applications** tab.
4. Click **Import**.



5. Click **Choose File** and choose an application archive ZIP file.
6. Click **Upload.**
7. Review the **Import Options**.



8. Select **Forms** to import forms from the archive file into the target environment. The existing forms will be overwritten. This is the default option.
9. Select **Form Groups** to import form groups from the archive file into the target environment. The existing form groups will be overwritten. This is the default option.
10. Select **Organization Properties** to import organization properties from the archive file into the target environment. The existing organization properties will be overwritten. This is the default option.
11. Select **Services & Functions** to import services and Transact Functions from the archive file into the target environment. The existing services and Transact Functions will be overwritten. This is the default option. For more information, see import services and import Transact Functions.
12. Select **Preserve Existing Services**, so any services that already exist in the target environment will not be modified in any way. Only new (to the target environment) services will be imported.

If you import a new version of a service, clear the **Preserve Existing Services**, then manually activate this service version and update the configurations that refer to the old version of the service to point to the new version.

13. Select **Service Connections** to import service connections from the archive file into the target environment. The existing service connections will be overwritten. This is the default option. For more information, see import service connections.
14. Select the **Preserve Existing Connections**, so any service connections that already exist in the target environment will not be modified in any way. Only new (to the target environment) service connections will be created.
15. Select **Delivery Channels** to import delivery channels from the archive file into the target environment. The existing delivery channels will be overwritten. This is the default option.
16. Select **Preserve Delivery Channels**, so any delivery channels that already exist in the target environment will not be modified in any way. Only new (to the target environment) delivery channels will be created .
17. Click **Import** to complete the process.

You can also check the REST Application Package API documentation pertaining to your Manager version, for example v2105/tsg/rest-api/rest-application-package-api.html#put-update-application-package.

Next, learn how to configure organization's security.

Configure Organization Security

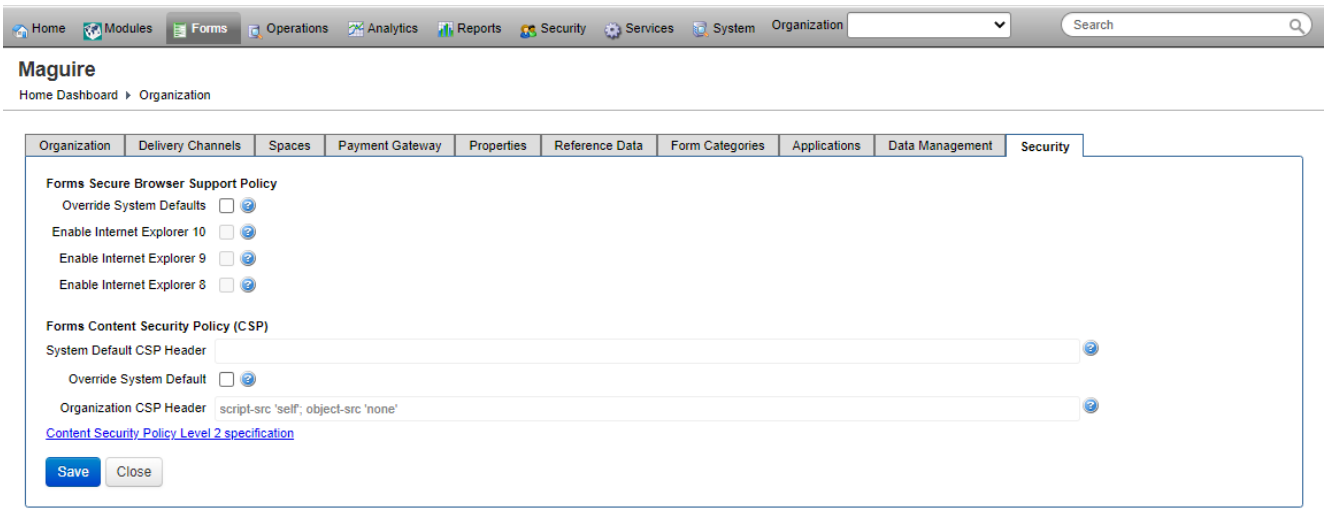
Journey Manager (JM) | System Manager / DevOps | 19.05

Manager allows you to configuring an organization security to allow or refuse form rendering on certain browsers or form requests with certain headers. Manager has a black list of older IE browsers that are not supported. Otherwise, it does a white list, so new browsers (meaning new mobile devices) may come into the market and they are not blocked automatically.

> The system settings are defined in the default Form Submission Access Controller service.

To configure organization security:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Security** tab to override browser's support policies (see the **Forms Secure Browser Support Policy** section) and specify a **Content Security Policy**[1] header that will be used by all forms (see the **Forms Content Security Policy (CSP)** section).



4. Select the **Override System Defaults** checkbox to override the system default supported secure browser configurations.
5. Select the **Enable Internet Explorer 10** checkbox to enable the Microsoft IE10 browser.

---

[1]Content Security Policy (CSP) is a computer security standard introduced to prevent cross-site scripting (XSS), clickjacking and other code injection attacks resulting from execution of malicious content in the trusted web page context. CSP provides a standard method for website owners to declare approved origins of content that browsers should be allowed to load on that website—covered types are JavaScript, CSS, HTML frames, web workers, fonts, images, embeddable objects such as Java applets, ActiveX, audio and video files, and other HTML5 features. For more information, see https://www.w3.org/TR/CSP/

Microsoft IE10 browser is not supported by Microsoft and Manager for security reasons.

6. Select the **Enable Internet Explorer 9** checkbox to enable the Microsoft IE9 browser.

Microsoft IE9 is not supported by Microsoft and for security reasons.

7. Select the **Enable Internet Explorer 8** checkbox to enable the Microsoft IE8 browser.

Microsoft IE8 is not supported by Microsoft and Manager for security reasons.

8. Provide the form CSP security HTTP header value used if not overridden by the organization CSP header value in the **System Default CSP Header** field.
9. Select the **Override System Default** checkbox to override the system default Content Security Policy (CSP) configuration.
10. Provide the form CSP security HTTP header value in the **Organization CSP Header** field. Any form space property **CDN Server URL** value will be automatically injected into `script-src-self` value.
11. Click **Save** to update the changes.

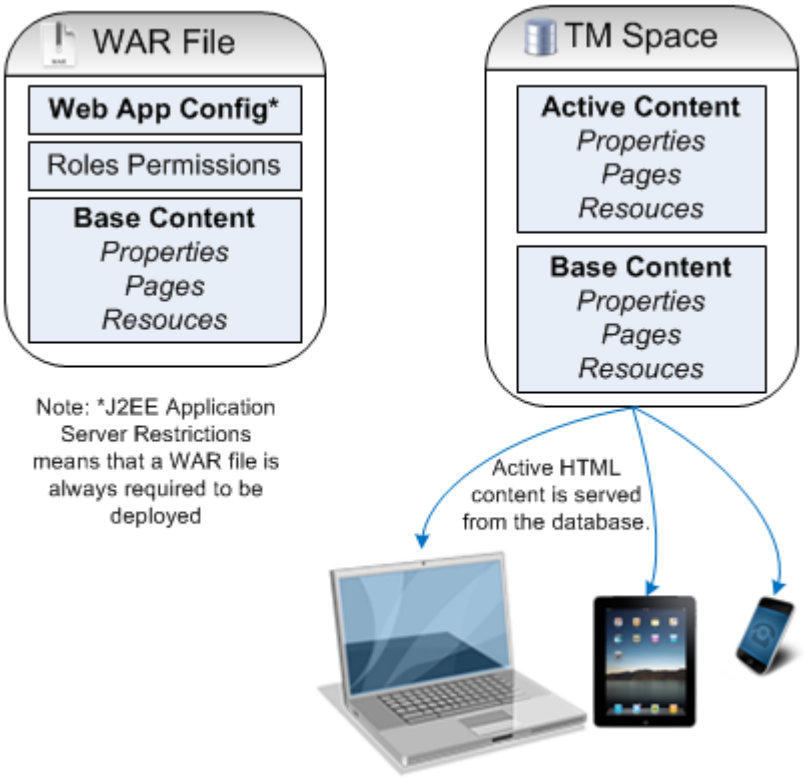
Next, learn how to view organizations.

Form Spaces Overview

Journey Manager (JM) | System Manager / DevOps | 21.11

Manager provides form spaces that are used to host forms and expose them to users, handle form requests, such as form submissions, list tasks assigned to users, and perform user authentication. You can configure a form space to define whether hosted forms require authentication or not by selecting a security manager and authentication mode.

In Java Enterprise Edition (JEE) terms a form space is a Java Web Application, which is deployed using a Web Application Archive (WAR) file. Unlike traditional JEE applications, Manager form spaces WAR files are used to initialize the Manager database with appropriate configurations and HTML content. The Java application code is provided by the application server modules and the HTML content is rendered from the Manager database. We use this model to make application upgrades easier. J2EE Application Server restrictions mean that you are always required to deploy a WAR file to apply new changes. After that, the WAR files content becomes the base content in the database. Then, the active content in a form space is either edited in Manager or imported as an archive ZIP file.



The form space name follows this pattern: **`avoka-sf-portal-<portal context>.war`**.

Every form in the Journey platform is associated with one or more configured form spaces. If a form is associated with multiple form spaces, the form and all its saved and completed submissions will be visible and available to all of those spaces.

A form space can be licensed for internal and external transactions depending on the customer license agreement. You can generate transaction licensing reports to view internal and external transaction trends.

To create a new form space, you can import it as a **WAR**[1] file. You can then promote it from one environment to another.

Structure of Form Space WAR File

A form space WAR file includes configurations and HTML content only. It doesn't include any application source code. The typical form space WAR file contains:

- Page Templates as HTML files. They are stored in:
  - Unsecured root directory, which can be accessed anonymously.
  - Secure directory, which requires an authentication user session. This directory contains the pages users sees on the form space after logging in, for example, the **Tasks** or **To Do** page.
- Resources that are used to style a form space, including:
  - css
  - images
  - includes: HTML snippets
  - js: JavaScript files
- WEB-INF that contains the list of XML files and the classes. The most important files are:
  - web.xml: The web application deployment descriptor file describes how the web application should be deployed to the application server. It is required for every web application. This file has the context path space name substitutions applied by the installer.
  - jboss-web.xml: This file has the context path space name substitutions applied by the installer and the context path specified in the installer has been written into the **`context-root`** element.
- WEB-INF/classes contains the compile Java packages and classes. The most important files are:
  - db-config.xml: It is used to create all the new form space properties when the WAR file is deployed. The Roles for the form space are also defined here. This file has the context path space name substitutions applied by the installer.

Default Form Spaces

Manager comes with the following default form spaces, which you can opt in or out during Manager installation:

Salesforce

Salesforce is the form space to enable integration with Salesforce.

---

[1]WAR file (Web Application Resource or Web application ARchive) is a file used to distribute a collection of JAR-files, JavaServer Pages, Java Servlets, Java classes, XML files, tag libraries, static web pages (HTML and related files) and other resources that together constitute a web application.

TransactField App

TransactField App is the business field work personnel form space that allows working offline. It provides a mobile app and PC application which enables business field staff to complete forms offline. Once online, the application allows them to sync their work with the Manager server, and receive new work tasks to complete. TransactField App supports authenticated users only. | Deprecated in 21.11

Web Plug-in

Web Plug-in is the default form space for forms to be discovered and linked to public web sites. It provides a form space for customer facing sales and service transactions. It is linked from your organization's website, but can also be embedded on your website if the form is fairly small. Web Plugin supports unauthenticated and authenticated users. Maestro requires this form space for building and publishing form versions.

Work Space

Work Space allows business staff to perform work functions as a group by creating tasks, collaboration jobs, completing ad-hoc forms, and providing help desk support. The work space is generally used by authenticated users but it also supports unauthenticated transactions as well.

> You should use Work Spaces instead.

Pre-configured Form Spaces

Additionally, you can install the following pre-configured form space.

Work Spaces

Work Spaces is the recommended form space for the business staff to perform work group functions. | 18.11

Maguire

Maguire is the example of self-service form space. | Deprecated in 18.05

> If a form space becomes deprecated, you may need to remove it manually from your Manager instance as it will not work.

Manager comes with a number of modules, which configuration might look similar to form spaces. However, modules are very different to form spaces - you can think of them as the separate products deployed in Manager.

Next, learn how to view form spaces.

View Form Spaces

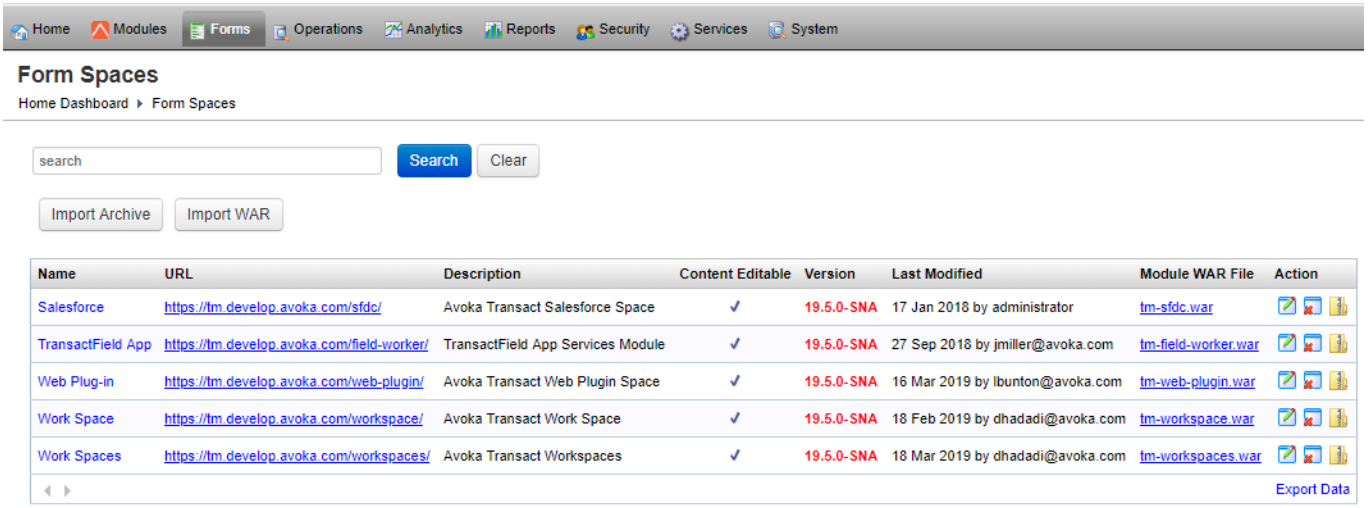
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view the list of available form spaces, so you can administer and manage them.

> You must be an administrator to perform some of these operations.

To view form spaces, select **Forms > Form Spaces.**

Form spaces are shown page by page, with the maximum number of records per page configured in the users preferences.



To search for specific form spaces, specify a name or a part of the name and click **Search.**

Click **Clear** to reset the search criteria.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

> A red version number indicates that the form space version doesn't match the Manager version, so you might need to rebuild the form space.

You can also perform the following actions:

- Configure a form space
- Remove a form space
- Export a form space
- Import a form space

Next, learn how to configure a form space.

Create a Form Space

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager allows administrators to create form spaces using the Journey Manager installer, which also performs string substitutions on the context path and form space name into the portal files. Each WAR file must have a unique context path per Manager environment, such as DEV, UAT, or PROD.

> It's not recommended to create a copy of an existing form space WAR file as it can result in deployment issues, if all the required substitutions are not done. If the form space copy has resulted in two WAR files with the same context, the J2EE application server will only deploy one WAR file.

To create a form space:

1. Run the Journey Manager installer - **`Avoka-TM-Setup.jar`** file.
2. Select **Create a Transact Work Space** and click **Next**.
3. Provide a unique form space name in the **Space Name** field, which Manager uses as the primary key when the form space is imported.

   > Do not use the ampersand (&) character in the form space name.

4. Type a unique context path in the **Context Path** field, which is a part of the URL as per the following pattern: **`https://{Server Domain Name}/{context path}`**. It's the base path to access the Web Application Archive WAR file. If the URL is **`https://creativewave.transactcentral.com/cwf`**, the context path will be **`cwf`**. In this case, the Manager URL will be **`https://creativewave.transactcentral.com/manager`**.

   > The context path must be unique per environment. Do not use the ampersand (&) character in the context path.

5. Provide a meaningful description in the **Description** field and click **Next**.
6. Select the **Production servers MUST use secure cookies** checkbox. It is selected by default. You can clear this setting for a test form space to run on your own PC. Do not configure test form spaces for clients.
7. Select the **Groovy Services** checkbox. It is selected by default. You can clear this setting for a test form space to run on your own PC.
8. Click **Next**.
9. Specify a directory where the installer creates a form space in the **Installation Directory** field. Default is the **`C:\avoka-tm-custom-work-space`** directory on a Windows PC.

If you are creating several form spaces, it's recommended to create a unique installation directory, for example, **C:\avoka-tm-custom-work-space-cwf**.

10. Click **Next**.
11. Click **Install** to create the form space WAR file in the specified directory.
12. Locate the form space WAR file in the **deploy** directory created in your installation directory, for example, **C:\avoka-tm-custom-work-space-cwf\deploy\avoke-sf-portal-cwf.war**.
13. Open the **readmy.html** file, located in the installation directory, for example, **C:\avoka-tm-custom-work-space-cwf\readme.html**, to check how to use the form space.
14. Import the form space into Manager.

Next, learn how to configure form space properties.

Configure a Form Space

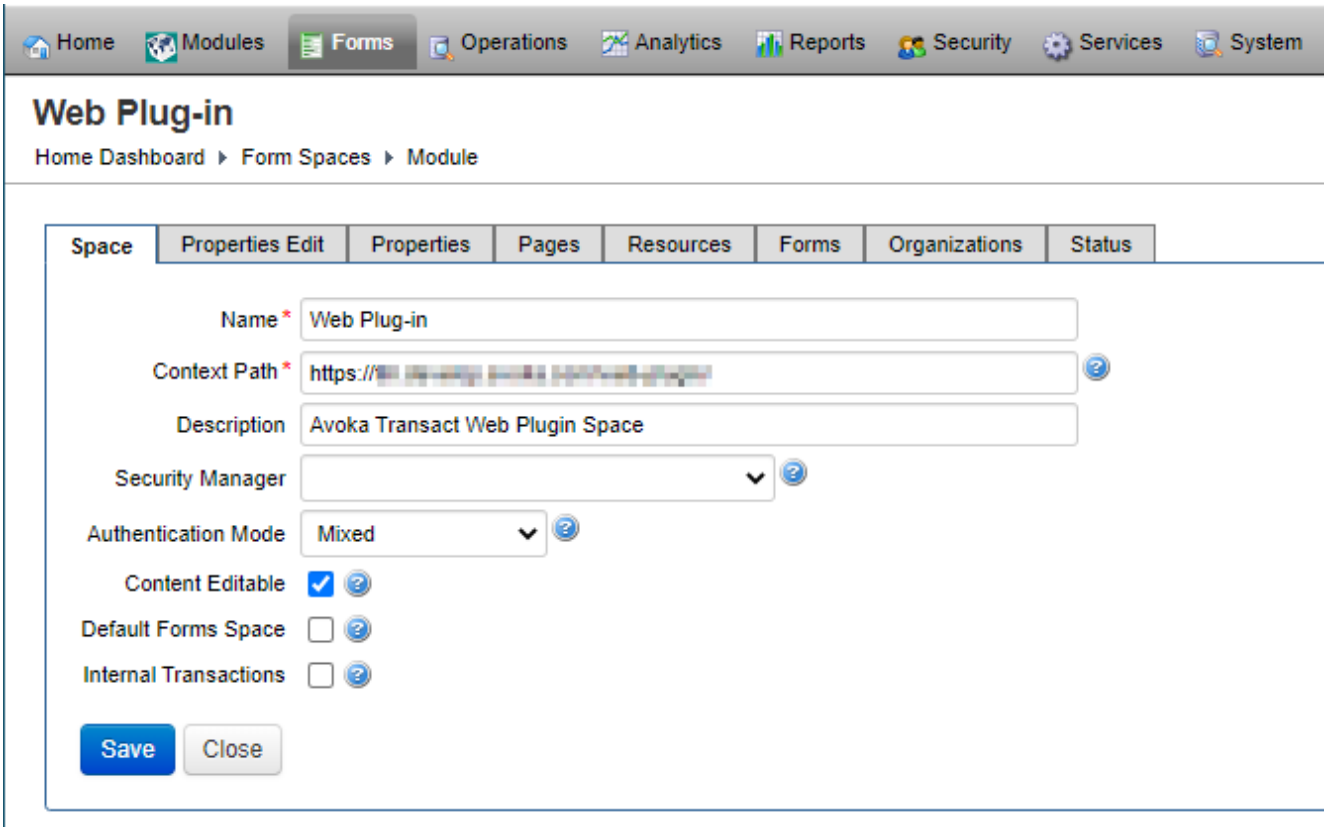
Journey Manager (JM) | System Manager / DevOps | 18.11

Manager allows you to configure form spaces according to your specific requirements, including **2 Factor Authentication**[1] that significantly improves a form space security.

> You must have an administrator role assigned to your account to perform this configuration.

To configure a form space:

1. Select **Forms > Form Spaces.**
2. Locate a form space and click **Edit.**
3. Select the **Space** tab.



[1]Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.

4. Edit the name of the form space in the **Name** field.
5. Provide the web application's **fully qualified domain name**[1] in the **Context Path** field, using the following format: **`http://hostname.com/contextPath/servletPath/pathInfo`**, for example, **`https://elearning.avoka.com/feedback/`**.

   When a Manager server starts up, a context path is created for each deployed form space unless an entry with the same form space name or context path already exists.

6. Change the context path to allow **Apache**[2] mappings. This should be the URL under which the form space is accessible to users. This context path is used during a form submission. If the URL does not match the form space context path (including the port number), the submission will not be accepted.

```
ProxyPassMatch ^/cwf(.*)$ ajp://elearning.avoka.com:9009/feedback$1
ᵖroxyPassReverse /cwf ajp://elearning.avoka.com:9009/feedback
```

7. Provide a brief description of the types of forms that is hosted on this form space in the **Description** fie.
8. Select a **security manager**[3] to authenticate user login from the **Security Manager** dropdown list. Different security managers allow you to implement various user authentication flows, for example, **2 Factor Authentication**[4].

   You can leave this field blank but assign this form space to a required security manager. For more

---

[1]Fully qualified domain name (FQDN), sometimes also referred to as an absolute domain name, is a domain name that specifies its exact location in the tree hierarchy of the Domain Name System (DNS). It specifies all domain levels, including at least a second-level domain and a top-level domain.

[2]Apache HTTP Server is free and open-source cross-platform web server software, released under the terms of Apache License 2.0. Apache is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation.

[3]Security managers are the Journey Manager's components that are responsible for User Authentication, User Authorization, User Self-Registration, and Session Management. Security mangers are designed to handle various sets of users requiring access to different form spaces and can be associated with one or more form spaces. Users can be external or internal, and they can require access to form spaces or a combination of form spaces and modules in different environments. You can find security managers under Security > Security Managers.

[4]Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.

information, see the **Configure a Security Manager** and **Assign Form Spaces to a Security Manager** chapters of security managers documentation.

9. Select the level of authentication that is required to access the form space from the **Authentication Mode** dropdown list:
   - **Mixed** refers to both authenticated and anonymous users being able to access forms in the form space.
   - **Authenticated Only** refers to only authenticated users being able to access forms in the space. An authenticated user is one that has entered a username and password that has been accepted by the system.
   - **Anonymous Only** refers to only anonymous users being able to access forms in the space. An anonymous user is one that has not provided a username and password to access the form.

   Forms configured with a different authentication mode to the authentication configured in the form space will be highlighted with a warning message.

10. Select the **Content Editable** checkbox to allow an administrator to edit all fields in space's content at runtime. Otherwise, fields are read-only and space content cannot be changed.

    Changes made to the **Content Editable** checkbox can only be applied once you clicked the Save button.

11. Select the **Default Form Space** checkbox to make this form space the default space to render forms.

    You can use this setting for testing forms in the Manager console.

12. Select the **Internal Transactions** checkbox to specify whether this form space is licensed for **Internal** (business users) transactions as well as **External** (customer) transactions. If not selected, this form space is licensed for external transactions only.

    You can change the type of licensing transactions of each form. | 18.11

13. Click **Save** to update the changes.

    You can configure form space's properties in the Properties Edit tab.

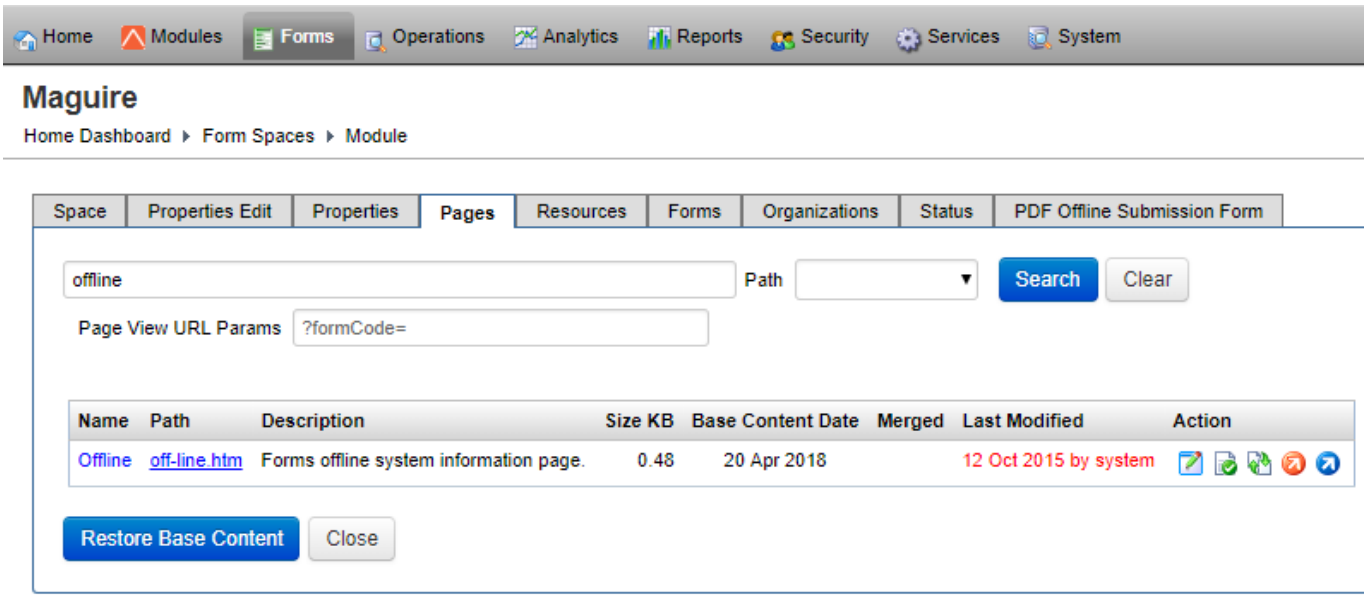
Next, learn how to configure form space properties.

Configure a Form Space Status

Journey Manager (JM) | System Manager / DevOps | 22.04

Manager allows you to restrict access to a form space by changing its status. You can use this functionality to disable a form space when it's not required any more, rather than removing it.

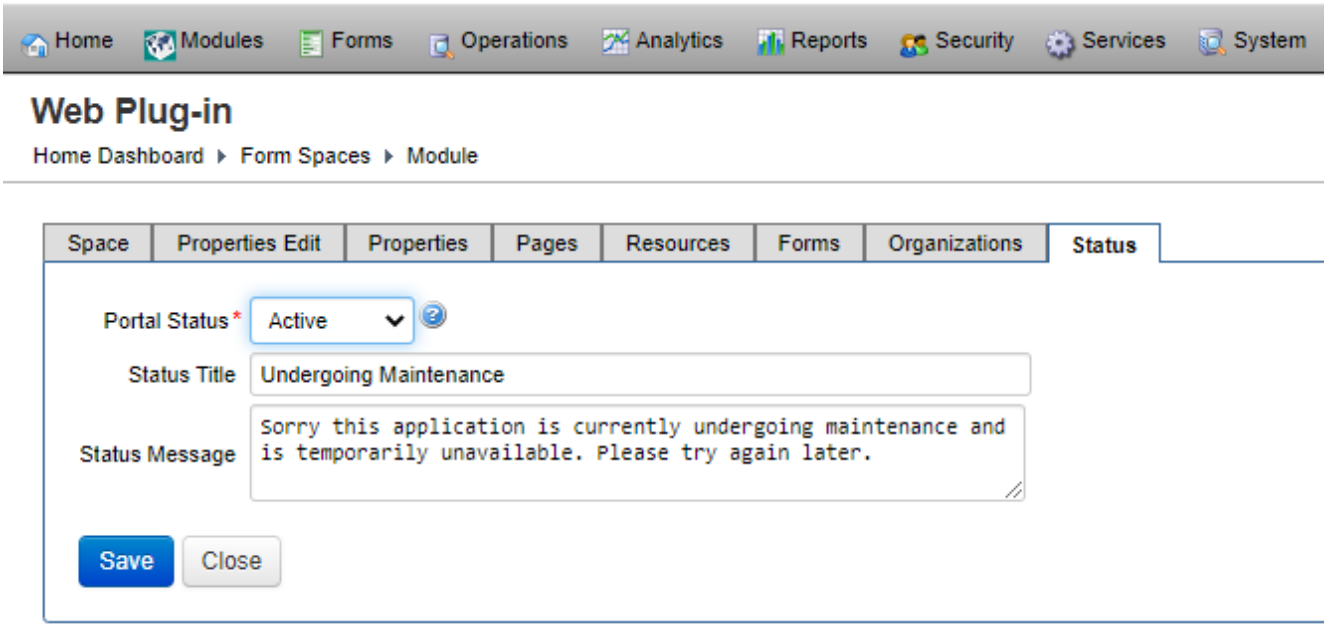
All requests made to any content of an unavailable form space are redirected to the Offline space page as shown below.



Since Manager 22.04, you don't need to do this form space status change manually on a day you want the status to change. Instead, you can schedule the status change using Notifications and Status Change. This is handy for automated deployments and maintenance tasks. Additionally, you can change the status of forms and form versions, which gives you extra flexibility and also reduce the scope of your application unavailability, as compared to applying the status change to the whole form space.

To configure the form space status:

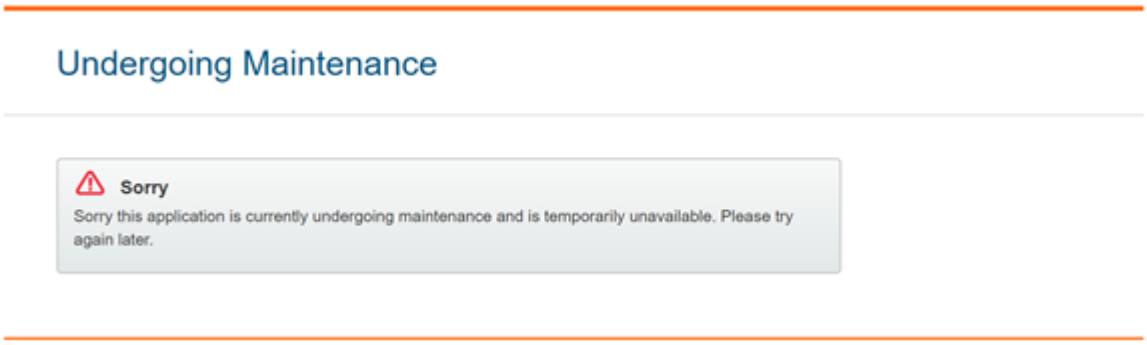
1. Select **Forms > Form Spaces.**
2. Locate a form space and click **Edit.**
3. Select the **Status** tab.



4. Select a status of the form space from the **Portal Status** dropdown list. A form space status can be one of the following: | 21.11

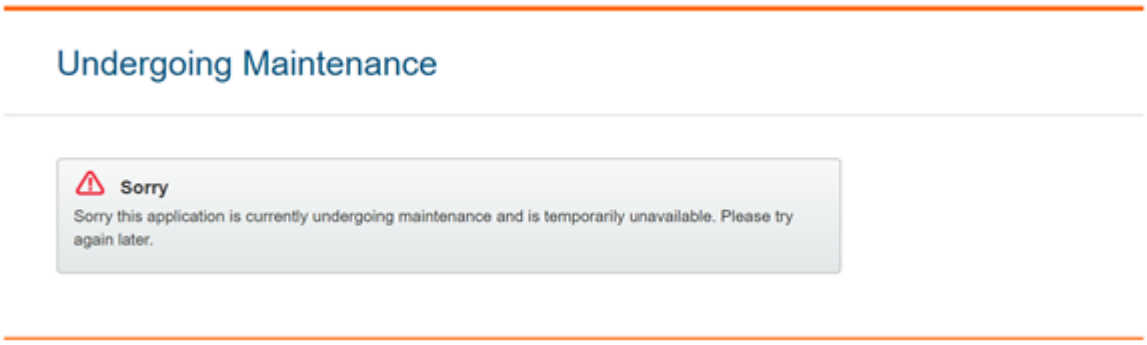
- Active - it is in use and readily available to all users. All functionality is allowed.
- Offline - it can't be used at all. No functionality is available, including DDS, Delivery and so on
- Inactive - similar to Offline.
- NoRender - it doesn't allow for new or saved and resumed forms, but ongoing open sessions can be completed. DDS, submits, saves, and delivery are allowed.

The example of an offline form space is shown below.



5. Edit a page heading that is shown on the page in the **Status Title** field.
6. Edit a message that is displayed on the page in the **Status Message** field. This is only relevant when the space is in offline status. Each status can have a message configured to explain why the form space is restricted. Once the form space is back to an Active status, all content is accessible.

The example of an offline form space message is shown below.



Form requests can be better handled by setting up a status message specific to the form. To do that, add an **Offline Status Message** property to the form. This message overrides the status message when form space users attempt to access a page specific to the form.

7. Click **Save** to update the changes.

Next, learn how to export a form space.

Assign Organizations to a Form Space

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to assign organizations to a form space. All forms that will be accessible to users must be associated with at least one form space. The form spaces available to a form are determined by the form spaces associated with the organization.

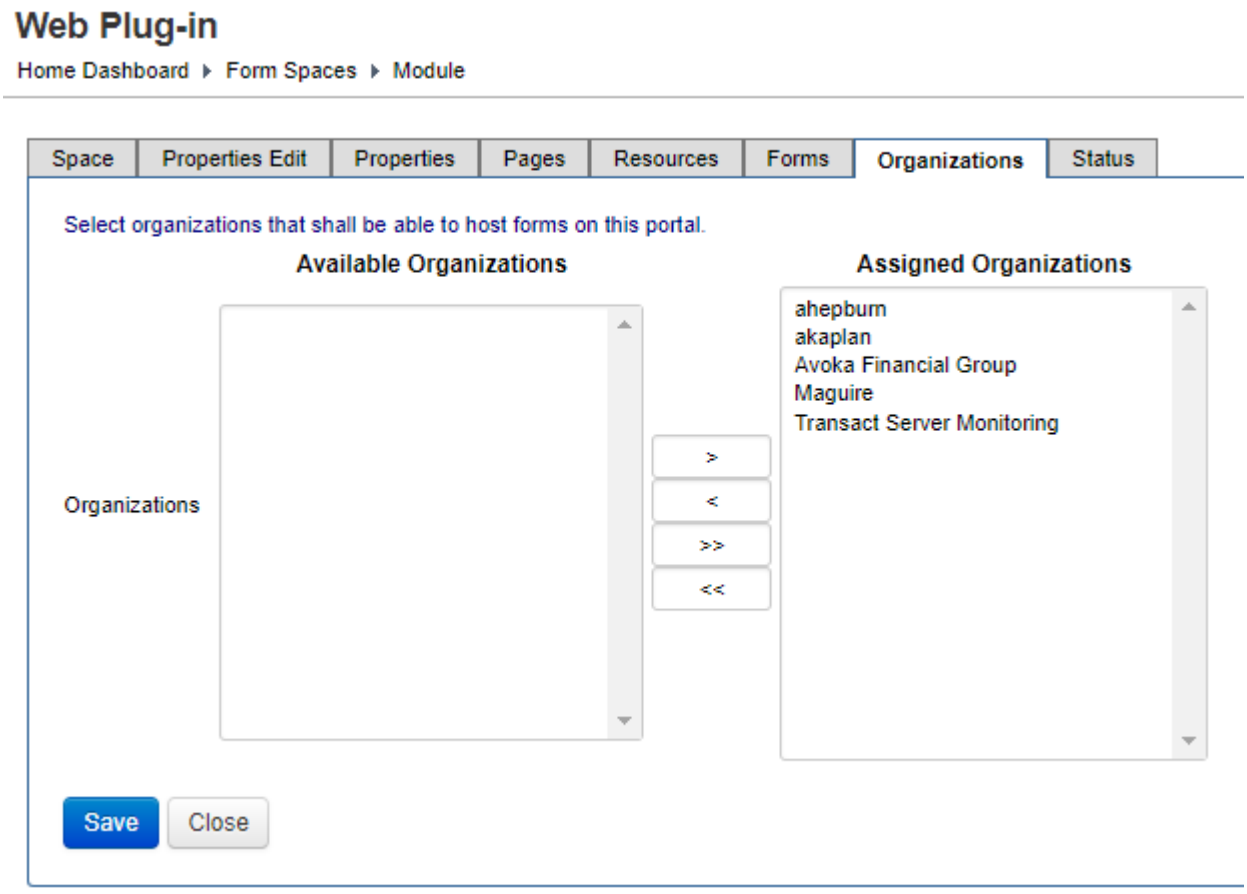
The Web Plug-In space is the default space, used for building form versions from Maestro. It is required for all users and organizations working with Maestro forms. If the Web Plug-In form space is not assigned, you will receive an error message when building the form version.

To assign organizations to a selected form space:

1. Select **Forms > Form Spaces**.
2. Locate a form space and click the **Edit**.

> The form space must be editable.

3. Select the **Organizations** tab to see all assigned organizations.



4. Select one or more organizations from the **Available Organizations** that you want to assign to a form space
5. Click the **>** to move the selected organization from the **Available Organizations** to the **Assigned Organizations**.
6. Click the **<** to move the selected organization from the **Assigned Organizations** to the **Available Organizations**.
7. Click the **>>** to assign all organizations.
8. Click the **<<** to revoke all organizations.
9. Click **Save** to update the changes.

You can also assign form spaces to an organization.

Associating form spaces with organizations may require changes to your currently configured system when using Maestro 5.1 and higher.

Next, learn how to assign form spaces.

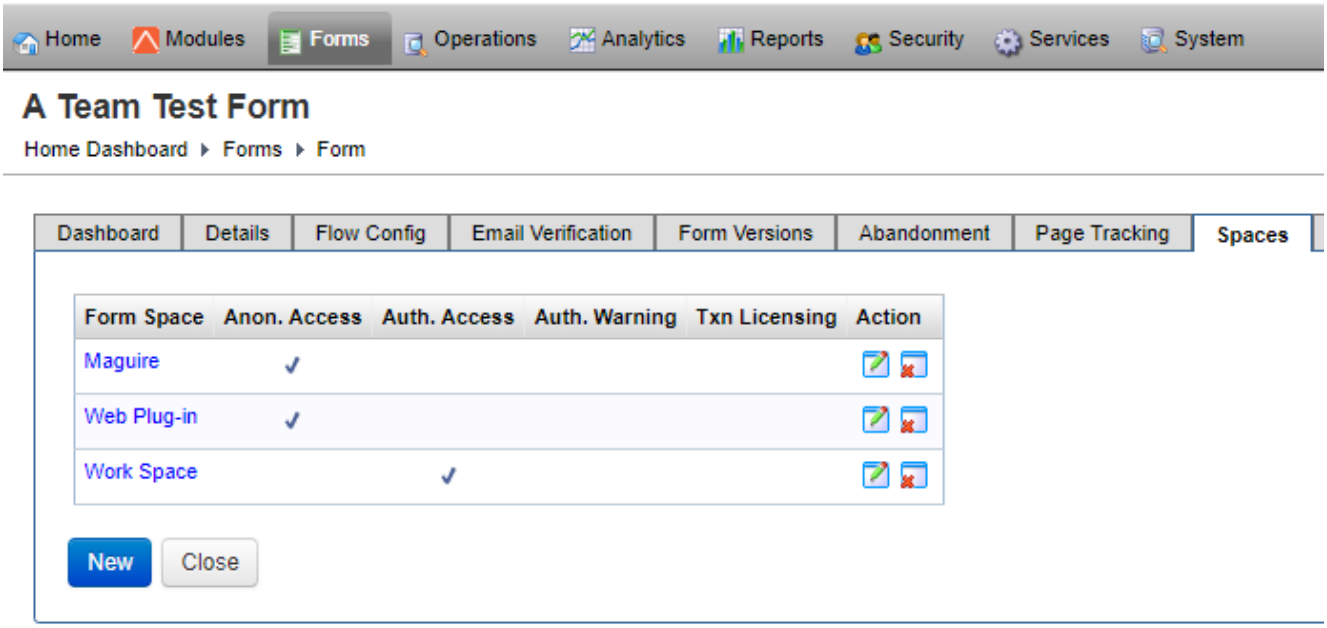
Assign Form Spaces to a Form

Journey Manager (JM) | Platform Developer | 18.11

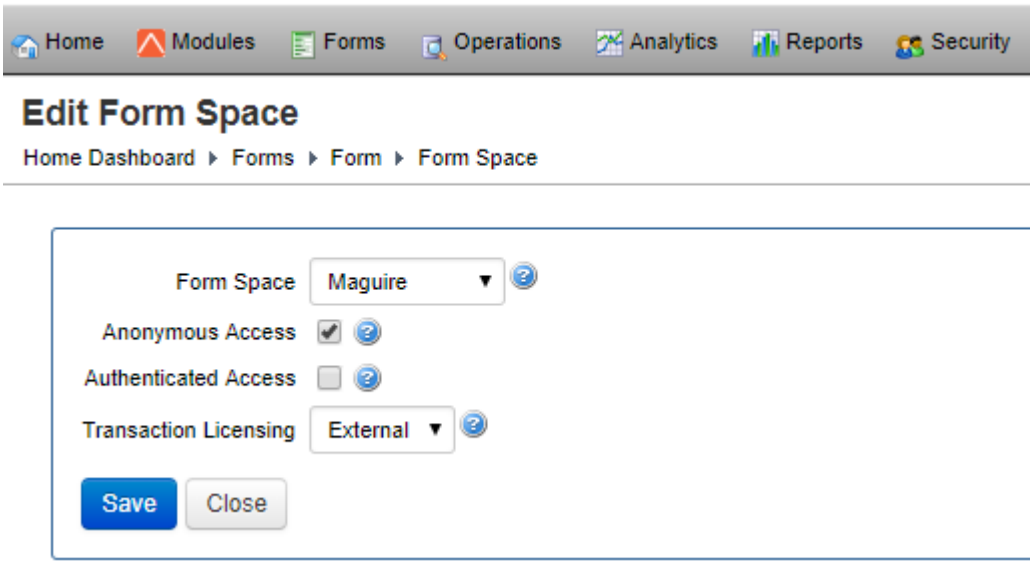
Manager allows you to assign form spaces to a form and configure user access to the form.

To assign a form space to a form:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select the **Spaces** tab to see the current form spaces assigned to this form and their configuration.



4. Click **New** to assign a form space.



5. Select a form space from the **Form Space** dropdown list.
6. If you want anonymous users to access the form space, enable **Anonymous Access**.
7. If you only want authenticated users to access the form space, enable **Authenticated Access**.

> You can select both the **Anonymous Access** and **Authenticated Access**. However, if the access authorized at the form space level is different to that one set at a form level, a warning message will be shown in the **Auth Warning** column for the selected space.

8. Select a transaction licensing type from the **Transaction Licensing** dropdown list to specify whether this form in this form space is licensed for **Internal** (business stuff) or **External** (customer) transactions. Leave this field blank to use the form space's **Internal Transaction** licensing configuration. | 18.11
9. Click **Save** to update the changes.

Next, learn how to assign organizations to a form space.

Remove a Form Space

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to remove a form space when you don't need it any more, for example, when a form space is deprecated. However, it's recommended to disable the unused form spaces rather than removing them.

> All forms associated with a form space will not render once the form space is removed.

To remove a form space:

1. Select **Forms > Form Spaces**.
2. Locate a form space.
3. Find all forms associated with this form space and assign them to a different form space to avoid any form rendering issues.
4. Click **Remove.**
5. Click **OK** to confirm it.

> To create a new form space, import it as a WAR file.

Next, learn how view form space.

Export Form Spaces

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to export form spaces as ZIP or WAR archive files. The main difference between these exports is that a ZIP archive contains only configurations related to a specific form, whilst a WAR archive contains all form space configurations. You can use the export form space functionality to:

- Backup or archive form spaces.
- Re-create or migrate form spaces in another Manager.

To export a form space as a ZIP archive:

1. Select **Forms > Form Spaces**.
2. Click **Export Archive** to export the form space as a form space archive ZIP file, which will be downloaded to your PC.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to import form spaces.

Import Form Spaces

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

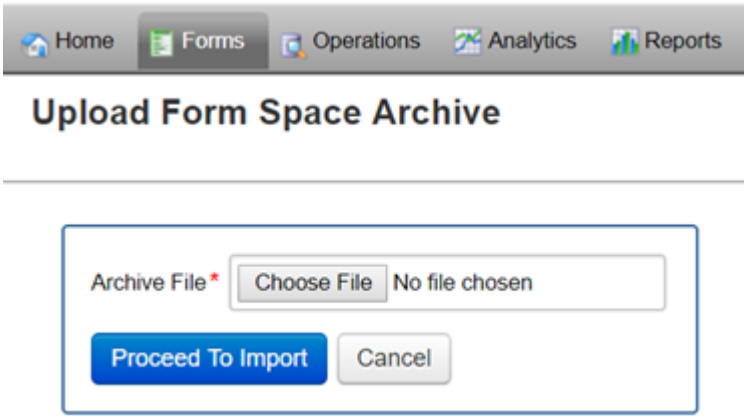
You can use the import form spaces functionality to:

- Migrate one or several form spaces.
- Re-create one or several form spaces in another Manager.
- Promote one or several form spaces to other environments.

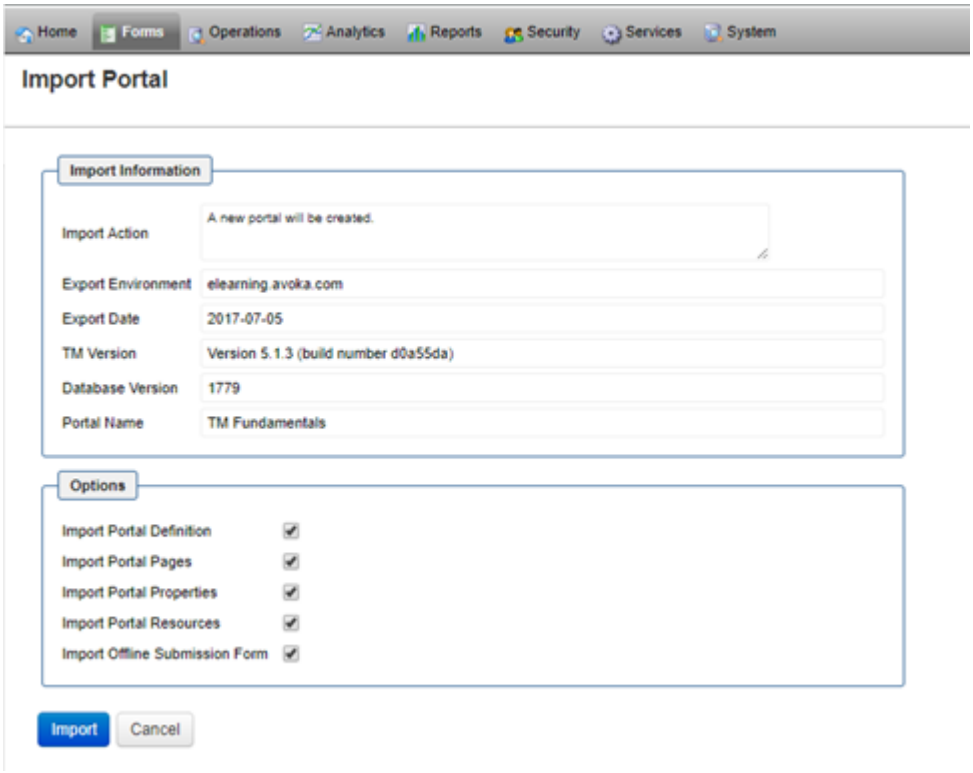
Manager allows you to import form spaces as ZIP or WAR archive files. The main difference between the imports is a ZIP archive contains only form's pages related configuration, whilst a WAR archive contains the whole form space configuration.

To import one or several form spaces as ZIP archives:

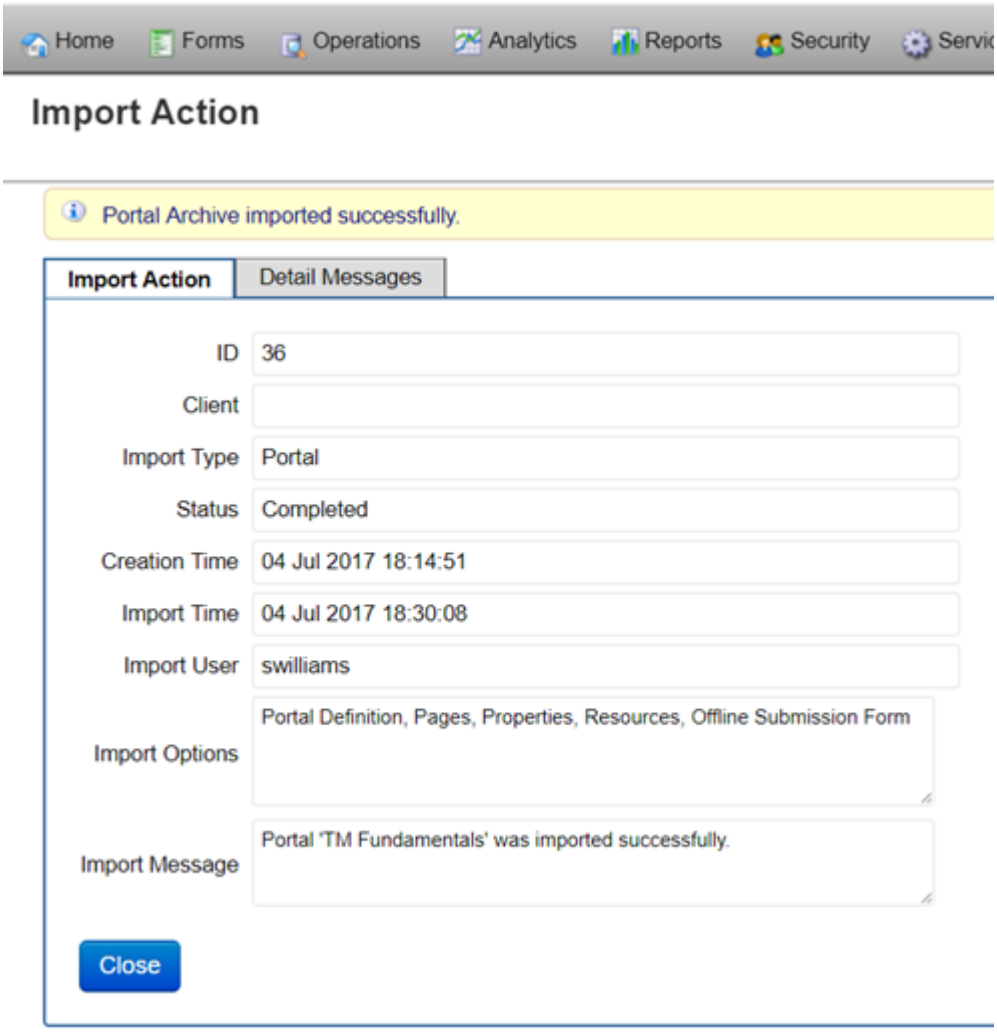
1. Select **Forms > Form Spaces**.
2. Click **Import Archive**.



3. Click **Choose File** and choose a form space archive ZIP file, which has been previously exported.
4. Click **Proceed To Import**.
5. Review the **Import Options**.



6. Select **Import Portal Pages** to import pages associated with this form space.
7. Select **Import Portal Resources** to import resources associated with this form space.
8. Select **Import Portal Properties** to import properties associated with this form space.
9. Select **Import Offline Submission Form** to import offline submission forms associated with this form space.
10. Click **Import** to complete the process.



11. Click **Close** and check the list of form spaces to verify the imported form spaces have been added.

Next, learn how to export form spaces.

Migrate Form Spaces

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides a mechanism to migrate form spaces from one environment to another. This allows you to develop and test form spaces in different environments before deploying them into production. You can also use this process to back up your form spaces before major changes, such as a Manager software upgrade, so you will be able to restore them in case of any unforeseen errors.

Migrating a form space doesn't migrate forms associated with it. It does, however, migrate the following resources:

- Pages
- Resources
- Properties

To migrate one or several form spaces, follow the process below:

1. Export form spaces.
2. Import form spaces to another Manager environment.

Manager allows you to migrate form spaces as ZIP or WAR archive files. The main difference between ZIP and WAR files is that a ZIP archive contains only the form's pages related configuration, whilst a WAR archive contains the whole form space configuration. This is illustrated below.

A ZIP archive has:

```
osf/
pages/
portal-config.xml
resources/
```

A WAR archive has:

```
account-pending.htm             login.htm
border-dummy-content.htm        logout.htm
border-template.htm             META-INF/
card-cvv-info.htm               not-authorized.htm
card-payment.htm                not-supported.htm
click/                          off-line.htm
confirmation.htm                receipt-challenge.htm
create-account.htm              redirect.jsp
error.htm                       resources/
error-500.jsp                   save-challenge.htm
forgotten-password.htm          secure/
form.htm                        server-busy.htm
form-attachments.htm            session-expired.htm
form-expired.htm                submission-cancelled.htm
form-not-found.htm              submission-email-verification.htm
hosted-card-payment.htm         submission-errors.htm
hosted-payment.htm              submission-failed.htm
incompatible-adobe-reader.htm   submission-saved.htm
index.htm                       terms-and-conditions.htm
job-challenge.htm               WEB-INF/
job-tasks.htm                   wet-signature.htm
landing.htm
```

You can use the same process to perform a form space rollback.

Next, learn about form space hot deployment.

Export a Form Space as a WAR Archive

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to export form spaces as WAR or ZIP archive files.

To export a form space as a WAR archive, follow the process below:

1. Select **Forms > Form Spaces**.
2. Locate a form space.
3. Click **Module WAR File**.
4. Manager exports the form space into a form space archive WAR file, which will be downloaded to your PC.

Next, learn how to import a form space as a WAR archive.

Import a Form Space as a WAR archive

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

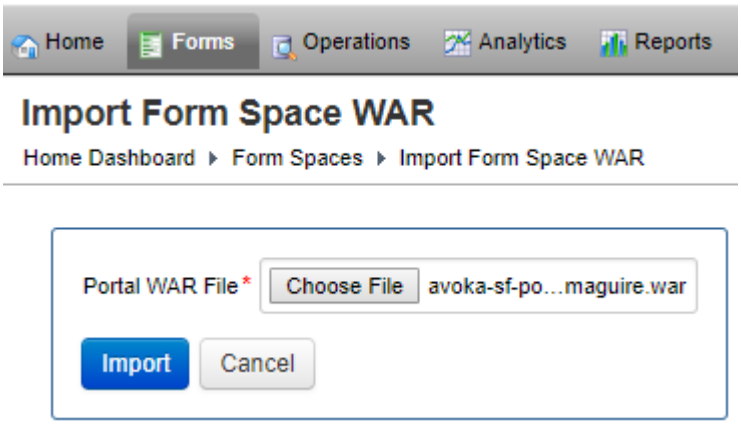
Manager allows you to import form spaces as WAR or ZIP archive files, for example, Maestro, as well as perform a form space hot deployment.

> Manager has permissions for user's export and import of a form space configuration as a ZIP archive, but not the WAR files. Therefore, for a user to import a form space archive, they need the Import Action Edit permission.

To import one or several form spaces as WAR archives:

1. Select **Forms > Form Spaces**.
2. Click **Import WAR**.

> The **Import WAR** button is only available on a single node system. If you have multiple server nodes, this button is grayed out and you will have to deploy a WAR file using a SSH session on your server.



3. Click **Choose File** and locate a form space archive WAR file, which has been previously exported.
4. Click **Import**. It might take 5-10 seconds for the import to work.
5. Check that the success message **"WAR file deployed, please wait a little while it is initializing."** is displayed on the top of the page.
6. Press **Close** and check the list of form spaces to verify the imported form spaces have been added.

> If you don't see the imported form spaces in the list, it could mean there is a problem with your user account configuration. Manager prior to version 4.3.0 allows an organization to be assigned for users with the **Global Access**. This is an issue because you can only see the form spaces that belong to the

assigned organization. When a new form space is created, it doesn't have an organization assigned and, hence, it may be hidden from you.

You might also need to restart a Manager server for the change to take effect.

You can also import a WAR file into Manager by copying it directly to a Manager server, if you have read and write access to the **`deployments`** directory.

To copy one or several form spaces as WAR archives:

1. Log onto the Manager server.
2. Copy the form space WAR file to the **`/avoka/transact/manager/server/standalone/deployments`** directory.
3. Log onto Manager.
4. Check the new form spaces have been added.

Next, learn how to export a form space as a WAR archive.

Standard Form Space Properties

Journey Manager (JM) | System Manager / DevOps | 21.11

Each form space has a number of predefined properties that allow you to customize a form space without any extra development efforts. For example, you can easily improve form space access security by using the CORs headers. Manager allows you to edit properties, if additional customization is required.

Some properties are straight-forward but some require more understanding, so they are listed below.

Cache Headers

The cache HTTP header value to set if defined.

```
Pragma: no-cache
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0, private
Expires: 1
```

CORS Headers

The **Cross-Origin-Resource-Sharing**[1] (CORS) security HTTP header values allow you to set the Access-Control-Allow-Origin header on the API calls to enable applications such as **Open UX**[2] to be able to support the calls. The **`CORSApprovedOrigins`** list defines the approved origins, which will be validated against the origin on a request and, if it matches the **`${request-origin}`**, it will be replaced with the provided origin. | 19.05

The example of CORS configuration is shown below:

[1]Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin. A web application executes a cross-origin HTTP request when it requests a resource that has a different origin (domain, protocol, and port) than its own origin.

[2]Open UX is the open API for development of financial customer journeys on the Temenos Journey platform. Teams skilled in use of React, Angular or other Javascript frameworks will find this the fastest path to developing production class applications.

```
{
        "ApplyCORSHeaders":"true",
        "CORSApprovedOrigins":"test-cors.org,avoka.com,temenos.com",
        "ExcludedCORSPaths":"/secure/servlet,/app,/secure/app/cache",
        "CORS_HEADERS":{
                "Access-Control-Allow-Origin":"${request-origin}",
                "Access-Control-Allow-Methods":"POST, PUT, GET, DELETE",
                "Access-Control-Allow-Credentials":"true",
                "Access-Control-Max-Age":"86400"
        }
}
```

For more information on CORS and how it works, see https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

Security Headers

The **Content Security Policy**[1] (CSP) security HTTP header improves overall forms' security. It is applied to all form's resources that are passed via each form space. If it is not set, the Form CSP Header Value parameter of the Form Submission Access Controller server is used as the default. | 18.11

The CSP response header helps you reduce XSS risks on modern browsers by declaring what dynamic resources are allowed to load via a HTTP Header, thereby protecting your solution from a range of vulnerability attacks.

The example of the CSP response header is shown below:

```
Content-Security-Policy: script-src 'self' 'unsafe-eval'; object-src 'none'; connect-src
'self'
X-Frame-Opticns: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Prctection: 1; mode=block
```

By default, Manager has strict CSP settings configured. Occasionally, you need to include a capability or 3rd party service in your forms that require changes to the default CSP settings. You can configure the following settings in Manager in order to allow your solution to operate as expected yet still maintain a strong CSP configuration.

---

[1]Content Security Policy (CSP) is a computer security standard introduced to prevent cross-site scripting (XSS), clickjacking and other code injection attacks resulting from execution of malicious content in the trusted web page context. CSP provides a standard method for website owners to declare approved origins of content that browsers should be allowed to load on that website—covered types are JavaScript, CSS, HTML frames, web workers, fonts, images, embeddable objects such as Java applets, ActiveX, audio and video files, and other HTML5 features. For more information, see https://www.w3.org/TR/CSP/

System Wide Settings

1. Log in to Manager as a Global Administrator.
2. Select **Services > All Services**.
3. Select **Form Submission Access Controller** from the **Type** dropdown list. There should only be one item of this type.
4. Click the service and select the **Parameters Edit** tab.
5. Modify a CSP in the **Form CSP Header Value** field as required and click **Save**. For more information, see configure service parameters edit.

Organization Level Settings

The Organization CSP header limit is 2 KB. | 21.11

1. Log in to Manager as an Organization Administrator.
2. Select **Forms > Organizations**.
3. Find your organization and select the **Security** tab.
4. Under **Forms Content Security Policy (CSP)**, select the **Override System Defaults** checkbox and click **Save**.
5. Modify the value of **Organization CSP Header** as required and click **Save** . For more information, see configure organization security.

LinkedIn Example

The LinkedIn SDK requires a JavaScript library to be loaded from their remote server. The default CSP settings are usually something like

```
script-src 'self'; object-src 'none'
```

These settings permit scripts to be loaded from the same server, but no other. To permit the remote LinkedIn JavaScript file to load we must modify these setting to allow JavaScript sources from the relevant LinkedIn domains as follows:

```
script-src 'self' https://platform.linkedin.com https://www.linkedin.com 'unsafe-inline';
object-src 'none';
```

Due to the way the LinkedIn integration uses inline JavaScript we need to add the 'unsafe-inline' directive.

Note that when we relax these settings, we are only adding known and trusted domains to the configuration, as opposed to using wildcards which could compromise this security layer.

X-Frame-Options

The **X-Frame-Options**[1] HTTP header improves overall forms' security. It is applied to all form's resources that are passed via each form space. The X-Frame-Options include:

- NONE
- SAMEORIGIN - the page can only be displayed in a frame on the same origin as the page itself.
- ALLOW-FROM URI - the page can only be displayed in a frame on the specified origin, for example www.example.com.
- blank - no X-Frame-Options header will be set.

Next, learn how to configure form space properties.

[1] X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page in a <frame>, <iframe>, <embed> or <object> . Sites can use this to avoid clickjacking attacks, by ensuring that their content is not embedded into other sites.

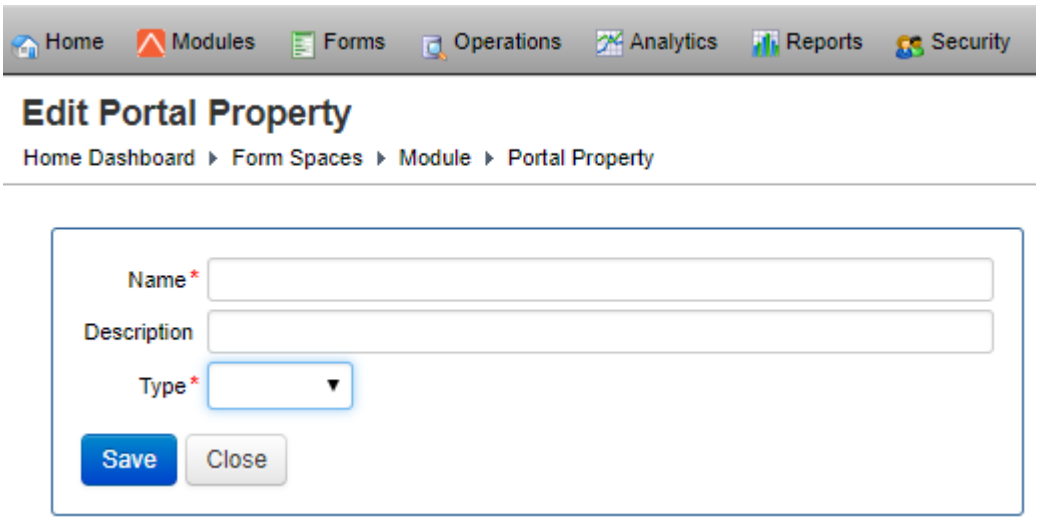
Create Form Space Properties

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Each form space has a set of properties, which are used to customize its business logic. Manager comes with several predefined form space's properties that have their own unique functionality. These properties can be created, configured, or removed in the **Properties** and the **Properties Edit** tabs. Manager also allows you to create new properties for a form space to implement your specific requirements.

To create a new property:

1. Select **Forms > Form Spaces**.
2. Locate a form space and then click **Edit.**
3. Select the **Properties** tab.
4. Click **New**.



5. Enter a property unique name.
6. Provide a property description.
7. Select a property type from the **Type** dropdown list.
8. Click **Save** to update the changes.
9. Check the newly created property in the **Properties Edit** tab.

To remove a property:

1. Select **Forms > Form Spaces**.
2. Locate a form space and then click **Edit.**
3. Select the **Properties** tab and locate a property you don't need any more.
4. Click **Remove.**

Next, learn how to configure form space properties.

Configure Form Space Properties

Journey Manager (JM) | System Manager / DevOps | 21.11

Manager comes with a set of standard form space properties that can be applied to all spaces that are created in this Manager instance. These properties have default values that can be configured via the **Properties** and **Properties Edit** tabs. You can use form spaces properties to override organization's properties. Subsequently, you can create form's properties to override the form space's properties.

Manager automatically sends emails to form users to notify the form has been changed. You can adjust the email format using the settings below.

> Some of form space properties are used to configure Work Space's tabs.

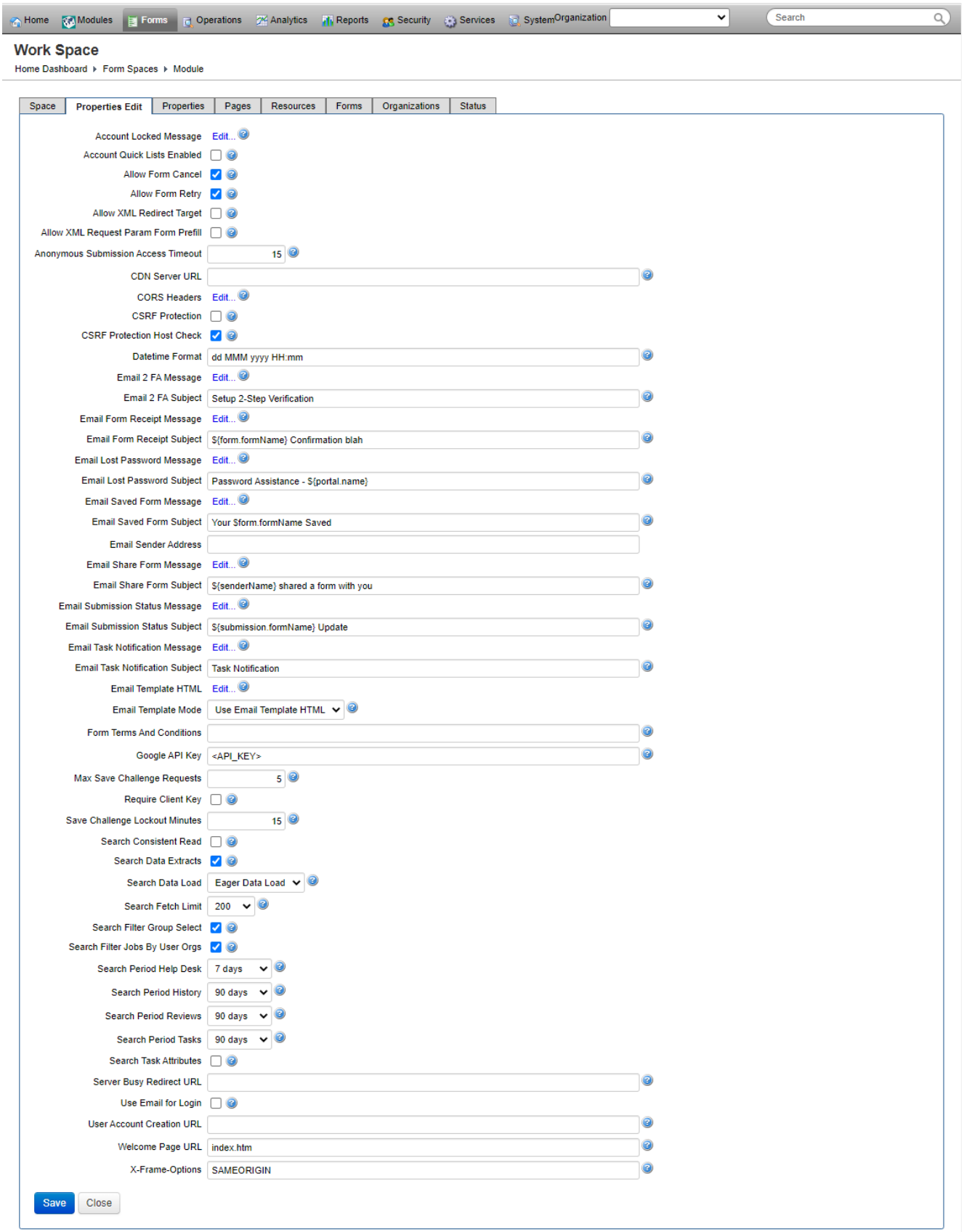
> A form space can be set as not editable, so you can't change its configuration. To enable this, select the **Content Editable** checkbox in the **Space** tab.

To configure the properties in the **Properties** tab:

1. Select **Forms > Form Spaces**.
2. Locate a form space and then click **Edit.**
3. Select the **Properties** tab.
4. Click **Edit** to configure the selected property. Check the standard properties to learn more about their functionality.
5. Edit the value of the property.
6. Click **Save** to update the changes.

To configure form space properties in the **Properties Edit** tab:

1. Select **Forms > Form Spaces**.
2. Locate a form space and click **Edit.**
3. Select the **Properties Edit** tab to view and edit properties. Check the standard properties to learn more about their functionality.



4. Edit **Account Locked Message**, which is a message displayed at login to users whose accounts are locked.

5. Select the **Account Quick Lists Enabled** checkbox to show Account Home Page quick selection lists and Tasks menu count. Note enabling this option can impact performance.
6. Select the **Allow Form Cancel** checkbox to allow users to cancel a submission at the payment stage.
7. Select the **Allow Form Retry** checkbox to allow users to go back to the form if payment has not been completed yet.
8. Select the **Allow XML Redirect Target** checkbox to allow XML form data to specify redirect target.
9. Select the **Allow XML Request Param Form Prefill** checkbox to allow XML request parameters, such as **`xmlData`** and **`schemaSeed`**, to be used in form prefill.
10. Edit a timeout for an anonymous user to access submitted form receipts in the **Anonymous Submission Access Timeout** field. After an anonymous user submits a form, Manager sends them a URL link to download a form receipt. This link will timeout after the Anonymous Submission Access Timeout period. However, if the user accesses the URL link using the same **user agent**[1] (browser) with the same **IP address**[2], this timeout doesn't apply, because Manager re-establishes a session. In this case, the user can access the form receipt multiple times without any time restrictions. The default timeout is 15 min.
11. Edit **CDN Server URL** - Optional CDN Server URL for caching static content.
12. Edit CORS Headers.
13. Select the **CSRF Protection Host Check** checkbox to enable Cross Site Request Forgery (**CSRF**[3]) protection by validating the host name in a request.
14. Select the **CSRF Protection** checkbox to enable CSRF protection by using an the extra token in a request. | 21.11

    You must restart a Manager server or re-deploy a form space for the change to take effect.

15. Edit **Datetime Format** used in the date pickers.
16. Edit **Email 2FA Message** - an email message, which is sent to a user, for **2FA**[4] user account enrolment.
17. Edit **Email 2FA Subject** - an email subject, which is sent to a user, for 2FA user account enrolment.

---

[1]User agent is software (a software agent) that is acting on behalf of a user. One common use of the term refers to a web browser that retrieves, renders and facilitates end user interaction with Web content.

[2]IP address is a unique string of numbers separated by full stops that identifies each computer using the Internet Protocol to communicate over a network.

[3]Cross-site request forgery (CSRF), also known as one-click attack or session riding, is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the web application trusts.

[4]Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.

18. Edit **Email Form Receipt Message** - an email message for form confirmation or receipt email.
19. Edit **Email Form Receipt Subject** - to change an email subject for form confirmation or receipt email.
20. Edit **Email Lost Password Message** - a lost password reset email message.
21. Edit **Email Lost Password Subject** - a lost password reset email subject.
22. Edit **Email Saved Form Message** - an email message for user saved form email.
23. Edit **Email Saved Form Subject** - an email subject for user saved form email.
24. Edit **Email Sender Address** - a sender email address for generated emails, overriding the global email sender address.
25. Edit **Email Share Form Message** - an email message for user shared form email.
26. Edit **Email Share Form Subject** - an email Share Form Subject.
27. Edit **Email Submission Status Message** - an email message for submission processing status update.
28. Edit **Email Submission Status Subject** - an email subject for submission processing status update, for example, `${submission.formName} Update`.
29. Edit **Email Task Notification Message** - an email message for task notification.
30. Edit **Email Task Notification Subject** - an email subject for task notification.
31. Edit **Email Template HTML** - an email border template HTML.
32. Edit **Email Template Mode** - the location of the email HTML border template to use.
33. Edit **Form Terms And Conditions** - the form Terms and Conditions statement the user must accept before accessing the form.
34. Edit **Google API Key** - Google API Services Access Key.
35. Edit **Max Save Challenge Requests**, default 3, which is the maximum number of save challenge requests that can be performed when returning to a saved form before a user is locked out. This maximum is inclusive of any attempts from other sessions for the same submission.
36. Edit **Save Challenge Lockout Minutes**, default 15, which is the duration in minutes a submission is locked out when the maximum number of save challenge requests has been exceeded, as per the **Max Save Challenge Requests** property. If the **Save Challenge Lockout Minutes** is set to 0, then the submission is permanently locked out, which requires an administrator to manually unlock this submission.

    > You can also configure the **Save Challenge Lockout Minutes** property in a form version to override this form space level's property. | 21.05

37. Select the **Search Filter Group Select** checkbox to enable the **Groups** dropdown list in the Work Space Tasks and History tabs. This is the default option.
38. Select the **Search Consistent Read** checkbox to search using database Consistent Read in the Task List, History, Reviews and Help Desk pages.
39. Select the **Search Data Extracts** checkbox to search for the Submission Data extracts in Task List, History, Reviews and Help Desk pages.

40. Select a data load type from the **Search Data Load** dropdown list, such as **Eager Data Load** or **Lazy Data Load**, to apply to certain portal pages (Help Desk). If "Lazy Data Load" is used, the page will not attempt to load result data on initial access, instead waiting for the user to initiate a search. This can help with performance on systems with large transaction volumes. "Eager Data Load" means that the page will automatically load data when initially accessed.
41. Select the search results fetch limit for the Task List, History, Reviews and Help Desk pages from the **Search Fetch Limit** dropdown list. Default is 200.
42. Select the **Search Filter Group Select** checkbox to show the Group Select in the Search Filter on Task, History pages.
43. Select the **Search Filter Jobs by User Org** checkbox to enable the **Job Groups** dropdown list in the Work Space Reviews tab and allows only job coordinators to see collaboration jobs that they are responsible for.
44. Select the default Search Period for the Help Desk pages from the **Search Period Help Desk** dropdown list.
45. Select the default Search Period for the History pages from the **Search Period History** dropdown list.
46. Select the default Search Period for the Reviews pages from the **Search Period Reviews** dropdown list.
47. Select the default Search Period for the Task List pages from the **Search Period Tasks** dropdown list.
48. Select the **Search Task Attributes** checkbox to search for the Task attributes Subject, Message and Address in the Task List and History pages.
49. Edit **Server Busy Redirect URL** - external resource URL to redirect users to if the server is too busy to handle the form request. The default is the **Server Busy** form space page.
50. Select the **Use Email for Login** checkbox to use the email address as the user login name for Local Manager user accounts. It's not selected by default.
51. Edit **User Account Creation URL** to specify an alternative user create account URL.
52. Edit **Welcome Page URL** - the default welcome page URL, such as index.htm.
53. Edit X-Frame-Options.
54. Click **Save** to update the changes.

> If you are using a customized form space, some of these properties may not be implemented.

To create a new form space property in the **Properties Edit** tab:

1. Select **Forms > Form Spaces**.
2. Locate a form space and click **Edit.**
3. Click **New** to create a new property.
4. Provide a property unique name in the **Name** field.
5. Select a property type from the **Type** dropdown list.
6. Click **Save** to update the changes.

Next, learn how to a configure form space email template.

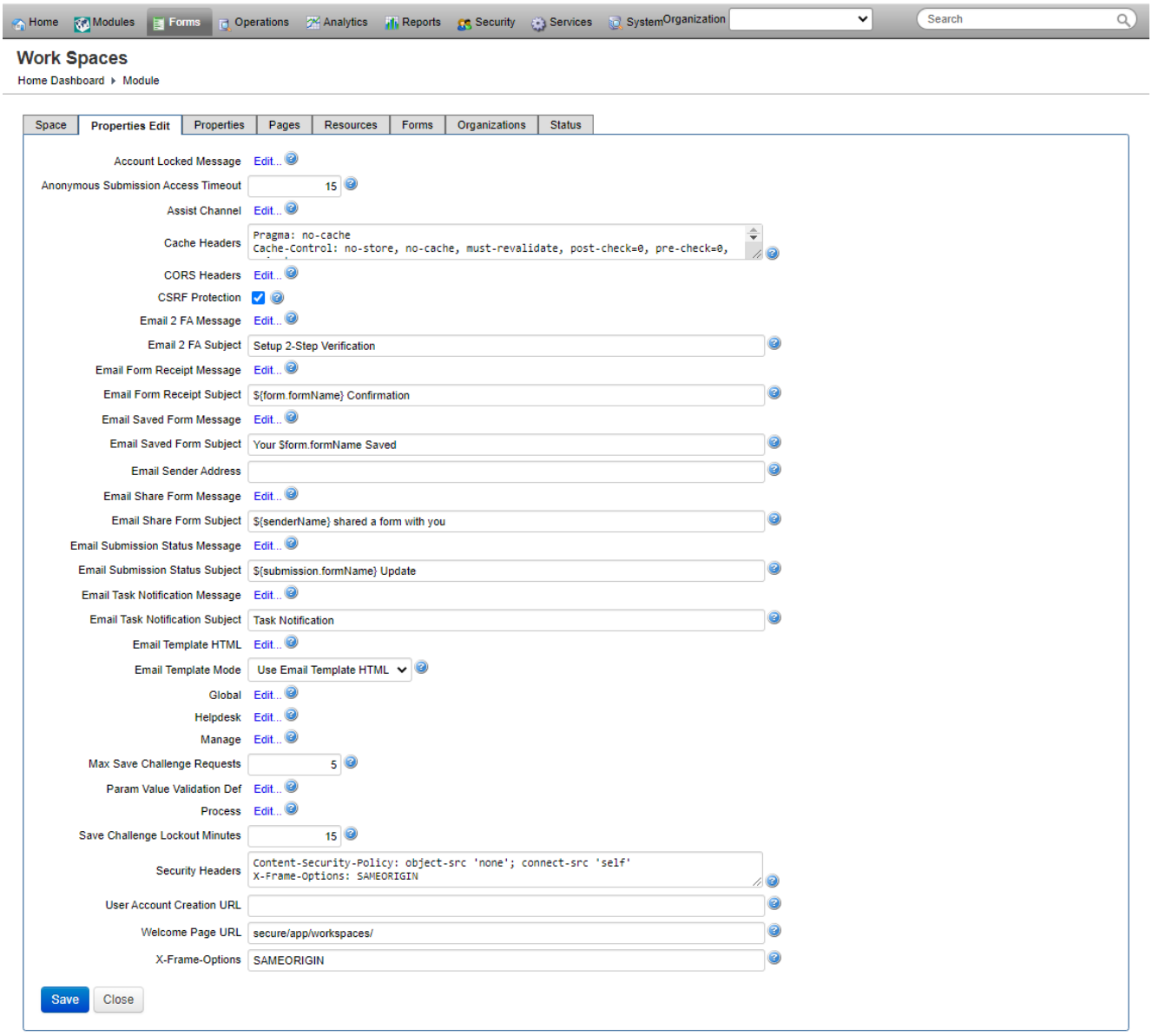
Configure Workspaces Properties

Journey Manager (JM) | System Manager / DevOps | 21.11

Manager allows you to configure specific properties of your Workspaces pages, which are not part of the standard form space properties. You can adjust some of the properties to improve Workspaces data fetch performance that will result in a better user experience.

To configure the Workspaces's properties:

1. Select **Forms > Form Spaces**.
2. Locate the **Workspaces** and click **Edit**.
3. Select the **Properties Edit** tab to view and edit properties as described in configure form space properties.



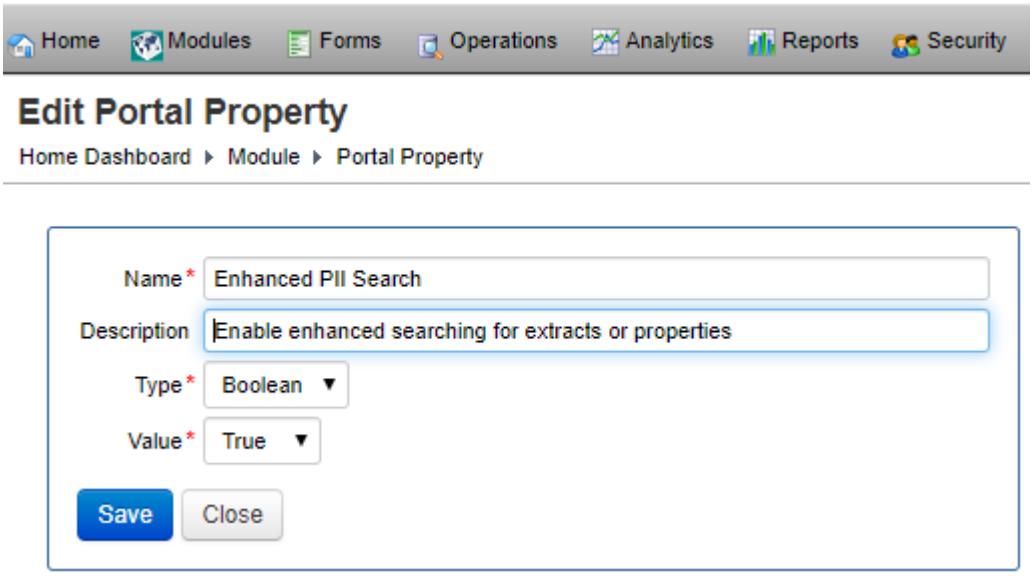
4. Select the **Enhanced PII Search** checkbox to enable enhanced searching for submission extracts or properties, which can significantly increase Workspaces data fetch performance. This option is not created by default, so you may need to create it. | 19.11.3
5. There's an optional search limit, which can be configured in the **Enhanced PII Search Limit** field, to restricts SQL predicate results used within the Enhanced PII Search logic. The default is 999. | 19.11.3

> It is not recommended to change this setting unless you are experiencing slow search queries by any transaction fields or it is advised by Temenos. This option is not created by default, so you must add it as a new property or contact Temenosto do it for you.

6. Click **Save** to update the changes.

To create the **Enhanced PII Search** property:

1. Select **Forms > Form Spaces**.
2. Locate the **Workspaces** and click **Edit**.
3. Create the new Boolean property as shown below:



4. Click **Save** to update the changes.

Next, learn how to a configure form space email template.

Configure a Form Space Email Template

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

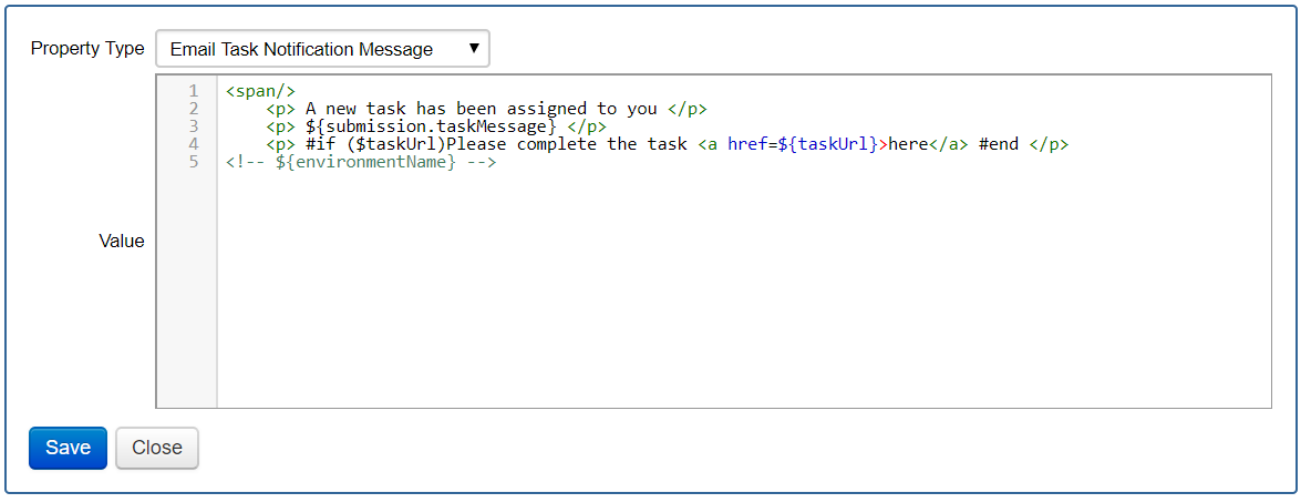
Manager extensively uses emails to send various notifications to form users and administrators. Manager applies email templates to insert form's dynamic content into generated emails. For example, you can have a form that lets a user to select multiple products and, when the form is completed, to send an email with a subject displaying all the products entered in the form. You can achieve this by configuring an email template which has access to a data model from which it canextract product names or iterate over them.

Email templates are created using the Velocity Template Language (VTL) markup, which is one of the simplest and easiest methods of incorporating dynamic data into an HTML document and is made up of References and Directives. References begin with a **$** and are used to retrieve a value, for example, **$a = FirstName**. Directives begin with a **#** and are used to perform an action, for example, **#get**, **#set**, **#end**.

> The email templates can be customized at organization, form space and form levels.

To configure a form space email template:

1. Select **Forms > Form Spaces**.
2. Locate a form space and click **Edit.**
3. Select the Properties tab to see several email templates available.
4. Locate a template and click **Edit**. The common email template is an Email Task Notification Message one shown below.



5. Update the template email body as an HTML text in the **Value** field. In this example, the template describes a task for a recipient to complete. It consists of both the static text, “A new task has been assigned to you”, and dynamic text, **$taskUrl**, which is replaced with the URL when the email is rendered.
6. Click **Save** to update the changes.

Next, learn how to view form space pages.

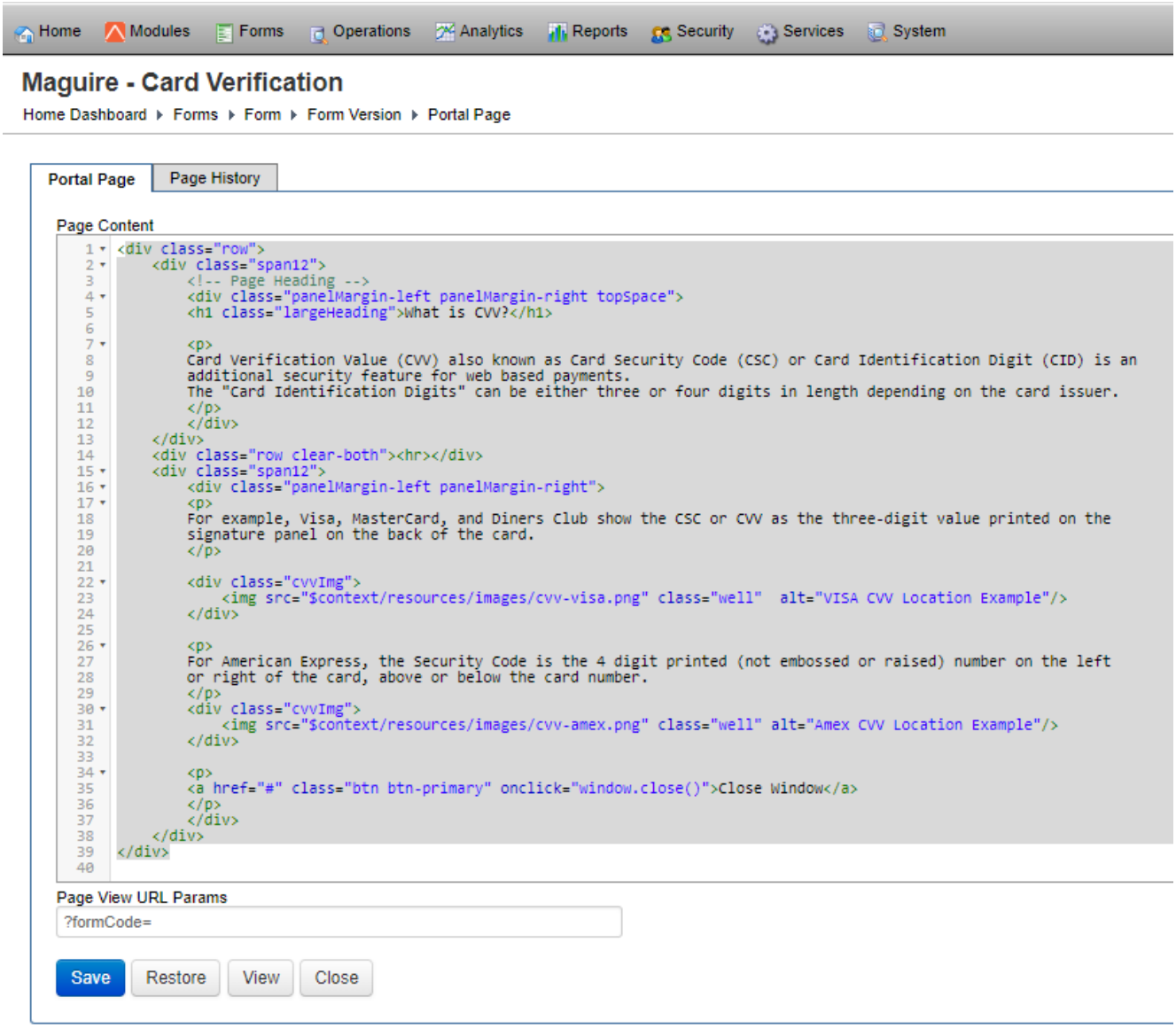
Configure Form Space Pages

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager comes with a range of pre-configured HTML pages that can be displayed to end users as they are or with a minor text editing to customize them to better suit individual needs. Only the HTML elements of these generic pages can be configured. The page behavior and response can't be changed. These pages are configured in form spaces.

To configure a page:

1. Select **Forms > Form Spaces**.
2. Locate a form space and click **Edit**.
3. Select the **Pages** tab.
4. Find the page that you want to edit and then click **Edit**.

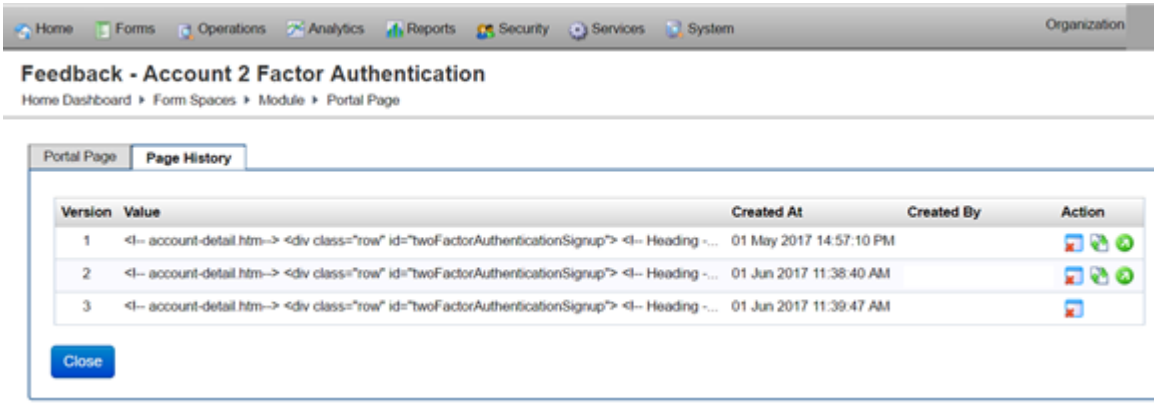


5. Edit the HTML page content. Update the **Page View URL Params** value to change the URL to this page.

> To configure a pages, ensure that the **Content Editable** checkbox is selected.

6. Click **Save** to update the changes.
7. Click **View** to check how the updated page looks. A new version of the page is created each time the page is edited and saved.

To view the history of the page, click the **Page History** tab.



Click **Restore** to cancel the latest changes and restore the page content to its base content.

Click the **Restore Base Content** icon to restore the selected version's content and apply it as the base page content.

Next, learn how view form space resources.

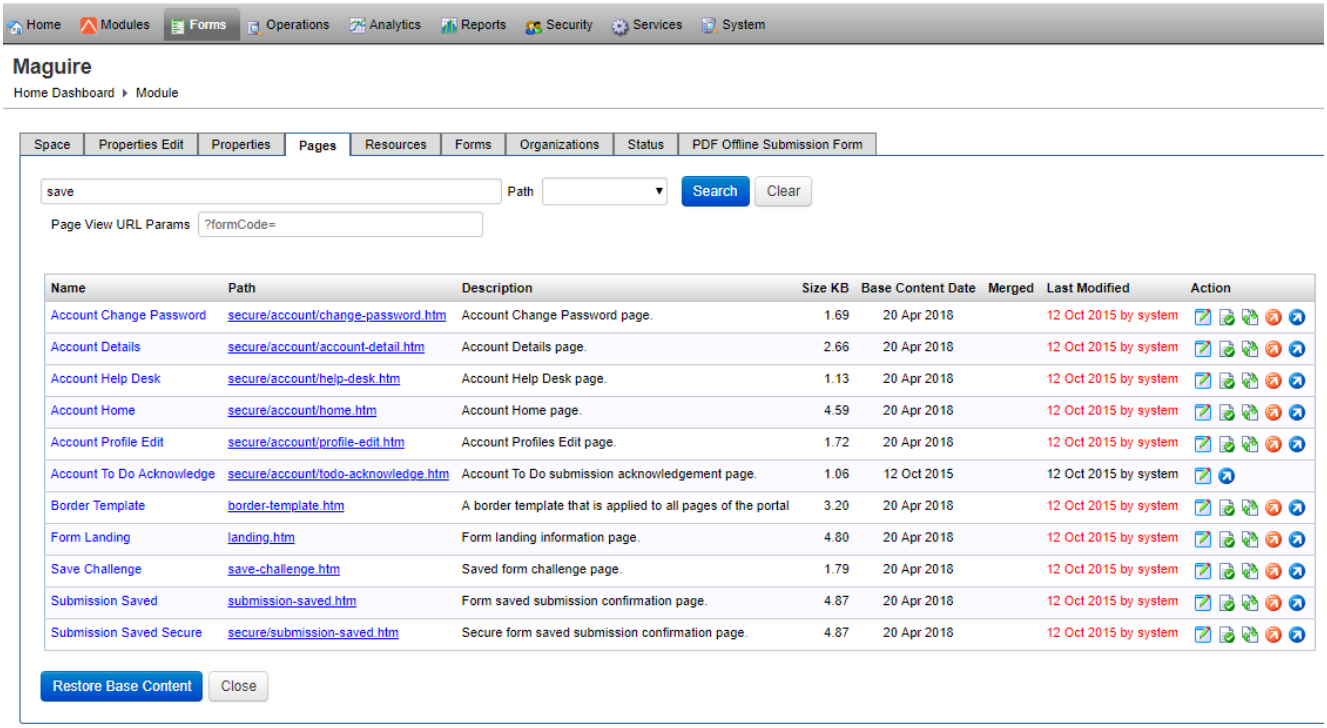
View Form Space Pages

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides a range of generic HTML pages that are configured to be used to support and surround all forms associated with a form space. These pages are not part of the form, but rather are a part of the entire application that represents the form. Common examples are a Forgotten Password page, a Form Not Found page, and a Terms and Conditions page.

To list all pages currently associated with a form space:

1. Select **Forms > Form Spaces**.
2. Locate a form space and click **Edit.**
3. Select the **Pages** tab.



4. To filter or search the form spaces by one or more criteria, specify the following settings and click **Search**:
   - Page name or a part of its name.
   - Path.

   Click **Clear** to reset the search criteria.

   The **Page** tab list pages in alphabetical order and displays the most important information for each page. The Last Modified column identifies if a page has been recently edited. If the page has been modified, the last modified date and the name of the user who modified the page is displayed. If the page's content is different to its base content, the entry is highlighted in red, and additional actions (to view the differences between the current and base content and to restore the base content) are made available.

You can perform one of the following actions on a selected page:

- Click **Edit** to configure a page.
- Click **Restore Base Content** to restore the page content back to the original base content. This removes all latest modifications and returns the content back to its original form. Click **Restore Base Content** at the bottom of the page to restore all pages with their Base Content.
- Click **View Content Diff to Base Page Content** to compare the current modified version of the page with the base version of the page.
- Click **Mark Content Merged** to promote the modified page to the base page. This marks the modified page as a new generic base version of the page.
- Click **View** to view the page in a browser. If the page needs certain URL parameters to display correctly, for example, a form code, enter the parameter(s) into the **Page View URL Params** field first.

A page base content is the page's content as defined in the space's WAR file.

Only users with administrator's permissions can perform all the above-mentioned actions. Regular users can only edit and view pages.

Next, learn about form space resources.

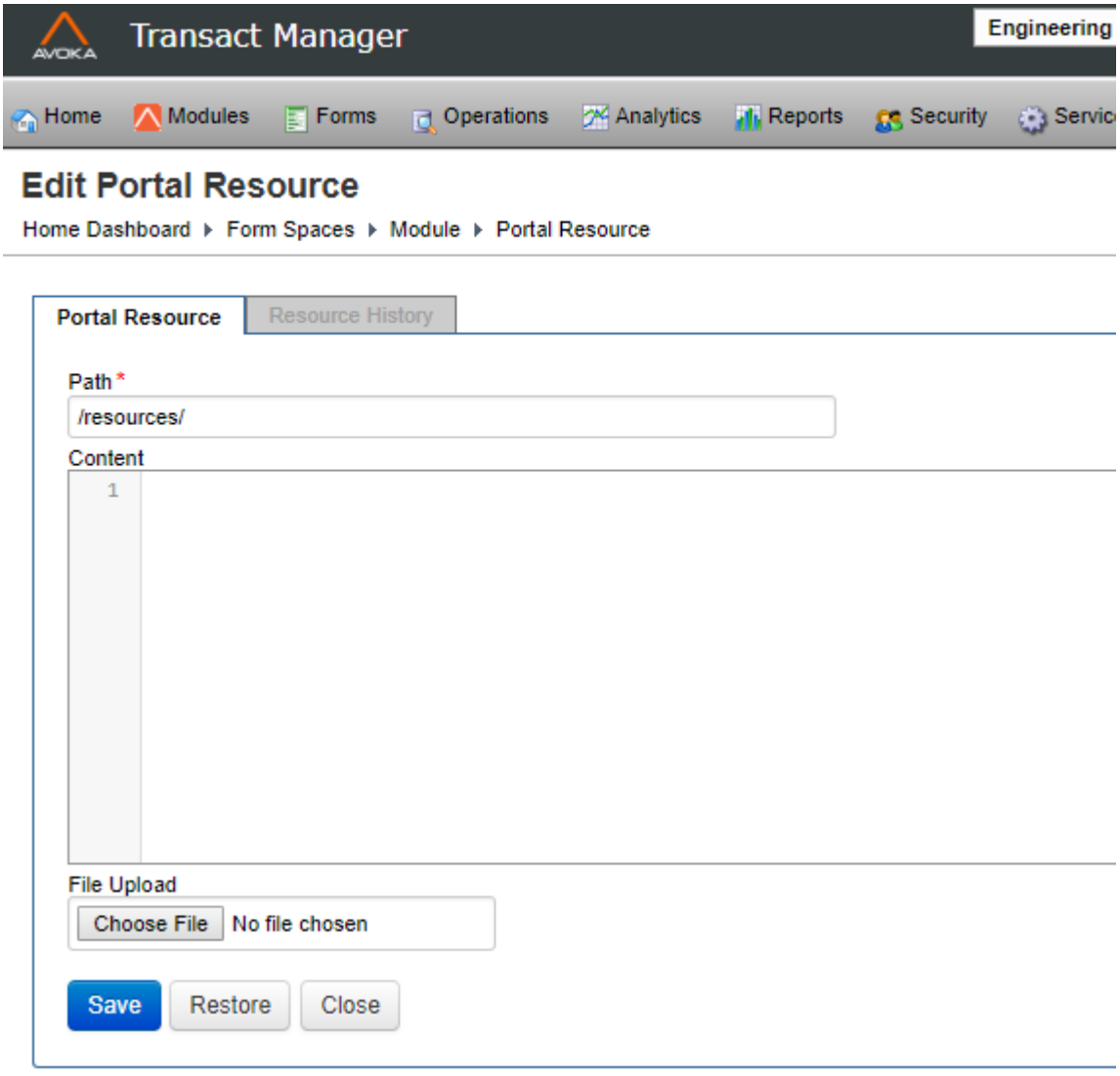
Create a Form Space Resource

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows users with administrator privileges to create their own unique resources for form spaces.

To create a new resource:

1. Select **Forms > Form Spaces**.
2. Locate a form space and click **Edit**.
3. Click the **Resources** tab and then click **New** as shown below.



4. Enter a path of a new resource where it will be hosted. The path is relative to the form space context path, for example, `resources/CSS/learning.css`.
5. Enter a content of the new resource. You can add content directly into the content text area for a CSS resource or any other textual resources, for example, a Word document.
6. Click **File Upload** to attach a file for resources that require attachments, such as images or PDFs.

7. Click **Save** to update the changes. This creates the resource.
8. Configure the resource.

Next, learn how to view all forms hosted on a form space.

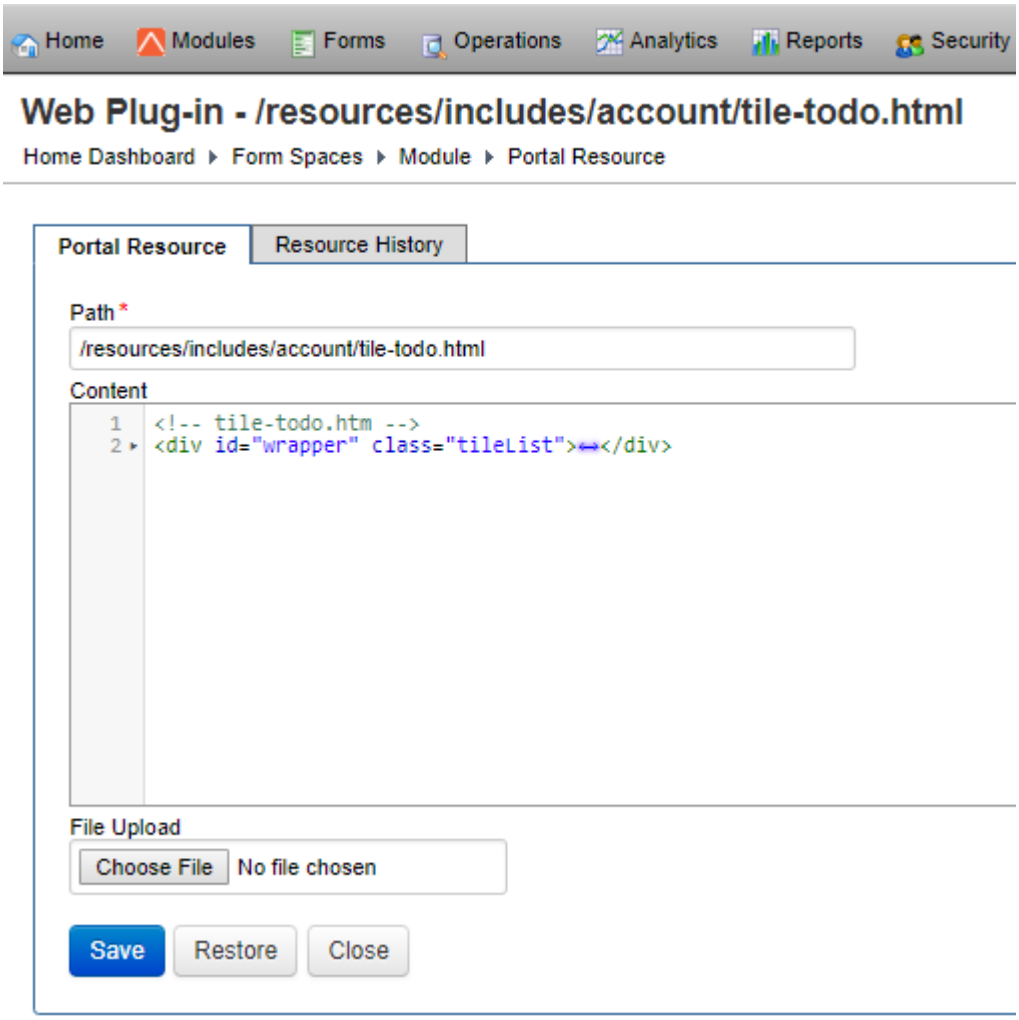
Configure a Form Space Resource

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view and configure resources belonging to a form space.

To configure a form space resource:

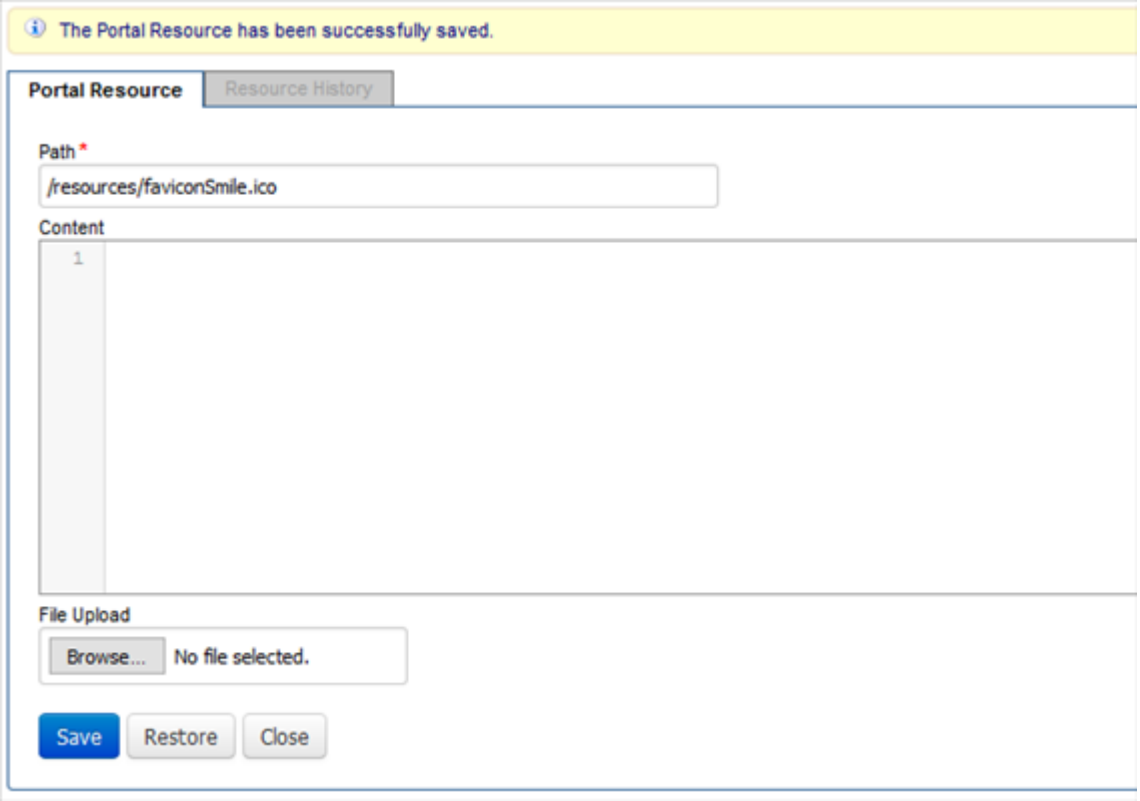
1. Select **Forms > Form Spaces**.
2. Locate a form space and click **Edit**.
3. Select the **Resources** tab.
4. Locate a resource and click **Edit**



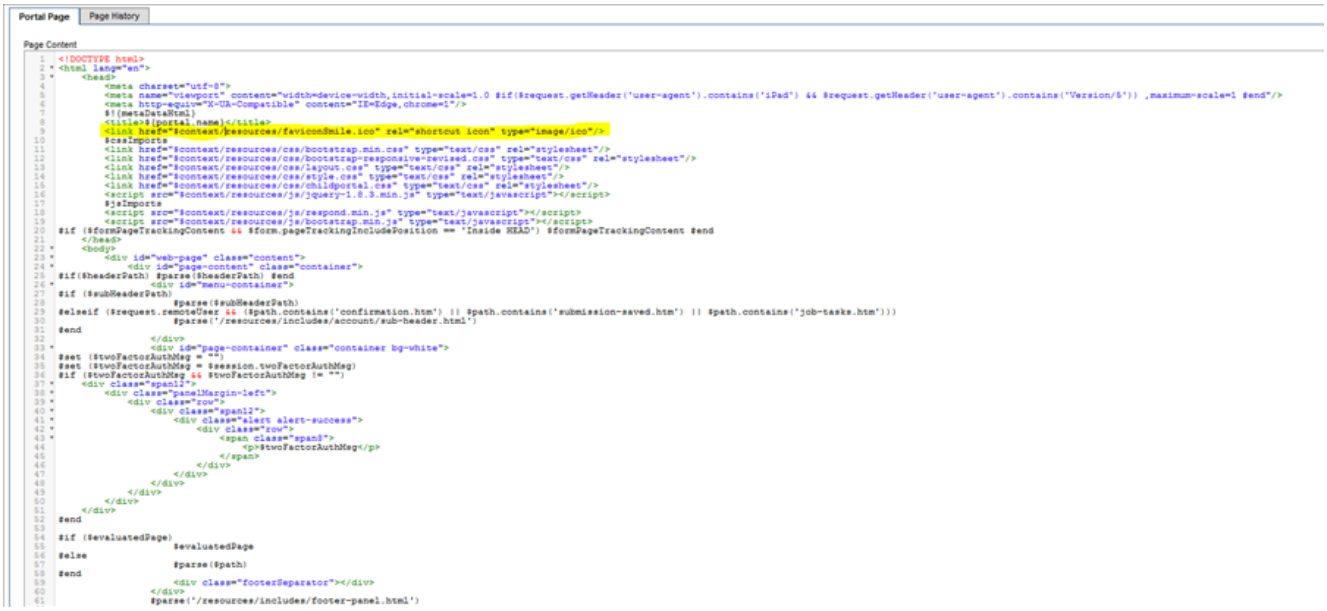
5. Edit the path and upload a new file, for example the GIF image resource.
6. Edit the content of a text file, for a CSS or HTML files only.
7. Click **Save** to update the changes.
8. Click **Restore** to undo any new changes.

To add a new fav icon file:

1. Select **Forms > Form Spaces**.
2. Locate a form space and click **Edit**.
3. Select the **Resources** tab.
4. Click **New** and upload a file. Give it a different file name from the standard one as browsers cache these things very aggressively, and they are hard to clear out. Make sure it is in the correct format.



5. Edit the Space Page “Border Template” and set the path to the new fav icon image.



To view a form space resource history:

1. Select **Forms > Form Spaces**.
2. Locate a form space and click **Edit**.
3. Select the **Resources** tab.
4. Locate a resource and click **Edit**
5. Click the **Resource History** tab to view the history of a form space resource showing every edit that has ever been made to this resource.
6. Click **Restore** to roll back the resource to a previous version.
7. Click **Compare** to compare the content of a previous version of the resource with the current one.
8. Click **Remove** to delete the selected version of the resource.
9. Click **Close**.

Next, learn how to view all form resources.

View Form Space Resources

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

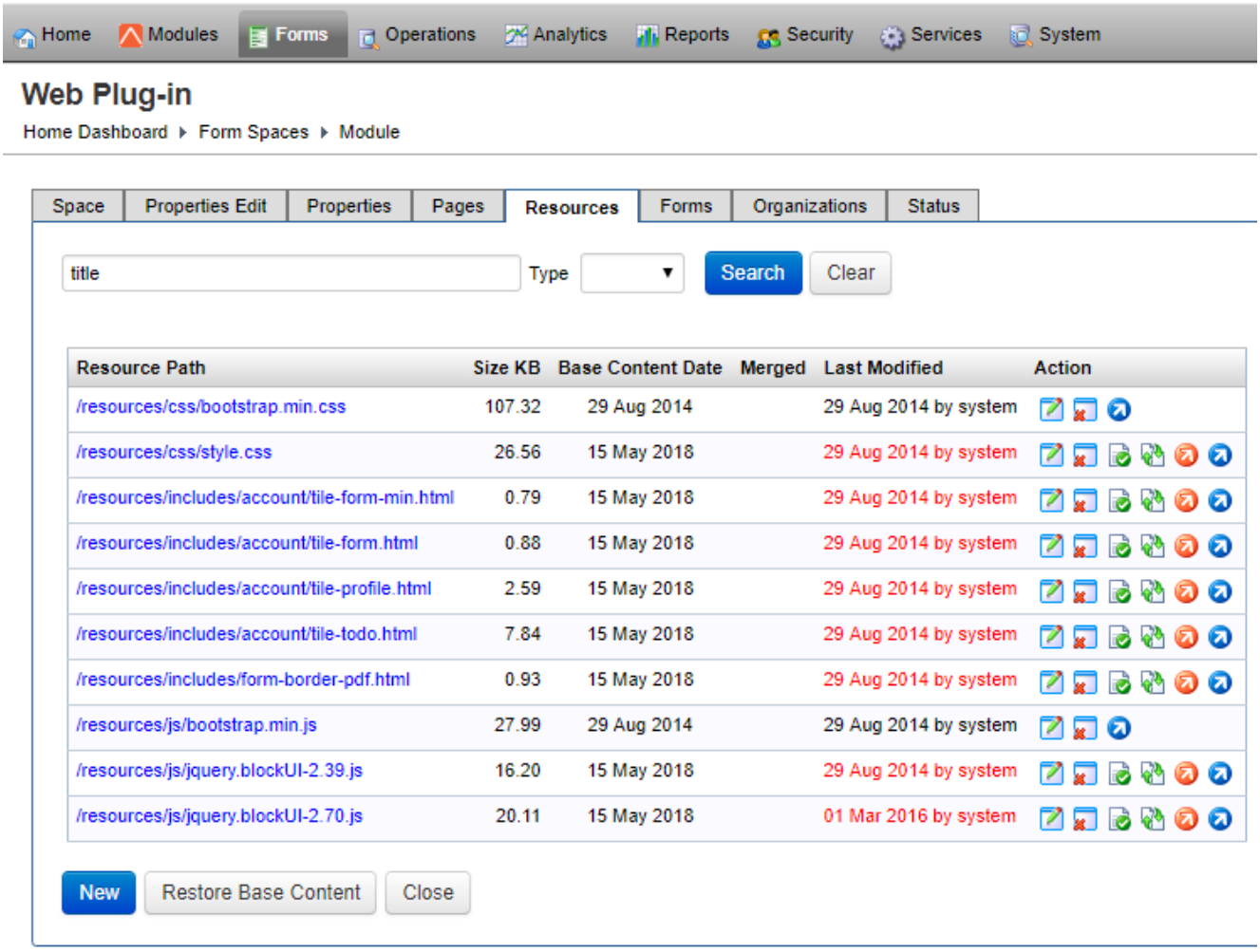
Manager provides form space resources that are support files of a form. You can view all resources, create new, and update the existing ones. For example, you could create a new image for the 'Forgotten Password' space page.

A form resource type can be one of the following:

- CSS
- Image: GIF, JPG, PNG.
- PDF
- HTML
- JavaScript
- Plain text

To view form space resources:

1. Select **Forms > Form Spaces**.
2. Locate a form space and click **Edit.**
3. Select the **Resources** tab.



The Last Modified column shows the date when the resource was modified and the name of the user who modified it. If the content is different from the base content (the resource's content as defined in the space's WAR file), the entry is highlighted in red, and additional actions (to view the differences between the current and base content of the resource and to restore the base content of the resource) are made available, as shown below.



4. To filter or search the resources list by one or more criteria, specify the following settings and click **Search**:
   - Any text message that resource name and content may include.
   - Resource types.
5. Click **Clear** to reset the search criteria.

You can perform the following actions on a selected resource:

- View and configure the content of the resource.
- Restore the resource's content back to the original base content. This removes all modifications and return the resource's content back to its base.
- Promote the modified resource to the base resource. In other words. This makes the modified page the new generic (base) version of the resource.

Next, learn how to view all forms hosted on a form space.

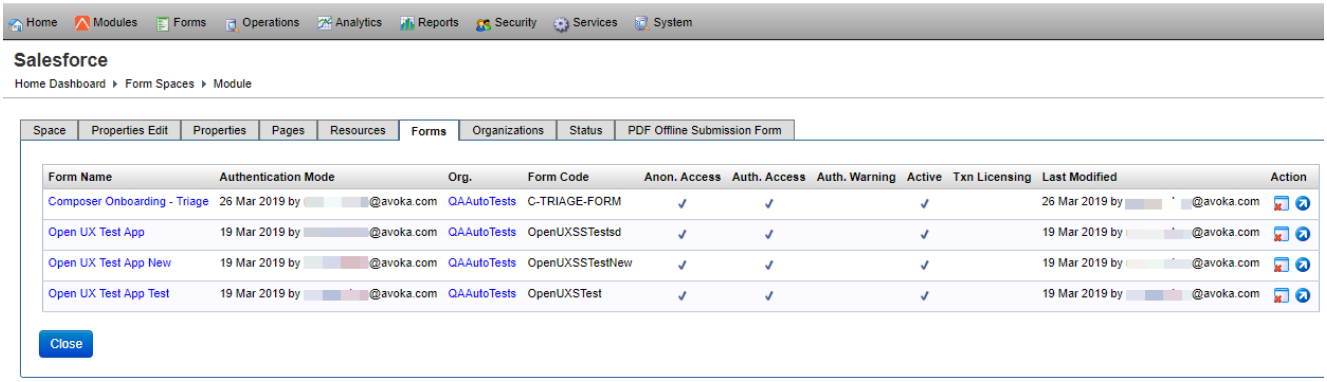
View Forms Hosted on a Form Space

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view all forms hosted on each form space, so you can administer and manage them.

To view all forms hosted on a form spaces:

1. Select **Forms > Form Spaces.**
2. Locate a form space and click **Edit**.
3. Select the **Forms** tab to see all forms listed page by page, with the maximum number of records per page configured in the users preferences.



4. Click **Close**.

You can also perform the following actions:

- View a form
- Remove a form from this form space
- Form test render

Next, learn how to assign organizations to a form space.

Form Space Hot Deployment

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Hot deployment of a form space is the process of updating a form space WAR file without restarting Manager. This enables developers to update a form space with few changes, re-build it as a WAR file, and then deploy it to Manager on-the-fly, so the changes can be available to the form users right away.

To perform a hot deployment, prepare a WAR file and then import it.

Use this functionality in a development environment only and restrict user's access to form space import to prevent or minimize accidental or malicious deployment of WAR files that can potentially be very harmful to the Managerenvironment.

> Do not use this feature to upgrade to a new version of Manager. Follow the Manager upgrade instructions instead. If you have already used this feature to upgrade, verify that your form space is operating as expected.

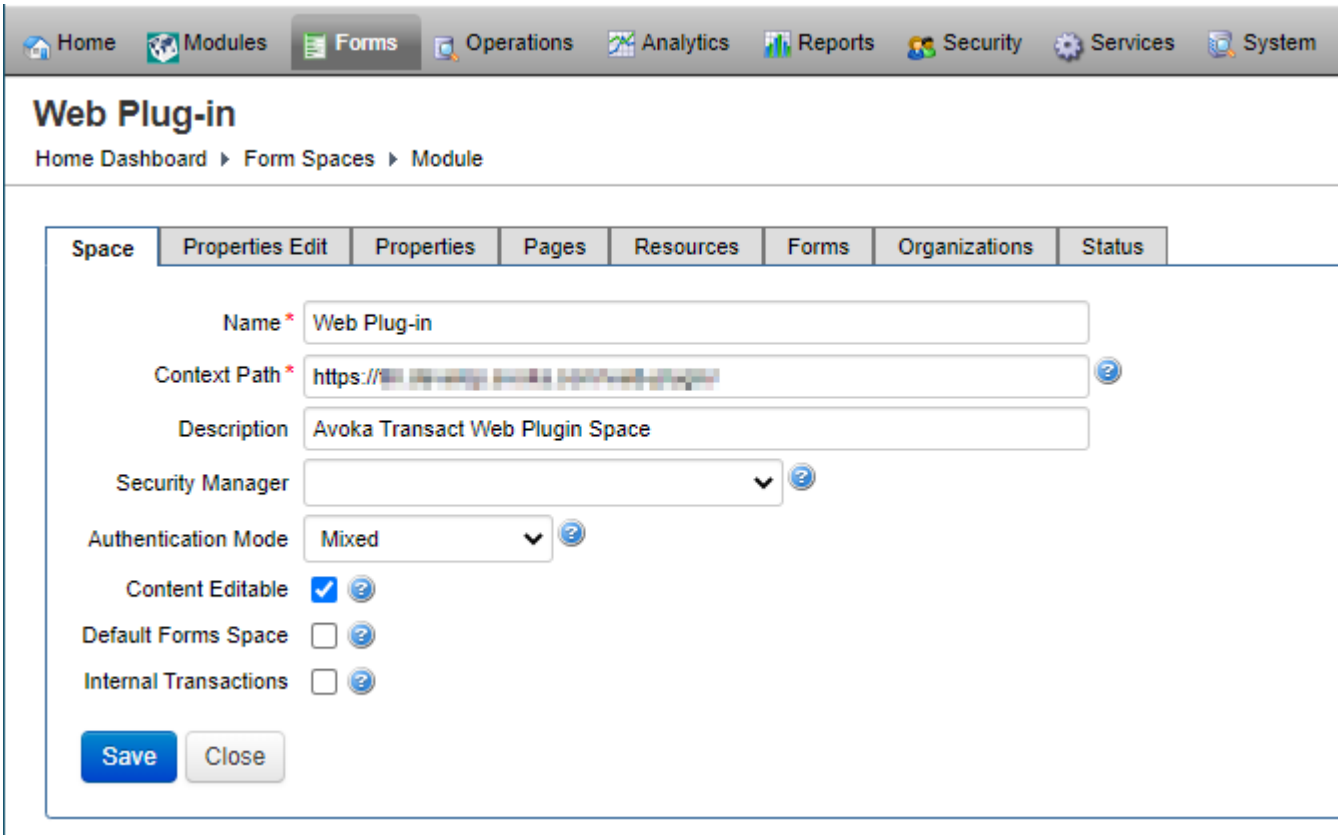
Next, learn how to migrate a form space.

Web Plug-In Form Space

Journey Manager (JM) | System Manager / DevOps | 21.11

Manager comes with the default Web Plug-In form space, which exists on every server, to ensure no errors occur on client systems during importing and publishing forms. It's also used to monitor a Manager server, so you do not remove it.

You can log in to this form space without user authentication, as there is no security manager configured and the **Authentication Mode** is set to **Anonymous Only**, as shown below.



You can change the form space configuration and update properties, if required.

If you try to build a Manager form version in Maestro from an organization that does not have access to the Web Plug-In form space, you will get the error: "This design does not have access to the Web Plug-in Space". In this case, you have to assign your organization to the Web Plug-In form space.

Once the form has been imported into Manager, you can assign a form space to the form.

The Maguire form space was the default form space in previous versions of Maestro. The Maguire form space

is an example only and should only be used in a development environment. The Maguire form space is an installation verification sample, which is used to ensure a server is configured properly.

Next, learn how to view all form spaces.

Host an Embedded Form

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager allows you to host an embedded form in a form space's web page, so your form appear as if it were a seamless part of an existing website. There are various ways of doing this, which are discussed in integrate forms to existing websites.

This article focuses on the process of embedding a form inside an iframe element. In particular, it shows how to use a JavaScript helper method, showEmbeddedForm, to embed your form.

Reasons for employing the iframe method can include:

- Customers may not wish to have branding included in the form, preferring instead to manage their own branding within the pages where they want to show their minimally or non-branded forms
- Customers may wish to take advantage of controlled advertising services to surround the form with dynamic ads
- URLs will always reflect the customer's domain, rather than an external URL if the form is presented via a redirection
- Embedding the form may provide a more consistent user experience across the whole web site

How does it work?

Transact Manager includes the ability to embed a form inside a Form Space page. This is accomplished by JavaScript code which injects an iframe element into the page, and renders a form into it.

This guide will show you how to configure Transact Manager through the management console to accomplish this.

In following this guide, you will be:

- Setting up appropriate security access permissions to your form:
  - Assigning the Organization which owns the form to the Form Space
  - Assigning the Form to the Form Space
- Editing a Page in the Form Space to force it to show the Form. In this article, we will be editing the 'index.htm' page to host the form
- Inserting JavaScript in the page to cause it to create an `iframe` and render the form into it

The prerequisite for this guide is that you already have created your form in your Organization.

Step-by-step guide

1. If you need to create a new Form Space, follow the instructions at Managing Form Spaces
2. Ensure your Form Space is functioning correctly and can be accessed from a browser

3. Ensure your Form is built and can be rendered in one of the standard Form Spaces, such as Maguire or Web Plug-In
4. Open the Form Space in Transact Manager, select the Organizations tab, and assign the Organization which owns the form to the Form Space
5. Open the Form in Transact Manager, select the Spaces tab and assign the Form Space to the Form. Take note of the Form Code in the Form Details pane of the Form's Dashboard tab
6. Open the Form Space again, and select the Pages tab
7. Click the 'Index' link in the Name column of the Pages list (do not click the index.htm link in the Path column)
8. Paste this code into the Page Content box, change 'your-form'code' to your own form code, then click Save. You might also want to change the style to suit your form.

```
<style>
    #formDiv {height: 1000px;}
    .topSpace {display: none;}
    iframe {height: 100%;}
</style>
<div id="formDiv">
</div>
<script type="text/javascript" src="$context/smartform-webplugin.js"></script>
<script type="text/javascript">
    var replacableParamsMap = { id: '', formCode: '' };
    var queryParamsMap = { id: 'formDiv', formCode: 'your-form-code' }, tuple;
    var queryParams = document.URL.split('?')[1];
    if (typeof queryParams != 'undefined') {
        queryParams.split('&').forEach(function (queryParam) {
            tuple = queryParam.split('=');
            var paramName = tuple[0];
            var paramValue = tuple[1];
            if (typeof replacableParamsMap[paramName] === 'undefined' && typeof
queryParamsMap[paramName] !==  'undefined') {
                queryParamsMap[paramName] = [queryParamsMap[paramName]];
                queryParamsMap[paramName].push(paramValue);
            }
            else {
                queryParamsMap[paramName] = paramValue
            }
        });
    }
    Avoka.SmartForm.WebPlugin.showEmbeddedForm(queryParamsMap);

</script>
```

9. Open the Form Space again and select the Pages tab

10. Now click on the 'index.htm' link in the Path column - a new web browser tab or window should open, and the form should render into it

Explanation of Form Page Content

The JavaScript code above simply gathers query parameters from the URL, adds a couple of default values, and calls the showEmbeddedForm function to do the heavy lifting, passing the collected parameters.

The code contains the following:

- Some style to give the embedded form some height and to hide headers
- A JavaScript library which contains the showEmbeddedForm function
- A JavaScript script containing
  - A queryParamsMap map object which is populated with:
  - default 'id' value which identifies a div on the web page to contain the injected iframe
  - default 'formCode' value to identify the form to be rendered into the iframe
  - Some code to take all of the query-string parameters from the URL and add them to the queryParam-sMap
  - If the parameter name is reserved 'id' or 'formCode', it will be replaced in the map with the value from the query-string parameter
  - If the parameter name is not reserved, it will be added to the map if it is not already mapped
  - If an unreserved parameter name has already been mapped, the map value will be converted to an array and the new value appended to it
  - A call to the showEmbeddedForm function, which is passed the accumulated map as a parameter
  - The function will use the 'id' and 'formCode' values to inject an iframe into the 'id' HTML element and render the 'formCode' form into it
  - All other parameters will be passed to the showEmbeddedForm function

This is illustrated below:

Page Content




```
<style>
    #formDiv {height: 1000px;}
    .topSpace {display: none;}
    iframe {height: 100%;}
</style>
<div id="formDiv">
</div>

<script type="text/javascript" src="$context/smartform-webplugin.js"></script>
<script type="text/javascript">
    var replacableParamsMap = { id: '', formCode: '' };
    var queryParamsMap = { id: 'formDiv', formCode: 'creditcard' }, tuple;
    var queryParams = document.URL.split('?')[1];

    if (typeof queryParams != 'undefined') {
        queryParams.split('&').forEach(function (queryParam) {
            tuple = queryParam.split('=');
            var paramName = tuple[0];
            var paramValue = tuple[1];

            if (typeof replacableParamsMap[paramName] === 'undefined' && typeof queryParamsMap[paramName] !==  'undefined') {
                queryParamsMap[paramName] = [queryParamsMap[paramName]];
                queryParamsMap[paramName].push(paramValue);
            }
            else {
                queryParamsMap[paramName] = paramValue
            }
        });
    }

    Avoka.SmartForm.WebPlugin.showEmbeddedForm(queryParamsMap);

</script>
```

Next, learn how to configure form space properties.

Map a Template to a Model

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to map a template to a model

The following table lists the model variable names and types for each page.

- Java type names are not fully-qualified unless clarity is needed. For example, class String is the normal java.lang.String unless otherwise noted.
- Java types marked as an Entity are in the package com.avoka.fc.core.entity unless otherwise noted.
- Model classes that "inherit from" other model classes inherit the variables of those classes. For example, the model for ErrorPage not only includes the variables listed for it, but also all of the variables for BorderPage.
- The fact that the model can include a given variable does not mean that the variable is non-null at runtime.

Page	Model Class	Variable Name	Java Type	Inherits From
	PortalUtils-Form			
(global)	This is not a Model class, but a utility method used by Model classes to set certain properties in the Model from the Form PortalUtils.addFormAndVersionProperties(Page, Form)	form	(Entity) Form	
formProperties	Map<String, String>			
	BorderPage	headerPath	String	
serverBuild	String			
portal	(Entity) Portal			
detectJavaScriptTimeout	int			
disableUserProfileEdit	boolean			
allowUserCreation	boolean			
permissionSet	Set<String>			
error.htm	ErrorPage	entityId	String	BorderPage
errorLog	(Entity) ErrorLog			
errorContext	String			
showDetails	Boolean			
messageKey	String			
errorMessage	String			

errorTitle	String			
submission-cancelled.htm	SubmissionCancelledPage	formRendererUrl	String	BorderPage PortalUtils-Form
formCode	String			
submission-saved.htm	SubmissionSavedPage	submission	(Entity) Submission	BorderPage PortalUtils-Form
		isAnonymous	boolean	
		formUrl	String	
		userAccount	(Entity) UserAccount	
		userName	String	
		logoutMsg	String	
		todoCount	int	
		evaluatedPage	String	

Next, learn how to configure a form space email template.

How to Unlock a Transaction

Journey Manager (JM) | System Manager / DevOps | 21.05

Manager allows you to configure a form space as well as a form version with the **Save Challenge Lockout Minutes** property, which is the duration in minutes a submission is locked out when the maximum number of save challenge requests has been exceeded. This property can be set to 0, which makes a form transaction permanently locked out. In this case, you need an administrator to manually unlock it.

Another option is to unlock a form transaction using an API. Unfortunately, Manager doesn't have any setting or a Fluent API to unlock such a transaction, so the only way to do this is by using Core API Groovy code.

To unlock the transaction with a script:

1. Copy the following Groovy script to the Groovy console.

```
import com.avoka.fc.core.dao.DaoFactory
import com.avoka.fc.core.entity.Submission
import org.apache.cayenne.access.DataContext

Submission submission = DaoFactory.submissionDao.getSubmissionByTrackingNumber
("9367VSY")
submission.setSaveChallengeFailures(0)
submission.setSaveChallengeLastFailure(null)
DataContext.getThreadObjectContext().commitChanges()
```

2. Update the tracking code with the transaction reference you want to unlock.
3. Click **Execute Script**.

After you unlocked the transaction, update the **Save Challenge Lockout Minutes** property either on a form space or a form version level to make sure it will not happen again.

Next, learn how to a configure form space email template.

Upgrade a Form Space

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Upgrading form spaces is a necessary task when Manager updates are done. While the form spaces may continue to work "as-is", important security updates and valuable new features may be getting skipped without the form space upgrade. When Manager is upgraded is it important to make sure that the form spaces get rebuilt. For a system that has content only form spaces (not custom Java Spaces) this is a relatively simple but necessary step as the form space contains the base page mark-up which is used in Manager by the page and resource merging facilities. If the war files are not rebuilt to represent the Manager version you are upgrading to, then you will not see any changes to the mark-up that have been made as part of the product. These may include new features and bug fixes or security patches.

After a Manager upgrade each Space in the Space/Space list should have a version number which matches the version of Manager you are upgrading to. The following is an example screen shot that shows a system that has been upgraded from 4.2 to 4.3 but the main form space has not been rebuilt.

We have seen instances where security scans have raised issues because the Space mark-up has not been merged in with the new mark-up for the later TM version. Please see the 'Styling and Mark-up' section below.

Rebuilding Spaces

Content Only Spaces

will automatically upgrade the standard TWeb and TField Spaces(Maguire, TransactField App, Web Plug-in), any other Spaceneeds to be rebuilt manually. Content only TWeb Spaces can easily be built using the Manager Installer. At the moment TField Spaces need to be manually prepared and there are some instructions in the Manager install guide to help with this.

Custom Java Spaces

Custom Java Spaces are more complex and the form spaces need to be manually prepared and the code level customization need to be merged into a clean Java Space for the version of Manager you are upgrading to. This can be complex and time consuming and is beyond the scope of this article. You will need to refer to the original builders of the form space for help with this aspect of the re-build.

Important Points when re-building and deploying a Space:

One of the key points when upgrading a Transact Manager Spaceis that you want to replace the old form space with a new form space and maintain the forms and user relationships with in Manager. The following are a few important points to follow to make sure this happens:

1. Make sure the Space war file name stays the same.
2. Make sure the Space context stays the same.

3. Space Name needs to be the same. This is set in this Manager installer when creating a TWeb Space or directly in the db-config1.xml file's portal element (portal name="testPortal") if you are manually building a form space.
4. Replace the Space in the deployment directory. On a single node system you can deploy the war file through the Manager admin console.
5. Merge in the styling and mark-up changes.

Styling and Mark-up

Merging in styling and mark-up is an important step and this can be time consuming if the styling or mark-up as moved far away from the standard product set.

1. If the Spacewar includes all the styled pages then hit the restore base content button on the resources and pages tabs within the Space config in Manager. Beware - This will replace all styling cached in the Manager database.
2. If the styling is in Manager then you have to merge in the pages and resources manually within the Manager console.

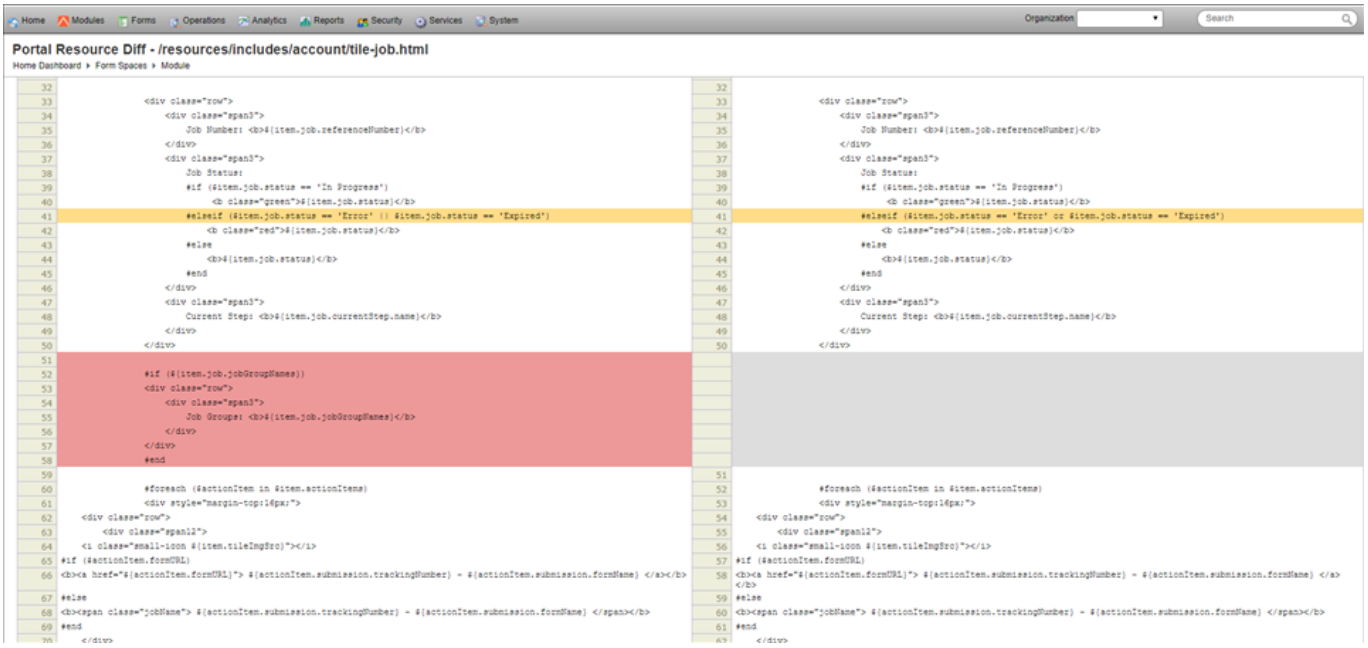
A developer is needed if the original form space was customized, and the level of effort needed depends on how extensive the customization is. When the form space assets ZIP file is imported, it can bring with it some legacy code that ignores or misuses feature and security upgrades. Resolving this requires a manual review, so you need to do the following

Select **Forms -> Form Spaces** and click the **Resources** tab.

Take note of the Base Content Date, Last Modified, and Merged columns. The Last Modified column will be populated in a red font if that resource has been changed. Comparing the Base Content Date and date in Last Modified can give a clue to the age of the modified content.

Use the red “View Content Diff to Base Resource” Action icon to open a new tab displaying the base resource next to the modified resource. Differences between the files will be color highlighted to make changes, additions, and deletions apparent.

This screen is only for review - there are no controls or ability to save from here.



Review all changes, and evaluate for functional impact. If a change only modifies a color, or font, or ordering or placement of screen elements, it may not impact security or new features and can be left in. Base content changes that introduce a new feature, or patch a bug, need to be examined and incorporated into the resource. New, vital, security enhancements may be left out if the new base content is not merged into the resource!

Use an editor suitable for the type of resource to make changes, and load the updated resource file into Transact Manager. Depending on the file type, you can use the built-in text editor, or re-upload the file (for images, etc.). Test the updated resources, then use the “Mark Content Merged” Action icon when the resource edits and tests are complete.

The Pages tab in the Module screen is similar to the Resources tab, but lists just the Page definitions in the Transact Form Space. Pages differ from html stored as a Resource in that Pages can be “navigated to” -- they are referenced as links, buttons, or tabs, in the Form Space and generally have a Java object backing them. Hence they are listed on their own tab. Despite these differences, Pages can be altered from base content and need to be reviewed during an upgrade just like Resources. Follow the same steps as listed above for every Page that has been changed.

Next, learn how to a configure form space email template.

Work Space

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager comes with a Work Space web application deployed as a specialized form space, which allows users and business staff to perform the following actions:

- Work with tasks and collaboration jobs
- Complete and review forms
- Access draft and saved forms
- Provide help desk support

Usually, a form is hosted as an unsecured page on the form space. An anonymous user must provide an email address when filling the form. Manager relies on this email to send status updates and saved transactions, and assign tasks back to the user. Additionally, one of the form fields can be used to provide a security challenge question, which the user must enter to open a saved form or anonymous task created by a collaboration job. However, Work Space can be configured to apply various user authentication flows, including **2FA**[1], to strengthen its security. In this case, you need to log in to use the web application.

Work Space provides users with the following views:

- Home is the Work Space dashboard.
- Forms allows you to find and select from the list of available forms.
- Tasks allows you to complete your outstanding forms and tasks.
- History provides you with the history view of forms you have submitted and their processing status.
- Reviews allows you to search for review and approval collaboration jobs.
- Help Desk allows you to search for forms saved or completed by your customers.
- Account allows you to edit your account settings and change your password.

> Some views are only shown to users with certain roles and privileges, see each view's documentation for details.

Work Space is generally used for dealing with authenticated user transactions, but it also supports unauthenticated anonymous transactions. These permissions must be configured in Manager.

---

[1] Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.

You should configure the group's permission to edit tasks, claim, assign and return them to a group in order to start using Work Space. Manager provides several authenticated user roles for the Work Space portal.

> You should use the Work Spaces instead.

Next, learn how to configure work space properties.

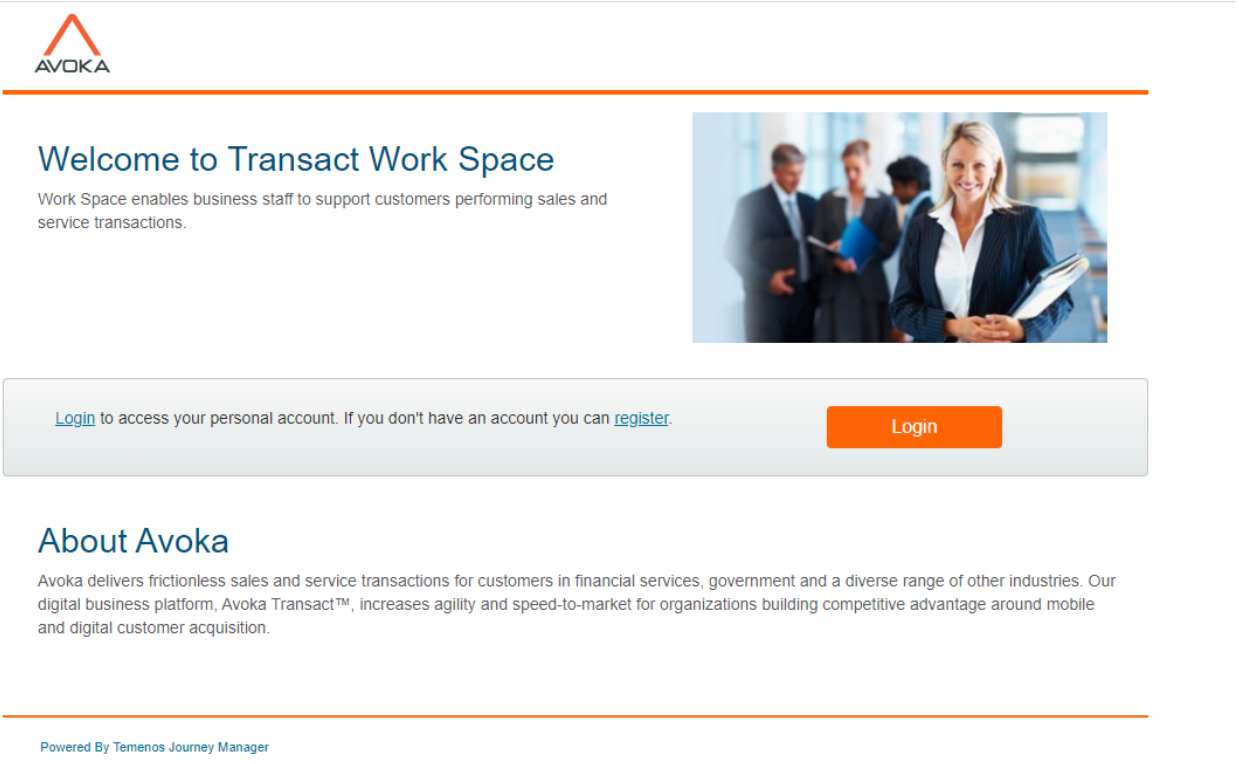
Work Space Login

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

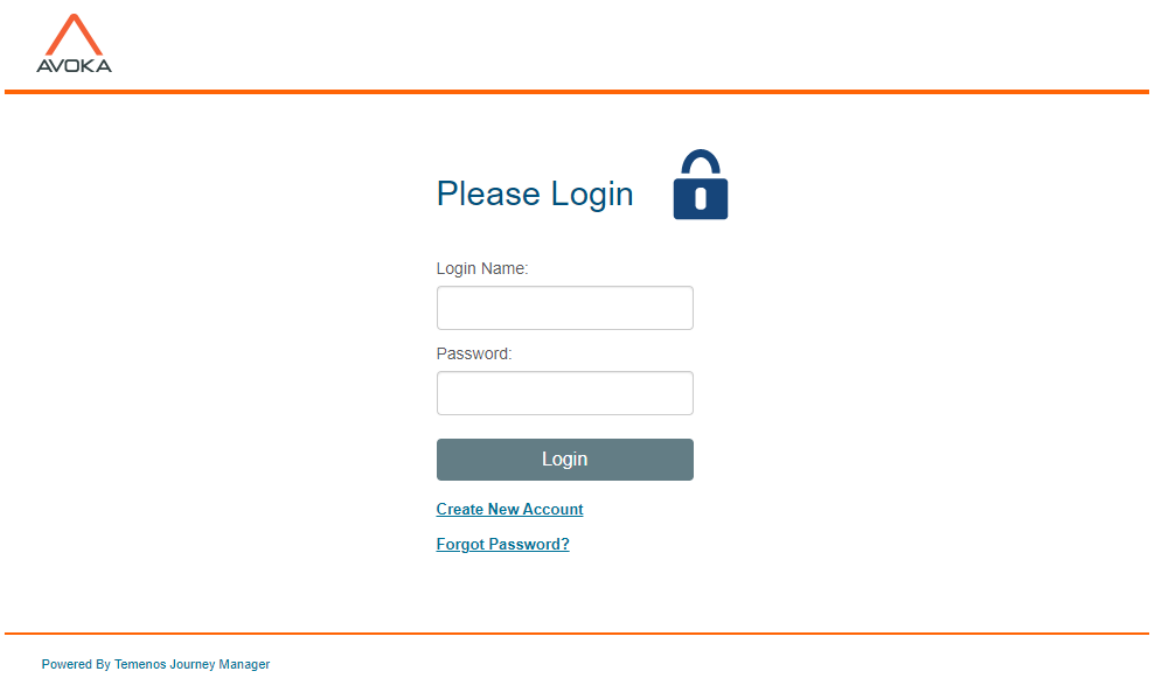
Work Space can be configured to authenticate users before allowing them to use it. In this case, you need to log in using your account credentials. If you don't have your account, contact your system administrator or support team to create one for you.

To log in to Work Space:

1. Go to the Work Space URL in your browser, for example, **`https://YOUR-DOMAIN-URL/-workspace/login.htm`**

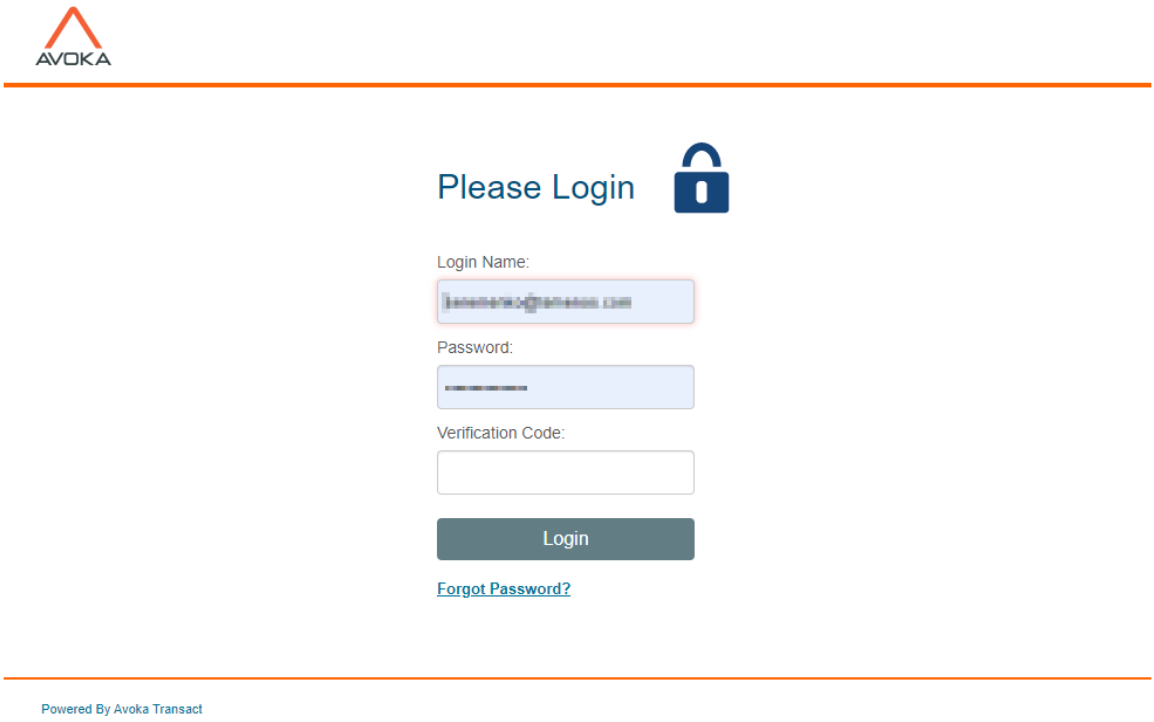


2. Click **Login**. Manager displays a login screen for you to enter your credentials.



Work Space can be configured to use **2FA**[1] to verify your identity, so you must provide an additional verification code to login.

[1]Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.



3. Provide your user name in the **Login Name** field.
4. Enter a password in the **Password** field.
5. Provide a unique one-time code in the **Verification Code** field. If this is your first login and you haven't set up your Authentication App, you don't need to enter the code.
6. Click **Login**. Manager checks your credentials and, if they are correct, redirects you to the home page. If your system is in the process of setting up 2FA, you may be taken to the enrollment page where you can start the 2FA enrollment process.

Next, learn how to change a password.

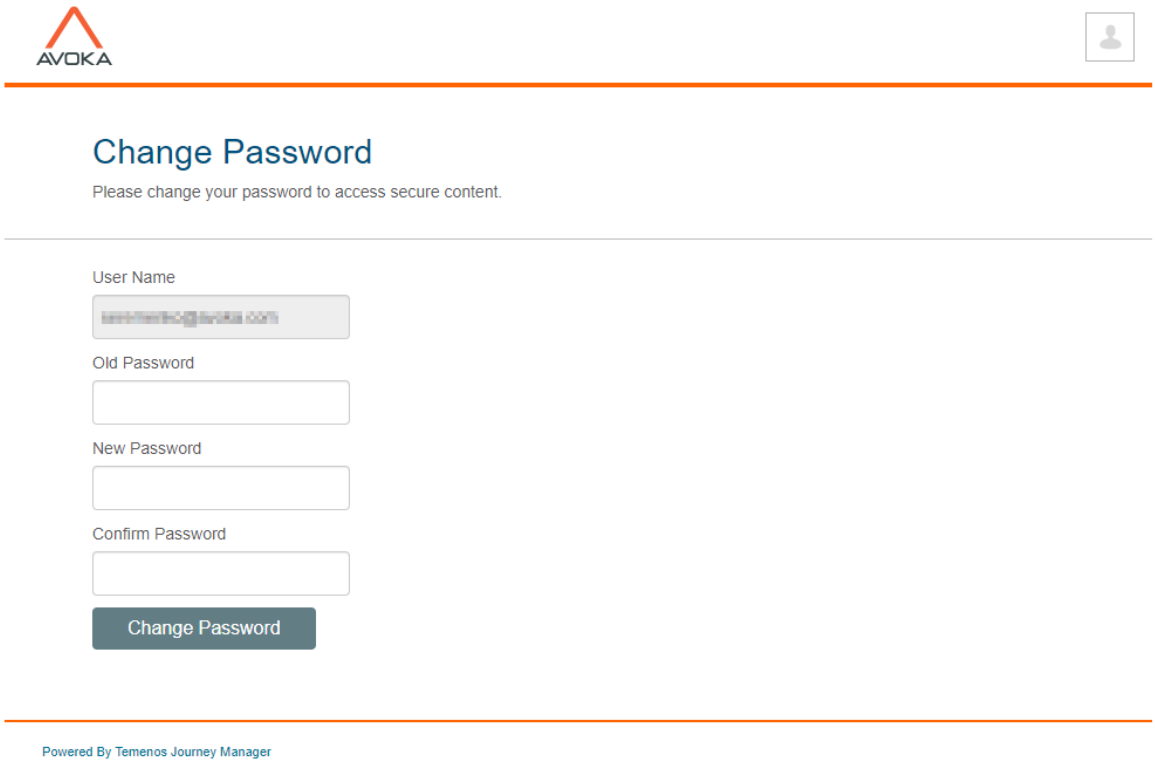
Work Space Change Password

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Work Space can be configured to authenticate users with a user name and a password. There are situations when you need to change your password, for example, on a first login or when your password was compromised.

To change your password:

1. Click a profile icon or the **Account** tab.

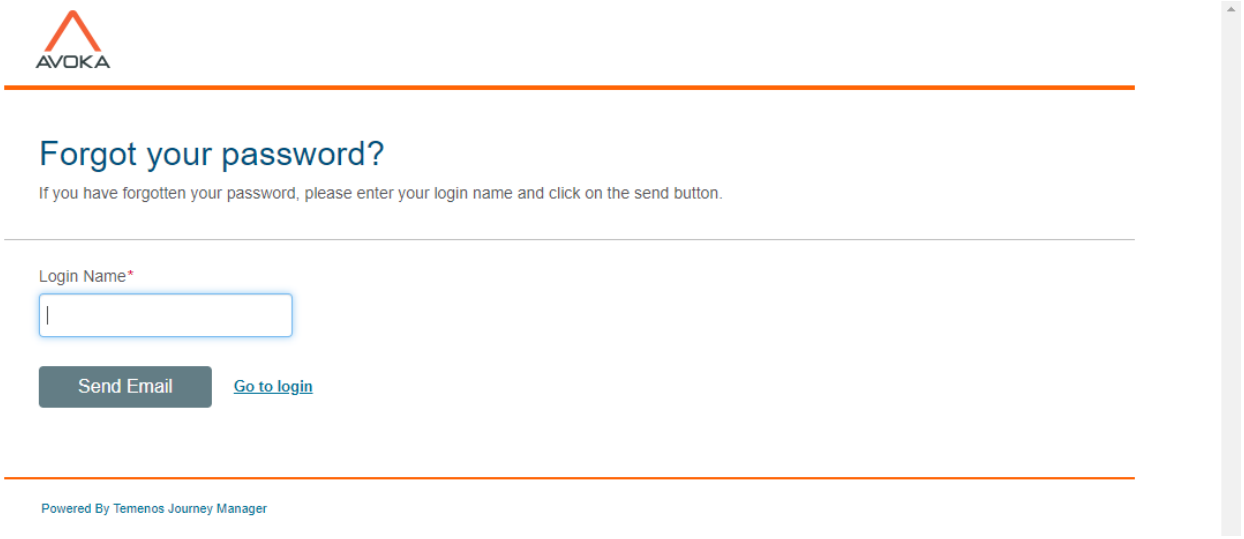


2. Provide the old password in the **Old Password** field.
3. Enter a new password in the **New Password** field and confirm it in the **Confirm Password** field.
4. Click **Change Password**.

If you have an existing Work Space account but can't log in anymore, you must request a new temporary password.

To request a new temporary password:

1. Click **Forgot Password** on the login screen.



2. Provide your login name in the **Login Name** field.
3. Click **Send Email**. Manager will send a new temporary password to an email address configured in your user account.
4. Try to log in again and you will be asked to change your password.

If you still can't access Work Space with no clear indication why, try the following actions to resolve it:

1. Check Login History to see any login failures and lockouts. Account unlocked and failures continue.
2. Click Reset 2FA Key to request a new verification code. Check there is not error message displayed when you request the reset.
3. Check what form spaces, the user account a part of, use a security manager with 2FA.
4. Revoke all form spaces from the user account, but the one with 2FA.
5. Click Reset 2FA Key again to generate a new verification code.
6. Check your email box and follow the instructions install and configure the Authenticator App. You may need to re-sync your Google Authenticator App.
7. Log in to Work Space again using the new code.

Next, learn about the Home tab.

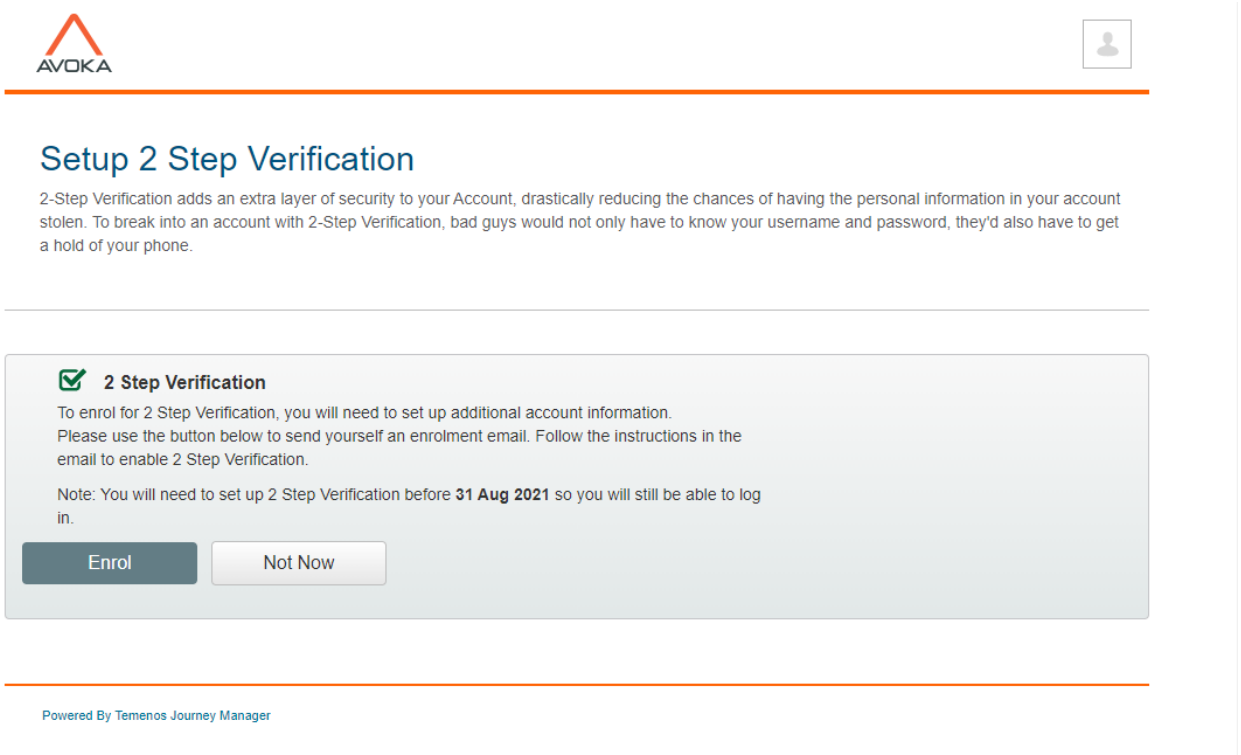
Work Space 2 Factor Authentication

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Work Space can be configured to authenticate users using a user name, a password, and an additional verification code, which is **2FA**[1]. In this case, on your first login, you are asked to set up 2 step verification.

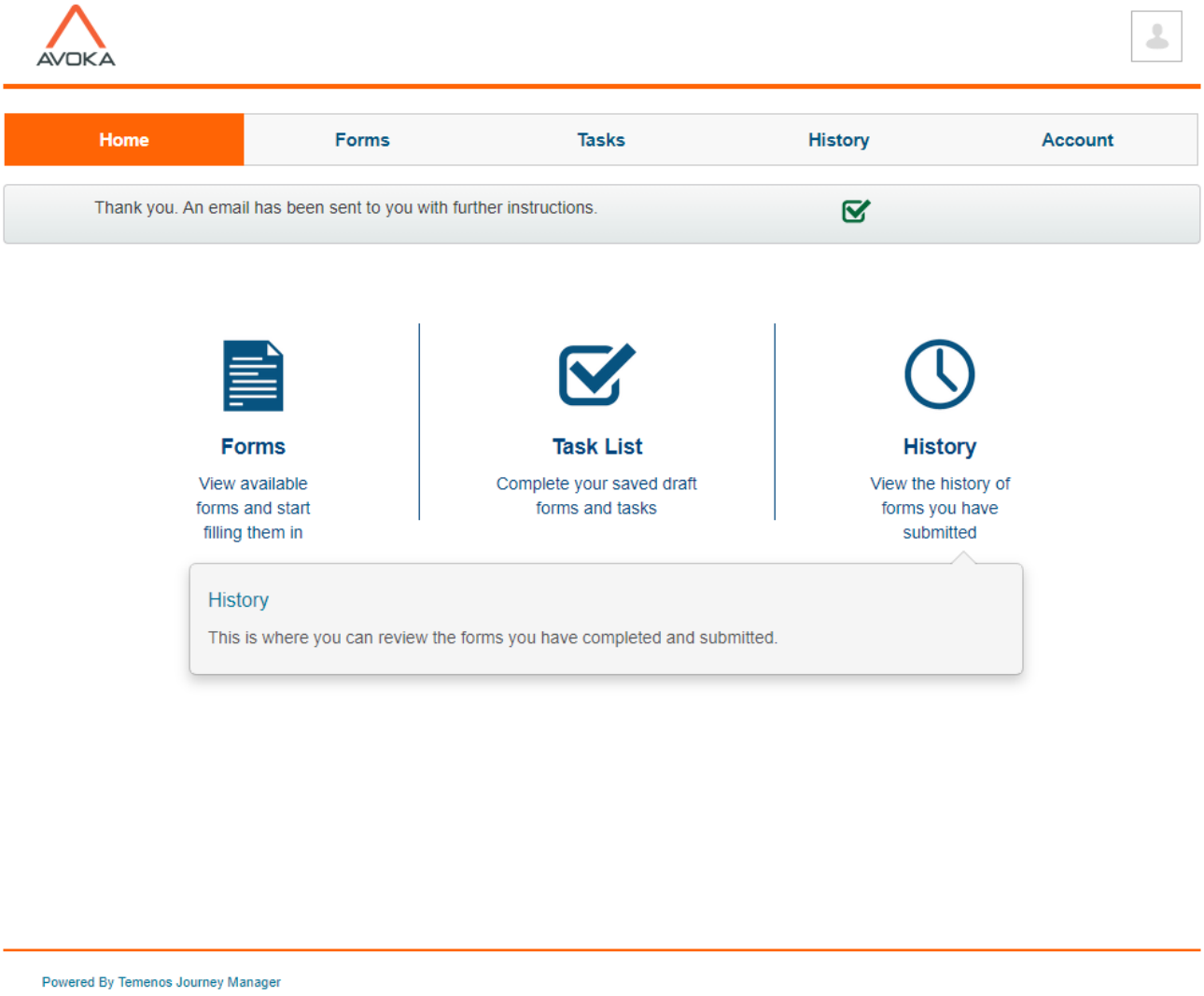
To set up 2 step verification in Work Space:

1. Log in to Work Space using your user name and password.
2. If your system (a form space that manages your forms) is in the process of setting up 2FA, you may be taken to the enrollment page where you can start the 2FA enrollment process.



3. Click **Enrol** to configure it now. Manager sends you an enrollment email with 2 Factor Authentication instruction. You need to complete the instructions before you will be able to login again.

[1]Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.



Otherwise, click **Not Now** to ignore it for now - you will be prompted to set up your 2FA next time to log in to Work Space. Manager redirects you to the home page.

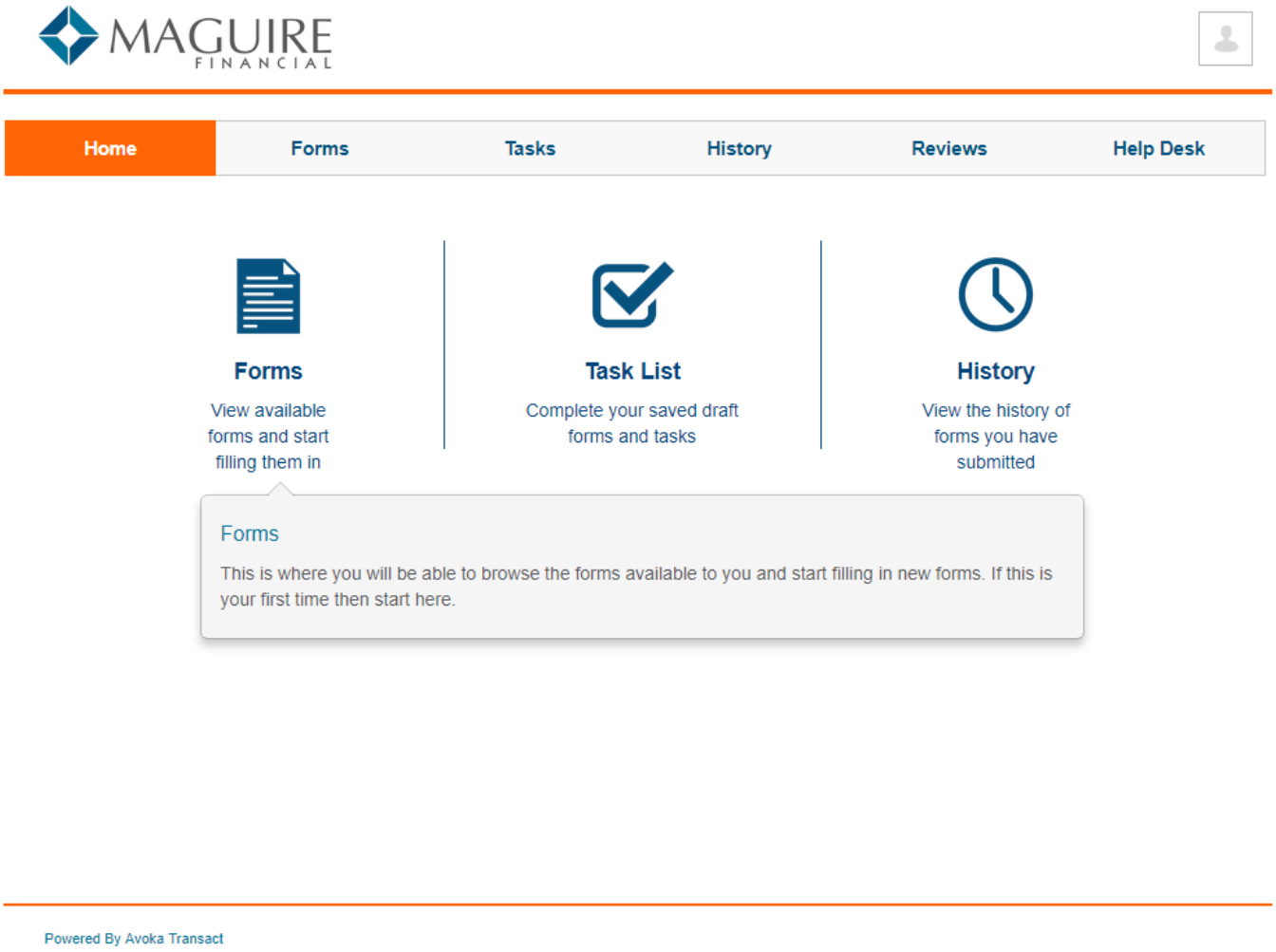
4. Open an email sent to you by Manager and follow the instruction to install and configure an Authenticator App.
5. Log out of Work Space and log in again using your credentials and a verification code from the Authentication App.

Next, learn about the Home tab.

Work Space Home

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Work Space has the **Home** tab, which is the dashboard of this application, that allows users to view relevant form and task details and navigate between the tabs.



The form is hosted as an unsecured page on the form space. An anonymous user must provide an email address when filling the form. Manager relies on this email to send status updates and saved transactions, and assign tasks back to the user. Additionally, one of the form fields can be used to provide a security challenge question, which the user must enter to open a saved form or anonymous task created by a collaboration job.

Work Space provides users with the following views:

- **Home** is the Work Space dashboard.
- **Forms** allows you to find and select from the list of available forms.
- **Tasks** allows you to complete your outstanding forms and tasks.
- **History** provides you with the history view of forms you have submitted and their processing status.
- **Reviews** allows you to search for review and approval collaboration jobs.
- **Help Desk** allows you to search for forms saved or completed by your customers.
- **Account** allows you to edit your account settings and change your password.

> Some views are only shown to users with certain roles and privileges, see each view's documentation for details.

Work Space is generally used for dealing with authenticated user transactions, but it also supports unauthenticated anonymous transactions. These permissions must be configured in Manager.

You should configure the group's permission to edit tasks, claim, assign and return them to a group in order to start using Work Space. Manager provides several authenticated user roles for the Work Space portal.

> You should use the Work Spaces instead.

Next, learn how to configure work space properties.

Work Space Forms

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Work Space has the **Forms** tab that lists all forms available to a user to start working on.

To filter or search forms:

1. Enter a search string in the search text field.
2. Click **Go**.
3. Click a form from the list to start working on.

Manager uses form groups to restrict the list of forms that are shown to the user. By default, a form is not assigned to a form group so it will appear in the list. When a form group is assigned to a form, and the form group has the **New Form** permission set, then it will be only visible to members of the form group.

Next, learn about Work Space tasks.

Work Space Tasks

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Work Space has the **Tasks** tab that lists saved forms, assigned tasks and submitted forms available to a user to start working on. Here, the submitted forms are ones that require additional attachments or payments and are not used the most. However, most collaboration jobs use the in-form transaction management features of the form space, which means the submitted forms will be empty.

To filter or search tasks:

- Enter a search string, for example, a job name, in the search text field.
- Select the time interval, when collaboration jobs were started, from the **Last** dropdown list The default is 90 days.
- Select a collaboration job status from the **Status** dropdown list, which can be assigned, saved or submitted.
- Select a form group from the **Groups** dropdown list, which can be:
- Blank - for any tasks that are assigned to the user account that you are logged in with.
- All Groups - for all form groups.
- A chosen form group.

> This selection is only available if a form space has the **Search Filter Group Select** checkbox selected. For more information, see configure a form space properties.

- Select the **Group Items** checkbox to show group items only .

A collaboration job creates tasks and assigns them to an authenticated or anonymous user, or a group. An assigned task can be one of the following service types:

- Form Task - the form XML can be pre-filled. Assigned to a user or a group and displayed to secure users in the Task tab.
- Review Task - the review task copies the form XML and attachments from a previous submission. Assigned to a user or a group and displayed to secure users in the Task tab.
- Anonymous Task - the task can only be assigned to an email. The anonymous form XML can be pre-filled.

Next, learn about Work Space history.

Work Space History

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Work Space has the **History** tab that shows a list of all user's submissions. A submission that starts a collaboration job can contain a processing status that is updated as the collaboration job progresses through its steps.

The user can filter the result using a search and selecting all group submissions checkbox.

Next, learn about Work Space reviews.

Work Space Reviews

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Work Space has the **Reviews** tab that allows job coordinators to list all reviews and approvals of collaboration jobs and also search for a particular collaboration job to view the following information:

- The collaboration job's details.
- The current step of the collaboration job.
- The current collaboration job's task and its status.
- The collaboration job's history, including previous submissions and receipts.

To filter or search the collaboration jobs by one or more criteria, specify the following settings and click **Go**:

- Enter a job number, tracking code, or contact email in the search text field.
- Select the period of time from the **Last** dropdown list.

> Only last 90 days are listed by default.

- Select a status of collaboration jobs from the **Status** dropdown list. The default is **In Progress**, so only the review and approval collaboration jobs with the in progress status are listed. To see all collaboration jobs, select the blank status.
- Select a group from the **Job Groups** dropdown list. The default is **[All Groups]**, so all collaboration jobs with all job groups that you are responsible for are listed. Select **[No Groups]** to see collaboration jobs that are not allocated to any job group. This option is required because jobs not allocated to any job group do not appear when [All Groups] is selected. If your company allocates all collaboration jobs to job groups, this option will allow for the changeover period where you can still see jobs before the job groups were introduced.

A user can click on the heading of completed forms to download a task's receipt.

A coordinator can perform one of the following actions for tasks that are **Assigned** or **In Progress**:

- Claim a task
- Claim and open a task
- Assign to and return to the group (if claimed first)

Next, learn about Work Space help desk.

Work Space Help Desk

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Work Space comes with the **Help Desk** tab that allows you to search for forms that have been saved, completed, or abandoned by users.

By default, the Help Desk is not shown to authenticated users and doesn't display any forms. You have to configure the following role permissions in Manager to allow users to use it:

1. Assign the **Work Space Staff** role to the users so they can see the **Help Desk** tab in their work space.
2. Make sure they have the **Help Desk View** permission so they can view the **Help Desk** tab.
3. Assign the **Help Desk Authenticated Edit** permission to the users so they are able to work with customers' transactions.

You can customize the help desk page according to customer's requirements. It is often the case for companies with large transaction volumes, when special configuration needs to be performed to be able to filter form submissions within their form space help desk page:

- The user can filter the help desk list by a date range using the **Last** dropdown. | 18.05
- The user can filter the help desk list by form.
- The user can filter the help desk list by form group, for example, to include authenticated users.
- The Help Desk page doesn't automatically load form submissions when it's displayed first time, so the user can revise the filters first and then click **Go** to fetch the required transactions. This is particularly useful for system with large volumes of transactions.

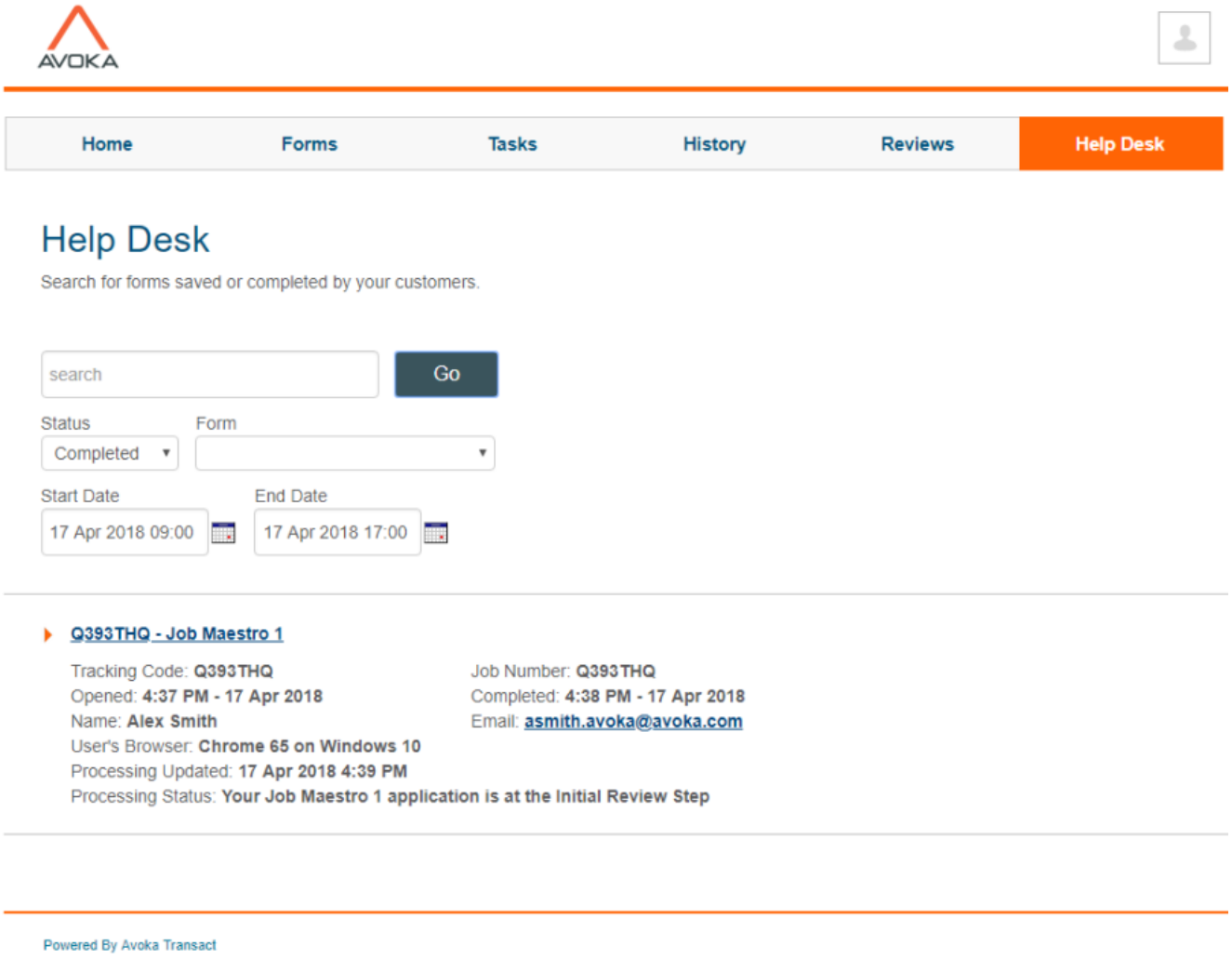
To demonstrate the recommended configuration, consider the following examples for anonymous and authenticated user form submissions.

Anonymous Users Only

Help desk users often check specific transactions and enter their tracking code in the search box and the click **Go**. To limit the volume of displayed transactions, you can use these filtering options:

1. Enter a complete transaction's job number, tracking code or one of the data extract fields in the search text field. By default, any saved forms are listed.
2. Select a transaction status from the **Status** dropdown list to see the forms that were saved, completed or abandoned.
3. Specify a date range from the **Start Date** and **End Date**. A Start Date must be selected before any submissions are displayed.
4. Select a form name from the **Form** dropdown list.

5. Click **Go** to fetch the list of transactions, as shown below.

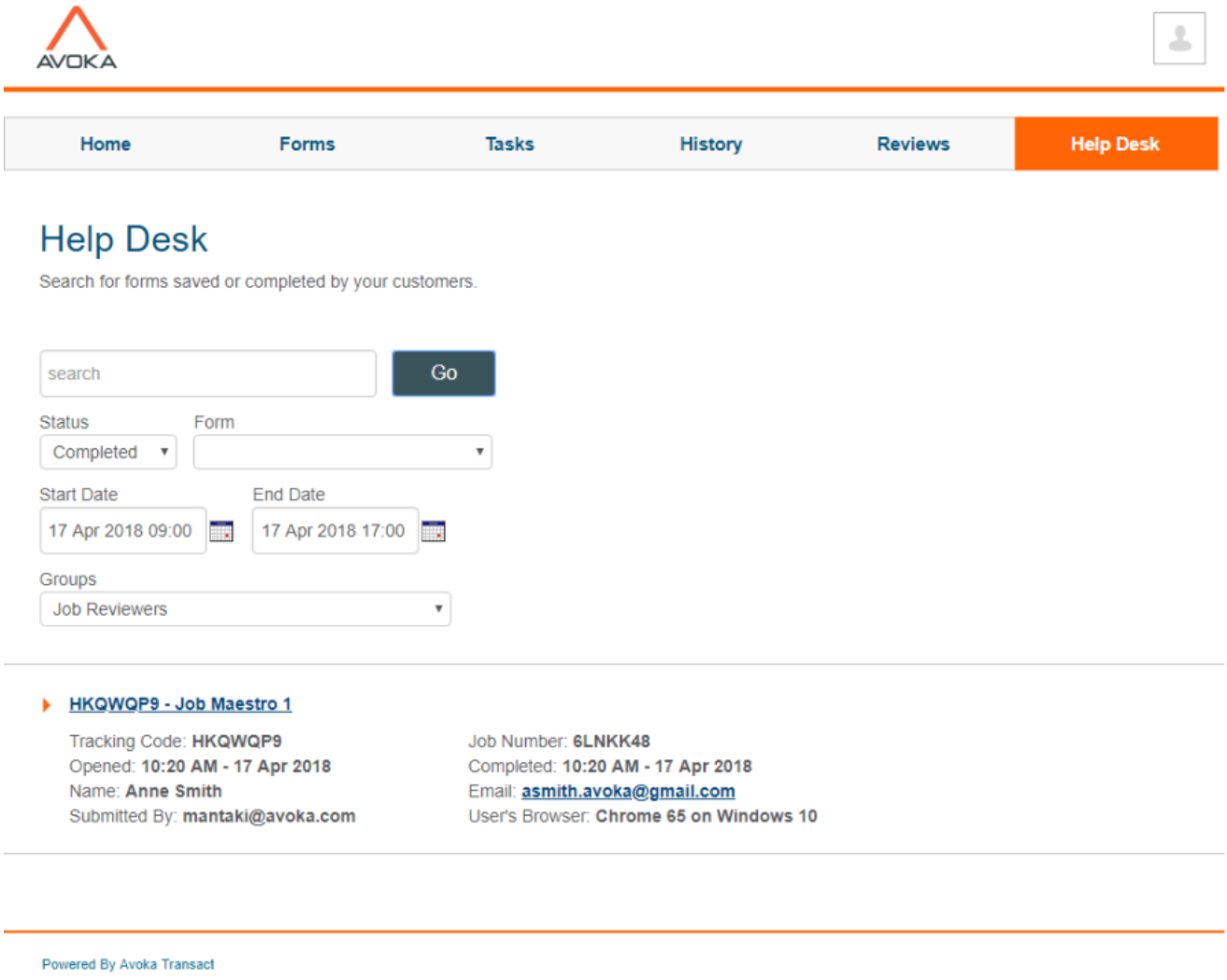


Anonymous and Authenticated Users

You can configure the help desk page to include submissions from authenticated users. This allows help desk users to narrow down their searches with the Groups filter. To limit the volume of displayed transactions, you can use the filtering options explained above with this additional option:

1. Select a group from the **Groups** dropdown list, which has the following options:
   - Blank to search for forms that are not assigned to any group. This is the default option.
   - [All Groups] to search for forms that belong to all form groups.
   - An individual form group name for forms that belong to a specific form group.

2. Click **Go** to fetch the list of transactions, as shown below.



Next, learn how to update account settings.

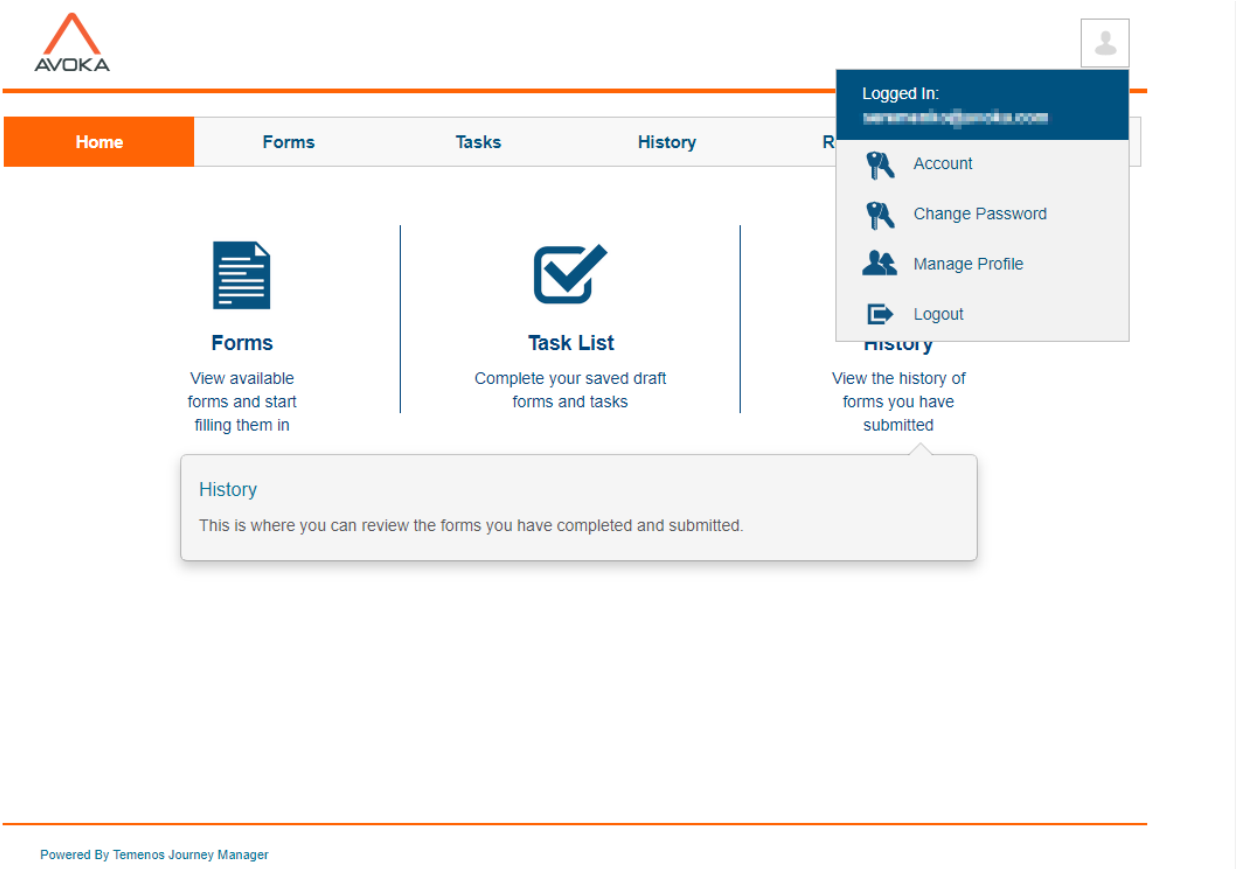
Work Space Account

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

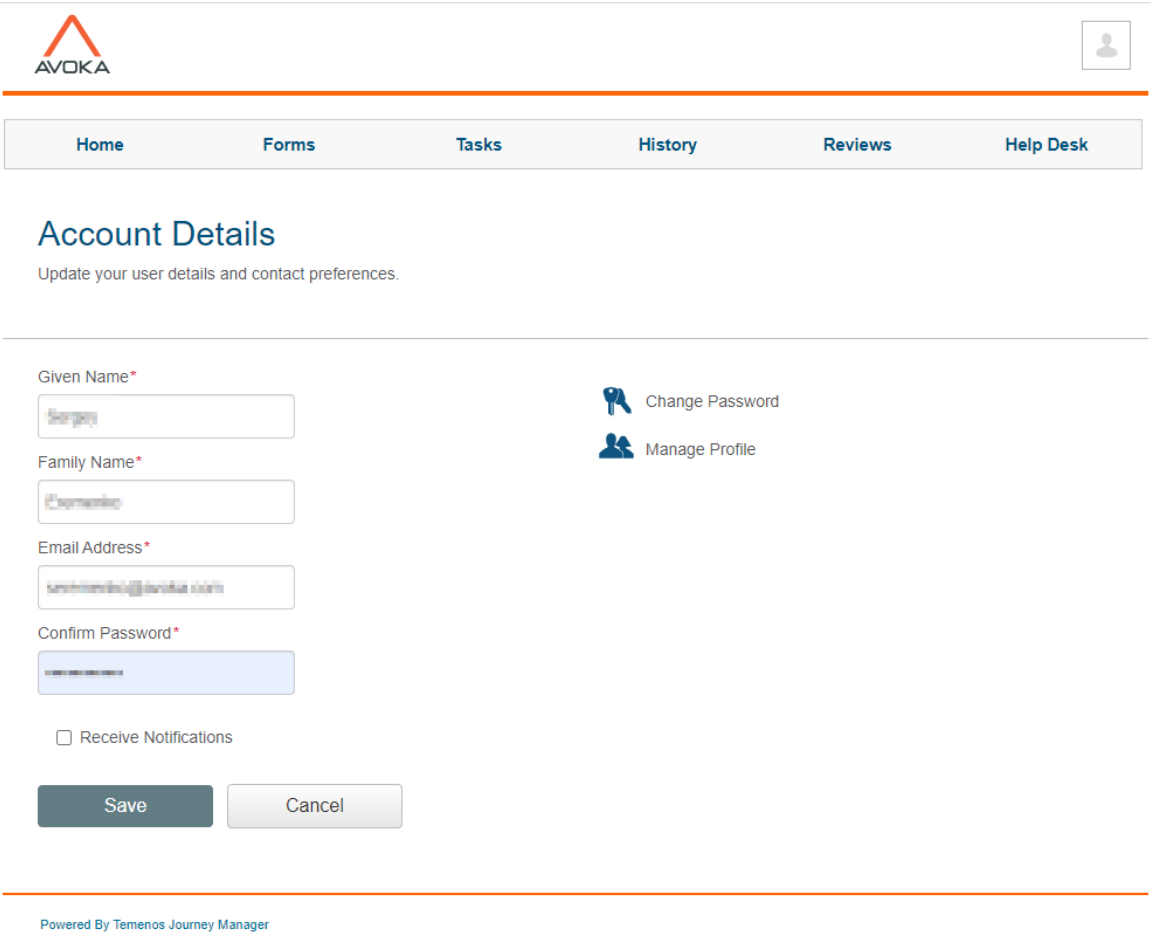
Work Space allows you to manage your user account and profiles as well as change your password.

To configure your account:

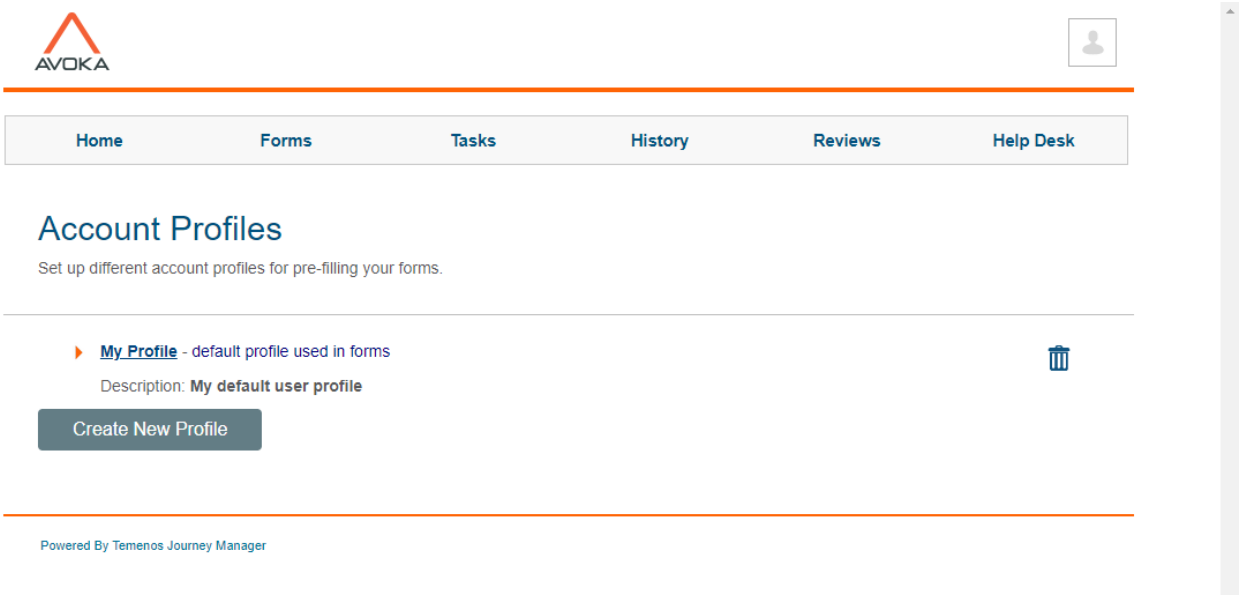
1. Click a profile icon or the **Account** tab.



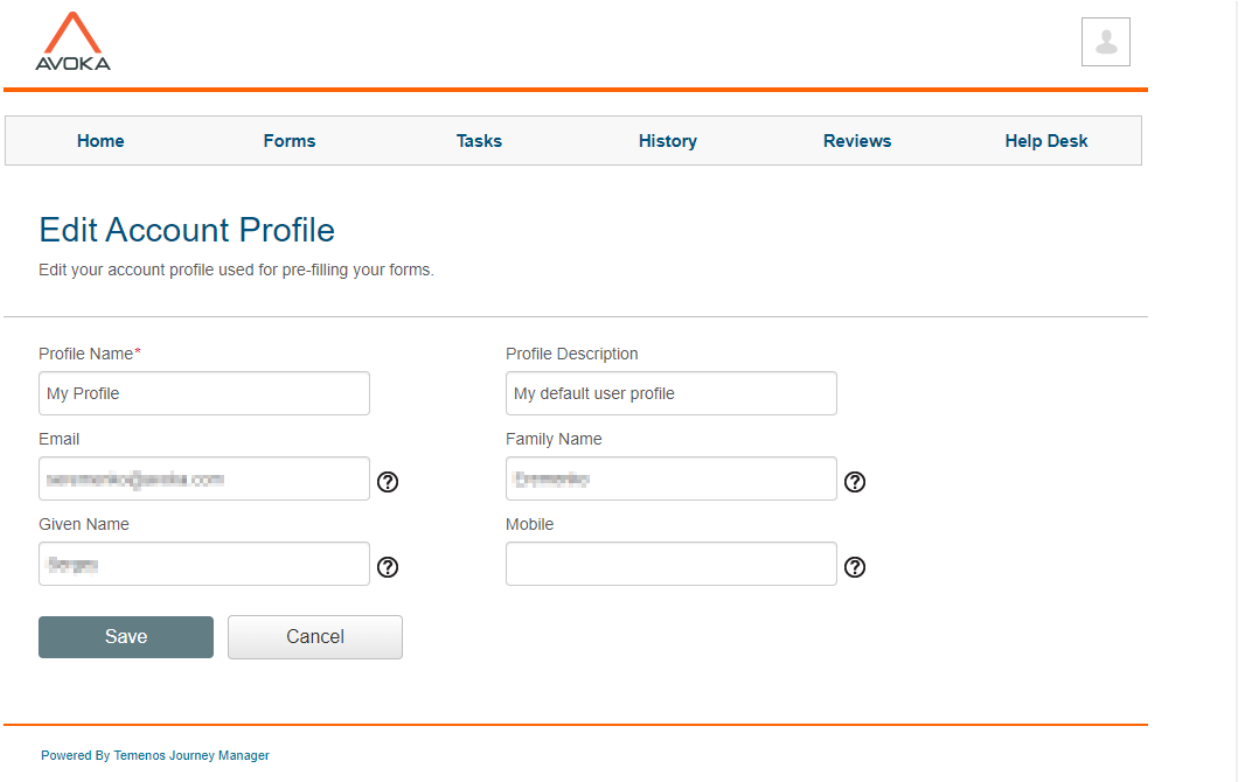
2. Select **Account** to update your details.



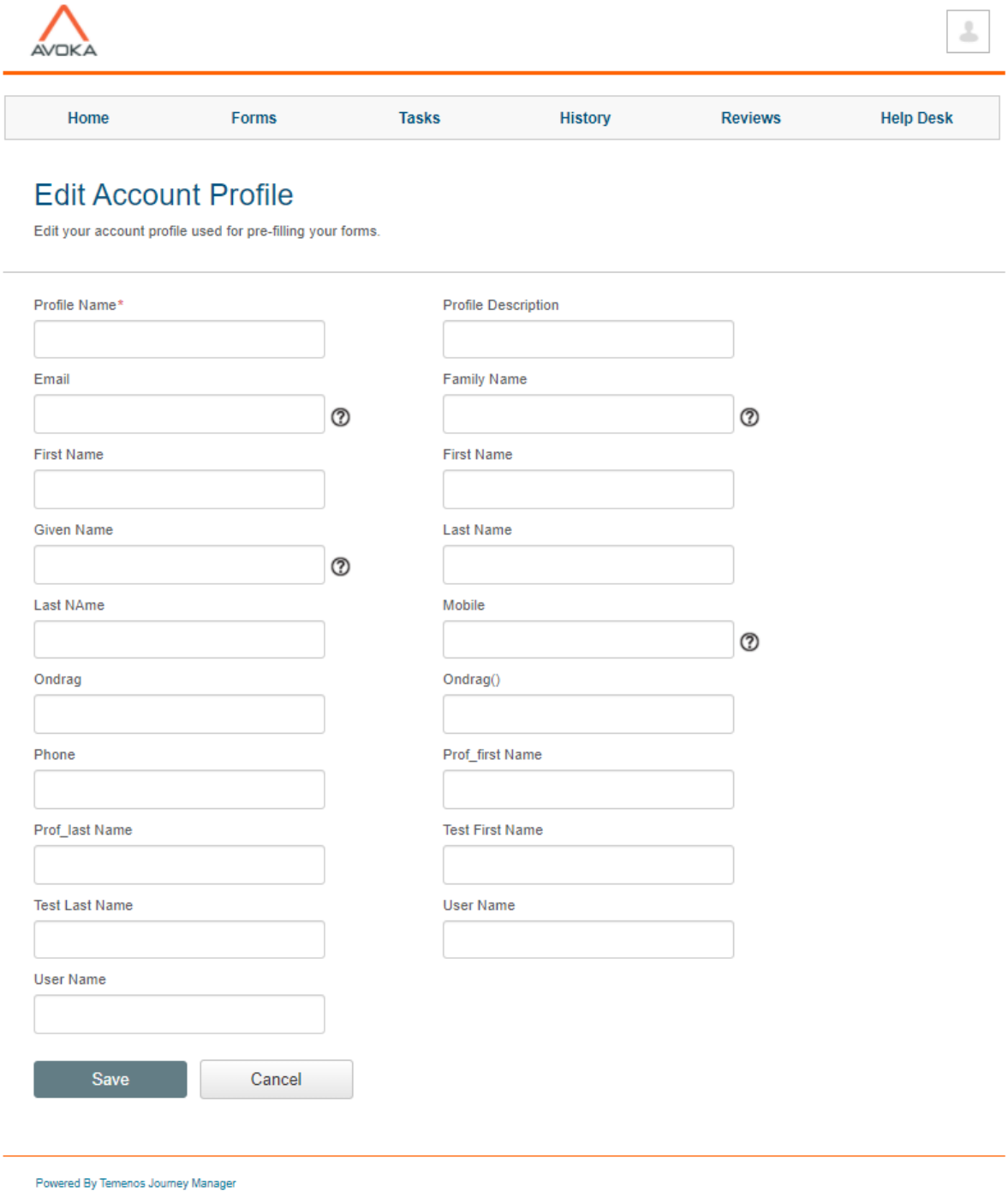
3. Check and update your details and select the **Receive Notifications** checkbox to allow Manager to send you email notification on various activities, such as tasks and reviews.
4. Select **Change Password** to change your password.
5. Select **Manage Profile** to create new or update an existing profile.



6. Select an existing user profile to view or edit it or click **Delete Profile** to remove it.



7. Provide a profile name and description, complete all personal details. Click **Save** to update the changes.
8. Click **Create New Profile** to create a new profile.



9. Provide all required information and click **Save** to create a new profile.
10. Select **Logout** to exit Work Space. You will be redirected to the home page.

Next, learn about the Home tab.

Configure Work Space Properties

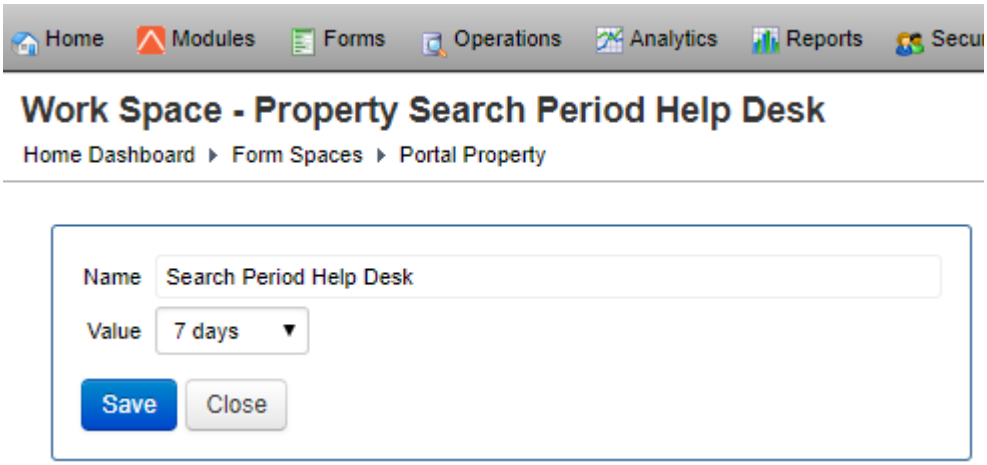
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to configure specific properties of your work space pages.

> You must have an administrator role assigned to your account to perform this configuration.

To configure the page's properties:

1. Select **Forms > Form Spaces**.
2. Locate the **Work Space** and click **Edit**.
3. Select the **Properties Edit** tab to view the following work space related properties:
   - Search Consistent Read
   - Search Data Extracts
   - Search Data Load
   - Search Fetch Limit
   - Search Filter Group Select
   - Search Filter Jobs By User Orgs
   - Search Period Help Desk
   - Search Period History
   - Search Period Reviews
   - Search Period Tasks
   - Search Task Attributes
4. Click **Edit** to and update each property value if required.



5. Click **Save** to update the changes.

Searching and filtering options of the work space also depend on Manager version, individual form space configuration and specific pages' customization. You should consider the following:

- Remove the **Last** dropdown from the page, because of the new, more specific date range, such as Start Date and End Date date pickers. Otherwise, it can become confusing to provide both.
- A form space can be configured in such a way that when switching to the work space page, the form submissions are not initially loaded. The user can select filter options and then click **Go**. This option is particularly helpful for help desk page where performance issues may arise from large volumes of transactions that need to be searched.
- A form space can be configured to allow a user to filter by form groups. This is only relevant to display authenticated form user transactions.

The work space configuration varies for anonymous and authenticated users.

To include the anonymous user transactions only, configure the help desk markup with the following filter options:

- Search box
- Status dropdown
- Form dropdown
- Start Date date picker
- End Date date picker

To include the anonymous and authenticated user transactions:

1. Configure the help desk markup with the following filter options:
   - Search box
   - Status dropdown
   - Form dropdown
   - Start Date date picker
   - End Date date picker
   - Groups dropdown
2. Configure the work space staff user role:
   - Assign the **Work Space Staff** role to the users so they can see the Help Desk tab in their work space.
   - Make sure they have the **Help Desk View** permission so they can view the Help Desk tab.
   - Assign the **Help Desk Authenticated Edit** permission to the users so they are able to work with customers' transactions.
3. Configure a work space property to allow the help desk user to filter by form groups by selecting the **Search Filter Group Select** checkbox. This will display the Groups dropdown list for help desk users that are assigned to at least one form group.

Next, learn how to change work space page markup.

Configure Work Space Page Markup

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to configure specific properties of your work space pages and customize their markup.

> You must be an administrator to configure the work space page.

To customize the work space pages' markup:

1. Remove the **Last** dropdown list by finding and deleting these lines:

```
<span class="searchSelect smallDivider">
 Last
 <br/>
 $form.fields.startDate
 </span>
```

2. Add the **Form** dropdown list and the **Start Date** and **End Date** date pickers by adding these lines:

```
## A new field for form select has been added  <span class="searchSelect smallDi-
vider"> Form <br/> $form.fields.formSelect </span>
## A new field for start date has been added <span class="searchSelect smallDivider">
Start Date <br/> $form.fields.startDateField </span>
## A new field for end date has been added <span class="searchSelect smallDivider">
End Date <br/> $form.fields.endDateField </span>
```

3. Add the **Groups** dropdown list to include the authenticated user forms by adding these lines:

```
#if ($form.fields.groupSelect)
 <span class="searchSelect smallDivider">
 Groups <br/>
 $form.fields.groupSelect
 </span>
 #end
```

The complete example of the help desk page markup is shown below.

```
<style type="text/css">
.searchSelect {
    display: inline-block;
    margin-right:10px;
}
#form_startDate {
    width: 110px;
}
#form_formStatus {
    width: 110px;
}
#form_startDateField {
    width: 120px;
}
#form_endDateField {
    width: 120px;
}
#form_groupSelect {
    width: 330px;
}
#form_formSelect {
    width: 240px;
}
.calendar td {
    min-width: 0px;
}
</style>

## If using the startDateField or endDateField, ensure the two lines below are not com-
mented out
<script src="$context/click/jscalendar/calendar.js" type="text/javascript"></script>
<script src="$context/click/jscalendar/calendar-en.js" charset="UTF-8" type-
e="text/javascript"></script>
<script src="$context/resources/js/searchTable.js" type="text/javascript"></script>
<!-- find-form.htm-->
<div class="row" id="findform">
    <!-- Header -->
    <div class="span12 topSpace divider">
        <h1>Help Desk</h1>
        <p>Search for forms saved or completed by your customers.</p>
    </div>
    <!-- Search panel -->
    <div class="row">
        $form.startTag()
```

```
        <div class="row span6">
            <span class="searchField smallDivider">
                <br>
                $form.fields.search
            </span>
            <span class="searchButton smallDivider">
                $form.fields.go
            </span>
            <span class="searchSelect smallDivider">
                Status
                <br/>
                $form.fields.formStatus
            </span>

## A new field for form select has been added
            <span class="searchSelect smallDivider">
                Form
                <br/>
                $form.fields.formSelect
            </span>

## A new field for start date has been added
            <span class="searchSelect smallDivider">
                Start Date
                <br/>
                $form.fields.startDateField
            </span>

## A new field for end date has been added
            <span class="searchSelect smallDivider">
                End Date
                <br/>
                $form.fields.endDateField
            </span>

## New fields for group select and group filter have been added
            #if ($form.fields.groupSelect)
            <span class="searchSelect smallDivider">
                Groups <br/>
                $form.fields.groupSelect
            </span>
            #end
            #if ($form.fields.groupFilter)
            <span class="searchSelect smallDivider">
```

```
                Group Items <br/>
                $form.fields.groupFilter
            </span>
            #end

        </div>
        $form.endTag()
    </div>
</div>
<div class="row"><hr></div>

## See template in: /resources/includes/account/tile.htm
<div id="tableHolder" class="row">
    $tileList
</div>
```

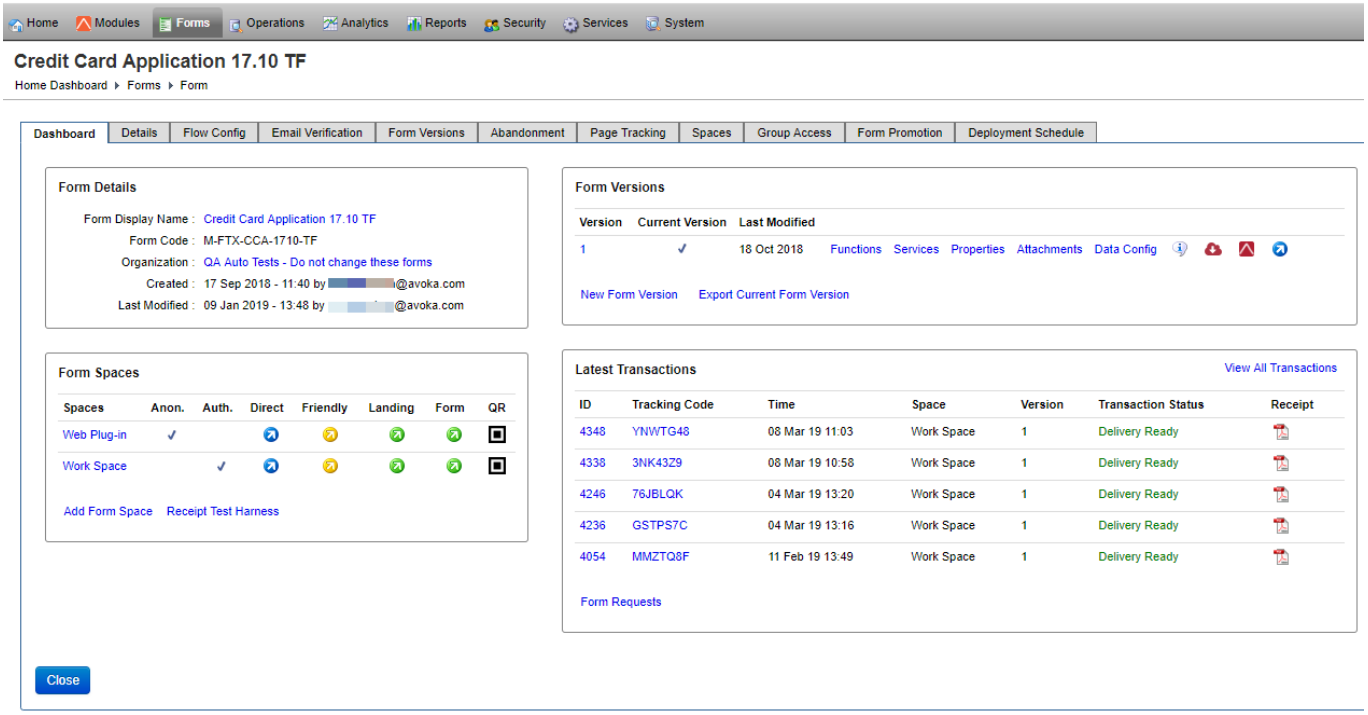
Next, learn how to configure work space page properties.

Form Dashboard

Journey Manager (JM) | Platform Developer | 17.10

Manager provides a Form Dashboard where you can quickly see form's configuration in a glance.

To navigate to the Form Dashboard, locate the form and click **Edit**.



The Dashboard consists of the following sections:

- Form Details
- Form Versions
- Form Spaces
- Latest Transactions

Form Details

This section displays basic form details, such as the form's display name, a form code, and its organization.

Form Versions

This section lists all available versions of this form. The current version of the form, which is the version that will be rendered when users request the form (unless A/B testing is enabled), is identified by a check-mark.

You can perform the following actions from this section of the Dashboard:

- Click a specific form version to see its configuration
- Click **Make Current** to make this form version the current form version
- Click **Functions** to list functions configured for this form version | 17.10

> The Functions link only appears when the **Use Transact Functions** checkbox is selected from the Build Options in Maestro. | 17.10

- Click **Services** to list services associated with this form version.
- Click **Properties** to list the properties configured for this form version.
- Click **Attachment** to list the attachments and their rules configured for this form version.
- Click **Data Config** to show data options configured for this form version. The options include form XML data, property prefill mapping, request parameter prefill mapping, and form data extract mapping.
- Click **Form Design**  to displays the form design and build audit details.
- Click **Update Design from Maestro**  to update the selected form version based on any changes made to the original form version in Maestro. Only the same user who updated the selected form version last time can use this button to update the form. All other users will see this error message: **Could not load latest published Maestro Form Design Version - error: 401:Unauthorized**. If there have been no changes to the form, the following message is displayed: **No Maestro Form Design changes to update**.

> This icon only appears if the form has been designed in Maestro.

> In Manager 5.1, this icon was 

- Click **Open Form Design**  to open the related form version in Maestro. This icon only appears if the form was designed in Maestro. | 17.10
- Click **From Test Render**  to render this form version in the default form space.
- Click **New Form Version** to create a new version of this form.
- Click **Export Current Form Version** to export the current form version (version 3 in this case) on your local PC.

Form Spaces

This section lists form spaces where the form is hosted on and authentication mode, **Anon.** or **Auth.**, users are required to access the form. If the form is configured with the **Auth.** mode only, users will need to log in first to access the form.

You can navigate to the hosted form by using one of the following links:

- Direct to this form via the selected Form Space's URL.
- Friendly to this form's landing page using a user-friendly URL format. You can use this functionality to test the form.
- Landing to this form's landing page. This is the page where users can log in. You can use this functionality to test the form.
- Form to the form page in the selected form space where it will be rendered.
- QR Code to this form's **QR code**[1] to scan this QR code using a QR scanner app.
- Click **Add Form Space** to assign a form space to this form.
- Click **Receipt Test Harness** to generate and test a receipt for this form as a PDF or HTML file.

Prior to version 17.10, this section was named **Form URLs**.

Latest Transactions

This section lists all latest transactions of user submissions of this form.

You can perform the following actions from this section of the Dashboard:

- Click **ID** or **Tracking Code** to see the transaction details
- Click **View PDF Receipt** to view and download a PDF version of this form
- Click **Form Requests** to view all form requests or search for specific form requests

Next, learn about form details.

[1] QR Code (Quick Response Code) is a machine-readable code consisting of an array of black and white squares, typically used for storing URLs or other information for reading by the camera on a smartphone.

Deploy Forms

Journey Manager (JM) | Form Builder System Manager / DevOps | 5.1 & Higher

The Journey platform allows you to deploy forms created in various tools. Maestro is our preferred tool to create your forms and components.

To deploy a Maestro form to Manager:

1. Build a form in Maestro.
2. Download the built form as a form version ZIP file from Maestro.
3. Import the form version ZIP file into Manager.
4. Configure the form in Manager, for example, delivery channels, receipts and data retention.

After the form is deployed into Manager, you can still update it in Maestro and sync up the form changes by clicking **Update Design from Maestro** from the form dashboard.

Caching of resources in deployed forms is straight forward with Manager, all resources bundles with the deployed form in the form archive are cachable but only in the context of the specified form. This means that the first time a user loads a form version they will have to download all the relevant bundled resources referenced by the form, excluding any external resources. This is to prevent issues with deploying a form that contains a new image or JavaScript file and due to browser caching getting the old version as it is still valid according to the cache. This is done by using the `<base href=""/>` element in the HTML `<head>`. Republishing a Maestro form of the same version will result in a different value being used for this base hef.

> The initial release of Maestro was generally available in August 2016, along with Manager 5.0, which is the first fully supported version of Manager. Publishing forms to any version prior to Manager version 5.0 is not supported. Many features may work but troubleshooting and support will not be provided for Manager versions prior to 5.0.

Next, learn how to view forms.

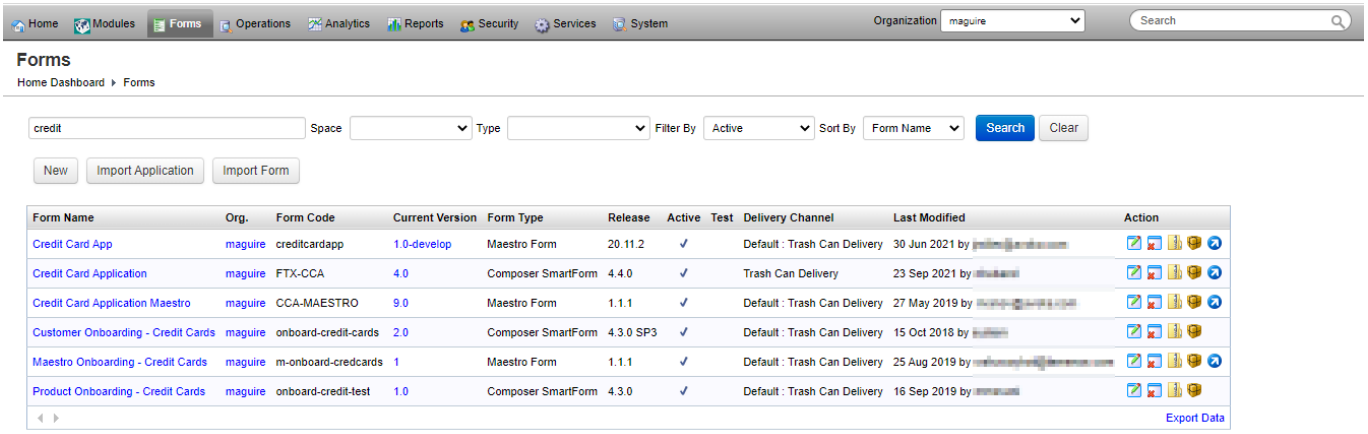
View Forms

Journey Manager (JM) | Platform Developer | 21.11

Manager allows you to view all available forms, so you can easily find, manage, and update them.

To view all forms, select **Forms > Forms**.

Forms are shown page by page, with the maximum number of records per page configured in the users preferences.



To filter or search the forms by one or more criteria, specify the following settings and click **Search**:

- Organization
- Form name or part of a name
- Form space
- Form type

  A form type can be one of the following:

- Composer SmartForm
- Maestro Form
- JS App - refers to an Open UX form
- Dynamic PDF Form
- Static PDF Form
- Other Document
- Status. A form status can be one of the following: | 21.11
- Active Non-Test - it's an active form except that of in Test Mode.
- Active - it is in use and readily available to all users. All functionality is allowed.
- Offline - it can't be used at all. No functionality is available, including DDS, Delivery and so on
- Inactive - similar to Offline.
- NoRender - it doesn't allow for new or saved and resumed forms, but ongoing open sessions can be completed. DDS, submits, saves, and delivery are allowed.

To change the form list order, select one of the following from the **Sort By** dropdown list:

- Form Name
- Created By
- Created Date
- Modified By
- Modified Date

Click **Clear** to reset the search criteria.

You can also perform the following actions:

- Create a form
- Configure a form
- Remove a form
- Export forms
- Render a form
- Import forms
- Import application

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to configure a form.

Configure a Form

Journey Manager (JM) | Platform Developer | 5.1 & Higher

After you have built a form in Maestro, you have to configure it in Manager.

To configure a form in Manager:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit** to display the form's dashboard.
3. Details - general options including delivery channels.
4. Flow Configuration - form flow options.
5. Email Verification - email verification required for this form.
6. Abandonment - abandonment settings to override the organization's abandonment settings.
7. Page Tracking - third-party page tracking and analytics.
8. Assign form spaces that this form is hosted on.
9. Group Access - enlist groups that can access this form in case it has some restrictions.

Next, learn how to remove a form.

Configure Form Details

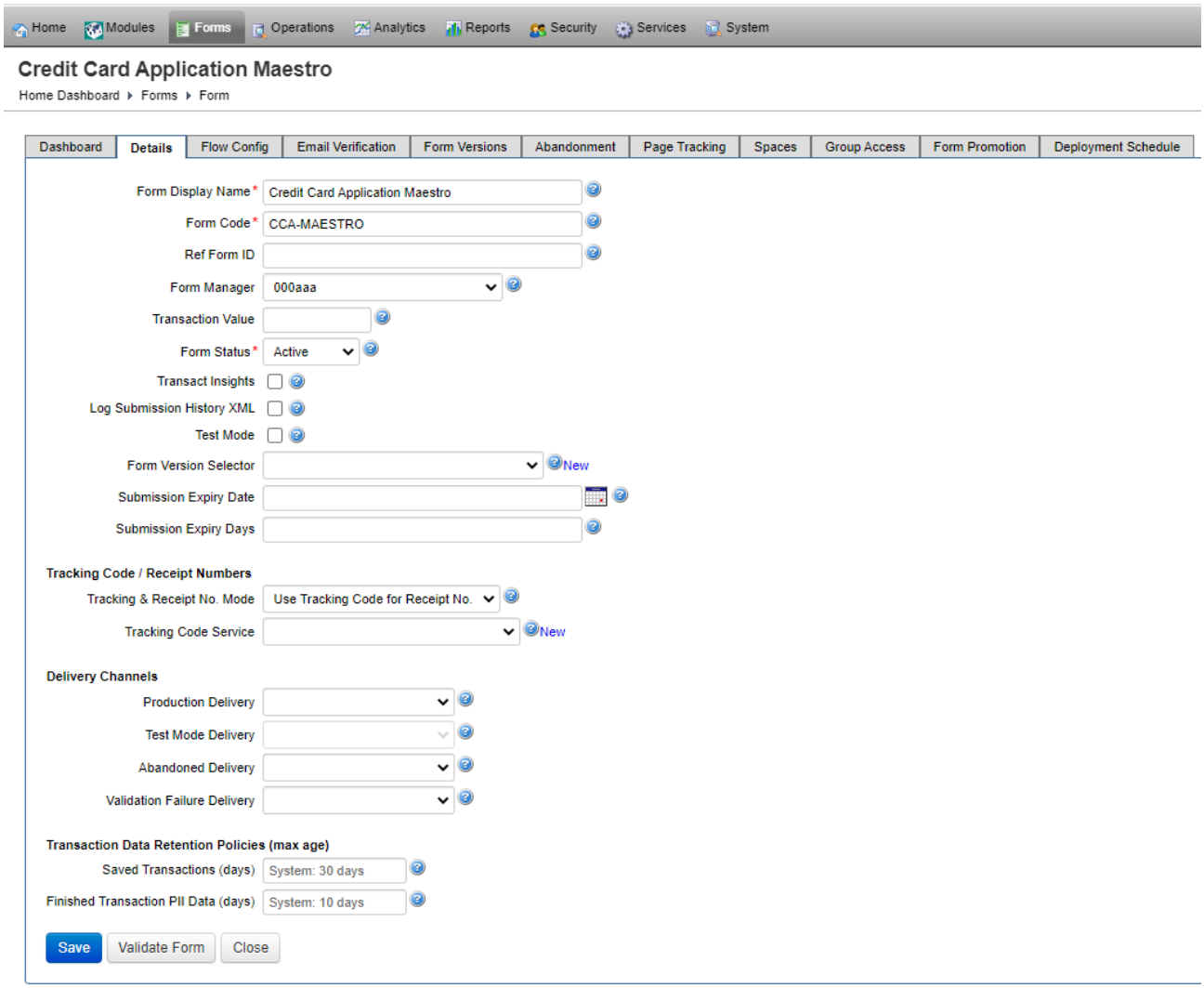
Journey Manager (JM) | Platform Developer | 21.11

Manager allows you to fine-tune a form by configure the following details:

- General form information
- Tracking codes or receipt numbers
- Delivery channels
- Transaction data retention policies

To configure form details:

1. Select **Forms > Forms**.
2. Locate a form and click the **Details** tab to edit the form configuration.



3. Edit the name of the form that is displayed to the user in the **Form Display Name** field. The name uniquely identifies the form within a specific Organization. It will also be used in the long form URL (Form Friendly URL).
4. Edit a form code in the **Form Code** field. This is a short alphanumeric string, which uniquely identifies the form across all organizations. It is generally not displayed to users.
5. Edit a ref form ID in the **Ref Form ID** field. This is the identifier used internally by the Organization and will be passed through during various web service calls such as prefill and receipt generation. It is optional, only used if there is an identifier that is used internally by the Organization that the form version is associated with.
6. Select a form manager from the **Form Manager** dropdown, who will be notified via email whenever the form's status changes or a form version is promoted. The SmartForm Manager user is associated with a form's organization.
7. Provide the form's transaction value to the business in dollars in the **Transaction Value** field. This value is used for reporting purposes.
8. Select a status from the **Form Version Status** dropdown list. A form version status can be one of the following: | 21.11
   - Active - it is in use and readily available to all users. All functionality is allowed.
   - Offline - it can't be used at all. No functionality is available, including DDS, Delivery and so on
   - Inactive - similar to Offline.
   - NoRender - it doesn't allow for new or saved and resumed forms, but ongoing open sessions can be completed. DDS, submits, saves, and delivery are allowed.

   Select the **Inactive** checkbox to retire a form without deleting it.

9. Select the **Transact Insights** checkbox to enable this form with Journey Analytics. For more information, see how to enable applications to collect analytics
10. Select the **Log Submission History XML** checkbox to log all form XML changes so you can view transaction history and transaction XML difference. This is handy for transaction troubleshooting. If a form uses Transact functions, this option creates a new submission history record for the `formUpdate` operation.

    Excessive logging can degrade performance of Manager applications with high throughput, so ensure that the form doesn't have the **Log Submission History XML** selected. Otherwise, every time the form does a formUpdate or background save operation, Manager stores a full copy of the submission XML which can result in large growth of the submission history XML data table on high throughput systems with modern form applications .

11. **Form Version Selector** allows you to select the service responsible for selecting the version of the form that will be rendered at request time. If left blank, the system default Form Version Selector service is used. Normally this should be Current Form Version service.

12. Select how to generate tracking and receipt codes from the **Tracking & Receipt No. Mode** dropdown list. You have the following options:
    - Use Tracking Code for Receipt No
    - Use Receipt Number for Tracking Code

    We recommend using Tracking Code to improve performance and scalability. For more information, see configure receipt numbers and tracking codes.

13. Select a service to generate form tracking codes from the **Tracking Code Service** dropdown list. You have the following options:
    - Blank - the default Tracking Code service is used
    - Fluent Tracking Number
    - Random Tracking Number
14. Select a delivery channel from the **Production Delivery** dropdown list to send successful submissions through.
15. Select a delivery channel from the **Test Mode Delivery** dropdown to send successful test submissions through.

    This dropdown is enabled if the form is in test mode, that is the **Test Mode** checkbox is selected.

16. Select a delivery channel from the **Abandoned Delivery** dropdown list to send abandoned submissions through.
17. Select a delivery channel from the **Validation Failure Delivery** dropdown list to send submissions with data validation errors through.
18. Select the maximum age of user saved transactions, after which they will be automatically abandoned, from the **Saved Transactions (days)** dropdown list.
19. Select the amount of time to keep PII transaction data for after the transaction has finished from the **Finished Transaction PII Data (days)** dropdown list.

    For more information on data retention configuration, see how to configure form data retention.

20. Click **Validate Form** to check whether the form is active, has no invalid versions or expired dates and a few other parameters using values defined in the **Test Import Tag** metadata tag.
21. Click **Save** to update the changes.

Next, you may want to configure form flow.

Configure Form Abandonment

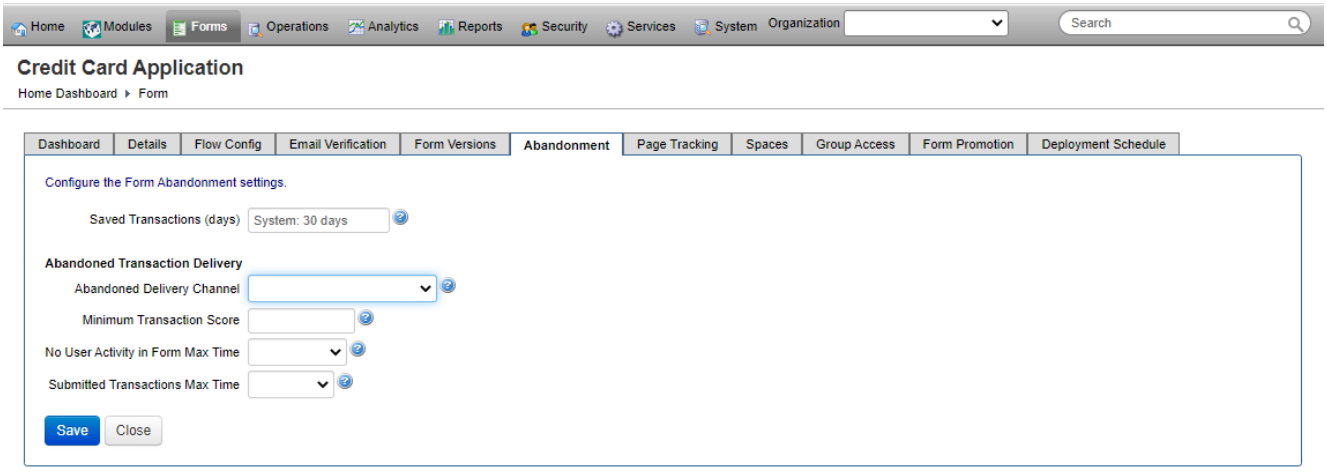
Journey Manager (JM) | Platform Developer | 5.1 & Higher

Transactions can become abandoned based on the user actions, such as canceling a transaction, or based on the form's abandonment settings. You can use abandonment configuration to define when a form is considered as **abandoned**[1] after it has been saved, and what happens after a form transaction is abandoned. Manager allows you to configure abandonment rules of form's transactions at a global, organization, or form level. These configurations depend on:

- Customer requirements
- Server performance
- Specific data retention policies

To configure form abandonment:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Click the **Abandonment** tab.



4. Specify the maximum age of user saved transactions after which they will be automatically abandoned (marked as Abandoned) in the **Saved Transaction (days)** field. The default is 30 days, which is either the organization data retention setting, if exists, or the system data retention setting. When a saved transaction reaches this maximum age, it is marked as Abandoned by the system. The maximum age is based on the number of days a form has been untouched since it was last saved.
5. Select the organization delivery channel for delivering abandoned submissions from the **Abandoned Delivery Channel** dropdown list. The next three fields are enabled when a delivery channel is selected.

[1]Any transaction that has been deemed as Abandoned in Manager. When a user leaves the application without explicitly canceling it, the transaction lives in Started status for a period defined in Manager, after which it is considered as an Abandoned transaction.

Manager version 17.10 and higher has the following fields available for Composer forms only. However they are still displayed for Maestro forms.

6. Edit the **Minimum Transaction Score** to specify the transaction score that a form transaction must reach for it to be delivered to the Abandoned Delivery Channel. A transaction score is a numerical value calculated by the form, generally based on how much of the form was filled in. When left blank, all abandoned form transactions are delivered. Otherwise, only those that have been “adequately” filled in, meaning they have reached the Minimum Transaction Score, are delivered. Minimum Transaction Score specifies a minimum value representing how much of the form must be filled in for it to be delivered to the abandoned delivery channel. That is, a transaction score is a numerical value calculated by the form to indicate how much of the form was filled in. If you would like only submissions that exceed a certain transaction score to be delivered, enter the minimum transaction score into this field.
7. Select a time from the **No User Activity in Form Max Time** dropdown list to specify a time frame for allowing any form transaction by an anonymous user to remain untouched before marking it as abandoned. This is the period of time after which anonymous transactions are marked as abandoned. These transactions have been saved in the background and untouched by a user.

A form transaction must also have been background saved for this setting to apply.

8. Select a time from the **Submitted Transactions Max Time** dropdown list to specify a time frame for allowing any form transaction that has been submitted by a user to remain incomplete before marking it as abandoned. This is the maximum age before a submitted but incomplete, for example missing attachments, transaction are marked as abandoned.

A form transaction may be incomplete because it’s waiting for a post-submission requirement such as an attachment to be provided by the user.

9. Click **Save** to update the changes.

You can view abandoned transactions to troubleshoot user form interactions.

Next, learn hot to configure form details.

Configure Form Email Verification

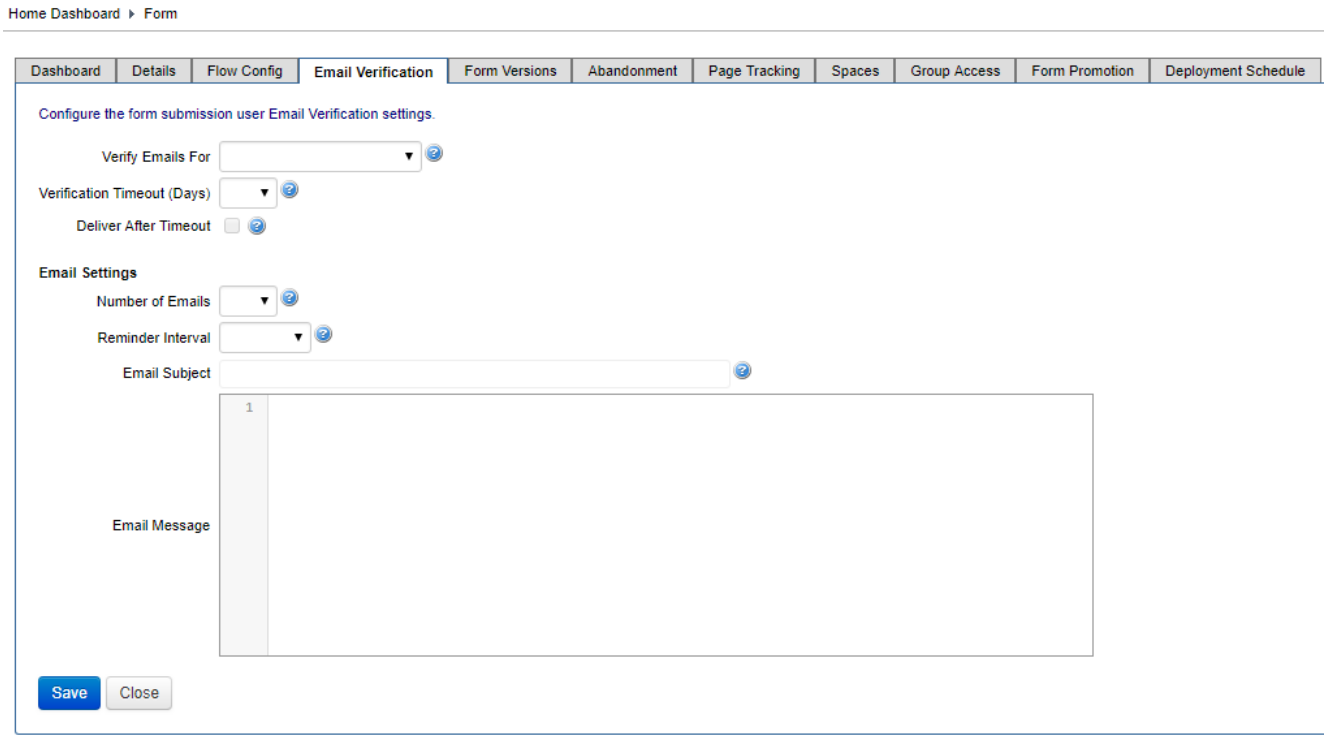
Journey Manager (JM) | Platform Developer | 5.1 & Higher

A verification email is used to confirm the email address of a form user. Manager allows you to configure these verification emails for selected forms. Once configured, these emails are sent when a user submits a form. Commonly, a verification email includes a link that the user must click to verify their email address. A form does not send a verification email by default, so you have to enable this option manually.

> For submission email verification to work, a contact email address must be stored for the submission. For more information, see configuration mapping.

To configure a form with a verification email:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Click the **Email Verification** tab.



4. Select **Verify Emails For** to specify when to send an Email Verification to the form user.
   - **Blank** - to disable email verifications. This is the default option.
   - **Anonymous Submissions** - to enable email verification for submissions from an anonymous user only.
   - **All Submissions** - to enable email verification for submissions from all users.

If the Email Verification is selected, you can configure the following options.

5. Select **Verification Timeout** to specify how many days (at maximum) you want to give a user to verify their email address. After verification timeout has been reached, the form submission will be abandoned by default unless opting to deliver submission anyway (Deliver After Timeout is selected).
6. Select **Deliver After Timeout** to indicate once a user misses the due date as per the Verification Timeout, whether to deliver the submission anyway (selected) or to abandon the submission (unselected is the default).

Select the Deliver After Timeout checkbox only when email verification is not mandatory.

7. Select **Number of Emails** to specify how many emails in total will be sent to a user as a reminder to verify their email address.
8. Select **Reminder Interval** to specify the time (hours or days) to wait before sending each reminder email. For example, if Number of Emails is 3 and Reminder Interval is 7 days, then a user that hasn't verified their email address will receive up to 3 verification emails every week.
9. Edit **Email Subject** to specify the subject text of the Verification Email. It is predefined as Please Verify Your Email.
10. Edit **Email Message** to specify the message defined using basic HTML mark-up. It is predefined with a message. When making changes to the Email Message HTML, ensure the message always contains the URL that the user needs to access to complete submission email verification (the placeholder is **`${verificationUrl}`**).
11. Click **Save** to update the changes.

Next, learn about form versions.

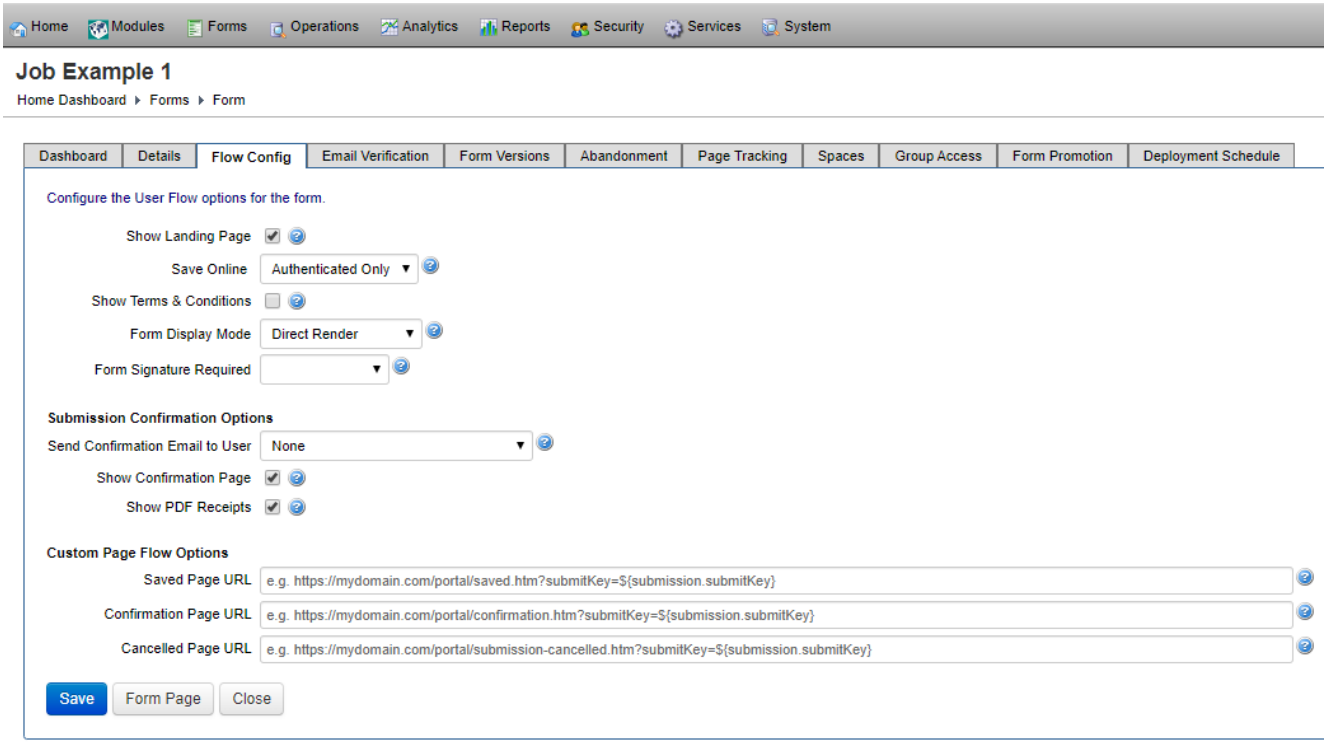
Configure Form Flow

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to customize several options for the flow of a form. The flow of a form includes events, for example, user redirect to a custom URL when resuming a saved form.

To configure a form flow:

1. Select **Forms > Forms**.
2. Locate a form and select the **Flow Config** tab.



3. Select an option from the **Send Confirmation Email to User** dropdown to determine whether a confirmation email should be automatically sent to a user who submits the form and what is to be included.

- **None** to never send a confirmation email.
- **Confirmation** to send the confirmation email only.
- **Confirmation and Receipt** attachment to send the confirmation email along with an attached PDF receipt of the form.
- **Confirmation and Receipt Link** will send the confirmation email along with a secure URL for the form user to download his/her PDF form receipt.

Form submission configuration options are also available in Maestro by selecting **Form Options > User Experience > Auto-Send Confirmation Emails**. See Maestro form options for more details.

4. **Show PDF Receipts –** By default, this option is selected to offer PDF receipts to a user after submission. This may be unselected if you do not want this to occur.

Manager handles the form flow when a user saves, submits or cancels a form. However, if necessary, it is possible to redirect the user to a custom location for each of the following by specifying the custom URL:

5. **Saved Page URL** - Enter a custom URL, if you wish to redirect the user after saving his/her form entry.
6. **Confirmation Page URL** - Enter a custom URL, if you wish to redirect the user after submitting his/her form entry.
7. **Cancelled Page URL** - Enter a custom URL, if you wish to redirect the user after cancelling his/her form entry.

   You may use the following Velocity variables when constructing your custom URLs for the above 3 fields:

- **$submission** The submission object. For example, **`${submission.submitKey}`** would be substituted with the submit key (a GUID).
- **$formDataMap** The map of submission data extract values. For example, **`${formDataMap.firstName}`** would be substituted by the value in the map for the key "firstName".
- **$request** The HTTP servlet request.

Manager supports legacy Composer SmartForms. When configuring the Flow of a Composer SmartForm, several additional options are made available from the **Flow Config** tab.

For a Composer's SmartForm, the following additional options can be configured.

8. **Show Landing Page** – Selecting this option will display the Form's landing page. If this checkbox is not selected, the form's landing page will not be displayed, and the form user will be directed straight to the rendered form.
9. **Save Online** – This dropdown menu allows you to decide whether a form can be saved online to be completed later and whether this option should only be made available to authenticated users. Select one of the following options from the Save Online dropdown:
   - Disabled – Select this option to disable the save online feature.
   - Enabled – Select this option to enable the save online feature.
   - Authenticated Only – Select this option to enable the save online feature for authenticated users only.
10. **Show Terms and Conditions** – Select this checkbox to display a Terms and Conditions page before the form user can access the form.
11. **Form Display Mode** – This dropdown menu determines how the form should be displayed. Select one of the following options from this drop down.

- Direct Render – Select this option to display the form directly from the servlet.
- Embed with Div – Select this option to display the Form inside a DIV (a div defines a section of a page).
- Embed with iFrame – Select this option to display the form inside of an iFrame

12. **Form Signature Required** – This dropdown determines if a signature is required at submission time. Select one of the following options from the dropdown menu.
    - Wet Signature - To ask users to physically sign forms
    - eSignature - To ask for an 'electronic' signature by requesting that a user confirm a receipt presented to them after submission.
13. **Show Confirmation Page** – Select this checkbox to display the space's confirmation page when a user has successfully submitted the form.
14. Click **Save** to update the changes.
15. Click **Form Page** to test how the form is rendered when accessed via the form's landing page.

Next, learn how to configure email verification.

Configure Form Group Access

Journey Manager (JM) | Platform Developer | 5.1 & Higher

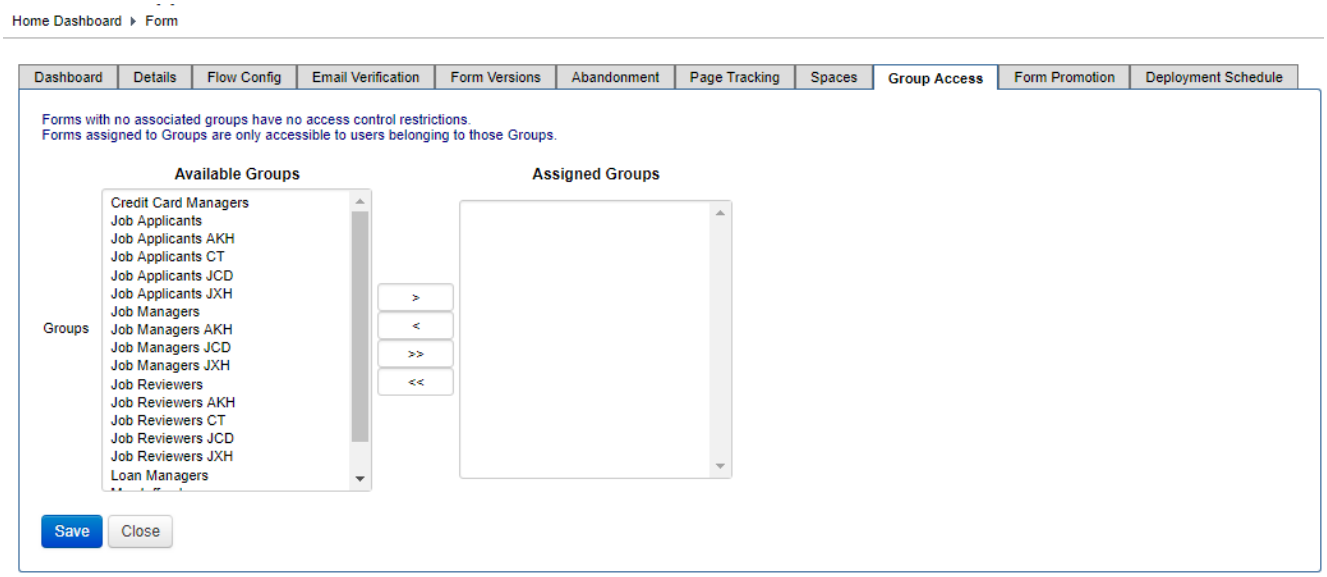
Manager allows you to restrict access to a form, so users belonging to the assigned groups can view and select this form when logged into a form space. Forms with no associated groups have no access control restrictions.

> A farm is restricted when it is assigned to at least one group.

In most cases, forms are unrestricted and available to all users accessing the form space, so for a typical form, the Group Access tab should not have any assigned groups.

To assign groups to a form:

1. Select **Forms > Forms**.
2. Locate a form.
3. Select the **Group Access** tab.



4. Select one or more groups from the Available Groups list that you want to assign to the form (on Windows, use the Shift and Ctrl keys to select multiple groups).
5. Click **>** to move the selected groups from **Available Groups** to **Assigned Groups**.
6. To revoke any assigned groups, select the groups from Assigned Groups and click **<.**
7. To assign all groups, click **>>.**
8. To revoke all groups, click **<<.**
9. Click **Save** to update the changes.

A form's assigned groups can be changed at any stage.

Next, learn how to promote a form.

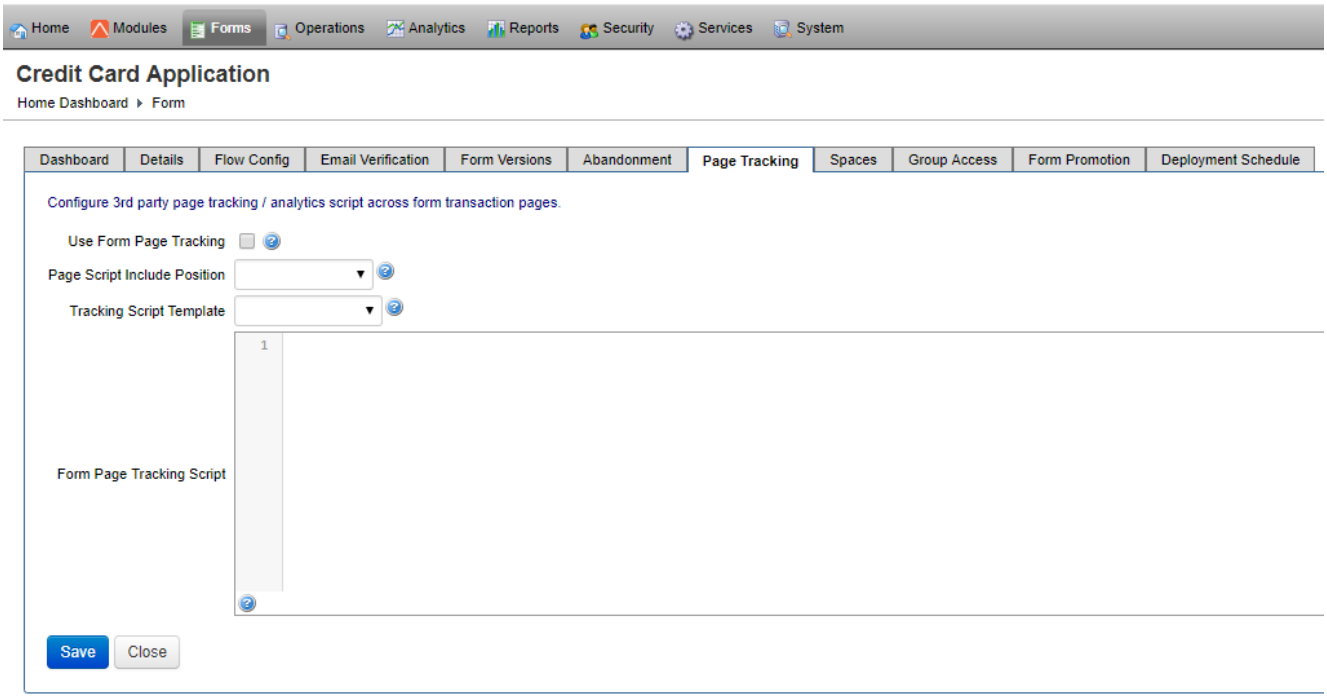
Configure Form Page Tracking

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to add third-party page tracking and analytics tags to a form using Google Analytics, Marketo and NetResults template scripts. However, you can also add any other page tracking as a custom script.

To configure page tracking for a form:

1. Select **Forms > Forms**
2. Locate a form and click **Edit**.
3. Select the **Page Tracking** tab.



4. Select the **Use Form Page Tracking** checkbox if you want this form to use third-party page tracking.
5. Select an HTML page script from the **Page Script Include Position** dropdown list to specify the page script in relation to the HTML mark-up, which can be one of the followings:
   - Inside <HEAD> - to place the script inside the head of the HTML mark-up.
   - After <HTML> - to place the script outside or after the HTML mark-up.
6. Select **Tracking Script Template** to specify the third-party page tracking system template that you use for this form. The selected template populates the **Form Page Tracking Script** text box below with the related script for this template.

- **Google Analytics**[1] - replace 'UA-XXXXX-X' with your Google Analytics account ID.
- Marketo - replace 'MUNCHKIN_ACCOUNT_ID' with your Marketo account ID.
- NetResults - use as is.
- Blank - for a custom script where you provide a script in the **Form Page Tracking Script** text box.

7. Add the third-party script to use for the page tracking in the **Form Page Tracking Script** field. The example below shows the Google Analytics scrip:

```
<!-- Google Tag Manager -->
<noscript/noscript>
< script >(function(w,d,s,l,i){...});</script>
<!-- End Google Tag Manager -->
```

8. Click **Save** to update the changes.
9. Configure the form's Analytics options in the Maestro editor.

For more information on Google Tag Manager, see How to Add Google Tag Manager Support.

Next, learn how to assign form spaces to a form.

[1]Google Tag Manager is a tag management system (TMS) that allows you to quickly and easily update tracking codes and related code fragments collectively known as tags on your website or mobile app.

Configure Form Test Mode

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager provides you with the Form Test Mode configuration so you can test a form before it is deployed in production where form applicants will access it. If you do not have multiple environments, such as test and UAT, the Form Test Mode is a great way of artificiality creating a software development cycle for a form, so developers can test a form's functionality.

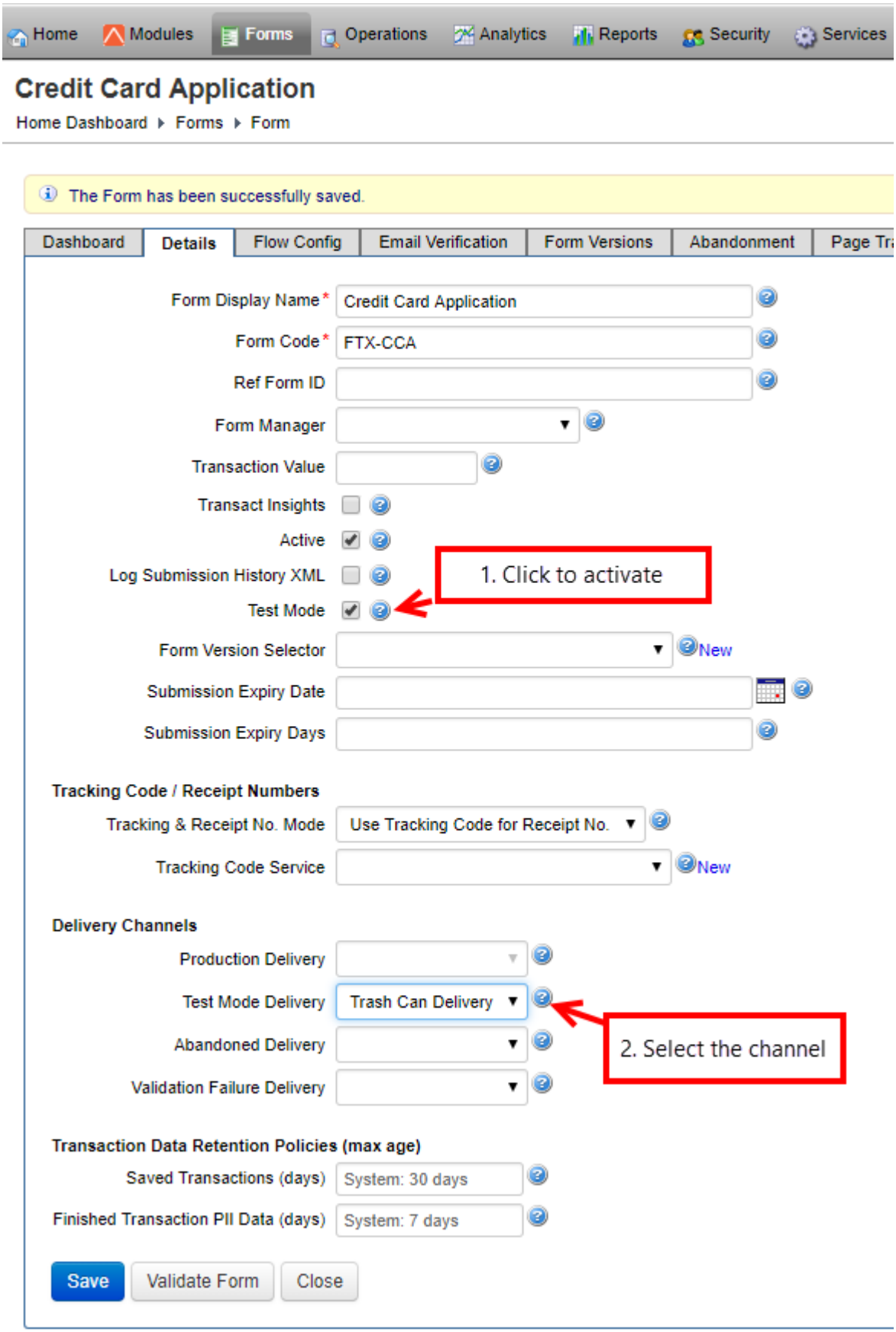
Manager uses the Test Delivery Channel to implement the Form Test Mode functionality, which imposes the following limitations:

- Forms in Test Mode are not available in TransactField App and Web Portal search pages.
- No transaction analytics and reports are created.
- Submissions don't add to an Organization's transaction count.

To configure the Test Mode for a Form:

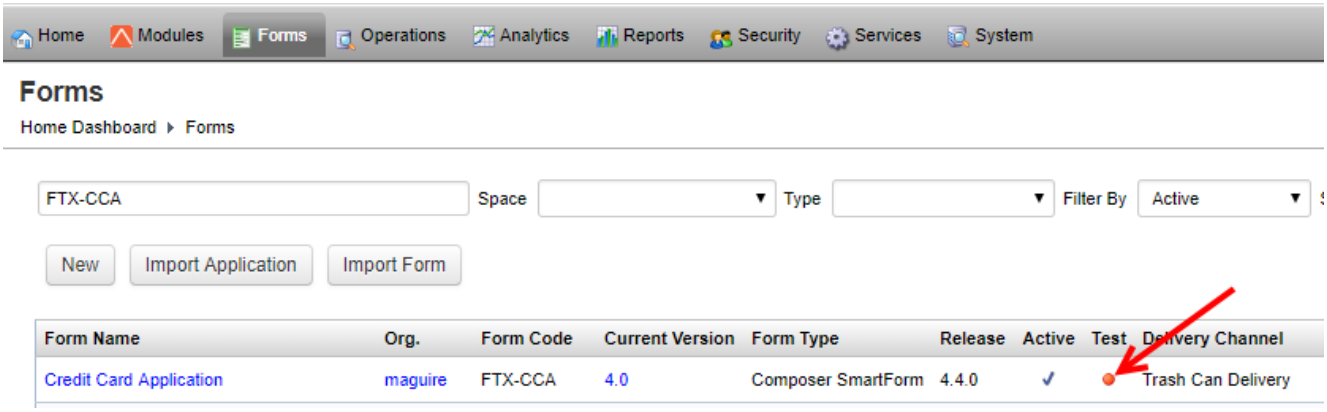
1. Select **Forms > Forms.**
2. Select a form and then click **Edit.**

3. Click the **Details** tab and select the **Test Mode** checkbox.


Home
Modules
Forms
Operations
Analytics
Reports
Security
Services
Credit Card Application
Home Dashboard ▸ Forms ▸ Form
The Form has been successfully saved.
Dashboard
Details
Flow Config
Email Verification
Form Versions
Abandonment
Page Tr
Form Display Name*
Credit Card Application
Form Code*
FTX-CCA
Ref Form ID
Form Manager
Transaction Value
Transact Insights
Active
Log Submission History XML
Test Mode
1. Click to activate
Form Version Selector
New
Submission Expiry Date
Submission Expiry Days
Tracking Code / Receipt Numbers
Tracking & Receipt No. Mode
Use Tracking Code for Receipt No.
Tracking Code Service
New
Delivery Channels
Production Delivery
Test Mode Delivery
Trash Can Delivery
Abandoned Delivery
Validation Failure Delivery
2. Select the channel
Transaction Data Retention Policies (max age)
Saved Transactions (days)
System: 30 days
Finished Transaction PII Data (days)
System: 7 days
Save
Validate Form
Close

4. Click **Save** to update the changes. **DO NOT** close this configuration page yet as there are still steps to complete.
5. Select an organization's delivery channel from the **Test Mode Delivery** dropdown menu, which will be used to deliver submissions when the selected form is in Test Mode. This example uses the Trash Can Delivery channel, so no submissions will be sent out.
6. Click **Save** to update the changes.
7. Open the Form Dashboard to verify that the form is in the Test Mode by checking whether the word **Yes** is displayed next to Test Mode.

   Alternatively, view the form list and check if a form has a red dot is displayed in the Test column, meaning the form is in Test Mode.



Next, learn how to render a form.

Render a Form

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to render a form without opening it in the Maestro editor, so you can check how the form will appear to a form user. The form is rendered using its configuration, such as organization and form properties, attachments and so on, but no data prefill is performed.

If you want to test form's behavior, you need to use the form test functionality, which gives you a form usability experience identical to that of a real form applicant.

To render a form:

1. Select **Forms > Forms**.
2. Locate a form and click **Form Test Render**. The form will be loaded in a new browser tab where you can fill it in normally to validate its functionality.

Next, learn how to view all forms.

Test a Form

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to test a form without opening it in the Maestro editor, so you can check how the form will appear to a form user. The form is rendered using its configuration, such as organization and form properties, attachments and user data prefill.

Testing a form is similar to rendering it, however the form request is treated as if a form user had requested a new form. In which case, you are directed to the landing page, where you may get additional options depending on the user's form space, for example, the login. The landing page takes you to the form, which you can then fill in, submit and make payments or attach supporting documentation as needed.

Usually, you test a form to validate its behavior from the Form Dashboard, which allows you to test the form in both the production and test modes. You can also test the form by finding and opening them in the user's form space. However, this may show you only active forms in production mode, depending on how the form space is configured.

The Dashboard allows you to use both the Landing and Friendly URLs to the forms:

- Landing URL - **`http://www.host.com/tmportal/landing.htm?formCode=lf-exp`**
- Friendly URL - **`http://www.host.com/tmportal/exp/local-form/`** , which uses the client code (**`exp`**) and the form name (**`local-form`**) to form the URL. When creating the friendly URL, Manager converts the client code and form name to lower case and replace spaces with dashes (**`-`**), and then remove duplicate dashes. In addition, ampersands (**`&`**) are converted to (**`and`**), and all other non-alphanumeric characters are stripped out.

To test a form:

1. Open a form in the Form Dashboard.
2. Click the **Landing Page** link or the **Friendly URL** link next to the form space where you want to test the form. The landing page for the form will be loaded in a new browser tab where you can fill it in normally to validate its functionality.

Next, learn how to configure Form Test Mode.

Import Forms

Journey Manager (JM) | Platform Developer | 19.11

Manager allows you to export and import a current form version or all versions of a selected form. This process is commonly used to migrate forms from one environment to another. You can use this functionality to:

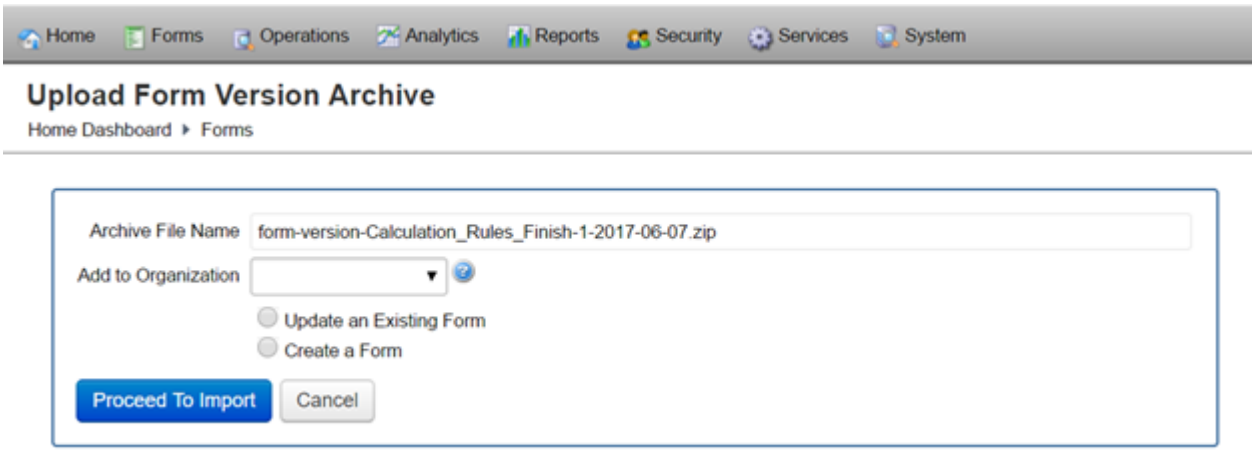
- Rollback one or several forms
- Re-create one or several forms in another Manager
- Promote one or several forms to other environments

> In Manager prior to version 17.10, **Import Form** was named **Import Form Version.**

Manager also allows you to import a form as an application. The main difference between these imports is an archive file format that contains application XML configuration for applications or form XML configuration for forms.

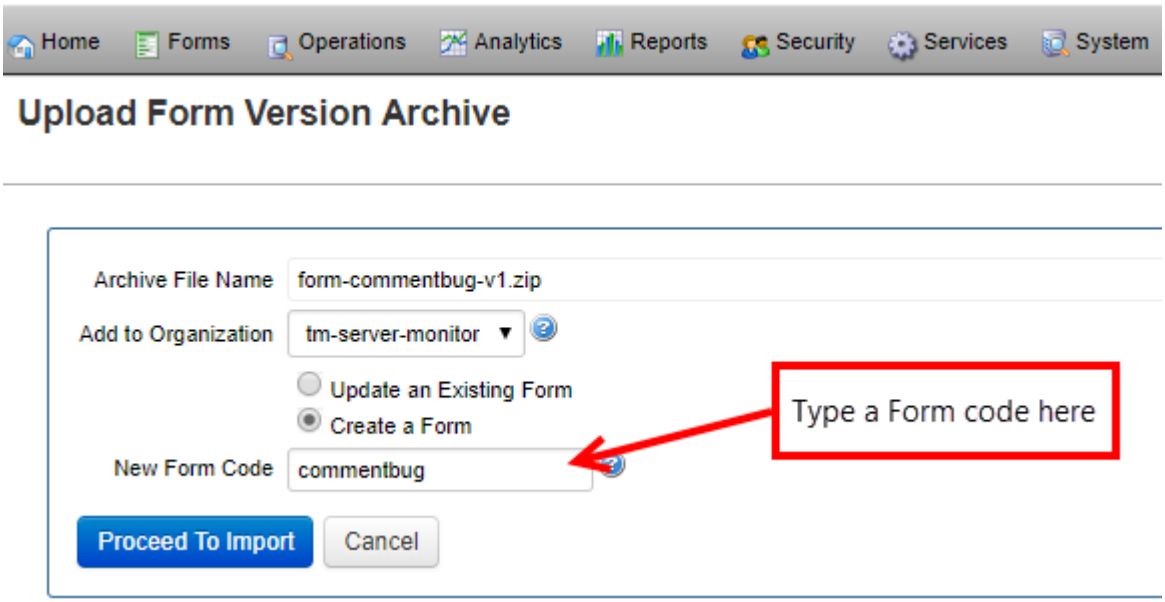
To import one or several forms:

1. Select **Forms > Forms**, and then click **Import Form.**
2. Click **Choose File** and choose a form archive ZIP file.
3. Click **Upload.**

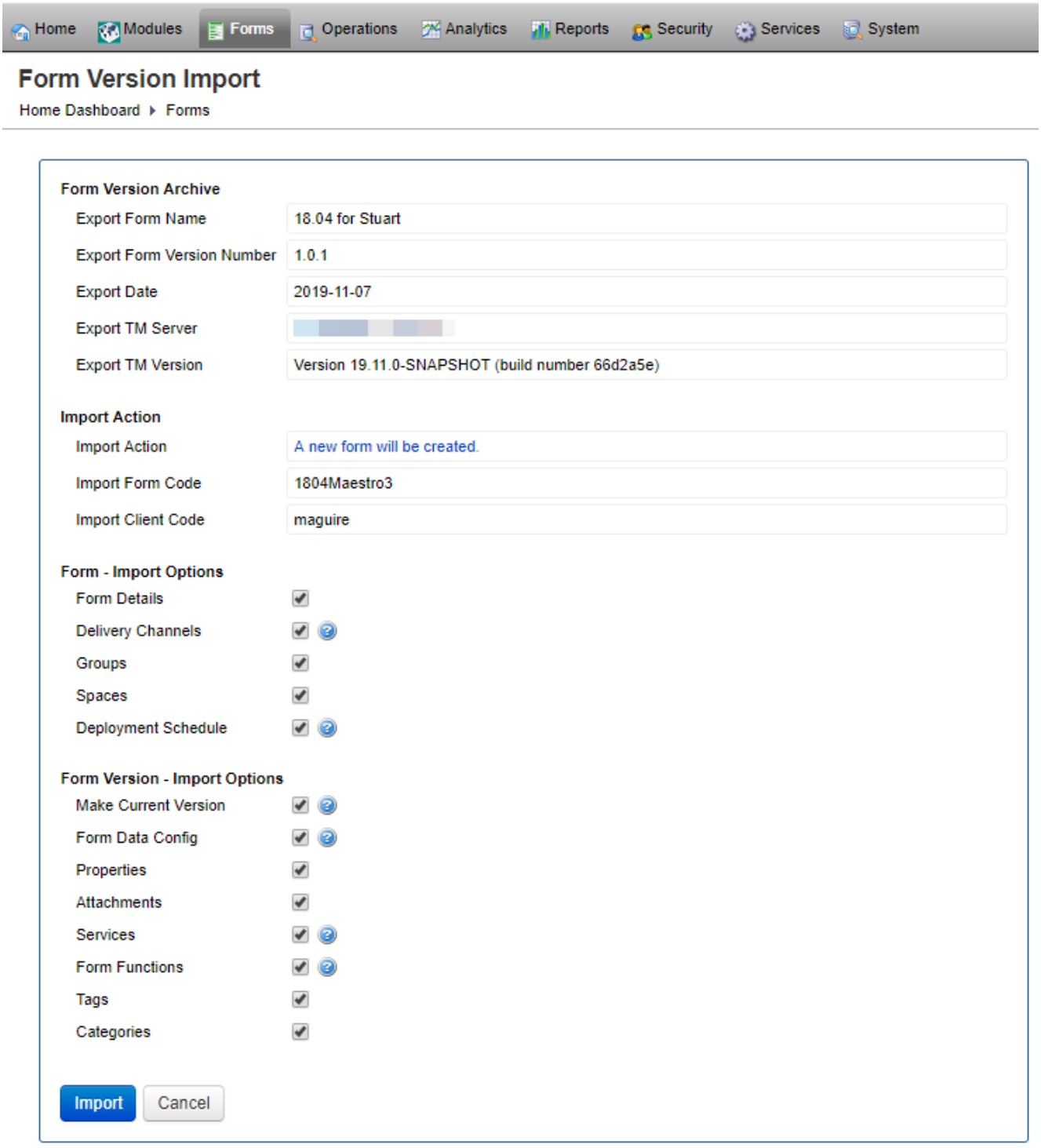


4. Select an organization to add the form to from the **Add to Organization** dropdown, or leave it blank. You will only see the **Add to Organization** dropdown menu if you have access to more than one organization.
5. Select **Update an Existing Form** if the form is already in Manager, so it will be updated with the new version.
6. Select **Create a Form** if the form is not in Manager. Type a form code in the text field as shown below. Manager will automatically fill this text field with a suggested name for the form. If you are unhappy with the suggested name, you can change it by selecting the **New Form Code** text field. The **Form Code Name**

uniquely identifies the form across all the organizations in the current server environment. The **Form Code Name** is an alphanumerical string that cannot include spaces.



7. Click **Proceed To import**. Clicking this button will display the options screen for the form. It is on this screen that you can select or unselect any options that you may want to override. Ensure that you select the option, **Make Current Version**, if you want to make the form that you are uploading the "current version" of the form that is the form your form users will be completing.
8. Review the **Import Options**.



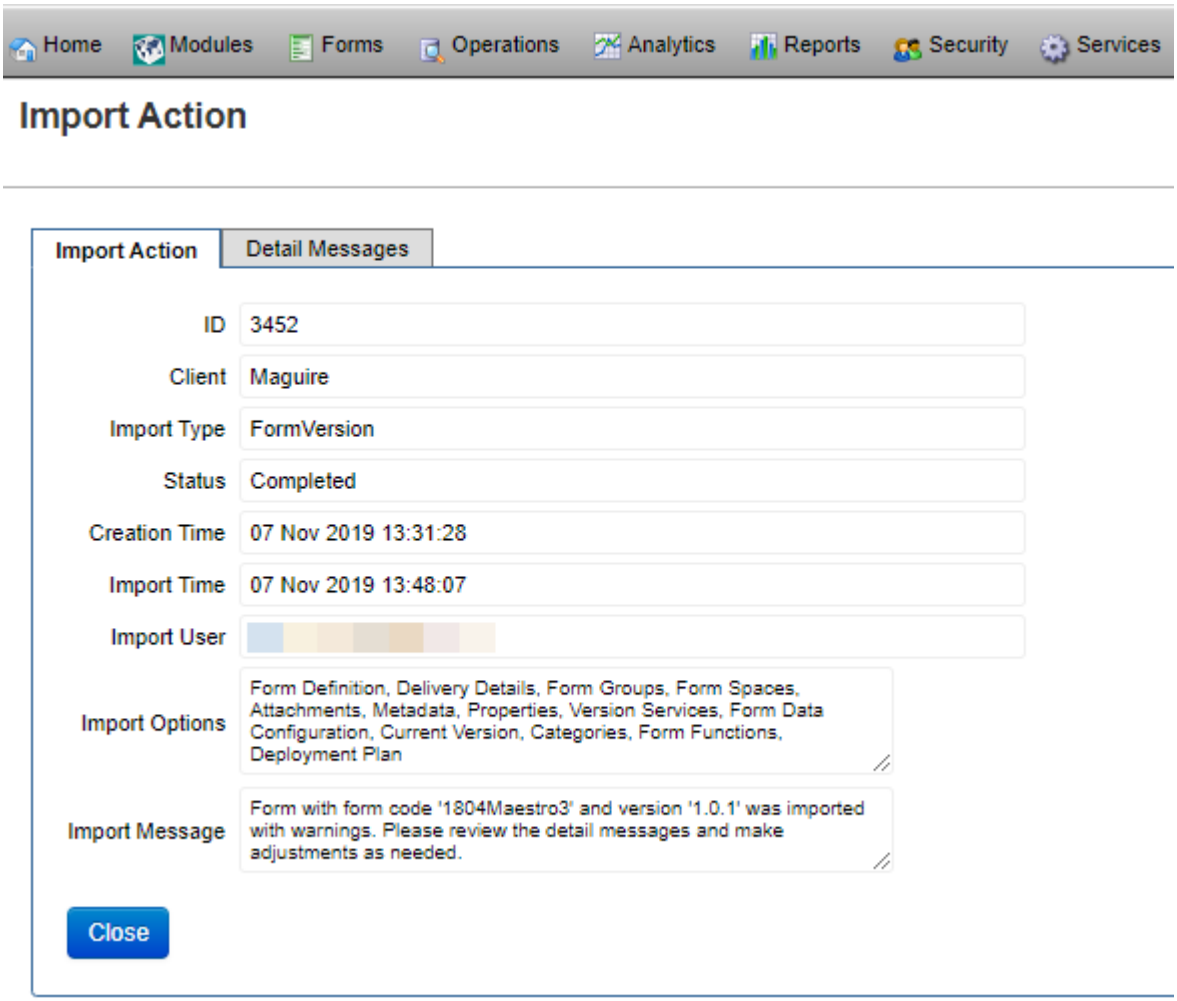
9. Select the appropriate **Form – Import Options**, which are associated with all versions of the form. They are all selected by default.
10. Select the appropriate **Form Version – Import Options**, which are applied to a selected version of the form, normally the current version.
11. Select the **Form Functions** checkbox to import form functions used by the form version. The existing form functions on the target server will be updated with that of from the imported archive. The form functions are

referenced by the name and version, so they must exist on the target server prior to the import, otherwise they can't be imported. | 19.11

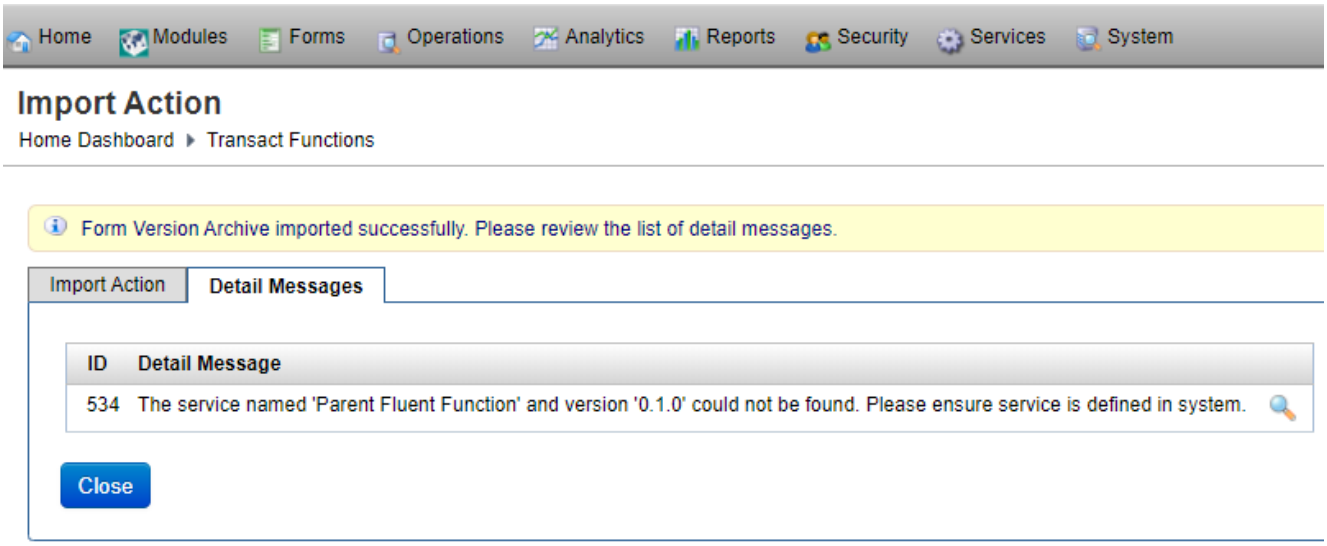
12. Click **Import** to complete the process.

> If the **Create a New Form** option was selected during the import process, and there was a form naming conflict (ie a form with the same name already existing on the server), the form will be renamed, and a message will be displayed in the **Detail Message** tab.

13. Check the import status and message in the **Import Action** tab.



14. Select the **Detailed Messages** to see whether there were any errors or warnings during the import.



If errors are sever, you may want to update Manager configuration to fix the error and import the form again.

15. Click **Close** and check the list of forms to verify the imported forms have been added.

> You can also import and export Maestro forms using the Maestro dashboard.

When importing a form that has come from a more recent version of Manager, you may get an IllegalArgumentException error. To resolve this error:

1. Log onto Manager you have imported the form into.
2. Select **Forms > Forms** and locate the form.
3. Click the **Current Version** of the form and change the Form Type to **Composer Smart Form** . This allow you to render this form correctly now.

Next, learn how to export forms.

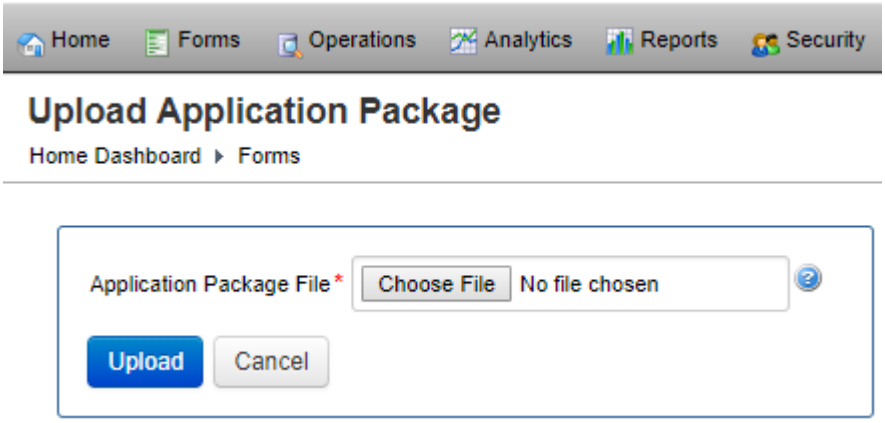
Import a Form Application

Journey Manager (JM) | Platform Developer | 17.10

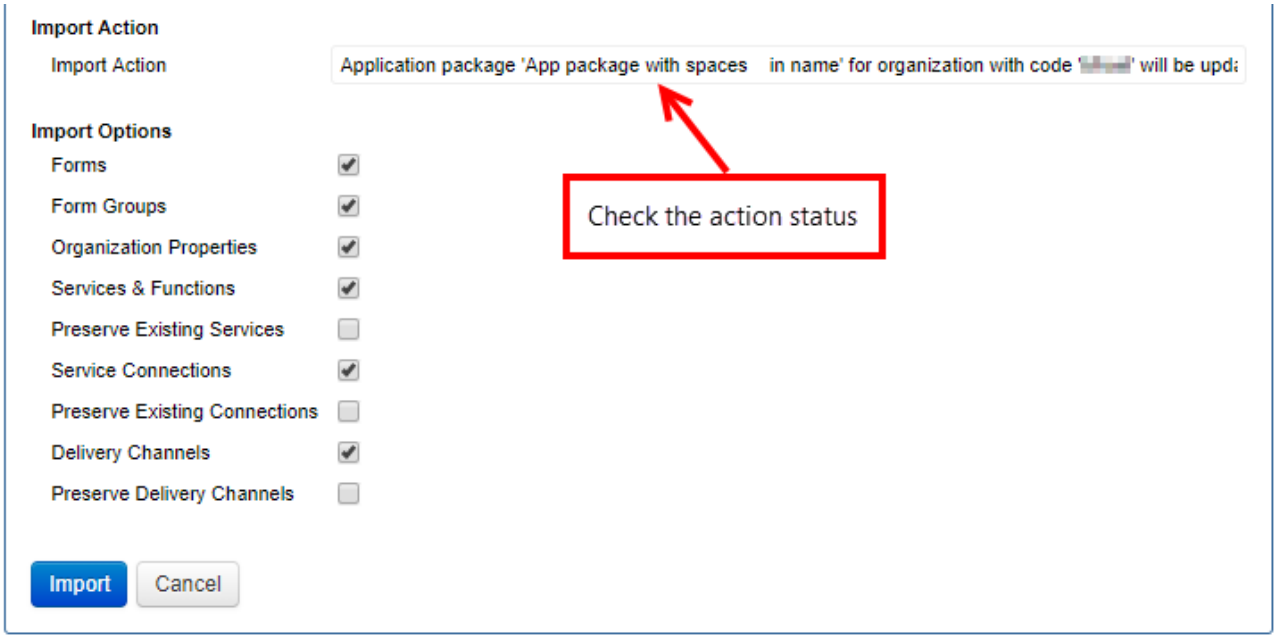
Manager allows you to import a form as an application.

To import one or several forms as applications:

1. Select **Forms > Forms** and click **Import Application**.



2. Click **Choose File** and choose an application archive ZIP file.
3. Click **Upload.**
4. Review and update the **Import Options** as shown below.



5. Click **Import** to complete the process.
6. Click **Close**.
7. Check the list of forms to verify the imported forms have been added.

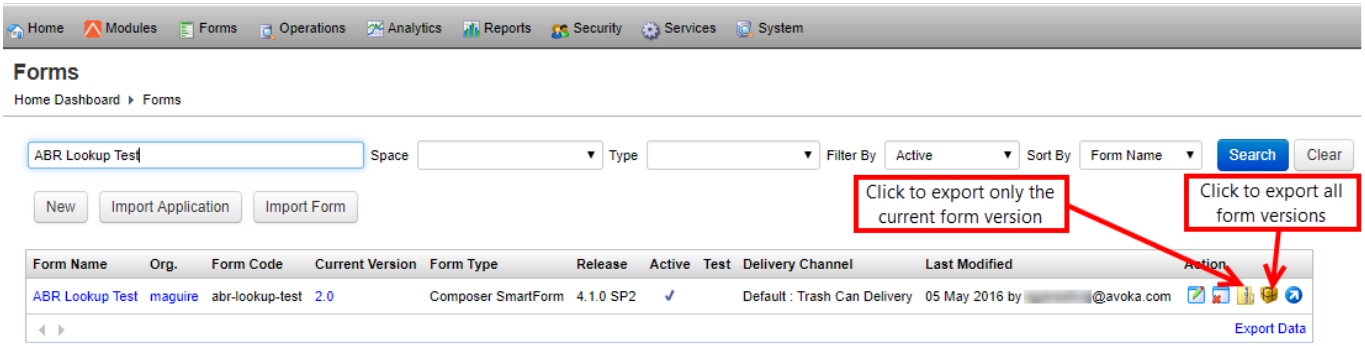
Next, learn how to import a form.

Export Forms

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to export and import a current form version or all versions of a selected form. This process is commonly used to migrate forms from one environment to another. You can use this functionality to:

- Backup or archive forms.
- Re-create or migrate forms into another Manager.



To export a current form version, select **Forms > Forms**, then click the **Export Current Form Version** icon. Manager exports the current form into a form archive ZIP file, which will be downloaded to your machine.

To export all form versions, select **Forms > Forms,** and then click **Export All Form Versions**. Manager exports all form versions and configurations into a form archive ZIP file, which will be downloaded to your machine.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

> You can also import and export Maestro forms using the Maestro dashboard.

When exporting a form from a more recent version of Manager to an older version, you may receive the error message **Could not set property 'formDataEncryptFlag' on entity TemplateVersion**. However, as we're exporting between two different versions of Manager, the original form has this flag set on it but it doesn't exist in the new environment so it can't be set. This should not stop the form from functioning.

Next, learn how to import forms.

Remove a Form

Journey Manager (JM) | Platform Developer | 5.1 & Higher

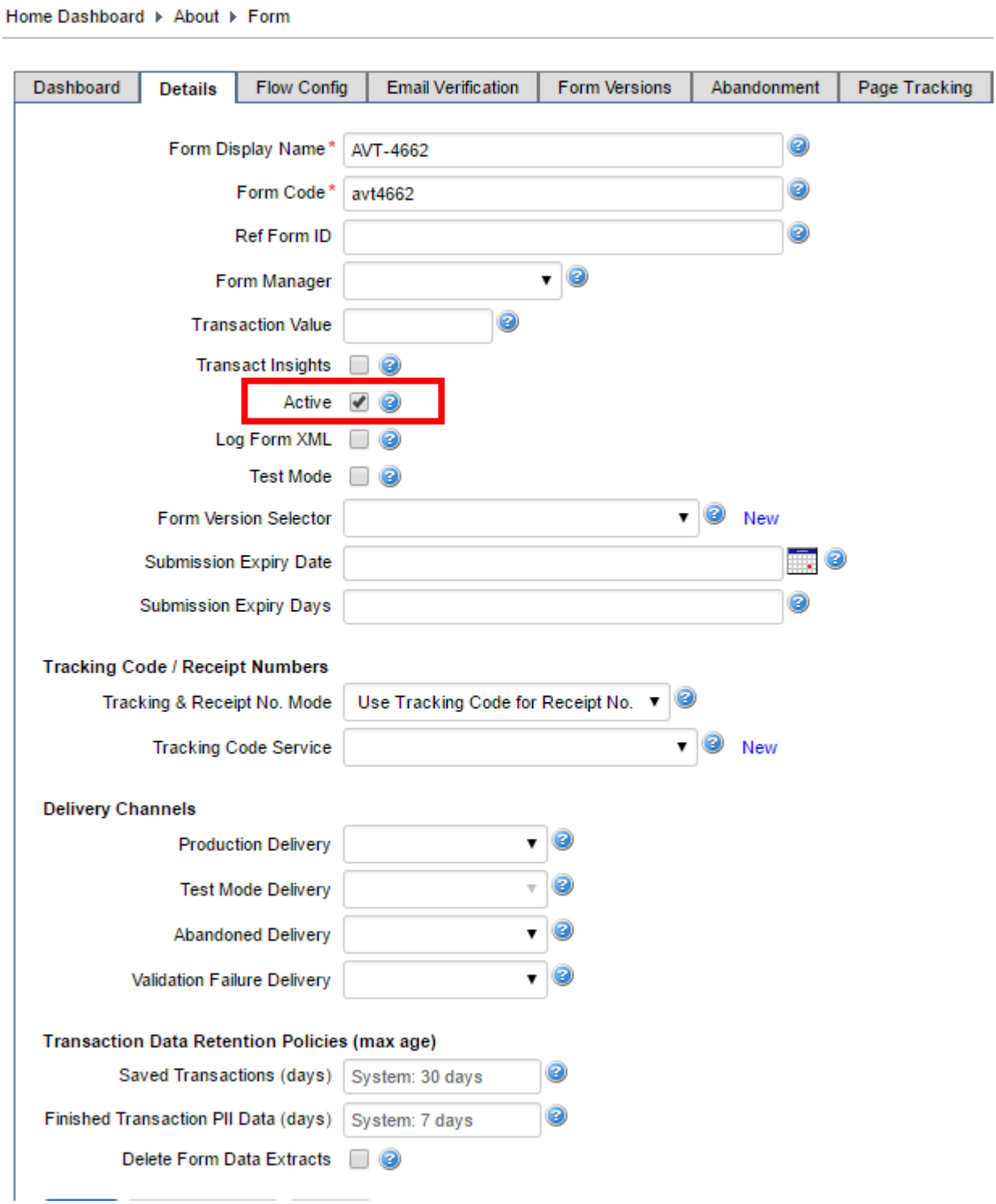
Manager allows you to remove a form, which also deletes all form versions and submissions of that form.

> It is recommended to make a form inactive rather than remove it, as it can impact many form users.

Make a Form Inactive

To make a form inactive:

1. Select **Forms > Forms**.
2. Locate a form.
3. Select the **Details** tab.
4. Clear the **Active** checkbox.



5. Click **Save** to update the changes.

Remove a Form

Removing a form will delete all form versions and submissions of the form.

To delete a form, follow these steps:

1. Select **Forms > Forms**.
2. Locate a form.
3. Click **Remove** and click **OK** confirm your decision.

Next, learn how to view all forms.

Migrate Forms

Journey Manager (JM) | Platform Developer | Form Builder | 5.1 & Higher

Manager provides a mechanism to migrate forms from one environment to another. It allows you to develop and test forms in different environments before migrating them over to the production environment.

To migrate one or several forms:

1. Export forms from one environment.
2. Import forms to another Manager environment.

> You can use the same process to perform a form rollback.

Next, learn how to view forms.

Rollback Forms

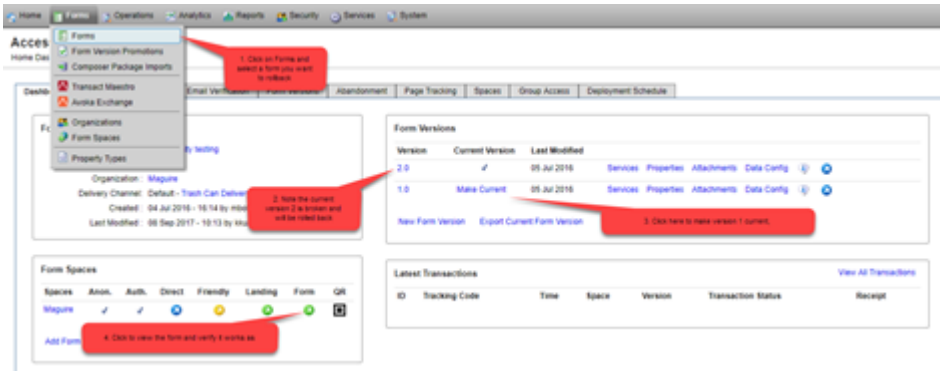
Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager provides a mechanism to rollback forms so you can restore your environment to correct form deployment failure or an error. Forms, installed as application packages, can be easily rolled back to previous versions in. However, you must:

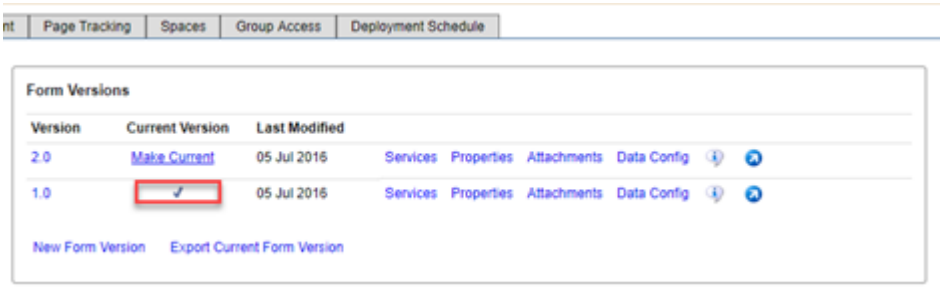
- Understand all components involved into the workflow
- Have backups of the current and previous versions
- Maintain your change logs records, so you know the latest working versions of applications

To rollback a form:

1. Locate a form you want to rollback.
2. Click the **Make Current** link of the previous form you want to rollback to, for example the version 1.



3. Confirm that the current version is what you have rolled back to, for example 1.



4. View the form in the browser and verify it works as expected. Your rollback is complete.

If you have only one application deployed, you can not roll it back as shown above. However, and you can restore it from a backup using form import, or contact our Customer Care team for help.

Be careful of having versions of forms or services which are incompatible with each other, as it will result in a

error and inability to rollback to your previous forms.

Next, learn how to view forms.

Save Challenge

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager comes with a Save Challenge feature that allows users to resume their form applications, so they can complete or cancel them. This is an important step in the onboarding process when a user needs to save a form application to get some required information or a web browser is closed or crashed. Therefore, the Save Challenge provides a secure mechanism to return to the form.

To enable a Save Challenge:

1. Select the **Supports Save Challenge** checkbox in a form version.
2. Specify the **Save Challenge XPath** field in Form Configuration Mapping, for example, the Save Challenge XPath is set as **`//DateOfBirth`**.
3. Check the **`supportsSaveChallenge`** is set to **`true`** in the SDK **`form-def.json`** file.

   > The **`application-package-config.xml`** is updated with this option after running the **`app-package`** command.

4. Check the Save Challenge is enabled in Transaction Status.

Next, learn abut other Manager's features.

Maestro Form Static Content Externalization

Journey Manager (JM) | Form Builder ▪ System Manager / DevOps | 18.11

Manager includes a new Journey Maestro form rendering mode, **Static Content Externalization**[1] (SCE), which significantly reduces the time it takes to load large forms. This is important because long application loading times can significantly increase the bounce rate with users closing the browser window before starting the application.

Using this new Maestro SCE rendering mode option, Manager will externalize 85-95% of the Maestro HTML form content, which can then be served in parallel by **Content Distribution Networks**[2] (CDNs). This reduces the amount of time it takes for a form to initialize as the initial HTML page is downloaded quickly and the larger static images, **JSON**[3] and **JavaScript**[4] content can be downloaded in parallel by separate browser threads. By serving this static content out of CDN, the content is more likely to be close to the users device, reducing the network latency. The content can also be cached by the users browser.

It's possible to achieve a 2x reduction in form load times for large Maestro applications.

There are also benefits of using Maestro SCE rendering mode option for the Manager server as we have observed a 20% throughput improvement under performance load tests

The SCE mode works with older Maestro release forms and does not require any changes to the Maestro forms to use it. This feature doesn't change the structure of the form but simply externalizes static resources, such as images, JSON and JavaScript content. The form is still a single page form with static resources have been separated out.

Next, learn how to enable the Maestro SCE Mode opt in feature.

---

[1] Static Content Externalization (SCE).

[2] A content delivery network or Content Distribution Network (CDN) is a system of distributed servers (network) that deliver pages and other Web content to a user, based on the geographic locations of the user, the origin of the webpage and the content delivery server.

[3] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

[4] Javascript (JS) is a scripting languages, primarily used on the Web. It is used to enhance HTML pages and is commonly found embedded in HTML code. JavaScript is an interpreted language. Thus, it doesn't need to be compiled.

Server Monitor Test Form

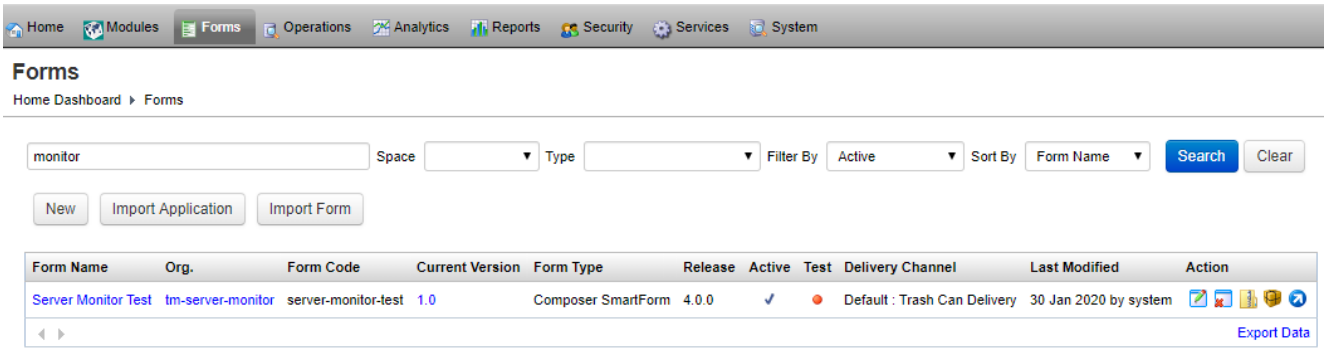
Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager comes with a **Server Monitor Test** form, which you can use to check Manager server heartbeat to make sure it's up and running. This is often done for the cloud based installations.

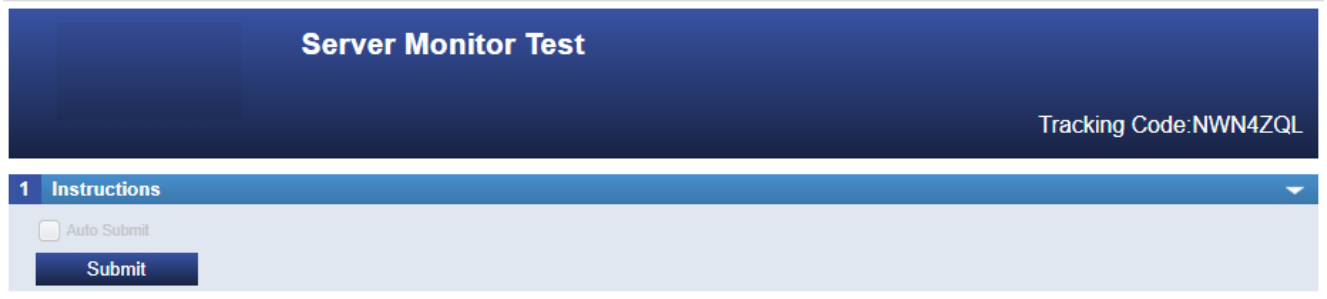
> The Server Monitor Test form is only available if the **Deploy Monitoring Form checkbox** is selected during Journey Manager installation.

To run the Server Monitor Test:

1. Select **Forms > Forms**.
2. Search for the **Server Monitor Test** form.



3. Click **Form Test Render**.
4. Click **Submit**.



5. Check the submission was successful.
6. Type your email address and click **Send Now** to send the PDF copy to validate the delivery.

Next, learn how to view all forms.

Integrate Forms to Existing Websites

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to integrate its forms to existing websites. The modern trend is for launching forms from existing websites is to open a new browser tab or window. This has a few important advantages:

- Works better on mobile devices where screen real-estate is limited.
- Prevents the user from moving away from the data entry experience by clicking on links in the surrounding page.
- Is simpler from an implementation perspective

You can find several integration strategies listed below:

iFrame Embedding

Form Display Mode is Embed with iFrame.

Pros

- Simple
- No "leakage" of JavaScript or CSS from parent page to form page

Cons

- Pages although appear to be integrated are actually separated causing a number of issues.
- Refreshing of the page is problematic.
- Back button does not work properly.
- Difficulties in maintaining styling with significant effort needed on BT to maintain.
- Difficult to pass control between the pages.
- Complexities in the JavaScript as the parent control needs to be passed to Manager.
- Potential responsive design issues, especially on smaller devices.

<DIV> Embedding

Form Display Mode is Embed with a DIV.

Pros

- Solves some of the problems associated with iFrames.

Cons

- Complexity around the sharing of JavaScript used to render the contents of the form into the DIV element.
- Potential for code bleed/CSS clashes.
- We don't recommend this integration.

Launch a new tab or window

Form Display Mode is Direct Render.

Pros

- Clean code separation.
- Clear boundary of ownership.
- Control easily and intuitively passed between NextGen UI and the Manager on-boarding pages.
- Control to the Avoka pages is managed at the URL level.
- Recommended by Avoka as their preferred approach with their other customers.

Cons

- The form must be built with a simple header that includes a small amount of branding from the web site.
- Visual design - need to be simplified in terms of shared header.

Configure the Form Display Mode

The Form Display Mode is the process of embedding the form, either with a DIV or iFrame, to display the form inside a portal page, where that page typically provides the header, footer and sidebars. Direct Render displays the form using a Servlet and does not include any elements outside the form itself. For more information on the **Form Display Mode** setting, see form flow configuration.

Form URLs

These different methods of displaying the form mean that different URLs are used by Transact, as follows.

URL	Usage
<portal>/form.htm?formCode=<form code>	Form Page which will embed a new HTML or PDF form in the page using an IFRAME or DIV tag. The form is identified by the formCode parameter.
<portal>/secure/form.htm?formCode=<form code>	Same as above but under a secure path, which requires the user to be authenticated
<portal>/form.htm?submitKey=<submit key >	Form Page which will embed a saved HTML or PDF form in the page using an IFRAME or DIV tag. The form submission is identified by the submitKey parameter.
<portal>/form.htm?taskKey=<task key >	Form Page which will embed an assigned HTML or PDF form task in the page using an IFRAME or DIV tag. The form task is identified by the taskKey parameter.
<portal>/servlet/SmartForm.html?...	Same as above options but doing a direct render , where by the form is not being embedded in another HTML page

When a HTML form is being rendered on a mobile/tablet device (iPad, phone, etc.) it will automatically redirect to the /servlet/SmartForm.html path, even if the form is configured to be embedded in the form page. This is because embedding HTML forms does not always work well on mobile/tablet devices.

Next, learn how to view all forms.

Advanced Debugging of Maestro Forms

Maestro | Form Builder | 5.1 & Higher

Occasionally while developing Maestro forms you may see a behavior that doesn't seem right and you can't figure out why. This usually occurs in relation to rules that you've created using the script editor. You can learn how to do the basic debugging of Maestro forms, but sometimes you need to apply more advanced debugging techniques that you can use to find out what is going on in your script and in the form data, especially debugging of CSS and styling issues is not covered.

Recommended Skill Set

Debugging Maestro forms in the browser is a technical task and requires some understanding of web application development in general, specifically the web standards of HTML, CSS and JavaScript.

Additionally, forms produced by Maestro run on AngularJS, which is a JavaScript based open-source front-end web application framework backed by Google. There is a very strong community supporting this technology and while it is not required that you are familiar with AngularJS, it would be a benefit for you to have some basic knowledge of the framework. We would recommend you review the tutorials to develop your understanding of this technology:

Debugging Tools

Before getting started, ensure you have appropriate tools for debugging JavaScript in the browser. These include:

Modern Browser

Most modern browsers come with good developer tools these days. We prefer to use Google Chrome, but Firefox and Microsoft Edge can also be used. This article assumes you are using Google Chrome which we recommend.

Developer Tools Enabled

All these browsers come with developer tools out of the box and allow you to install plugins/extensions to provide additional capabilities. There are a number of ways to enable these developer tools but universally, hitting the F12 key while viewing a page will bring up the built-in developer tools.

AngularJS Browser Plugin

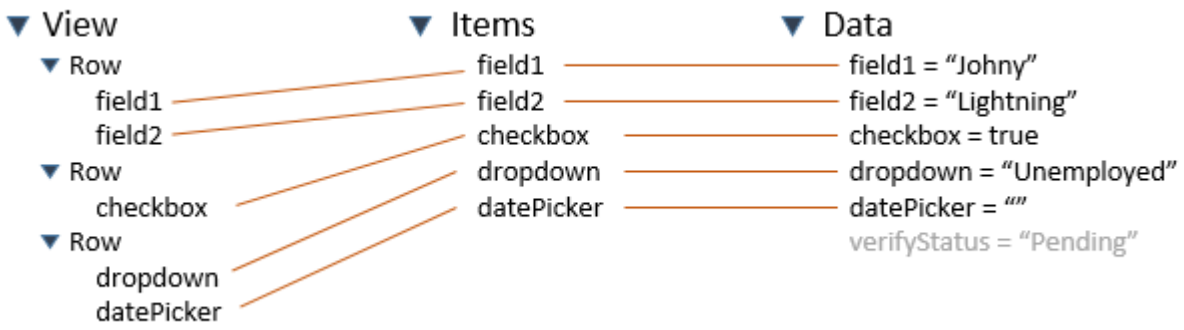
Maestro forms are built on the AngularJS framework and plugins are available to provide specific support for AngularJS web applications. In Chrome, we use AngularJS Batarang, Firefox also has AngularJS plugins that may provide the same capability.

Maestro Form Architecture

The architecture of a form consists of the following main elements:

- Form View - A hierarchical structure that defines the visual layout of the form.
- Form Items - Each component (e.g. Field, Button) or container (e.g. Block, Section) within the form structure is considered an item. Form items is a list containing every item in the form.
- Form Data - Think of form data simply as a map of key / value pairs. Most data elements are linked to a form item (e.g. a field) but data elements can also exist autonomously.

These 3 elements are all closely related. Form items appear in the Form View and have Data elements associated with them:



Form View

The form view is a hierarchical structure that consists primarily of horizontal rows, each containing one or more components. The hierarchical structure represents the layout of the form and all components within it.



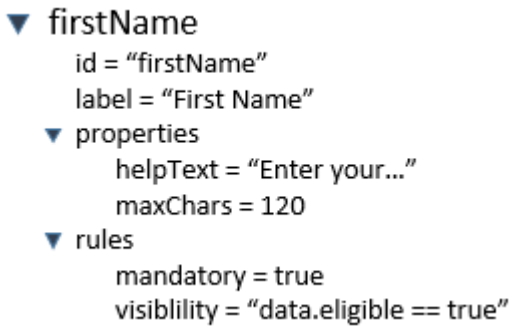
Being hierarchical, rows in the view can be containers for lists of more rows in a recursive manner.

Form Items

Each form item in Maestro is uniquely identified by a field Id. Maestro generates these field Ids based on the name of the field but also allows you to override the generated value and manually specify a field Id so long as they are unique in the context of the form.

Having guaranteed unique field Ids for all items within a form allows us to have a flat list (array) representation containing all items in the form, making access to these items much simpler than having to navigate a view hierarchy to find them.

Items can contain UI elements, but also have properties and rules associated with them. By way of demonstration, consider a mandatory first name field that has a maximum length of 120 characters and a visibility rule:



Form Data

All data in the form is stored in a single object called the Form Data object. Like form items, each form data element must have an id that is unique in the context of the form. If a data element exists with the same Id as a form item, that form item will read and write its data to that data element. Data elements can exist without being bound to a form item.

Depicted below is 4 data elements, 3 of which are bound to fields (items) and one that is un-bound (verifyStatus). The middleName data element is blank suggesting that nothing has been entered into the middleName field.

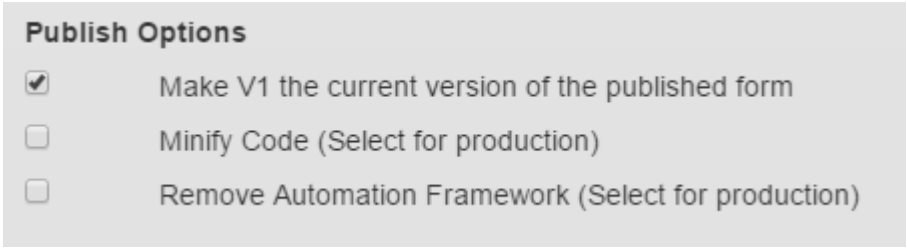


Data elements that are not bound to form items (verifyStatus in this example) will not be Saved/Submitted with the form data and so only exist for the duration of the session.

Note that the Data Field component type can be used to bind data elements without having to display them in the visual layout so that they get persisted with the form data.

Publishing Forms for Debug

When you publish a Maestro form there are a number of options available to you.



Some of these are very relevant to debugging, these are:

Publish Options	Description
Minify Code	Deselect this option. Code minification is the process of compressing code to reduce the size and therefore load time, but does not affect the operation of the code. The process removes unnecessary characters from the code,white space, new line, comments etc... The result of minification is a single line of code that is very long and difficult to read and debug so ensure that this option is deselected for debugging.
Remove Automation Framework	Deselect this option. The automation framework built into Maestro forms facilitates UI driven automated tests such as those that would be run from test tools like Selenium. This framework also contains some features that will assist in debugging JavaScript so it is handy to keep the automation framework for debugging.

Activating a Debug Session

Tostart a debugging session you should:

- Load the published Maestro form in a modern browser of your choice
- You must access the published version of the form, DO NOT try and debug a form in Preview mode in the Maestro designer.
- If you are intending to debug JavaScript (e.g. Maestro rules) then ensure you have deselected the Minify Code and Remove Automation Framework publish options as described above.
- Once the published form is loaded in the browser you can activate the browser developer tools by hitting the F12 key.

Shortcuts for Republishing Forms

While debugging forms you may need to republish your form with changes multiple times and you need to refresh the page with your published form on it each time. The following keyboard shortcuts are available for you to use to streamline this process:

- CTRL-P: When used from the Maestro Design page will bring up the Publish dialog. (Command-P on Mac)
- CTRL-ENTER: Will trigger the publish function. Remember to deselect the Minify Code publish option first.
- CTRL-TAB: This will switch you to the last active browser tab which in most cases will be the published form.
- CTRL-R: Will refresh the currently active browser window.

Refreshing the page with the republished form on it should leave the browser developer tools active so you should not need to reactivate them each time.

Inspecting the Form Object

In the Maestro framework, the Form object contains everything else you might need to access for debugging including the View, Items and Data (see Maestro Form Architecture), so to access any of these elements you need to scope the Form object. There are a number of ways to do this from the Console tab in the browser developer tools:

- Using JQuery you can scope the form object in this manner:

```
$("Form").scope().Form
```

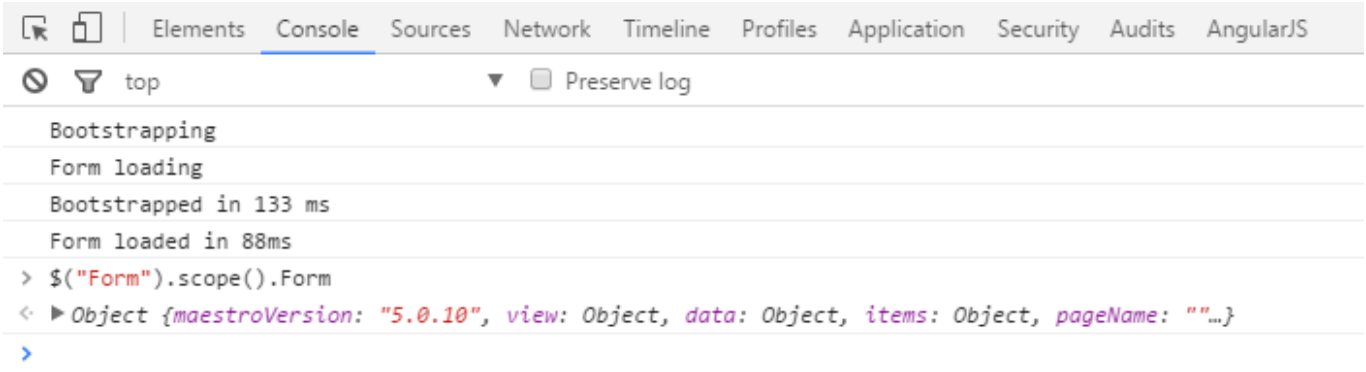
- If you have included the Automation Framework (see Publishing Options) you can short cut this access by using the maestro root element included with that framework:

```
maestro.Form
```

- If you have AngularJS Batarang (or similar) installed, they may also allow you to access the Form object by way of the $scope object (You need to select an element in the Elements tab first - e.g. body or form):

```
$scope.Form
```

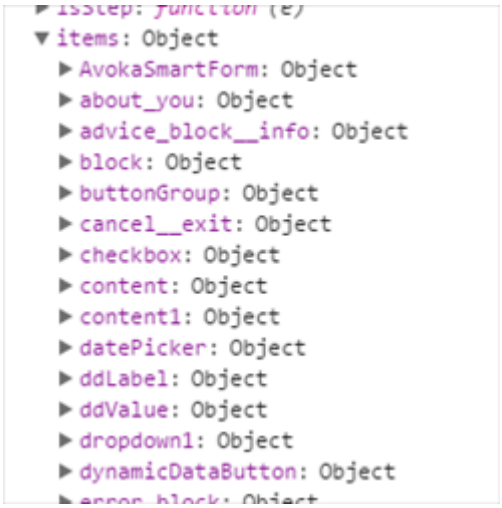
Enter one of these directives into the Command prompt in the Console tab and press Enter to scope the Form object:



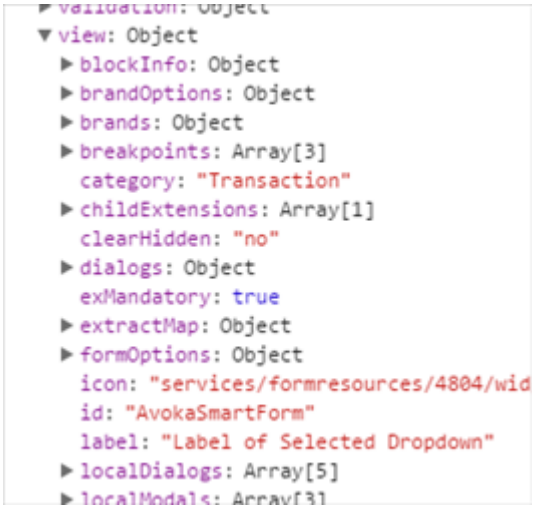
Opening up this object you will see a long list of sub elements but among those you will find the 3 key elements to the Maestro Form Architecture, the Form Data object,



The Form Items,



And the Form View.



Of course, if you have a JavaScript debugging session active (as described below) you will have access to the Form Data object in the function scope and can readily inspect its contents.

Logging an Item to the Console

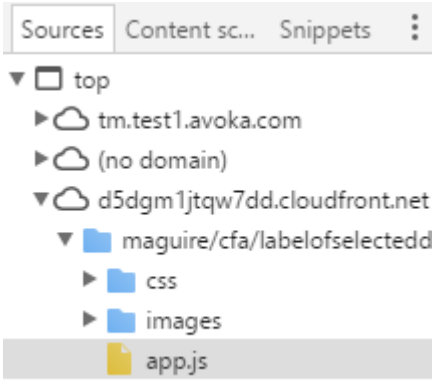
There is a handy Util function to generate a printable representation of an item and log it to the console as follows:

```
$scope.Util.logItem($scope.Form.items.personalDetailsBlock)

"personalDetailsBlock
┌ firstName
└ middleNames
- lastName
- dateOfBirth
 ┌ day
 | month
 └ year
 — date
 — daysData
- emailAddress
- mobileNumber
"
```

Locating a JavaScript Rule for Debug

All rules in Maestro are implemented as JavaScript so to locate the logic implemented in a Maestro rule you must find the JavaScript code that implements that rule. You can do this as follows:

1. In the developer tools window select the tab that lists the page source files (in Chrome this is titled Sources).
2. Open the app.js file by either using CTRL-O (CMD-O on Mac) and typing in the name of the file, or by searching in for it the file sources hierarchy then double-click to open it (the location of this file will vary depending on your installation and the existence of a CDN in your environment, but it should only be a couple levels deep).



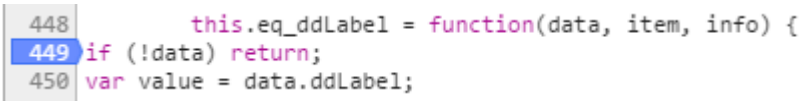
3. In the app.js file, search for and locate the JavaScript function that contains your rule in the app.js file. There are several ways to do this depending on the capabilities of the browser dev tools, but most commonly you will search by a code snippet or by the function name:
4. To find the function using a code snippet, select a snippet of code from your rule script by clicking in the app.js file and using CTRL-F to activate the find feature. You may need to review your rule script in the Maestro designer and select a snippet that is likely to be unique. For example, below I have searched for the snippet Form.getItemFromPath('data.buttonGroup') and located the function eq_ddLabel that contains my rule script:

```
447
448         this.eq_ddLabel = function(data, item, info) {
449 if (!data) return;
450 var value = data.ddLabel;
451
452 var item = Form.getItemFromPath('data.buttonGroup');
453 var selectionObj = Util.find(item.properties.options, "value", data.buttonGroup);
454
455 return (selectionObj && selectionObj.label) || "";
456 };
457
```

5. To search by JavaScript function name you will need to know the naming conventions for these functions (see the section on JavaScript Function Naming below). If you know the function name you can either search for the function name as a snippet (as above) or if your dev tools support it you can use the JavaScript Member search capabilities - in Chrome this can be activated using CTRL-SHIFT-O:



6. To debug your rule, set a break-point on one of the lines before the code you want to step through by clicking in the left column area and confirming the break-point indicator:



7. Now you are ready to debug the rule. You can return to your form window and perform the action that will trigger the rule to execute and step through your rule script using the standard JavaScript debug functions.

A word of caution against modifying JavaScript in-line in the browser debug tools. Results of this sort of change are often unreliable. While it is less convenient to republish a form each time you want to change a rule - this is the recommended approach.

Rule Types and JavaScript Function Naming

Every rule that you put into your Maestro form will have a corresponding JavaScript function in the app.js file. The name of this function will be generated based on a standard naming convention that consists of a rule type code and the field ID with an underscore separator as follows:

```
<rule-type-code>_<field-id>
```

The following codes represent the types of rules used by the Maestro foundation widgets:

Rule Type Code	Description
sh	Short forshow, this is the code for avisibilityrule that controls whether the field is presented on screen. The result of this rule should be a boolean value where true = visible, false = read-only.
us	Short forusable, this is the code for aneditabilityrule that controls the read-only status of a field. The result of this rule should be a boolean value where true = editable, false = read-only.
md	Short for mandatory, this is the code for amandatory ifrule that controls the required state of an item. The result of this rule should be a boolean value where true = mandatory, false = optional.
ok	Interpreted asOK, this is the code for avalidationrule that controls the error state of the field. The result of this rule should be a string that is either blank (for valid values) or con-

	tains the error message (for invalid values).
chok	Interpreted as Change OK, this is the code for avalidate after changerule that augments the standard validation rule to provide support for dynamic data validations. The result of this rule can either contain the error message (like the standard validation rule) or a promise.
eq	Short for equals, this is the code for a calculation rule. The result of this rule should be the calculated value of the field. Warning: In Maestro, calculation rules are not triggered if the field they belong to is not visible.
click	Aclick ruleis an action rule triggered by clicking on a clickable UI item.
blur	Ablur ruleis an action rule triggered when the focus leaves a field.
focus	Afocus ruleis an action rule triggered when a field acquires the focus in the browser window.
change	Achange ruleis an action rule triggered when the value of a field changes.
dc	An acronym fordynamicclass, this rule can be used to apply a css class to a field dynamically based on logic.
load	Anaction rulethat is triggered at the time the form is loaded.
presubmit	Anaction rulethat is triggered prior to form submit.
postsubmit	Anaction rulethat is triggered after the form submit event.
onSuccess	This rule type is specific to the Dynamic Data Button and is an action rule that is executed upon successful completion of the dynamic data function.
onFailure	This rule type is specific to the Dynamic Data Button and is an action rule that is executed upon failure from a dynamic data call.

Note: widget developers can create additional rule types for their widgets.

Examples

Function Name	Rule
sh_previousAddress	A visibility rule on the previousAddress block
eq_totalIncome	A calculation rule on the totalIncome currency field
ok_emailAddress	A validation rule on the emailAddress field
click_verifyIdentity	A click rule on the verifyIdentity button
load_AvokaSmartForm	A load rule on the form

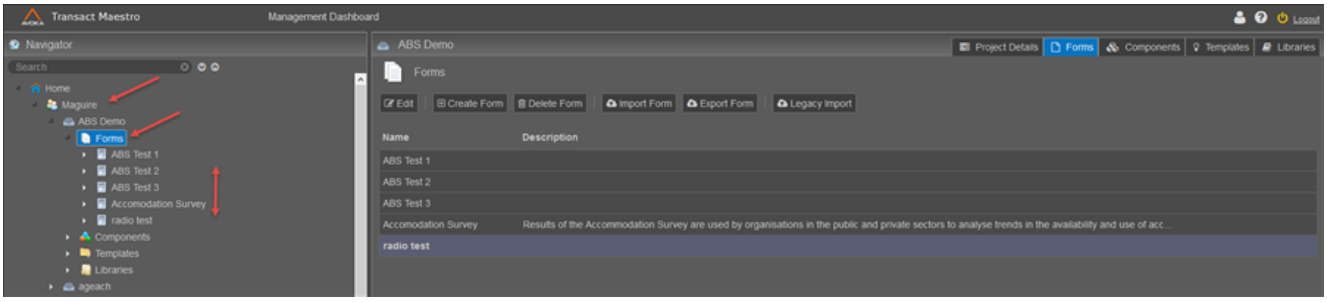
Next, learn about ?

Change the Maestro Form Description

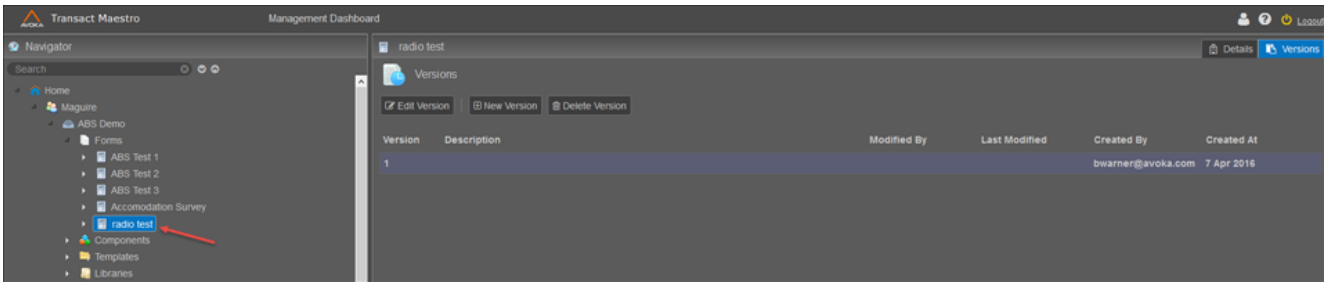
Maestro | Form Builder | 5.1 & Higher

To change a Maestro form description:

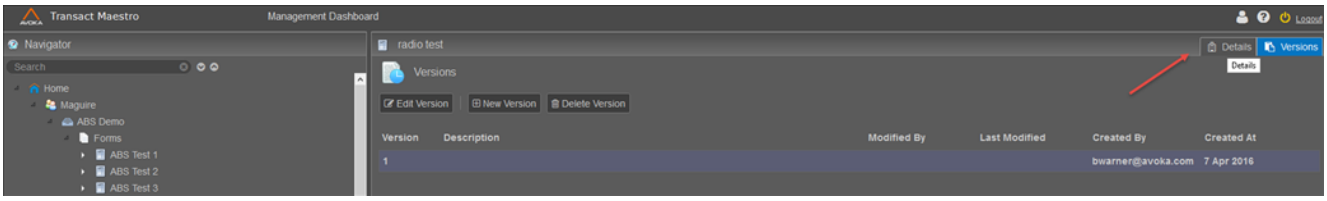
1. Open Maestro to see the list of Organizations.



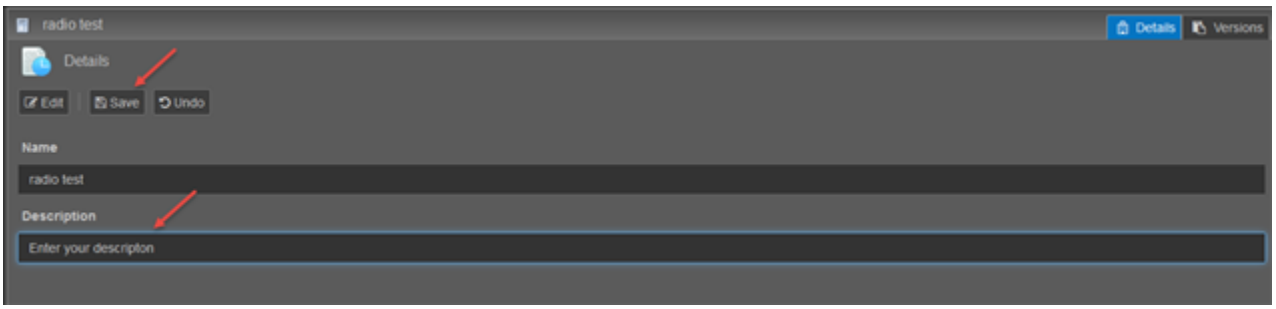
2. Select your Organization followed by Forms:
3. Using the right hand list, click on the Form you wish to update the Description for:



4. Click the Details tab:



5. Enter or update the Description for the Form. The Save button will enable once a change is detected so when your Description is entered, you can save:



Next, learn how to open a Maestro form.

Change a Maestro Form Version Description

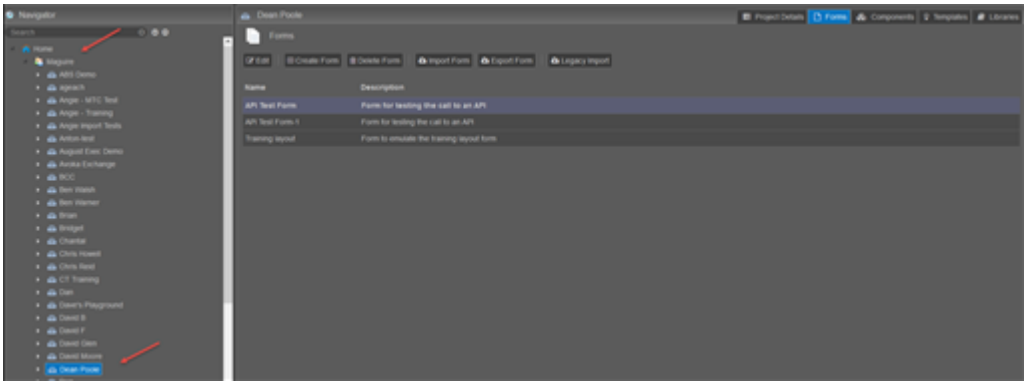
Maestro | Form Builder | 5.1 & Higher

When creating a new version of a Maestro form, you may need to change the description you applied to the version or add at a later date. There are two methods to change the version description.

Method 1

To change a Maestro form version's description:

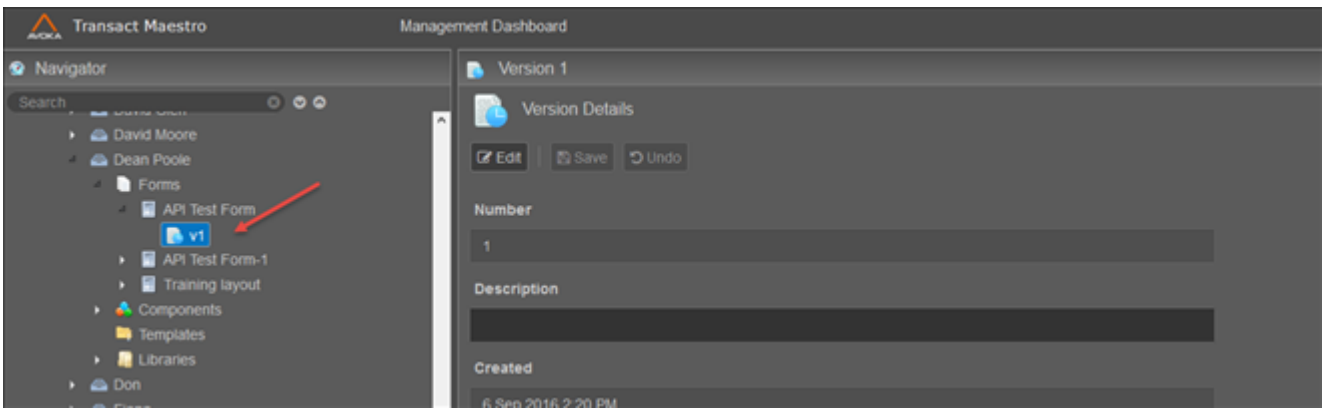
1. Open Maestro to see the list of Organizations.



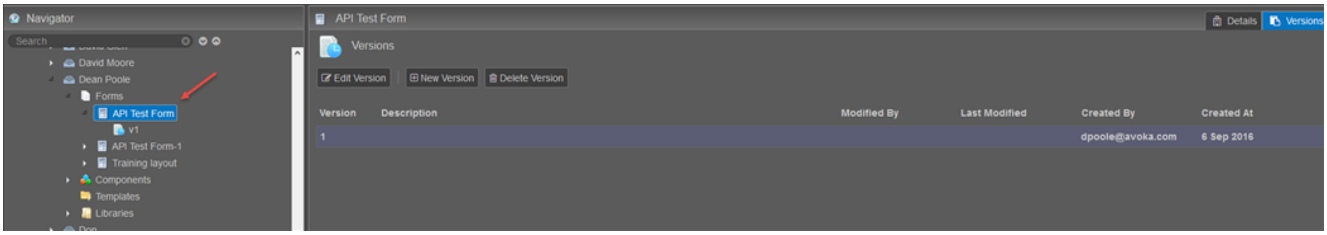
2. Select your Organization followed by Forms:
3. Locate your Form and expand it so you can see any versions that apply. You will see either your existing Description of that Version or none at all:



4. Click the version directly. In the Navigator pane, click on the version to be updated that is listed underneath the Form name:



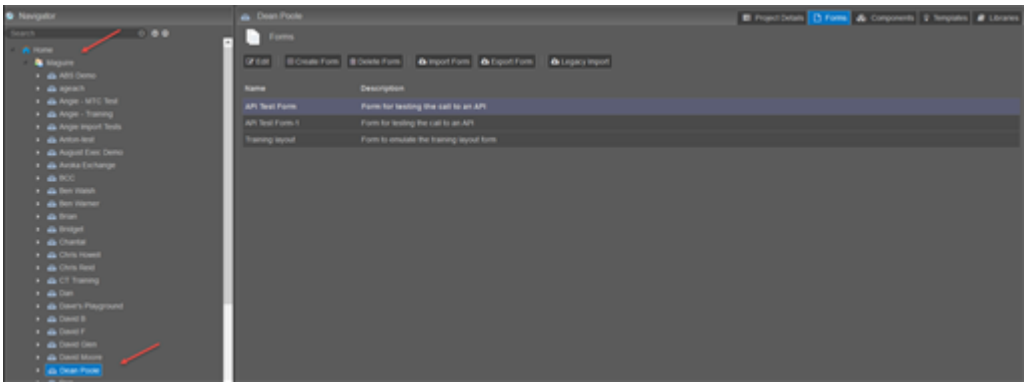
5. This will bring up the following screen and you can now update the Description:



6. Click **Save** to update the changes.

Method 2

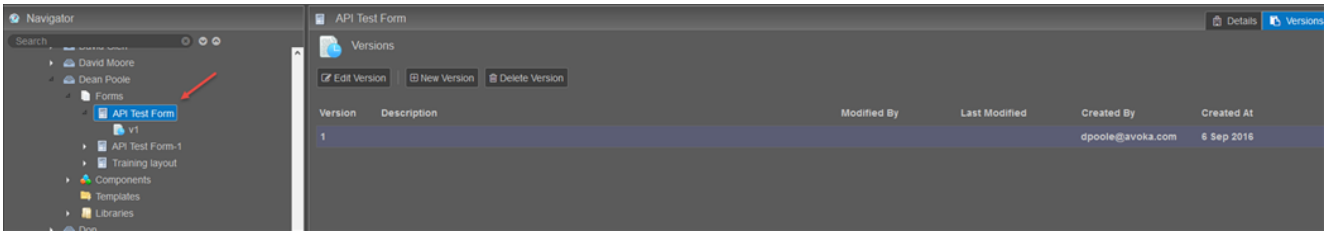
1. Open Maestro to see the list of Organizations.



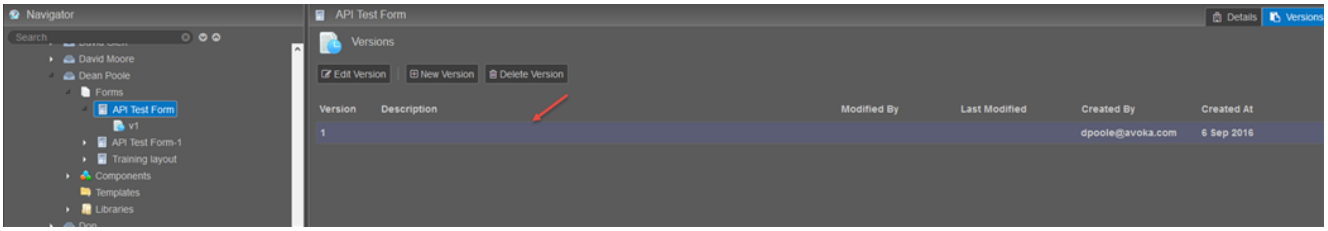
2. Select **Organization > Forms**.
3. Locate your form and expand it so you can see any versions that apply. You will see either your existing description of that version or none at all.



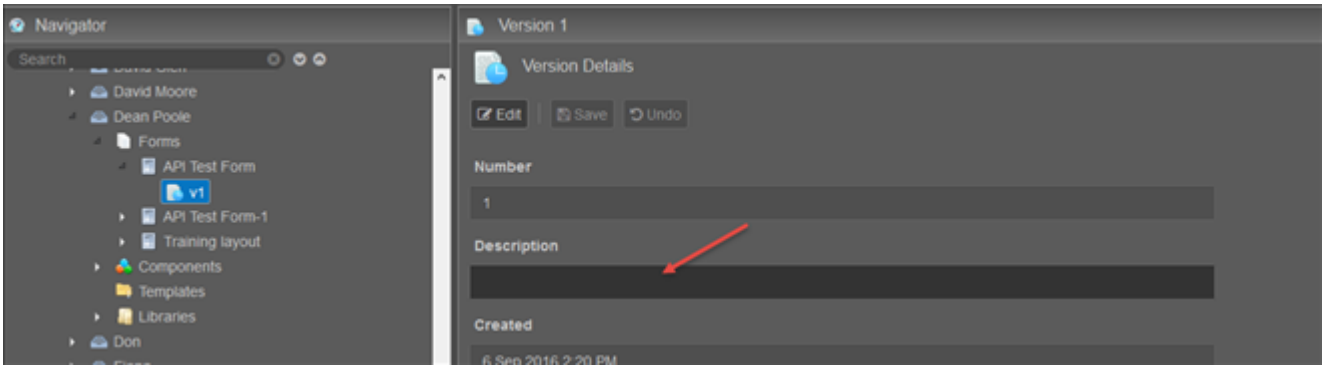
4. Double click in the list and click on the **Form Name** in the Navigator pane.



5. Double click on the **Version** to be updated.



6. Change or add your description in the **Description** field.



7. Click **Save** to update the changes.

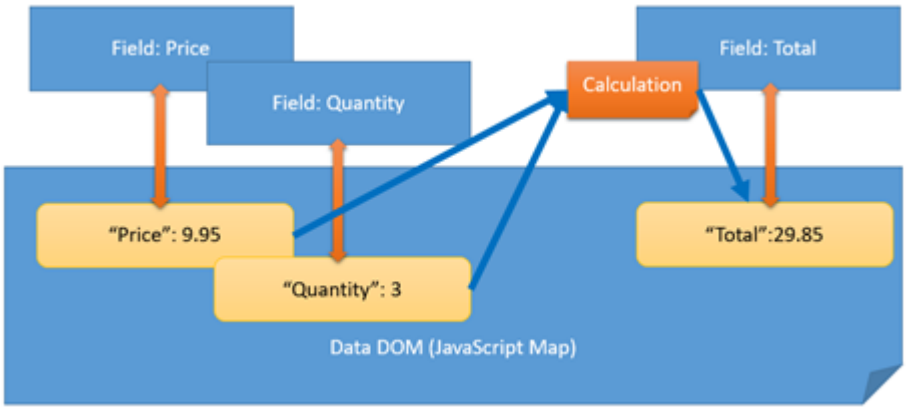
Next, learn how to open a Maestro form.

Maestro Data Models

Maestro | Form Builder | 5.1 & Higher

The Maestro data model is a powerful yet simple abstraction that allows you to use much of the Journey Maestro functionality when you are developing a form, so it is important to understand it.

When you type a piece of data into a component in Journey Maestro, the data is not only displayed in the component itself but is also copied to an internal data structure known as the data model, or the data **DOM**[1]. The component and its value are sometimes known as the view. These terms both derive from well-known programming pattern Model-View-Controller. In our case, the controller is any business rule that is defined in Maestro, as shown below.



Maestro uses a unique field ID to identify both the component and its value in the data model. Internally, the data model is implemented as a JavaScript map. You can think of the data model as being a spreadsheet with two columns: the ID of the component and its value. This is illustrated with a sample table.

Field ID	Value
"Price"	9.95
"Quantity"	3
"Total"	29.85

[1]The Document Object Model (DOM) is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document. The objects can be manipulated programmatically and any visible changes occurring as a result may then be reflected in the display of the document.

The component's value and the value in the data model are automatically synchronized in both directions. Therefore, when you are programming in Maestro, it is usually much easier to modify the data in the data model and let the automatic synchronization take care of updating the visible values in the components themselves. If you refer to the above diagram, you will see that the calculations refer to the values in the data model, rather than referring directly to the components themselves.

For example, when you are in the Script Editor, and double-click on the price component, what is inserted into your script is "data.price". This is not actually referring to the component itself, but rather referring directly to the component's value in the data model. The "data." is the clue that you are referring to the data model. The values in the data model can be both read and written, and the synchronization will take care of updating all the components in the view automatically.

To learn how to use Data Mapper in Journey Maestro, see data mapper.

It is important to ratify your data model upfront as much as possible before setting out - and do not assume that a model which represents a downstream integration is a good way to operate. For more information on data XML location, see data XML location.

For example, if you are integrating with Vendor A, and Vendor A requires the following format of output for "pets":

```
<Classification>
   <PartA>
      <Pets>
         <Pet>
            <Name/>
            <Type/>
         <Pet>
      </Pets>
   </PartA>
<Classification/>
```

It may be tempting to include this structure as part of your form. This has the benefit of making the payload meet the requirements for integration with little work. However, the forms will suffer as small tweaks to the schema for output require republishing every form.

Consider a client data dictionary, as it has several benefits, including:

- Ownership - is in the client's estate, they have to agree on data definitions up front and it removes uncertainty.
- Integration – switching vendors in and out becomes less problematic as you have one working model and various translation layers.
- Certainty – over time, model ownership synthesizes to good working practice, extensibility, and surety of operation.

When working with complex data structures, you can't set complex JSON (arrays, nested structures) into a data field bound to the payload XML (saved data to Manager) and expect it being prefilled back on resume as a JSON object value. This is not supported in Maestro. Any value stored in an XML bound data field must be of a primitive type which can be easily serialized as a string (int, float, string, boolean). If you do store a complex JSON to a bound data field, Maestro will try to recreate the JSON structure as XML nodes with elements structure. It will be still saved to Manager as form XML but when resuming a form, Maestro won't recreate the JSON object from the XML data.

To go around this, you can choose one of the following recommendations:

- Try to use repeats which mimic your JSON structure using Maestro widgets and bindings for data storage and have unbound data field as the data source for your data drive implementing calculate rules from the data in the repeat.
- Store the JSON to a bound data field as a serialized JSON string via **JSON.stringify()** but you would need to decentralize it back from the bound data field into another one used as the data source for you data driven widget on form load.

Next, learn about the unified application data document model.

Data Mapper

Maestro| Form Builder | 5.1 & Higher

The Maestro editor allows you to map an existing client schema, represented with an XML file, into a form. This is a quick and easy way to set up a form compatible the desired submission output. For more information, see form data XML.

To map form fields to submission data:

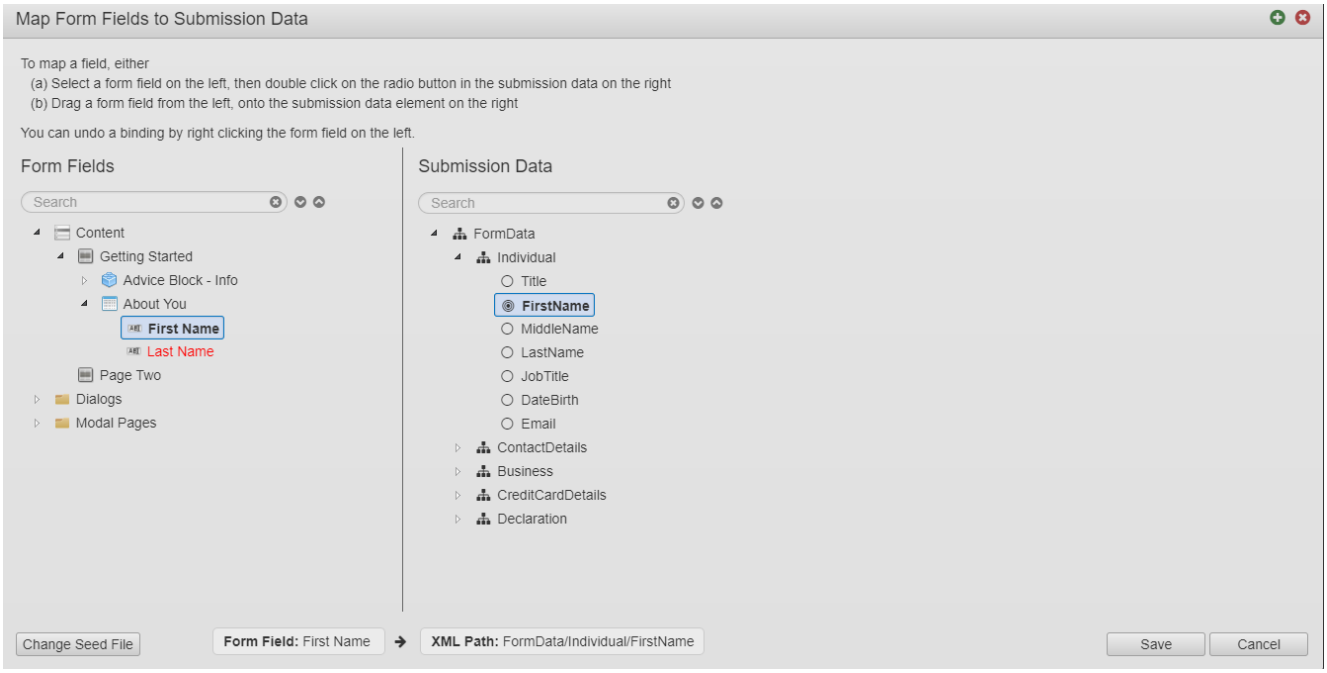
1. Open a form in the Maestro editor and click **Settings Menu**.
2. Select **Open Data Mapper** to map form fields to submission data.



3. Click **Select Seed Document** to choose an XML file to upload from your PC. A file content can be like one shown below:

```
<root>
	<firstName/>
	<lastName/>
</root>
```

4. Select the **Clear existing bindings** checkbox to clear data bindings.
5. Click **Load** to upload the file.



6. Map the components from the **Form Fields** column to the **Submission Data** column by dragging it across.
7. Click **Save**.
8. Click **Build and Render Form** to see how this mapping affects the form

Next, learn about form XML data payload.

Data XML Location

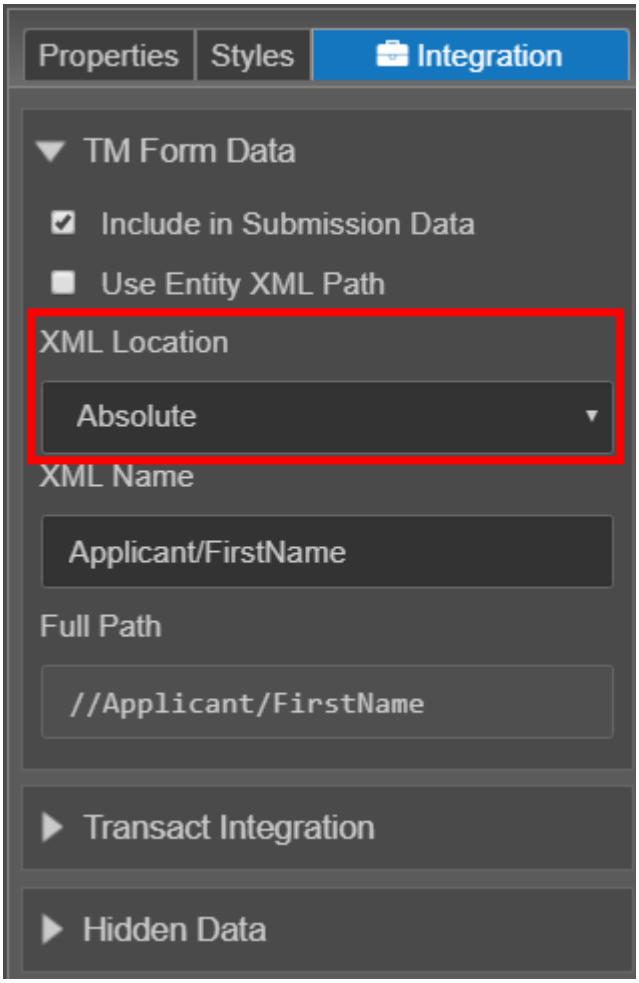
Maestro | Form Builder | 5.1 & Higher

Journey Maestro allows you to specify the following XML Locations for some components:

- **Relative** - when the component is dragged between sections and blocks, it assumes a hierarchy relative to its location and thus break the data model enforcement.
- **Absolute** - to guarantee the location of the XML data upon submission of the form, you must ensure that in all cases, other than inside repeating elements, where the component is not associated with an entity, that the XML location is set as Absolute.

To change the XML Location of a component, follow these steps:

1. Open a form and select a component.
2. Click the **Integration** tab.



3. Expand **TM Form Data** and select **Include in Submission Data**.
4. Clear the **Use Auto-generated XML Name** checkbox.
5. Select **Absolute** from the XML Location dropdown list.

6. Update the **XML Name**.

7. Click **Build and Render Form** to see how this rule is displayed when the form is built and rendered.

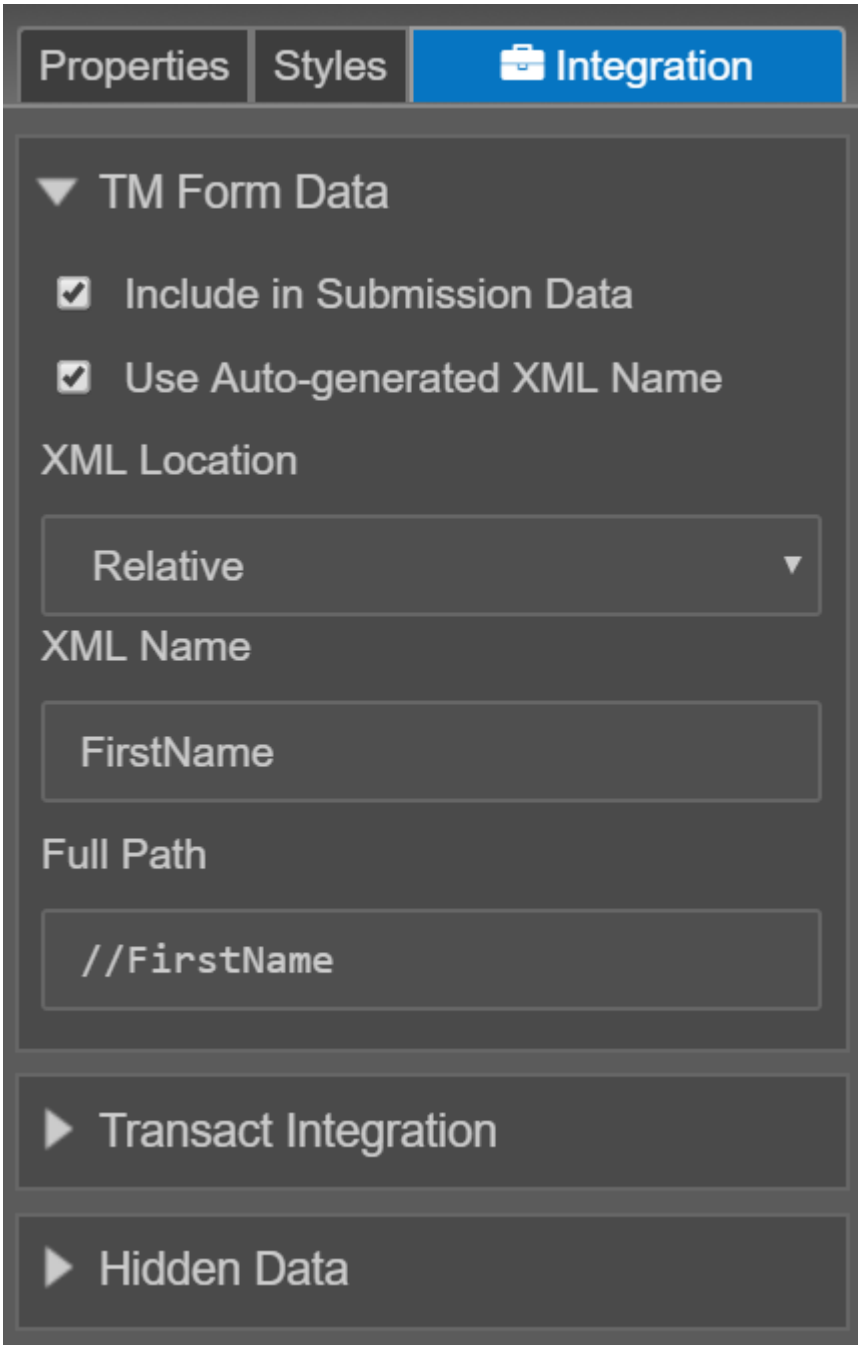
Next, learn about Form Xml Data Payload.

Component XML Name and Entity XML Path

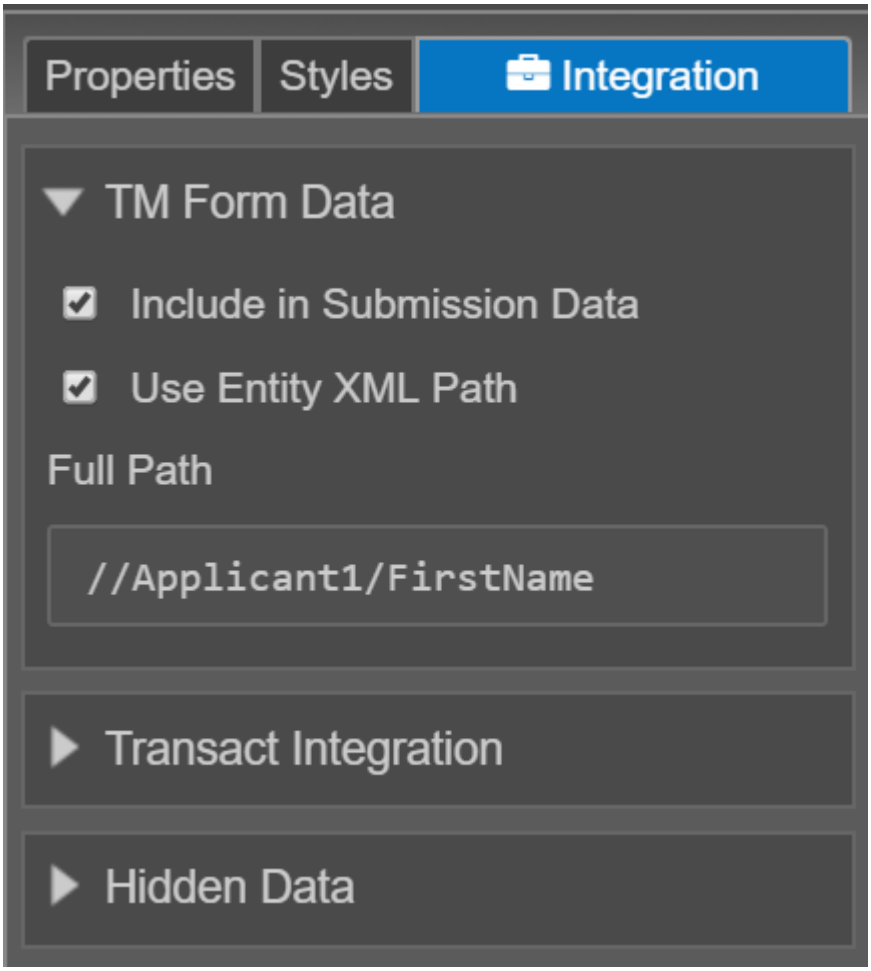
Maestro | Form Builder | 5.1 & Higher

Journey Maestro's **Integration** pane shows a component's XML Name and Full Path properties. By default, XML Name and Full Path are populated based on the component ID value. You can see the difference between an ID without an entity and one with an entity below:

ID without an Entity



ID with an Entity



If **Use Auto-generated XML Name** is not selected, you can change the XML Location, and XML Name and Full Path are not automatically updated when the component ID changes.

If **Use Entity XML Path** is not selected, you can change the Full Path and it is not automatically updated based on the component ID changes.

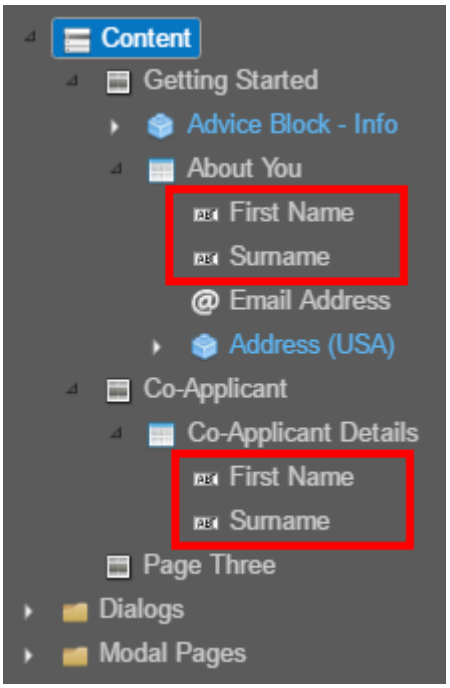
Next, learn how Journey Maestro data model.

Form XML Data Payload

Maestro | Form Builder | 5.1 & Higher

Each Journey Maestro form has the XML Data Payload, which is the XML layer that represents the user's input data of the form and is used for external systems integration. This layer is created when a form is built. Most components are added to the XML layer by default, except for pages, sections, blocks, and data fields. For more information, see data model overview.

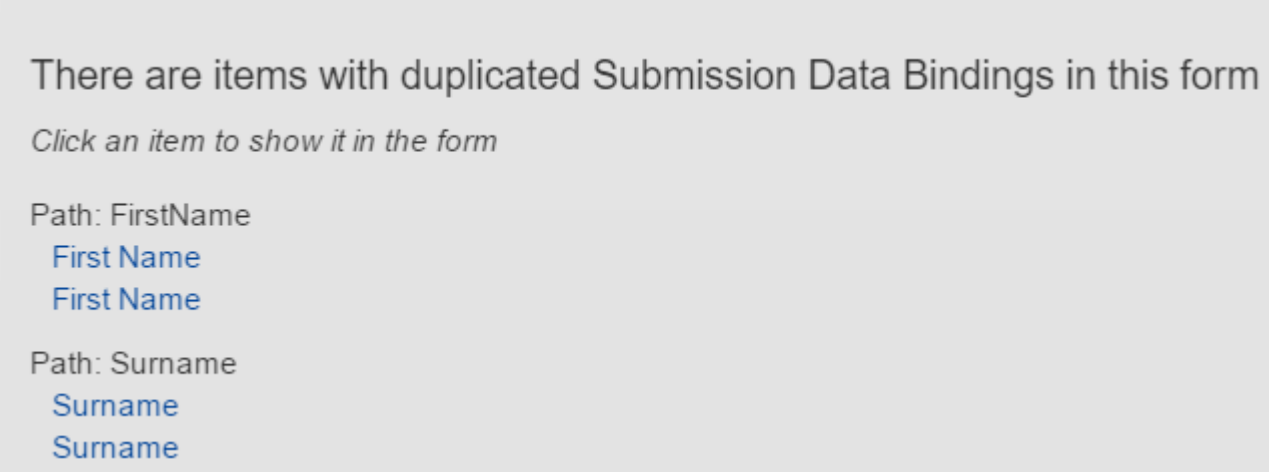
The default form XML data payload is a flat structure XML document, which only includes components, but not the hierarchy they belong to. This is illustrated by the following form with two fields, First Name and Surname, that appear in two different sections of the form.



By default, the XML tags created for each of these items are identical: FirstName and Surname present twice in the XML document, as shown below.

```
<AvokaSmartForm>
    <FirstName>
    </FirstName>
    <Surname>
    </Surname>
    <EmailAddress>
    </EmailAddress>
    <AddressUSA>
    </AddressUSA>
    <SFMData>
```

Only the first XML tag is created for First Name and for Surname, so this results in a duplicate bindings error.



Next, learn about component data binding.

Submission Data Extract Overview

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager provides submission data extracts that allow the form builder to configure certain components that can be extracted from the **XML**[1] payload and then saved in an easy to read format, such as an Excel spreadsheet. This data extract can then be used for integration with external systems or distributed via emails for further processing.

Submission data extracts usually represent only a subset of the form data, not all of it. To select which components are included in the data extract, you need to configure them either in Maestro or Manager.

Submission data extracts can also be used in:

- Collaboration jobs. You can pass information from a form to a collaboration job, for example, to customize email messages by taking the user's first name and adding it to the email's greeting.
- Overlay receipts. Forms can be implemented as pixel-perfect receipts by using data extract values in **AcroForms**[2].
- Card. A card is a repeating element in the form space, which can be customized from data in the form using data extracts.
- Form spaces' searches. You can use data extracts when searching form spaces. For example, you can search for a task that has been assigned to you by using the data extracts and not the entire XML file.

Form data extracts are a great way to quickly inspect key data elements in Manager and allow you to make business decisions or meet reporting requirements.

There are some best practices when using form data extracts:

1. Do not extract **Personally Identifiable Information**[3], because it might be downloaded as a bulk **CSV**[4] file using Manager. Using data extracts in this way is not the intended design, rather it is used for reporting purposes for generalizing on a particular set of data, or testing. It is best to keep any PII out of the data extracts, and rather send this data to your secure backend systems for processing.
2. Do not extract every single piece of data in your form using data extracts. There are many other ways of integrating data into systems and sending your form data to your processing areas. Extracting all the data in

---

[1]XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
[2]AcroForms are the original PDF forms technology that accepts input in both Forms Data Format (FDF) and XML Forms Data Format (XFDF). Many 3rd party vendors support AcroForms.
[3]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.
[4]CSV is a comma-separated values fil, which s a delimited text file that uses a comma to separate values. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas.

your form using submission data extracts adds a level of development and ongoing maintenance overhead, which is not necessary given the many other options to retrieve your data.

Next, learn how to configure component data extracts.

View Form Submission Data Extract

Journey Manager (JM) | Platform Developer | 5.1 & Higher

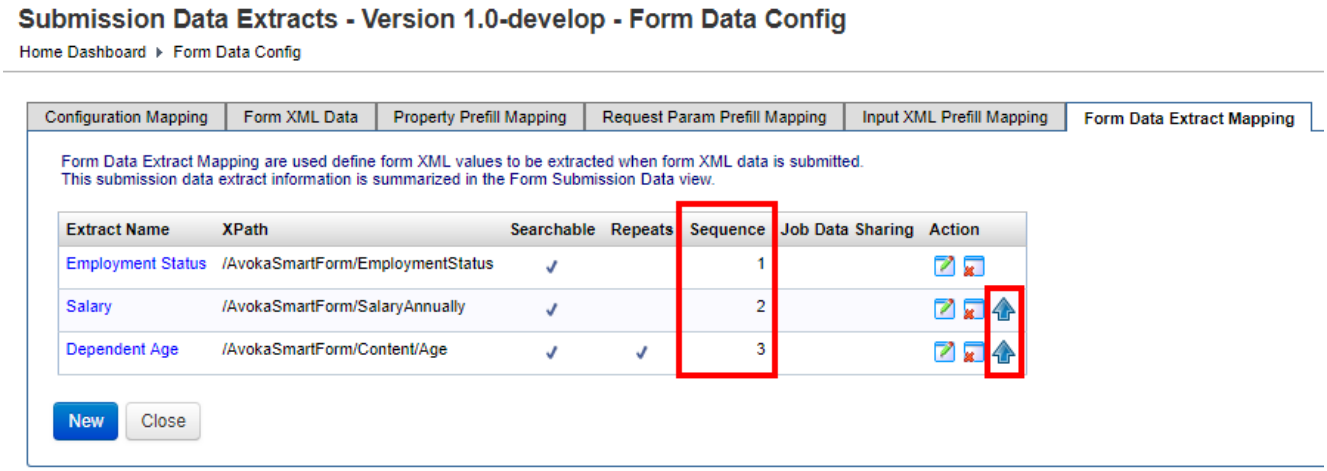
Manager allows you to view data extracts of any submitted form.

To view submission data extracts:

1. Go to the Home Dashboard.
2. Locate a form transaction from the **Latest Transactions** section and click **ID**.
3. Select the **Form Data Extract** tab. For more information, see view form data extract.

To configure and rearrange data extract mapping:

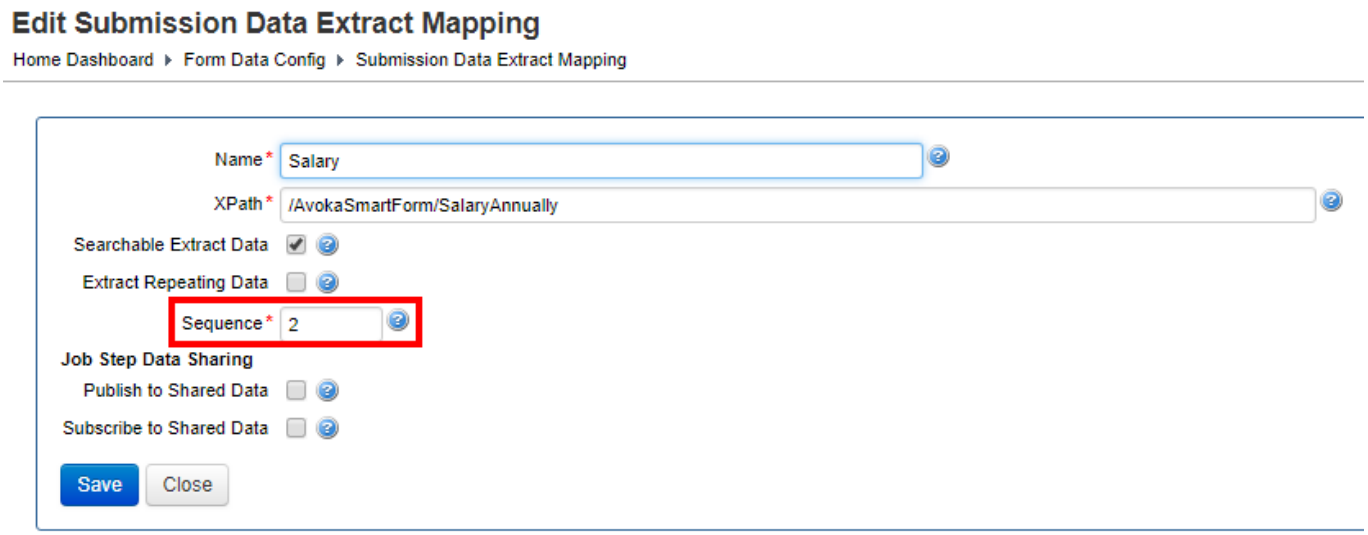
1. From Form Versions section, select a form version and then select **Data Config.**
2. Select the **Form Data Extract Mapping** tab.



3. To change the order that the extracted submission data appear in the table in Form Data Extract tab, use the arrows to move data items, one at a time, up in the sequence. This is the easiest way to reorder the data extracts.

The sequence can also be changed in each submission data extract mapping. Select Extract Name, update Sequence (but must be unique) and click **Save**.

This order only applies to the data extracted and does not rearrange the component order on the Maestro form as displayed to the user.



Next, learn about form prefill options.

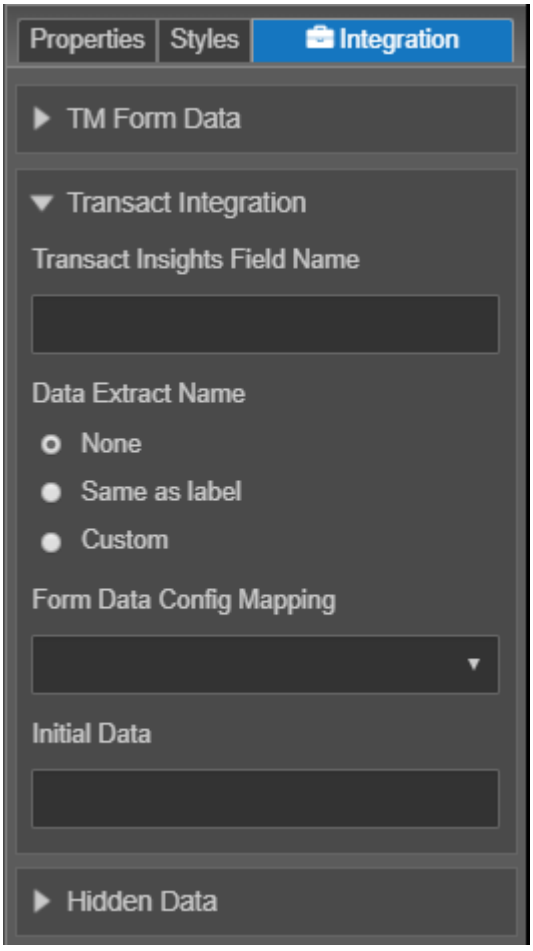
Configure Component Data Extract

Maestro | Form Builder | 5.1 & Higher

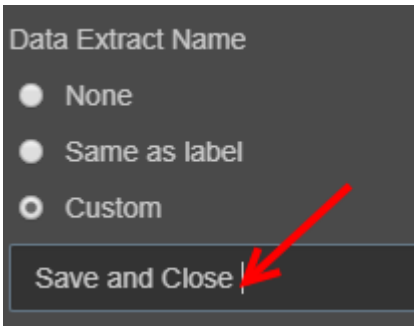
Manager allows you to configure data extract options for each component, which can be used during the integration.

To configure a component data extract:

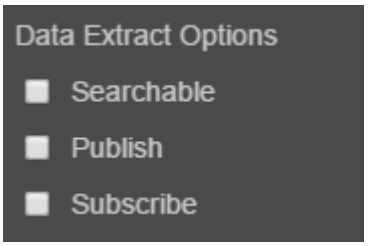
1. Open a Maestro form.
2. Select a component.
3. Select the **Integration** tab. For Maestro releases prior to 17.10, select the **Data pane** tab.
4. Expand **Integration** and edit the **Data Extract Name** options.



- **None** doesn't include the component in the data extract. This is the default option.
- **Same as label** includes the selected component in the data extract and it uses the component's label name as the extract name, for example Save and Close, in the Manager.
- **Custom** includes the selected component in the data extract and provides a text field for you to enter a component name to be used in Manager as shown below.



5. Expand the **Data Extract Options**, which only visible if **Same as label** or **Custom** is selected.



- **Searchable -** the whole data extract value can be used for as search criteria in the Manager.
- **Publish** is used by Collaboration Jobs in conjunction with the Subscribe option. Using Publish and Subscribe is an alternative mechanism to share data between tasks in a Form Bundle - Job Step.
- **Subscribe** consumes the publish event using the published value.

6. Click **Build and Render Form** to see how this rule is displayed when the form is built and rendered. Test out the form data extracts by completing and submitting a form.

Next, learn about submissions data extract.

Form Data Configuration Overview

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager uses **XML**[1] format to represent data it is working with. Manager automatically generates a default XML data structure to handle the data passed between the server and a form. When a user selects a form, Manager starts to controls the data flow that can be one of the followings:

- Standard default form processing.
- Customized form processing, which you can configure. This is the most common form of processing,

Form data configuration addresses one of a common requirements which is to prefill a form with user's details.

Before proceeding with data configuration in Manager, there are several data settings that should be configured in Maestro. Once the Maestro form version is built and published, these data configuration is transferred to Manager.

The list below identifies details on how to customize forms in Manager for the following data requirements:

- Prefill a form with user details using one of the supported prefill methods.
- Property Prefill Mapping.
- Request Param Prefill Mapping.
- Input XML Prefill Mapping.
- Capture and store additional details that are passed along with the form submission that may be required for handling such things as attachments, receipts, and anonymous submissions. Refer to Configuration Mapping for more information.
- Provide an alternative data structure, according to your enterprise's requirements, to the default XML. Refer to Form XML Data for more information.
- Extract some details for easy access for monitoring and managing form submissions. Data extracts are also required for some collaboration job features.

Form Builders configure these data customizations by defining data mappings from Manager or Maestro consoles. This is so that individual data elements can be externally defined and mapped to your form's XML data structure when the form is rendered.

Next, learn how to view form data configuration.

---

[1] XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

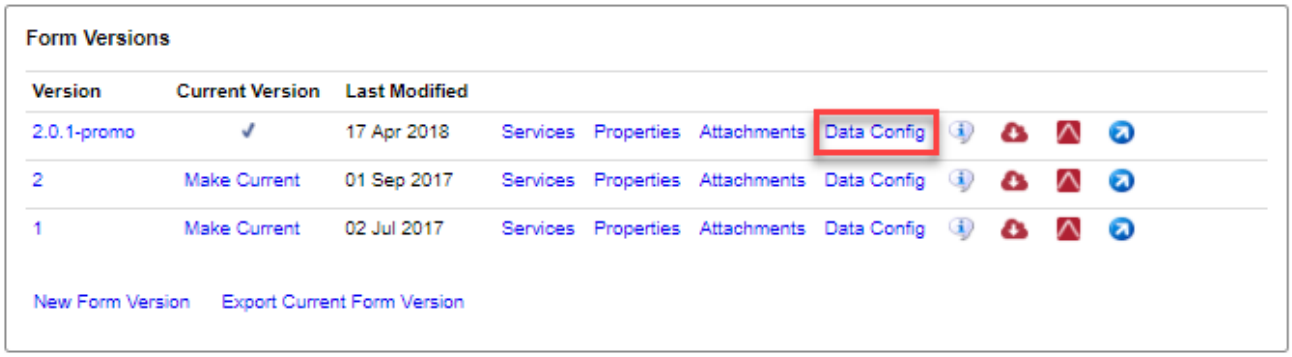
View Form Data Configuration

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to view data configuration for each form version.

To view data configuration of a form version, follow these steps:

1. Select **Forms > Forms.**
2. Locate the form and go to the **Form Versions** section.
3. Click **Data Config**.



The action icons displayed against each form version differ in Manager prior to v17.10.

To view data configuration of a form version from the **Form Versions** tab:

1. Select **Forms > Forms.**
2. Locate the form and click the **Form Versions** tab.
3. Click **Form Data Configuration** .

Next, learn about form configuration mapping.

Form Configuration Mapping

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager supports the following form operations:

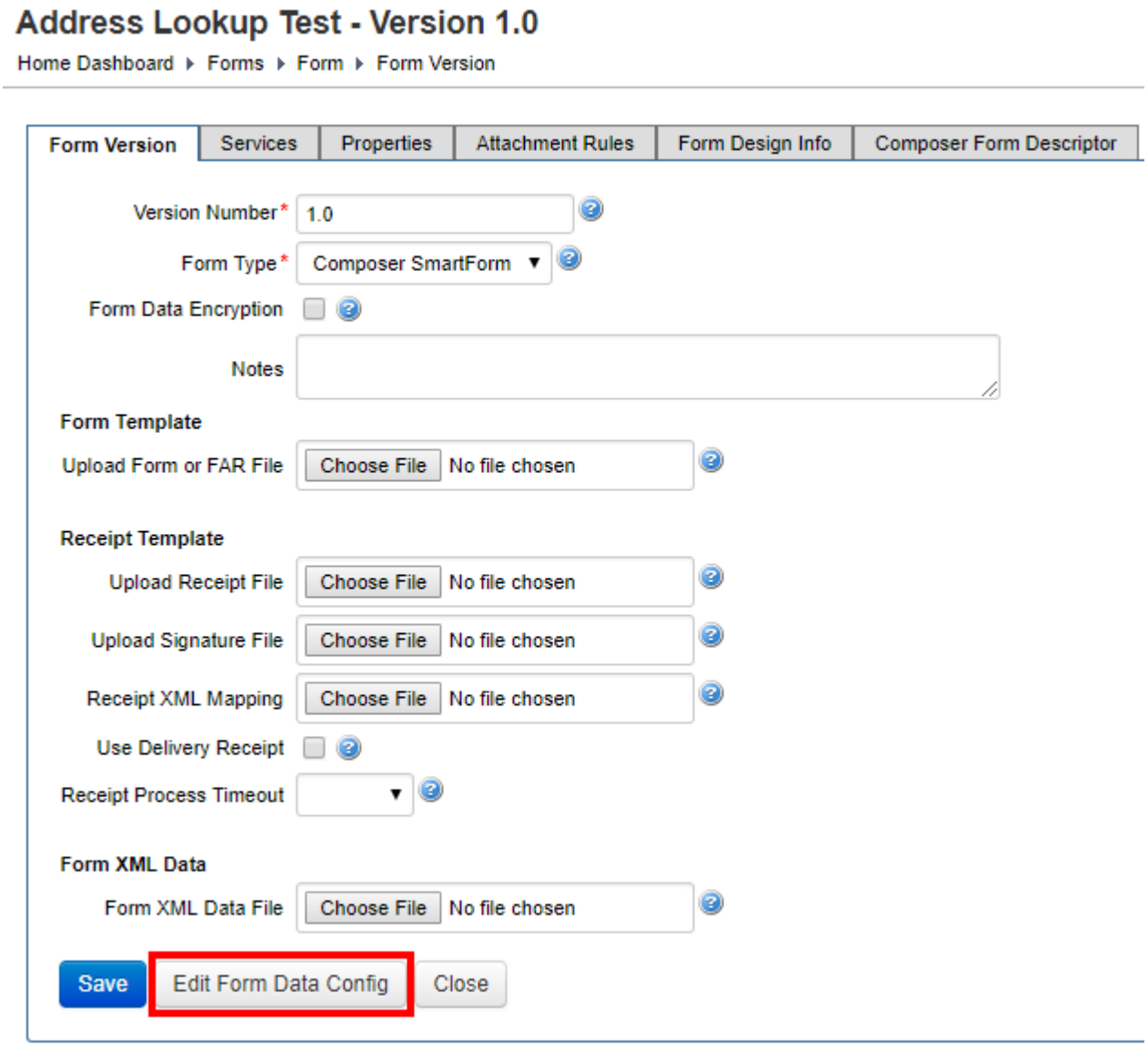
- Attachments
- Payments
- Signatures
- Email verifications
- Form saving and retrieval
- Anonymous submissions

This mechanism provides some data elements to pass key details into and out of a user's form session.

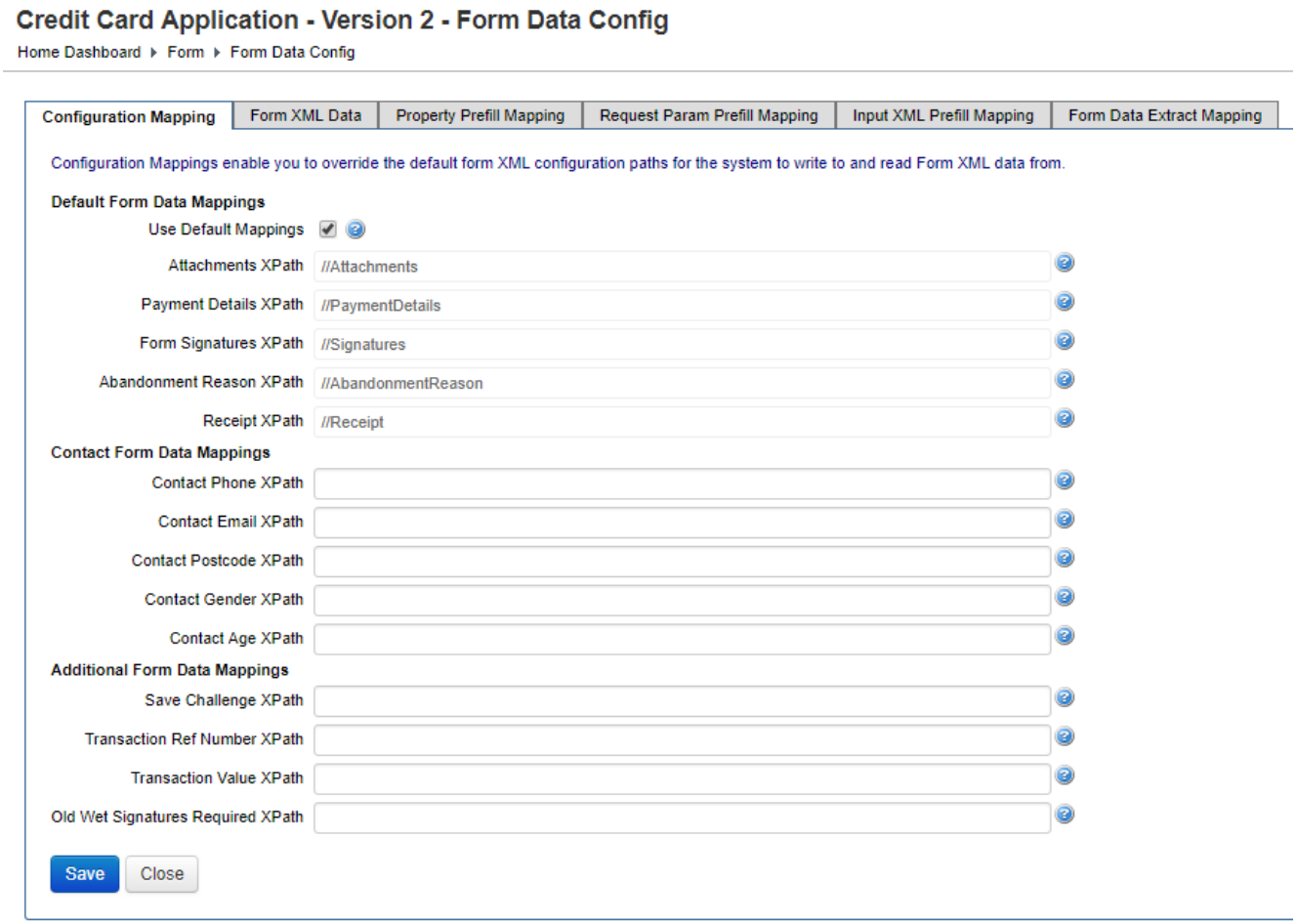
> Configuration mappings enable you to override the default form XML configuration paths for the system to write to and read from XML data form.

To configure form version's configuration mappings:

1. Select **Forms > Forms**.
2. Select a form and then click **Edit**.
3. Select a form version and click **Edit Form Data Config**.



4. Click the **Configuration Mapping** tab.



5. Select the **Default Mappings** checkbox to use default Configuration Mappings for attachments, payment details, receipts, and signatures. Clear this checkbox to change the defaults for the following five **XPath**[1] locations.

> If Use Default Mappings is not selected, you can change the default XPath in the Form XML Data.

6. Use **Default Mappings** - If selected, Manager will use default Configuration Mappings for Attachments, Payment Details, Receipts, and Signatures. Only when this is unselected, may you change the defaults for the following five XPath locations.
7. **Attachments XPath** - If Use Default Mappings is unselected, you may change the default XPath in the Form XML Data that contains the Attachments element.
8. **Payment Details XPath** - If Use Default Mappings is unselected, you may change the default XPath in the Form XML Data that contains the **`PaymentDetails`** element.

---

[1]XPath (XML Path Language) is a query language for selecting nodes from an XML document. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document. XPath was defined by the World Wide Web Consortium (W3C).

9. **Form Signatures XPath** - If Use Default Mappings is unselected, you may change the default XPath in the Form XML Data that contains the Signatures element. This element determines the type of Form Signature required. This must have a **`SignatureTypeSelected`** child element specifying the type of signature required.
10. **Abandonment Reason XPath** - If Use Default Mappings is unselected, you may change the default XPath in the Form XML Data that contains the **`AbandonmentReason`** element.
11. **Receipt XPath** - If Use Default Mappings is unselected, you may change the default XPath in the Form XML Data that contains the Receipt element.
12. **Contact Phone XPath** - The XPath that points to the element in the Form XML Data where the user's contact telephone number is stored. This may also be defined in Maestro.
13. **Contact Email XPath** - The XPath that points to the element in the Form XML Data where the user's contact email address is stored. This may also be defined in Maestro.
14. **Contact Postcode XPath** - The XPath that points to the element in the Form XML Data where the user's contact postcode is stored.
15. **Contact Gender XPath** - The XPath that points to the element in the Form XML Data where the user's contact gender is stored.
16. **Contact Age XPath** - The XPath that points to the element in the Form XML Data where the user's contact age is stored.
17. **Save Challenge XPath** - The XPath in the Form XML that contains the Save Challenge element. This may also be defined in Maestro.
18. **Transaction Ref Number XPath** - The XPath in the Form XML Data that points to the element that contains the transaction reference number which will be stored with the submission. This is provided for backward compatibility.
19. **Transaction Value XPath** - The XPath in the Form XML Data pointing to an element that contains the numeric value of this transaction.
20. **Old Wet Signatures Required XPath** - The XPath in the Form XML Data pointing to an element that contains true if the form requires wet signatures.
21. Click **Save** to update the changes.

Next, learn how to configure form XML data.

Form XML Data

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to view and edit form's data, in a machine-readable XML format, representing a form and form user's input provided into the form. Each Maestro form that you deploy into Manager contains a **`formseed.xml`** archive file describing all relevant form data in an XML format. If you have specific data schema requirements, you can edit the form's XML data structure or upload a form seed file to update the form XML data. When you export a form from Manager, the archive file also contains a **`formseed.xml`** file so you can check and modify the XML data structure as required.

The form XML data structure varies depending on a form content, but it usually includes the following elements:

- **AvokaSmartForm** - the root element that incloses all other tags. The example is shown below:

```
<AvokaSmartForm>
...
</AvokaSmartForm>
```

> If a form is configured to uses **Unified App Data**, then **<Root>** is the root element of the form XML data.

- **SFMData** - Smart Form Data that contains transaction data, such as **formCode**, **TemplateVersionNumber**, **ReceiptNumber**, **PaymentDetails**, and so on. The example is shown below:

```
<SFMData>
    <SystemProfile>
      <DisplayMode/>
      ...
    </SystemProfile>
    <PaymentDetails>
      <Payment>
        <SubTotal/>
        <GST/>
        <ServiceFee/>
        <Total/>
      </Payment>
      <PaymentType/>
      ...
 </PaymentDetails>
</SFMData>
```

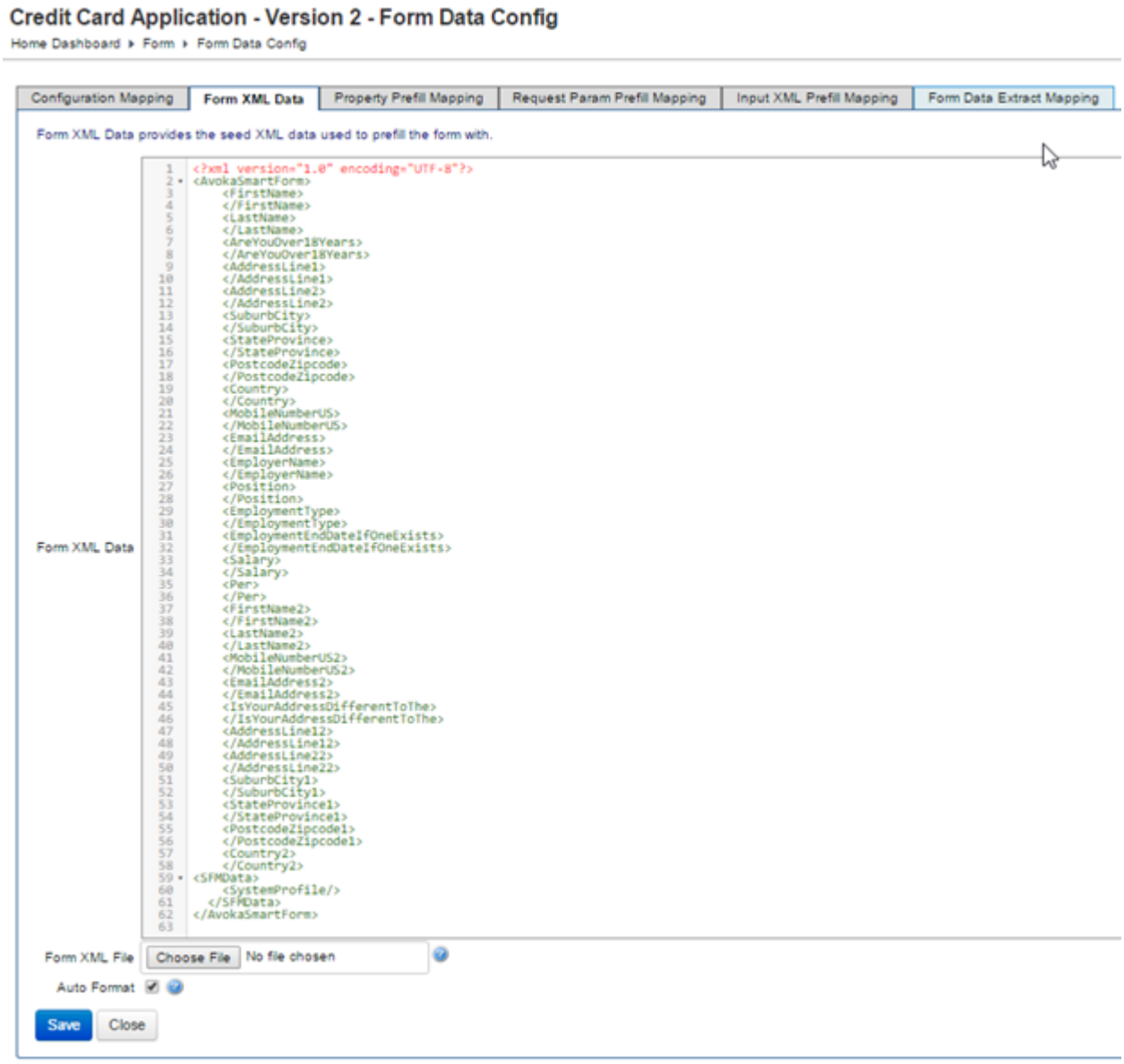
- Form specific elements, such as **FirstName**, **LastName**, **EmailAddress**, and so on. The example is shown below:

```
<FirstName/>
<LastName/>
<DateOfBirthDropdown>
    <Day/>
    <Month/>
    <Year/>
    <Date/>
</DateOfBirthDropdown>
<EmailAddress/><AddressAustralia>
    <AddressLine1/>
    <AddressLine2/>
    <City/>
    <State/>
    <PostCode/>
</AddressAustralia>
```

You can download the complete example of form XML data here.

To view or edit the form XML data:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select a form version and click **Edit Form Data Config**.
4. Click the **Form XML Data** tab to view the XML data.



5. Edit the XML content directly in the **Form XML Data** tab or click **Choose File** to upload a seed XML file.
6. Select the **Auto Format** checkbox to automatically format the XML data. Otherwise, it retaina the original XML format when saving it.
7. Click **Save**. Manager will validated the updated XML file for any syntax errors before saving it.

Next, learn about form property prefill mapping.

Form Seed

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager generates a form seed XML file to help you create and edit form XML files for your applications.

One of the issues with XML Schema is that it's really difficult to read and interpret (by humans and by computers), and this is one of the reasons why we use seed files rather than XMLSchema or RelaxNG or DTD.

However, I was wondering whether we can get 80:20 of what we need very simply by having some simple "annotations" in the XML seed file.

- There are actually very few constraints we're interested in – max length, data type, regular expression, maxoccurs, enumeration.
- This isn't industry standard, but it's simple.
- If we can write a simple processor to process XSD and spit out the annotated seed file, that could go a long way.
- It gets around the complexities of multiple file imports and namespace issues.
- We could use this in Composer/Maestro, and we could also enforce these rules quite easily on the server.
- We could support two styles, storing the restrictions in either attributes or data.
  - Attributes are good but they don't work if fields are bound to attributes rather than elements (we do support this in Composer.)
  - Data is easy but it doesn't work for repeats. So you need both.
  - If we want to keep it simple we say "no binding to attributes", but this may not work for many pre-existing schema.
- We could potentially quite easily spit out both a simple XML Schema and also an annotated XML seed file directly from Composer – we already have all this information.

Example:

Here's the simple seed file with data

```
<root>
  <first_name>John</first_name>
  <last_name>Citizen</last_name>
  <age>61</age>
  <tfn>123456</tfn>
  <children>
    <child>
      <name>Jack</name>
      <birth>2000-01-01</birth>
    </child>
  </children>
</root>
```

Here's an augmented version:

```
<root>
  <first_name>maxLength="20" type="xs:string"<first_name>  <!-- restrictions in the data
-->
<last_name maxLength="20" type="xs:string"></last_name> <!-- restrictions in the attrib-
utes -->
<age type="xs:integer" minInclusive="18"></age>
<tfn>pattern= "[a-zA-Z0-9]{8}"</tfn>
<children>
  <child max_occurs="5">
    <name>maxLength="20"</name>
    <birth>type="xs:date"</birth>
  </child>
</children>
</root>
```

However:

- Adobe-Designer already does this. They have an augmented XML seed file that they call their Data Description. They use this to drive the tooling for data binding as well as their Web Service implementation.
- They use attributes with a namespace to avoid naming conflicts, which is an excellent idea.
- This means that we can also re-use the actual data within the seed file to specify the initial value of a field.

Here is a snippet of Adobe Data Description:

```
<dd:dataDescription xmlns:dd="http://ns.adobe.com/data-description/" dd:-
name="AvokaSmartForm">
    <AvokaSmartForm>
         <Plate dd:maxOccur="10">
            <Line1>
              <NewPlate/>
              <ReplacementPlate/>
              <AgencyOrderNumber/>
              <IsAssetWCF/>
            </Line1>
```

There are options to still specify meta-data on attributes, by prefixing the metadata attribute something like this:

```
<customer age="" dd:datatype="xs:string" dd:datatype_age="xs:integer" />
```

- dd:datatype refers to the element "customer"
- dd:datatype_age refer to the attribute "age"

Next, learn about form property prefill mapping.

Form Data Extract Mappings

Journey Manager (JM) | Platform Developer | 18.11

Manager allows you to use the data extract feature to extracts form data and store it in the external data elements against the form submissions. Whenever a user submits a form with data extracts, this data is stored against each form's submission as a convenient way of viewing and accessing essential data from within Manager. These data extracts may assist in transaction monitoring and reporting, especially for operational support teams. Data extracts are also required for supporting collaboration jobs.

Manager allows you to define data extracts mappings for form versions and automatically populates them.

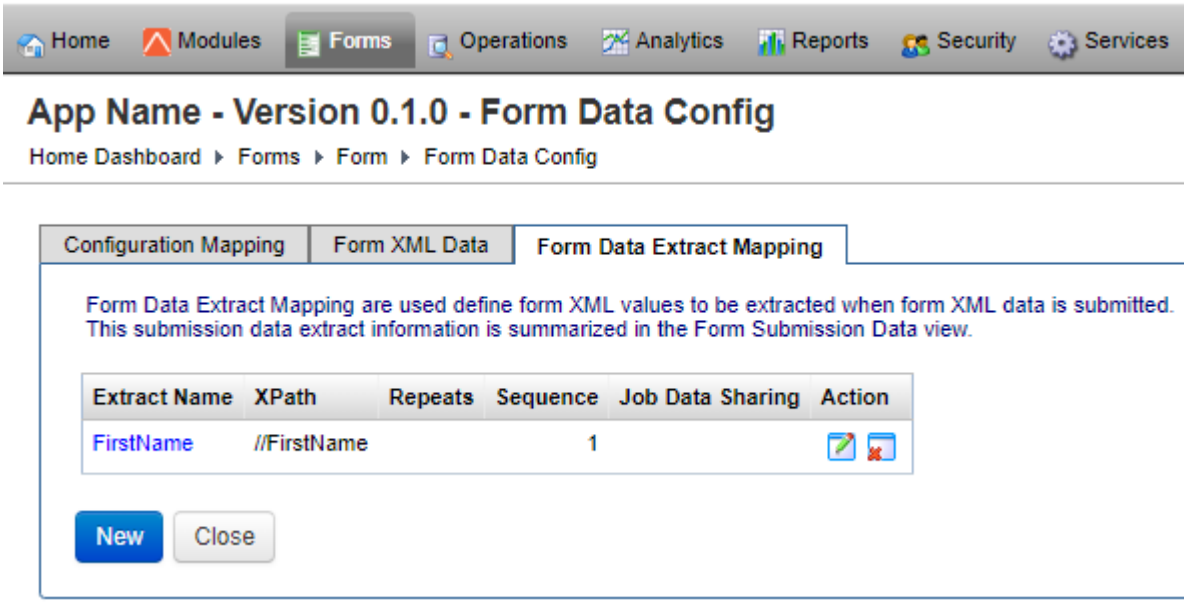
> We recommend to configure data extracts in Journey Maestro instead of Manager before proceeding. Do not configure data extracts in both Journey Maestro and Manager.

A data extract mapping specifies where to find the data to extract it from the form's **XML**[1], using an **XPath**[2], and how to name the extracted value, which will be stored in Manager.

To configure data extract mappings:

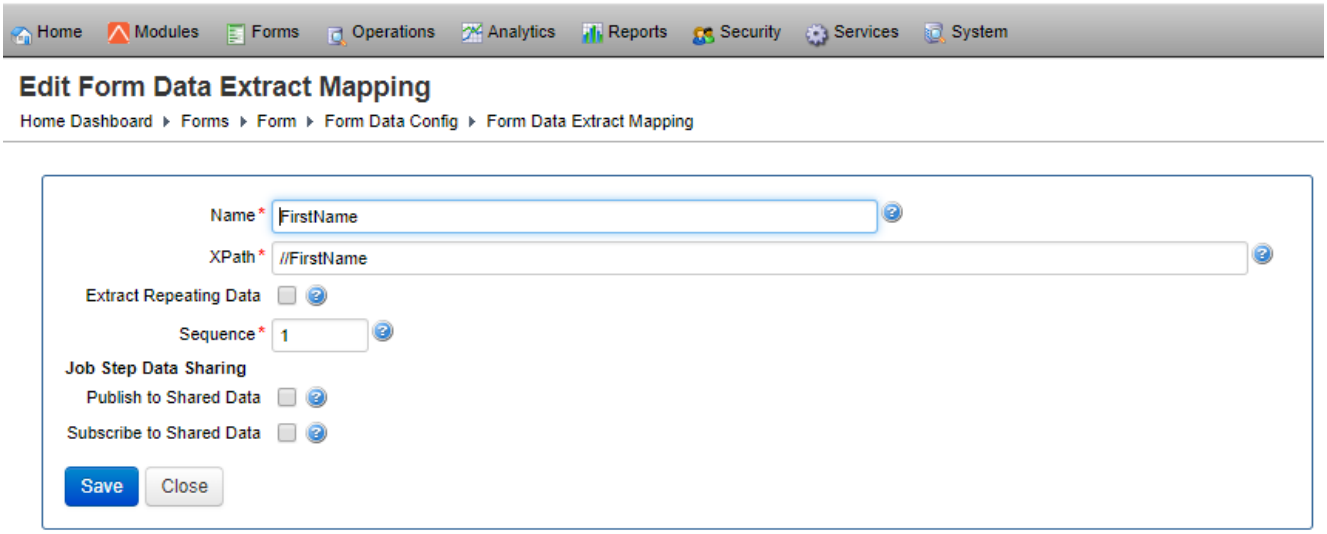
1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select a form version and click **Edit Form Data Config**.
4. Click the **Form Data Extract Mapping** tab to see the existing mappings.

[1]XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
[2]XPath (XML Path Language) is a query language for selecting nodes from an XML document. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document. XPath was defined by the World Wide Web Consortium (W3C).



Here, the extract field name is FirstName, so the extract reference used in the action property value will be **`$formDataMap.firstName`**, and the action property used in the job definition's will be **`{ "name": "Task Subject", "value": "Job Number: <b>$job.referenceNumber</b> $submission.form.formName From $formDataMap.firstName $formDataMap.lastName" }`**

5. Click **New** to create a new data extract mapping or click **Edit** to update the existing one.



6. Edit the label for the extracted, displayed or downloaded data item in the **Name** field.
7. Edit the XPath location in the **`formXml`** where the data item can be found in the **XPath** field. This is used in the **Publish / Subscribe** to update the **`formXml`**.
8. Select the **Searchable Extracted Data** checkbox to enable searching on encrypted data. For searching, the user must enter the full string, which is not case sensitive. | Removed in 18.11.

Manager 18.11+ is able to search encrypted PII data, so you don't need to mark form data manually whether it is as searchable or not.

9. Select the **Extract Repeating Data** checkbox to automatically extract all repeating XML elements in the form submission data extracts. This is only relevant for a data item extracted from a repeating component.
10. Edit the **Sequence**, which is the order that the extracted data item will display against the form submission. With the first data extract, it automatically starts at 1 and increments with each new data extract. This may be adjusted but must be a unique value. It is better to reorder using the **Reorder** icon from the **FormData Extract Mapping** tab.
11. Select the **Publish to Shared Data** checkbox to publish shared collaboration job step extract data to other step submissions. This is used by collaboration jobs in conjunction with the **Subscribe to Shared Data**, which is described below. It is configured in the Job Definition JSON in the Step by setting the `shareExtractData` attribute to true. Publish Subscribe is an alternative mechanism to share data between tasks in a form bundle's collaboration job step, which is made up of two or more separate tasks (submissions). When one task is submitted, Manager processes all the data extracts marked as **Publish** and puts the values in a map. It then goes through all the other form tasks in the step. Where the data extract has a value that subscribes, the data extract value will be overwritten with the published value. The `formXml` on the subscribing form will also be updated.

This is relevant for form bundles.

12. Select the **Subscribe to Shared Data** checkbox to subscribe to shared collaboration job step extract data changes. This is used by collaboration jobs in conjunction with the **Publish to Shared Data** configuration, which is described above. The subscribe consumes the corresponding Publish event using the published value. To use Subscribe, the XPath location needs to be set to a single node. You won't be able to use the double slash `//FirstName` to set the value using XPath. Data extracts specified when designing your form don't have the full XPath location.

This is relevant for form bundles.

13. Select the **Encrypt Extracted Data** checkbox to automatically encrypt the extracted data. This means that the value is unencrypted when accessed via the Manager console, rest API, as well as from entities within Groovy scripts such as delivery services. | Removed in 17.10.
14. Click **Save** to update the changes.

You can also perform the following actions:

- Delete a data extract mapping.
- Reorder the sequence by moving the mapping up one position.

Collaboration Job Contex

Manager passes a collaboration job's context to a form using the **SFMData.SystemProfile.Job** XML node. This is done during the form rendering. The snippet below highlights the **<Job>** XML node:

```
<?xml version="1.0" encoding="UTF-8"?>
<AvokaSmartForm>
    <SFMData>
        <SystemProfile>
            <DisplayMode>Receipt</DisplayMode>
            <HostContext>Page</HostContext>
            <ReceiptNumber>fct-3097-composer-3</ReceiptNumber>
            <SubmitDateString>12 Feb 2015 5:38:54 PM</SubmitDateString>
            <SubmissionMessage/>
            <FormDataSer-
viceURL>http://localhost:9080/maguire/servlet/FormDynamicDataServlet</FormDataServiceURL>
            <RequestLogKey>9c1b9756b3958671c9b2471e23f91d08</RequestLogKey>
            <TrackingCode>NVKMQ9</TrackingCode>
            <Job>
                <AvailableRoutes/>
                <Assignee/>
                <AssignRepeatIndex/>
                <AssignRepeatItem/>
                <ReferenceNumber/>
                <StepName/>
                <RouteName/>
            </Job>
            <FormCode>fct-3097-composer</FormCode>
....
```

The **<Job>** node has the following attributes:

AvailableRoutes

A collaboration job passes the **AvailableRoutes** property to the tasks created at a particular collaboration job step. The job definition below shows the 2 Step Review Job.

```
{
  "name": "Initial Review",
  "type": "",
  "actions": [
     {
       "name": "Create Task",
       "type": "Job Task Assign",
       "properties": [
         ...
       ]
     },
     {
       "name": "Handle Submission",
       "type": "Job Task Wait",
     }
  ],
  "routes": [
     { "name": "Approve",  "nextStep": "Application Delivery" },
     { "name": "Decline",  "nextStep": "Applicant Update" },
     { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "display": "false"
},
     { "name": "Terminate",  "nextStep": "Terminated Initial" }
  ]
}
```

When the collaboration job moves to the Initial Review step, it populates the form data with **<AvailableRoutes>Approve|Decline|Terminate</AvailableRoutes>**. The **Exceeds Threshold** route is not included due to the **display** item. This is used in a form as the read-only value.

ReferenceNumber

A **ReferenceNumber** is a Job Number field, which is displayed on forms with the Step Number as the heading. This is used in a form as the read-only value. For more information, see Composer documentation.

StepName

A **StepName** is a name of the current step that the collaboration job is currently at. The form can use this value to hide, show or set editability of fields and form sections. This is used in a form as the read-only value.

RouteName

A form user can set a new route in the form using a dropdown list or a radio button. If this is not set, the collaboration job uses the default route. This attribute is set by a form.

Next, learn how to test a new form data extract mapping.

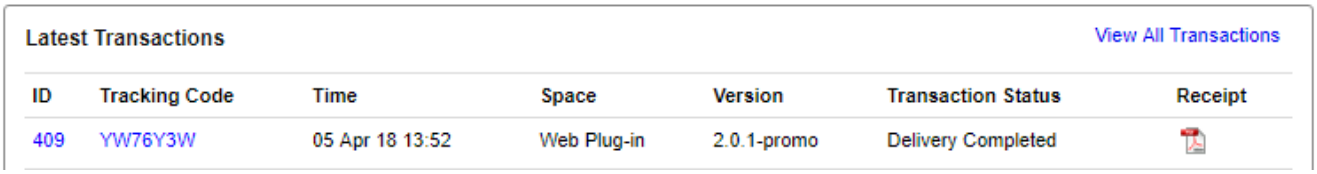
Test a Form Data Extract Mapping

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to test a form data extract mapping.

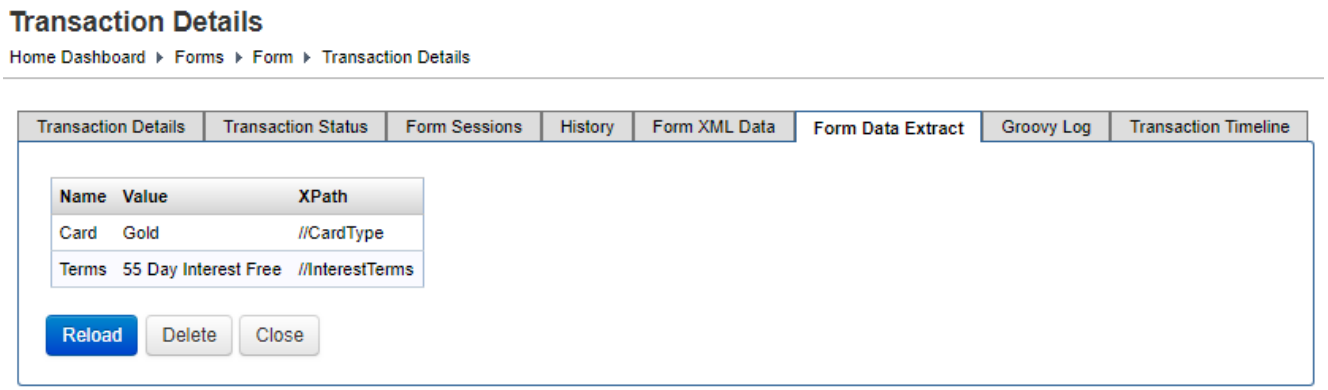
To test a data extract:

1. Render a form version.
2. Complete and submit the form
3. Select a credit card type and interest terms.
4. Locate the data extracts for this form submission.
5. Navigate to the **Latest Transactions** section and click **ID**.



You can view the same information from **Operations > Form submission Data**.

6. Click the **Form Data Extract** tab to view available form data extracts. In this example, a user of this form submission entered Gold for their Card and 55 Day Interest-Free for their Terms.



In Manager prior to v17.10, you can click the Form Submission Data icon from a form **Dashboard** to view form submissions. This functionality has been removed.

Form Prefill Options

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager offers several options to prefill a form with some data, including applicant data, when the form is rendered to the applicant. Form prefill is a great way to significantly improves user experience and reduce data entry errors. You need to configure or map form's data with prefill data, which will affect a form's XML at render time. Manager distinguishes the following mappings:

Property prefill mappings: client, form, user properties

This allows you to reuse data that is already stored in Manager:

- User data
- Organizational data (Reuse across multiple forms)
- Configure form specific properties without needing to customize or redeploy the form
- Can be configured to sync or update the user properties maintained in Manager

XML prefill mappings: external XML file containing prefill data

It provides the following benefits:

- Third party and external data
- Can create mappings to a particular XML schema

Parameter prefill mappings: HTTP request parameters, session attributes or cookies

It has the following advantages and disadvantages:

Pros:

Data can be provided as part of the URL (simplicity)

Cons:

Difficult to map and provide complex data structures.

Extracts of Submission Data can also be configured via the Form Data Configuration.

A form's data configuration can contain one or more of the following methods of prefill data mappings in a form's XML data structure:

- Property Prefill Mapping - properties for the form version are used for sourcing your data to prefill your form.
- Input XML Prefill Mapping - An external XML file containing prefill data is obtained at the time that the form is rendered. This needs a prefill service to obtain the data values.
- Request Param Prefill Mapping - prefill data is passed through as HTTP request parameters, session attributes or cookies.

Before a form is rendered and presented to form users, the various prefill methods are applied in the following order:

1. Form XML data
2. Property prefill - client forms
3. Request parameters
4. Property - user profiles

As a form can contain multiple prefill methods, it is important to understand the way this is handled in Manager. Where multiple mappings to the same form field exist, prefill data obtained at later stages overrides earlier obtained prefill data.

The form prefill is a flexible tool that allows you to implement various customer requirements. You can even mask some mask certain fields, such as data of birth, phone number or address, on a form resume to improve application security.

Next, learn about form property prefill mapping.

Form Input XML Prefill Mapping

Journey Manager (JM) | Platform Developer | 18.11

Manager allows you to prefill a form by passing an **XML**[1] prefill document with form details. A form should have XML input mappings configured, so you can use this functionality in several ways:

- When XML input mappings are configured, the XML is returned from a Groovy Prefill Service.
- Assign tasks in Manager, when the **Form Data** field is set to **Input Xml Data**, so the mappings are used to map data into the form seed.
- Create anonymous saved forms using the **`SubmissionTaskService.createAnonymousSavedForm`** and **`SubmissionTaskService.createFormTask`**, when the **`inputXmlData`** parameter is used to pass in the input XML object.

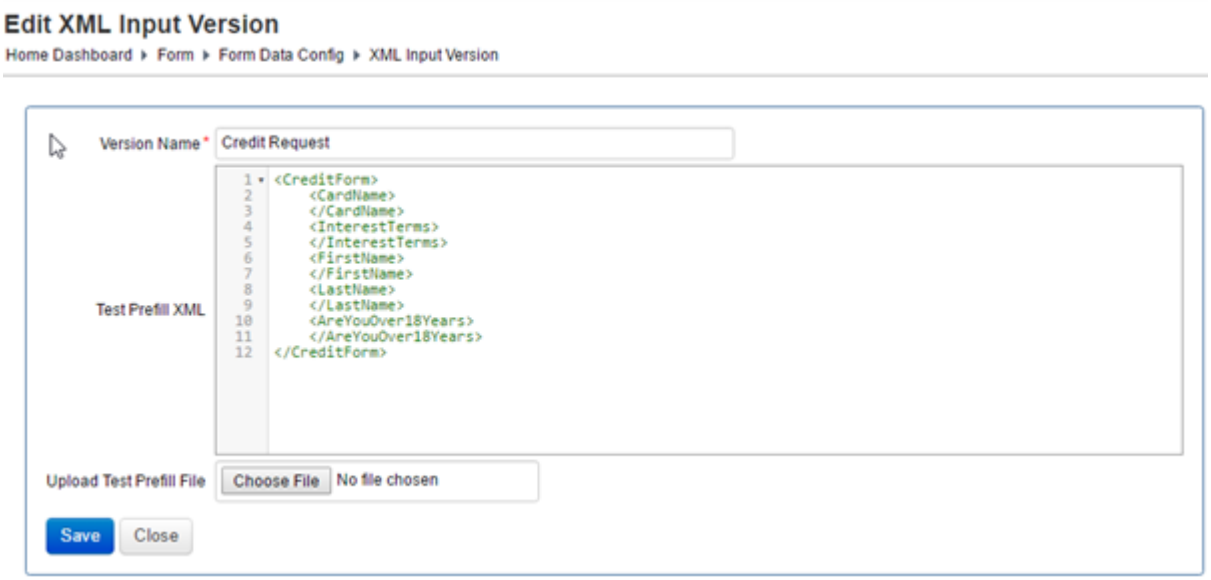
As an example, the XML prefill form passes all the details for the first page of the Credit Card Application form. This is handled by the form version prefill service which assembles the prefill details as an XML document.

> If you don't map properties, they won't be in a XML document that Manager sends to a form for the properties prefill.

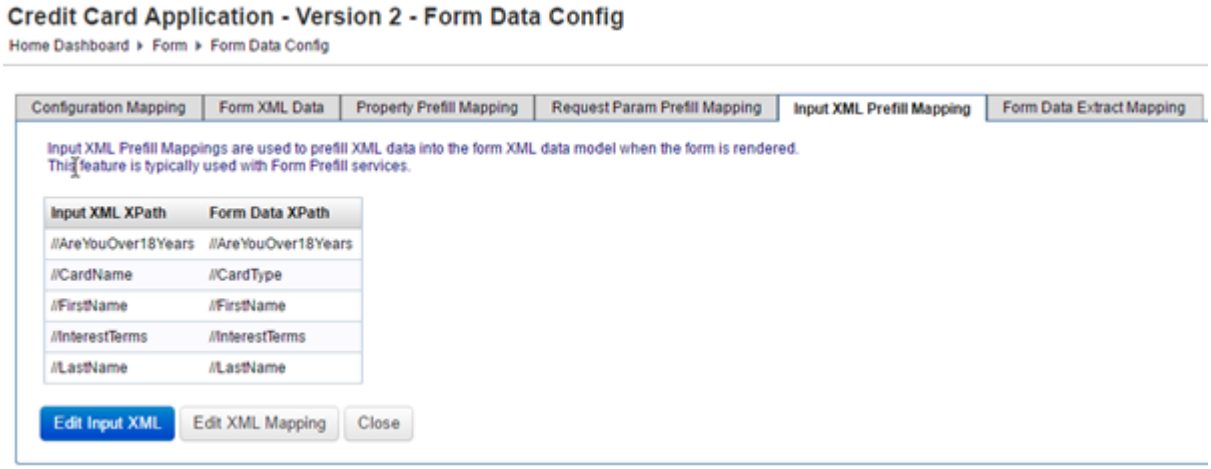
To configure a form input XML prefill mapping:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select a form version and click **Services** to configure the form version to invoke the prefill service.
4. Select a form prefill service from **Form Prefill Data Service** dropdown.
5. Click **Save** to update the changes.
6. Click the **Form Data Config** tab to configure the mapped fields.
7. Select the **Input XML Prefill Mapping** tab.
8. Click **Edit Input XML**.

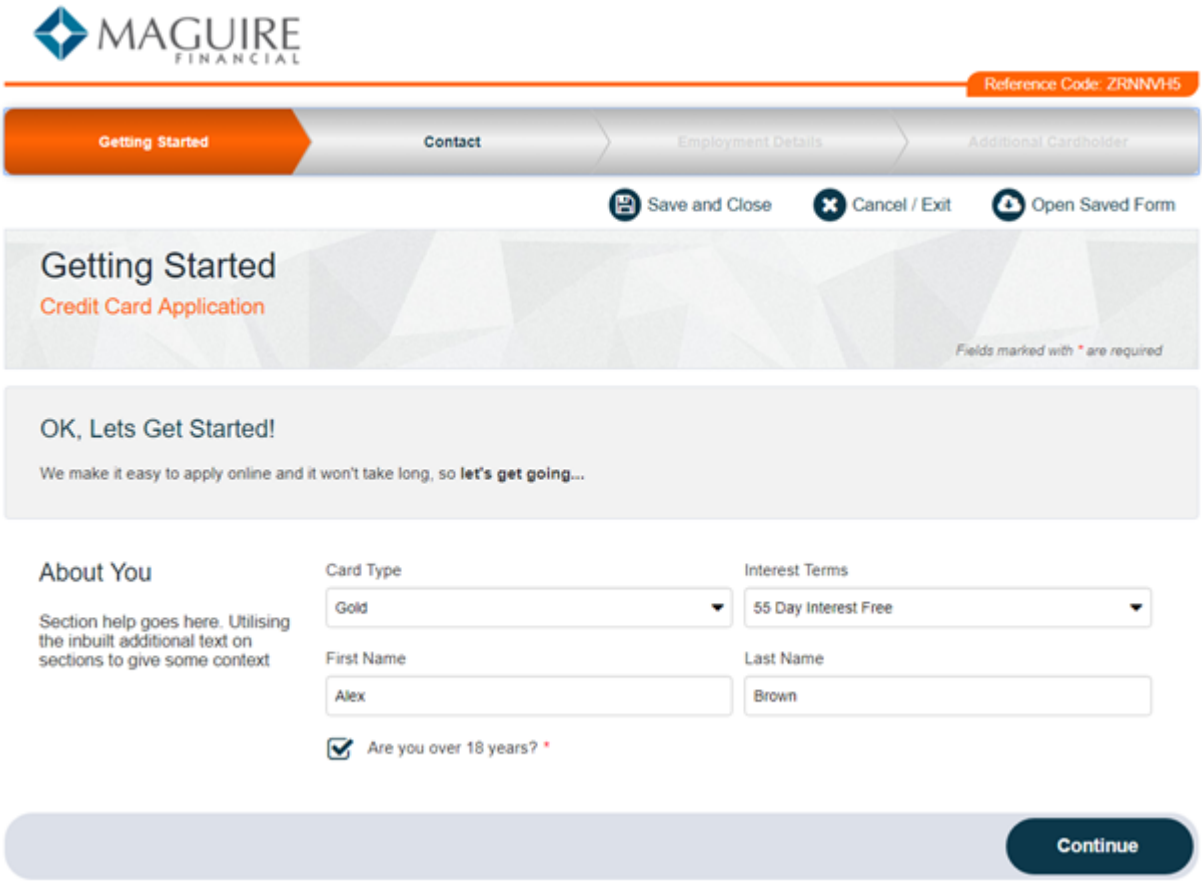
[1] XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.



9. Enter a name into **Version Name**.
10. Enter an XML form structure of the external fields that can be used for data prefilling. It must match the XML form that is generated by your **Form Prefill Data Service**. Alternatively, you can upload an XML file by clicking on **Choose File in Upload Test Prefill**.
11. Click **Save** to update the changes.
12. Click **Edit XML Mapping** to map each prefill data element from the source to the target. You have the following options to map elements depending on your Manager version:
    - Use drag and drop.
    - Use XPath configuration. | 18.11
13. Click **Save** to update the changes.
14. Check the mapped data as shown below



15. Check the rendered form that is prefilled with data provided by the Form Data Prefill Service.



It is also possible to use the prefill data service without mapping your fields. In this case, the exact same XML structure as the form XML must be returned from the prefill data service.

Next, learn about form data extract mapping.

Form Input XML Prefill Mapping Using Drag and Drop

Journey Manager (JM) | Platform Developer | Removed in 19.11.

As Adobe Flash is not supported after the end of 2020, this feature is not available in Journey Manager since version 19.11.

You should use Form Input XML Prefill Mapping Using XPath instead.

Manager allows you to map each prefill data element from the source to the target using drag and drop functionality. Manager uses **Adobe Flash**[1] to implement drag and drop.

To map prefill data elements:

1. Click **New Input XML** to create a new mapping or click **Edit XML Mapping** to change the existing one.
2. Review a source and a target of the property mappings in the table containing the following columns:
   - Source: Prefill Data XML.
   - Prefill Data XPath.
   - Form Data XPath.
   - Target: Form Data XML.
3. Drag and drop each data element from the **Source: Prefill Data XML** section to the **Prefill Data XPath** section.
4. Drag and drop each data element from the **Target: Form Data XML** to the **Form Data XPath**.



[1]Adobe Flash is a deprecated multimedia software platform used for production of animations, rich Internet applications, desktop applications, mobile applications, mobile games and embedded web browser video players. Flash displays text, vector graphics and raster graphics to provide animations, video games and applications. It allows streaming of audio and video, and can capture mouse, keyboard, microphone and camera input. Related development platform Adobe AIR continues to be supported.

5. Check that the XPaths are mapped correctly in the middle two sections. In the case of repeats, the parent of the repeatable object must be mapped to an identically named form data field.
6. Click **Save** to update the changes.

Next, go back to form input XML prefill mapping to finish the configuration.

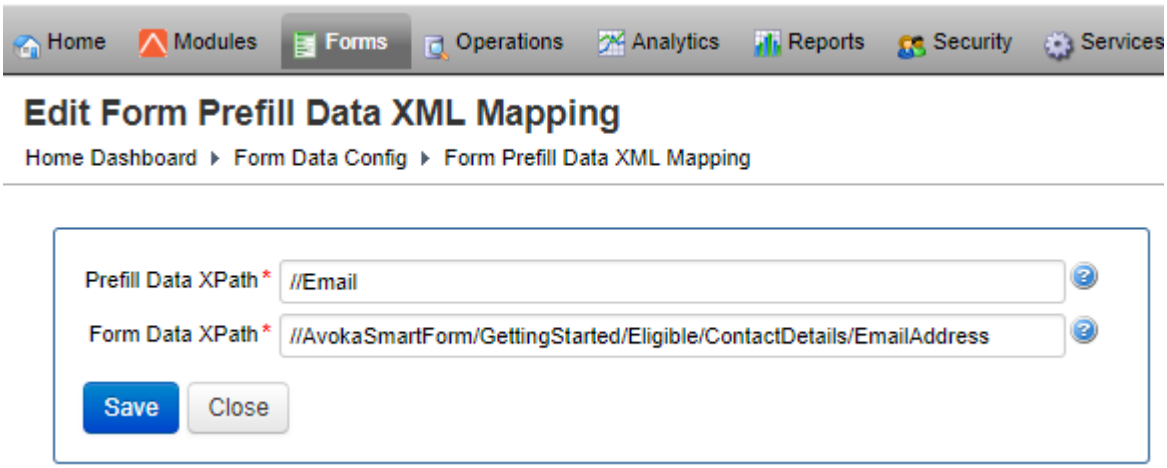
Form Input XML Prefill Mapping Using XPath

Journey Manager (JM) | Platform Developer | 18.11

Manager allows you to map each prefill data element from the source to the target using **XPaths**[1].

To map prefill data elements:

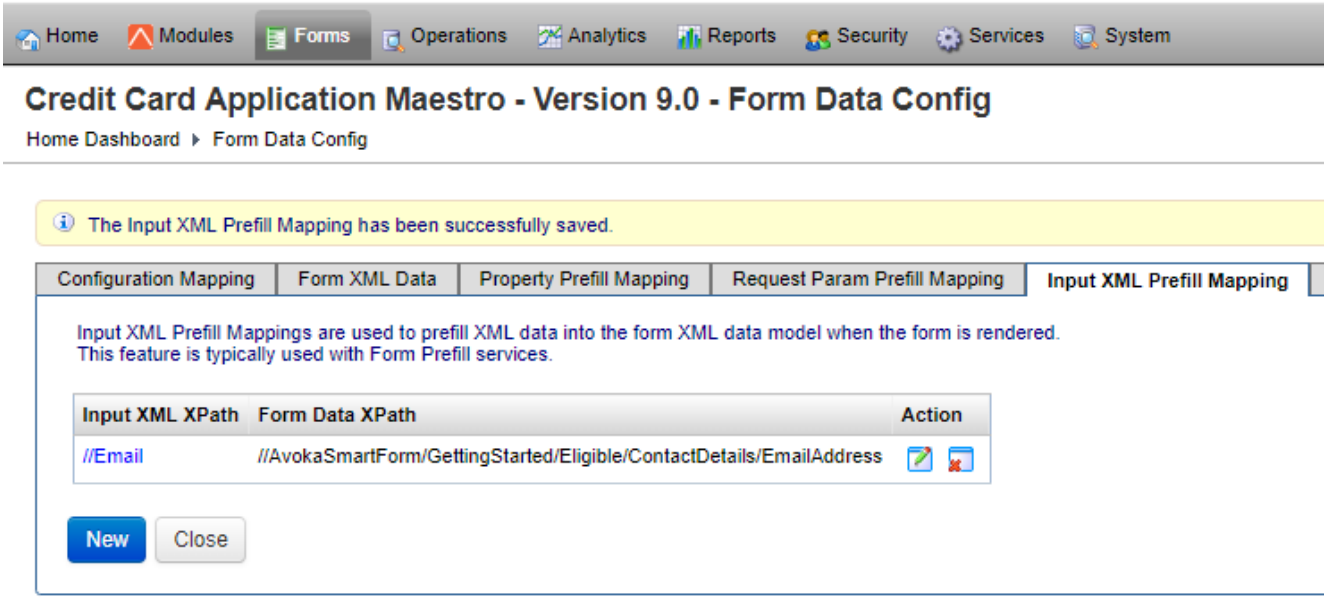
1. Click **New** to create a new mapping or click **Edit** to change the existing one.
2. Enter a valid sours XPath from the XML input version in the **Prefill Data XPath** field, for example, **`//Email`**.
3. Enter a valid target XPath from the form XML data in the **Form Data XPath** field, for example, **`//AvokaSmartForm/GettingStarted/Eligible/ContactDetails/EmailAddress`**.



4. Click **Save** to update the changes.

[1]XPath (XML Path Language) is a query language for selecting nodes from an XML document. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document. XPath was defined by the World Wide Web Consortium (W3C).

5. Check that the XPaths are mapped correctly.



Next, go back to form input XML prefill mapping to finish the configuration.

Form Property Prefill Mapping

Journey Manager (JM) | Platform Developer | 18.11

Manager allows you to map form properties to values within the form seed for prefill purposes. You can use this functionality in several ways. For example, when a user logs in to a form space and starts to fill in a form, user's contact details are already prefilled. This is achieved with the properties mechanism to prefill a form when it is rendered to a user. This mechanism relies on mapping a target form data field to the related user property using **XPath**[1] variables. These user properties are of special type so each value is automatically sourced from the logged in user's profile.

> If you don't map properties, they won't be in a XML document that Manager sends to a form for the properties prefill.

Form property can have one of the following scopes:

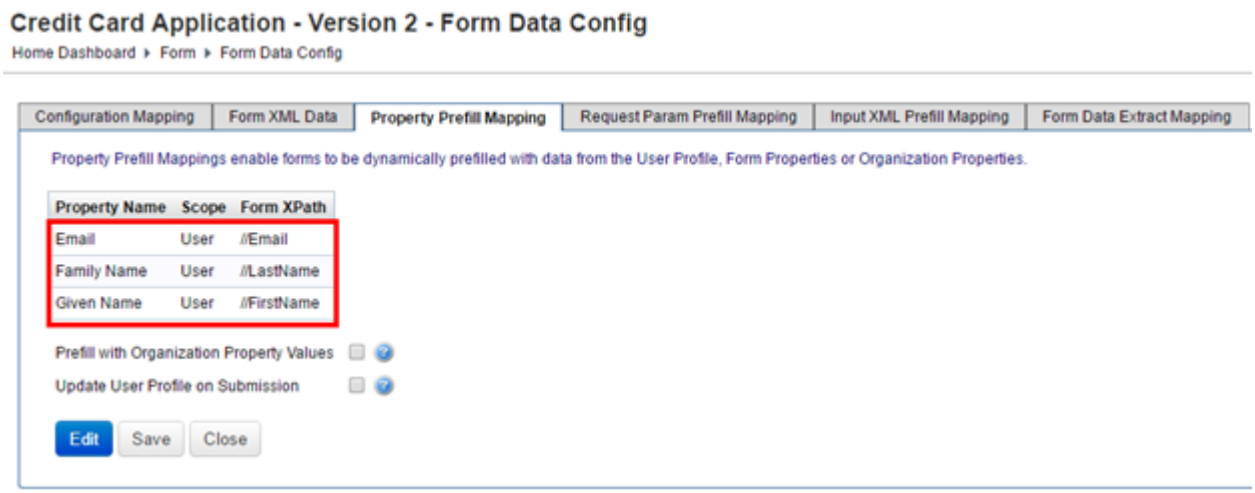
- **User** which obtains the value from the user's profile of the logged in user. These mappings will prefill the Form XML using the property value's stored in the logged in User's Profile. Once User Property Prefill mappings have been created for a Form Version, by setting the Form's System Profile tag UpdateUserProfile to true, on submission of a form, the User's User Profile properties will be updated to reflect any submitted changes to the User Property prefilled fields. This affects only user properties defined in the user profile, not the User Account's details directly. Further, a User may have multiple profiles with different values. Only the profile that is currently selected for the User will be used for prefill and saving. If no User Profile is currently selected, no user data will be prefilled or saved.
- **Form** which obtains its value directly or dynamically from a form version. These mappings will prefill the Form XML using the Form's Property value stored in the Form's Version Properties.
- **Client** which obtains its value directly or dynamically from an organization. These mappings will prefill the form XML using the client's property value stored in the form's version properties. These properties need to be sync'd to work correctly; Once the mappings are created, the Form Versions Properties screen provides a Sync Org Properties button which will import the required Properties based on the Client Property Fill Mappings.

To configure form property prefill mapping:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select a form version and click **Edit Form Data Config**.
4. Click the **Property Prefill Mapping** tab to view any existing mappings.

---

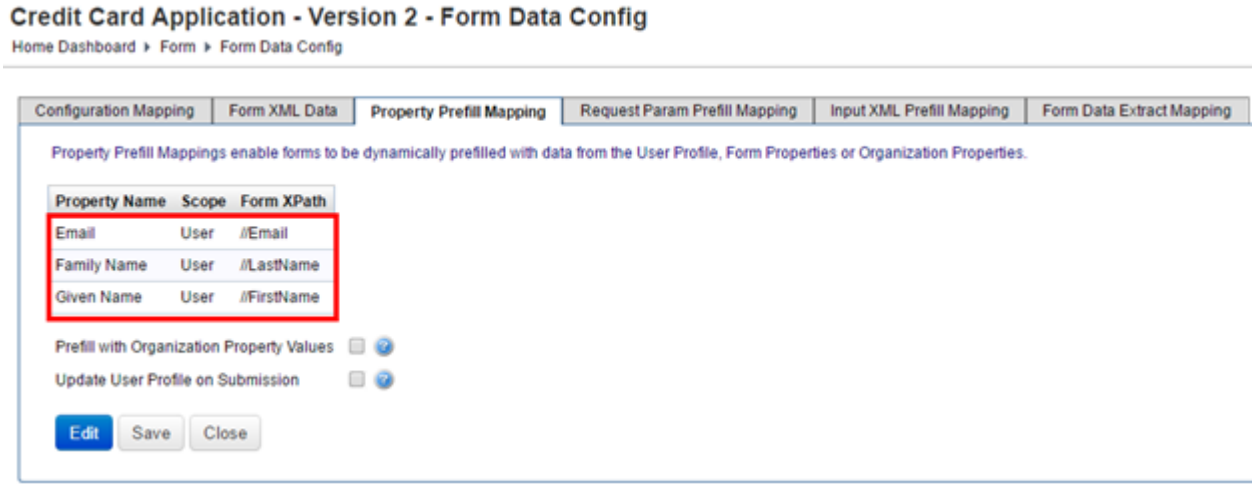
[1]XPath (XML Path Language) is a query language for selecting nodes from an XML document. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document. XPath was defined by the World Wide Web Consortium (W3C).



5. Click **Edit** to create or update the data mappings. You have the following options to map elements depending on your Manager version:

- Use drag and drop.
- Use XPath configuration. | 18.11

6. Click **Save** to update the changes.
7. Check the updated mappings. In this example, the scope of these properties is **User**, so the values are sourced from the logged in user's profile.

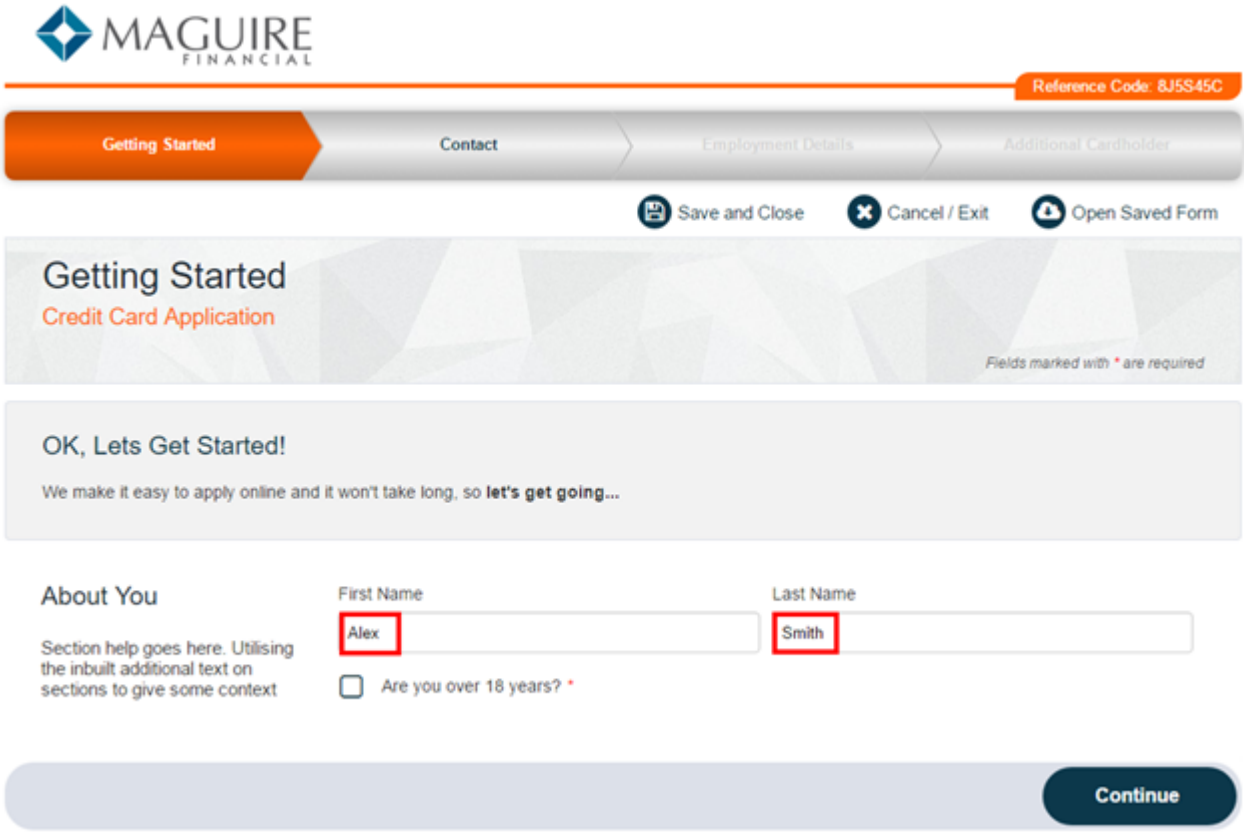


8. Select the **Prefill with Organization Property Values** checkbox to use organization (client scope) defined property values to prefill the form.

Organizations property values will be overridden by property values defined at the form version level.

This setting doesn't apply to user properties.

9. Select the **Update User Profile on Submission** checkbox to use the mapping for user properties at submission time to store data the user entered into the form back into the user profile.
10. Click **Save** to update the changes.
11. When a user logs onto a form and selects the first form, it is already prefilled with the user data such as First Name, Last Name, and Email from the user profile.



Watch the Manager property prefill video to learn how to configure and use property prefill of form scope.

Next, learn about form request parameter prefill mapping.

Form Property Prefill Mapping Using Drag and Drop

Journey Manager (JM) | Platform Developer | 5.1 & Higher

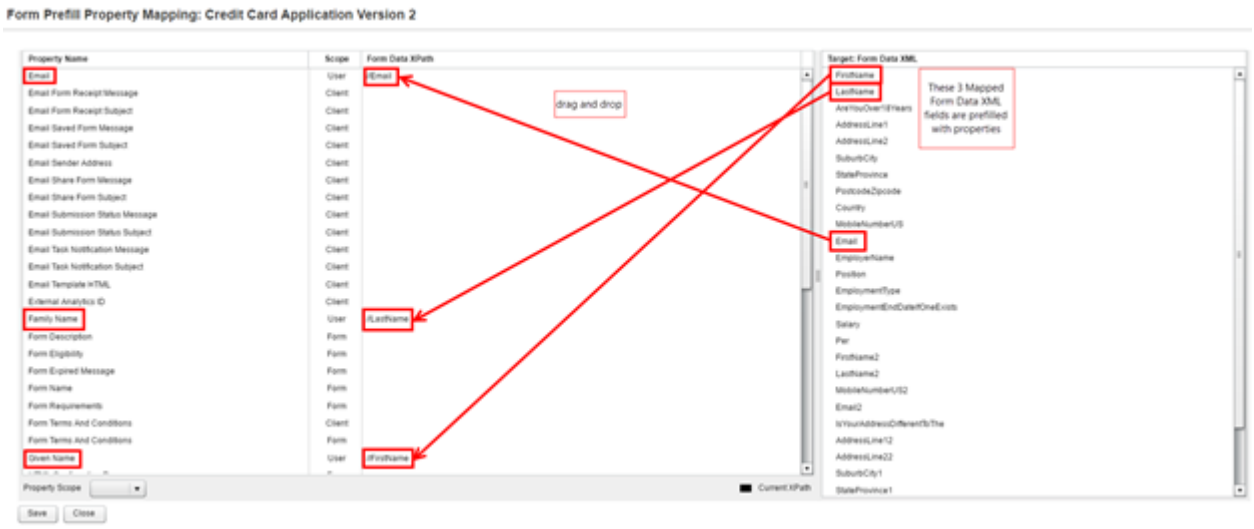
Manager allows you to map form properties to user contact details to prefill a form using drag and drop functionality. Manager uses **Adobe Flash**[1] to implement drag and drop.

> Adobe Flash will be phased out by the end of 2020.

To map prefill data elements:

1. Select a property scope from the **Property Scope** dropdown list that can be:
   - **Client** for properties that should be set on an organization level.
   - **Form** for properties where values are specific to each form version.
   - **User** for properties sourced from the user profile.
2. Review the mappings in the property mapping table containing the following columns:
   - Property Name.
   - Scope of the property.
   - Form Data XPath for the XPath of the target data element in the form XML, which is used for prefilling the corresponding field on the form.
   - Target: Form Data XML for the target data element in the form XML.
3. Drag and drop the field from the **Target: Form Data XML** column to **Form Data XPath** column against the related field in the **Property Name** column as shown below:

[1] Adobe Flash is a deprecated multimedia software platform used for production of animations, rich Internet applications, desktop applications, mobile applications, mobile games and embedded web browser video players. Flash displays text, vector graphics and raster graphics to provide animations, video games and applications. It allows streaming of audio and video, and can capture mouse, keyboard, microphone and camera input. Related development platform Adobe AIR continues to be supported.



4. Check that the XPaths are mapped correctly in the middle section.
5. Click **Save** to update the changes.

Next, go back to form property prefill mapping to finish the configuration.

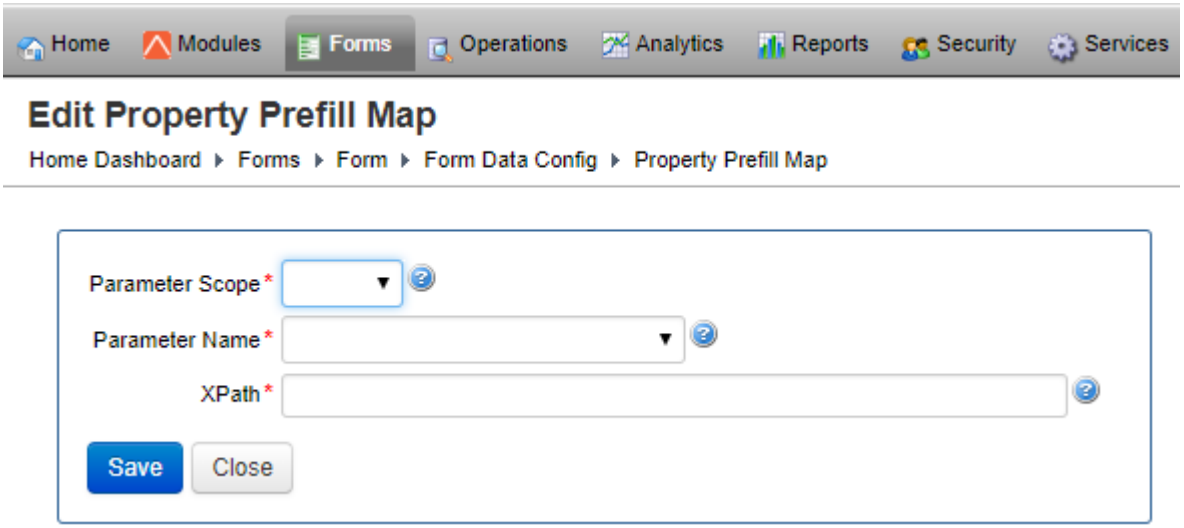
Form Property Prefill Mapping Using XPath

Journey Manager (JM) | Platform Developer | 18.11

Manager allows you to map form properties to values within the form seed for prefill purposes using **XPaths**[1].

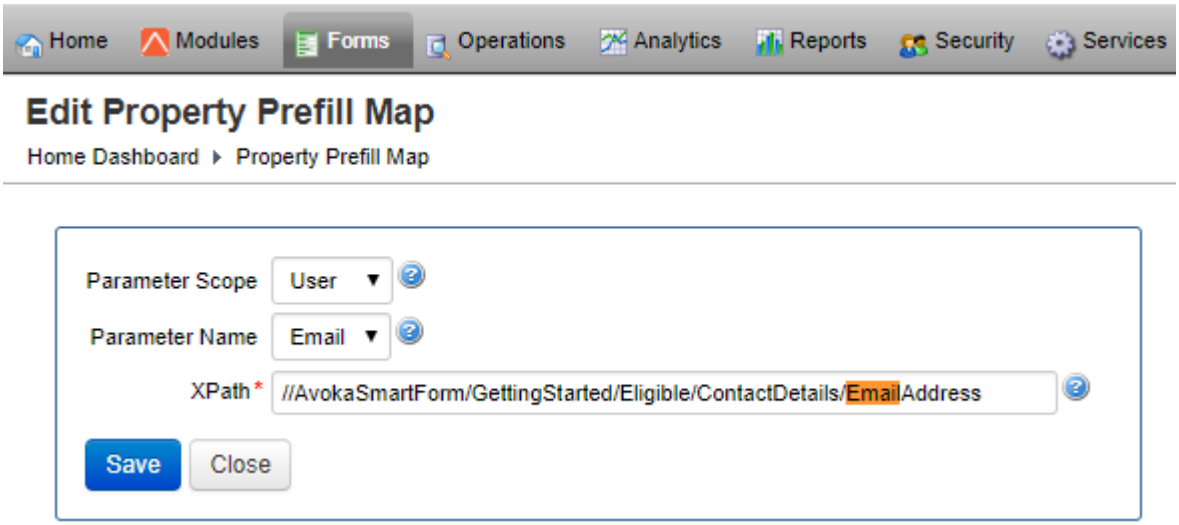
To map prefill data elements:

1. Click **New** to create a new mapping.



2. Select a property scope from the **Parameter Scope** dropdown list that can be:
   - **Client** for properties that should be set on an organization level.
   - **Form** for properties where values are specific to each form version.
   - **User** for properties sourced from the user profile.
3. Select a parameter name from the **Parameter Name** dropdown list, for example Email.
4. Enter a XPath to the valid XML element, for example **`//A-vokaSmartForm/GettingStarted/Eligible/ContactDetails/EmailAddress`**.

[1]XPath (XML Path Language) is a query language for selecting nodes from an XML document. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document. XPath was defined by the World Wide Web Consortium (W3C).

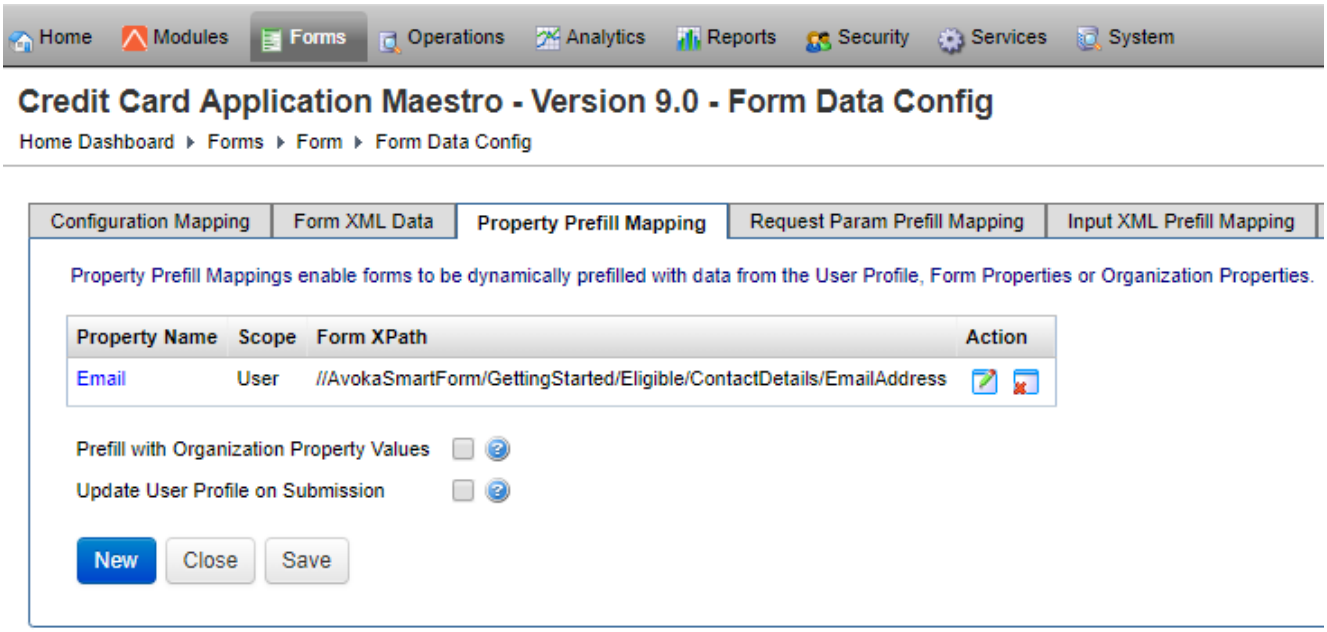


You can check the form XML structure in the Form XML Data tab. In our example, the .XML structure of interest is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<AvokaSmartForm>
       ...
       <GettingStarted>
       ...
               <Eligible>
               ...
                       <ContactDetails>
                              <EmailAddress>
                              </EmailAddress>
                              ...
                       </ContactDetails>
                       ....
</AvokaSmartForm>
```

If the value is invalid, the error message is displayed: **No XML element found for XPath: //E-mailAddress**.

5. Click **Save** to update the changes.
6. Check that the XPaths are mapped correctly.



7. Click **New** to create another mapping or click **Close** to return to the form configuration.

Next, go back to form property prefill mapping to finish the configuration.

Form Request Parameter Prefill Mapping

Journey Manager (JM) | Platform Developer | 5.1 & Higher

As another means of form prefill, data may be obtained from an **HTTP**[1] request and used to prefill the form when it is rendered. Prefill data can be passed in via **JSON**[2] Prefill Data, request URL parameters, session values or cookie values.

When the form is rendered, Manager searches for each request parameter prefill mapping to find a value. When a match is found, the value of the entity is used as the prefill data. However, if multiple matches are found, the later entry takes precedence, for example request parameters over session attributes:

1. Set of HTTP session attributes
2. Cookies
3. Request parameters
4. Special request parameter, such as `jsonPrefillData`.

> It is not recommended to pass any **PII**[3] details, for example, names or contact details, as parameters.

As an example, you can pass the selected card type and interest terms as request parameters to prefill a Credit Card Application form. If a customer has already chosen the specific credit card from your website, then there is no need for them to select the credit card again when filling in the Credit Card Application form.

The Credit Card Application form must include two dropdown menus that are used to indicate the user's chosen credit card. You can add them in Manager:

1. Add Card Type dropdown with options such as Student, Low Fees, Gold, and Platinum.
2. Add Interest Terms dropdown with options such as 55 Day Interest-Free and Low Rate.
3. Build your form.

To configure Request Param Prefill Mappings:

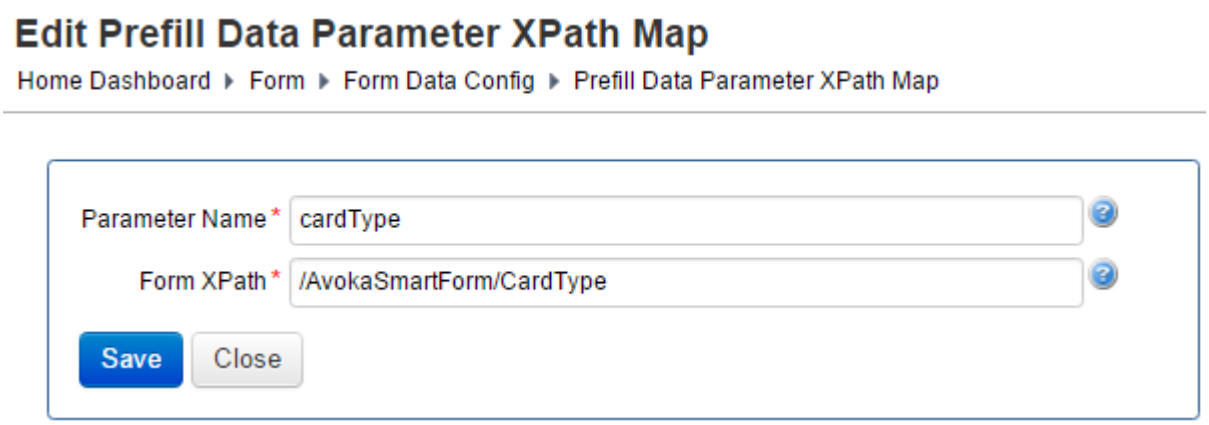
---

[1]HTTP (HyperText Transfer Protocol) is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

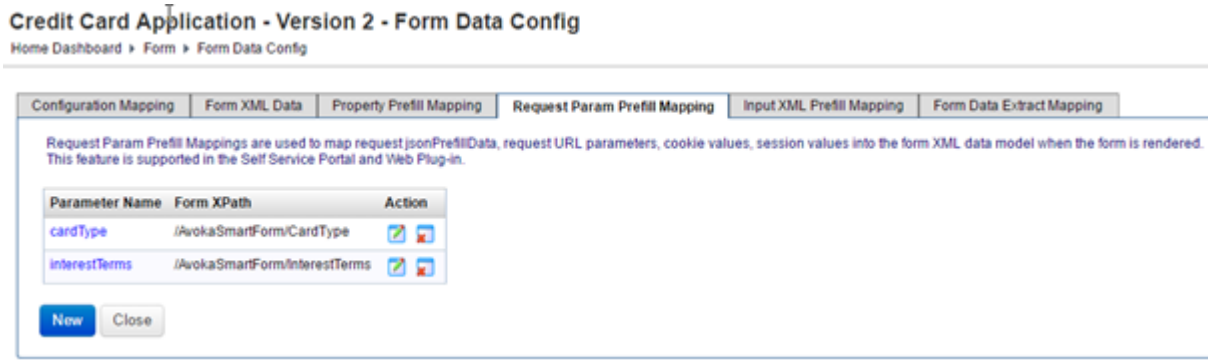
[2]JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

[3]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select a form version and click **Edit Form Data Config**.
4. Click the **Request Param Prefill Mapping** tab.
5. Click **New**.
6. Enter each mapping as shown below:

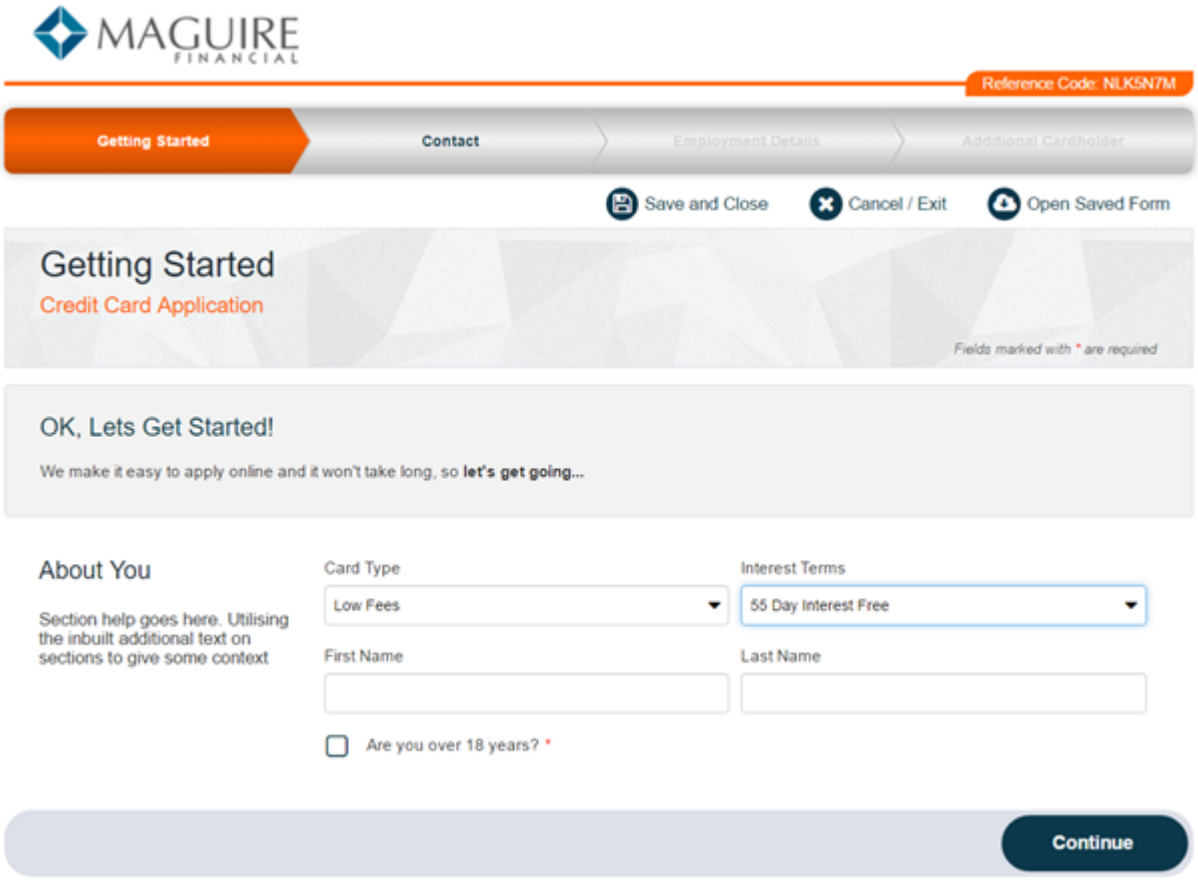


7. Click **Save** to update the changes.
8. Check the newly defined mappings.



To test these mappings, render your Credit Card Application form and try adding to the URL both parameters (card Type and interest Terms) with each value matching one of the related dropdown options, as defined in Maestro.

For instance, this shows that card type of Low Fees and interest terms of 55 Day Interest-Freewere passed as URL parameters to prefill the Credit Card Application form:



GET vs POST for Request Parameter Prefill Mapping

When using the Request Parameter Prefill Mapping feature, it's important to choose the appropriate HTTP method for your scenario. Manager supports both GET and POST requests with this feature. Each of these methods has its own unique behaviors. Many of these behaviors can be considered a pro or a con, depending on the situation.

GET Requests

- Easily cacheable
- Typically stored in browser history
- Can be stored in bookmarks
- Can be shared between customers
- Browser refresh function works without issue (although entered data will be lost, the same valid parameters can still be used to pre-fill the form)
- May store parameters in logging of network equipment
- Parameters visible in browser address bar
- Limited by maximum URI length

POST Requests

- Not limited by maximum URI length (can send more data)
- Parameters unlikely to be cached
- Parameters won't be stored in browser history
- Parameters won't be shared if the link is copied and shared/posted elsewhere
- Parameters won't be visible in browser address bar

Now that these differences are understood, let's consider a couple of scenarios where each of the methods would perform best.

Prefilling Product Information

In this scenario various pieces of product information need to be prefilled in order for the application form to know what product/sub-product the customer is applying for. Example parameters:

Parameter Name	Description
promoCode	A 10 character promotional code, which will update the form to display promotional rate/offers assuming the promotion is still active. (Optional)
productId	The 8 character numeric ID of the product being applied for, such as a particular type of loan or credit card. This will update the graphics and product wording in the form.

In this scenario, the parameters being prefilled are publicly available information. We're quite happy for our customers to share these links with their friends/family or on other sites such as social media. If the promo ID is invalid, appropriate messaging with be presented to the customer, so again it's OK for this information to be shared/cached/bookmarked/etc.

In this situation the best option is a GET request, which will ensure the required product ID parameter isn't lost if the link is reused.

Prefilling Customer Information

In this scenario a customer ID is being used to reduce the number of fields presented to the customer. Example parameters:

Parameter Name	Description
customerId	The ID our customer uses to login to online banking. This will be used to remove fields/pages from the form when that information is already known about the customer.

In this scenario, the parameter being prefilled is more sensitive in nature. Although there is no data harvesting risk, since we're removing sections rather than pre-filling personal data, this is still something we'd rather not have stored in browser history or shared accidentally.

In this situation the best option is a POST request, which will help ensure the parameter isn't stored in history, shared or visible in the browser address bar.

Next, learn about form input XML prefill mapping.

Add Repeating Content from XML Prefill Mapping Dynamically

Maestro | Form Builder | 5.1 & Higher

When you configure Input XML Prefill Mapping in Manager to prefill a form, you need to take into account how repeats will factor into the prefill XML. This is especially true when you want to provide inspectors with case history on a coversheet, or if field workers are required to check off items which are driven by ERP. Although there are other workarounds, this technique is quite elegant, as it minimizes coding, and allows for optimized dialog between form developers and integration teams. In this example, we define an array of JSON objects in a single XML value to store all the prefill data in the repeat. An array of JSON objects is the standard used by Dynamic Data. Therefore, there's some re-usability in response formats. Also, it can easily be updated with test data with a single button (good for demos or UI testing).

You need the following widgets on your form:

- A data element to store the JSON object (in this example, called JSONactionsData)
- A Repeating Block Template, customized to include whichever content you like (in this example, three fields). Ensure initial value is set to 0.
- A business rule, to set the contents of the repeat using the data element.
- A standard button, to set the contents of the data element (In this example, called "Simulate Lookup")
- Optionally, a field called "debug", to see the contents of the data element above.

You have to create the following business rule:

```
if (!sfc.isBlank({JSONactionsData}) && ({RepeatableContent}.getInstanceCount()==0)){
    //assume we only ever run once to avoid duplicates on save & resume ------^
    var obj = JSON.parse(unescape({JSONactionsData}));
    for (var i=0;i<obj.length;i++) {
        var newInstance = {RepeatableContent}.addInstance();
        sfc.setRawValue(sfc.convertToNode(sfc.normalizeUid(newInstance, "./RepeatIn-
ner/Item/_outerArea/_contentArea/RepeatingFields/_layoutAtAction/At")),obj[i].at);
        sfc.setRawValue(sfc.convertToNode(sfc.normalizeUid(newInstance, "./RepeatIn-
ner/Item/_outerArea/_contentArea/RepeatingFields/_layoutAtAction/Action")),obj[i].ac-
tion);
        sfc.setRawValue(sfc.convertToNode(sfc.normalizeUid(newInstance,
"./RepeatInner/Item/_outerArea/_contentArea/RepeatingFields/_lay-
outAtAction/readOnly")),"true");
    }
}
```

Where:

- **{JSONactionsData}** is the data element
- **{RepeatableContent}** is the Instance Manager of the repeat
- **_layoutAtAction** is the root custom field in the Repeating Template, in this case a block with three fields.

Then, you need to create the following button (Simulate Lookup):

```
var obj = [ {"at":"AAA", "action":"Check SCADA"},
            {"at":"BBB", "action":"Check 10 Open"},
            {"at":"BBB", "action":"Check 11 Open"},
            {"at":"AAA", "action":"Close BBB 30"}, ];
sfc.setRawValue( {JSONactionsData} , escape(JSON.stringify(obj)));
```

Where the only dependency is a node {JSONactionsData}.

Finally, you can use the following debug calculation:

If you want to see the data which we are storing in the data element, you will need to unescape it before displaying it.

```
if (!$ISBLANK({JSONactionsData})){
    return unescape({JSONactionsData});
} else return undefined;
```

This button sets the values. It makes a great demo, but it can also be extended to use data from a lookup or other type of prefill.

Next, learn more about form data XML.

Form Prefill for Anonymous Users

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager offers a wide range of options to support form prefill or pre-population including anonymous users.

There are several options for different use cases, which are safe to use for **Personally Identifying Information**[1] data meaning they are only to be used to pass non PII data, such as a product choice on a website.

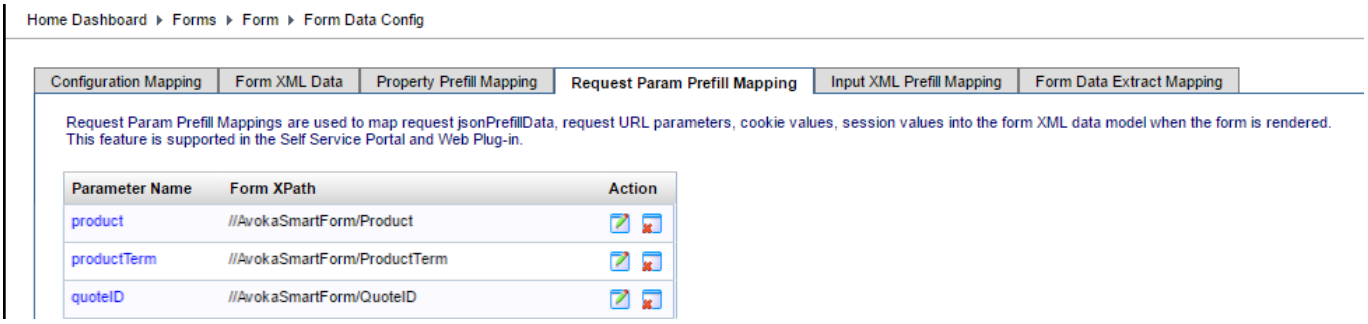
It should be mentioned that a term Anonymous is actually a bit misleading. The user might be authenticated in another system, they are just not a registered user in Manager.

You can solve this problem There are several options when tackling this problem.

- URL Parameter passing on a GET request
- HTTP request headers on a POST request
- Cookies
- Short life tokens
- Encrypted string

URL Parameter passing on a GET request

It doesn't get much simpler than this. All you need to do is add some form prefill mappings using the same name as the URL parameter you use as shown below.



Would work with a url such as https://tm.demo.avoka.com/test/servlet/SmartForm.html?formCode=cgf-apply-ga&-product=single&productTerm=Fixed&quoteID=GTM495780

Pros

- Very simple
- Out of the box

---

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

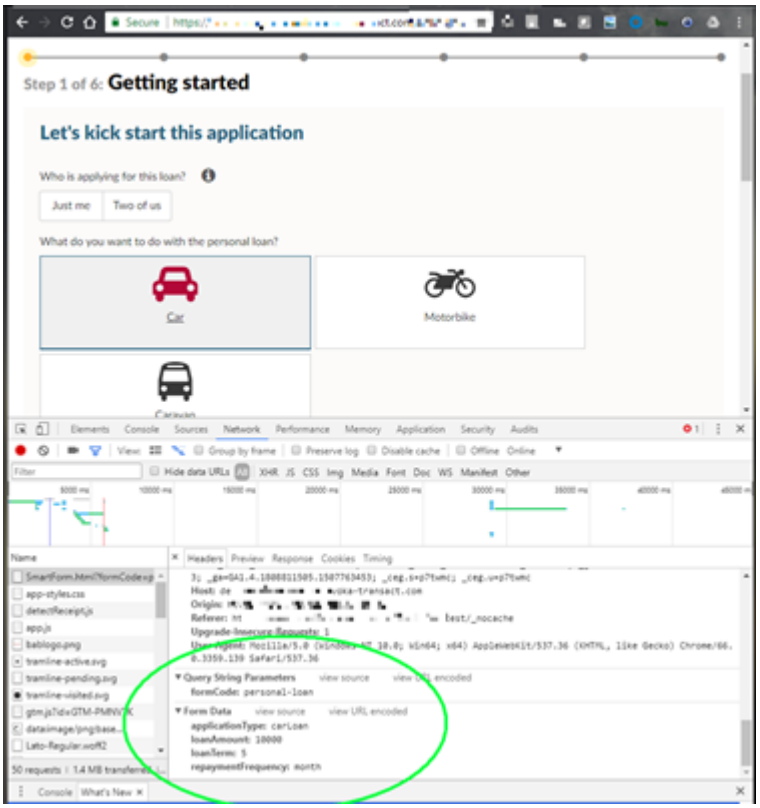
Cons

- It should only be used for non sensitive or PI data i.e. you should not be passing data such as the customer name

HTTP request headers on a POST request

This is pretty much the same mechanism as the above method but the data name/value pairs our provided as header attributes on the POST request. This implementation is very common in our solutions.

The example is shown below



The Manager config is exactly the same.

Pros

- Simple
- Out of the box
- Slightly more sophisticated than URL parameters

Cons

- It should only be used for non sensitive or PI data i.e. you should not be passing data such as the customer name

Encrypted Cookies

This is not an out of the box option and requires the creation of a custom form component and two TM groovy services.

The service requires three components:

- Cookie Manager (Composer/Maestro widget) - added to the form
- XXXCLIENT Cookie Decryption (Transact Dynamic Data service) - called by the Cookie Manager widget
- XXXCLIENT Cookie Encryption (Transact Submission Completed Processor service) - associated with the form

How it operates:

- XXXCLIENT Cookie Encryption: extracts form values from the "XXXCLIENTData" node and populates an encrypted cookie named "XXXCLIENTData".
- Cookie Manager: holds data fields populated from the encrypted "XXXCLIENTData" cookie. Also hold a business rule that invokes the "XXXCLIENT Cookie Decryption" dynamic data service to decrypt the contents of the cookie. The business rule is only invoked once during the form's initialization phase.
- XXXCLIENT Cookie Decryption: Dynamic data service invoked via form business rule within the Cookie Manager widget.

On submission the data from the form is encrypted and made available to any subsequent form in the solution that the user might access.

Pros

- Fairly secure

Cons

- Relies on the next form being opened on the same device

Short life tokens (SLT)

There are often several client systems from which secure Personally identifiable information (PII) can be sourced. For this reason, Avoka can provide a simple reusable prefill implementation that can be consumed by several “client system” sources e.g.

- Online Banking
- Customer portals
- CRM

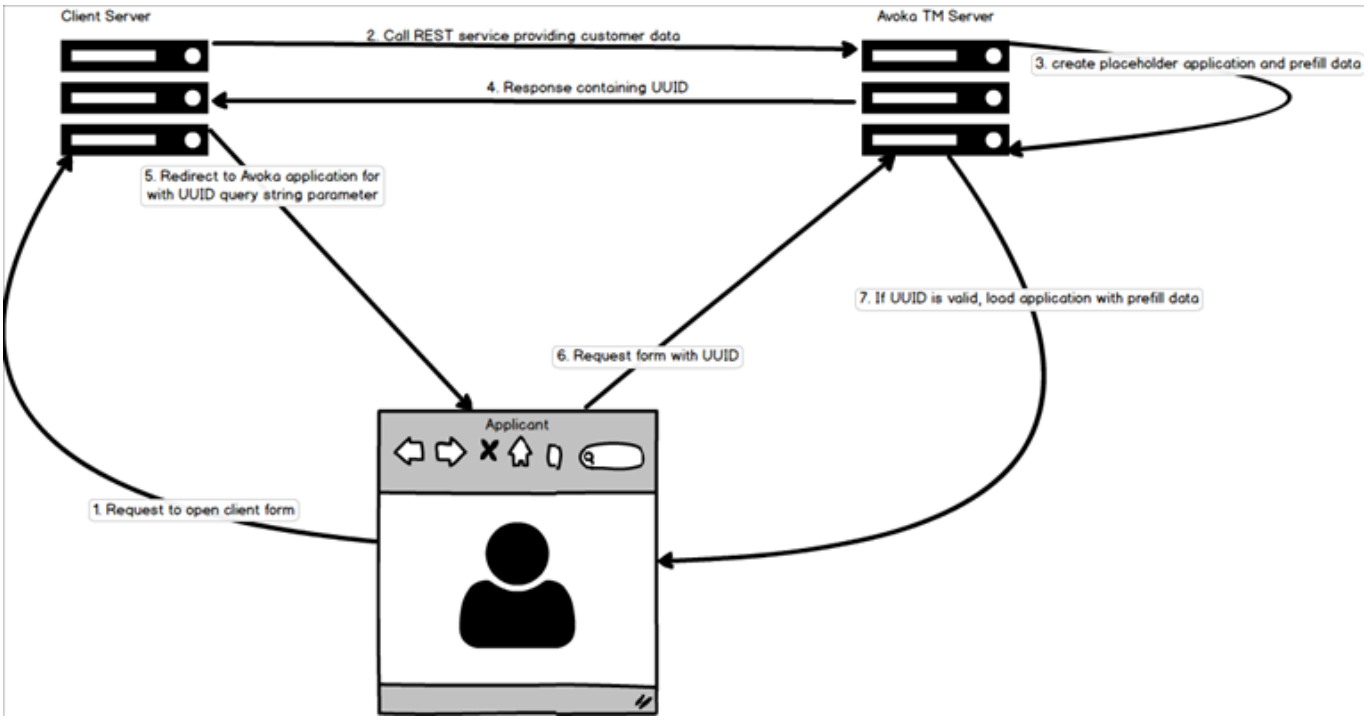
Avoka will expose a RESTful service to enable the “client system” to push customer data to Avoka when a customer requests the form from within the “client system” application. Avoka will pass back a unique identifier (a UUID) that will enable the client system to build a URL request that will open the Avoka form with the prefilled data.

For pre TM 5.0 implementations a "container" submission will be used to store the data. Since TM 5.0, storing the temp prefill data in MemCache for the very short time is possible and is a neater design than creating a temporary submission object to hold the prefill data for a short time frame.

The token will be short lived, if the UUID (submitkey) passed is for a container submission older than 30 seconds (configurable) Avoka will load an empty form. The container will be removed.

If the form prefill is successful, the form will load, and the container will be removed.

A tidy up task will be scheduled to remove any containers older than 1 minute.



If the RESTful customer data service (2) is unsuccessful the UUID will not be returned.

If the form prefill service in Avoka fails for one of the reasons below, Avoka will load an empty form.

For the prefill service call (2), IP whitelisting and basic auth. will be used to secure access.

Several URL parameters will need to be passed with the Request (5):

Example prefill data provided could be:

- Salutation
- First Name
- Last name
- Date of birth
- Address
- Mobile
- Email

Pros

- secure option
- single prefill service required in Avoka
- securely pass data from multiple sources

Cons

- does require some work by the client

Encrypted String

This is an enhancement on the prefill mapping GET or POST request, rather than sending name value pairs in clear text you encrypt all your prefill data into a single string. This is not a common implementation.

```
Generally the outline is as follows:
 # A public/private key pair is generated for the TM instance. The encryption (public)
key is shared with the external calling site/portal.
 # The external (referring) website/portal generates the secure token and adds this to
the HTTP request, either in the payload data or as a request cookie.
 # The TM prefill service parses the token from the request, decrypts and prefills the
form per BAU.

The encryption / decryption code is largely boiler-plate Java code and easily trans-
portable, e.g.:
/**
 * Encrypts or decrypts the specified byte value
 * Presumes that the decryption value has been Base64 decoded already, e.g.:
 * def URLCodec urlcodec = new URLCodec()
 * def byte[] value = urlcodec.decode(aString).decodeBase64()
 * Presumes that the encrypting value is in UTF-8 encoding, e.g.:
 * byte[] value = aString.getBytes("UTF-8") *
 * @param value
 * @param encrypt
 * @param keystore
 * @param keystorePass
 * @param keyAlias
 * @param aliasPass
 * @return byte[]
 */
private byte[] encryptDecrypt(byte[] value, boolean encrypt, byte[] keystore, String key-
storePass, String keyAlias, String aliasPass) {
 // access the keystore and load the decryption key
 KeyStore keyStore = KeyStore.getInstance(KeyStore.defaultType)
 ByteArrayInputStream bais = new ByteArrayInputStream(keystore)
 keyStore.load(bais, keystorePass.toCharArray())
 Key key = keyStore.getKey(keyAlias, aliasPass.toCharArray())
 int keyBuffer = key.getModulus().bitLength() / Byte.SIZE
 Certificate certificate = keyStore.getCertificate(keyAlias)
 Cipher cipher = Cipher.getInstance(key.algorithm)
 // encryption
 if(encrypt) {
 cipher.init(Cipher.ENCRYPT_MODE, certificate.publicKey)
 ArrayList encryptedArray = []
 value.toList().collate(keyBuffer-11).each() { bytes ->
 def encryptedData = cipher.doFinal((byte[])bytes)
 encryptedArray.addAll(encryptedData)
 }
 return (encryptedArray as byte[])
```

```
  }
  // decryption
  else {
  cipher.init(Cipher.DECRYPT_MODE, key)
  StringBuffer decryptedStr = "" << ""
  value.toList().collate(keyBuffer).each() { bytes ->
  def decryptedData = cipher.doFinal((byte[])bytes)
  decryptedStr << new String(decryptedData)
  }
  return decryptedStr.toString().getBytes("UTF-8")
  }
 }
```

Pros

- Fairly secure
- Prefill data is encrypted.
- It can be a little more flexible to a roll-you-own solution without resorting to more structured models (like SAML).

Cons

- For larger payloads, encrypted data will need to be sent as a POST request.
- Encryption keys need to be maintained (on both sides).
- Prefill data is encrypted. Debugging and operational support becomes more difficult.

Next, learn about form property prefill mapping.

Launch a Form From Another Form

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to launch one form from another one using prepop.

Sometimes you may have an existing database of information that you want to use as a basis for pre-populating a form. This data is often known as Context Data.

For example, you may have an existing customer database (or list of properties or contracts or medical records or accounts or property locations or whatever), and you want to be able to:

- search for a customer by various criteria
- view summary information for all matching customers
- click a button to open a form on behalf of that customer (there could be multiple forms to choose from)
- have that form open in new window, and be pre-populated with the customer information.
- The form submits in the usual way.

Technique

1. Search form

Build a search form. This will use a Dynamic data lookup to populate a table or a repeat with information from the database. (This tech note does not cover how to build this search form using dynamic data, that is covered elsewhere.)

There are two alternatives.

1. If there is just a small amount of data, pass all the data in request parameters to the new form, and use pre-fill mappings to map the request parameters into form field data.
2. If there is a larger amount of data, pass a unique identifier in a request parameter, and use a form pre-fill service to populate the form.

This article will look at the second technique which is slightly more complicated - the first technique is very similar

In the result row, including a button with the following script:

```
var param = {MyUniqueIdField}; // This parameter is the unique id that comes from the
repeating section
var formUrl = "https://t-
m.demo.avoka.com/finance/servlet/SmartForm.html?formCode=myformcode";
// Append a parameter to the end of the URL
window.open(formUrl + '&myparam=' + param,'_blank');
```

This script will launch the form named "myformcode" with an extra parameter.

Note that the Search form does not have a submit button - it is used only to retrieve data, and launch

Main form

The Main form will use a pre-pop script in order to fetch prefill data. There are several different ways to do the actual prepop, but the key issue is getting hold of the URL parameter that was passed from the Search form.

If you create a new service of type Groovy pre-fill script, it accepts several parameters, including:

```
formRequestParams : Map<String, String>
```

In order to retrieve our parameter, we would use something like this:

```
var myParam = formRequestParams['myparam'];
```

Thereafter, use this parameter to retrieve data from the backend system, and generate the XML pre-pop data.

Please see the following page for more information, and also refer to the Transaction Manager documentation.

Alternatives

- Another way to do this is to directly add a similar button into the native application that you use for your customers. However, sometimes this is not possible to do, and building the Search form within Transact may be useful.
- This sample hard-codes the location of the server ('formUrl" in the JavaScript code above). You can build up the URL dynamically by using some of the values injected into the form by Transaction Manager - you will need to use the DynamicDataServiceURL and strip out some of it in order to find the base URL for forms. This has not been included in this sample to keep it simple.

Restrictions

This technique cannot be used in the Transact Field App.

Next, learn about form property prefill mapping.

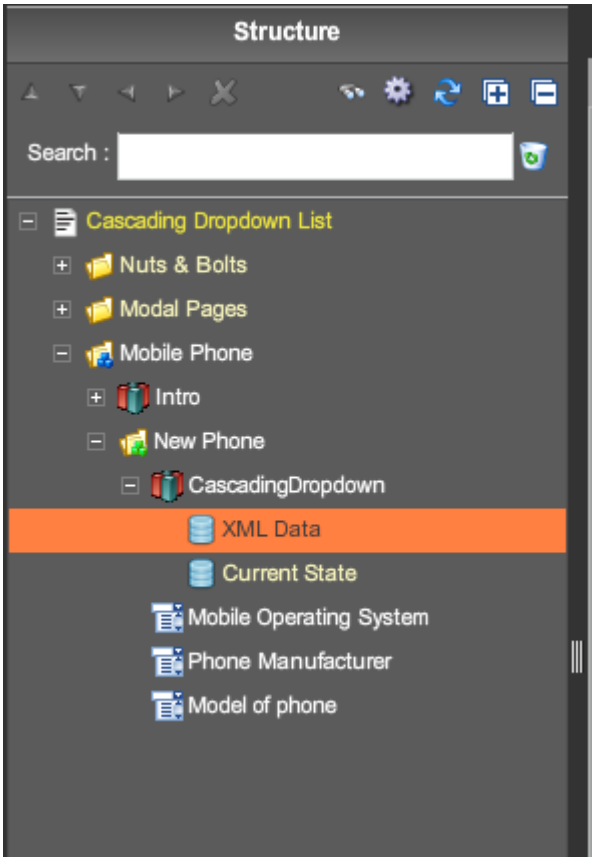
Prefill a Cascading Dropdown in Composer

Journey Manager (JM) | Platform Developer | 5.1 & Higher

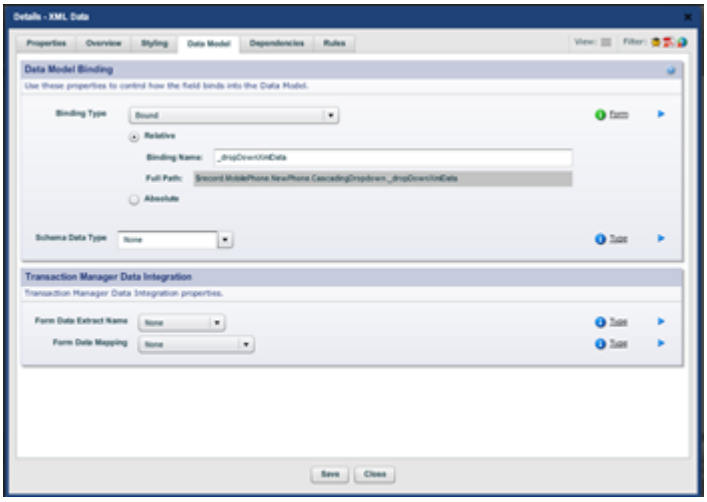
Manager allows you to update XML data that defines a cascading dropdown in a Composer form from the form properties defined within Manager.

To prefill a cascading dropdown:

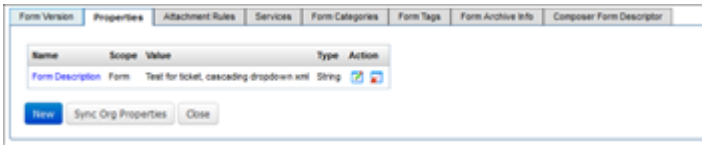
1. Open the structure panel in Composer and navigate to the XML data element to adjust the setting for the XML data model bindings in the form before publishing it.



2. Open the properties for the xml data. Navigate to the data model tab and change the binding type to Bound. Note the path as you may need it later.



3. Publish the form to Manager.
4. Navigate to your form in Transaction Manager and click on properties in the Form Versions block for the form you are making these changes to. Here we are going to add a new form property with some modified XML that will be used to update the form in Manager.

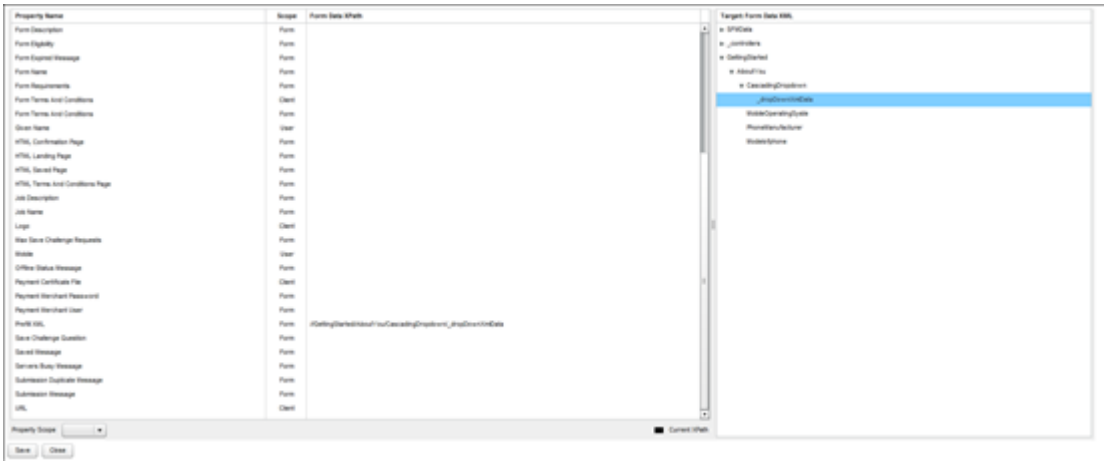


5. Click new and fill in the **Name** and set the data type to String. A new field will become visible; this is where you add the modified xml that will be used to update the form. For this example, another manufacturer was added to the Xml from the original example used.

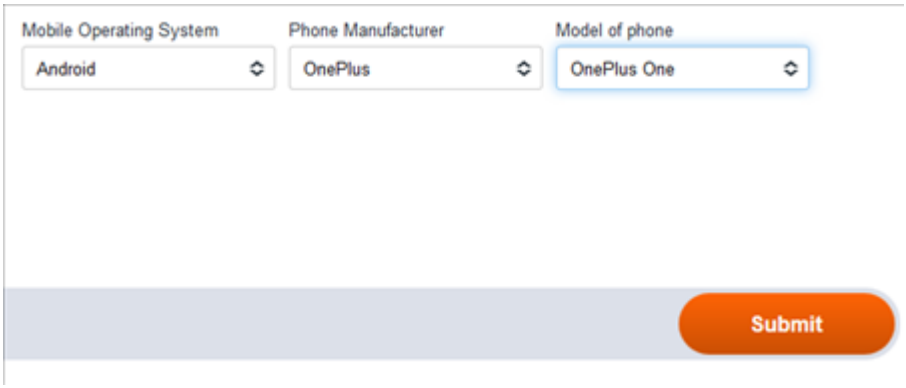
```
<root>
<os name="Android" value="Android">
  <manufacturer name="HTC" value="HTC">
    <model name="One" value="One"/>
    <model name="One X" value="One X"/>
  </manufacturer>
  <manufacturer name="Samsung" value="Samsung">
    <model name="Galaxy S3" value="Galaxy S3"/>
    <model name="Galaxy Note 3" value="Galaxy Note 3"/>
    <model name="Galaxy S4" value="Galaxy S4"/>
    <model name="Galaxy Express" value="Galaxy Express"/>
    <model name="Galaxy XCover 2" value="Galaxy XCover 2"/>
    <model name="Galaxy Fame" value="Galaxy Fame"/>
  </manufacturer>
  <manufacturer name="OnePlus" value="OnePlus">
    <model name="OnePlus One" value="OnePlus One"/>
  </manufacturer>
</os>
<os name="Windows Phone 7" value="Windows Phone 7">
   <manufacturer name="Nokia" value="Nokia">
      <model name="Lumia" value="Lumia" />
      <model name="Lumia 625" value="Lumia 625" />
   </manufacturer>
   <manufacturer name="Samsung" value="Samsung">
      <model name="OMNIA 7" value="OMNIA 7" />
   </manufacturer>
</os>
<os name="iOS" value="iOS">
  <manufacturer name="Apple" value="Apple">
    <model name="iPhone 5" value="iPhone 5"/>
    <model name="iPhone 5S" value="iPhone 5S"/>
    <model name="iPhone 5C Green" value="iPhone 5C Green"/>
    <model name="iPhone 5C Blue" value="iPhone 5C Blue"/>
    <model name="iPhone 5C Yellow" value="iPhone 5C Yellow"/>
    <model name="iPhone 5C Pink" value="iPhone 5C Pink"/>
    <model name="iPhone 5C White" value="iPhone 5C White"/>
  </manufacturer>
</os>
</root>
```

6. Click save and navigate back to the starting form page. Now click Data Config, located in the same field as Properties from earlier. In data config navigate to the Property Prefill Mapping tab. Now click the edit button.

7. On this page you will see a list of properties in a large table. Scroll down till you see the one you created earlier. On the right hand side of the page you will see the Targets pane click through the links there to open the path you saw when you changed the bindings for the Xml data in composer it should be something like the picture below.



8. Drag the _dropDownXmlData over to the property you created to make the link. Click save and then save again when it returns to the data config page. Now return to the form page. Now when you load the form in Transaction Manager it will reflect the changes made to the XML.



Next, learn about form property prefill mapping.

Form Versions Overview

Journey Manager (JM) | Form Builder Platform Developer | 17.10

Manager offers a flexible mechanism to create and preserve multiple versions of a form, which benefits in the following:

- Allows users to switch between different form versions.
- Allows controlled releases of new versions into a Manager environment.
- Provides the ability to roll back a form version if it has a problem or a bug. When developing and deploying forms, occasionally challenges will arise making it necessary to revert to a previous version of a form. Form versions are also required to support the form A/B testing feature in Manager.

You usually create a new form version in Maestro and then build a form version of the form and publish it in Manager.

This is the recommended approach as it is more manageable and easier to keep the form versions in sync across Maestro and Manager.

> It is not possible to create and maintain form versions in Transact Composer. Instead, Transact Composer users may choose to import the Transact Composer package, which is the published form, as a new version.

To start using a new form version, you need to apply the following configurations to it:

- Form version details
- Form version functions
- Form version services
- Form version attachment rules
- Form categories
- Form design info
- Job properties
- Job info

You can change the current form version at any time, whether to roll out a new release or roll back to an older version. The current form version is used whenever a user requests the form.

Next, learn how to configure a form version.

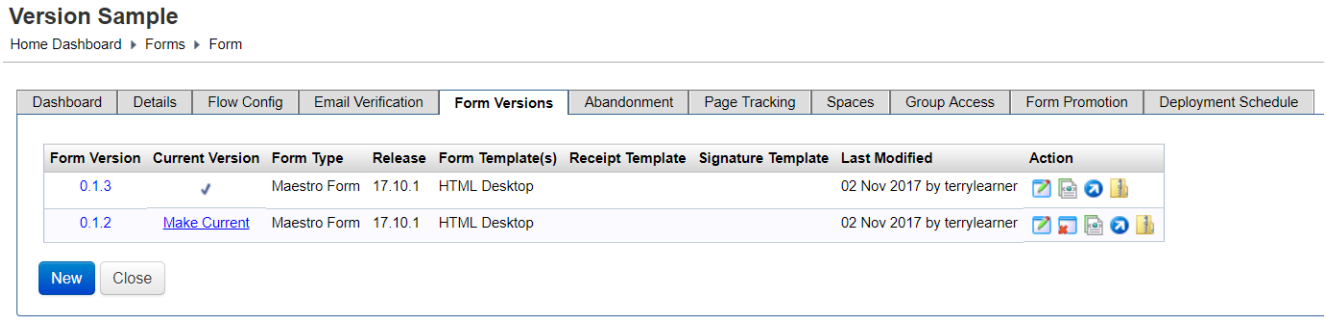
View Form Versions

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

Manager allows you to view the list of all form versions current configured.

To view all form versions of a form:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select the **Form Versions** tab to see the following details.



- Form Version - identifies the Form Version and corresponds to the Version Number in Maestro (most recent Form Versions listed first).
- Current Version - the ticked version is the current version. To nominate another version to be the current version, click the version's Make Current link.
- Form Type - identifies how the form was created.There are the following form types:
- Composer SmartForm - Composer FAR file (created when you use Publish as zip file option in Composer).
- Maestro Form - Maestro file (created when you download a form version after publishing).
- Release - identifies which release of Maestro/Composer the Form Version was published or built from.
- Form Template(s) - identifies the template of the form (Maestro/Composer forms are HTML Desktop).
- Receipt Template - (if exists) identifies the receipt template.
- Signature Template - (if exists) identifies the signature template.
- Last Modified - identifies when and whom last modified the form version in Maestro or Manager.
- Action - the following action options are available.

You can also perform the following actions:

- Click **Edit** to view and update a form version.
- Click Form Data Configuration to view and update form data configuration.
- Download the form template file.
- Click **Form Test Render** to render the form version in a new web browser tab.
- Click **Export Archive** to download a form version archive file.
- Click **Remove** to delete a form version.

> You can't remove the current form version.

Next, learn how to configure form abandonment.

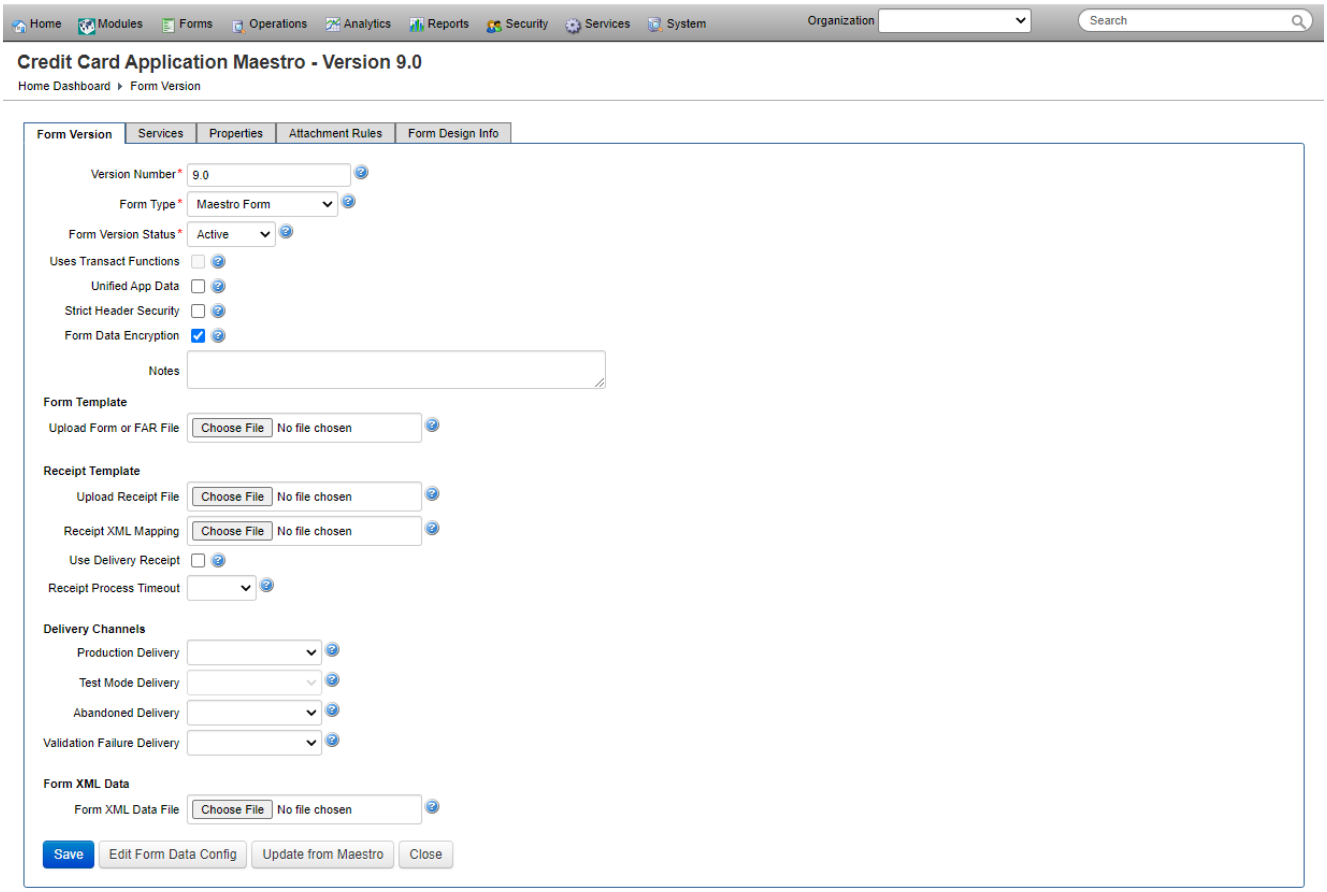
Configure a Form Version

Journey Manager (JM) | Form Builder Platform Developer | 21.11

Manager allows you to view and update configuration of a form version.

To configure the form version:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select a form version and click **Edit**..
4. Select the **Form Version** tab to view and modify the configuration.



5. Enter a version number of this form, built and published using Maestro, in the **Version Number** field.

   It's not recommended to update the version number manually.

6. Select a type of form deployed on the server from the **Form Type** dropdown list.

   A form type can be one of the following:

- Composer SmartForm
- Maestro Form
- JS App - refers to an Open UX form
- Dynamic PDF Form
- Static PDF Form
- Other Document

7. Select a status from the **Form Version Status** dropdown list. A form version status can be one of the following: | 21.11
   - Active - it is in use and readily available to all users. All functionality is allowed.
   - Offline - it can't be used at all. No functionality is available, including DDS, Delivery and so on
   - Inactive - similar to Offline.
   - NoRender - it doesn't allow for new or saved and resumed forms, but ongoing open sessions can be completed. DDS, submits, saves, and delivery are allowed.
8. The **Uses Transact Functions** checkbox specifies whether the Transact Function API programming model is available on the server. Transact Functions allow you to execute various functionality at crucial form events, such as form opening, saving or resuming, which gives a standardized approach to handling them. | 17.10

   > You can only change this setting by selecting or cleaning the **Use server side Transact Functions (requires TM 17.10 +)** checkbox in Maestro when you build a form.

9. Select the **Unified App Data** checkbox to enable the Unified Application Data Document Model in this form version. It provides you with the fine-grained control of the output data by decoupling the client-side data from the server side. For example, you may decide not to send sensitive fields to the user's browser. | 17.10
10. Select the **Strict Header Security** checkbox to enforce a strict policy that ensures that the form session and request HTTP headers, such as `IP Address`, `Referer`, and `User-Agent`, are coming from the same User Agent and IP Address that was used for the user's last login session to this form space. There is also an additional check to verify that the request was originally intended for this system. | 17.10

    > You can configure the **Strict Header Security** setting of a security manager, which can overwrite the form version's configuration.

11. Select the **Form Data Encryption** checkbox to use HTML client-side form data XML encryption, in addition to standard **TLS**[1] transport encryption. This is the default option.

---

[1] Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network.

Clear the **Form Data Encryption** checkbox if you need to run automated tests, in SIT and UAT environments, that require viewing and modifying the `formData` payloads sent from the web browser to Manager during POST requests.

12. Type the comments relevant to this form version in the **Notes** field.
13. Click **Upload Form or FAR File** to deploy a form template file in one of the following formats: PDF, **XDP**[1], and FAR.
14. Click **Upload Receipt File** to deploy the receipt templates file in one of the following formats: PDF, XDP, and FAR.
15. Click **Upload Signature File** to deploy the template file (PDF or XDP) to be presented to users for signatures. If not specified, the receipt file is used.
16. Click **Receipt XML Mapping** shows how receipt XML data is mapped to **AcroForm**[2] fields using **XQueries**[3]. You can use this in overlay receipts.
17. Select the **Use Delivery Receipt** checkbox to specify a separate delivery of a PDF receipt where **ReceiptMode** is set to **Delivery**. This enables delivery PDF receipt to display additional **Office Use Only** data, which is not printed on the user's PDF copy of the receipt.
18. Select a timeout for the PDF receipt rendering from the **Receipt Process Timeout** dropdown list. By default, no timeout is enforced.
19. Select a delivery channel from the **Production Delivery** dropdown list that will be used to deliver successful submissions for this form version if the form version is not in the test mode.
20. Select a delivery channel from the **Test Mode Delivery** dropdown that will be used to deliver successful submissions for this form version if the form version is in the test mode.

    This dropdown is enabled if the form is in test mode, that is the **Test Mode** checkbox is selected.

21. Select a delivery channel from the **Abandoned Delivery** dropdown list to deliver abandoned submissions.
22. Select a delivery channel from the **Validation Failure Delivery** dropdown list to deliver submissions with data validation errors.
23. Click **Form XML Data File** to upload a form XML seed data file, containing the data structure and the XML schema, to generate a blank seed XML file to be used in the form.
24. Click **Save** to update the changes.

---

[1]XML Data Package (XDP) format is a file format developed by Adobe Systems for packaging PDF data into XML files; contains the entire PDF document contents including form and template data; may also include character-encoded sections for binary content.

[2]AcroForms are the original PDF forms technology that accepts input in both Forms Data Format (FDF) and XML Forms Data Format (XFDF). Many 3rd party vendors support AcroForms.

[3]XQuery is a specification for a query language that allows a user or programmer to extract information from an Extensible Markup Language (XML) file or any collection of data that can be XML-like.

25. Click **Edit Form Data Config** to configure form version data mapping.
26. Click **Update from Maestro** to update the published form version using Maestro.
27. Click **Save** to update the changes.

Next, learn how to configure services for a form version.

Configure a Form Version Metadata

Journey Manager (JM) | Form Builder | Platform Developer | Removed in 18.05.

Manager allows you to view and update metadata of a form version. Metadata can be used to run queries on the set of forms defined in Manager using a Web Service API. Each form version maintains its own set of organization and form metadata independently.

To add a metadata value:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select a form version.
4. Select the **Form Tags** tab and click **New**.
5. Choose a metadata tag to assign a value to from the **Tag Name** dropdown. If no suitable metadata tag exists in the system, you can either create one first, or leave the **Tag Name** dropdown blank.
6. Provide a name in the **Name** field.
7. Select a data type from the **Data Type** dropdown list.
8. Select a scope from the **Scope** dropdown list.
9. Set a value for this metadata for this particular form version in the Value field.
10. Click **Save** to update the changes. You will see that the metadata value has been added to the "Form Tags" tab.

To configure the form version metadata:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select a form version.
4. Select the **Form Tags** tab to view and update tags.
5. Click **Save** to update the changes.

Click **Sync Org Tags** to synchronize organization metadata values with the current values stored for the organization.

Click **Validate Tags** to validate the metadata values for the form version. Any errors, such as missing values for required metadata tags, will be reported so you can correct them.

Next, learn how to configure services for a form version.

Form Version Services

Journey Manager (JM) | Platform Developer | 5.1 & Higher

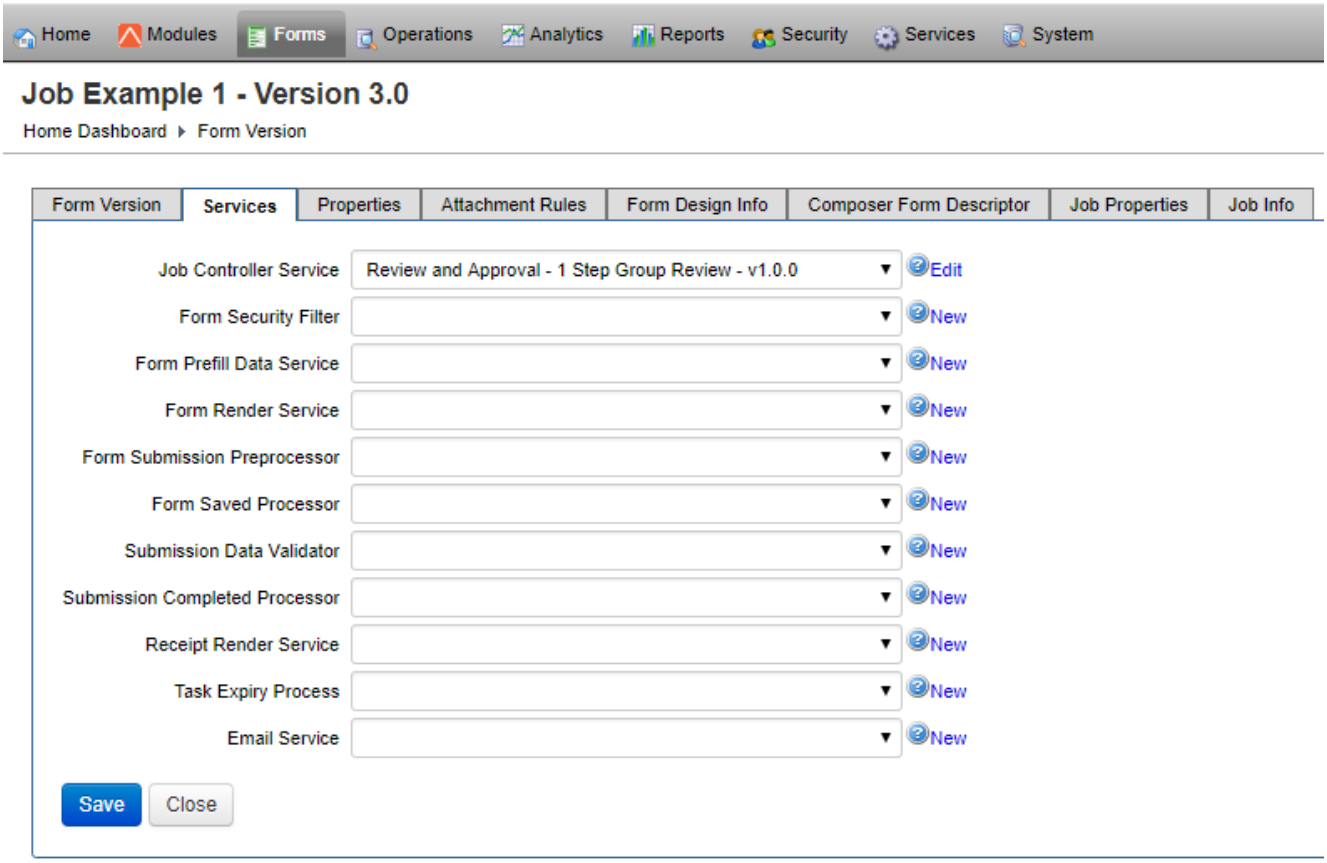
Manager comes with a set of standard services which you can assign and customize for each form version so they will be invoked at different stages of form processing. If services are not assigned, Manager uses the system defaults. It is recommended to assign services to a form version only when necessary.

You can also create and customize optional services for a form version, for example, when a form version is configured with a collaboration job. The optional services doesn't default to any services when not selected. For example, if a form version uses a collaboration job, then you need to select the applicable job controller service. Otherwise, this form is a standard form and no system default service will be used for the job controller service.

To view and assign services to a form version:

1. Select **Forms > Forms**.
2. Locate a form and click the **Form Version** tab.
3. Select a form version and click the **Services** tab.

> The list of available services is limited when the **Use Transact Functions** checkbox is selected from the Build Options in Maestro. | 17.10



4. Select a Job Controller service from the **Job Controller Service** dropdown list, which is invoked when a form submission creates a new collaboration job.
5. Select a form security filter service from the **Form Security Filter** dropdown list, which controls user access to the form. This service is called when a form is rendered before Manager access controllers run.
6. Select a form prefill data service from the **Form Prefill Data Service** dropdown list, which is invoked to provide XML prefill of data when a form is rendered.
7. Select a form render service from the **Form Render Service** dropdown list, which is called to render the form template.
8. Select a form submission preprocessor service from the **Form Submission Preprocessor** dropdown list, , which is called when a form is submitted so it can review, modify or reject the submission before it is stored in Manager.
9. Select a form saved processor server from the **Form Saved Processor** dropdown list, which is called when a form is saved online. Optional.
10. Select a submission data validator service from the **Submission Data Validator** dropdown list, which is called to validate the submitted form XML data
11. Select a submission completed processor service from the **Submission Completed Processor** dropdown list, which is invoked when a submission changes to a form status Completed (form submission for a collaboration job is complete) and no more user steps are required.
12. Select a receipt render service, such as Dynamic PDF Receipt service, from the **Receipt Render Service** dropdown list, which is called to render the form submission as a PDF receipt. If not specified, the form uses the system default rendering service.
13. Select a task expiry service from the **Task Expiry Service** dropdown list, which is invoked when tasks expire..
14. Select an email service from the **Email Service** dropdown list, which is called to send submission related emails to users. This service uses the system default.
15. Click **Edit** to configure a service.
16. Click **New** to create a new service.
17. Click **Save** to update the changes.

Next, learn how to configure form version properties.

Form Version Properties

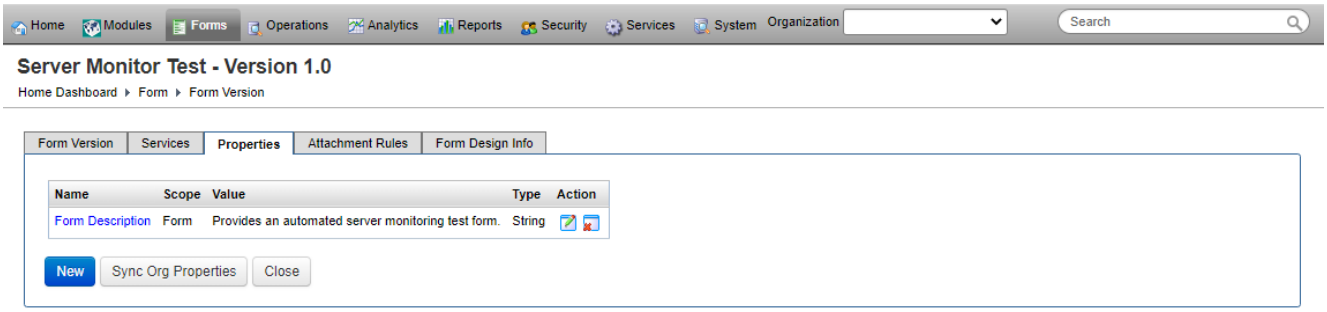
Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to create, edit and view properties of a selected form versions. You can use properties to define common reusable form elements. It can be a layout and content of a specific email message that include any required dynamic data, such as user and form details. In some cases, you need to write a script to make use of these elements. Properties are commonly used for prefilling data of form versions dynamically so you don't need to update a form when data changes.

When creating a form version, Manager includes a standard set of properties for it with already prefilled default values. Form version's properties allow you to create properties to override the existing form space's and organization's properties.

To view properties for a form version:

1. Select **Forms > Forms**.
2. Locate a form version and click **Edit**.
3. Select the **Properties** tab.



4. Click **New** to create a new property.
5. Click **Edit** to modify an existing property.
6. Click **Remove** to remove an existing property.
7. Click **Sync Org Properties** to synchronize a form version with the organization properties. Manager refreshes all form version properties customized where:

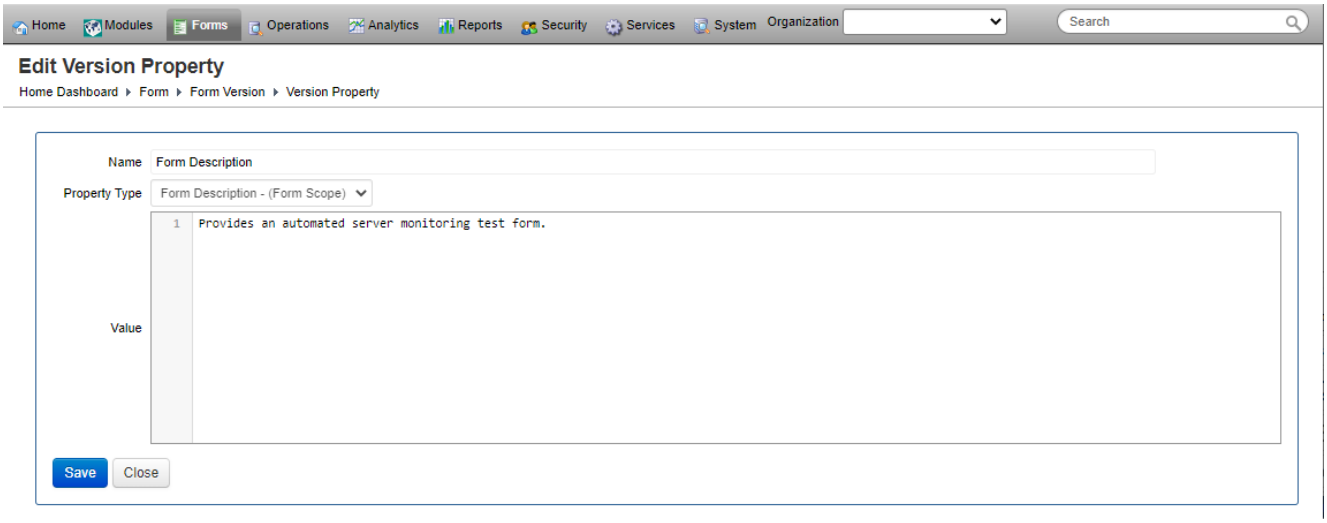
- The Property Scope is Client.
- There exists a customized property of the same name for the organization (of the form).

> Before proceeding with the Sync Org Properties, it's recommended that you back it up, because the sync is done immediately without requesting any confirmation.

To create a new form version property:

1. Select **Forms > Forms**.
2. Locate a form version and click **Edit**.

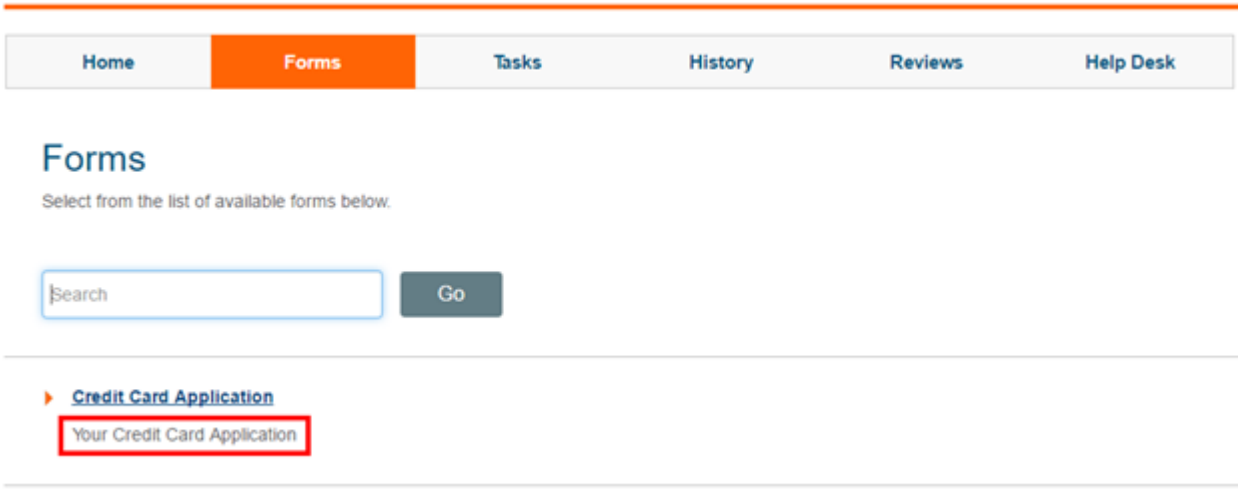
3. Select the **Properties** tab and click **New.**



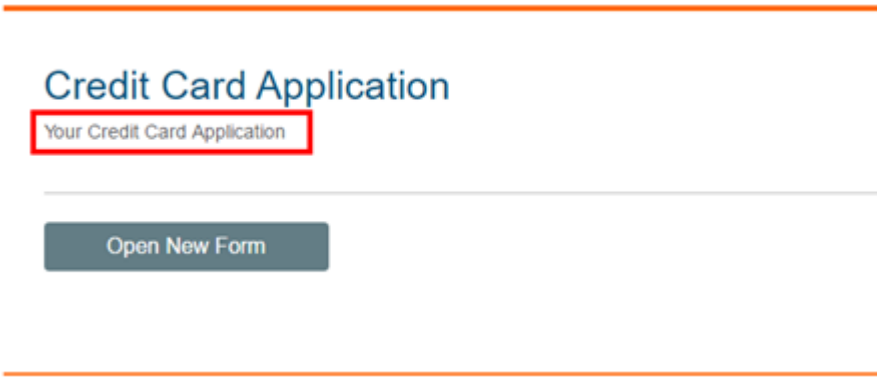
4. Provide a property name in the **Name** field, if the property doesn't have any property type, that is the **Property Type** dropdown is left blank. Otherwise, the name is populated based on the selected property type below.
5. Select a desired property type from the **Property Type** dropdown list, which populate the **Name**, **Data Type**, and the **Value** fields. You can leave the property type blank when no suitable type can be found in the dropdown list. Alternatively, you may also add a new property type and then use it here.
6. Select a data type from the **Data Type** dropdown list. The field is populated depending on the property type you select above. However, you have to select an appropriate data type when the property type was left blank.
7. Provide or update a value in the **Value** field, which is populated based on the property type you selected. It coan be a file upload field, depending on the property type.
8. Click **Save** to update the changes.

To edit a form version property:

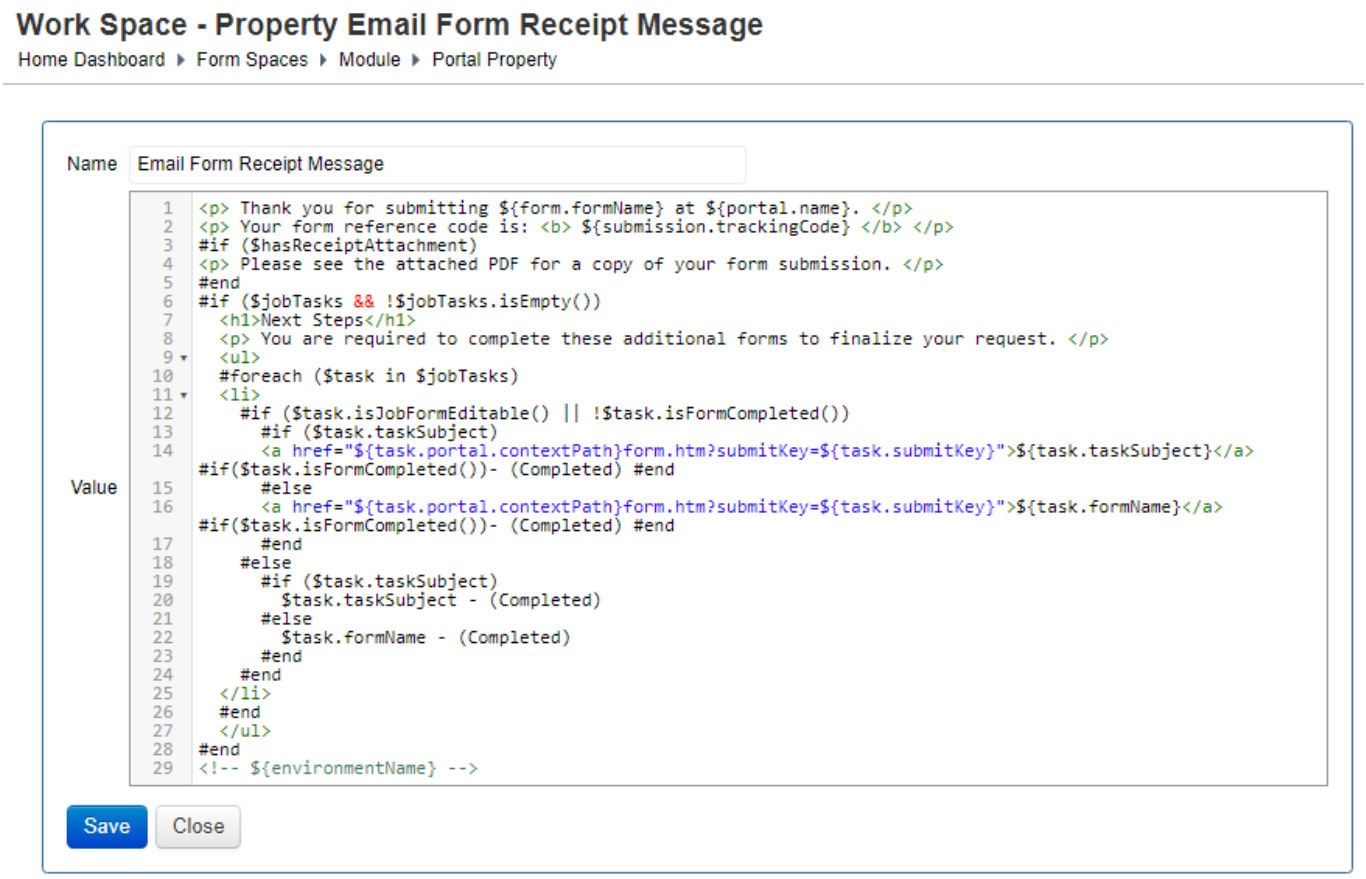
1. Select **Forms > Forms**.
2. Locate a form version and click **Edit**.
3. Select the **Properties** tab.
4. Locate a form version property, for example the **Form Description**, and click **Edit.**
5. Update the property with a new description for your form in the **Value** field to assist form users in choosing the correct form, the form description is displayed in the list of forms a user can view in the form space, as well as, on the form's landing page.
6. Click **Save** to update the changes.
7. Check the list of forms in the form space, with the form description that was entered in the form version property:



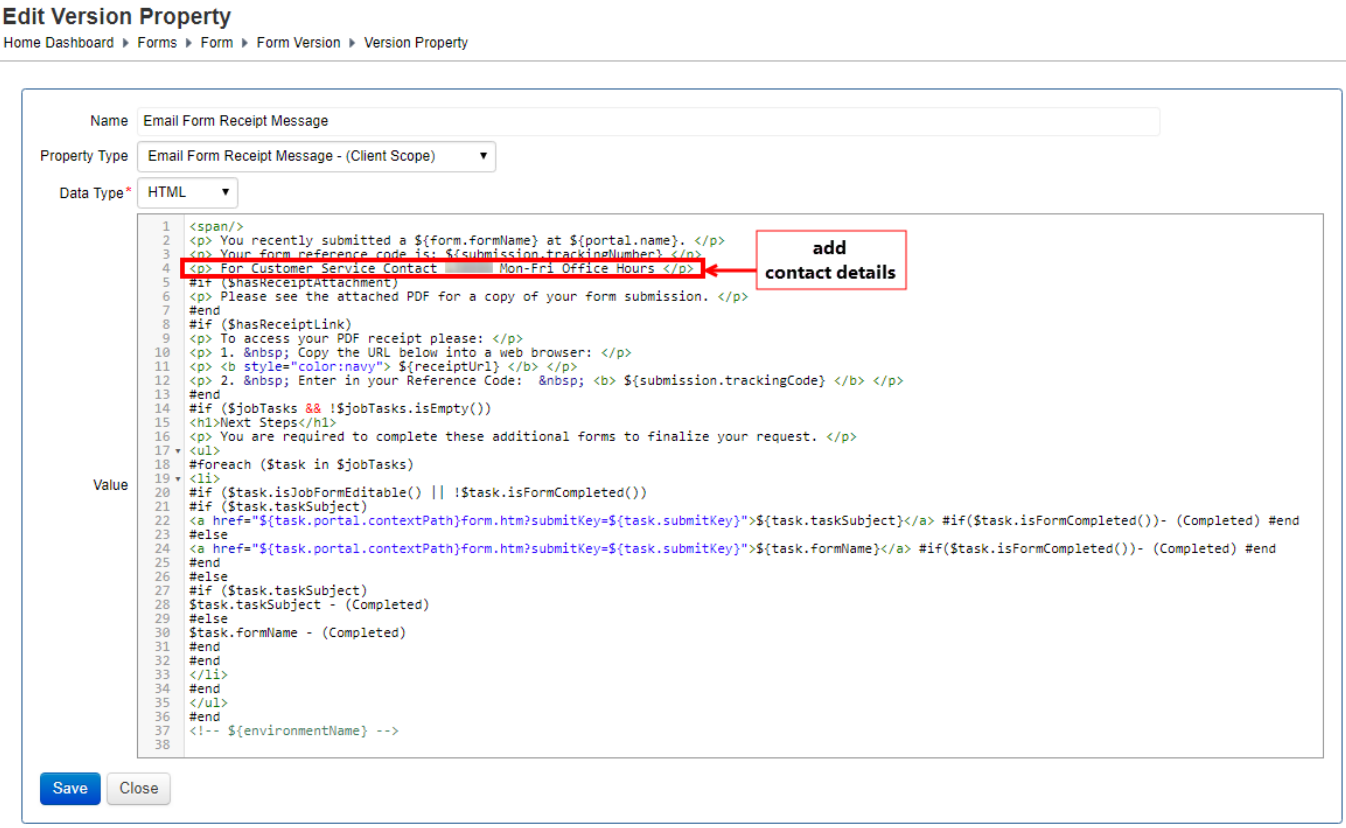
Then click the form to see the form's landing page, with the form description that was entered in the form version property, as shown below:



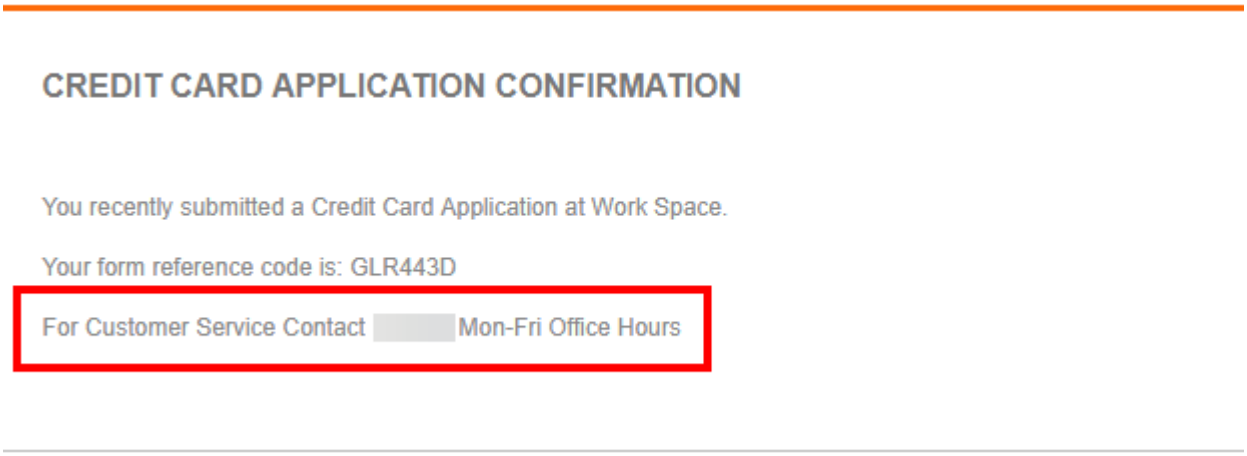
You may want to customize a form version property, for example, the form receipt email message by overriding the default **Email Form Receipt Message** property, which was initially created for a form space as shown below.



Let's suppose you need to add contact details to the message. You locate the existing message and edit its value preserving the HTML format for the message as shown below.



Now, when a form user submits this form, an email message will include the customer service contact details. However, this override doesn't affect the email receipt messages for any other form.



For information about creating and using custom properties, see properties.

Next, learn how to configure form version attachment rules.

Form Categories

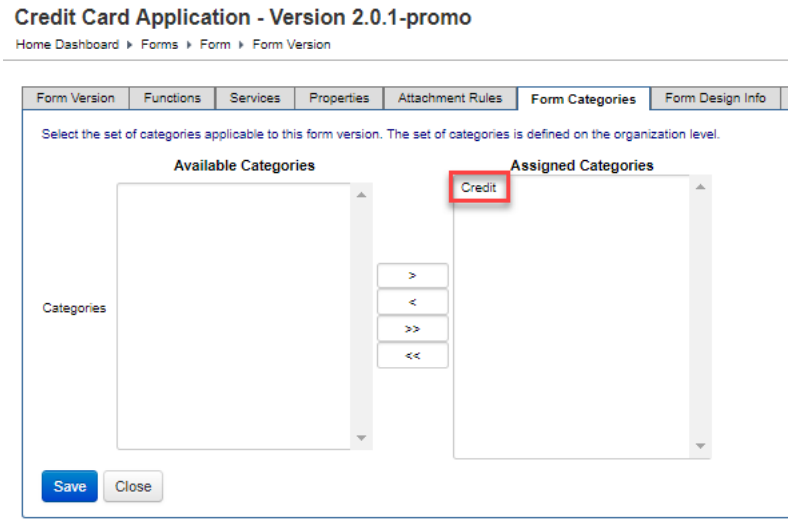
Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to assign forms to one or more form categories, for example "Business", to enable form space users to search forms by category, if the form space supports it. Form categories are especially useful if an organization has many forms.

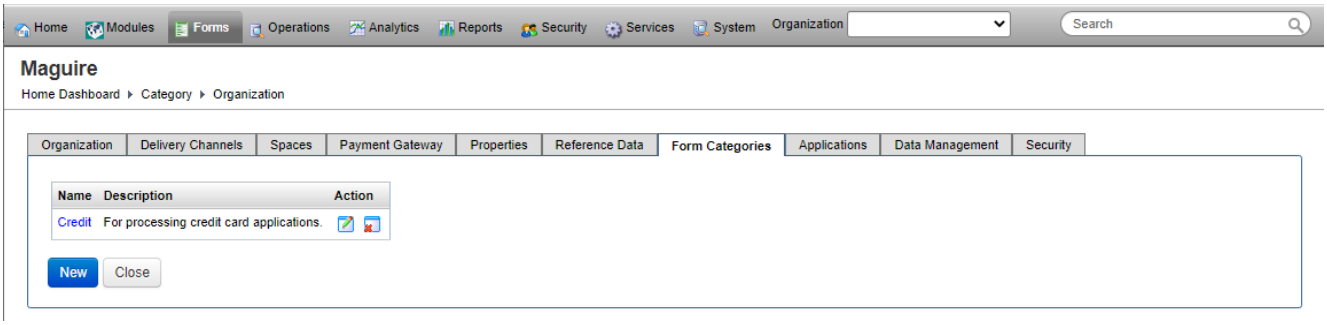
> You must configure form categories on the organization level before you can assign them to your form versions.

To assign a form version to an organization's form category:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select a form version.
4. Select the **Form Categories** tab.
5. Select one of the available categories from the **Available Categories** and then click **>** to move them to the **Assigned Categories**.



6. Click **Save** to update the changes.
7. Check the category.



8. When at least one form version is assigned to a form category, the **Category** dropdown list is displayed in the list of form versions and can be used to filter the list of forms by a form category.
9. Click **Close**.

Next, learn about form design info.

Form Design Info

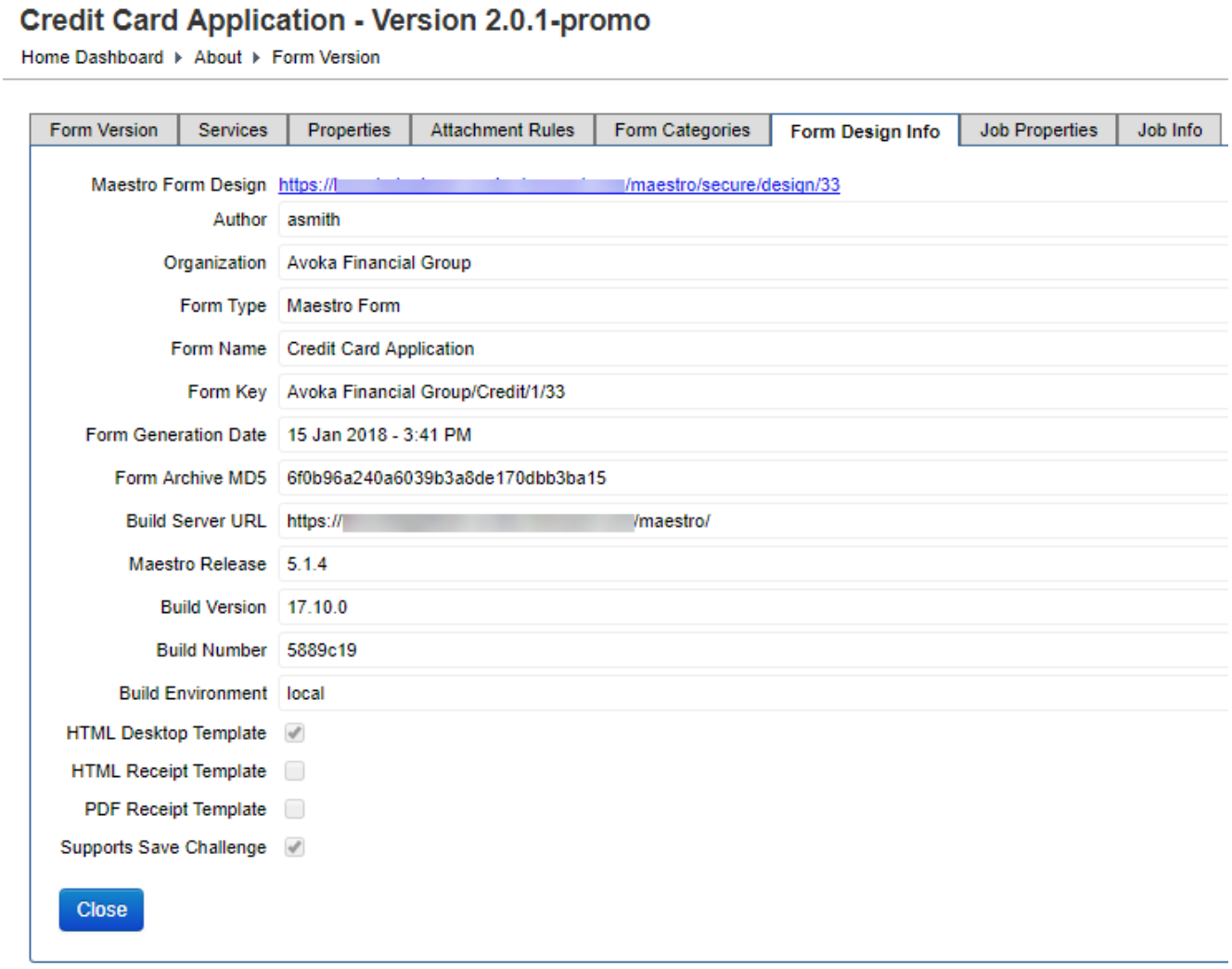
Journey Manager (JM) | Platform Developer ▪ Form Builder | 17.10

Manager allows you to view key information about a form design, which is useful when you want to develop and build a form version continuously. You can navigate back to the originating form version in Maestro using the **Maestro Form Design** url to update form elements.

> In Manager releases prior to v17.10, the Form Design Info tab is named the Form Archive Info tab.

To view form design information for a form version:

1. Select **Forms > Forms**.
2. Locate a form version.
3. Click **Form Design Info** to see the following form design and build audit details:



- **Maestro Form Design -** displays a direct link back to the corresponding form version in Maestro. Click this link to open the form version. If you are not already logged in to Maestro, you will be prompted to enter your credentials.
- **Author -** the login that was used to create the form version.
- **Organization** - the organization that the form belongs to.
- **Form Type** - the type of form created, such as Maestro or Composer.
- **Form Name -** the name of the form.
- **Form Key** - the key of the form.
- **Form Generation Date -** the date that the form was generated (built).
- **Form Archive MD5** - the MD5 checksum of the form archive.
- **Build Server URL** - the URL of the server used to build the form.
- **Maestro Release** - the Maestro release number that was used to build the form.
- **Build Version** - the version of the core Maestro platform used to build the form. A build version that is different than the Maestro Release version means that the form has not been upgraded to the new release and hence has been built to be compatible with the older version.
- **Build Number** - the number of the build used to build the form version.
- **Build Environment** - the environment used to build the form. For Maestro it is normally local,but for Composer can reflect the test and staging environments.
- **HTML Desktop Template** - indicates whether this form version has an HTML desktop template.
- **HTML Receipt Template** - indicates whether this form version has an HTML receipt template. This is for Composer use.
- **PDF Receipt Template** - indicates whether this form version has a PDF receipt template. This is for Composer use.
- **Supports Save Challenge** - indicates whether this form version supports the Save Challenge feature. If this option is selected, Manager takes a user, who is trying to resume a form application, to a form page with the Save Challenge dialog. Otherwise, it redirects a user to the **`web-plugin/save-challenge.htm`** portal page.

4. Click **Close**.

For Composer forms, the details displayed on the Form Design Info tab are slightly different. These variations in the fields are applicable to Composer features including legacy ones.

Next, learn about form version details.

Job Properties of a Form Version

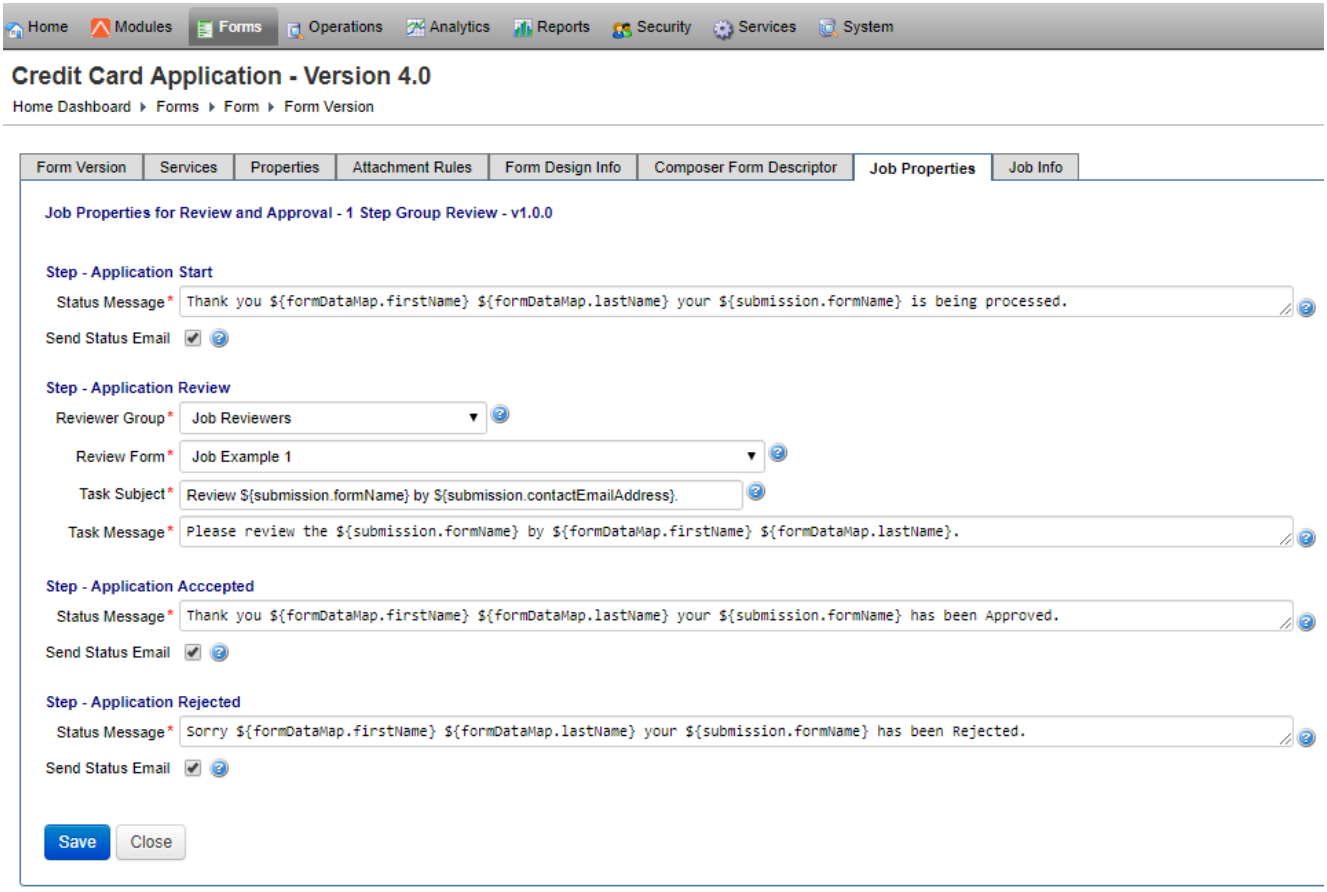
Journey Manager (JM) | Platform Developer | 5.1 & Higher

Journey Manager allows you to assign a job controller service to a form version so it becomes a part of a collaboration job. Then, you can configure job properties of this form so they are used by the collaboration job.

> The **Job Properties** tab is only available to a form version that is associated with a review and approval collaboration job. It isn't available to an onboarding collaboration job.

To configure collaboration job's properties:

1. Locate a form version.
2. Select the **Job Properties** tab to customize the job controller of a collaboration job. The set of parameters you can configure depends on the collaboration job associated with this form version. These parameters are grouped by each collaboration job's step.



3. Enter the message that will be sent to the applicant to update the status of their application in the **Status Message** field.
4. Select the **Send Status Email** checkbox to send an email to an applicant with the status update of a form application.

5. Select the form which will be used to review the application from the **Review Form** dropdown menu.
6. Select the staff group that will review the application from the **Reviewer Group** dropdown.
7. Select the **Tasks Claimable** checkbox to allow group reviewers to claim review tasks.
8. Enter the subject line for the review task email in the **Task Subject**.
9. Enter the detailed message for the review task email in the **Task Message**.
10. Click **Save** to update the changes.

Next, learn about form version job info.

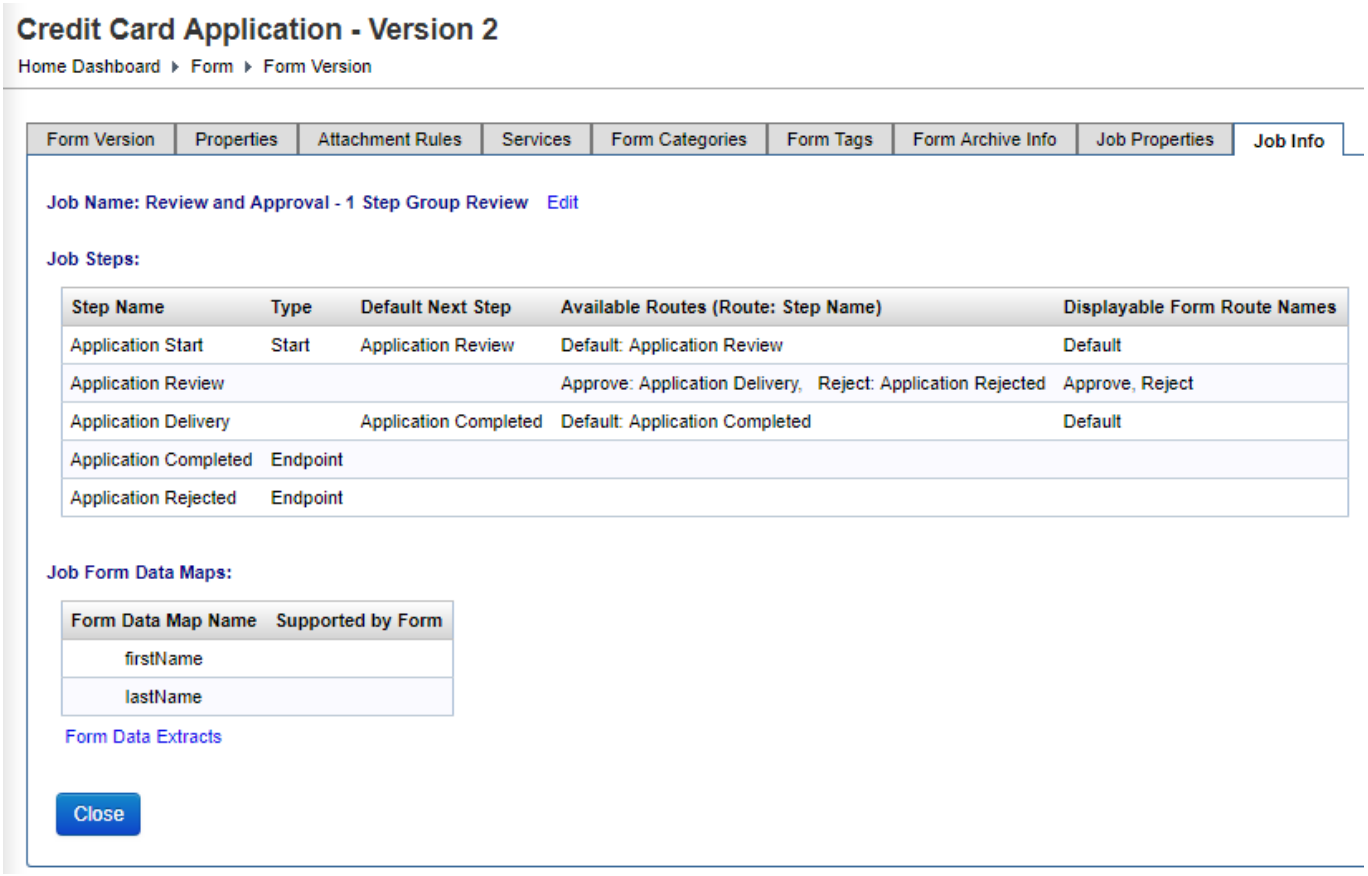
Job Information of a Form Version

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to assigned a job controller service to a form version so the form becomes a part of a collaboration job. You can view collaboration job information related to this form version in the Job Info tab.

> The Job Info tab is only visible if a job controller service is configured for a form version.

To view collaboration job information for a form version, select the form version and click the **Job Info** tab.



Next, learn about form version job properties.

Delete a Form Version

Journey Manager (JM) | Form Builder Platform Developer | 17.10

Before deleting a form version in Manager, consider if it is necessary to do so. When you delete a form version, all its related entities including submissions are also deleted.

Instead of deleting a form version, you may wish to:

- Leave the form version and assign another form version as the current version.
- Retire the entire form by deactivating it.

> Any form version except the current version can be deleted.

Since v17.10, the Journey platform follows open source semantic versioning conventions. Following this convention, all new form version names are recommended to start at version 1.0-develop until they reach production. In versions of Maestro prior to 17.10, form versions are incremented by whole numbers (for example version 1, version 2, version 3, etc.).

To delete a form version:

1. Select **Forms > Forms**.
2. Locate a form and select **Form Version** tab.
3. Check if the form version you wish to delete is the current form version - the **Current Version** column is selected. If it is so, reassign the correct form version as the current form version by clicking on its **Make Current** link.
4. Click **Delete**.
5. Click **OK** button to confirm your decision.

Next, learn how to delete Maestro form versions.

Delivery Channels Overview

Journey Manager (JM) | System Manager / DevOps | 21.05

Journey Manager comes with several delivery channels to allow you to send user form submissions to a platform backend or a third-party system for processing. After a user submits a form, a delivery channel determines what happens to the submitted, abandoned or invalid form transactions.

A delivery channel transfers submitted user form details to a platform back-end or to a third-party system using one of the following mechanisms:

1. PDF receipt - a readable copy of a form with user details.
2. Form XML data - a machine-readable copy of form details in **XML**[1] format.
3. Attachment - a file which is a part of a form submission, for example, a copy of a driver license.

You can configure different delivery channels for:

- An organization - you can configure the organization-wide delivery channels, which are used by all forms and form versions if their delivery channels are not configured.
- A form level - you can configure a delivery channel, different to the organization's one, for individual forms. This gives you an extra flexibility to have different delivery flows for various forms within one organization.
- A form version level - you can configure a delivery channel, different to the form's one, for individual form versions. This means you can implement requirements when each form version of a form should use various delivery channels. | 21.05 .

Manager determines a delivery channel to use in this order:

1. The form version's delivery channel
2. The form's delivery channel, if the above is not set
3. The organization's delivery channel, if the above is not set
4. The Trash Can delivery process, if the above is not set

A delivery channel can have one of the following delivery methods, which you can customize to suit your specific requirements and your company policies.

Delivery Process

This is the most widely used method and is recommended as a more customizable approach for an immediate and secure form of delivery.

---

[1]XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

Delivery Process invokes a service definition in Manager to perform delivery of submission details. This is typically a customizable Groovy delivery service.

Delivery is marked as complete if the service indicates that delivery was successful.

If delivery has not been completed (by default 72 hours after the form was submitted), escalation emails are sent out to all global and organization administrators in the Receive Delivery Escalation Alerts group.

Email

The Email method sends the submission details as attachments to one or more configured email addresses. This email can include a link to the submission attachments (if any were included by the user). In Manager, delivery is marked as complete as soon as the email is sent.

Email Secure

The Email Secure method differs from Email in that the email sent does not contain any submission details. Instead, it contains a link to a secure page in Manager where the submission details can be downloaded.

Delivery is marked complete only after a user has accessed this secure page and confirmed that the submission has been received.

If delivery has not been acknowledged within a certain period, which is 24 hours by default, reminder notifications are sent to the recipients defined in the delivery channel. This continues daily until a recipient acknowledges it.

If the delivery channel's Email Addresses field is left blank, a reminder is sent to all global and organization administrators in the Receive Delivery Escalation Alerts group.

If delivery remains unacknowledged (by default 72 hours after the form was submitted), escalation emails are sent out to all global and matching organization administrators in the Receive Delivery Escalations Alerts group.

REST Service

The REST Service method stores the submission details within Manager and offers a secure method of the enterprise retrieving these submission details, using a REST endpoint, when convenient.

For more information on REST service delivery, see the Groovy services guide.

Web Service

The Web Service method notifies your enterprise by making a web service call, using a pre-defined API. For more information, see the Manager Web Service API documentation.

Web services may be either defined as part of the **Integration Agent**[1] or defined separately by a developer.

Delivery is marked as complete once the web service client confirms that it has received the submission.

If delivery has not been completed (by default 72 hours after the form was submitted), escalation emails are sent out to all global and organization administrators in the Receive Delivery Escalation Alerts group.

Trash Can

The Trash Can Delivery process is a do-nothing delivery process for form testing. This is usually configured as the default and won't be available in the organization configuration, but is available in the form configuration.

Next, learn how to create a delivery channel.

[1]Transact Integration Agent is a software appliance hosted inside the client organization's data center, which receives form submission deliveries and file attachments from the Journey Manager server.

Create a Delivery Channel

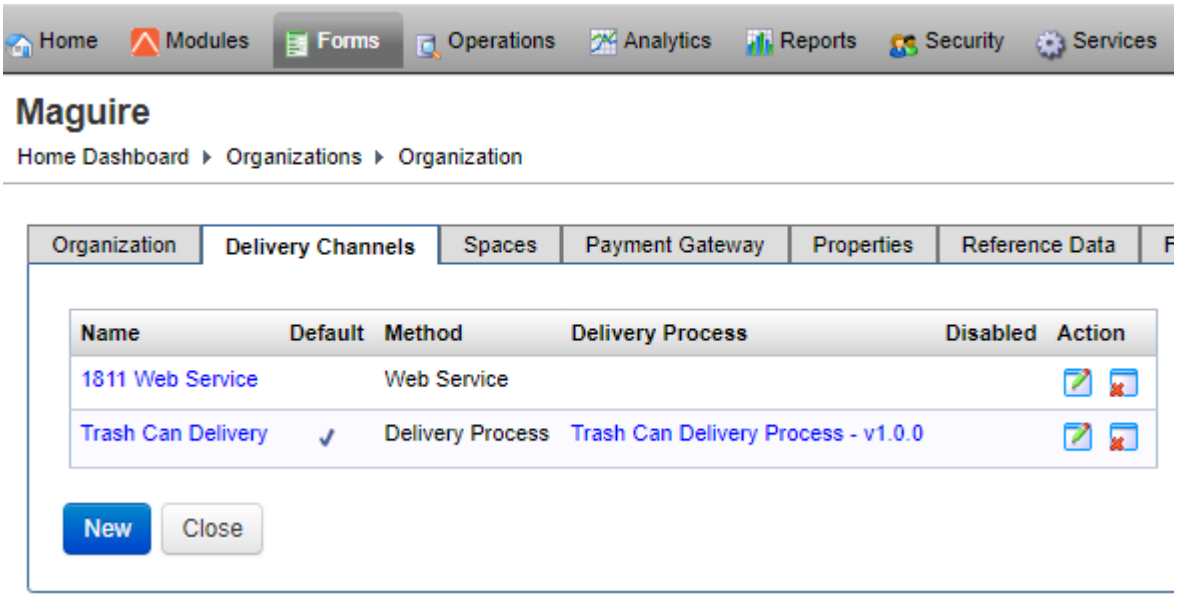
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure a delivery channel for an organization.

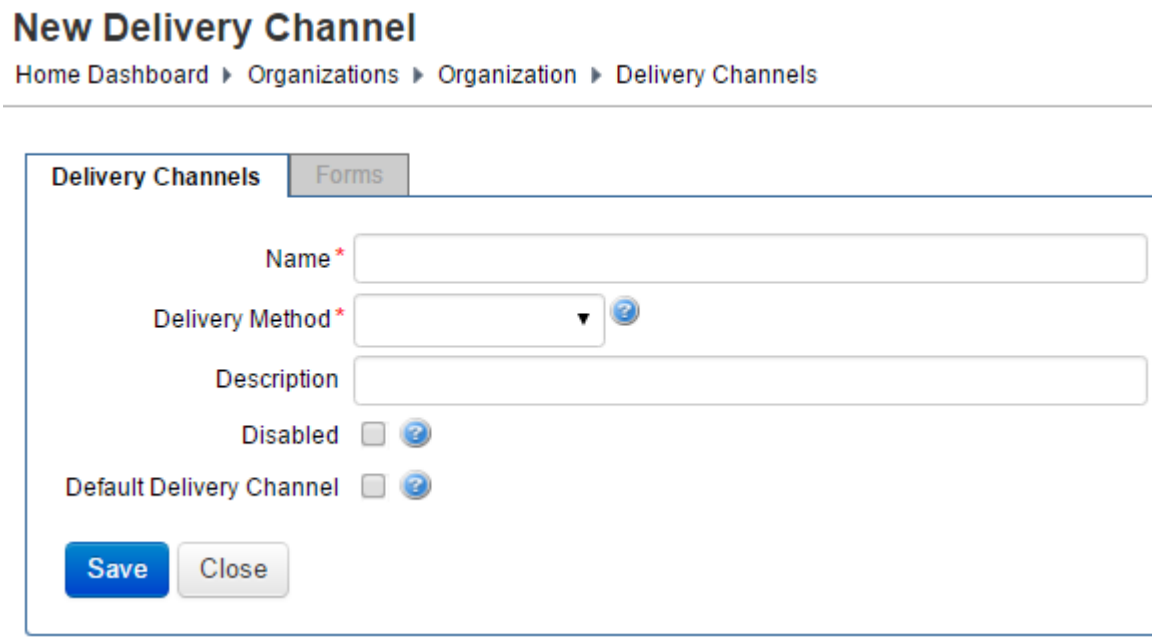
> A delivery channel configuration is different for each delivery method.

To create a delivery channel:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Delivery Channels** tab to see a list of all delivery channels for the organization.



4. Click **New** to create a new channel.



5. Enter a unique name, within your organization, for the delivery channel in the **Name Enter** field.

> The default **Trash Can Delivery** is set by default, so collaboration jobs can run. However, you may want to choose user email delivery.

6. Select the delivery method from the **Delivery Method** dropdown list.

   A delivery channel method can be one of the following:

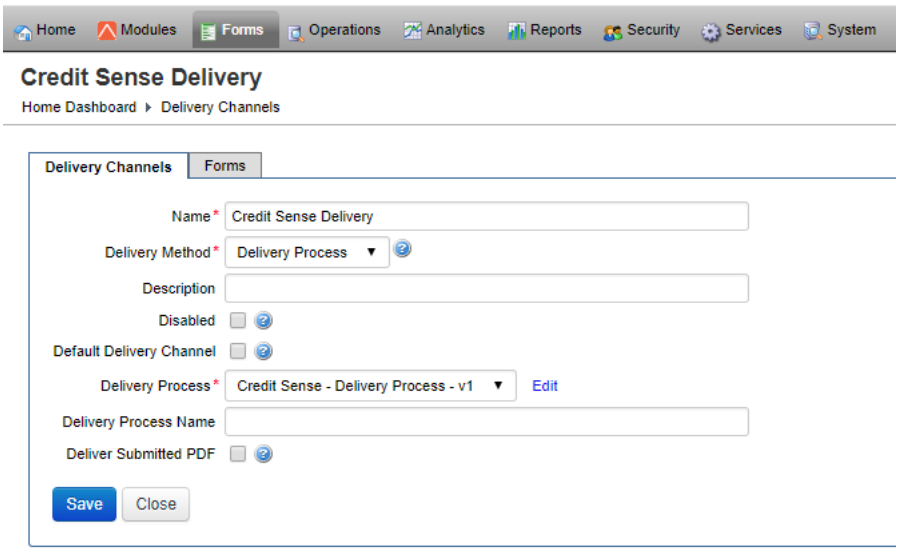
   - Delivery Process
   - Email
   - Email Secure
   - Web Service
   - REST Service

7. Enter a description of your delivery channel in the **Description** field.
8. Select the **Disabled** checkbox to disable the delivery channel temporarily.
9. Select the **Default Delivery Channel** checkbox to make it the default delivery channel for any forms within this organization that have not yet been associated with a delivery channel.
10. Click **Save** to update the changes.

To configure a delivery channel:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Delivery Channels** tab to see a list of all delivery channels for the organization.

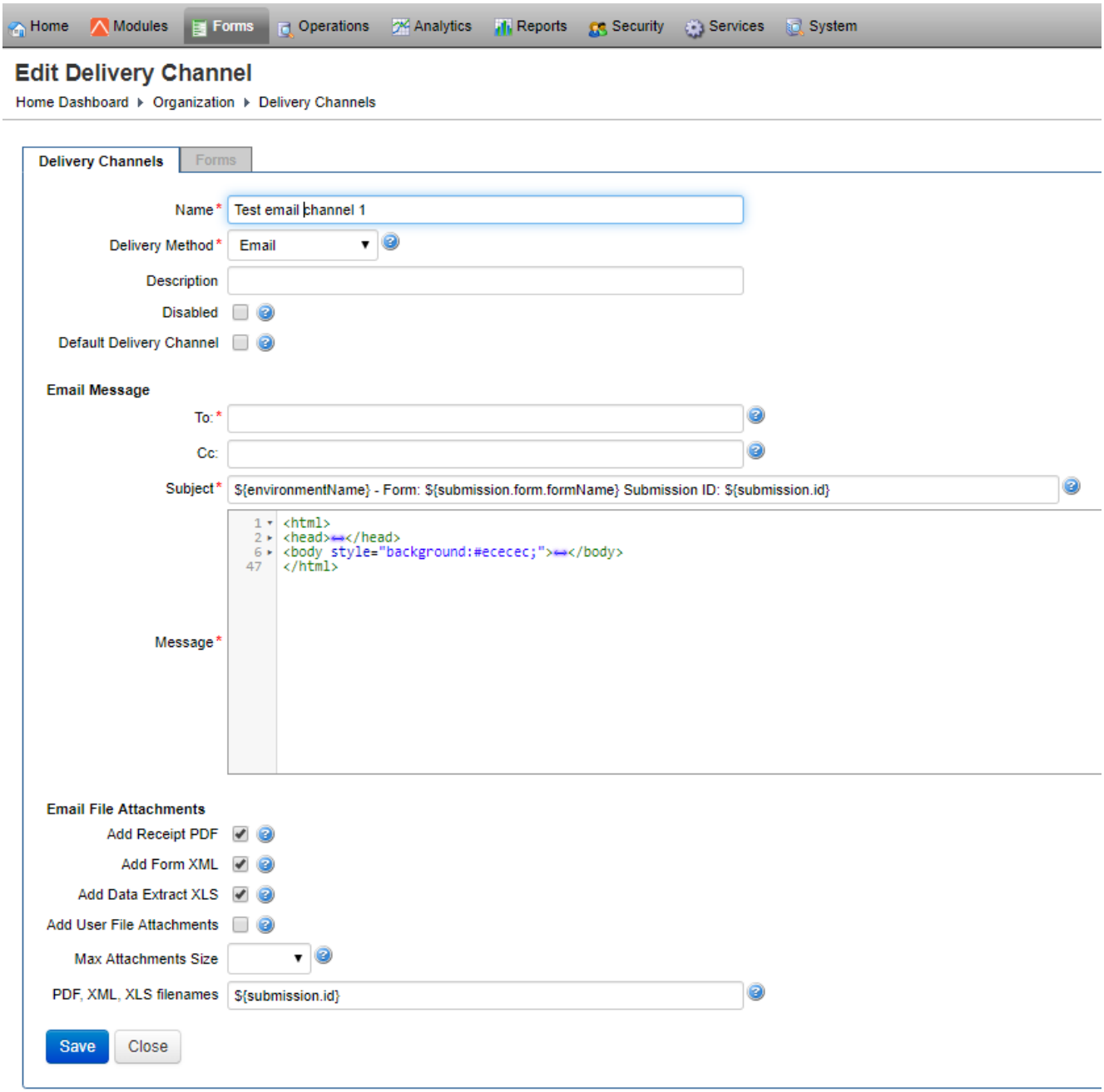
4. Click **Edit** to update the existing one.



5. Select a delivery process from the **Delivery Process** dropdown list.

> A delivery process invokes a service definition in Manager to perform the delivery.

6. Enter a name into the **Delivery Process Name** field.
7. Select the **Deliver Submitted PDF** checkbox to include the submitted PDF in editable form in the delivery.
   | 17.10
8. Configure email specific settings:

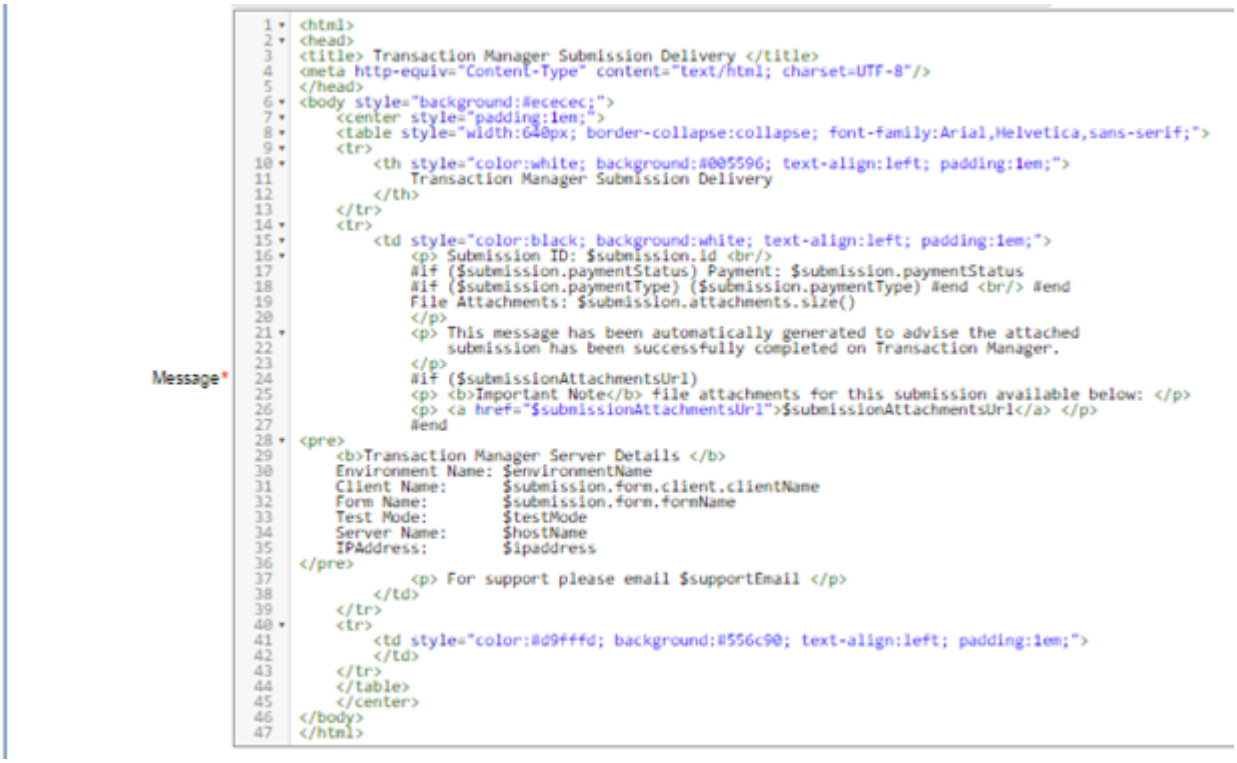


Email delivery sends out an email to a set of email addresses. The email can contain the submission data. Delivery is marked as completed as soon as the email has been sent.

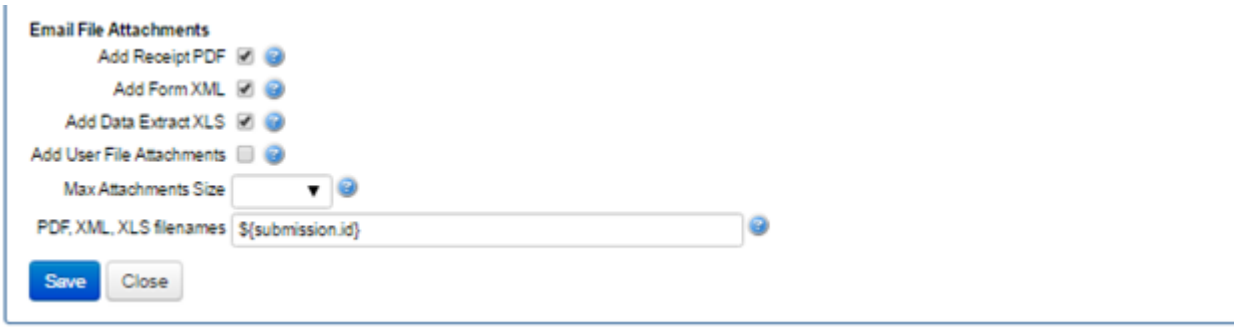
9. Enter one or more email addresses for the email recipient(s) separated by commas in the **To** field.
10. Enter one or more email addresses separated by commas for the email recipient(s) in the **Cc** field (optional).
11. The **Subject** field is pre-populated with identifying details of the environment, form name, and submission identifier. You may choose to customize this subject by changing it here. It is possible to change the subject

at the organization level as a means of affecting the defaults for the subject whenever a new email delivery channel is to be created.

12. The **Message** field is pre-populated with a message. You may choose to customize this message by changing it here.



13. Edit the **Email File Attachments** section:



14. The **Add Receipt PDF** checkbox is already ticked to attach the PDF receipt with the email. Deselect this checkbox if you do not want to include the PDF Receipt.
15. The **Add Form XML** checkbox is already ticked to attach the form submission XML file with the email. Deselect this checkbox if you choose not to include it.
16. The **Add Data Extract XLS** checkbox is already ticked to attach the Excel file of the form submission data extracts with the email.
    Deselect this checkbox if you choose not to include it.

17. If there are file attachments that a user must include with the email, then select the **Add User File Attachments** checkbox.
18. If your email server limits the email size, then choose the appropriate size (in megabytes) from the **Max Attachments Size** dropdown. Whenever the limit is reached, a link to the submission is provided instead.
19. The **PDF, XML, and XLS filenames** field is the PDF receipt file name, form XML and Excel Data Extract XLS files created. It is already set to the submission id as a **Velocity**[1] variable, **`${submission.id}`**. The file name must comprise only supported Velocity variables, such as:
    - Receipt number - **`${receiptNumber}`**
    - Submission - **`${submission}`**
    - Form - **`${form}`**

    Manager automatically appends the correct file extensions, such as .PDF, .XML or .XLS.
20. Click **Save** to update the changes.

If you create a new Delivery Channel of the type Groovy Delivery Process you have the option to set it up with a default template for a Fluent, Fluent with Checkpoints, or a Groovy Delivery process. The actual service code can then be updated to suit the clients needs.

Next, learn how to create Email Secure delivery channel.

[1]Apache Velocity Templates are used extensively throughout Journey platform to define html page layout, email body and subject. A template is combined with model containing a number or objects to produce a String. For more information see http://velocity.apache.org/engine/devel/user-guide.html

Create an Email Secure Delivery Channel

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager comes with the Email Secure delivery channel that is like the standard email delivery in that it sends out an email to notify the organization of a submission. However, the email itself does not contain any submission data; instead, it contains a link to a secure page on Manager which provides the submission data for download.

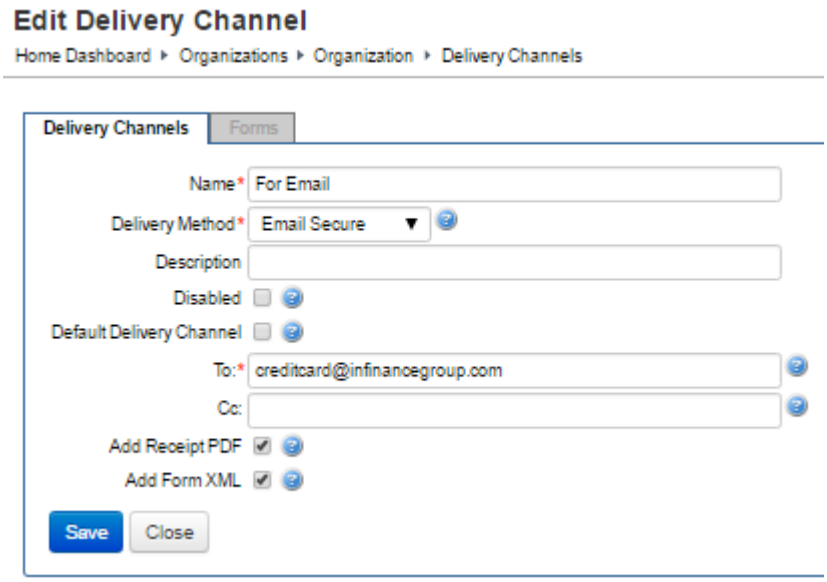
In addition, delivery completes only after an administrator user has accessed this page and confirmed that the submission has been received.

If delivery has not been acknowledged within a certain time period (by default 24 hours), reminders are sent out to the recipients defined in the delivery channel. If the delivery channel's "Email Addresses" field is blank, the reminder is sent to all global and organization administrators in the Receive Delivery Escalation Alerts group.

If delivery remains unacknowledged (by default 72 hours after the form was submitted), escalation emails are sent out to all global and matching organization administrators in the Receive Delivery Escalations group.

To create a delivery channel:

1. Create a delivery channel.



2. Enter or edit the remaining fields to create or update an **Email Secure** delivery channel.
3. Enter one or more email addresses for the email recipient(s) separated by commas in the **To** field.
4. Enter one or more email addresses separated by commas for the email recipient(s) in the **Cc** field (optional).
5. The **Add Receipt PDF** checkbox is already ticked to attach the PDF receipt with the email. Deselect this checkbox if you choose not to include it.
6. The **Add Form XML** checkbox is already ticked to attach the form submission XML file with the email.

Deselect this checkbox if you choose not to include it.

7. Click **Save** to update the changes.

Next, learn how to create a Web Service delivery channel.

Create a Web Service Delivery Channel

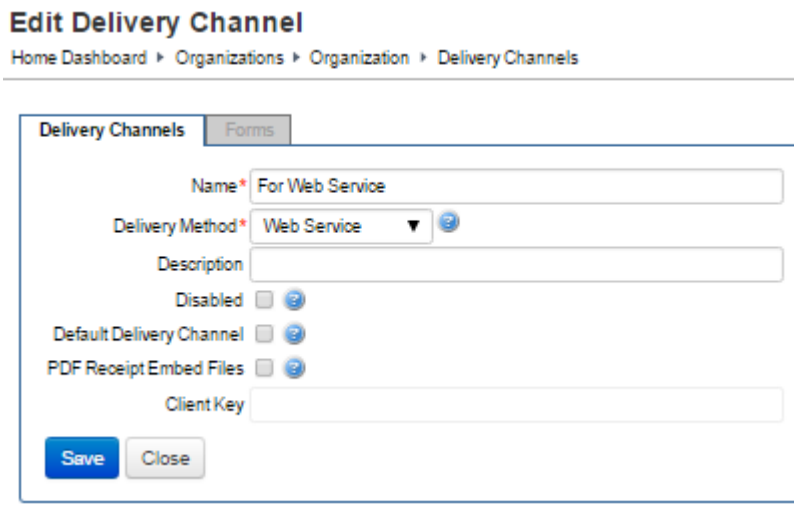
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager comes with the Web Service delivery channel that notifies an organization by making a web service call, using a pre-defined API.

If delivery has not been completed (by default 72 hours after the form was submitted), escalation emails are sent out to all global and organization administrators in the Receive Delivery Escalation Alerts group.

To create a delivery channel Web service:

1. Create a delivery channel.



2. Enter or edit the remaining fields to create or update a **Web Service** delivery channel.
3. Select **PDF Receipt Embed Files** if you wish to embed form XML and attachment files in the PDF receipt.
4. **Client Key** will be displayed once the delivery channel has been saved.
5. Click **Save** to update the changes.

Next, learn how to create a REST delivery channel.

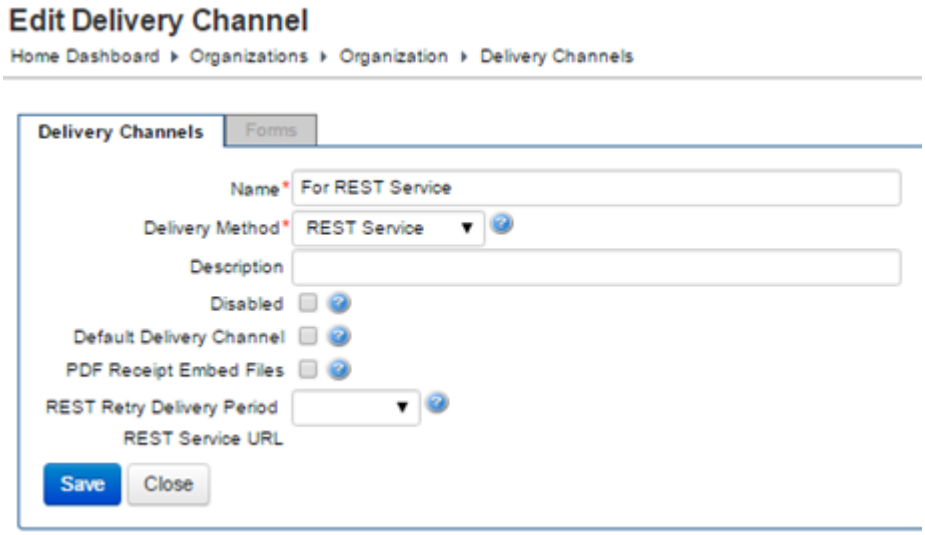
Create a REST Delivery Channel

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager provides a REST endpoint that can be used to retrieve submission data.

To create a delivery channel REST service:

1. Create a delivery channel.



2. Enter or edit the remaining fields to create or update a **REST Service** delivery channel.
3. Select the **PDF Receipt Embed Files** checkbox if you wish to embed form XML and attachment files in the PDF receipt.
4. Select from the **REST Retry Delivery Period** (optional) dropdown if you wish to impose a time limit for submissions that are In-Progress. Once the time limit is reached a submission is reset to delivery status Ready.

> REST Service URL is not entered but will be displayed once the delivery channel has been saved.

5. Click **Save** to update the changes.

Next, learn how to view delivery channels.

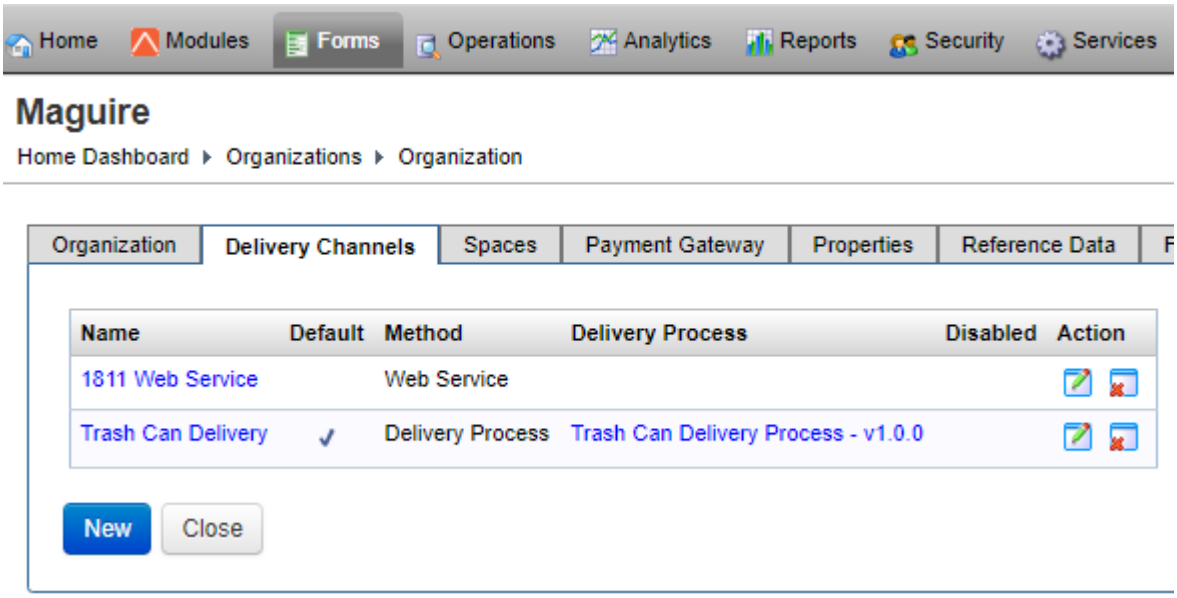
View Delivery Channels

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view the list of available delivery channels for an organization.

To view the delivery channels for an organization:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Delivery Channels** tab to see all delivery channels for this organization.



4. Click **New** to create a delivery channel.
5. Click **Edit** to configure a delivery channel.
6. Click **Remove** to remove a delivery channel when you don't need it any more.

> We recommend disabling a delivery channel, rather than deleting it.

Next, learn how to view submission delivery.

Associate a Delivery Channel With a Form

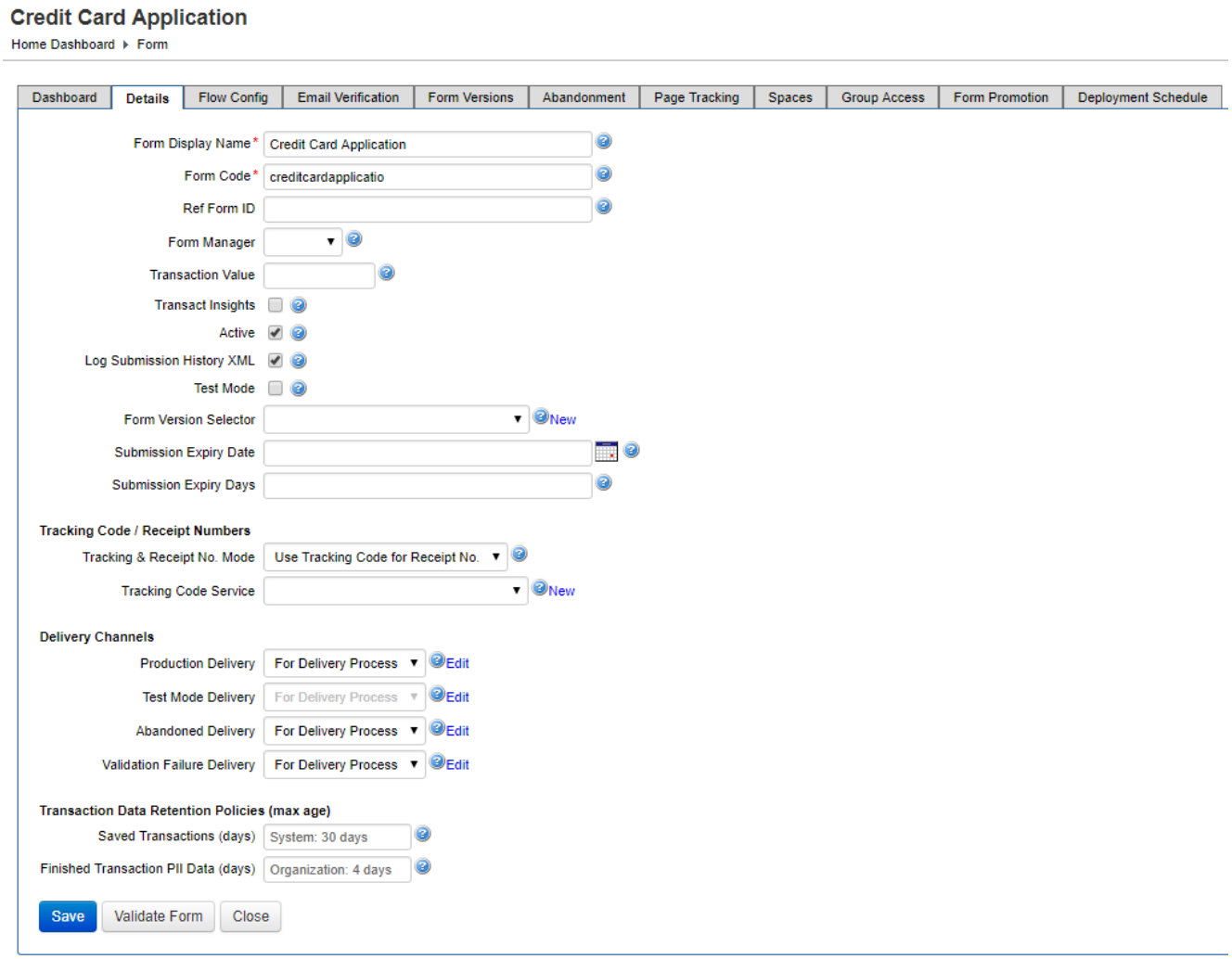
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager allows you to associate a delivery channel with a form, so when a user opens the form one of the following takes place:

- A user successfully completes and submits the form, so Manager delivers the form details to the platform back-end or to a third-party system.
- A user cancels the form, so Manager delivers the abandoned form details to the platform back-end or to a third-party system.
- A user submits the form but a form validation error occurs, so Manager delivers the failed form details to the platform back-end or to a third-party system.

To associate a form with a delivery channel:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select the **Details** tab and configure the **Delivery Channels** section.



4. Select a delivery channel from the **Production Delivery** dropdown list that will be used to deliver successful submissions for this form if the form is not in the test mode.
5. Select a delivery channel from the **Test Mode Delivery** dropdown that will be used to deliver successful submissions for this form if the form is in the test mode.

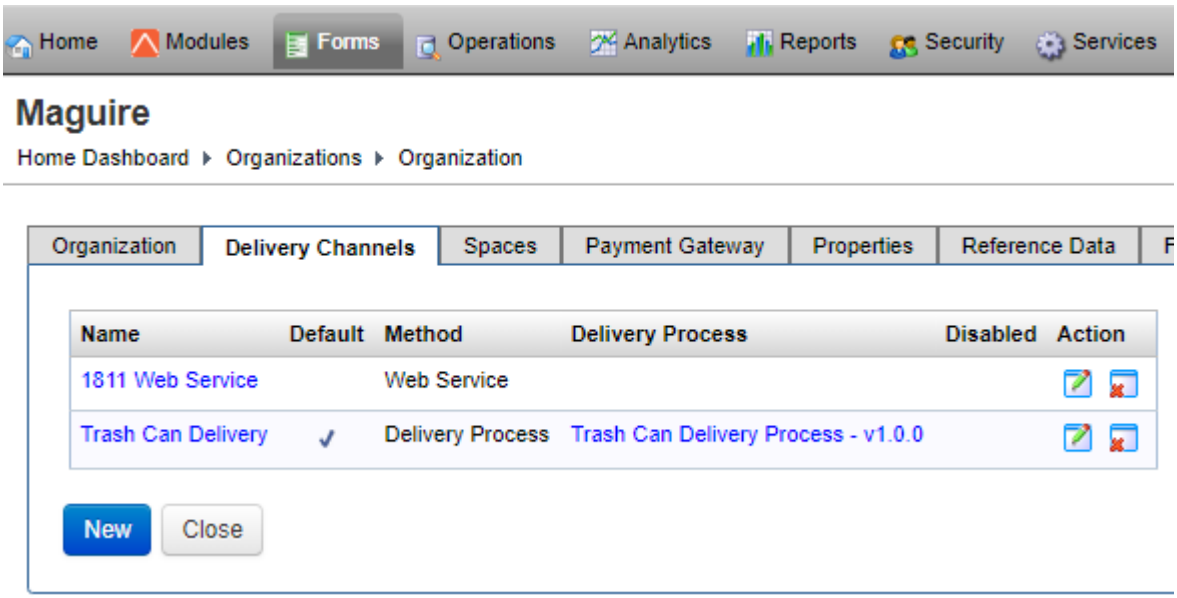
   This dropdown is enabled only if the form is in test mode, that is the **Test Mode** checkbox is selected.

6. Select a delivery channel from the **Abandoned Delivery** dropdown list to deliver abandoned submissions.
7. Select a delivery channel from the **Validation Failure Delivery** dropdown list to deliver submissions with data validation errors.
8. Click **Save** to update the changes.

For more information on other form details, see form configuration.

To view the forms associated with a delivery channel:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Delivery Channels** tab and locate a delivery channel you wish to check.
4. Select the **Forms** tab to view the details.



5. Click **Close**.

Next, learn how to view transactions delivery.

Remove a Delivery Channel

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to remove a delivery channel from an organization when it is not required any more.

> We generally suggest disabling a delivery channel rather than removing it.

> Reassign the form's delivery channel to a valid delivery channel before removing it.

To reassign a delivery channel:

1. Check if any forms are associated with the delivery channel.
2. Reassign each form associated with this delivery channel to a valid delivery channel.

To remove a delivery channel:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Delivery Channels** tab.
4. Locate a delivery channel you wish to remove and click **Remove**.

Next, learn how to create a delivery channel.

View Transactions Delivery

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

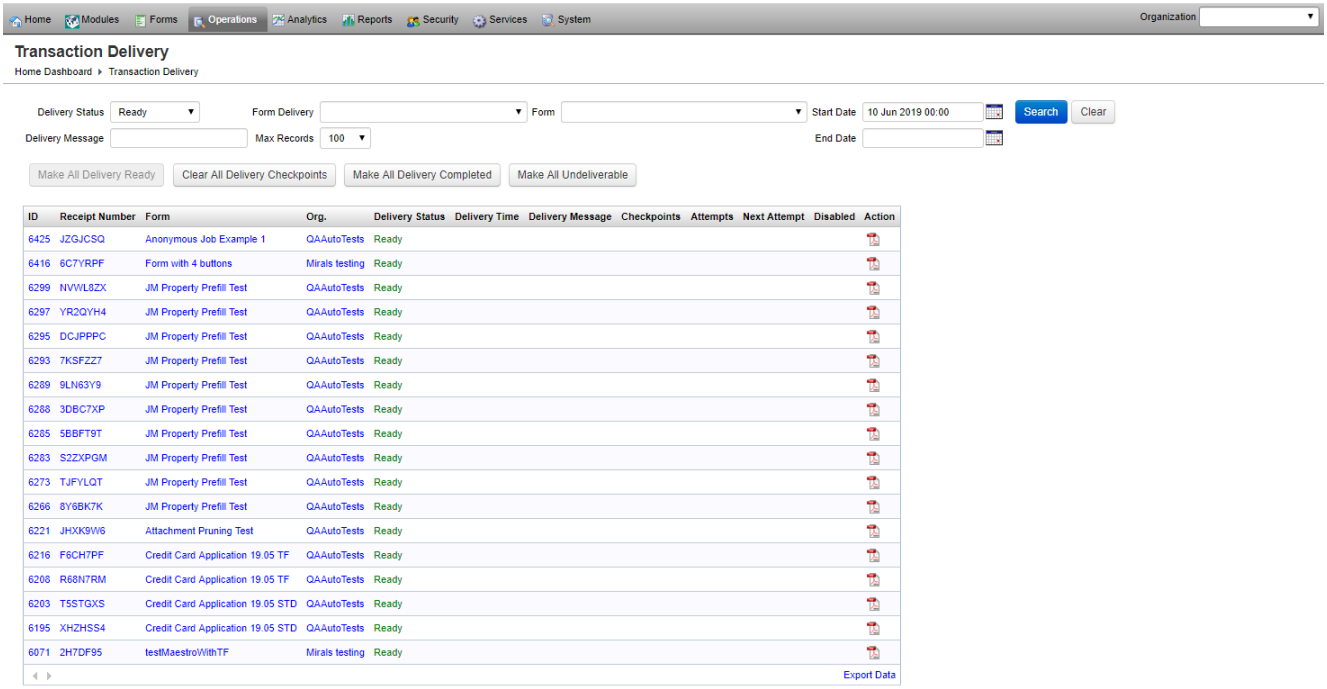
Journey Manager keeps track of each form during its life cycle, from the moment a user opens the form and until the form's data is purged from the system. It allows you to check the following information:

- User interactions with a form
- Delivery of submissions and all relevant notifications
- Form abandonment by users
- Form validation errors

If you are getting Receive Delivery Escalation Alerts email notifications for a particular submission delivery, you can use this functionality to troubleshoot this delivery failure and subsequently fix it or mark it as **Not Required** or **Unavailable**, so the alert will go away.

To view form transactions delivery:

1. Select **Operations > Transaction Delivery**.



2. To filter or search the submissions by one or more criteria, specify the following settings and click **Search**:

- Delivery Status, which can be one of the following: **Ready**, **Send Email**, **In Progress**, **Pending**, **Completed**, **Error**, and **Undeliverable**. Use the status to narrow down on transactions you are interested it. For example, if you are look for a stuck transaction, check transactions that have statuses other then Completed or Undeliverable.
- Delivery Message
- Form Delivery
- Max Records
- Form
- Start Date
- End Date

Click **Clear** to reset the search criteria.

3. You can update the following bulk operations to change the status of all selected submissions:
   - Click **Make All Delivery Ready** to reset delivery status to **Ready** so the delivery will be retried.
   - Click **Clear All Delivery Checkpoints** to clear checkpoints, if the delivery service supports delivery checkpoints.
   - Click **Make All Delivery Completed** to set delivery status to **Completed** without retrying delivery.
   - Click **Make All Undeliverable** to set delivery status to **Undeliverable**, so this delivery will not be reattempted.
4. Click on a transaction receipt number to view the details of each submission. You can change the delivery status for the individual submission, if required.
5. Click **View Receipt** to view and download a transaction receipt as a PDF file.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

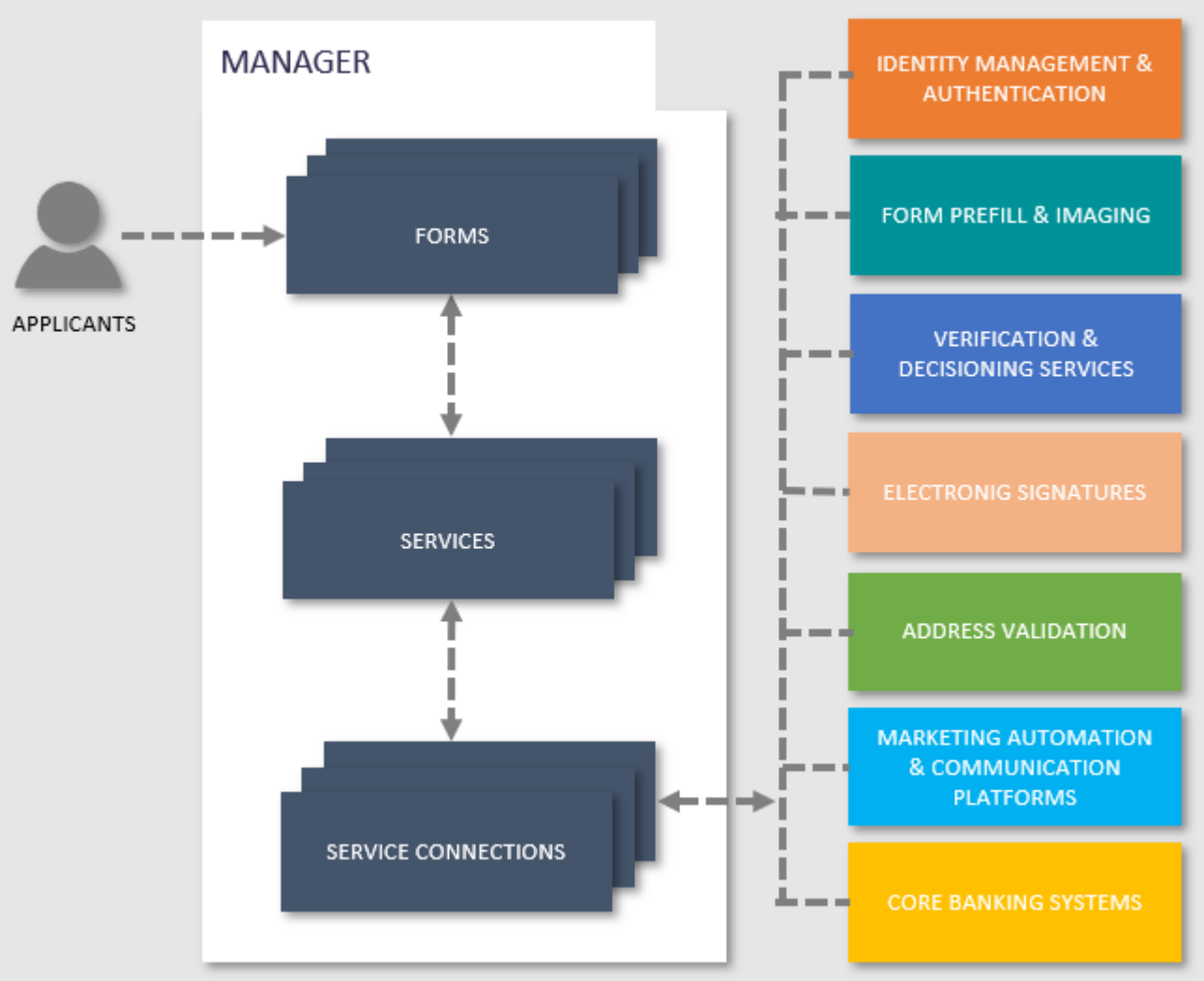
Next, learn how to create a delivery channel.

Service Connection Overview

Journey Manager (JM) | System Manager / DevOps | Platform Developer | 5.1 & Higher

Journey Manager comes with services that require connections to external servers to perform their functionality. These connection details are managed separately to services using service connections. This allows you to reuse service connections in different services and change their configurations without modifying services.

Manager includes pre-built connectors for best in class services such as fraud detection, ID verification, digital signatures, and over 30 different connection partners. The integration has already been done, so each connector can be added to the user experience as needed when designing the application.



Manager uses service connection types to define service connections functionality, so you can create your own service connection based on a selected service connection type. You can define a service connection that is:

- Global - it doesn't belong to an organization and thus can be used by all services.
- Organization specific - it belongs to a single organization and can be used only by services that belong to this organization.

You can also remove, import and export service connections.

Next, learn how to view all service connections.

Service Connection Types

Journey Manager (JM) | System Manager / DevOps | 20.05

Manager uses service connection types to define and group service connections according to their functionality. A service connection type can:

- Act as a template by providing the applicable set of parameters for each service connection type.
- Determine how it is used within Manager.

Manager has the following predefined service connection types:

Service Connection Type	Description	Used in
AWS KMS	Data storage.	
AWS S3	Data storage.	
AWS SQS	External publish queue.	AWS SQS system event publisher service
AWS STS	External publish queue.	
Azure Storage	External publish queue.	Azure submission data storage service Azure system event publisher service
DocuSign	DocuSign service.	
HTTP Endpoint	Generic HTTP service.	
SalesForce	SalesForce CRM.	
Integration Gateway	TIG service.	

You can create new service connections based on the existing types.

Next, learn how to view service connections.

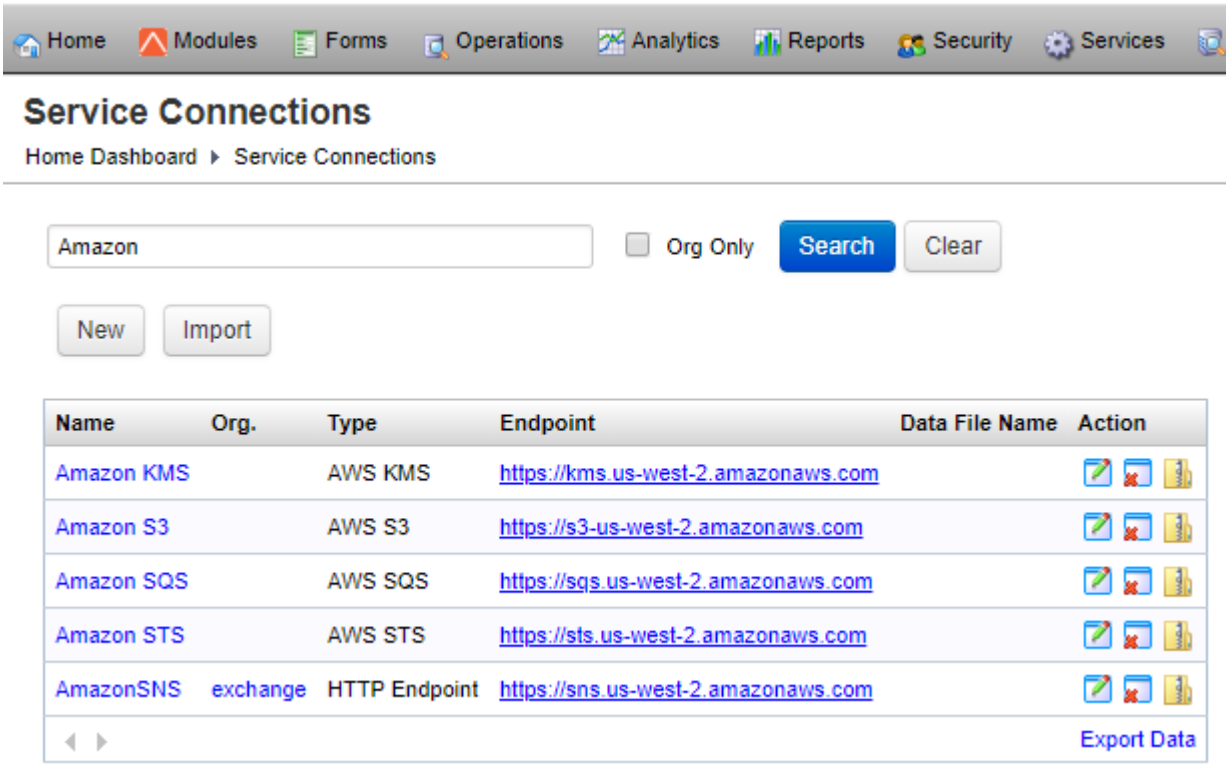
View Service Connections

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view all service connections, so you can easily find an existing service connection and use it in your service configuration.

To view service connections, select **Services > Service Connections.**

Service connections are shown page by page, with the maximum number of records per page configured via the user's preferences.



To filter or search the services by one or more criteria, specify the following settings and click **Search**:

- Service connection name or part of a name, or the **endpoint**[1] URL.
- Select the **Org Only** checkbox to show service belonging to an organization. This will exclude global service connections, which are available to all services.

Click **Clear** to reset the search criteria.

You can also perform the following actions:

[1]Endpoint is a web address (URL) at which clients of a specific service can gain access to it. By referencing that URL, clients can get to operations provided by that service.

- Create a service connection.
- Configure a service connection.
- Remove a service connection.
- Export service connections.
- Import service connections.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

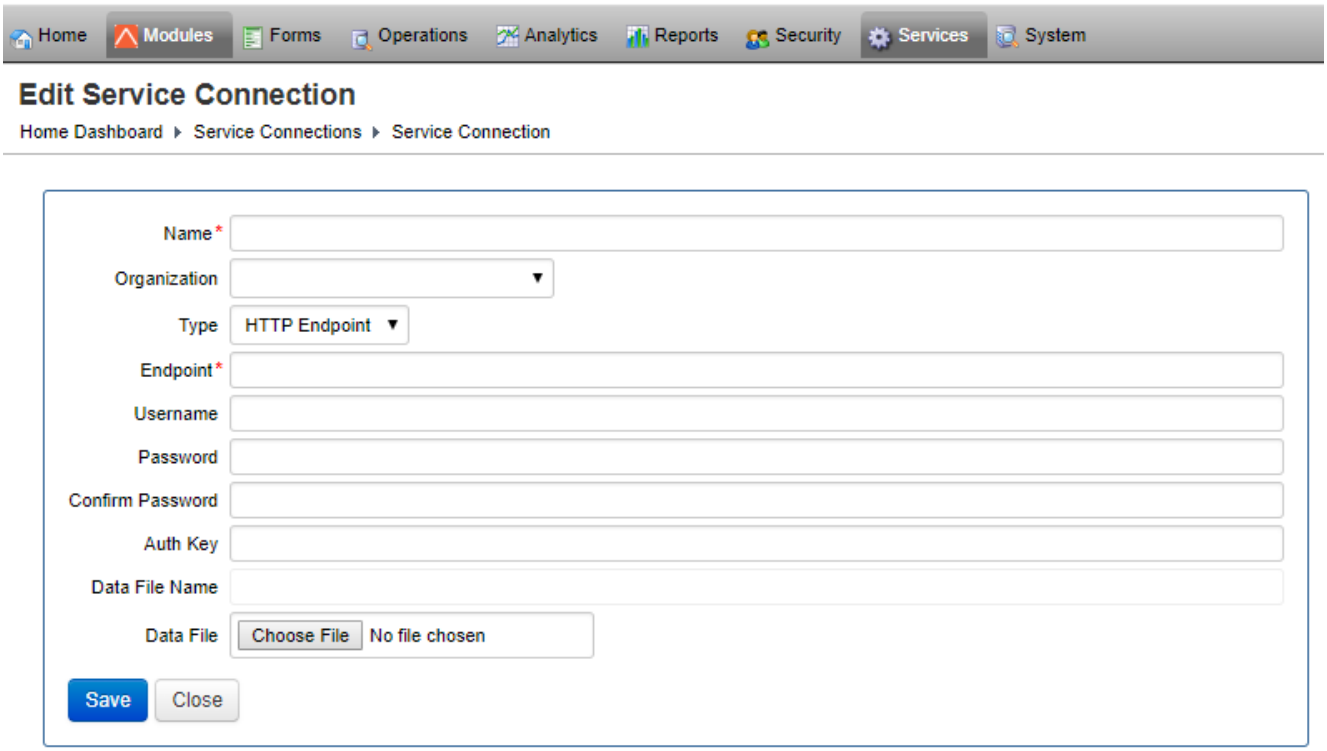
Create a Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create new service connections, which services can use to implement a client's requirements.

To create a new service connection:

1. Select **Services > Service Connections**.
2. Click **New**.



3. Provide a unique service connection name in the **Name** field.
4. Select an organization from the **Organization** dropdown list to restrict this service connection to a single organization. Leave is blank to create a global service connection that can be used by all services.
5. Select a service connection type from the **Type** dropdown list.
6. Enter the **Endpoint** [1] as a URL to an external service.

> The username and password fields should not contain certain HTML characters, such as > or <. If they do, you may find the service throws an exception when it's called. If this happens, check your logs for further

[1] Endpoint is a web address (URL) at which clients of a specific service can gain access to it. By referencing that URL, clients can get to operations provided by that service.

details.

7. Configure connection specific options.
8. Click **Save** to update the changes.

Next, learn how to remove a service connection.

Configure a Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create new and configure existing service connections, which services can use to implement client's requirements.

To configure a service connection:

1. Select **Services > Service Connections**.
2. Locate a service connection and click **Edit**.
3. Update a service connection name in the **Name** field.
4. Select an organization from the **Organization** dropdown list to restrict this service connection to a single organization. Leave is blank to make it a global service connection that can be used by all services.
5. Select a service connection type from the **Type** dropdown list.
6. Enter the **Endpoint** [1] as a URL to an external service.

> Service connection configuration options are different for each service connection type. Refer to a specific service connection type configuration for more details.

7. Click **Save** to update the changes.

Next, learn how to remove a service connection.

[1] Endpoint is a web address (URL) at which clients of a specific service can gain access to it. By referencing that URL, clients can get to operations provided by that service.

Remove a Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to remove a service connection if it's no longer required.

> Manager irrevocably removes all data associated with the service connection. This action cannot be rolled back.

To remove a service connection:

1. Select **Services > Service Connections**.
2. Locate the service connection.
3. Update all services using this service connection so they are not left without a valid service connection. Otherwise, they will most likely fail.
4. Click **Remove** and then click **OK** to confirm.

Next, learn how to view service connections.

Migrate Service Connections

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides a mechanism to migrate service connections from one environment to another. This allows you to develop and test a service connection in different environments before migrating it over to the production environment.

To migrate one or several service connections:

1. Export service connections
2. Import service connections

> You can use the same process to perform a service connection backup and rollback.

Next, learn how to view service connections.

Export Service Connections

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows administrators to export service connections. You can use this functionality to:

- Back-up or archive service connections
- Re-create or migrate service connections in another Manager
- Promote service connections to another environment

To export a service connection:

1. Select **Services > Service Connections.**
2. Locate a service connection and click **Export Archive** to export a service connection.

Manager exports it into a service connection archive ZIP file, which will be downloaded to your PC.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to import service connections.

Import Service Connections

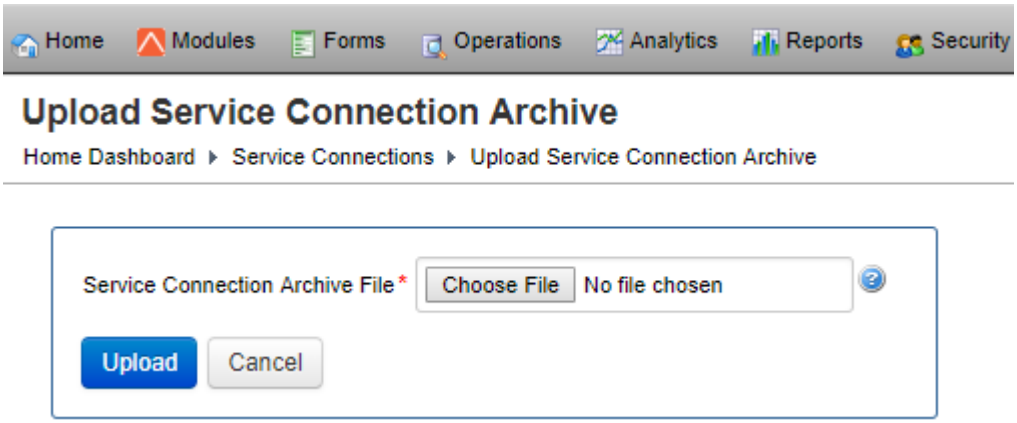
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to import service connections. Use this functionality to:

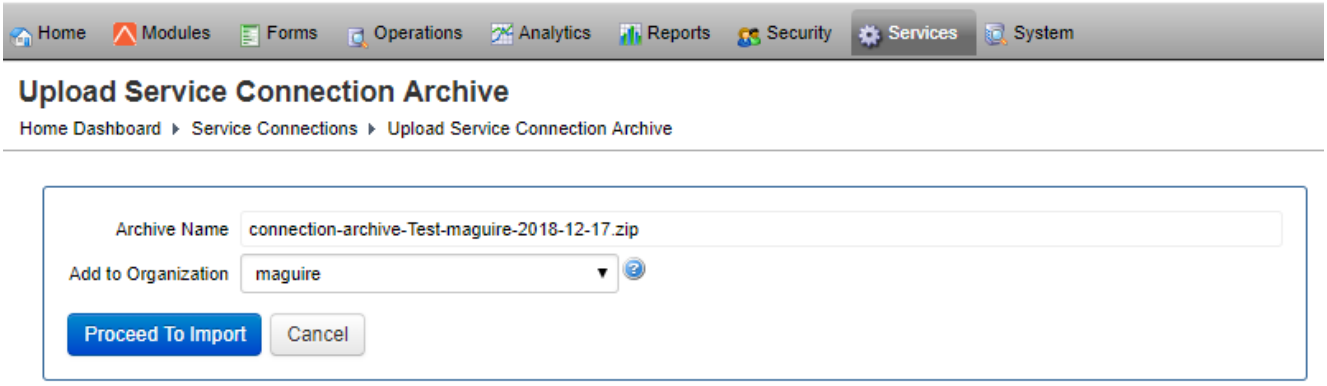
- Roll-back service connections
- Re-create service connections in another instance of Manager
- Promote service connections to another environment

To import a service connections:

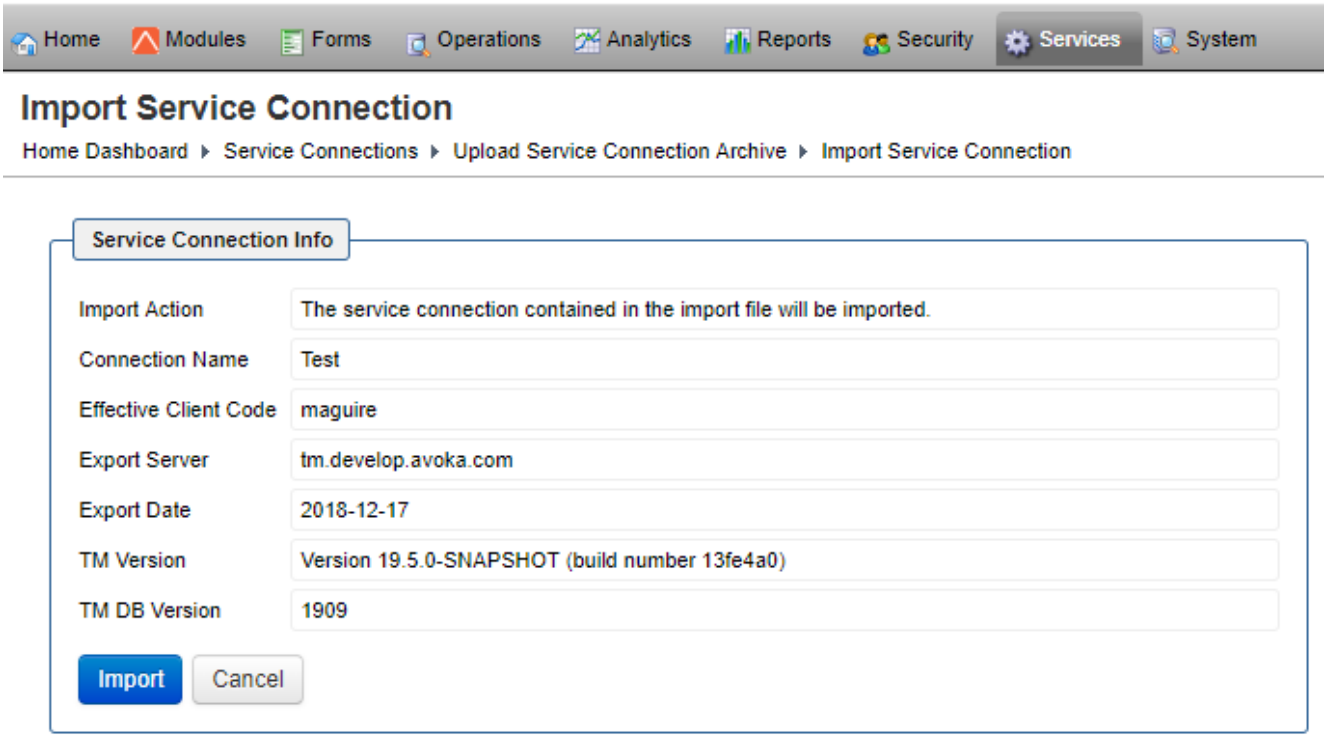
1. Select **Services > Service Connections.**
2. Click **Import.**



3. Click **Choose File** and choose a service connection archive ZIP file, which has been previously exported.
4. Click **Upload.**



5. Manager reads the organization that the service connection belongs to from the export archive, if any. You can choose to instead restrict the service connection to an organization by selecting it from the **Add to Organization** dropdown list.
6. Click **Proceed To import**.
7. Review the **Import Options** as shown below.



8. Click **Import** to complete the process. Check the **Detail Messages** tab for any warnings logged during import.
9. Click **Close** and check the list of service connections to verify the imported service connection have been added.

Next, learn about service connection types.

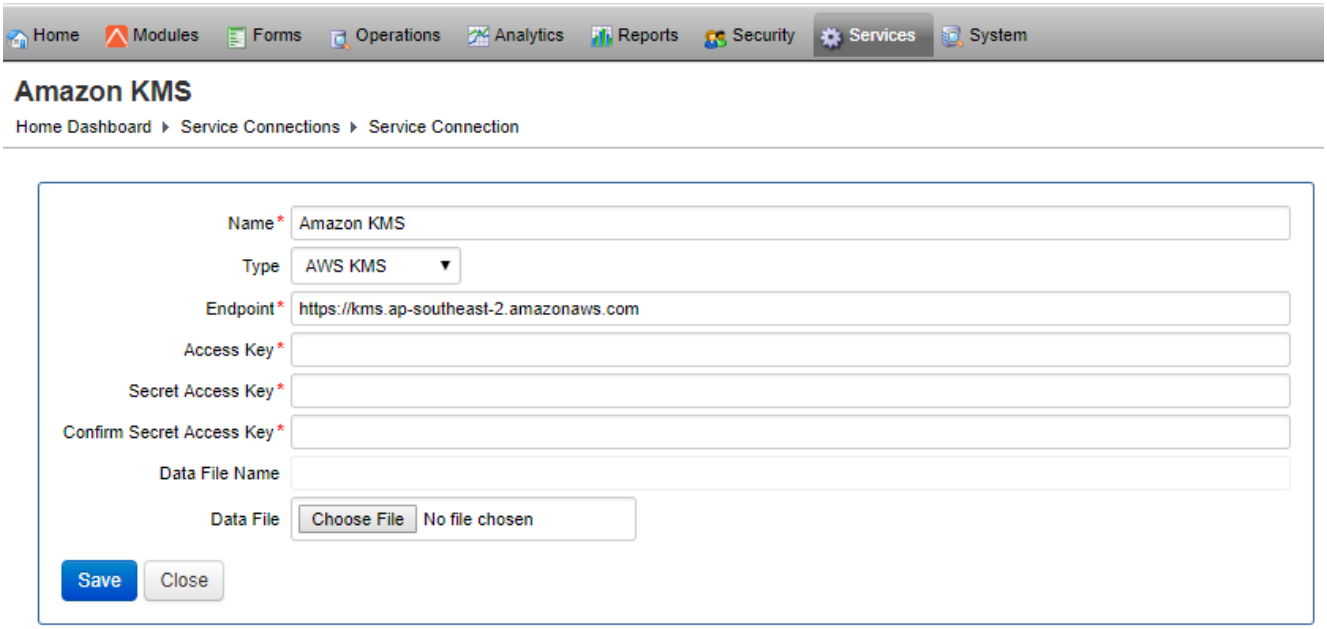
AWS KMS Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure a **AWS KMS**[1] service connection to use it in a service that requires data storage.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate an Amazon KMS service connection or click **New** to create a new service connection.



3. Provide a unique service connection name in the **Name** field.
4. Select the AWS KMS type from the **Type** dropdown list.
5. Enter the **Endpoint** as a URL for the AWS KMS queue service.
6. Provide the **Access Key**, which is an AWS access key ID. You use access keys to sign programmatic requests that you make to AWS if you use AWS CLI commands or using AWS API operations. You must use both the access key and secret access key together to authenticate your requests. For more information, see https://-docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html.
7. Provide the **Secret Access Key**.
8. Confirm the secret access key in the **Confirm Secret Access Key** field.

---

[1] AWS Key Management Service (KMS) is a managed service that makes it easy for you to create and control the encryption keys used to encrypt your data, and uses FIPS 140-2 validated hardware security modules to protect the security of your keys. AWS Key Management Service is integrated with most other AWS services to help you protect the data you store with these services. AWS Key Management Service is also integrated with AWS CloudTrail to provide you with logs of all key usage to help meet your regulatory and compliance needs.

9. Enter the **Data File Name**.
10. Click **Data File** to upload a data file, if it's required for this service connection.
11. Click **Save** to update the changes.

Next, learn about other service connection types.

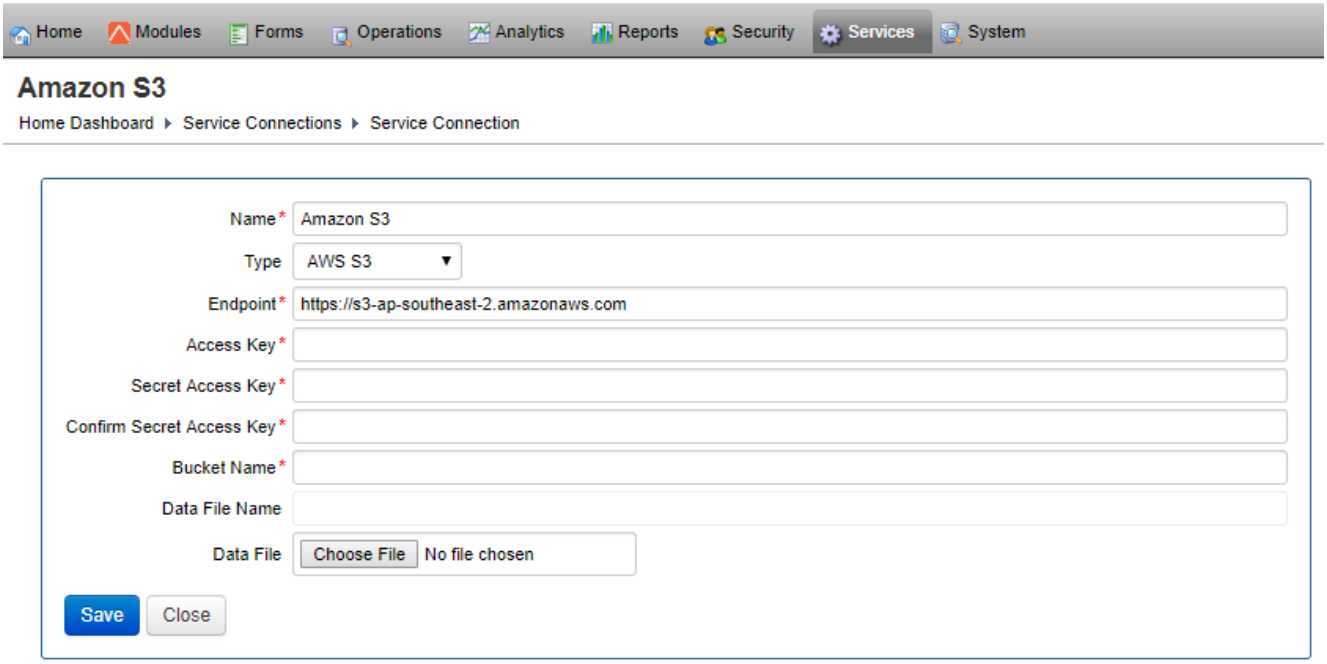
AWS S3 Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure a AWS S3 service connection to use it in a service that requires data storage.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate an **Amazon S3** service connection or click **New** to create a new service connection.



3. Provide a unique service connection name in the **Name** field.
4. Select the AWS S3 type from the **Type** dropdown list.
5. Enter the **Endpoint** as a URL for the AWS S3 service.
6. Provide the **Access Key**, which is an AWS access key ID. You use access keys to sign programmatic requests that you make to AWS if you use AWS CLI commands or using AWS API operations.You must use both the access key and secret access key together to authenticate your requests. For more information, see https://-docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html.
7. Provide the **Secret Key**.
8. Confirm the Secret Key in the **Confirm Secret Key** field.
9. Enter the **Bucket Name**.
10. Enter the **Data File Name**.
11. Click **Data File** to upload a data file, if it's required for this service connection.
12. Click **Save** to update the changes.

Next, learn about other service connection types.

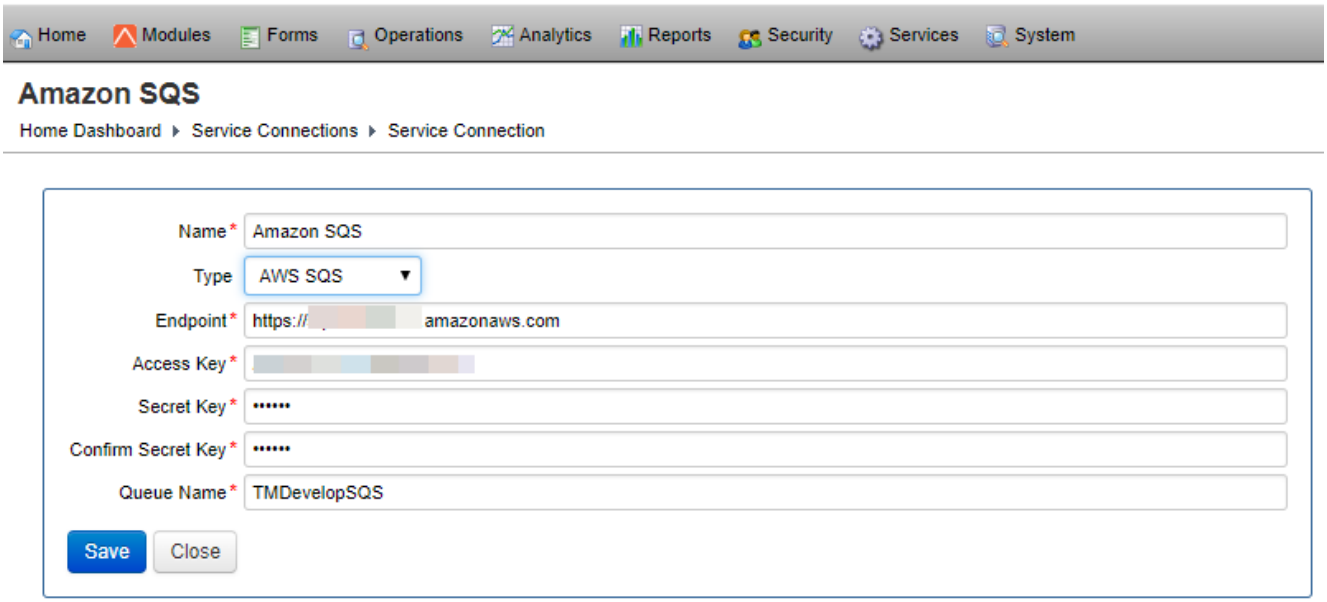
AWS SQS Service Connection

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager allows you to create and configure a service connection to an **AWS SQS**[1] queue, which is used in the AWS SQS System Event Publisher service to publish events to that queue.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate an Amazon SQS service connection or click **New** to create a new service connection.



3. Provide a unique service connection name in the **Name** field.
4. Select the AWS SQS **type** from the **Type** dropdown list.
5. Enter the **Endpoint** as a URL for the AWS SQS queue service.
6. Provide the **Access Key**, which is an AWS access key ID. You use access keys to sign programmatic requests that you make to AWS if you use AWS CLI commands or using AWS API operations. You must use both the access key and secret access key together to authenticate your requests. For more information, see Amazon documentation.
7. Provide the **Secret Key**.
8. Confirm the Secret Key in the **Confirm Secret Key** field.
9. Enter the **Queue Name**.
10. Click **Save** to update the changes.

---

[1] Amazon Simple Queue Service (Amazon SQS) is a pay-per-use web service for storing messages in transit between computers. Developers use SQS to build distributed applications with decoupled components without having to deal with the overhead of creating and maintaining message queues.

Next, learn about other service connection types.

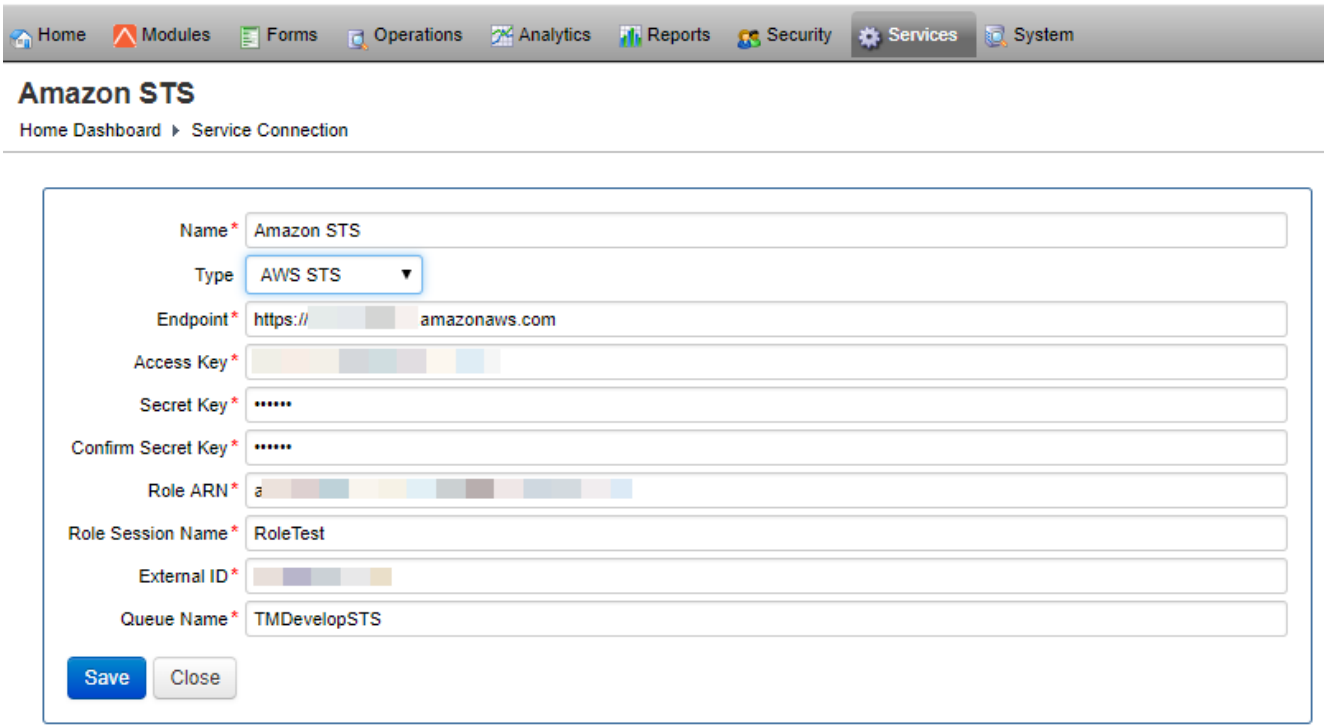
AWS STS Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure a **AWS STS**[1] service connection to use in the AWS SQS System Event Publisher service to publish events to the correct queues using the AWS STS protocol.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate an Amazon STS service connection or click **New** to create a new service connection.



3. Provide a unique service connection name in the **Name** field.
4. Select the AWS STS type from the **Type** dropdown list.
5. Enter the **Endpoint** as a URL for the AWS STS queue service.
6. Provide the **Access Key**, which is an AWS access key ID. You use access keys to sign programmatic requests that you make to AWS if you use AWS CLI commands or using AWS API operations. You must use both the access key and secret access key together to authenticate your requests. For more information, see https://-docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html.
7. Provide the **Secret Key**.
8. Confirm the Secret Key in the **Confirm Secret Key** field.

---

[1] The AWS Security Token Service (STS) is a web service that enables you to request temporary, limited-privilege credentials for AWS Identity and Access Management (IAM) users or for users that you authenticate (federated users).

9. Provide the **Role ARN**[1]. The Amazon Resource Name (ARN) of the role to assume..
10. Enter the **Role Session Name**, which is an identifier for the assumed role session. Use the role session name to uniquely identify a session when the same role is assumed by different principals or for different reasons. In cross-account scenarios, the role session name is visible to, and can be logged by the account that owns the role.
11. Provide the **External ID**. A unique identifier that might be required when you assume a role in another account. If the administrator of the account to which the role belongs provided you with an external ID, then provide that value in the ExternalId parameter. This value can be any string, such as a passphrase or account number. Because a cross-account role is usually set up to trust everyone in an account, the administrator of the trusting account might send an external ID to the administrator of the trusted account. For more information, see https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-user_externalid.html.
12. Enter the **Queue Name**.
13. Click **Save** to update the changes.

Next, learn about other service connection types.

[1]An Amazon Resource Name (ARN) is a file naming convention used to identify a particular resource in the Amazon Web Services (AWS) public cloud. ARNs, which are specific to AWS, help an administrator track and use AWS items and policies across AWS products and API calls.

Azure Service Connection

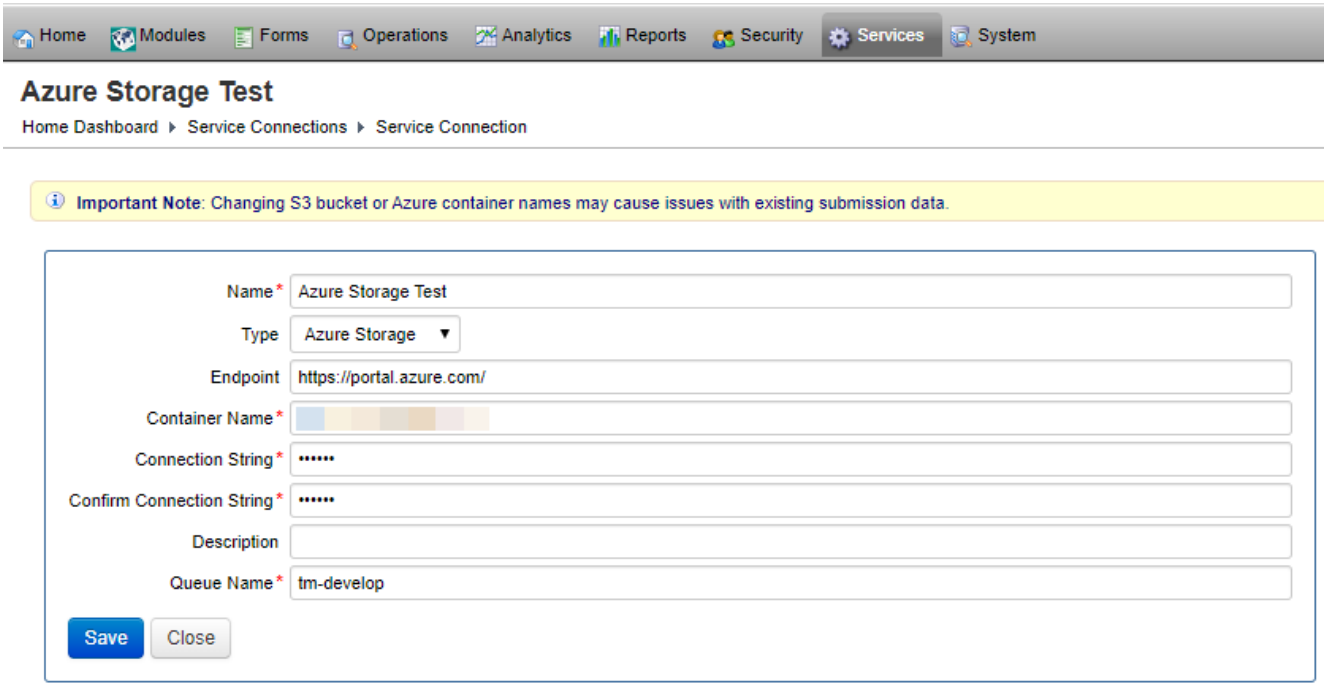
Journey Manager (JM) | System Manager / DevOps | 20.05

Manager allows you to create and configure a service connection to an **Azure**[1] Storage, which is used by various services, such as the Azure service connection service and Azure submission data storage service.

> Changing Azure container names may cause issues with existing submission data.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate an **Azure Storage** service connection or click **New** to create a new service connection.



3. Provide a unique service connection name in the **Name** field.
4. Select the **Azure Storage type** from the **Type** dropdown list.
5. Enter a URL to the Azure storage in the **Endpoint** field.
6. Provide a connection string for Azure storage in the **Connection String** field. For more information, see Microsoft Azure documentation.
7. Provide a container name in the **Container Name** field, which is used for Azure Blob. Both the Azure Blob and Azure Queue are the part of Azure Storage so they use the same connection.
8. Enter a description of this service connection in the **Description** field.

---

[1]Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers.

9. Enter a default queue name in the **Queue Name** field, which is used as a fallback if an Azure service doesn't explicitly specify queue names in the **Audit Log Queue**, **Job Queue**, **Submission Queue**, **System Health Queue**, **User Auth Queue**.
10. Click **Save** to update the changes.

Next, learn about other service connection types.

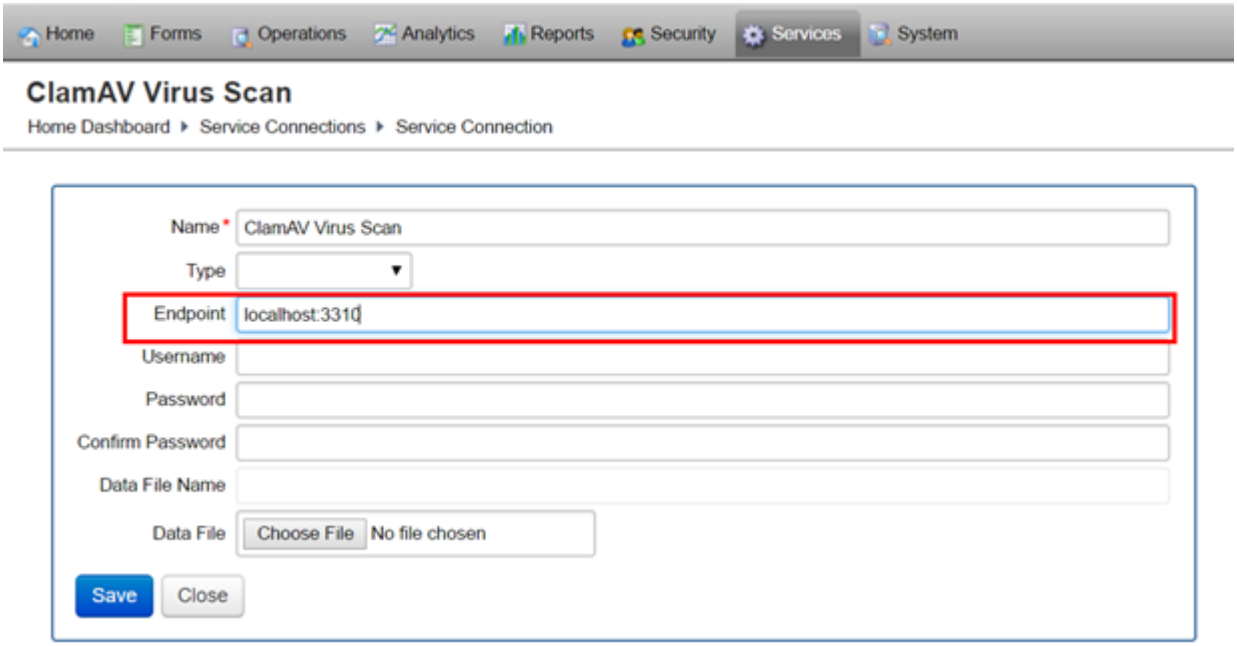
ClamAV Virus Scan Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure a ClamAV service connection to the Clam AV engine.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate a ClamAV Virus Scan service connection or click **New** to create a ClamAV Virus Scan service connection.



3. Provide a unique service connection name in the **Name** field, for example ClamAV Virus Scan.
4. Update the **Endpoint** with the correct server name and port number in the format **`hostName:portNumber`**, if the ClamAV Engine configuration does not use the default server name or port: **`localhost:3310`**.
5. Click **Save** to update the changes.

Next, learn how to configure the attachment virus scan service.

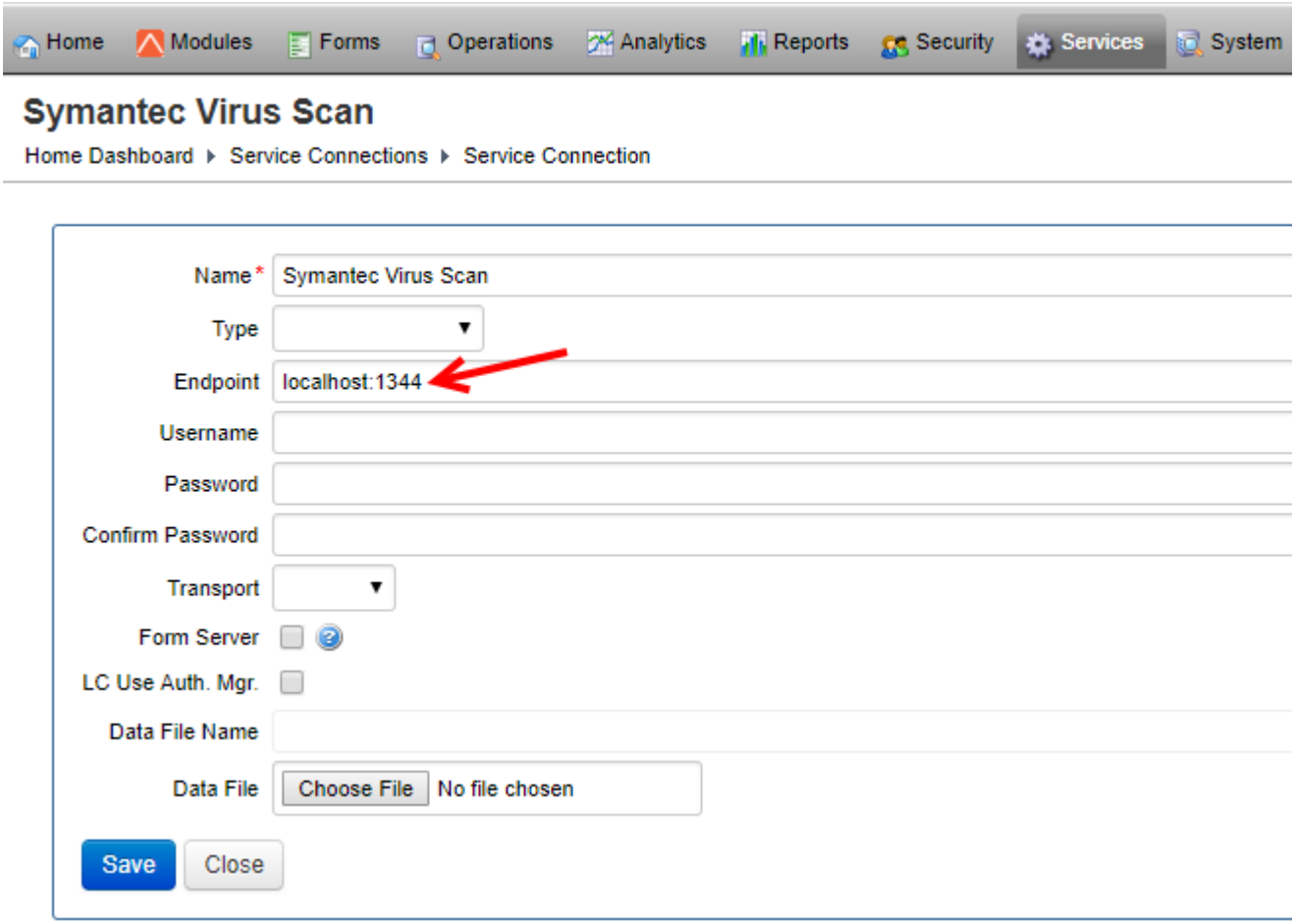
Symantec Virus Scan Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure a Symantec Virus service connection to the Symantec Virus Scan engine.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate a Symantec Virus Scan service connection or click **New** to create a Symantec Virus Scan service connection.



3. Provide a unique service connection name in the **Name** field, for example Symantec Virus Scan.
4. Update the **Endpoint** with the correct server name and port number in the format **`hostName:portNumber`** if the Symantec Protection Engine configuration does not use the default server name or port (**`localhost:1344`**).
5. Click **Save** to update the changes.

Next, learn how to configure attachment virus scan service.

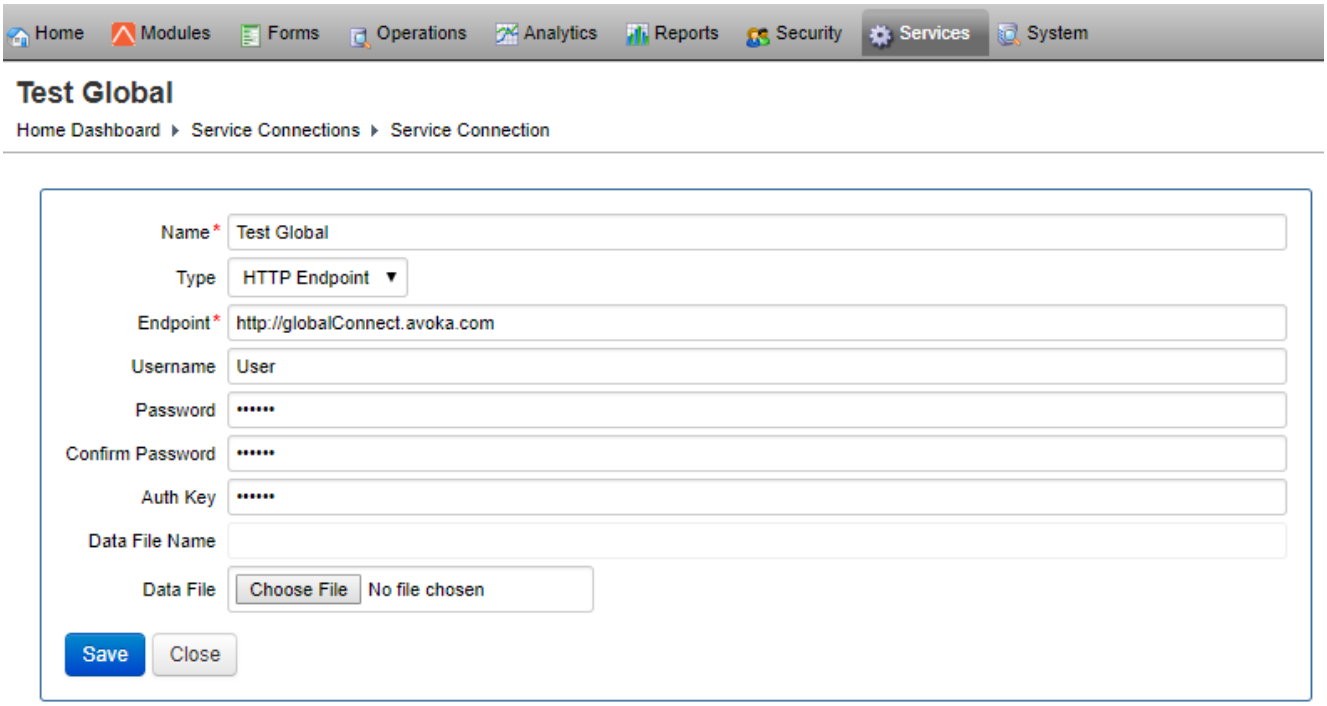
HTTP Endpoint Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure an HTTP **Endpoint**[1] service connection to an external service.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate an HTTP Endpoint service connection or click **New** to create a new service connection.



3. Provide a unique service connection name in the **Name** field.
4. Select the **HTTP**[2] Endpoint type from the **Type** dropdown list.
5. Enter the **Endpoint** as a URL for the external service.
6. Enter the **Username**.
7. Enter the **Password**.
8. Confirm the password in the **Confirm Password** field.
9. Enter the **Auth Key**.
10. Provide the **Data File Name**.

---

[1]Endpoint is a web address (URL) at which clients of a specific service can gain access to it. By referencing that URL, clients can get to operations provided by that service.

[2]HTTP (HyperText Transfer Protocol) is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

11. Click **Data File** to upload a data file, if it's required for this service connection.
12. Click **Save** to update the changes.

Next, learn about other service connection types.

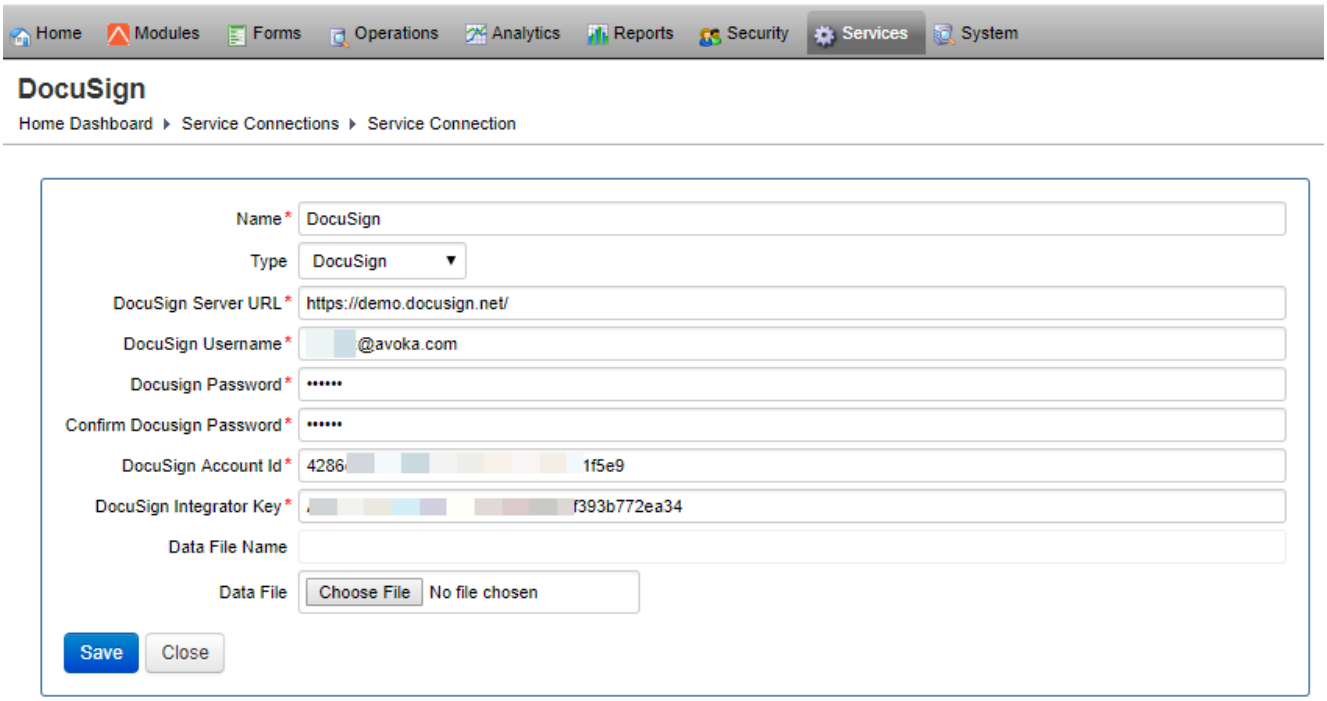
DocuSign Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure a **DocuSign**[1] service connection to use it in a service that uses electronic exchanges of contracts and signed documents.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate a DocuSign service connection or click **New** to create a new service connection.



3. Provide a unique service connection name in the **Name** field.
4. Select the DocuSign type from the **Type** dropdown list.
5. Enter the **DocuSign Server URL** as a URL to the DocuSign service.
6. Provide the **DocuSign Username** to log in on to the service.
7. Provide the **DocuSign Password**.
8. Confirm the password in the **Confirm DocuSign Password** field.
9. Enter the **DocuSign Account Id**, which identifies a DocuSign account.

[1]DocuSign is the leader in eSignature transaction management. The Journey platform uses this solution to speed the rate at which all digital documents are completed, signed, and returned.

10. Provide the **DocuSign Integrator Key**, also known as an API key, is a unique **GUID**[1] used to identify your API integration. An integrator key is required to authenticate your API calls with DocuSign..
11. Enter the **Data File Name**.
12. Click **Data File** to upload a data file, if it's required for this service connection.
13. Click **Save** to update the changes.

Next, learn about other service connection types.

[1]A GUID (global unique identifier) is a term for a number that its programming generates to create a unique identity for an entity such as a Word document. GUIDs are widely used in Microsoft products to identify interfaces, replica sets, records, and other objects.

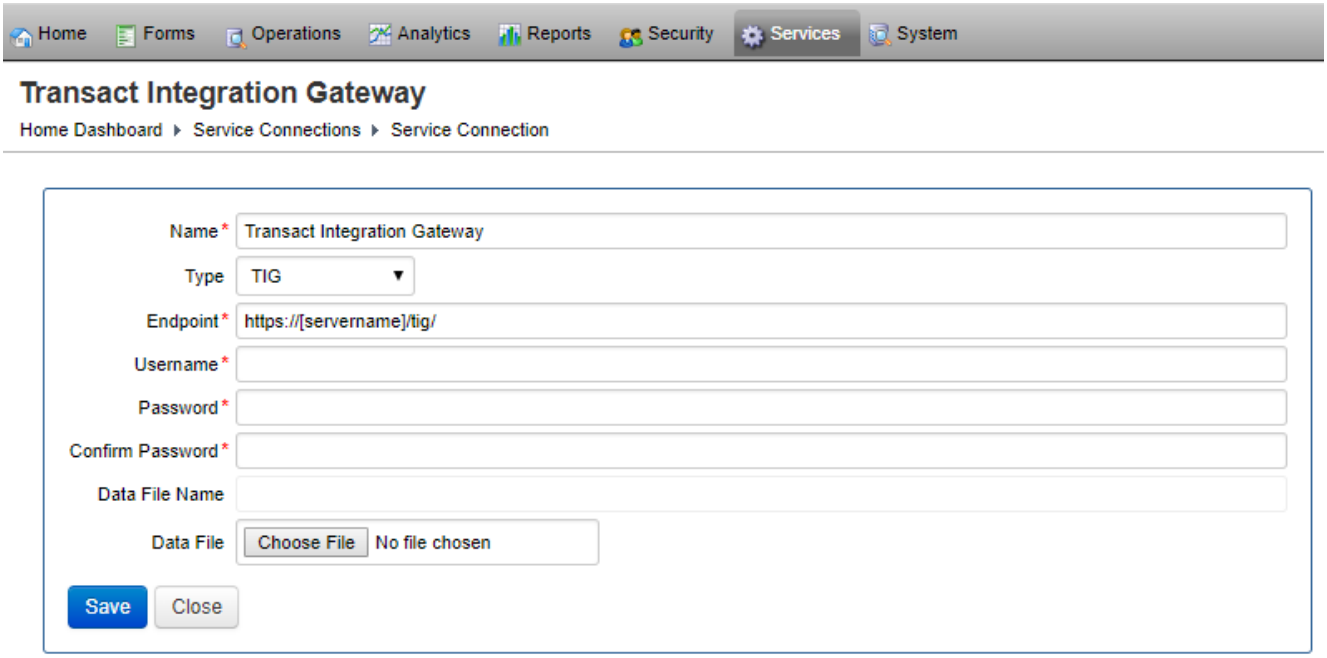
Integration Gateway Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager allows you to create and configure a Transact Integration Gateway (TIG) service connection to Transact Integration Gateway.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate a Transact Integration Gateway service connection or click **New** to create a new service connection.



3. Provide a unique service connection name in the **Name** field.
4. Select the TIG type from the **Type** dropdown list.
5. Enter the **Endpoint** as a URL of the TIG server you want to connect to.
6. Enter the **Username**.
7. Enter the **Password**.
8. Confirm the password in the **Confirm Password** field.
9. Provide the **Data File Name**.
10. Click **Data File** to upload a data file, if it's required for this service connection.
11. Click **Save** to update the changes.

Next, learn about other service connection types.

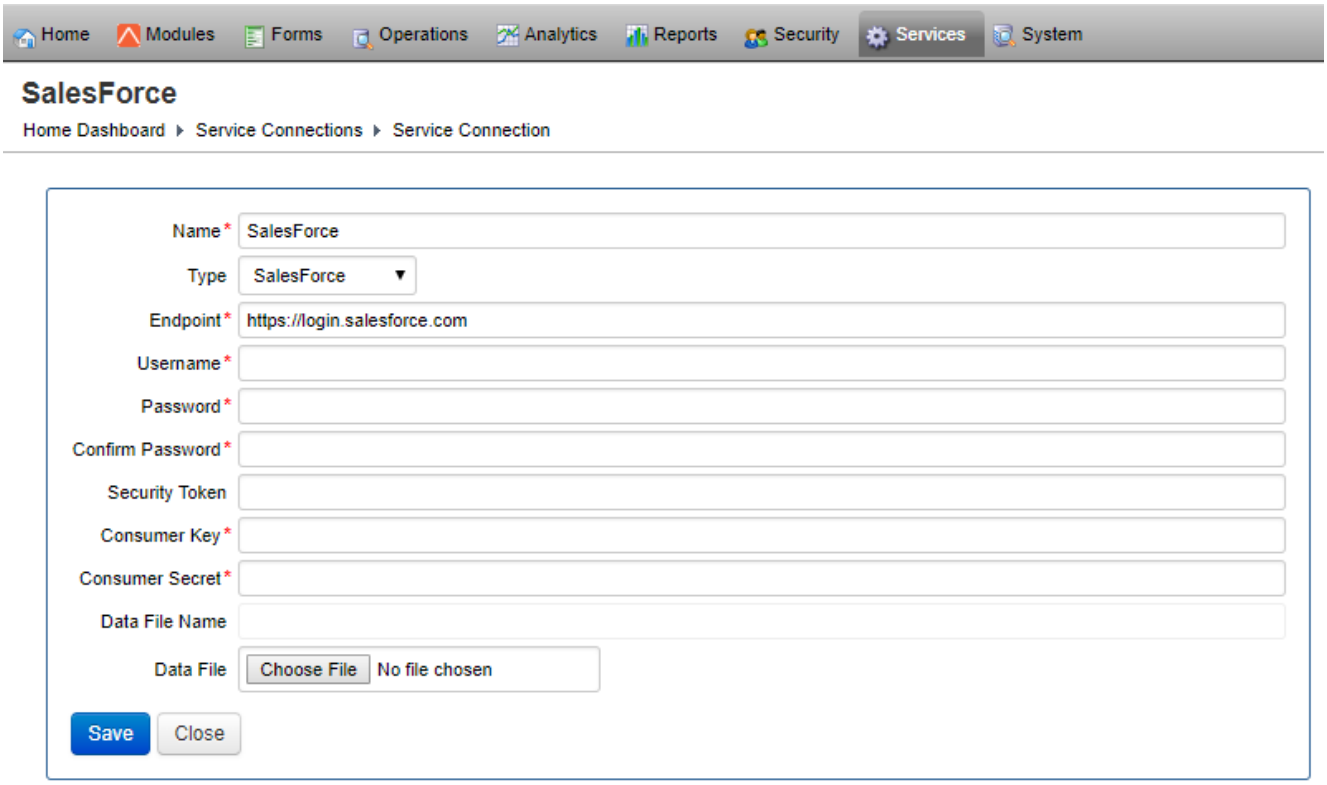
Salesforce Service Connection

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure a **SalesForce**[1] service connection to an external Salesforce service.

To configure a service connection:

1. Select **Services > Service Connections.**
2. Locate a SalesForce service connection or click **New** to create a new service connection.



3. Provide a unique service connection name in the **Name** field.
4. Select the **HTTP**[2] Endpoint type from the **Type** dropdown list.
5. Enter the **Endpoint** as a URL for the external service.
6. Enter the **Username**.
7. Enter the **Password**.
8. Confirm the password in the **Confirm Password** field.
9. Enter the **Security Token**.

---

[1]Salesforce is the customer relationship management (CRM) platform.

[2]HTTP (HyperText Transfer Protocol) is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

10. Provide the **Consumer Key**.
11. Enter the **Consumer Secret**.
12. Specify the **Data File Name**.
13. Click **Data File** to upload a data file, if it's required for this service connection.
14. Click **Save** to update the changes.

Next, learn about other service connection types.

Services Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager comes with services that are software modules configured to perform specific business-related functionality, usually by executing **Groovy**[1] scripts. Services are often used to call external services hosted on premise or on cloud. Manager supports the **Amazon**[2] or **Azure**[3] clouds.

You can use services to implement the typical client onboarding solutions, such as:

- Data prefill for existing customers or using photo ID
- Address look-up and validation
- Identity verification, **KYC**[4], **OFAC**[5]
- Deposit and credit decisions
- Save and resume
- Follow-up leads.
- eSign documents
- Channel cross-over
- Core system integration
- Cross-sell next product

The example of a customer acquisition, onboarding and origination process using Manager's services is shown below:

---

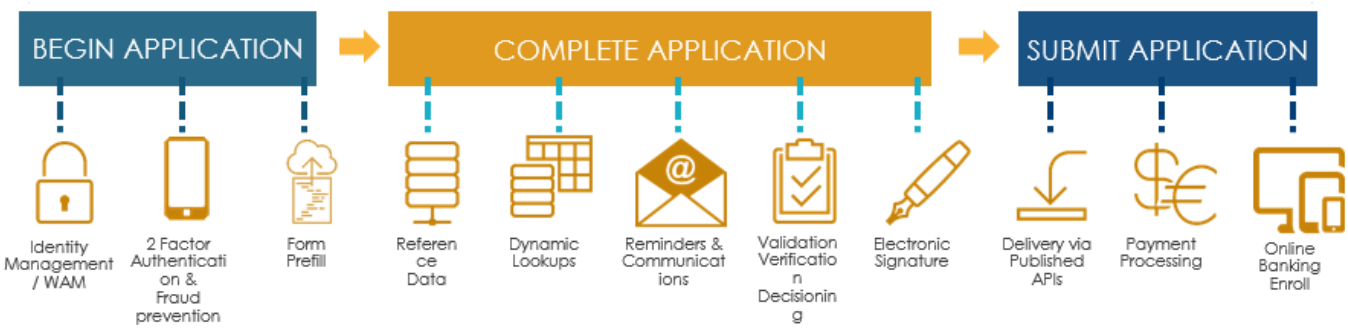
[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

[2]Amazon Web Services (AWS) is a comprehensive, evolving cloud computing platform provided by Amazon. It provides a mix of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings.

[3]Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers.

[4]Know your customer (KYC) requires that banks have a reasonable belief that they know the true identity of customers and establish reasonable and practical risk-based procedures for verifying the identity of each customer. The procedures must include account opening procedures that specify the identifying info that will be obtained from each customer, such as name, date of birth, address, ID number (e.g. SSN).

[5]Office of Foreign Asset Control (OFAC) enforces economic and trade sanctions. Banks must perform an OFAC screen on customers opening an account.



Manager defines several categories to distinguish between services.

A service category can be one of the following:

- Core Global Services (Core) - Generic Manager services that are part of the installation.
- Form Services - Usually customer specific **Groovy**[1] services.
- Job Services - Usually customer specific services.
- Transact Functions - Customer specific services using Transact Fluent API.

A service has a service type defining functionality it implements, for example, virus scanning. There can be multiple services per service type. However, only one of them is set as the default service for each type. You can define your own service types and create corresponding services to implement specific requirements, for example, to notify customers via SMS, using Fluent Groovy API.

Services can be defined for an organization, so they are not visible to other organizations and their forms.

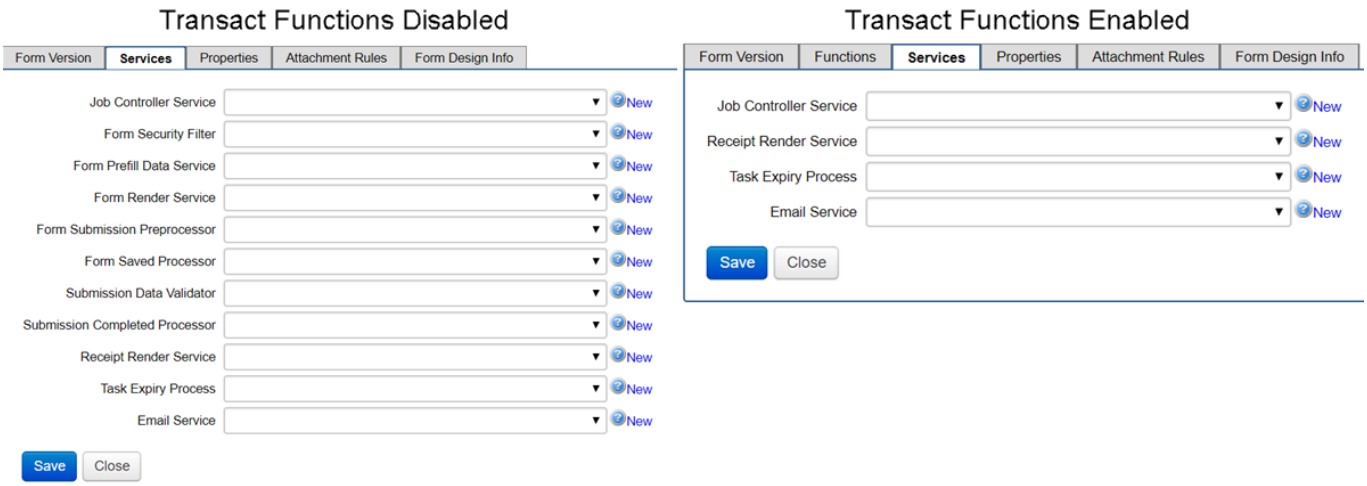
Services can have various parameters, which you can customize to control their run-time behavior.

Some services require a service connection, which defines a connection to an external service, for example, the AWS SQS service connection.

You can migrate existing services from one environment to another to improve your development process.

Forms extensively use services. You can easily configure services to be used by a form. However, when building a TM form version, you have the option of enabling or disabling Transact Functions, which limits the range of services available to the form. The comparison is illustrated below.

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.



Next, learn how to view all services.

Service Types

Journey Manager (JM) | System Manager / DevOps | 20.11

Manager uses service types to define services for various tasks, such as rendering forms or dynamic data calls.

A service type:

- Acts as a template by providing the applicable set of parameters to define a service.
- Determines how a service can be used.
- Can have a default service (if one is selected as the default).

Each service type belongs to one or more categories and has a defined scope.

A service category can be one of the following:

- Core Global Services (Core) - Generic Manager services that are part of the installation.
- Form Services - Usually customer specific **Groovy**[1] services.
- Job Services - Usually customer specific services.
- Transact Functions - Customer specific services using Transact Fluent API.

A service scope can be:

- Global - a service has a global access within Manager.
- Organization - a service has access within a selected organization only.
- Both - Global or Organization, as required.

Manager has the following predefined service types, by which services are grouped:

Service Type	Category	Scope	Description
Data Retention Management	Core Global Service	Global	Performs data purging including **PII**[2] data.

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

[2]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

Delivery Function	Transact Function	Both	Performs the submission data delivery using Fluent functions.
Delivery Process	Form Service		Passes the submission data and attachments, as well a customizable Groovy delivery service across, to perform a delivery.
Dynamic Data	Form Service		Obtains data dynamically from external systems or services. Forms can use this service for data pre-fill, lookup or validation.
Email Queue	Core Global Service		Processes emails stored in the email queue.
Email Service	Core Global Service Form Service		Provides a default **SMTP**[1] email service. You can also create a Groovy email service.
Find Delivery Process ID	Core Global Service		Updates the delivery process ID assigned by **Adobe LiveCycle**[2] for submissions delivered to an Adobe LiveCycle process.

[1]SMTP (Simple Mail Transfer Protocol) is a TCP/IP protocol used in sending and receiving e-mail. However, since it is limited in its ability to queue messages at the receiving end, it is usually used with one of two other protocols, POP3 or IMAP, that let the user save messages in a server mailbox and download them periodically from the server. In other words, users typically use a program that uses SMTP for sending e-mail and either POP3 or IMAP for receiving e-mail. On Unix-based systems, sendmail is the most widely-used SMTP server for e-mail. A commercial package, Sendmail, includes a POP3 server. Microsoft Exchange includes an SMTP server and can also be set up to include POP3 support.

[2]Adobe LiveCycle Enterprise Suite (ES4) is an service-oriented architecture Java EE server software product from Adobe Systems used to build applications that automate a broad range of business processes for enterprises and government agencies. LiveCycle ES4 is an enterprise document and form platform that allows capturing and processing information, delivering personalized communications, and protecting and tracking sensitive information. It is used for purposes such as account opening, services and benefits enrollment, correspondence management, request for proposal processes, and other manual based workflows.

Fluent Function	Transact Function	Both	Implements any service adhering to Transact Fluent API
Form Meta Data	Core Global Service	Global	Renders the **metadata**[1] related to the form as HTML
Form PDF Pre Render	Core Global Service		This service is called when a new Composer SmartForm with an XDP form template is created. It uses an Adobe LiveCycle orchestration to pre-render a PDF from the XDP template. This PDF is used for form rendering, which speeds up rendering and reduces the load on LiveCycle's processes.
Form Prefill	Form Service		Retrieve prefill XML data for a form. The service can be defined at a form version level, otherwise a global default service is used. The service is called at form render time.
Form Saved Processor	Form Service		This service is called when a form is saved to Manager, and the form version in question has been associated with a Form Saved Processor service. The default service allows you to specify custom logic in a Groovy script.
Form Security Filter	Form Service		Provides a security filter for a form.
Form Server Load Balancer	Core Global Service		This service is called by various operations involving LiveCycle -for example, rendering forms and receipts - and is used to ensure capacity to perform the operation is available. It also performs load balancing if more than one LiveCycle server is available.
Form Submission Access Controller	Core Global Service		Provides security features based around the completion of a form.

[1]Metadata is data that describes other data. Meta is a prefix that in most information technology usages means "an underlying definition or description." Metadata summarizes basic information about data, which can make finding and working with particular instances of data easier. For example, author, date created and date modified and file size are examples of very basic document metadata. Having the abilty to filter through that metadata makes it much easier for someone to locate a specific document.

Form Version Selector	Core Global Service Form Service		Selects the appropriate form version to render a form
Groovy Service	Form Service	Both	Provides **Groovy**[1] code that can be reused by other Groovy services in Manager.
Job Action	Job Service		Provides a Job Action service related to collaboration jobs. You can implement it using the **GrooveyJobActionService** or the **FluentJobActionService**. See **com.avoka.fc.core.service.job.impl.JobActionService** API.
Job Action Wait	Job Service		Provides a Job Action Wait service related to collaboration jobs. This service waits until all the steps of the previous **Invoked** actions have been completed. You can implement it using the **GrooveyJobActionService** or the **FluentJobActionService**. See **com.avoka.fc.core.service.job.impl.JobActionWaitService** API.
Job Controller	Job Service Form Service		Creates a new collaboration job for a form submission and then processes it. This service needs to be associated with the form version.

[1] Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

Job Delivery	Job Service	Provides a Job Delivery service related to collaboration jobs. This service performs actual delivery within a collaboration job. This service is used in conjunction with the Job Delivery Wait service to kick off the standard Transact Delivery process. The delivery process is completed by the Transaction Processing Scheduled Job, not the job controller service. See **com.avoka.fc.core.service.job.impl.JobDeliveryService** API.
Job Delivery Wait	Job Service	Provides a Job Delivery Wait service related to collaboration jobs. This service waits for a delivery to be complete as a part of a collaboration job. The Job Delivery Wait service is an asynchronous mechanism. When the delivery is completed by the Transaction Processing job it calls a callback method that allows the Job Delivery Wait to complete. See **com.avoka.fc.core.service.job.impl.JobDeliveryWaitService** API.
Job Expiry	Job Service	Provides a Job Expiry service related to collaboration jobs. You can implement it using the **GroovyJobActionService** or the **FluentJobActionService**. .See **com.avoka.fc.core.service.job.impl.JobExpiryService** API.
Job Form Start	Job Service	Provides a Job Form Start service related to collaboration jobs. This service associates the initial form submission with the initial step. It can add a submission processing status message to the initial submission. These appear in the user's submission history. It can also send a status update via email. See **com.avoka.fc.core.service.job.impl.JobFormStartService** API.
Job Process Message	Job Service	Provides a Job Process Message service related to collaboration jobs. This service adds a submission processing status messages to the specified Submission. These appear in the user's submission history. This action service can be used anywhere in the job but is especially useful in endpoint steps. It can also send a status update via email. See **com.avoka.fc.core.service.job.impl.JobProcessMessageService** API.

Job Receipt Wait	Job Service	Provides a Job Receipt Wait service related to collaboration jobs. This action polls the submissions until the receipts have completed. The generation of the receipts is completed by a separate Transact Scheduled Job 'Transaction Processing' which by default runs every 5 minute. This action will complete when all receipts have been generated for this job's submissions. If any receipt has not completed the action returns a result of "In Progress". This will mean the Job Action / Job processing will stop and retry again in 1 minute. This is required as the existing Job Task Wait service does not wait for receipts to be created. See **com.avoka.fc.core.service.job.impl.JobReceiptWaitService** API.
Job Task Assign	Job Service	Provides a Job Task Assign service related to collaboration jobs. The Job Task Assign Service is used for creating and assigning tasks in Journey Manager. It has the configuration properties associated with: • Status Updates and Email: This service can add a submission processing status messages to the specified Submission. These appear in the user's submission history. It can also send a status update via email • Task Assignment: These properties are used in the creation and assignment of user and group tasks. Tasks can be categorized as: • Standard Tasks - to an authenticated user or a group. • Review Tasks - these are the similar to the standard tasks but form xml and attachments are copied from a previous submission. • Anonymous Save - A submission record is saved but not assigned to a user in the database. They are usually assigned to a email address, the user is sent a notification email. They click on a link. The user has to enter a challenge response before they have access to the task form. See **com.avoka.fc.core.service.job.impl.JobTaskAssignService** API.

Job Task Wait	Job Service		Provides a Job Task Wait service related to collaboration jobs. A Job Task Wait Service is paired in the 1 step with 1 or more Job Task Assign Action that creates a task. It provided an asynchronous wait mechanism described below in State Transition section. that allows the user time to complete the task via submission. When a submission comes in the Job Task Wait will execute. The code will check that JobTaskAssign actions have completed. • Not All Complete: it will go back and wait for more submissions. • All Complete: It will allow execution to the next step. See **`com.avoka.fc.core.service.job.impl.JobTaskWaitService`** API.
Media Type Scan	Core Global Service		Scans and checks the contents of an attachment matches the specified file extension. | 20.11
Payment Gateway	Core Global Service	Both	Processes payments by connecting to a payment gateway and making a payment transaction.
Payment Reminder	Core Global Service		Sends out reminder emails about submissions that have outstanding payments older than a certain age; that is, the **Reminder Age Hours** service parameter. Emails are sent to administrators belonging to the **Receive Outstanding Payment Alerts** group. This service is periodically called by the payment reminder job.
Receipt Number	Core Global Service Form Service		Calculates receipt numbers for new submissions.

Render Composer HTML Form	Core Global Service	Global	Renders HTML forms created in Maestro or Composer.
Render Form	Core Global Service Form Service		Uses an orchestration in **LiveCycle**[1] to render a form when it is opened to be filled in.
Render Offline Submission Form	Core Global Service		Uses an orchestration in LiveCycle to render a form when it is submitted after being saved offline.
Render Receipt	Core Global Service		Renders a form's receipt using one of the following orchestrations: **LiveCycle**[2], **PhantomJS**[3], Static PDF, IText.

[1]Adobe LiveCycle Enterprise Suite (ES4) is an service-oriented architecture Java EE server software product from Adobe Systems used to build applications that automate a broad range of business processes for enterprises and government agencies. LiveCycle ES4 is an enterprise document and form platform that allows capturing and processing information, delivering personalized communications, and protecting and tracking sensitive information. It is used for purposes such as account opening, services and benefits enrollment, correspondence management, request for proposal processes, and other manual based workflows.

[2]Adobe LiveCycle Enterprise Suite (ES4) is an service-oriented architecture Java EE server software product from Adobe Systems used to build applications that automate a broad range of business processes for enterprises and government agencies. LiveCycle ES4 is an enterprise document and form platform that allows capturing and processing information, delivering personalized communications, and protecting and tracking sensitive information. It is used for purposes such as account opening, services and benefits enrollment, correspondence management, request for proposal processes, and other manual based workflows.

[3]PhantomJS is a scripted, headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment. PhantomJS is based on WebKit making it a similar browsing environment to Safari and Google Chrome (before Chrome's fork of WebKit evolved into Blink). It is open-source software released under the BSD License.

	Form Service	
Repository Publish	Core Global Service	Publishes a single form to the **LiveCycle**[1] repositories
Repository Synchronize	Core Global Service	Brings the LiveCycle repositories up to date after the set of form versions has changed.
Scheduled Service	Core Global Service Form Service	Executes custom logic defined as a Groovy script.
Submission Completed Processor	Form Service	Processes completed submissions.
Submission Data Extraction	Core Global Service	Provides a service to extract the XML data document from a HTML form submission.
Submission Data Storage	Core Global Service	Stores and retrieves submission, receipt and attachment data.

[1]Adobe LiveCycle Enterprise Suite (ES4) is an service-oriented architecture Java EE server software product from Adobe Systems used to build applications that automate a broad range of business processes for enterprises and government agencies. LiveCycle ES4 is an enterprise document and form platform that allows capturing and processing information, delivering personalized communications, and protecting and tracking sensitive information. It is used for purposes such as account opening, services and benefits enrollment, correspondence management, request for proposal processes, and other manual based workflows.

Submission Data Validator	Form Service	Validates submitted data.
Submission Delivery Controller	Core Global Service	Performs submission delivery, marks submissions as undeliverable, and sends reminders.
Submission Email Verification	Core Global Service	Sends email verification requests to users who have submitted a form.
Submission Preprocessor	Form Service	Process submission data before it's submitted.
Submission Receipt	Core Global Service	Creates and stores receipts for new submissions, and reattempts to render receipts for submissions where previous attempts failed.
Submission Service	Core Global Service	Performs submissions.
System Event Publisher	Core Global Service	Sends various server events to external systems. You can use both the Azure and AWS system event publishers.
Task Expiry Process	Core Global Service Form Service	Invokes an external process to notify interested parties that the task has expired.

Test Suite	Form Service	Provides a test suite.
Tracking Number	Core Global Service Form Service	Calculates tracking codes for new submissions.
Transaction Processor	Core Global Service	Performs processing of all submissions.
Two Factor Authentication	Core Global Service	Validates the second factor in **two-factor authentication**[1].
Virus Scan	Core Global Service Form Service	Scans attachments stored in Manager for viruses using various antivirus software.

You can create new services based on the existing types.

Next, learn how to view all services.

---

[1]Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.

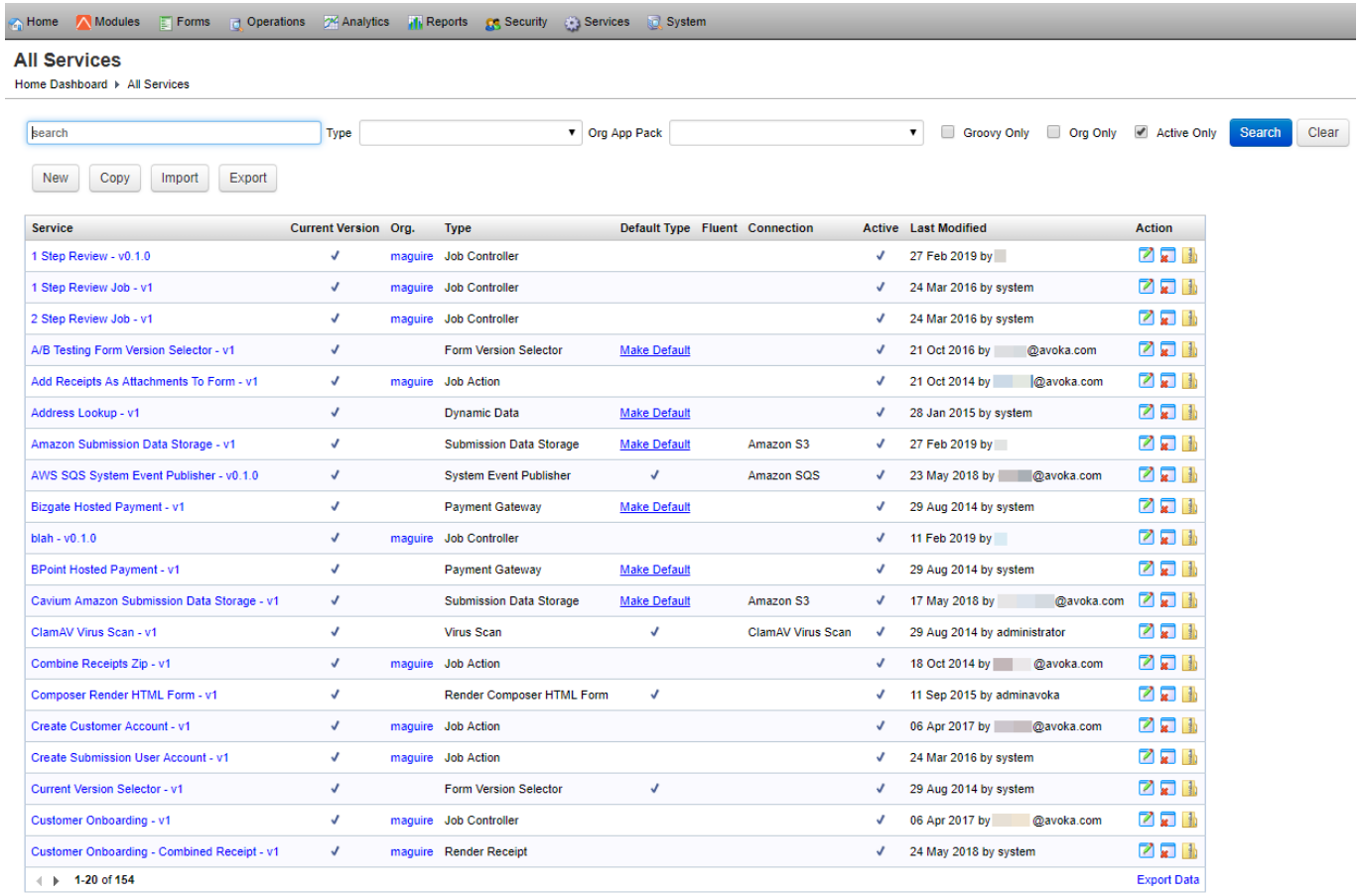
View Services

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view all services, so you can easily find and edit an existing service.

To view all services, select **Services > All Services.**

Services are shown page by page, with the maximum number of records per page configured in the users preferences.



To filter or search the services by one or more criteria, specify the following settings and click **Search**:

- Service name or part of a name
- Service type
- Org App Pack to see services belonging to a selected organization application package
- Groovy Only to see **Groovy**[1] services only
- Org Only to show organization service definitions only
- Active Only
- Organization to show services belonging to a selected organization.

Click **Clear** to reset the search criteria.

You can check the following service details:

- Service name
- Current Version
- Organization
- Type of a service.
- **Default Type** represents whether this service is the default service for this service type. Only a global services can be marked as the default one.
- Fluent shows if this is a **Fluent**[2] service.
- Connection is the service connection used by this service, if configured.
- Active shows whether this service is active or not. It must be active to be available.

> It is recommended to make a service not active rather then remove it when you don't need it any more.

- Last Modified shows the date when this service was last modified and by whom.

You can also perform the following actions:

---

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

[2]Fluent is a programming style that can be applied to a wide range of programming languages. Across the Journey platform, fluent is commonly associated with the Fluent SDK. The Fluent SDK is intended for use in developer integrated developer environment's (IDE) and contains libraries and project files for all the different services it supports.

- Create a service
- Configure a service
- Copy a service
- Remove a service
- Export services
- Import services
- Run a unit test to validate a selected service. This option is only available for Groovy, scheduled and virus scan services.
- Click **Make Default** to mark the selected service as a default one for all services of this type.

> To clear a service default status, open the service configuration and clear the **Service Type Default** check-box.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to view form services.

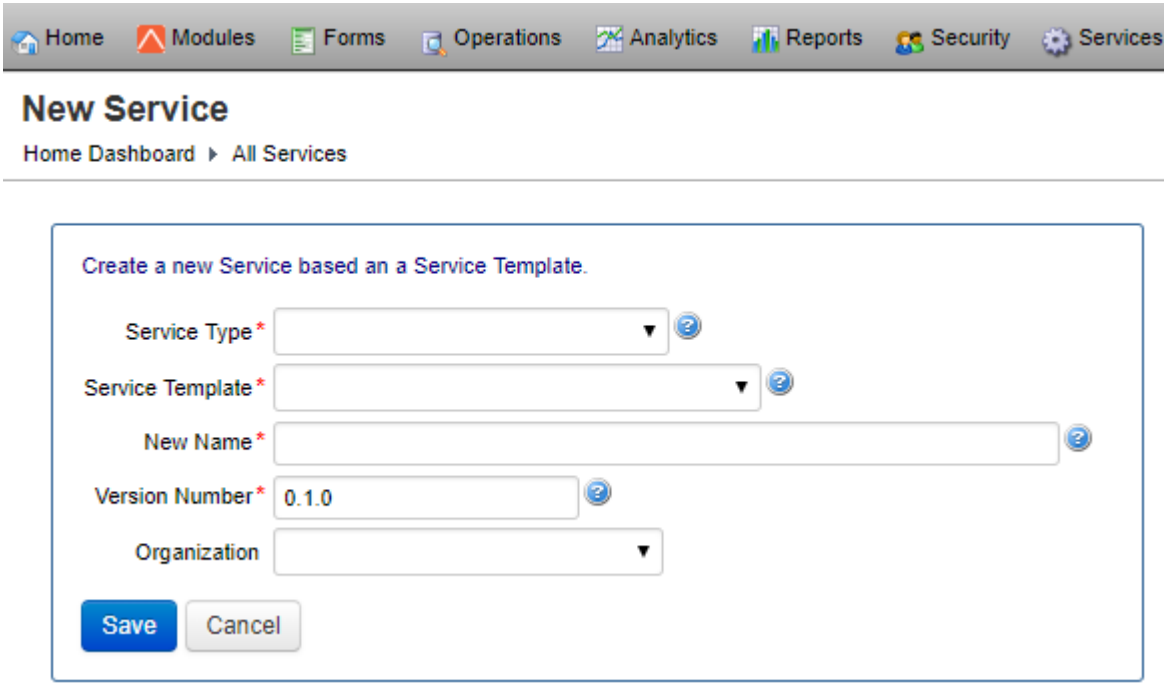
Create a Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create new services. You can use this functionality to implement new customer requirements.

To create a new service:

1. Select **Services > All Services**.
2. Click **New**.



3. Select a service type from the **Service Type** dropdown list to filter the types of **Service Template** to create your service from. Each service type may have multiple templates.
4. Select a service template from the **Service Template** dropdown list to create your service from.
5. Give the new service a globally unique name in the **New Name** field, which automatically generated based on the selected template.
6. Edit a new version number in the **Version Number** field.
7. Select an organization from the **Organization** dropdown list so the service can be associated only with forms and other entities belonging to that organization. If you do not select an organization, the service will be globally available.

> You must be a Journey Manager administrator to create or configure this functionality.

8. Click **Save** to update the changes.
9. Configure the service.

Next, learn how to remove a service.

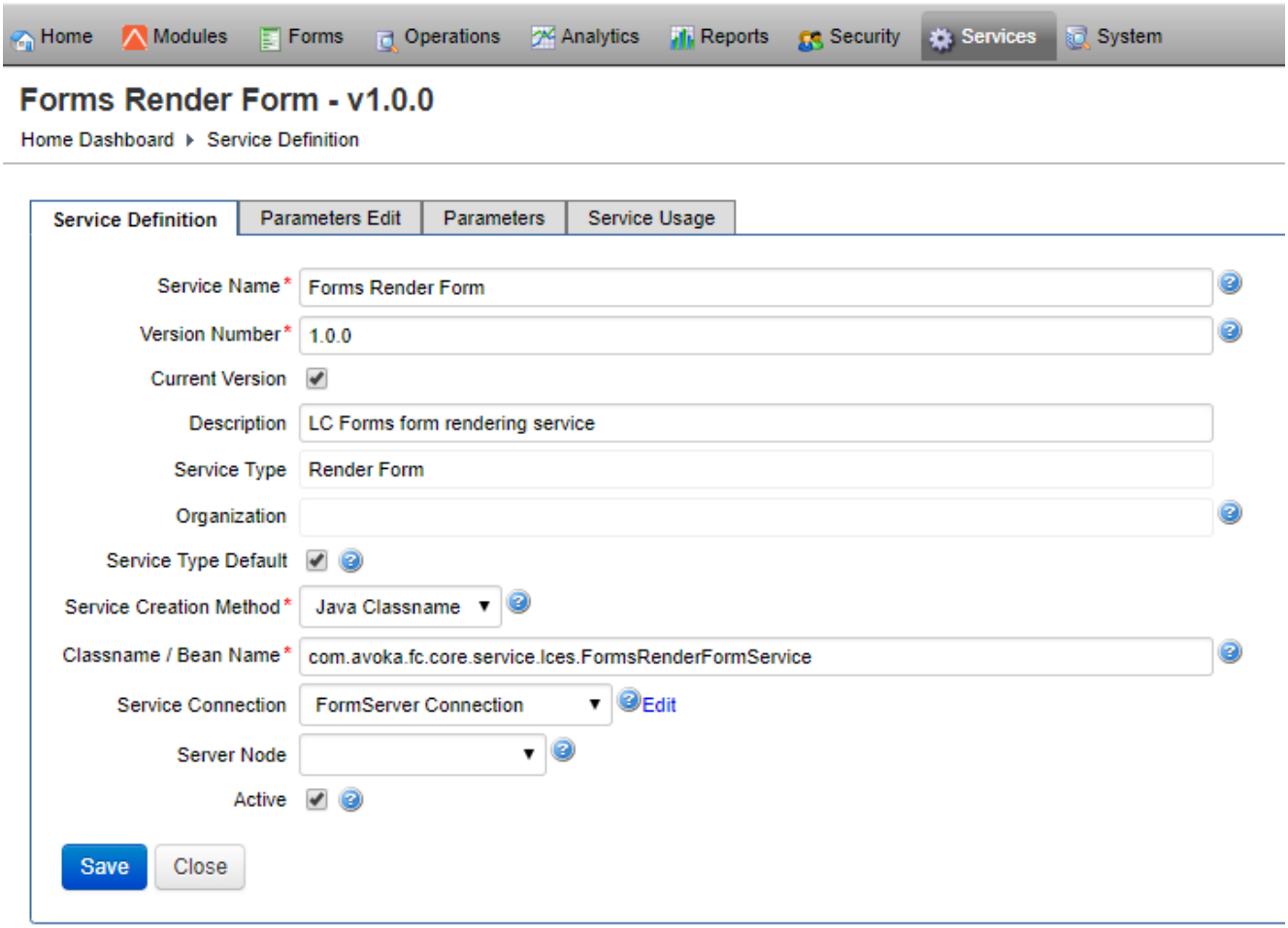
Configure a Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create new and configure existing services.

To configure a service:

1. Select **Services > All Services**.
2. Locate a service and click **Edit**.
3. Select the **Service Definition** tab to view or edit the standard service configuration.



4. Select the **Current Version** checkbox to make this service version current. This applies when there are multiple versions of this service, because a new version of this service is not automatically set as the current version.
5. Enter a meaningful service description in the **Description** field.
6. Select the **Service Type Default** checkbox to make this service as the default one, so it is executed for any service of this service type.

You can also make a service as default from the list of all services.

7. Select the service creation method from the **Service Creation Method** dropdown list:
   - Java Classname
   - Spring Bean
8. Update the class name or bean name of the service.
9. Select the service connection from the **Service Connection** dropdown list.
10. Select a server node to be associated with this service from the **Server Node** dropdown list.
11. Select the **Active** checkbox to make this service active. It is selected by default.

Clear the **Active** checkbox to make the service unavailable in various Manager's configurations by not showing it in the dropdown list of available services. This doesn't disable or deactivate the service itself, therefore existing forms will continue to use this service.

12. Click **Save** to update the changes.

Next, learn how to remove a service.

Service Types

Journey Manager (JM) | System Manager / DevOps | 20.11

Manager uses service types to define services for various tasks, such as rendering forms or dynamic data calls.

A service type:

- Acts as a template by providing the applicable set of parameters to define a service.
- Determines how a service can be used.
- Can have a default service (if one is selected as the default).

Each service type belongs to one or more categories and has a defined scope.

A service category can be one of the following:

- Core Global Services (Core) - Generic Manager services that are part of the installation.
- Form Services - Usually customer specific **Groovy**[1] services.
- Job Services - Usually customer specific services.
- Transact Functions - Customer specific services using Transact Fluent API.

A service scope can be:

- Global - a service has a global access within Manager.
- Organization - a service has access within a selected organization only.
- Both - Global or Organization, as required.

Manager has the following predefined service types, by which services are grouped:

Service Type	Category	Scope	Description
Data Retention Management	Core Global Service	Global	Performs data purging including **PII**[2] data.

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

[2]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

Delivery Function	Transact Function	Both	Performs the submission data delivery using Fluent functions.
Delivery Process	Form Service		Passes the submission data and attachments, as well a customizable Groovy delivery service across, to perform a delivery.
Dynamic Data	Form Service		Obtains data dynamically from external systems or services. Forms can use this service for data pre-fill, lookup or validation.
Email Queue	Core Global Service		Processes emails stored in the email queue.
Email Service	Core Global Service Form Service		Provides a default **SMTP**[1] email service. You can also create a Groovy email service.
Find Delivery Process ID	Core Global Service		Updates the delivery process ID assigned by **Adobe LiveCycle**[2] for submissions delivered to an Adobe LiveCycle process.

[1]SMTP (Simple Mail Transfer Protocol) is a TCP/IP protocol used in sending and receiving e-mail. However, since it is limited in its ability to queue messages at the receiving end, it is usually used with one of two other protocols, POP3 or IMAP, that let the user save messages in a server mailbox and download them periodically from the server. In other words, users typically use a program that uses SMTP for sending e-mail and either POP3 or IMAP for receiving e-mail. On Unix-based systems, sendmail is the most widely-used SMTP server for e-mail. A commercial package, Sendmail, includes a POP3 server. Microsoft Exchange includes an SMTP server and can also be set up to include POP3 support.

[2]Adobe LiveCycle Enterprise Suite (ES4) is an service-oriented architecture Java EE server software product from Adobe Systems used to build applications that automate a broad range of business processes for enterprises and government agencies. LiveCycle ES4 is an enterprise document and form platform that allows capturing and processing information, delivering personalized communications, and protecting and tracking sensitive information. It is used for purposes such as account opening, services and benefits enrollment, correspondence management, request for proposal processes, and other manual based workflows.

Fluent Function	Transact Function	Both	Implements any service adhering to Transact Fluent API
Form Meta Data	Core Global Service	Global	Renders the **metadata**[1] related to the form as HTML
Form PDF Pre Render	Core Global Service		This service is called when a new Composer SmartForm with an XDP form template is created. It uses an Adobe LiveCycle orchestration to pre-render a PDF from the XDP template. This PDF is used for form rendering, which speeds up rendering and reduces the load on LiveCycle's processes.
Form Prefill	Form Service		Retrieve prefill XML data for a form. The service can be defined at a form version level, otherwise a global default service is used. The service is called at form render time.
Form Saved Processor	Form Service		This service is called when a form is saved to Manager, and the form version in question has been associated with a Form Saved Processor service. The default service allows you to specify custom logic in a Groovy script.
Form Security Filter	Form Service		Provides a security filter for a form.
Form Server Load Balancer	Core Global Service		This service is called by various operations involving LiveCycle -for example, rendering forms and receipts - and is used to ensure capacity to perform the operation is available. It also performs load balancing if more than one LiveCycle server is available.
Form Submission Access Controller	Core Global Service		Provides security features based around the completion of a form.

[1]Metadata is data that describes other data. Meta is a prefix that in most information technology usages means "an underlying definition or description." Metadata summarizes basic information about data, which can make finding and working with particular instances of data easier. For example, author, date created and date modified and file size are examples of very basic document metadata. Having the abilty to filter through that metadata makes it much easier for someone to locate a specific document.

Form Version Selector	Core Global Service Form Service		Selects the appropriate form version to render a form
Groovy Service	Form Service	Both	Provides **Groovy**[1] code that can be reused by other Groovy services in Manager.
Job Action	Job Service		Provides a Job Action service related to collaboration jobs. You can implement it using the **GrooveyJobActionService** or the **FluentJobActionService**. See **com.avoka.fc.core.service.job.impl.JobActionService** API.
Job Action Wait	Job Service		Provides a Job Action Wait service related to collaboration jobs. This service waits until all the steps of the previous **Invoked** actions have been completed. You can implement it using the **GrooveyJobActionService** or the **FluentJobActionService**. See **com.avoka.fc.core.service.job.impl.JobActionWaitService** API.
Job Controller	Job Service Form Service		Creates a new collaboration job for a form submission and then processes it. This service needs to be associated with the form version.

[1] Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

Job Delivery	Job Service	Provides a Job Delivery service related to collaboration jobs. This service performs actual delivery within a collaboration job. This service is used in conjunction with the Job Delivery Wait service to kick off the standard Transact Delivery process. The delivery process is completed by the Transaction Processing Scheduled Job, not the job controller service. See **com.avoka.fc.core.service.job.impl.JobDeliveryService** API.
Job Delivery Wait	Job Service	Provides a Job Delivery Wait service related to collaboration jobs. This service waits for a delivery to be complete as a part of a collaboration job. The Job Delivery Wait service is an asynchronous mechanism. When the delivery is completed by the Transaction Processing job it calls a callback method that allows the Job Delivery Wait to complete. See **com.avoka.fc.core.service.job.impl.JobDeliveryWaitService** API.
Job Expiry	Job Service	Provides a Job Expiry service related to collaboration jobs. You can implement it using the **GroovyJobActionService** or the **FluentJobActionService**. .See **com.avoka.fc.core.service.job.impl.JobExpiryService** API.
Job Form Start	Job Service	Provides a Job Form Start service related to collaboration jobs. This service associates the initial form submission with the initial step. It can add a submission processing status message to the initial submission. These appear in the user's submission history. It can also send a status update via email. See **com.avoka.fc.core.service.job.impl.JobFormStartService** API.
Job Process Message	Job Service	Provides a Job Process Message service related to collaboration jobs. This service adds a submission processing status messages to the specified Submission. These appear in the user's submission history. This action service can be used anywhere in the job but is especially useful in endpoint steps. It can also send a status update via email. See **com.avoka.fc.core.service.job.impl.JobProcessMessageService** API.

Job Receipt Wait	Job Service	Provides a Job Receipt Wait service related to collaboration jobs. This action polls the submissions until the receipts have completed. The generation of the receipts is completed by a separate Transact Scheduled Job 'Transaction Processing' which by default runs every 5 minute. This action will complete when all receipts have been generated for this job's submissions. If any receipt has not completed the action returns a result of "In Progress". This will mean the Job Action / Job processing will stop and retry again in 1 minute. This is required as the existing Job Task Wait service does not wait for receipts to be created. See **com.avoka.fc.core.service.job.impl.JobReceiptWaitService** API.
Job Task Assign	Job Service	Provides a Job Task Assign service related to collaboration jobs. The Job Task Assign Service is used for creating and assigning tasks in Journey Manager. It has the configuration properties associated with: • Status Updates and Email: This service can add a submission processing status messages to the specified Submission. These appear in the user's submission history. It can also send a status update via email • Task Assignment: These properties are used in the creation and assignment of user and group tasks. Tasks can be categorized as: • Standard Tasks - to an authenticated user or a group. • Review Tasks - these are the similar to the standard tasks but form xml and attachments are copied from a previous submission. • Anonymous Save - A submission record is saved but not assigned to a user in the database. They are usually assigned to a email address, the user is sent a notification email. They click on a link. The user has to enter a challenge response before they have access to the task form. See **com.avoka.fc.core.service.job.impl.JobTaskAssignService** API.

Job Task Wait	Job Service		Provides a Job Task Wait service related to collaboration jobs. A Job Task Wait Service is paired in the 1 step with 1 or more Job Task Assign Action that creates a task. It provided an asynchronous wait mechanism described below in State Transition section. that allows the user time to complete the task via submission. When a submission comes in the Job Task Wait will execute. The code will check that JobTaskAssign actions have completed. • Not All Complete: it will go back and wait for more submissions. • All Complete: It will allow execution to the next step. See **`com.avoka.fc.core.service.job.impl.JobTaskWaitService`** API.
Media Type Scan	Core Global Service		Scans and checks the contents of an attachment matches the specified file extension. | 20.11
Payment Gateway	Core Global Service	Both	Processes payments by connecting to a payment gateway and making a payment transaction.
Payment Reminder	Core Global Service		Sends out reminder emails about submissions that have outstanding payments older than a certain age; that is, the **Reminder Age Hours** service parameter. Emails are sent to administrators belonging to the **Receive Outstanding Payment Alerts** group. This service is periodically called by the payment reminder job.
Receipt Number	Core Global Service Form Service		Calculates receipt numbers for new submissions.

Render Composer HTML Form	Core Global Service	Global	Renders HTML forms created in Maestro or Composer.
Render Form	Core Global Service Form Service		Uses an orchestration in **LiveCycle**[1] to render a form when it is opened to be filled in.
Render Offline Submission Form	Core Global Service		Uses an orchestration in LiveCycle to render a form when it is submitted after being saved offline.
Render Receipt	Core Global Service		Renders a form's receipt using one of the following orchestrations: **LiveCycle**[2], **PhantomJS**[3], Static PDF, IText.

[1]Adobe LiveCycle Enterprise Suite (ES4) is an service-oriented architecture Java EE server software product from Adobe Systems used to build applications that automate a broad range of business processes for enterprises and government agencies. LiveCycle ES4 is an enterprise document and form platform that allows capturing and processing information, delivering personalized communications, and protecting and tracking sensitive information. It is used for purposes such as account opening, services and benefits enrollment, correspondence management, request for proposal processes, and other manual based workflows.

[2]Adobe LiveCycle Enterprise Suite (ES4) is an service-oriented architecture Java EE server software product from Adobe Systems used to build applications that automate a broad range of business processes for enterprises and government agencies. LiveCycle ES4 is an enterprise document and form platform that allows capturing and processing information, delivering personalized communications, and protecting and tracking sensitive information. It is used for purposes such as account opening, services and benefits enrollment, correspondence management, request for proposal processes, and other manual based workflows.

[3]PhantomJS is a scripted, headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment. PhantomJS is based on WebKit making it a similar browsing environment to Safari and Google Chrome (before Chrome's fork of WebKit evolved into Blink). It is open-source software released under the BSD License.

	Form Service	
Repository Publish	Core Global Service	Publishes a single form to the **LiveCycle**[1] repositories
Repository Synchronize	Core Global Service	Brings the LiveCycle repositories up to date after the set of form versions has changed.
Scheduled Service	Core Global Service Form Service	Executes custom logic defined as a Groovy script.
Submission Completed Processor	Form Service	Processes completed submissions.
Submission Data Extraction	Core Global Service	Provides a service to extract the XML data document from a HTML form submission.
Submission Data Storage	Core Global Service	Stores and retrieves submission, receipt and attachment data.

[1]Adobe LiveCycle Enterprise Suite (ES4) is an service-oriented architecture Java EE server software product from Adobe Systems used to build applications that automate a broad range of business processes for enterprises and government agencies. LiveCycle ES4 is an enterprise document and form platform that allows capturing and processing information, delivering personalized communications, and protecting and tracking sensitive information. It is used for purposes such as account opening, services and benefits enrollment, correspondence management, request for proposal processes, and other manual based workflows.

Submission Data Validator	Form Service	Validates submitted data.
Submission Delivery Controller	Core Global Service	Performs submission delivery, marks submissions as undeliverable, and sends reminders.
Submission Email Verification	Core Global Service	Sends email verification requests to users who have submitted a form.
Submission Preprocessor	Form Service	Process submission data before it's submitted.
Submission Receipt	Core Global Service	Creates and stores receipts for new submissions, and reattempts to render receipts for submissions where previous attempts failed.
Submission Service	Core Global Service	Performs submissions.
System Event Publisher	Core Global Service	Sends various server events to external systems. You can use both the Azure and AWS system event publishers.
Task Expiry Process	Core Global Service Form Service	Invokes an external process to notify interested parties that the task has expired.

Test Suite	Form Service	Provides a test suite.
Tracking Number	Core Global Service Form Service	Calculates tracking codes for new submissions.
Transaction Processor	Core Global Service	Performs processing of all submissions.
Two Factor Authentication	Core Global Service	Validates the second factor in **two-factor authentication**[1].
Virus Scan	Core Global Service Form Service	Scans attachments stored in Manager for viruses using various antivirus software.

You can create new services based on the existing types.

Next, learn how to view all services.

---

[1]Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.

Configure Service Parameters

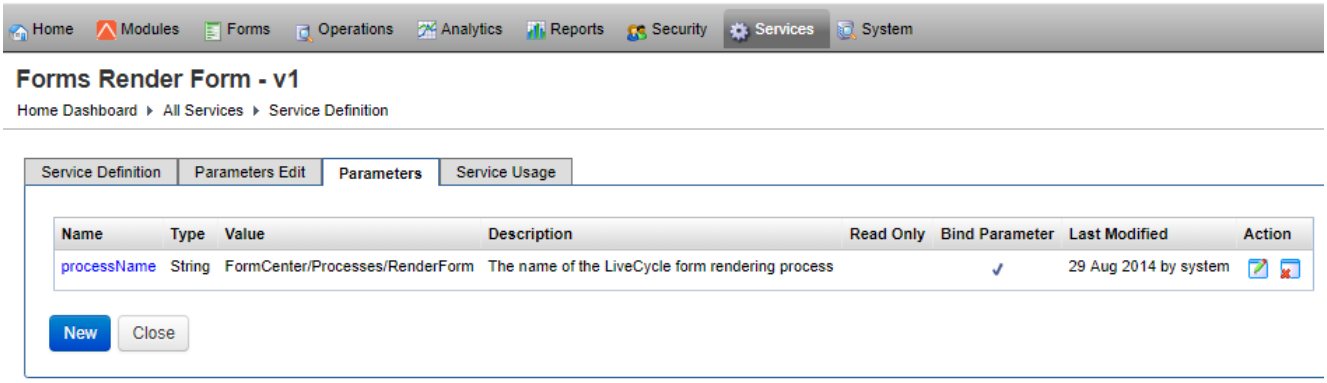
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view, create, configure, and remove service's parameters.

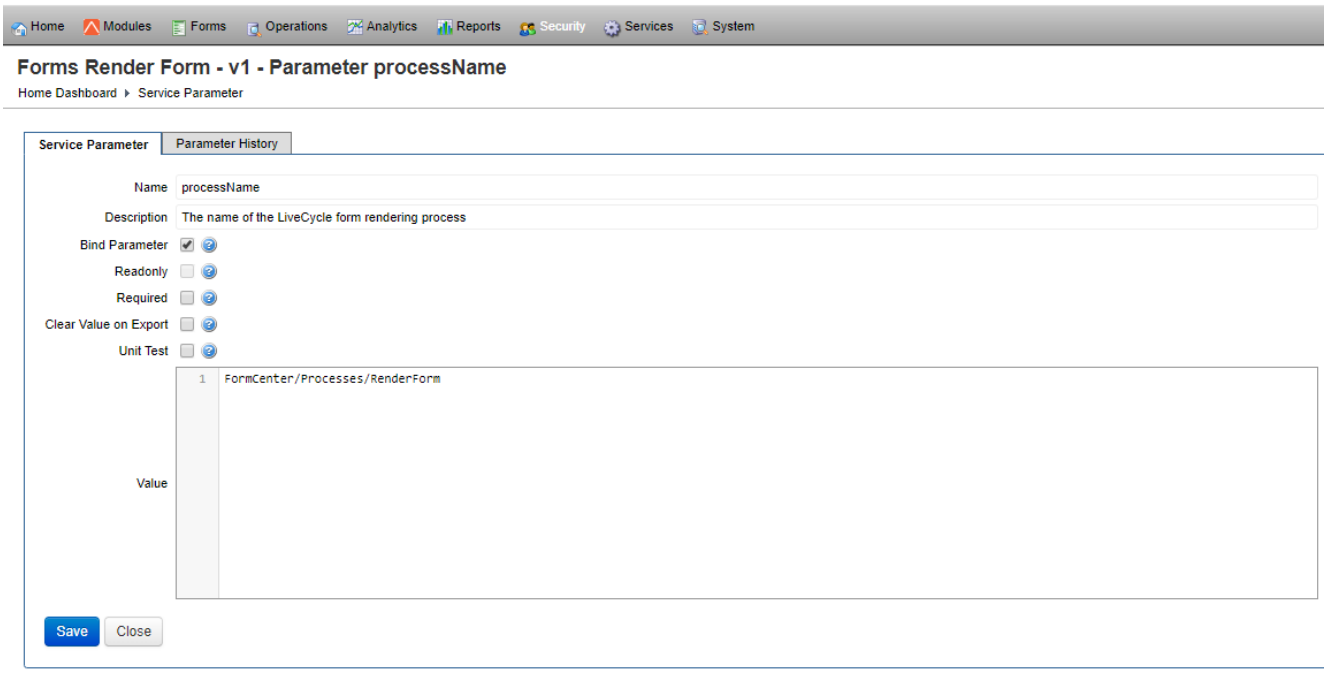
> You can update the same parameters in the Parameters Edit tab.

To configure service parameters:

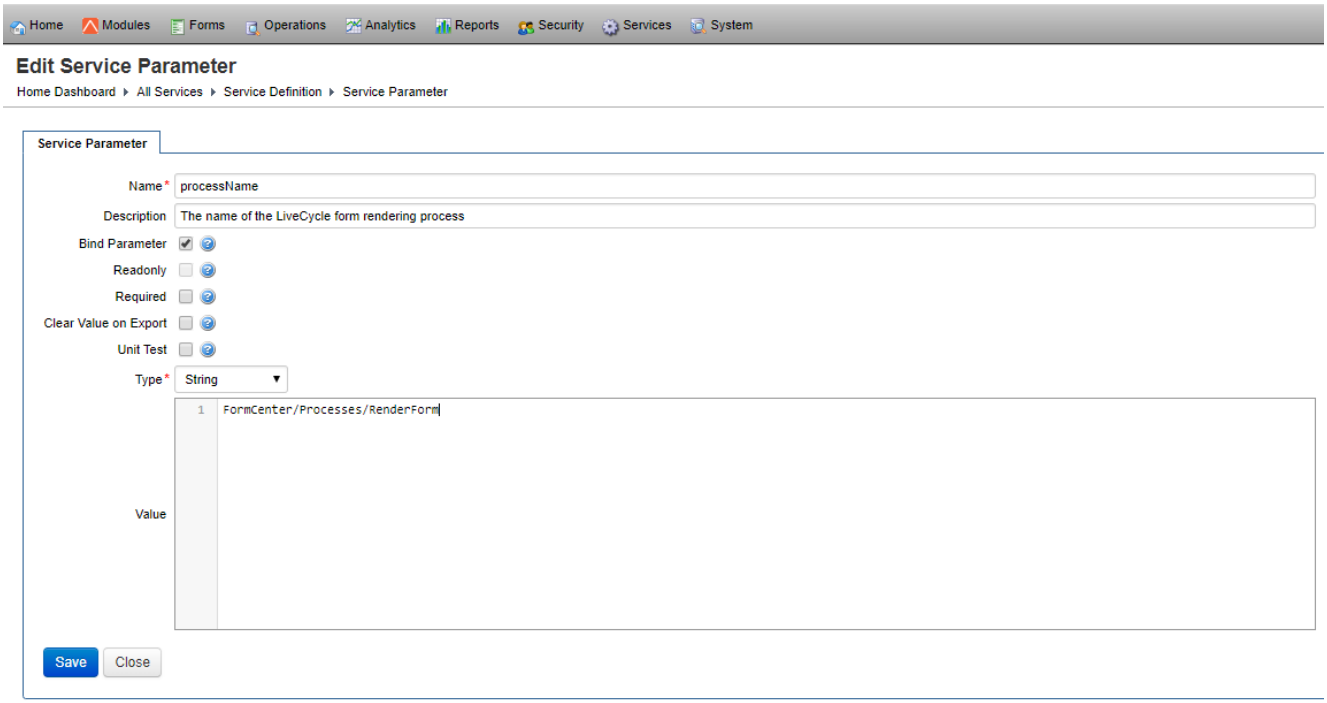
1. Select **Services > All Services**.
2. Locate a service and click **Edit**.
3. Select the **Parameters** tab to view or edit the service configuration.



4. Locate a parameter to modify and click **Edit**. For example, click **processName** to view its configuration as shown below.



5. Click **New** to create a new parameter.
6. Fill in all required fields.



- Select the **Bind Parameter** checkbox to inject the parameter value into the created service.
- Select the type of the parameter, which is a **String** in this example.

7. Click **Save** to update the changes.

Next, learn how to check where this service is used.

Configure Service Parameters Edit

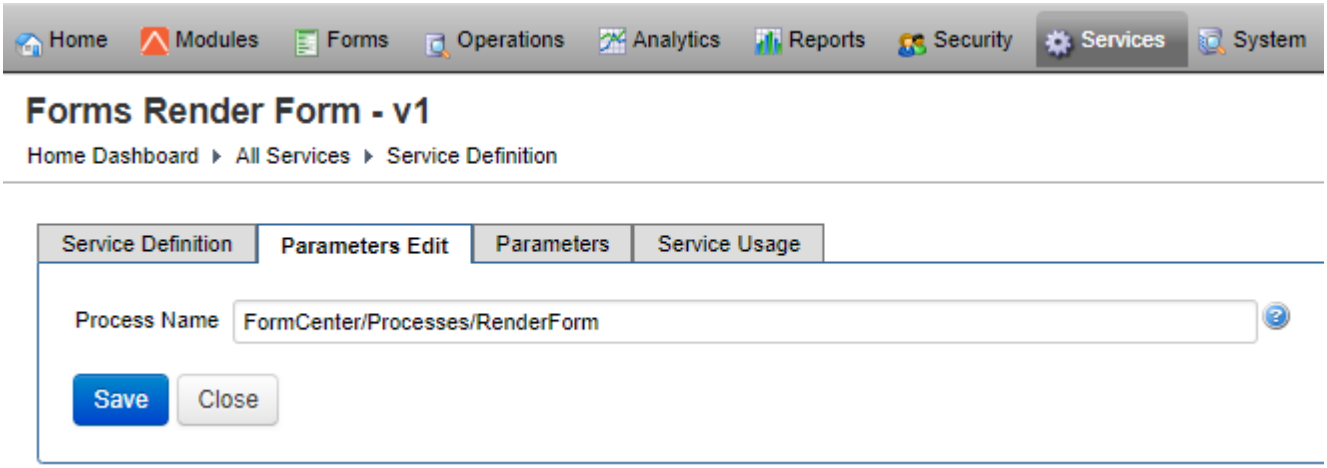
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view and configure service's parameters using the Parameters Edit tab, which is a quicker way than using the Parameters tab.

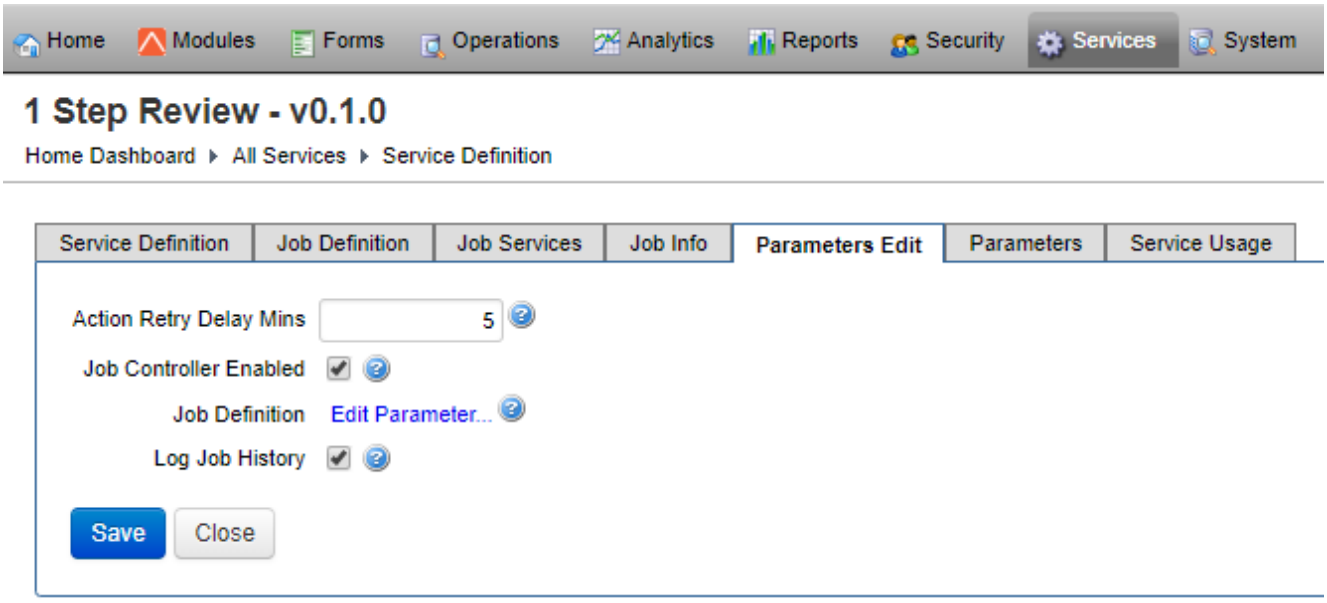
> You can update the same parameters in the Parameters tab.

To configure service parameters:

1. Select **Services > All Services**.
2. Locate a service and click **Edit**.
3. Select the **Parameters Edit** tab.



> The layout of this tab varies depending on the type of the properties.



4. Locate a parameter and click it to edit.
5. Click the **Parameter History** tab to see the change history.



6. Click **View Content Difference** to check what was changed. Select the **View Type** to show the difference side by side or inline.
7. Click **Restore History Content** to restore this history content.
8. Click **Remove Service Parameter History** to remove this history content.
9. Click **Save** to update the changes.

Next, learn how to check where this service is used.

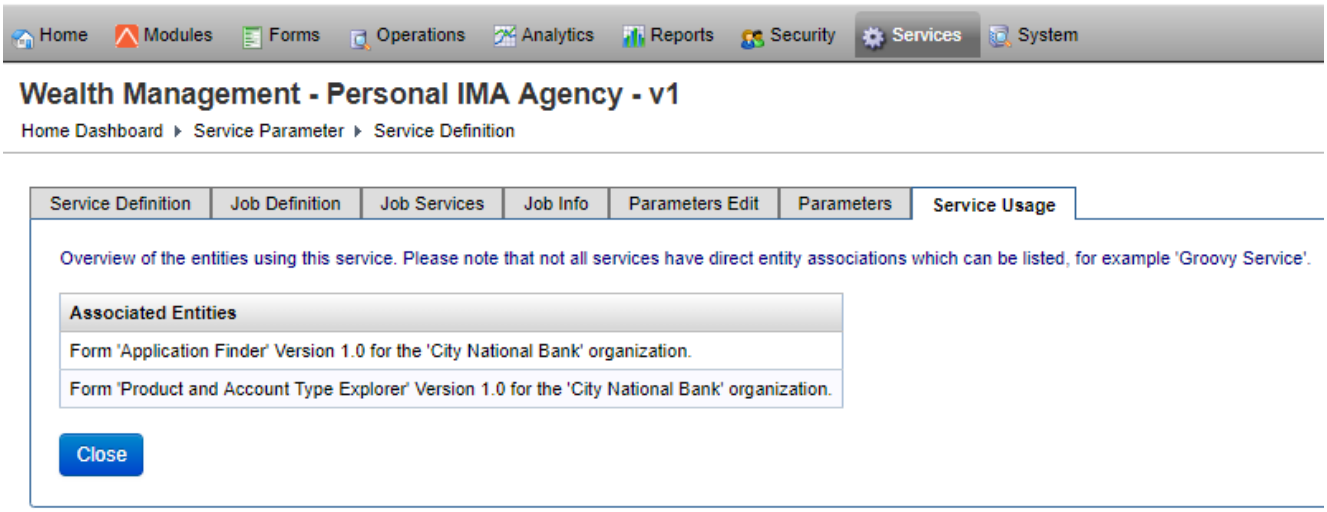
Service Usage

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager shows the forms linked to a service that allows you to determine where this service is used, so you can avoid unwanted side effects when modifying or removing the service.

To check service usage:

1. Select **Services > All Services**.
2. Locate a service and click the **Service Usage** tab.



This tab is shown for service types that can be explicitly associated with entities such as form versions, organizations and delivery channels. However, not all services have direct entity associations which can be listed, for example the Groovy Service.

3. Click **Close**.

Next, learn how to view all services.

Copy a Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

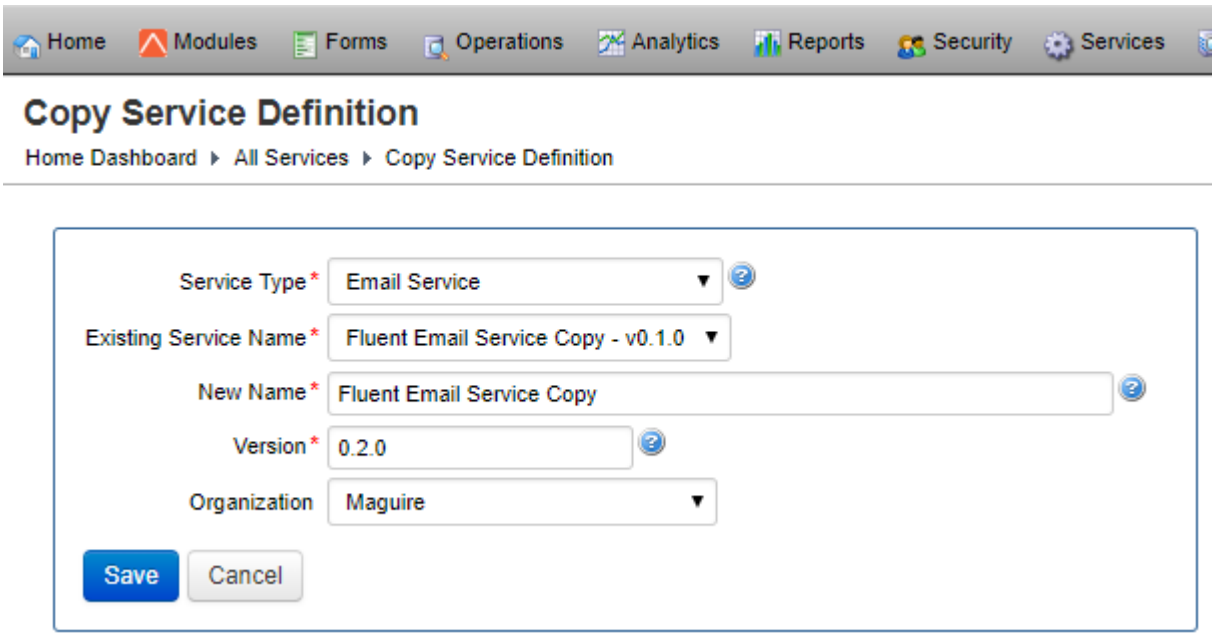
Manager allows you to copy an existing service. You can use this functionality to:

- Test services.
- Create multiple copies of the same service.
- Configure a new service from an existing service.

> The copied service inherits its base configuration and all service parameters from the original service rather than a service template.

To copy an existing service:

1. Select **Services > All Services**.
2. Click **Copy**.



3. Select a service type for which you want to create a new service definition from the **Service Type** dropdown list.
4. Select a service you want to create a copy of from the **Existing Service Name** dropdown list.
5. Enter a new globally unique service in the **New Name** field.
6. Select an organization from the **Organization** dropdown list to restrict the new service to a specific organization. If left blank, this service is a global service and is available to all organizations.
7. Click **Save** to update the changes. Manager creates the new service, which is identical to the original apart

from the name.

8. Configure the service according to your requirements.

Next, learn how to view all services.

Remove a Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

As you work on a service by improving its functionality to meet new requirements, you often create a new version of the service to keep tracks of changes. At some stage, a service or its old version is not required any more, so you want to disable it. Manager provides you with the following options:

- Deactivate or disable a service
- Remove a service

> We recommend deactivating a service, rather than removing it.

Deactivate a Service

To deactivate a service:

1. Select **Services > All Services**.
2. Locate the service and click **Edit**.
3. Select the **Service Definition** tab.
4. Clear the **Active** checkbox.

> Clear the **Active** checkbox to make the service unavailable in various Manager's configurations by not showing it in the dropdown list of available services. This doesn't disable or deactivate the service itself, therefore existing forms will continue to use this service.

5. Click **Save** to update the changes.

> To reactivate the service, select its **Active** checkbox.

Remove a Service

When you remove a service, Manager irrevocably removes all data associated with the service, so it can not be restored. Therefore, we recommend backing up the service before deleting it.

To remove a service:

1. Select **Services > All Services**.
2. Locate the service.
3. Check the service is not used anywhere, for example form versions, to avoid errors.

It is not recommended to delete a service that is the default for a service type.

4. Click **Remove** and then click **OK** to confirm.

Next, learn how to create services.

Migrate Services

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides a mechanism to migrate services from one environment to another. This allows you to develop and test a service in different environments before migrating it over to the production environment.

To migrate one or several services:

1. Export services
2. Import services

> You can use the same process to perform a service rollback.

Next, learn how to create a service.

Export Services

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows administrators to export services. You can use this functionality to:

- Back up or archive services.
- Re-create or migrate services in another Manager.
- Promote services to another environment.

To export one or several services:

1. Select **Services > All Services.**

> If you use services with multiple endpoints, select the correct service connection before the export.

2. Click **Export Archive** to export one service or click **Export** to export all services.

Manager exports them into a service archive ZIP file, which will be downloaded to your PC. The ZIP file contains service-config.xml file with all services' definitions.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to import services.

Import Services

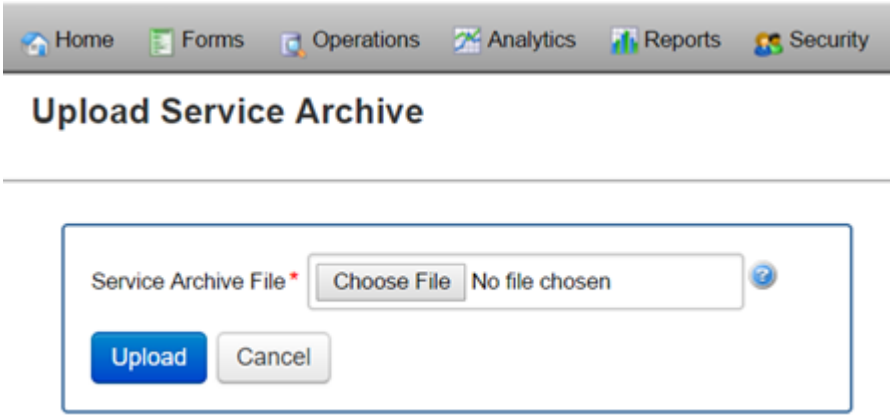
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to import service connections. Use this functionality to:

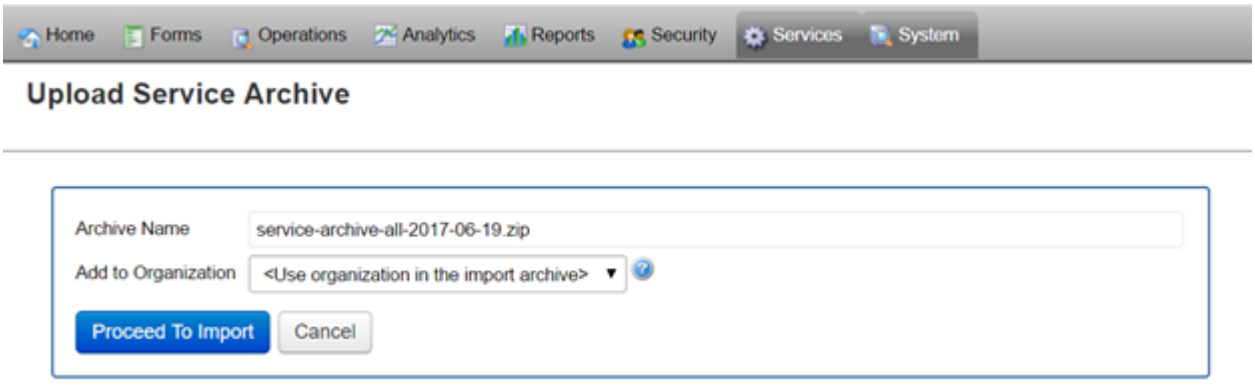
- Roll back one or several services.
- Re-create one or several services in another Manager.
- Promote services to another environment.

To import one or several services:

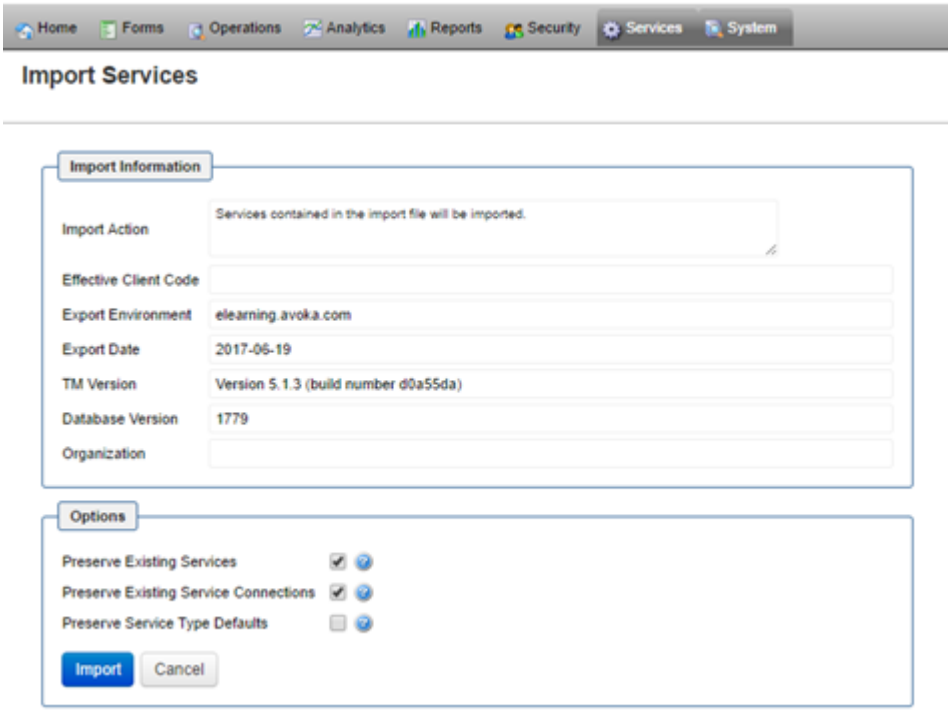
1. Select **Services > All Services.**
2. Click **Import.**



3. Click **Choose File** and choose a service archive ZIP file, which has been previously exported.
4. Click **Upload.**



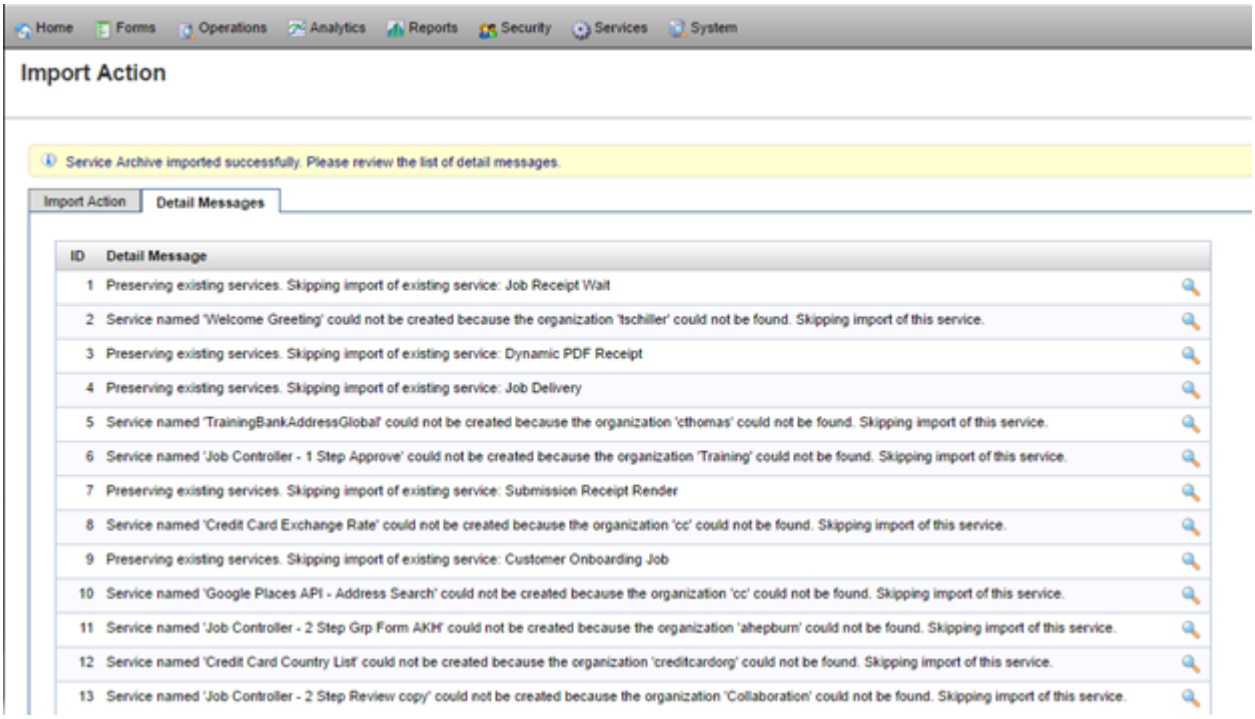
5. Select an organization that you want to add the services to or use the organization already configured in the import archive file.
6. Click **Proceed To import**.
7. Review the **Import Options** as shown below.



8. Select the **Preserve Existing Services**, so any services that already exist in the target environment will not be modified in any way. Only new (to the target environment) services will be imported.

> If you import a new version of a service, clear the **Preserve Existing Services**, then manually activate this service version and update the configurations that refer to the old version of the service to point to the new version.

9. Select the **Preserve Existing Service Connections**, so any service connections that already exist in the target environment will not be modified in any way. Only new (to the target environment) service connections will be created.
10. Select the **Preserve Service Type Defaults**, so none of the imported services will be promoted to be the default for their service types, regardless of their default status in the source environment. Otherwise, the service type default flag will be imported, and existing services may be denoted.
11. Click **Import** to complete the process. The expected result page looks like this.



12. Click **Close** and check the list of services to verify the imported services have been added.

Next, learn how to migrate services.

Core Global Services Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager comes with a set of standard services, which are designed to perform various specific functions. Services are grouped by service types and can be customized to suit your requirements.

Manager allows you to view, create new and manage the existing core global services.

Next, learn about other services.

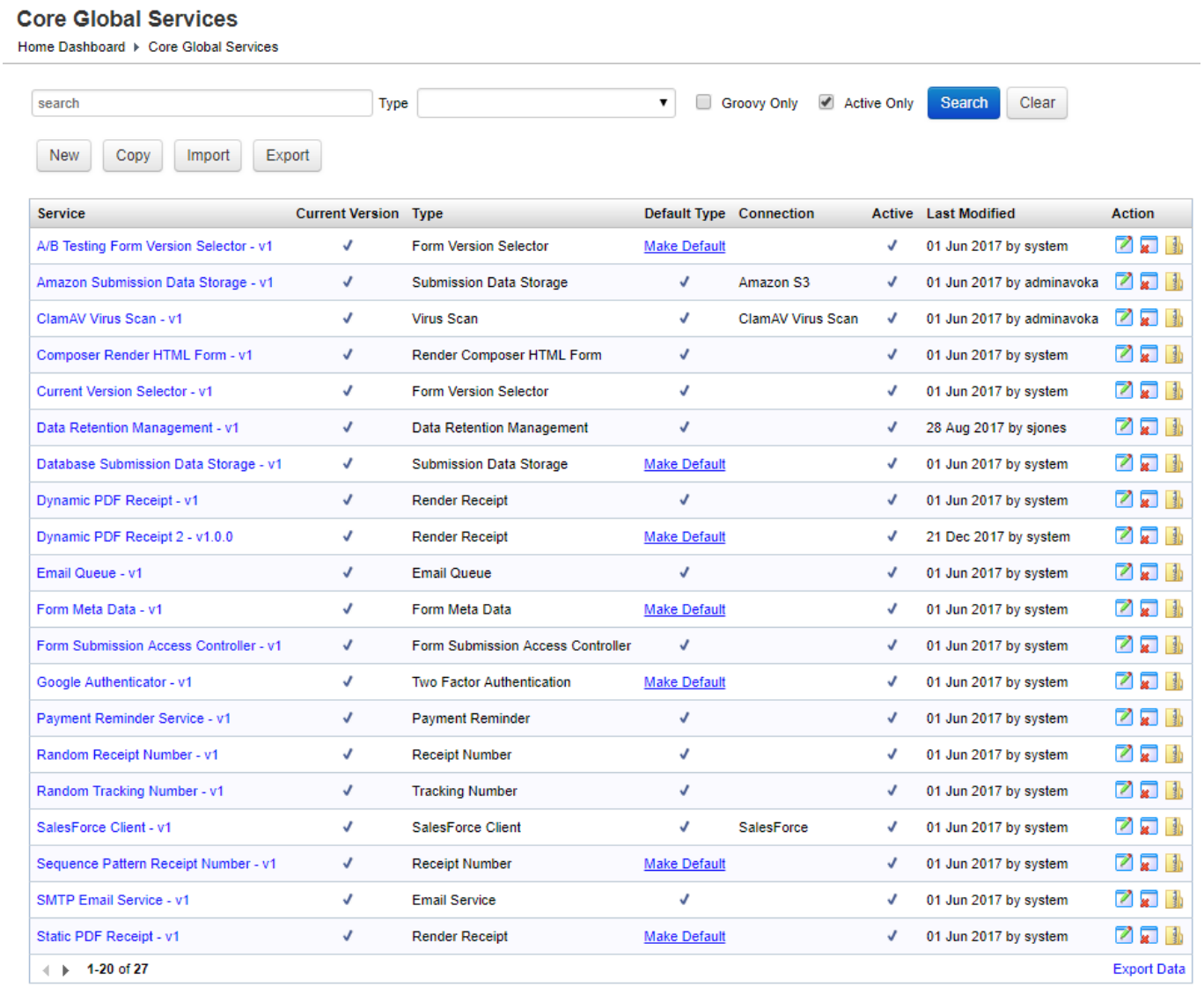
View Core Global Services

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view all core global services, so you can easily find an existing service.

To view core global services, select **Services > Core Global Services.**

Core global services are shown page by page, with the maximum number of records per page configured in the users preferences.



To filter or search the services by one or more criteria, specify the following settings and click **Search**:

- Service name or part of a name
- Service type
- Groovy Only to see only **Groovy**[1] services
- Active Only

Click **Clear** to reset the search criteria.

You can check the following service details:

- Service name
- Current Version
- Type of a service
- Default Type represents whether this service is the default service for this type of service. For example, the Dynamic PDF Receipt – v1 service is the default service of the Render Receipt service type.
- Connection is the service connection used by this service, if configured.
- Active shows whether this service is active or not. It must be active to be available.

> It is recommended to make a service inactive rather than remove it when you don't need it any more.

- Last Modified shows the date when this service was last modified and by whom.

You can also perform the following actions:

- Create a service
- Configure a service
- Copy a service
- Remove a service
- Export services
- Import services
- Run a unit test. This option is only available for groovy, scheduled and virus scan services.
- Click **Make Default** to mark the selected service as a default one for all services of this type.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

---

[1] Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

A/B Testing Form Version Selector Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

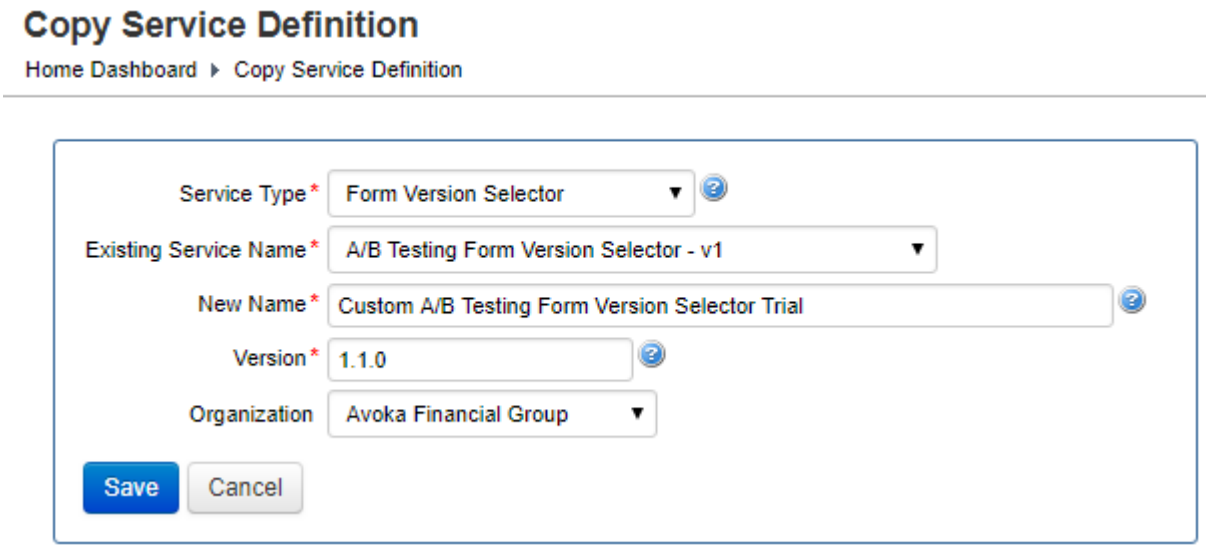
Manager comes with the A/B Testing Form Version Selector core global service, which you can link to a form that needs to support **A/B testing**[1]. This service extends the current version selector service.

By default, when a user selects a form, the A/B Testing Form Version Selector service chooses a form to render from all its form versions. Usually, you may need to test only the current version against the previous version.

> It's recommended to copy the default A/B Testing Form Version Selector service before using it for A/B testing.

To copy the A/B Testing Form Version Selector service:

1. Select **Services > Core Global Services**.
2. Copy the service.
3. Select a service of the **Form Version Selector** service type from the **Service Type** dropdown.
4. Select **A/B Testing Form Version Selector** from the **Existing Service Name** dropdown.



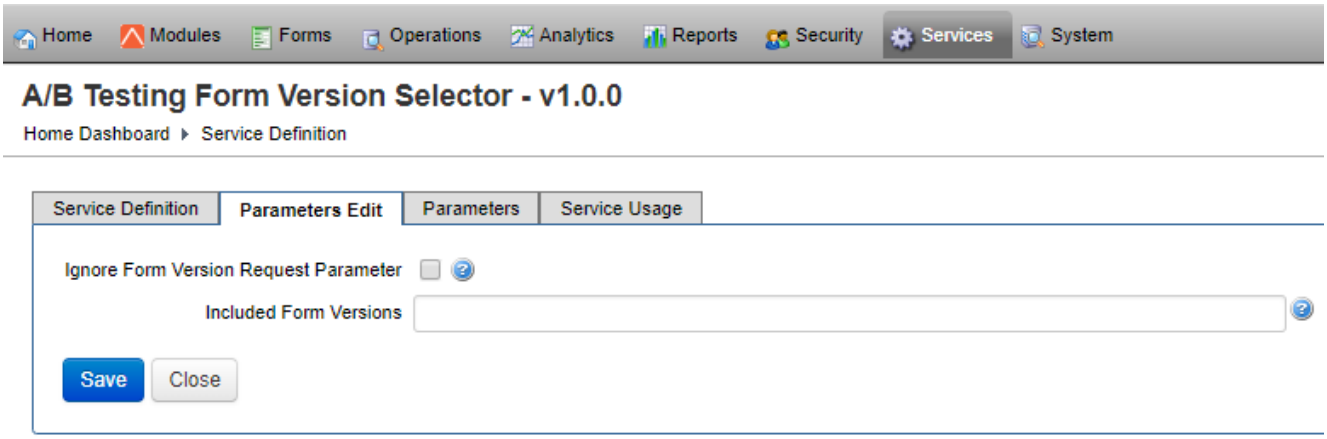
5. Click **Save** to update the changes.

To customize the A/B Testing Form Version Selector service:

---

[1] A/B testing (bucket tests or split-run testing) is a randomized experiment with two variants, A and B. It includes application of statistical hypothesis testing or "two-sample hypothesis testing" as used in the field of statistics. A/B testing is a way to compare two versions of a single variable, typically by testing a subject's response to variant A against variant B, and determining which of the two variants is more effective.

1. Select **Services > Core Global Services**. If the option isn't available, follow the above steps for copying the service.
2. Locate the service and click **Edit**.
3. Configure the standard service settings.
4. Click the **Parameters Edit** tab to view and edit the configuration.



5. Select the **Ignore Form Version Request Parameter** checkbox to always use the current form version regardless of the value of the **`tmFormVersion`** request parameter.
6. In the **Included Form Versions** text field, enter a comma-separated list of form versions to include in the selection. Leave it blank to include all versions.
7. Click **Save** to update the changes.

The global A/B Testing Form Version Selector service can also be customized from your form.

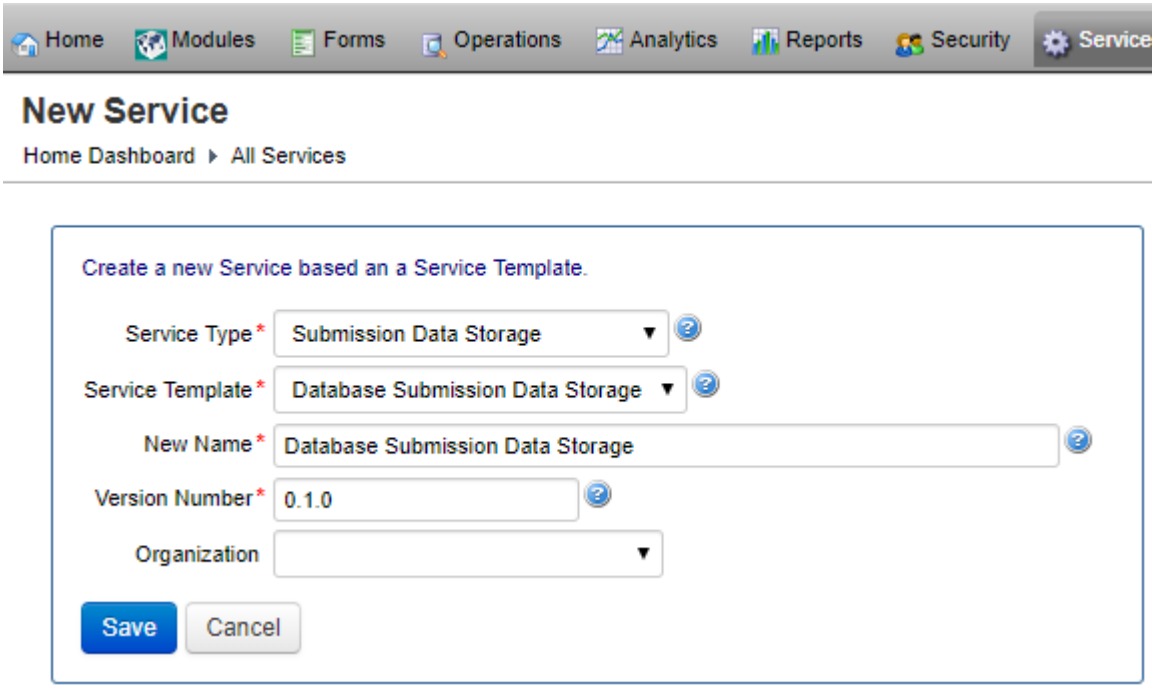
Database Submission Data Storage Service

Journey Manager (JM) | System Manager / DevOps | 20.05

Manager provides the database submission data storage service, which is a service for storing form submissions in a database. This service is the default submission data storage service for on premise installations, so it's created and configured for you and you don't need to do anything. If Manager is installed on AWS cloud, this service is set as the default, so you need to configure the Amazon submission data storage service and set it as the default. For Azure cloud installations, the default service is already set to the Azure submission data storage service.

To create the database submission data storage service:

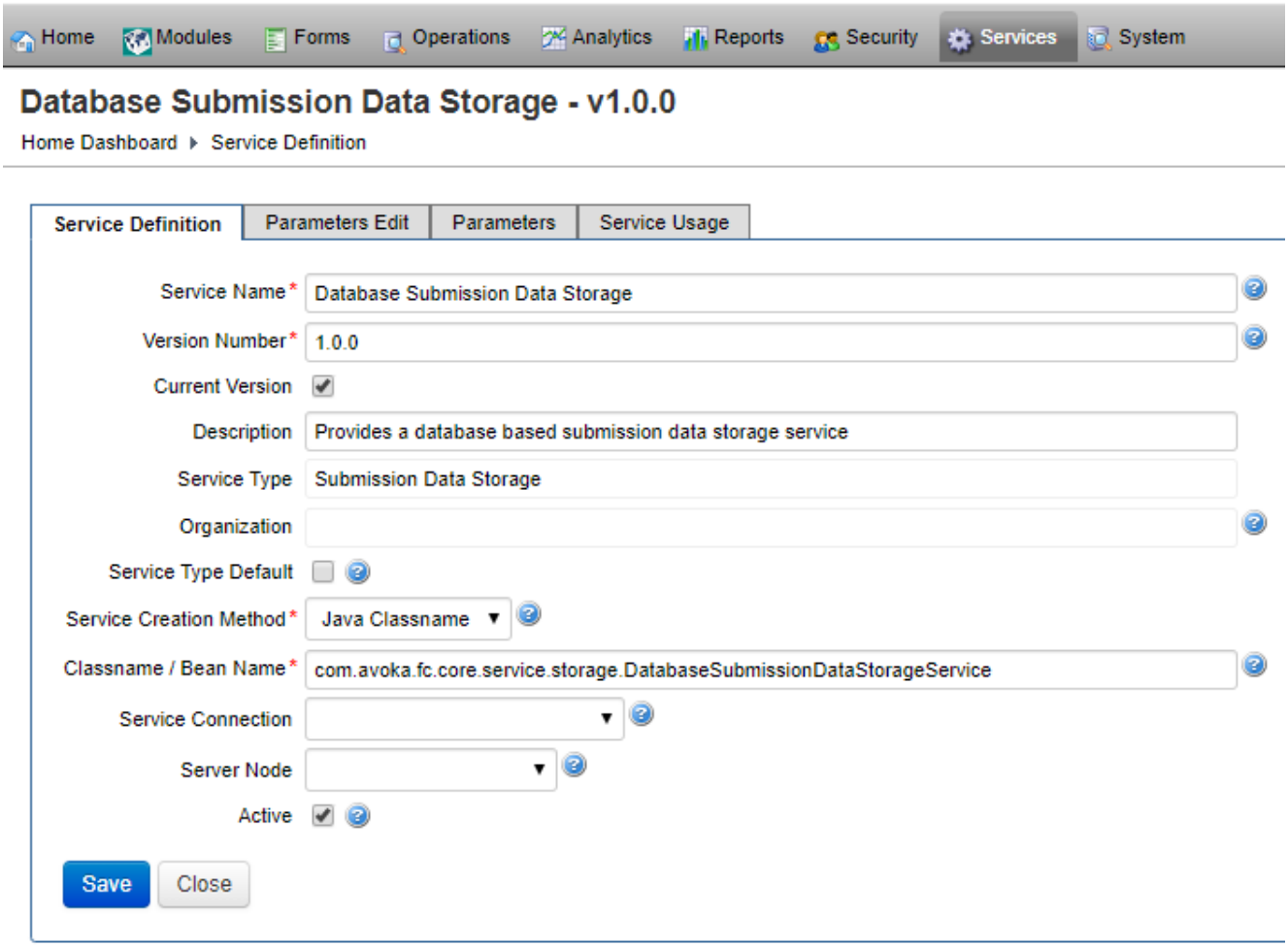
1. Select **Services > Core Global Services** and click **New**.



2. Select **Submission Data Storage** from the **Service Type** dropdown list.
3. Select **Database Submission Data Storage** from the **Service Template** dropdown list.
4. Provide a unique service name in the **New Name** field.
5. Specify a version in the **Version Number** field.
6. Select an organization from the **Organization** dropdown list.
7. Click **Save** to update the changes.

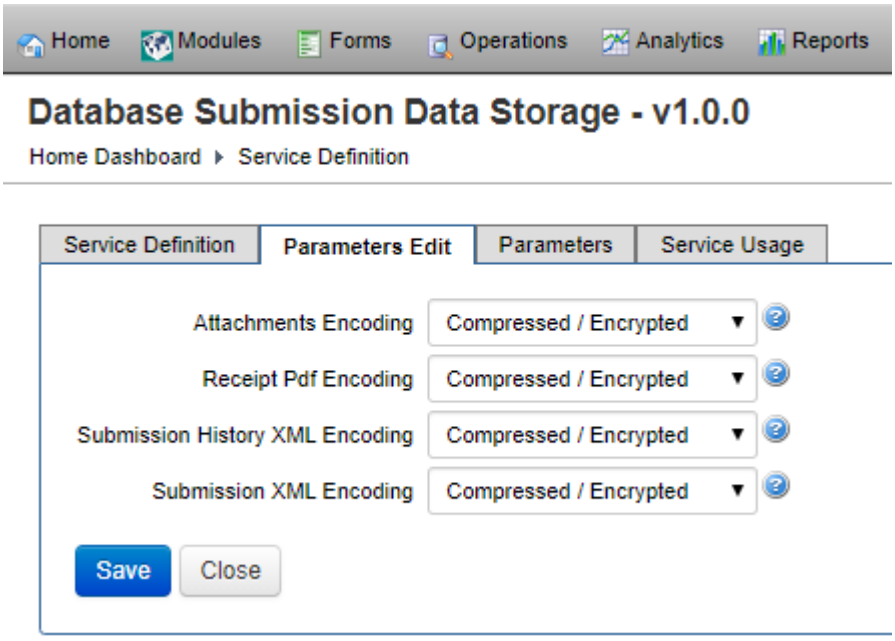
To configure the database submission data storage service:

1. Select **Services > Core Global Services**.
2. Locate the **Database Submission Data Storage** service and click **Edit**.
3. Click the **Service Definition** tab to configure the service.



The **Service Creation Method** and **Classname / Bean Name** fields are only available to users with appropriate permissions.

4. Click **Save** to update the changes.
5. Click the **Parameters Edit** tab to configure the service parameters.



The **Parameters** fields are only available to users with appropriate permissions.

6. Select a file attachments storage encoding from the **Attachments Encoding** dropdown list, which can be:
   - **Compressed / Encrypted**
   - **Compressed / Plain Text**
   - **Uncompressed / Encrypted**
   - **Uncompressed / Plain Text**
7. Select a PDF receipt storage encoding from the **Receipt PDF Encoding** dropdown list.
8. Select a submission history XML storage encoding from the **Submission History XML Encoding** drop-down list.
9. Select a submission XML storage encoding from the **Submission XML Encoding** dropdown list.
10. Click **Save** to update the changes.

Next, learn about the Amazon submission data storage service.

File System Submission Data Storage Service

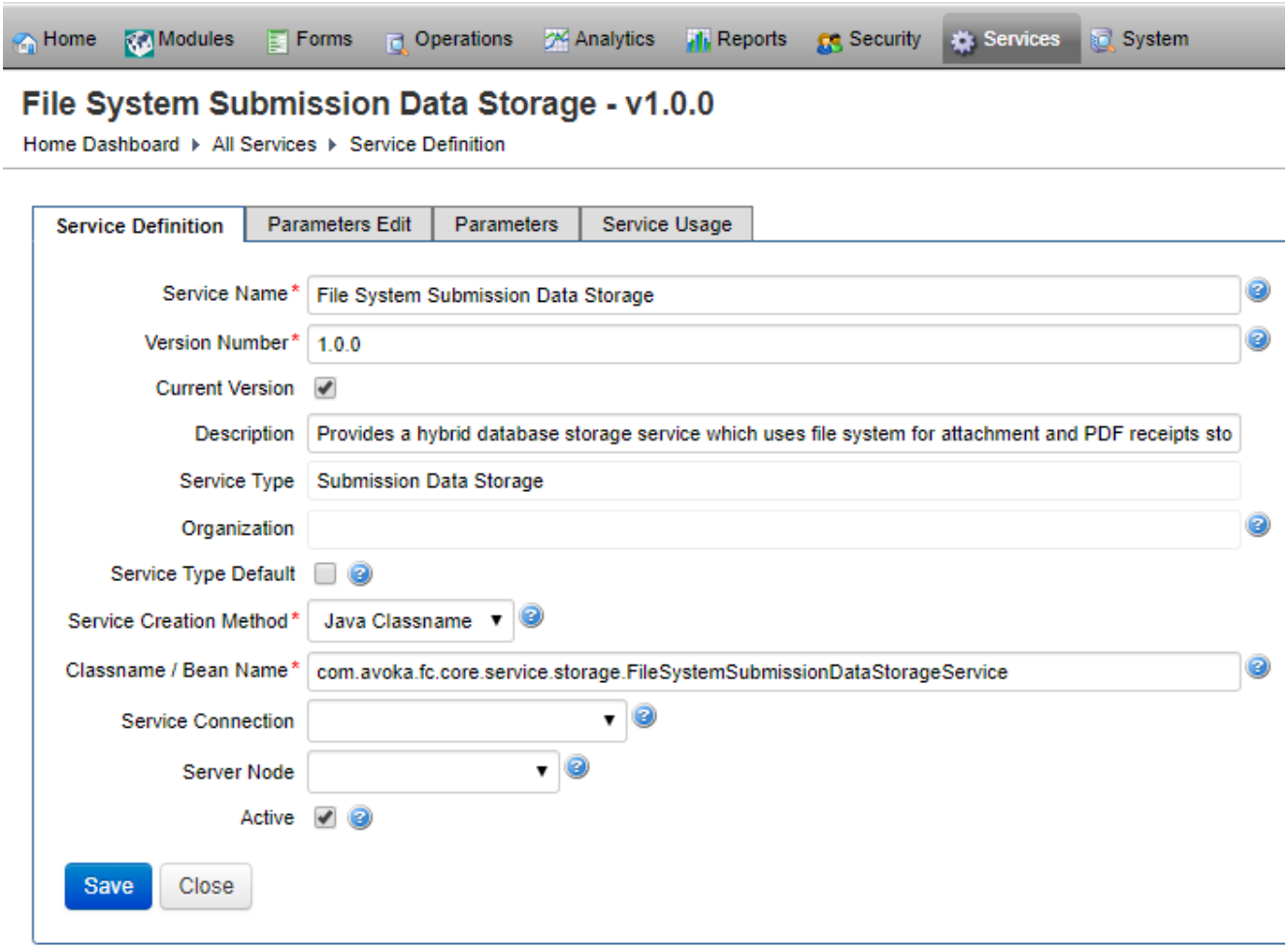
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides the file system submission data storage service, which is a hybrid database storage service that uses file system for attachment and PDF receipts storage. This service is created when your Manager is installed on premise.

> Cloud specific services and some service connections are not available and can't be manually created, unless the deployment environment matches your installation mode.
>
> To check the installation mode of your Manager server, select **System > System Info**, find the **Build Properties** section, and check the **`deployment.env`** value. For more information, see system info.

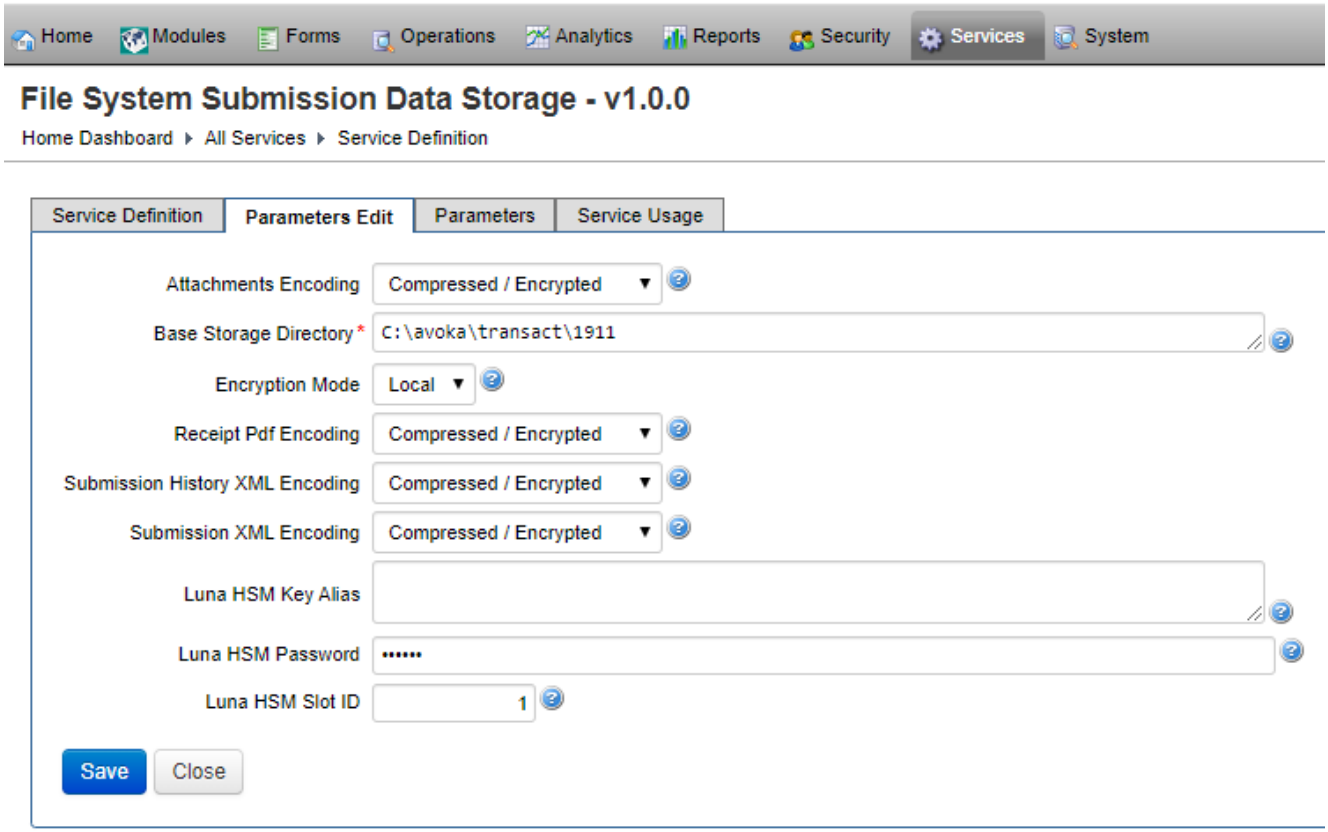
To configure the file system submission data storage service:

1. Select **Services > Core Global Services**.
2. Locate the **File System Submission Data Storage** service and click **Edit**.
3. Click the **Service Definition** tab to configure the service.



The **Service Creation Method** and **Classname / Bean Name** fields are only available to users with appropriate permissions.

4. Click **Save** to update the changes.
5. Click the **Parameters Edit** tab to configure the service parameters.



The **Parameters** fields are only available to users with appropriate permissions.

6. Select a file attachments storage encoding from the **Attachments Encoding** dropdown list, which can be:
   - **Compressed / Encrypted**
   - **Compressed / Plain Text**
   - **Uncompressed / Encrypted**
   - **Uncompressed / Plain Text**
7. Specify a base file storage directory for submission attachments in the **Base Storage Directory** field. This is a mandatory field and needs to be set up correctly.

8. Select a submission data encryption mode from the **Encryption Mode** dropdown list, which can be:
    - **Local**
    - **HSM**
9. Select a PDF receipt storage encoding from the **Receipt PDF Encoding** dropdown list.
10. Select a submission history XML storage encoding from the **Submission History XML Encoding** drop-down list.
11. Select a submission XML storage encoding from the **Submission XML Encoding** dropdown list.
12. Provide an Amazon CloudHSM SafeNet Luna HSM Key Alias in the **Luna HSM Key Alias** field.
13. Provide an Amazon CloudHSM SafeNet client password in the **Luna HSM Password** field.
14. Enter an Amazon CloudHSM SafeNet Luna HSM Slot ID in the **Luna HSM Slot ID** field.
15. Click **Save** to update the changes.

Next, learn about the database submission data storage service.

Amazon Submission Data Storage Service

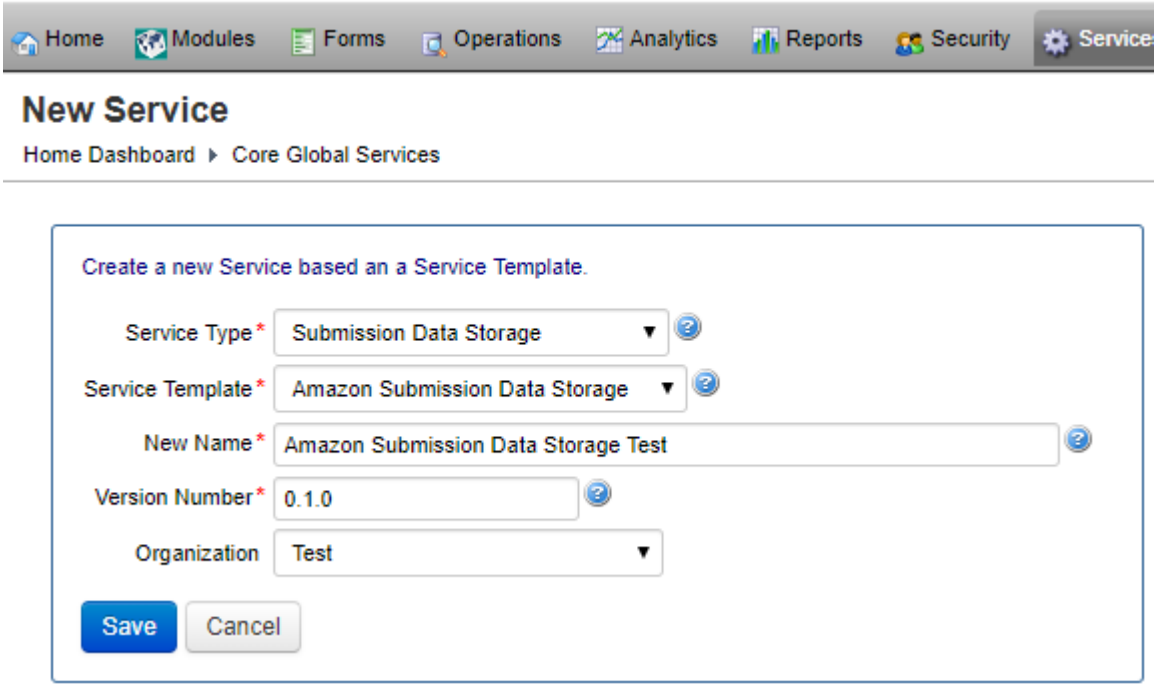
Journey Manager (JM) | System Manager / DevOps | 20.05

Manager provides the Amazon submission data storage service to persist form submissions and related transactions in the **AWS**[1] cloud. The service requires an Amazon service connection to connect to an AWS queue. For AWS cloud installations, the database submission data storage service is initially set as the default, so you need to configure the Amazon submission data storage service and set it as the default.

> Cloud specific services and some service connections are not available and can't be manually created, unless the deployment environment matches your installation mode.
>
> To check the installation mode of your Manager server, select **System > System Info**, find the **Build Properties** section, and check the **deployment.env** value. For more information, see system info.

To create the Amazon submission data storage service:

1. Select **Services > Core Global Services** and click **New**.



2. Select **Submission Data Storage** from the **Service Type** dropdown list.
3. Select **Amazon Submission Data Storage** from the **Service Template** dropdown list.
4. Provide a unique service name in the **New Name** field.

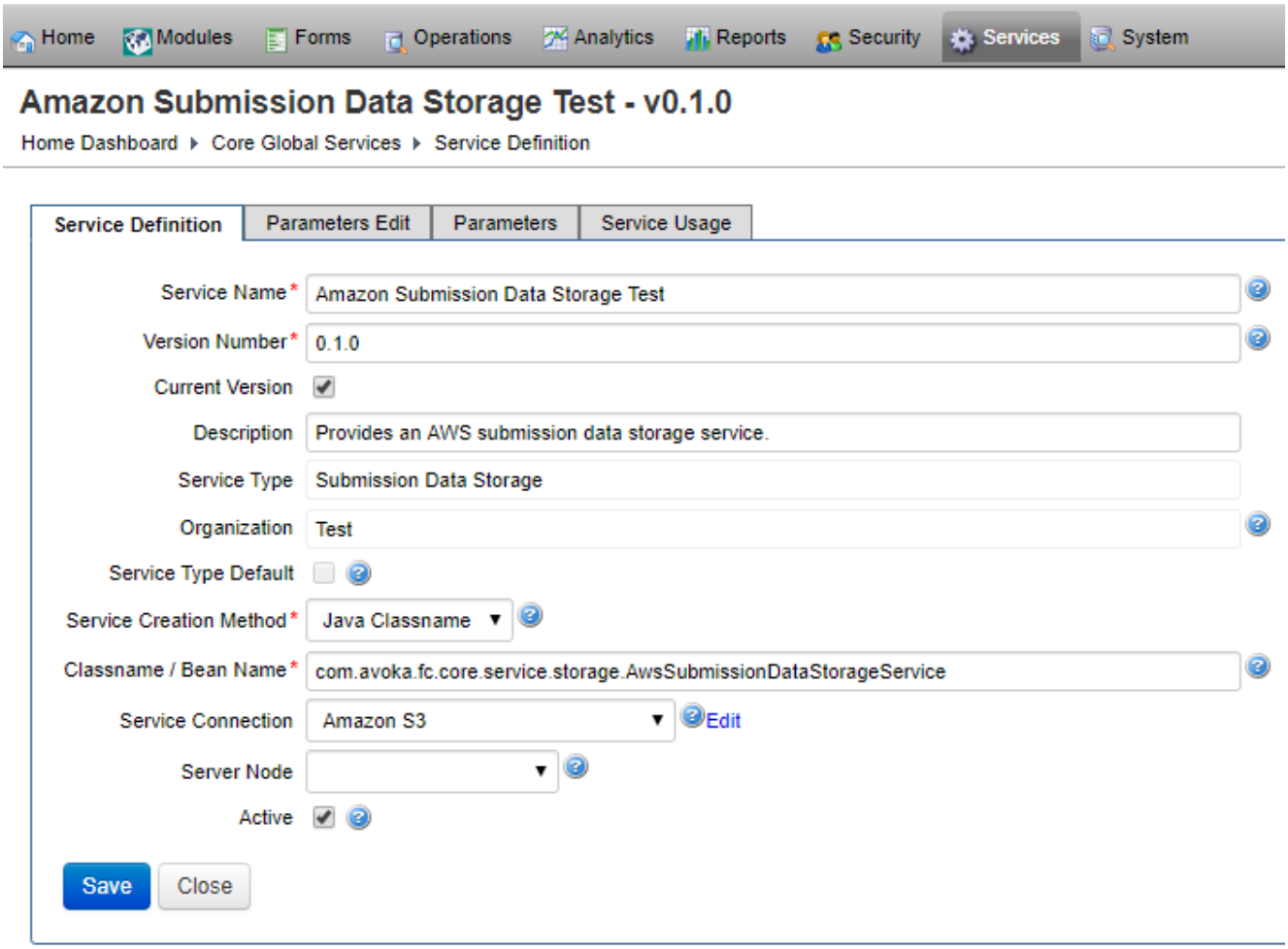
---

[1]Amazon Web Services (AWS) is a comprehensive, evolving cloud computing platform provided by Amazon. It provides a mix of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings.

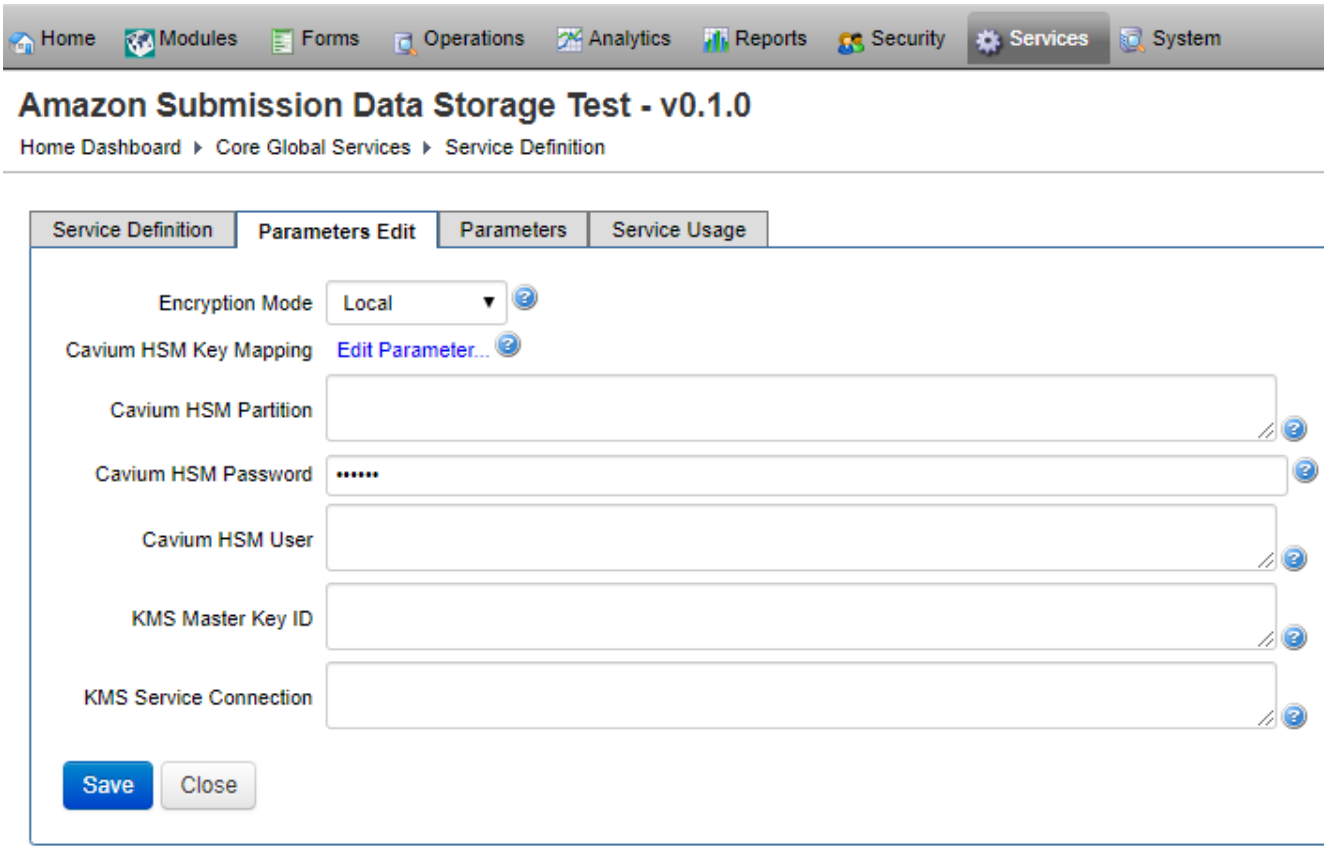
5. Specify a version in the **Version Number** field.
6. Select an organization from the **Organization** dropdown list.
7. Click **Save** to update the changes.

To configure the Amazon submission data storage service:

1. Select **Services > Core Global Services**.
2. Locate the **Amazon Submission Data Storage** service and click **Edit**.
3. Click the **Service Definition** tab to edit the standard service configuration.



4. Click the **Parameters Edit** tab to view or update the service configuration.



5. Select a submission data encryption mode from the **Encryption Mode** dropdown list, which can be Local, **KMS**[1], or Cavium **HSM**[2].
6. Provide a Cavium HSM Master Key Mappings as a **JSON**[3] document in the **Cavium Hsm Key Mapping** field.
7. Enter a Cavium HSM key alias in the **Cavium Hsm Partition** field.
8. Specify a Cavium HSM password in the **Cavium HSM Password** field.
9. Enter a Cavium HSM user in the **Cavium HSM User** field.

---

[1]AWS Key Management Service (KMS) is a managed service that makes it easy for you to create and control the encryption keys used to encrypt your data, and uses FIPS 140-2 validated hardware security modules to protect the security of your keys. AWS Key Management Service is integrated with most other AWS services to help you protect the data you store with these services. AWS Key Management Service is also integrated with AWS CloudTrail to provide you with logs of all key usage to help meet your regulatory and compliance needs.
[2]AWS CloudHSM is a cloud-based hardware security module (HSM) that enables you to easily generate and use your own encryption keys on the AWS Cloud. With CloudHSM, you can manage your own encryption keys using FIPS 140-2 Level 3 validated HSMs.
[3]JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

10. Provide an AWS Key Management System (KMS) Master Key ID used to generate transaction data keys in the **KMS Master Key ID** field.
11. Enter an AWS KMS Service Connection name in the **KMS Service Connection** field.
12. Click **Save** to update the changes.

> For Azure installations, check the Azure Submission Data Storage service that persists form submissions and related transactions.

Next, learn about the Azure Submission Data Storage service.

Azure Submission Data Storage Service

Journey Manager (JM) | System Manager / DevOps | 20.05

Manager provides the Azure submission data storage service to persist form submissions and related transactions in the **Azure**[1] Cloud Storage. The service requires an Azure service connection to connect to the cloud using the Azure Key Vault service.

This service is the default submission data storage service for Azure cloud installations. For more information, see the installation guide.

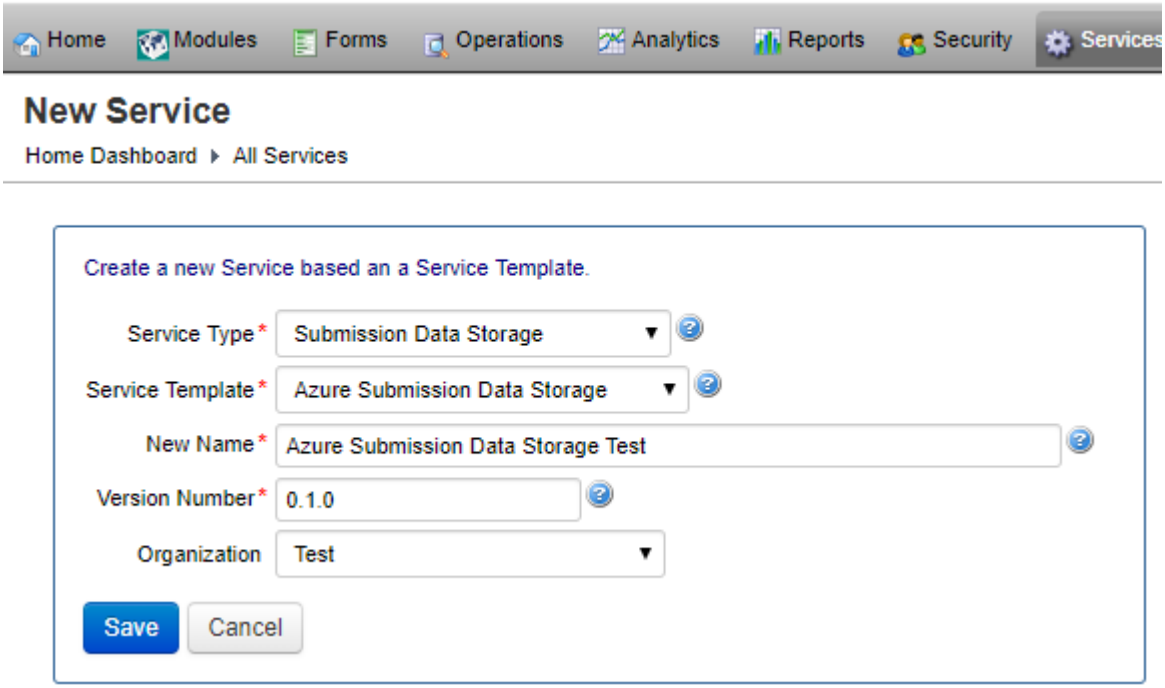
> Cloud specific services and some service connections are not available and can't be manually created, unless the deployment environment matches your installation mode.
>
> To check the installation mode of your Manager server, select **System > System Info**, find the **Build Properties** section, and check the **`deployment.env`** value. For more information, see system info.

Manager's implementation of an Azure service caches a connection to the Azure cloud service for 5 minutes to improve server performance, reduce network bandwidth, and prevent unnecessary load on the systems. However, if an error occurs when configuring the storage service, there may be a delay up to 5 minutes once the problem is rectified for the changes to have effect.

To create the Azure submission data storage service:

[1]Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers.

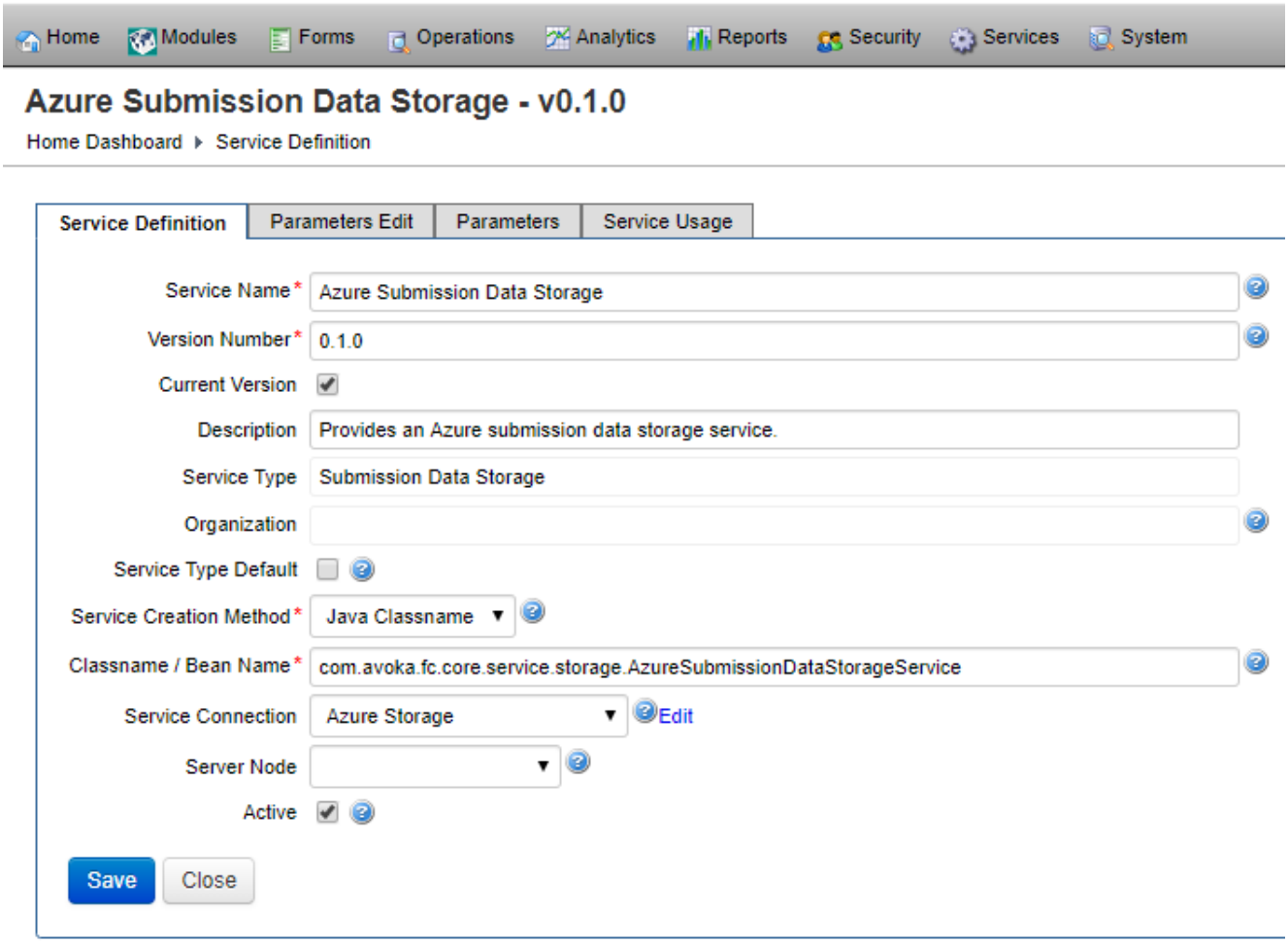
1. Select **Services > Core Global Services** and click **New**.



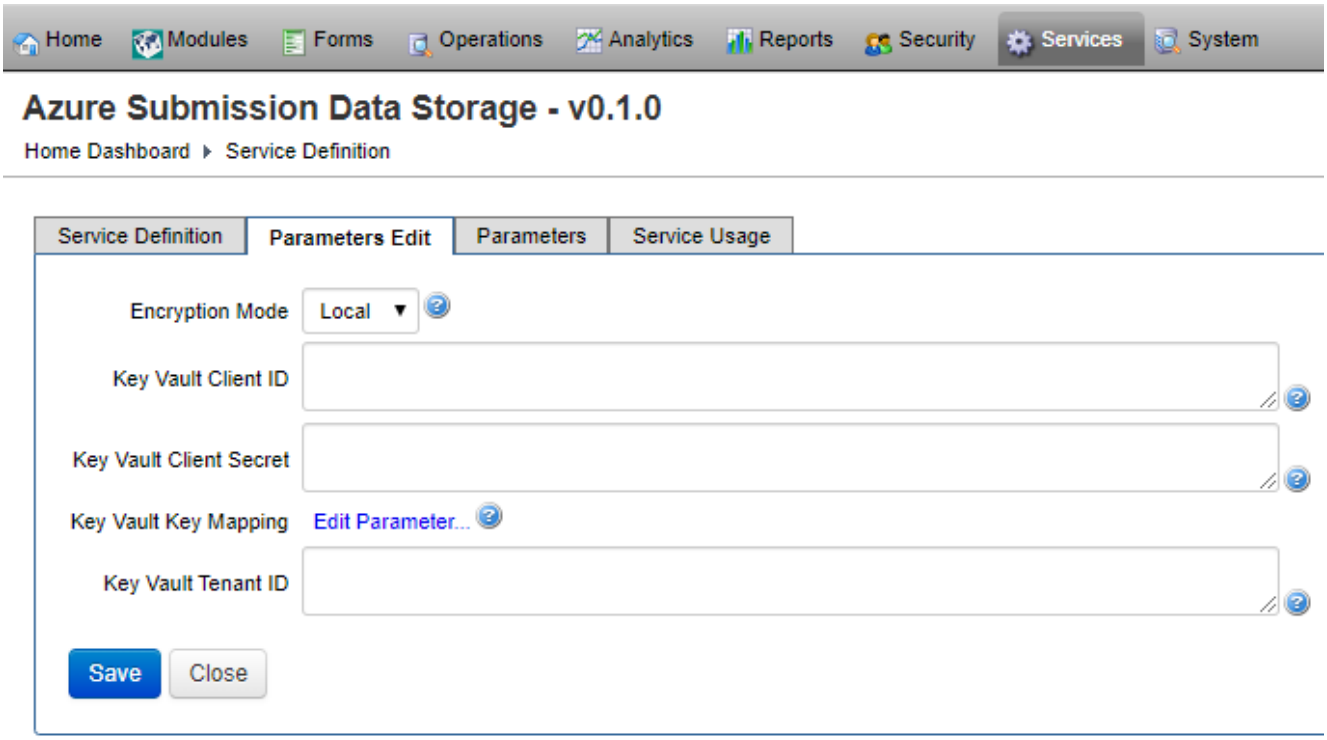
2. Select **Submission Data Storage** from the **Service Type** dropdown list.
3. Select **Azure Submission Data Storage** from the **Service Template** dropdown list.
4. Provide a unique service name in the **New Name** field.
5. Specify a version in the **Version Number** field.
6. Select an organization from the **Organization** dropdown list.
7. Click **Save** to update the changes.

To configure the Azure submission data storage service:

1. Select **Services > Core Global Services**.
2. Locate the **Azure Submission Data Storage** service and click **Edit**.
3. Click the **Service Definition** tab to configure the service.



4. Select the Azure Storage service connection from the **Service Connection** dropdown list, which the service will use to publish the events to. Alternatively, click the **Edit** link to configure the selected service connection.
5. Click **Save** to update the changes.
6. Click the **Parameters Edit** tab to configure the service parameters used for encryption key management. For more information, see Microsoft Azure documentation.



7. Select a submission data encryption mode from the **Encryption Mode** dropdown list, which can be **Local**, or **Azure**.
8. Provide a key vault client id in the **Key Vault Client ID.** field.
9. Enter a key vault client secret in the **Key Vault Client Secret** field.
10. Specify a key vault master key mappings as **JSON**[1] document in the **KeyVault Master Key Mapping** field.
11. Enter a key vault tenant id in the **Key Vault Tenant Id** field.
12. Click **Save** to update the changes.

For AWS installations, check the Amazon Submission Data Storage service that persists form submissions and related transactions.

Next, learn how to view all core global services.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

System Event Publisher Service

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager allows you to create and configure a System Event Publisher as a service to publish events to various queues, such as **Amazon**[1] or **Azure**[2]. Different System Event Publishers have the same configuration with the exception to a service connection.

> You need to create and configure a specific service connection before you can configure your System Event Publisher.

Manager comes with the following ready-to-use System Event Publishers:

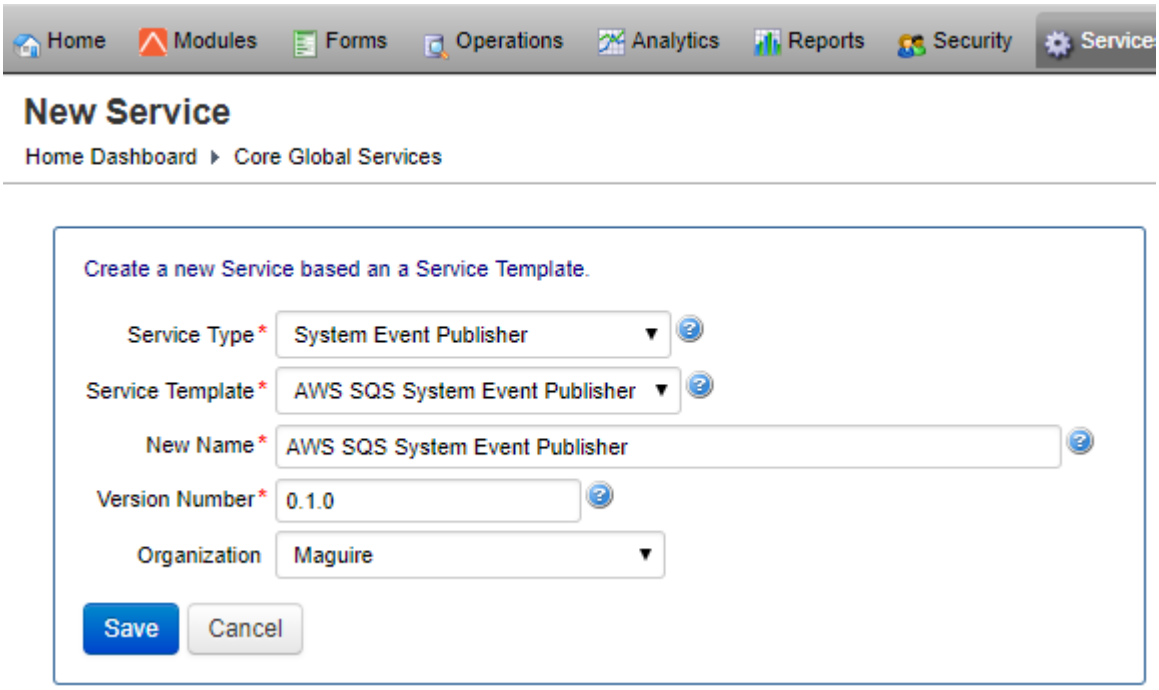
- AWS SQS System Event Publisher service
- Azure System Event Publisher service

> You must be a Journey Manager administrator to create or configure this functionality.

To create a System Event Publisher service:

1. Select **Services > Core Global Services**.
2. Click **New**.

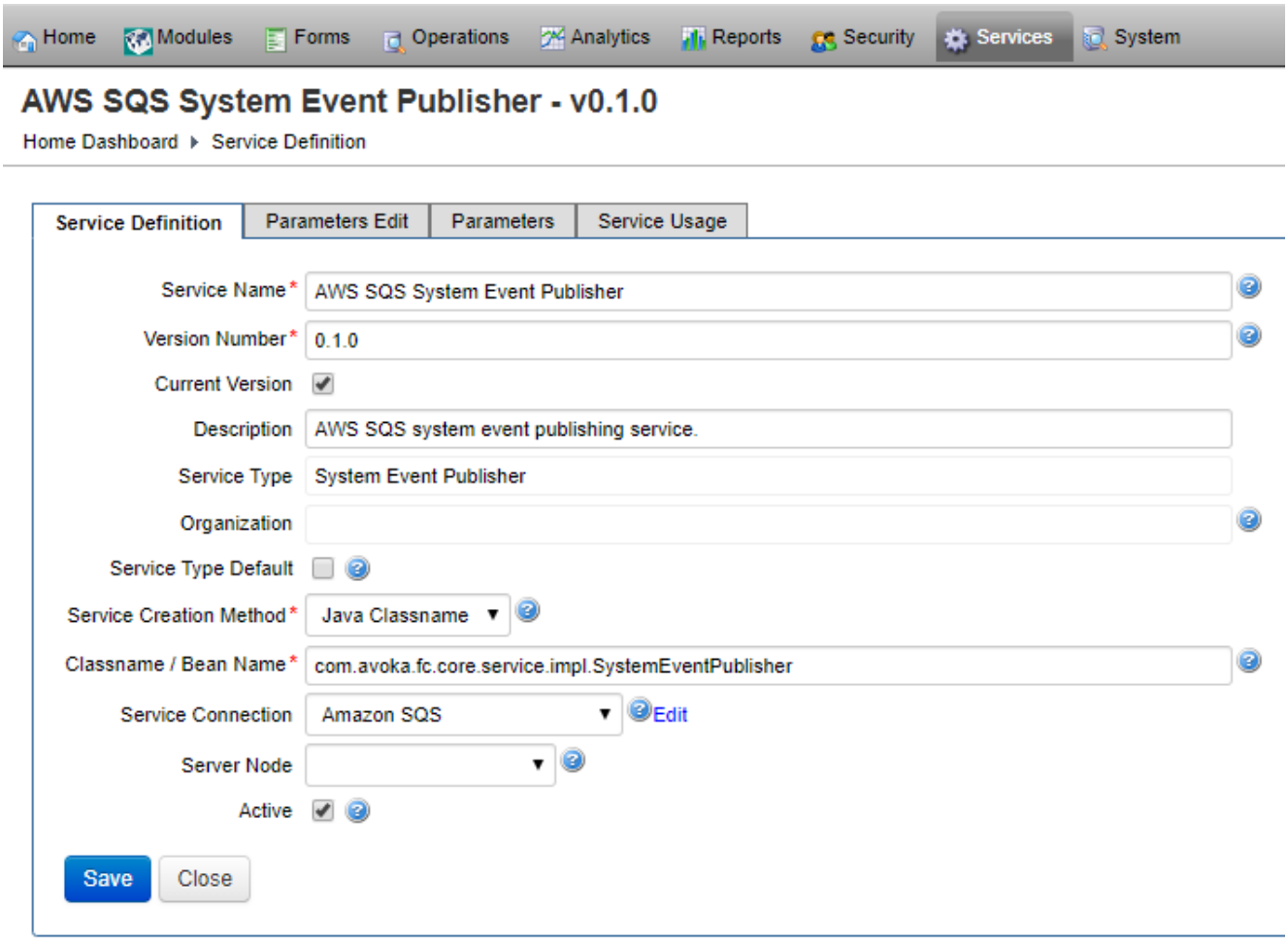
[1]Amazon Web Services (AWS) is a comprehensive, evolving cloud computing platform provided by Amazon. It provides a mix of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings.
[2]Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers.



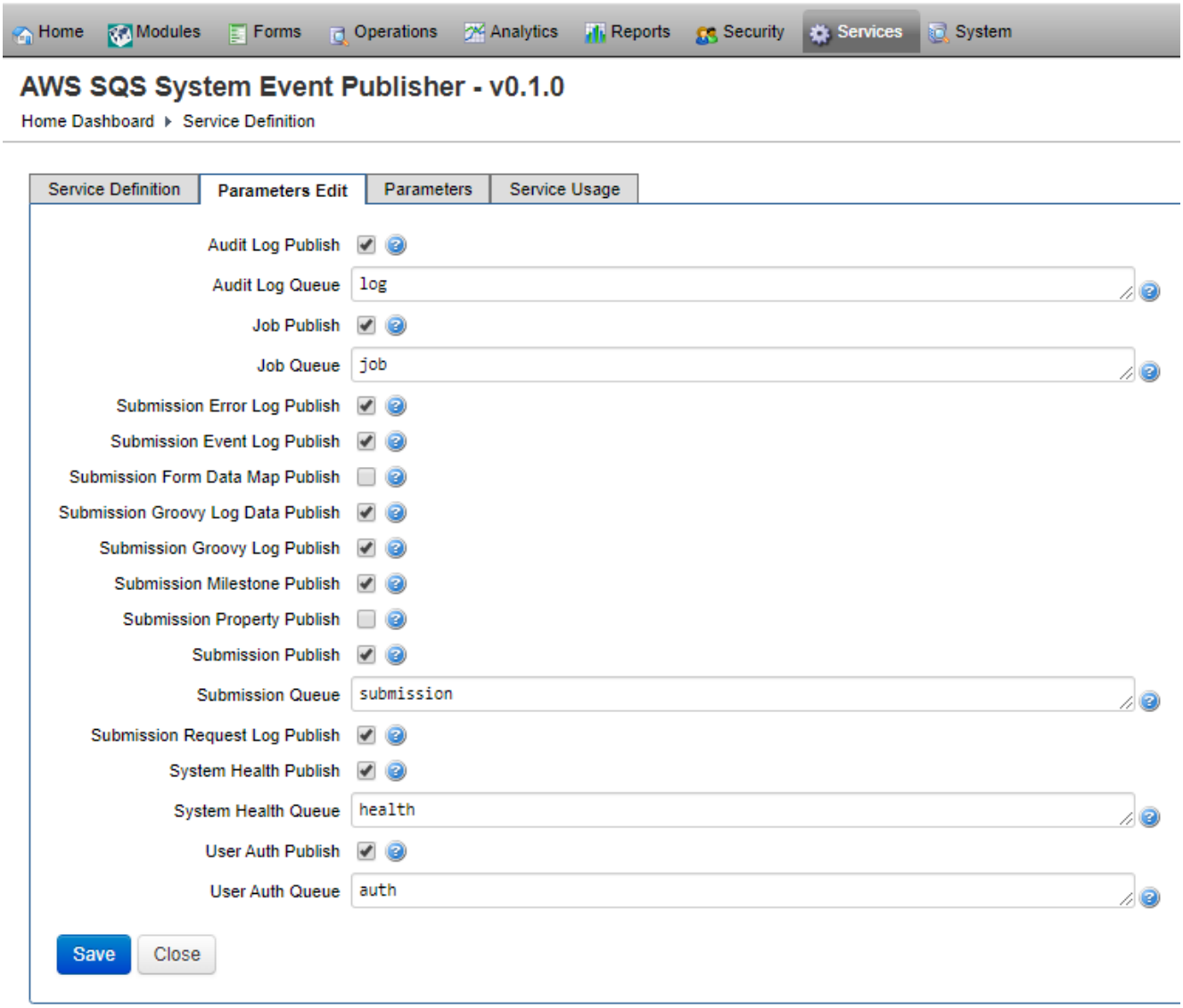
3. Select the **System Event Publisher** service type from the **Service Type** dropdown list.
4. Select a System Event Publisher template from the **Service Template** dropdown list to automatically pre-fills the new service with the default parameters.
5. Edit the service name in the **New Name** field, which is by default the name of a service template.
6. Enter a unique name in the **Version Number** field.
7. Select an organization for this service from the **Organization** dropdown list, or leave it blank for the global service.
8. Click **Save** to update the changes.

To configure a System Event Publisher service:

1. Select **Services > Core Global Services**.
2. Locate a System Event Publisher service and click **Edit**.
3. Select the **Service Definition** tab to configure the service.



4. Select a service connection from the **Service Connection** dropdown list, which the service will use to publish the events. Alternatively, click the **Edit** link to configure the selected service connection.
5. Click **Save** to update the changes.
6. Select the **Parameters Edit** tab to configure the service parameters.



You can edit the parameters in the **Parameters** tab as well.

7. Select **Audit Log Publish** to publish audit log events. It is not selected by default.
8. Specify a queue name in the **Audit Log Queue** field to publish audit log events to, or leave it blank to use the default queue specified in the **Queue Name** of this service connection.
9. Select **Job Publish** to publish job events.
10. Specify a queue name in the **Job Queue** field to publish job events to, or leave it blank to use the default queue specified in the **Queue Name** of this service connection.
11. Select **Submission Error Log Publish** to publish the submission error log as part of a submission event. The **Submission Publish** checkbox below must also be selected. It is selected by default.
12. Select **Submission Event Log Publish** to publish the submission event log as part of a submission event. The **Submission Publish** checkbox below must also be selected. It is selected by default.

13. Select **Submission Form Data Map Publish** to publish the submission form data map as part of a submission event. The **Submission Publish** checkbox must also be selected. It is selected by default.
14. Select **Submission Groovy Log Data Publish** to publish added submission groovy service log data as part of a submission event. The **Submission Publish** checkbox must also be ticked. It is selected by default.
15. Select **Submission Groovy Log Publish** to publish added submission groovy service logs as part of a submission event. The **Submission Publish** checkbox below must also be selected. It is selected by default.
16. Select **Submission Milestone Publish** to publish added submission milestones as part of a submission event. The **Submission Publish** checkbox below must also be selected. It is not selected by default.
17. Select **Submission Property Publish** to publish submission properties as part of a submission event. It is not selected by default.

    You also must select the **Submission Publish** checkbox below for the **Submission Property Publish** setting to take effect.

    Adding all properties to the event can significantly increase its size, which, in turn, may cause a system event publisher delivery error when using AWS. It depends on the underlying queue configuration. For AWS, if a message size exceeds the maximum allowed, an error will occur, for example, **`One or more parameters are invalid. Reason: Message must be shorter than 262144 bytes`**.

18. Select **Submission Publish** to publish submission events. It is selected by default.
19. Specify a queue name in the **Submission Queue** field to publish submission events to, or leave it blank to use the default queue specified in the **Queue Name** of this service connection.
20. Select **Submission Request Log Publish** to publish added submission request log as part of a submission event. The **Submission Publish** checkbox below must also be selected. It is selected by default.
21. Select **System Health Publish** to publish hourly system health events. It is not selected by default.
22. Specify a queue name in the **System Health Queue** field to publish system health events to, or leave it blank to use the default queue specified in the **Queue Name** of this service connection.
23. Select **User Auth Publish** to publish user authentication events. | 18.11
24. Specify a queue name in the **User Auth Queue** field to publish authentication events to, or leave it blank to use the default queue specified in the **Queue Name** of this service connection. | 18.11
25. Click **Save** to update the changes.

Next, learn how to configure an AWS SQS System Event Publisher service.

AWS SQS System Event Publisher Service

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager comes with the AWS SQS System Event Publisher service that allows you to publish events and elements to **AWS SQS**[1] queues. The service uses the AWS Security Token Service (STS) protocol to authenticate the AWS STS Service Connection. However, you can use the standard AWS SQS Service Connection as well.

> You must be a Journey Manager administrator to create or configure this functionality.

To create and configure the AWS SQS System Event Publisher service:

1. Create and configure the AWS SQS service connection or the AWS STS service connection for the STS authentication protocol.
2. Configure the AWS SQS System Event Publisher service using this service connection.

> AWS SQS has a maximum message size, so if an event you're trying to publish is larger than that, you'll get an error. For more information, see https://aws.amazon.com/sqs/faqs/

Next, learn how to configure an Azure system event publisher service.

[1] Amazon Simple Queue Service (Amazon SQS) is a pay-per-use web service for storing messages in transit between computers. Developers use SQS to build distributed applications with decoupled components without having to deal with the overhead of creating and maintaining message queues.

Azure System Event Publisher Service

Journey Manager (JM) | System Manager / DevOps | 20.05

Manager comes with the Azure System Event Publisher service that allows you to publish events and elements to queues stored in **Azure**[1] Storage.

> You must be a Journey Manager administrator to create or configure this functionality.

To create and configure the Azure System Event Publisher service:

1. Create and configure the Azure Storage service connection.
2. Configure the Azure System Event Publisher service using this service connection.

Next, learn how to configure an AWS SQS system event publisher service.

[1]Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers.

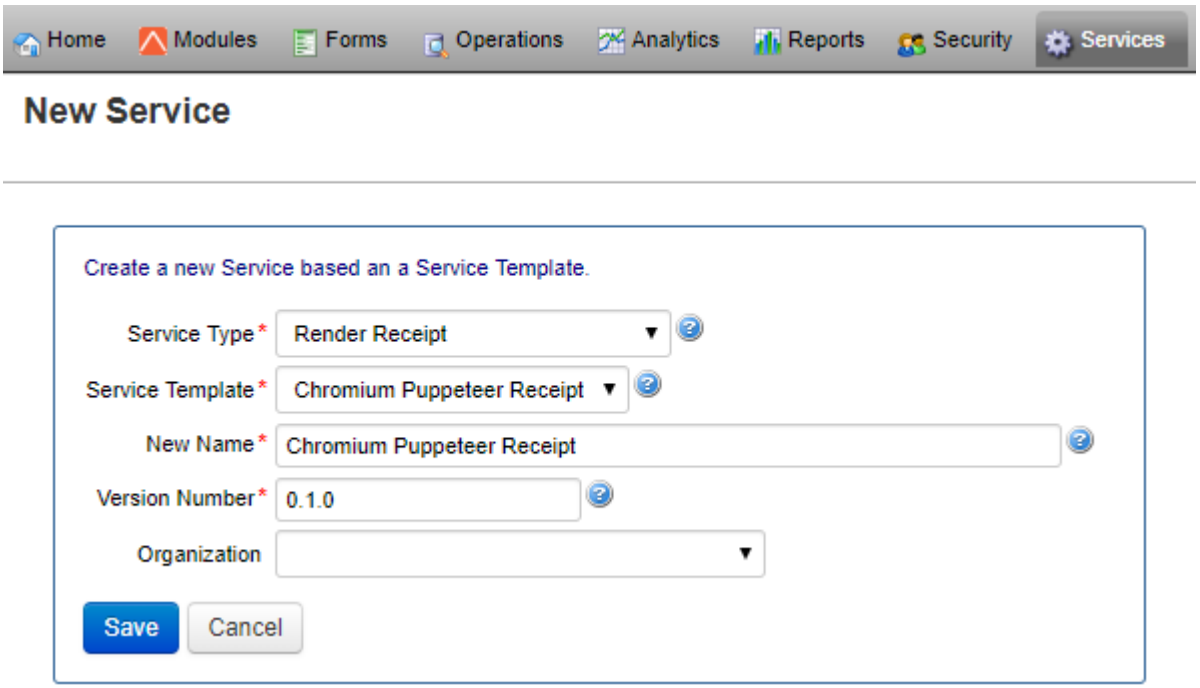
Chromium Puppeteer Receipt Service

Journey Manager (JM) | System Manager / DevOps | 19.05

Manager provides the Chromium Puppeteer Receipt service for generating dynamic PDF receipts using **Puppeteer**[1], which supports more modern styling options.

To create the Chromium Puppeteer Receipt service:

1. Select **Services > Core Global Services**. and click **New**.



2. Select **Render Receipt** from the **Service Type** dropdown list.
3. Select **Chromium Puppeteer Receipt** from the **Service Template** dropdown list.
4. Update the service name in the **New Name** field.
5. Update the service version in the **Version Number** field.
6. Select an optional organization from the Organization dropdown list.

> If organization is blank, the service will be global, otherwise it will be available to the selected organization only.

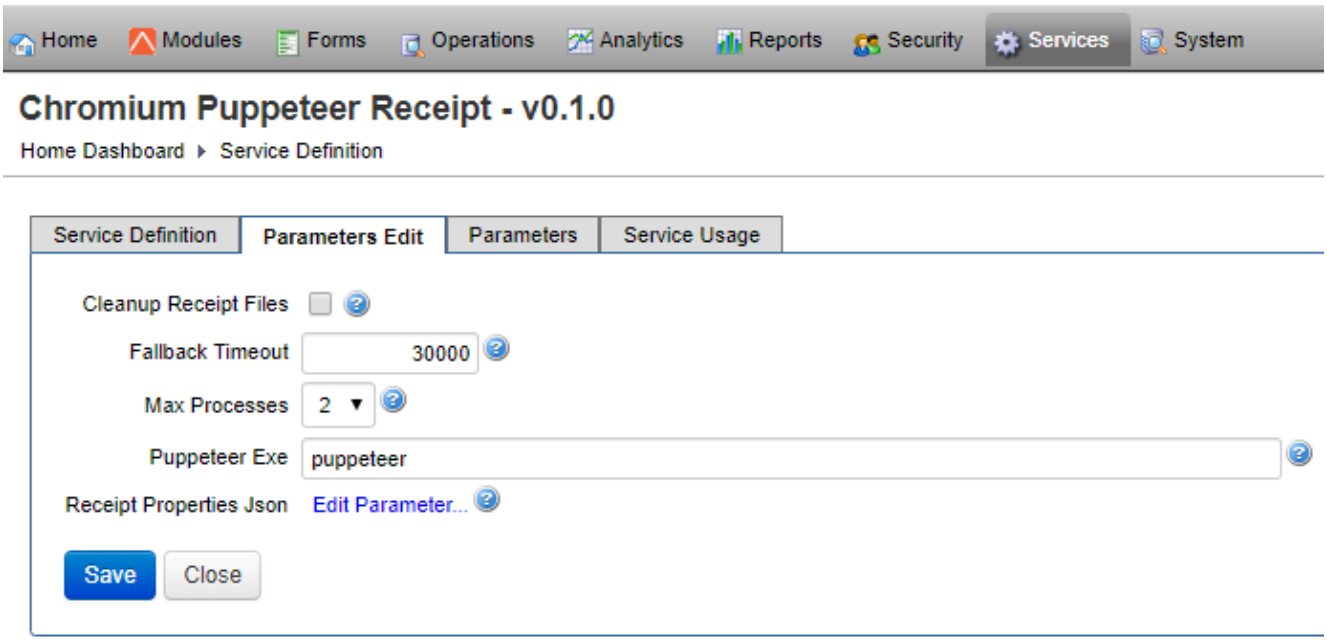
7. Click **Save** to update the changes.

To configure the Chromium Puppeteer Receipt service:

---

[1]Puppeteer is a Node library which provides a high-level API to control headless Chrome or Chromium over the DevTools Protocol. It can also be configured to use full (non-headless) Chrome or Chromium.

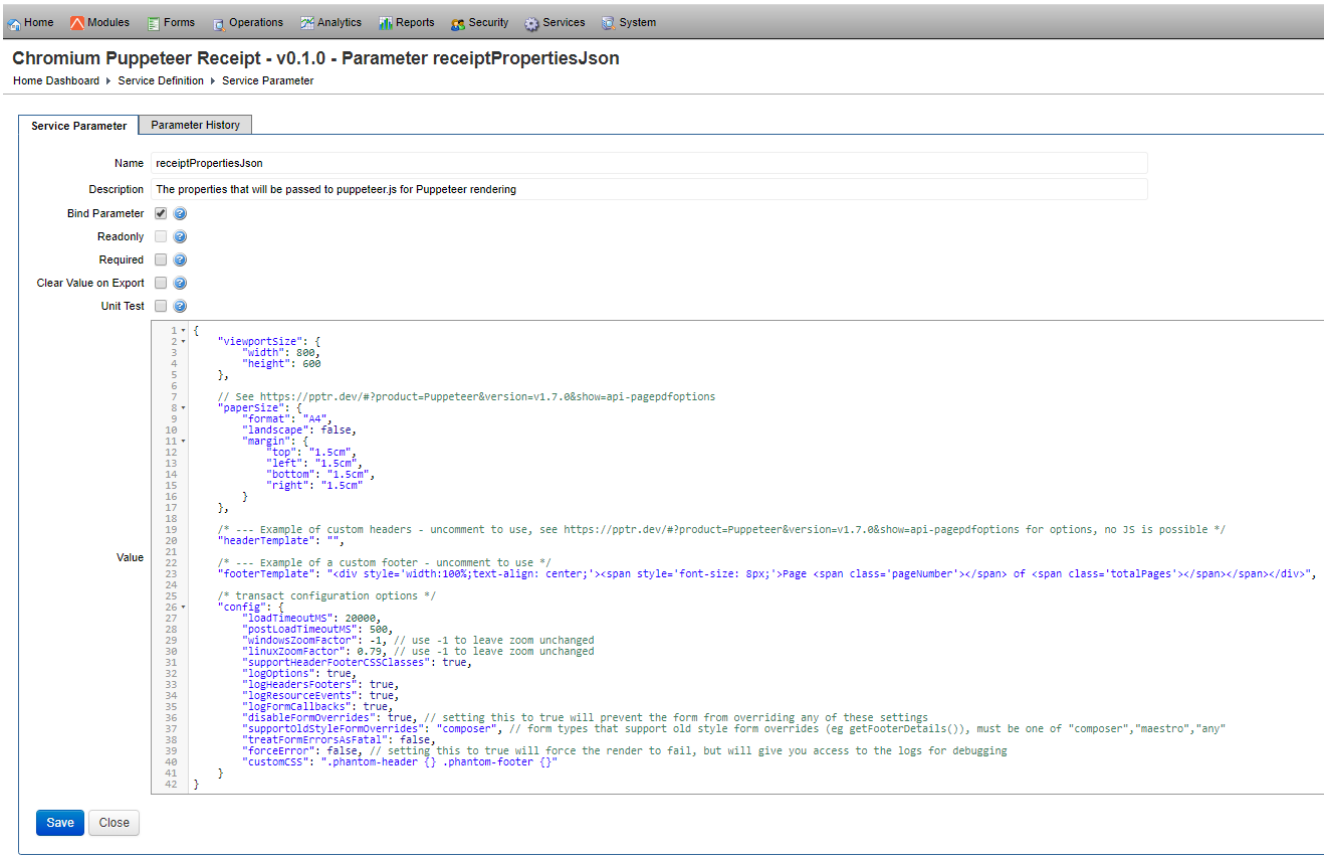
1. Select **Services > Core Global Services**.
2. Locate the service and click **Edit**.
3. Click the **Service Definition** tab to edit the standard service configuration.
4. Click the **Parameters Edit** tab to view or update the service configuration.



5. Select the **Cleanup Receipt Files** checkbox to clean up the receipt files after rendering has completed.
6. Edit the fallback process timeout (in milliseconds) in the **Fallback Timeout** field. The default is 3000.
7. Select the maximum number of concurrent processes from the **Max Processes** dropdown list. The default is 2.

> Changing **Max Processes** will not take effect until Manager is restarted.

8. The **Puppeteer Exe** field contains the file name of the puppeteer executable that will be run to generate a receipt. Modify the file name as required, without the Windows .exe file extension. The default value is puppeteer.
9. Click **Receipt Properties Json** to edit the properties that will be passed to puppeteer .js for rendering of customized headers and footers.



10. Click **Save** to update the changes.
11. Assign this service to your form version as the **Receipt Render Service**, so it can render a form submission as a PDF receipt. You can also make this service as default.

> Customizing the Chromium Puppeteer Receipt service will impact all receipts of forms that are linked to this service.

Next, learn how to view core global services.

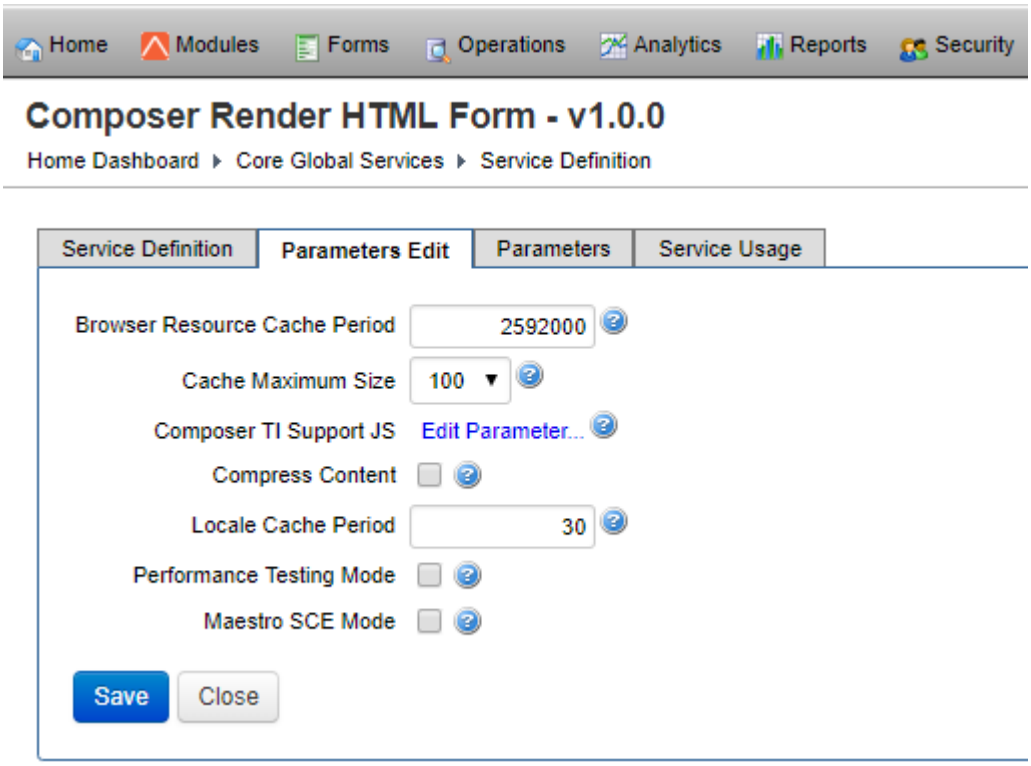
Composer Render HTML Form Service

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager comes with the Composer Render HTML Form core global service, which renders Composer SmartForms.

To configure the Composer Render HTML Form service:

1. Select **Services > Core Global Services**.
2. Locate the service and click **Edit**.
3. Configure the standard service settings.
4. Select the **Parameters Edit** tab to view and edit the configuration.



5. Edit the browser resource caching period in seconds in the **Browser Resource Cache Period** field. Default is 2592000.
6. Edit the server side form HTML and resource cache maximum size in the **Cache Maximum Size** field. Default is 100.
7. Click **Composer TI Support JS** to view and edit the Composer Transact Insights Support JavaScript file include. You can use the include file when Composer Form was not published with TI support.
8. Select the **Compress Content** checkbox to **GZIP**[1] compress the HTML content.

[1] GZIP is a file format and a software application used for file compression and decompression.

It's recommended to use Apache `mode_deflate` for HTML compression.

9. Edit the locale **JSON**[1] resources caching period in seconds in the **Locale Cache Period** field. Default is 30.
10. Select the **Performance Testing Mode** checkbox to enable Performance Testing mode which renders HTTP response header `xml_data` for use by automation tools.
11. Select the **Maestro SCE Mode** checkbox to enable Maestro **Static Content Externalization**[2] rendering mode. | 18.11
12. Click **Save** to update the changes.

Next, learn how to view core global services.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

[2] Static Content Externalization (SCE).

Current Version Selector Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager comes with the Current Version Selector core global service, which chooses the appropriate form version to render a form. The default service returns the current form version unless a specific version was requested.

If you would like to customize the way versions are selected, for example, for A/B testing, you can create your own form version selector service and implement custom logic in a **Groovy**[1] script.

> This service can be configured on the form level, otherwise, the system default version selector service is used.

To configure the Current Version Selector service:

1. Select **Services > Core Global Services**.
2. Locate the service and click **Edit**.
3. Configure the standard service settings.
4. Select the **Parameters Edit** tab to view and edit the specific service configuration.
5. Select the **Ignore Form Version Request Parameter** checkbox to always use the current form version regardless of the value of the **`tmFormVersion`** request parameter. This can prevent users loading and creating transactions of an old form version by providing the **`tmFormVersion`** URL parameter. This option is not selected by default.
6. Click **Save** to update the changes.

Next, learn about A/B testing form version selector service.

[1] Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

Data Retention Management Service

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager comes with the Data Retention Management core global service, which performs the following steps in the transaction life cycle:

- Task expiry
- Abandonment
- Data purging

This service is periodically called by the Data Retention job.

To configure the Data Retention Management service:

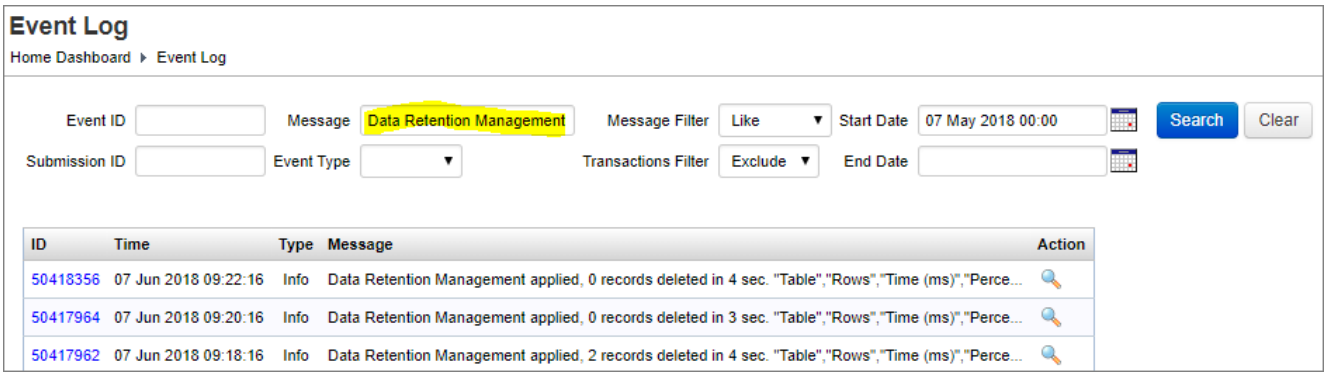
1. Select **Services > Core Global Services**.
2. Locate a service and click **Edit**.
3. Configure the standard service settings.
4. Select the **Parameters Edit** tab to view and edit the configuration.



5. This service uses the global Manager data retention configuration, but you can customize its data retention setting here.
6. Select the **Log to Event Log** checkbox to log the results of applying the data retention policy to the event log., so you can check what data was deleted from the system. The example of a log message recorded in the even logs is shown bellow:

```
Data Retention Management applied, 0 records deleted in 3 sec.

"Table","Rows","Time (ms)","Percentage"
"Delete Submission Data",0,77,2.0
"Job",0,7,0.2
"Submission",0,10,0.3
"Transaction History",0,4,0.1
"Audit Log",0,4,0.1
"User Auth Event",0,3,0.1
"Error Log",0,13,0.3
"Event Log",0,3784,96.2
"Groovy Service Log",0,3,0.1
"Offline Client System Log",0,3,0.1
"Offline Sync Call",0,4,0.1
"Offline Sync Log",0,3,0.1
"Scheduled Job History",0,3,0.1
"Security Manager Log",0,3,0.1
"Email Queue",0,4,0.1
"Composer Package Queue",0,3,0.1
"Import Action",0,5,0.1
```

Then you can search for particular retention event in the event logs using the following criteria:



7. Select the required purge mode from the **PII Search Purge Mode** dropdown list to specify when PII searching metadata, such as data extracts and transactions properties, will be deleted:
   - Form Completed - this is the smallest set of data to be deleted.
   - Delivery Completed - this is a bigger set of data to be deleted.
   - PII Purge Time - this is the biggest set of data to be deleted and it takes the longest time to do it.
   - Blank - not defined.
8. Click **Save** to update the changes.

If the **Include Data Retention** checkbox is selected, the Transaction Processor service will execute the Data Retention Management service.

Next, learn how to view all core global services.

Dynamic PDF Receipt Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides the following form-specific services for generating PDF receipts:

- The Dynamic PDF Receipt service, which is the legacy receipt render service.
- The Dynamic PDF Receipt 2 service, which provides more options for customizing the receipts.

The service is called when a form receipt is opened. The service uses an orchestration in **Adobe LiveCycle ES**[1] to render the receipt. Typically, receipts are rendered without buttons, text fields, and other interactive form feature.

This service uses the **PhantomJS**[2] headless internet browser, which has no user interface.

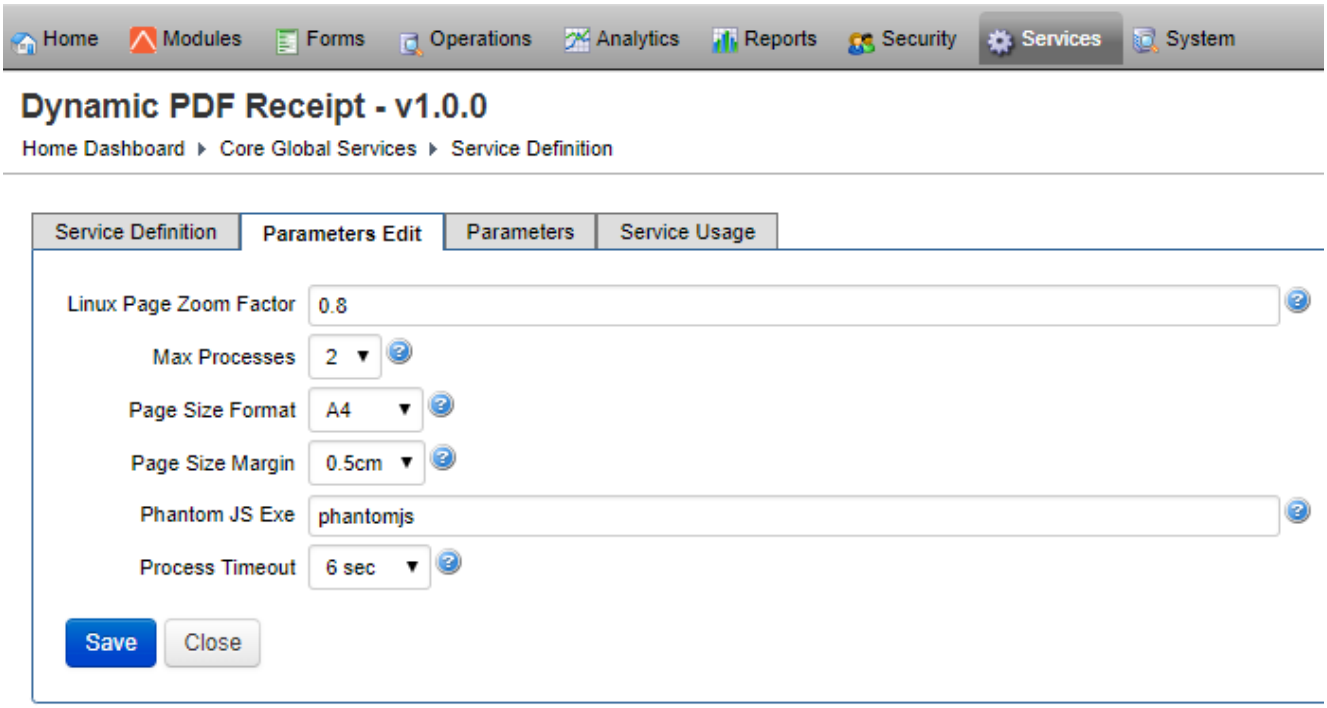
To configure the Dynamic PDF Receipt service:

1. Select **Services > Core Global Services**.
2. Locate the service and click **Edit**.
3. Click the **Service Definition** tab to edit the standard service configuration.
4. Click the **Parameters Edit** tab to view or update the service configuration.

---

[1]Adobe LiveCycle Enterprise Suite (ES4) is an service-oriented architecture Java EE server software product from Adobe Systems used to build applications that automate a broad range of business processes for enterprises and government agencies. LiveCycle ES4 is an enterprise document and form platform that allows capturing and processing information, delivering personalized communications, and protecting and tracking sensitive information. It is used for purposes such as account opening, services and benefits enrollment, correspondence management, request for proposal processes, and other manual based workflows.

[2]PhantomJS is a scripted, headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment. PhantomJS is based on WebKit making it a similar browsing environment to Safari and Google Chrome (before Chrome's fork of WebKit evolved into Blink). It is open-source software released under the BSD License.



5. Enter **Linux Page Zoom Factor**, which is the paper zoom factor for Linux servers. The default is 0.8.
6. Select the maximum number of concurrent **PhantomJS**[1] processes from the **Max Processes** dropdown list. The default is 2.

> Changing **Max Processes** will not take effect until Manager is restarted.

7. Select the paper size format from the **Page Size Format** dropdown list. The default is A4.
8. Select the paper size margin from the **Page Size Margin** dropdown list. The default is 0.5 cm.
9. Provide the **PhantomJS**[2] executable file name (excluding the Windows .exe file extension) in the **Phantom JS Exe** field. The default is phantomjs.
10. Select the PhantomJS rendering process timeout in milliseconds from the **Process Timeout** dropdown list. The default is 6 seconds.

---

[1]PhantomJS is a scripted, headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment. PhantomJS is based on WebKit making it a similar browsing environment to Safari and Google Chrome (before Chrome's fork of WebKit evolved into Blink). It is open-source software released under the BSD License.

[2]PhantomJS is a scripted, headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment. PhantomJS is based on WebKit making it a similar browsing environment to Safari and Google Chrome (before Chrome's fork of WebKit evolved into Blink). It is open-source software released under the BSD License.

11. Click **Save** to update the changes.

> Customizing the Dynamic PDF Receipt service will impact all receipts of forms that are linked to this service.

Next, learn how to configure dynamic PDF receipt 2 service.

Dynamic PDF Receipt 2 Service

Journey Manager (JM) | System Manager / DevOps | 17.10

Manager provides the **Dynamic PDF Receipt 2** service for generating PDF receipts. The service is the extension of the legacy Dynamic PDF Receipt service, which belongs to Render Receipt core global services, and provides several extra customization options. The Dynamic PDF Receipt 2 service uses the **PhantomJS**[1] headless Internet browser to produce a receipt. Receipts can have headers and footers, which you can customize by adjusting their properties in a JSON file.

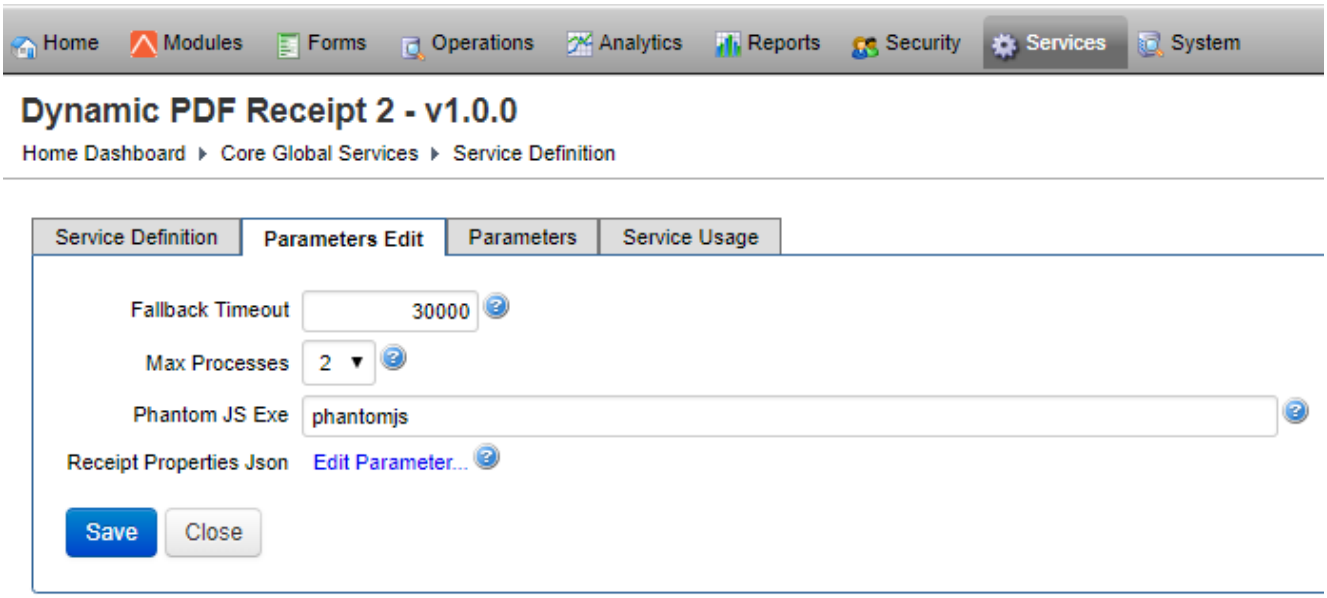
To create the Dynamic PDF Receipt 2 service:

1. Select **Services > Core Global Services**. and click New.
2. Select **Render Receipt** from the **Service Type** dropdown list.
3. Select **Dynamic PDF Receipt 2** from the **Service Template** dropdown list.
4. Update the service name in the **New Name** field.
5. Update the service version in the **Version Number** field.
6. Click **Save** to update the changes.

To configure the Dynamic PDF Receipt 2 service:

1. Select **Services > Core Global Services**.
2. Locate the service and click **Edit**.
3. Click the **Service Definition** tab to edit the standard service configuration.
4. Click the **Parameters Edit** tab to view or update the service configuration.

[1]PhantomJS is a scripted, headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment. PhantomJS is based on WebKit making it a similar browsing environment to Safari and Google Chrome (before Chrome's fork of WebKit evolved into Blink). It is open-source software released under the BSD License.

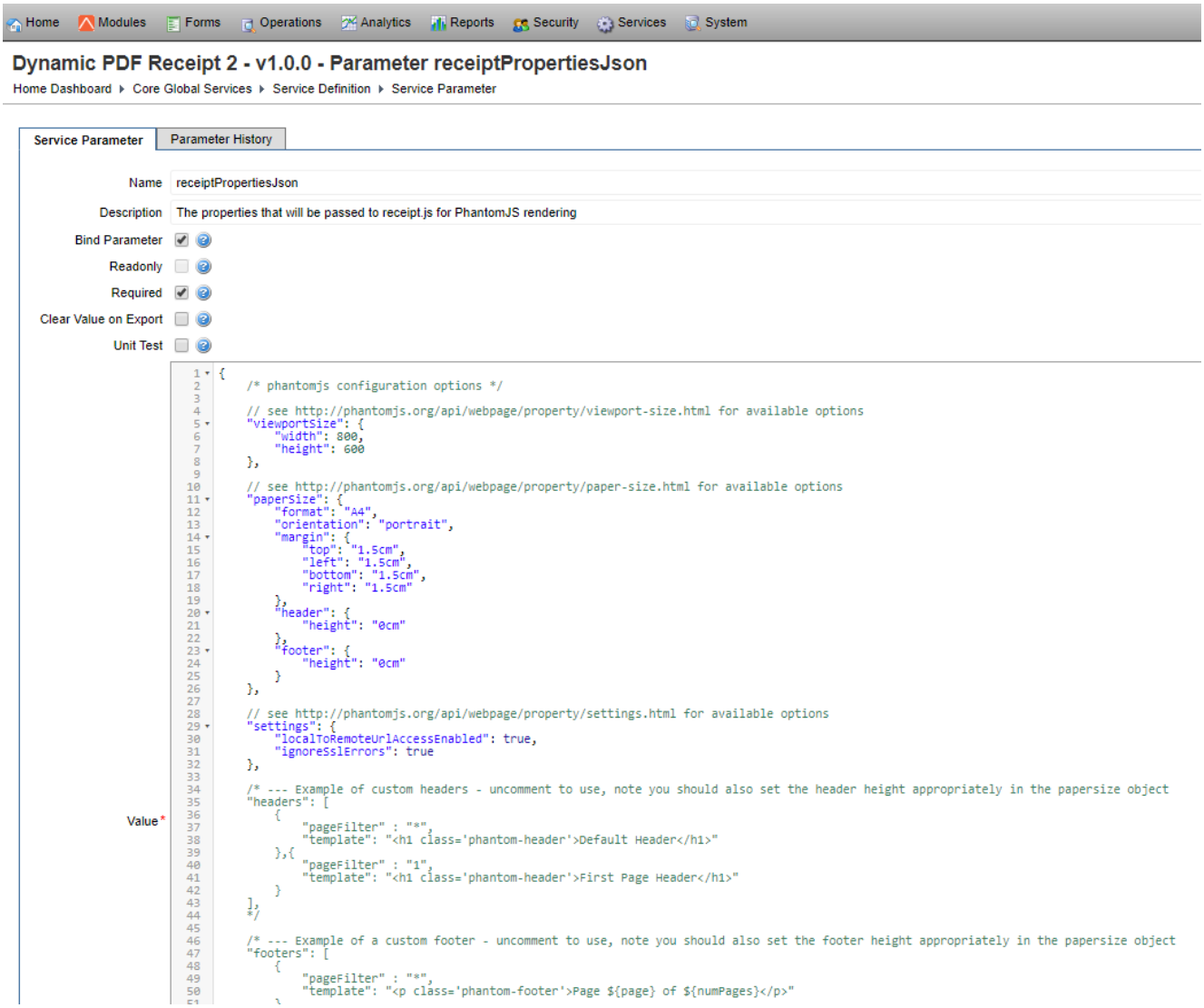


5. Edit **Fallback Timeout**, which is the fallback process timeout in milliseconds. The default is 3000.
6. Select **Max Processes**, which is the maximum number of concurrent PhantomJS processes. The default is 2.

> Changing **Max Processes** will not take effect until Manager is restarted.

7. Edit **Phantom JS Exe**, which is the **PhantomJS**[1] executable file name (excluding the Windows .exe file extension). The default is phantomjs.
8. Click **Receipt Properties Json** to edit the properties that will be passed to the receipt.js for customized headers and footers rendering.

[1] PhantomJS is a scripted, headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment. PhantomJS is based on WebKit making it a similar browsing environment to Safari and Google Chrome (before Chrome's fork of WebKit evolved into Blink). It is open-source software released under the BSD License.



9. Click **Save** to update the changes.

Next, learn how to view core global services.

Form Submission Access Controller Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

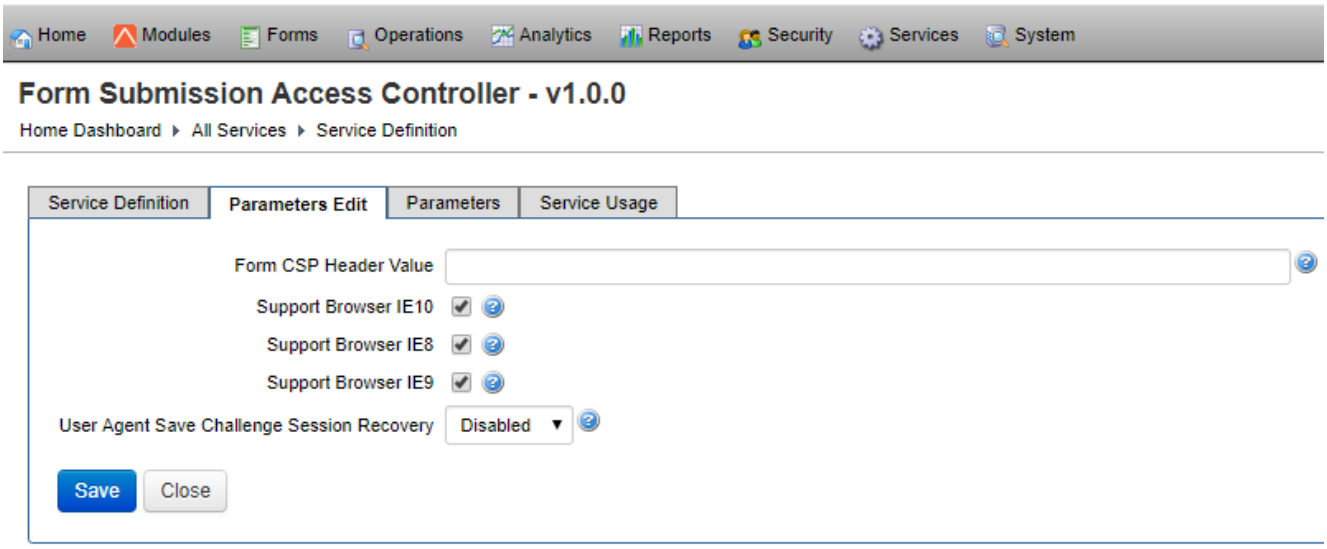
Manager has a variety of built-in security access rules. One of them is to allow a user to access form's submissions only if a user's session matches a session used to render that form. If the sessions don't match, the user is expected to re-enter the save challenge details. However, Composer forms prior to version 4.3 SP3 don't have this functionality within the form flow.

Manager comes with the form submission access controller service, which you can use to change and override the existing users access rules to form's submissions.

> If a session is re-established, access to the submissions will only be valid within the **Anonymous Submission Access Timeout** time configured for this form space.

To configure the form submission access controller service:

1. Select **Services > Core Global Services**.
2. Locate the service and click **Edit**.
3. Click the **Service Definition** tab to edit the standard service configuration.
4. Click the **Parameters Edit** tab to view or update the service configuration.



5. Specify the system wide **Content Security Policy**[1] (CSP) security HTTP header value in the **Form CSP Header Value** field.

Check the Security Header setting, which is a part of a form space configuration.

6. Select the **Support Browser IE10** checkbox to support the legacy Internet Explorer 10 (IE10) browser.

IE10 doesn't support **TLS**[2] 1.1-1.2 or Content Security Protection (CSP).

7. Select the **Support Browser IE8** checkbox to support the legacy Internet Explorer 8 (IE8) browser.

IE8 doesn't support TLS 1.1-1.2 or Content Security Protection (CSP).

8. Select the **Support Browser IE9** checkbox to support the legacy Internet Explorer 9 (IE9) browser.

IE9 doesn't support TLS 1.1-1.2 or Content Security Protection (CSP).

9. Select a save challenge session recovery mode from the **User Agent Save Challenge Session Recovery** dropdown list to override the existing behavior. You can use one of the following options:
   - Disabled - sessions aren't re-establish, so users must re-enter the save challenge details to recover sessions. This is the default option.
   - Enabled - **user agent**[3] check is enabled to allow sessions to be re-established.
   - 4.3.0SP2 - **user agent**[4] check is enabled to allow sessions to be re-established for forms built using Composer version 4.3 SP2 and higher.
10. Click **Save** to update the changes.

Next, learn how to view core global services.

---

[1]Content Security Policy (CSP) is a computer security standard introduced to prevent cross-site scripting (XSS), clickjacking and other code injection attacks resulting from execution of malicious content in the trusted web page context. CSP provides a standard method for website owners to declare approved origins of content that browsers should be allowed to load on that website—covered types are JavaScript, CSS, HTML frames, web workers, fonts, images, embeddable objects such as Java applets, ActiveX, audio and video files, and other HTML5 features. For more information, see https://www.w3.org/TR/CSP/

[2]Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network.

[3]User agent is software (a software agent) that is acting on behalf of a user. One common use of the term refers to a web browser that retrieves, renders and facilitates end user interaction with Web content.

[4]User agent is software (a software agent) that is acting on behalf of a user. One common use of the term refers to a web browser that retrieves, renders and facilitates end user interaction with Web content.

Payment Reminder Service

Journey Manager (JM) | System Manager / DevOps | 19.05

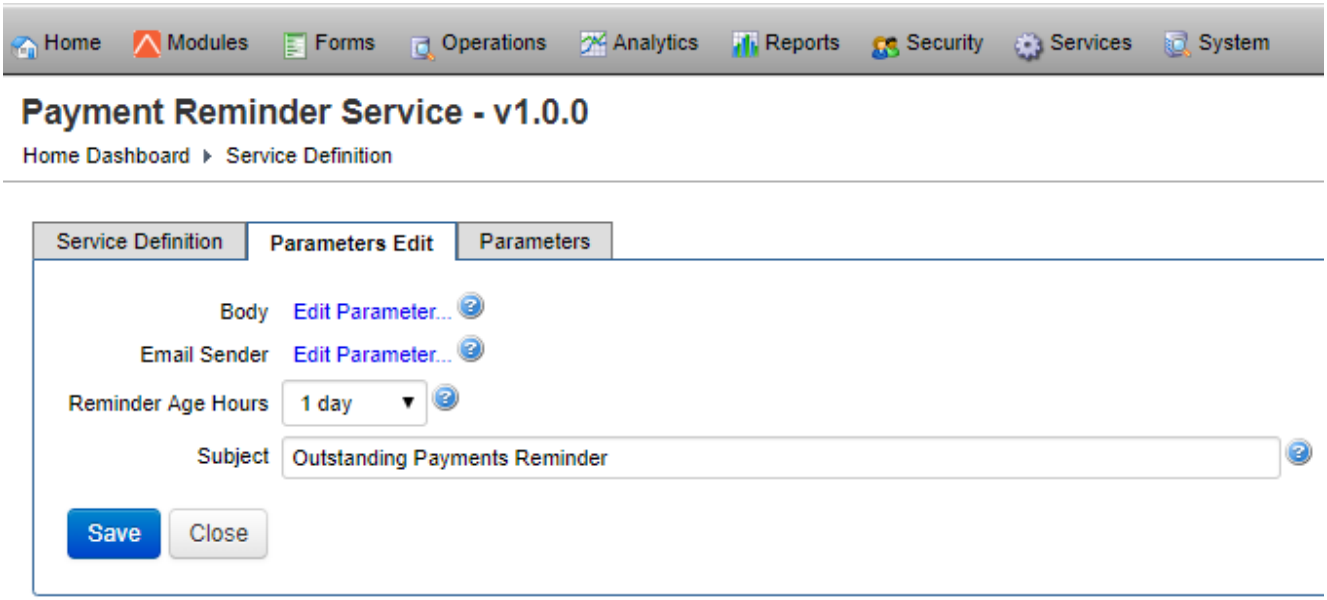
Manager provides a service to send out reminder emails about submissions with outstanding payments older than a certain age.

To configure a payment reminder service:

1. Locate an existing service or create a new one using the **Service Type** as Payment Reminder.

> You can also copy the existing payment reminder service to create a new one.

2. Click the **Parameters Edit** tab to view or change the service configuration.



3. Click **Body** to edit the email body template.
4. Click **Email Sender** to edit the email sender address for the payment alert emails.
5. Select the reminder age in hours for outstanding payment after which the alert email will be sent from the **Reminder Age Hours** dropdown list. Default is 1 day.
6. Edit the email subject template in the **Subject** field.
7. Click **Save** to update the changes.

Next, learn how to view all services.

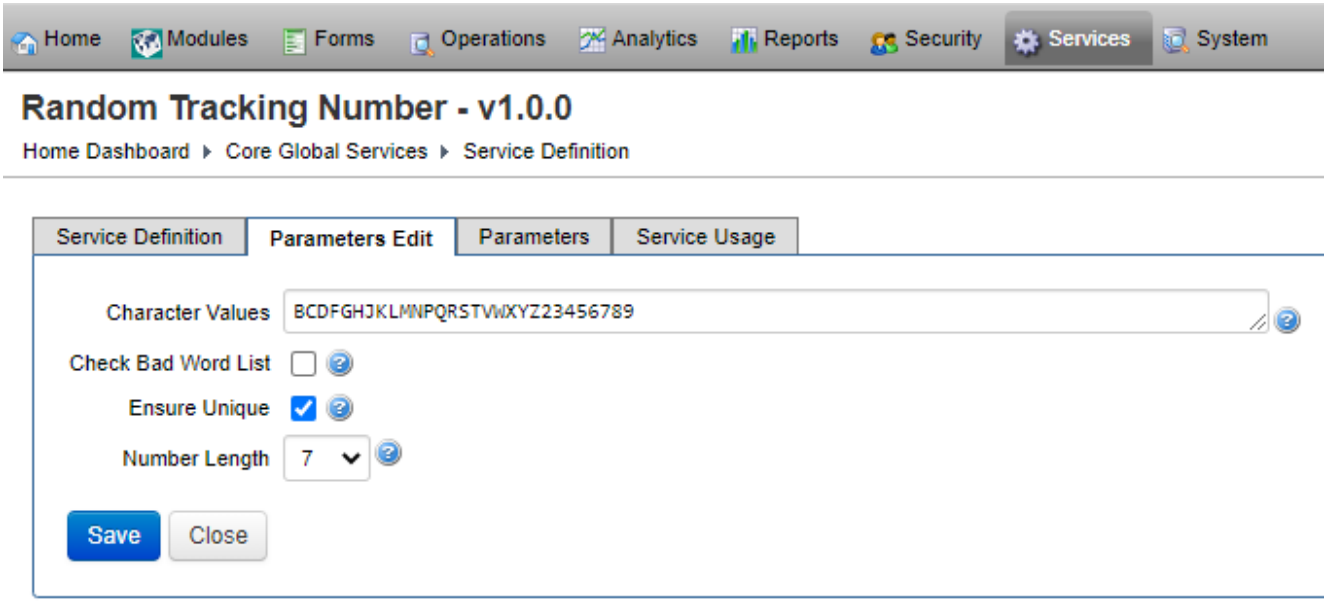
Random Tracking Number Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager comes with the Random Tracking Number service, which is the core global service, that generates a unique tracking code for each transactions. For more information on the uniqueness of the generated codes and how to configure them, see configure receipt numbers and tracking codes.

To configure the Random Tracking Number service:

1. Select **Services > Core Global Services.**
2. Locate the service and click **Edit**.
3. Configure the standard service settings.
4. Click the **Parameters Edit** tab to view and edit the configuration.



5. Edit **Character Values**, which is the list of character values to generate the random tracking number from.
6. Select the **Check Bad Word List** to validate a random tracking number against badword list. This is not required as available letters exclude vowels, and does incur some performance cost.
7. Select **Ensure Unique**, which Ensures the number is unique by checking against tracking number table.
8. Edit **Number Length**, which is the length of the tracking number.
9. Click **Save** to update the changes.

Next, learn how to view core global services.

Status Management Service

Journey Manager (JM) | System Manager / DevOps | 22.04

Manager provides the status management service to change statuses of form spaces, forms, and form versions, which have been scheduled using Notifications and Status Change. The service runs every 5 mins and processes status change events. It also add an entry to the event log every time the service runs. The entry includes:

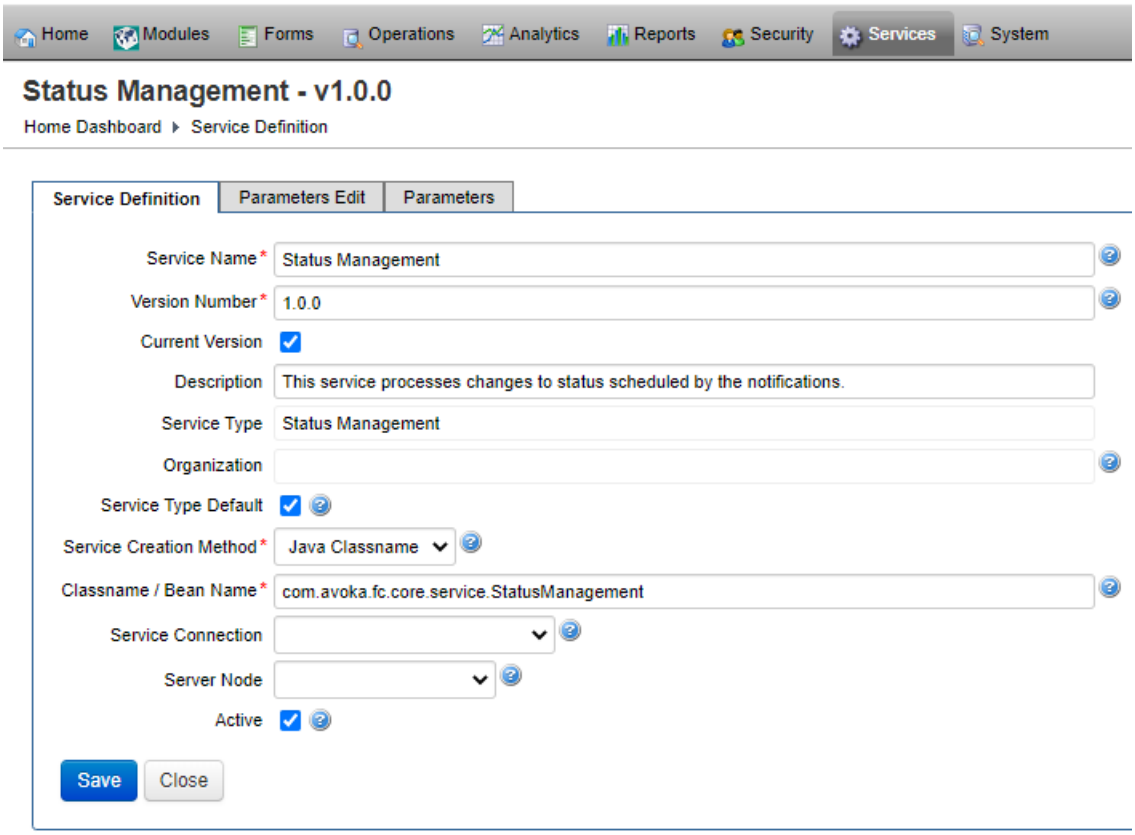
- Number of entities
- Names of specific entities changed

The log entry example is shown below:

```
Status Management completed in 0 sec. Duration 3 ms.
  Number of entities updated: 1 Detailed re
```

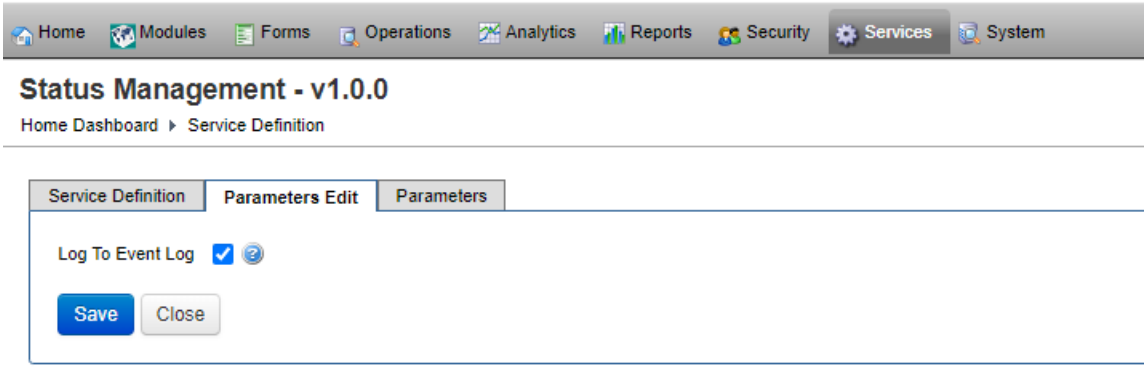
To configure the status management service:

1. Select **Services > Core Global Services**.
2. Locate the **Status Management** service and click **Edit**.
3. Click the **Service Definition** tab to configure the service.



4. Click **Save** to update the changes.

5. Click the **Parameters Edit** tab to configure the service parameters.



6. Select the **Log To Event Log** checkbox to add an entry to the event log every time the service runs.
7. Click **Save** to update the changes.

Next, learn how to view all core global services.

Submission Receipt Render Service

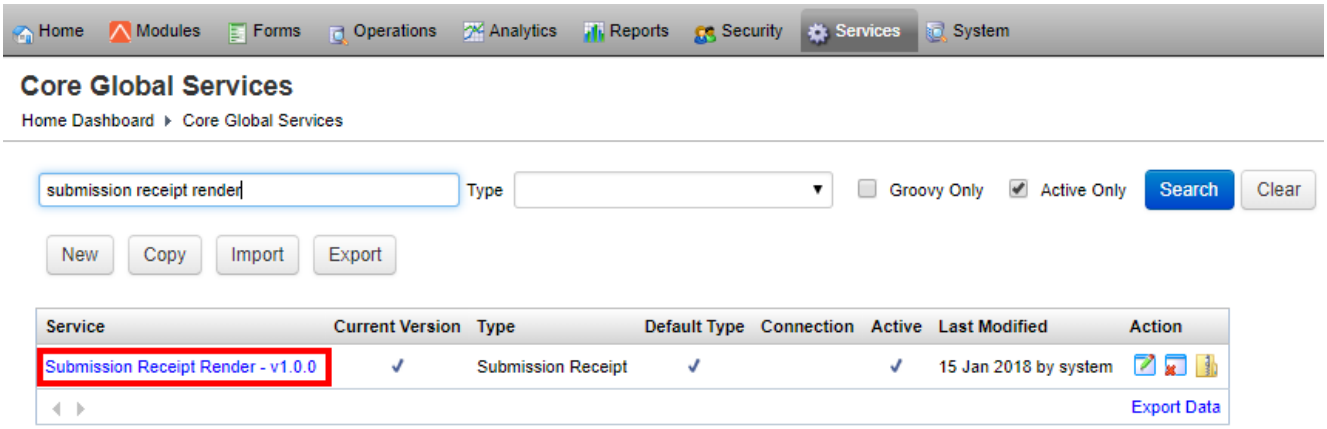
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides the Submission Receipt Render core global service, which creates and stores receipts for new submissions, re-attempts to render receipts for submissions where previous attempts failed, and generates PDF receipts in the background.

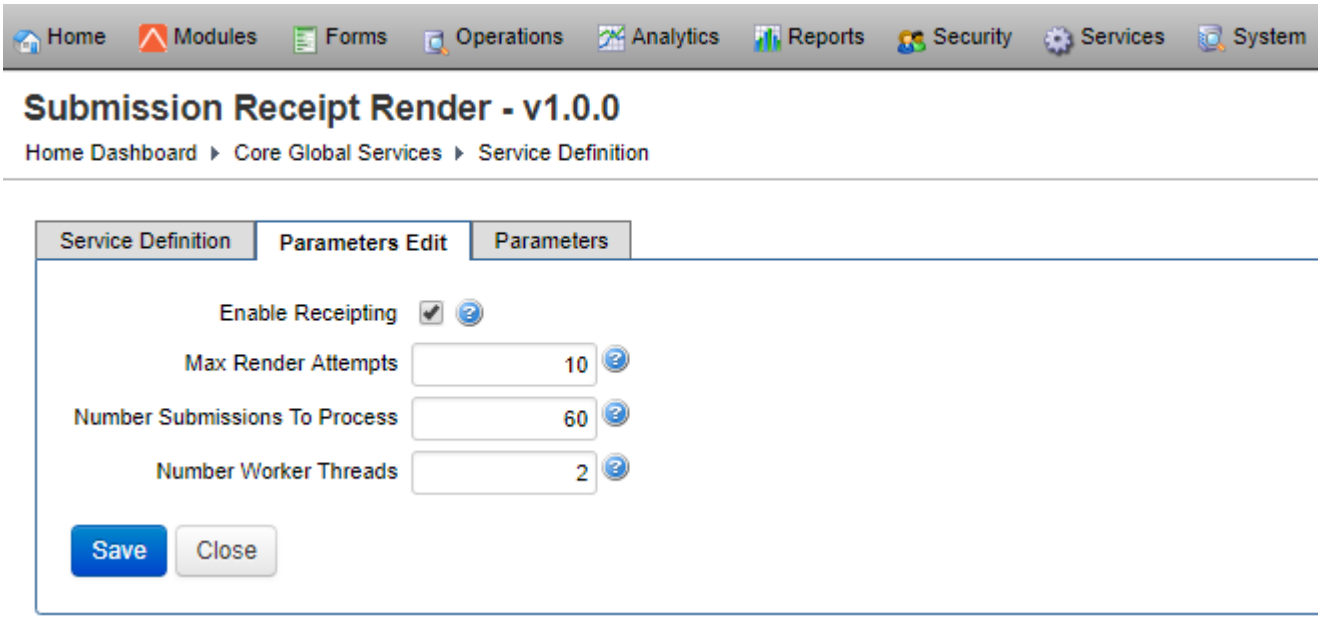
This service is called periodically by the Transaction Processor service.

To view the Submission Receipt Render service configuration:

1. Select **Services > Core Global Services.**
2. Locate the service and click **Edit**.



3. Configure the standard service settings.
4. Click the **Parameters Edit** tab to view and edit the configuration.



5. Select **Enable Receipting** to enable background PDF receipt generation. It is selected by default.
6. Edit **Max Render Attempts**, which is the maximum number of times a PDF receipt will attempt to render before failing and being placed into an error state. The default is 10.
7. Edit **Number Submissions To Process**, which is the maximum PDF receipt generation queue length.This is a balance, if the queue length is too long then during quiet times a single node may take all the receipts waiting for receipts and effectively serialise the process affecting responsiveness. The default is 60.
8. Edit **Number Worker Threads**, which is the number of concurrent PDF rendering processes to run on a single node. This number must not exceed the Max Processes parameter setting on the Dynamic PDF Receipt or Dynamic PDF Receipt 2 services.
9. Click **Save** to update the changes.

You can use this service to improve Manager load balancing.

Next, learn how to view core global services.

Transaction Processor Service

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager comes with the Transaction Processor core global service, which plays an important role in the transaction life cycle by performing the following steps:

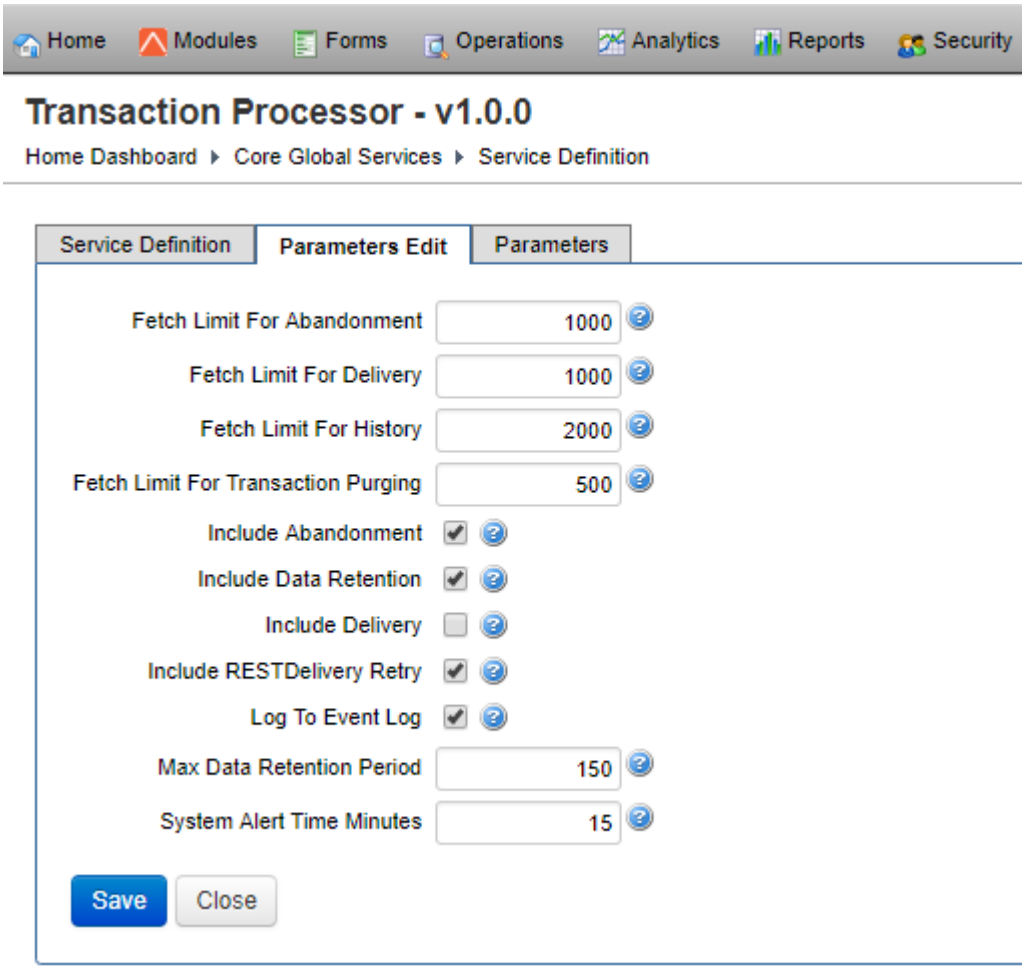
- Abandonment
- Receipt generation
- Delivery
- Task expiry
- Transaction history creation
- Data purging

The Transaction Processor service is called by the Transaction Processing scheduled job.

> Seek advice from Temenos before changing this service configuration.

To configure the Transaction Processor service:

1. Select **Services > Core Global Services**.
2. Locate the service and click **Edit**.
3. Configure the standard service settings.
4. Select the **Parameters Edit** tab to view and edit the configuration.



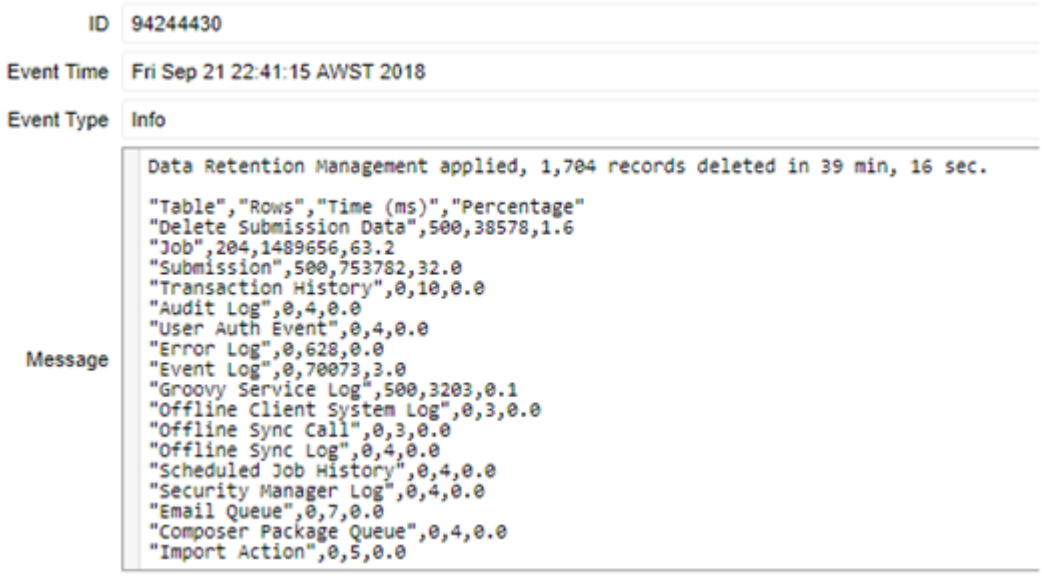
5. Edit **Fetch Limit For Abandonment**, which is the maximum number of records to consider for each abandonment type. Default is 1000.
6. Edit **Fetch Limit For Delivery**, which is the maximum number of records to consider for each delivery type (normal vs abandoned submission delivery). Default is 1000.
7. Edit **Fetch Limit For History**, which is the maximum number of records to process for transaction history creation. Default is 2000.
8. Edit **Fetch Limit For Transaction Purging**, which is the maximum number of items to be purged per each category: submissions, collaboration jobs, transaction history, error logs, and so on. Default is 500.

   It is important to understand that, for each category, the full hierarchy is purged. This means that:

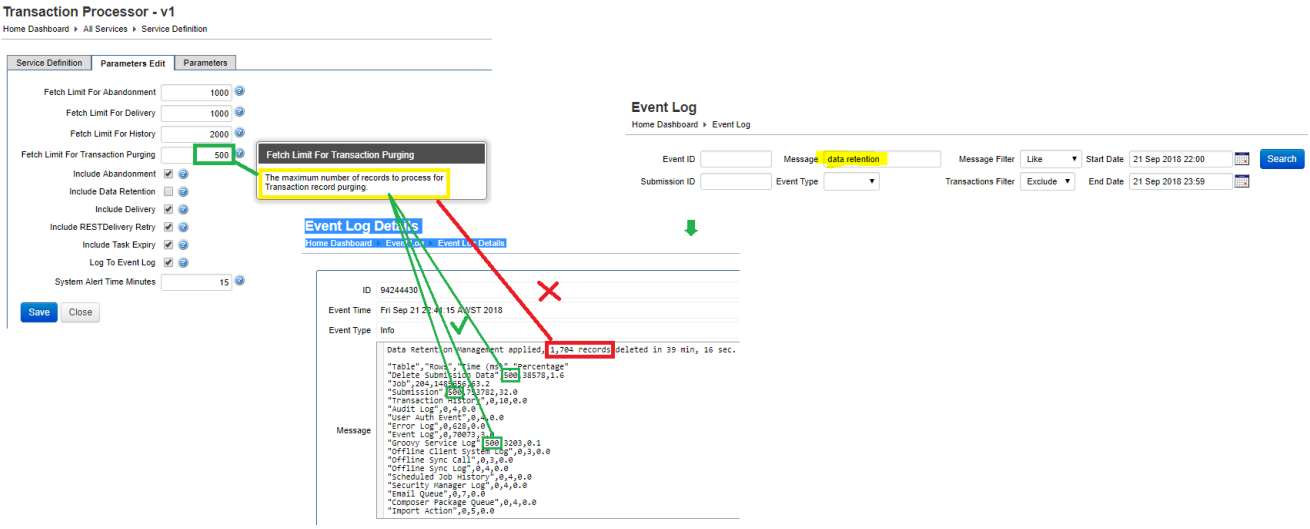
- For a submission - all related error logs are purged.
- For a collaboration job - all related submission are purged.

The Fetch limit only applies to the top level of the category. So in the example of submissions, 500 independent submission may be purged, but if collaboration jobs are purged, then the actual number of submission will be higher.

Open the event log and search for events using **data retention** key word to see the actual number of purged transactions of each category, as shown below.



This is also illustrated below.



9. Select the **Include Abandonment** checkbox to turn off abandonment. It is selected by default.

> Transactions that should be abandoned will not be processed.

10. Select the **Include Data Retention** checkbox to turn off data retention processing, which executes the Data Retention Management service. It is selected by default.

It is not recommended to turn off data retention as no data will be purged. However, you can do it for a short period of time to troubleshoot transaction processing issues.

11. Select the **Include RESTDelivery Retry** checkbox to turn off delivery. It is selected by default.
12. Select the **Include Task Expiry** checkbox to turn off task expiry. It is selected by default.

Transactions that should be expired will not be processed.

13. Select the **Log To Event Log** checkbox to log the results of the transaction processing to the event log.It adds two entries to the event log - one for the Transaction Processor and one for data retention, which includes timing information relating to the performance of the operations.It is not selected by default.

Disable logging unless you need to troubleshoot transaction processing issues.

14. Edit **Max Data Retention Period** to change the maximum period of time, in seconds, allocated for data retention purging. The default is 150 seconds. This allows the Transaction Processor to limit the amount of time it spends on data retention processing. That is, when purging takes longer than expected, the Transaction Processor will stop, move onto other tasks, and continue at a later time. This ensures that purging data does not affect other parts of the Manager system. | 18.11
15. Edit **System Alert Time Minutes**, which is the number of minutes that the system monitoring job will use when determining whether system alerts need to be raised about the transaction processing service. Default is 15 minutes.
16. Click **Save** to update the changes.

Next, learn how to view core global services.

Attachment Virus Scanning Service

Journey Manager (JM) | System Manager / DevOps | 19.11

To avoid users attaching malicious or infected files to their applications, Manager offers several antivirus services. If a antivirus scanning service is configured, Manager uses it to check each attached file for viruses, for example, during Maestro file upload . These attachments will only be delivered once they have been scanned and marked as free of viruses or malicious software. If an attachment contains a virus, it is automatically deleted and the form user, who uploaded the attachment, will be notified via email.

Manager provides the following preconfigured virus scanning services, so you can choose one depending on **Virus Scanning Engines**[1]:

- ClamAV attachment virus scanning service
- Symantec attachment virus scanning service
- Fluent attachment virus scanning service

You must configure at least one virus scanning service in Manager so it can perform regular virus scanning activities.

Next, learn how to view core global services.

[1]Virus Scanning Engine is Antivirus software (AV software), also known as anti-malware, which is a computer program used to prevent, detect, and remove malware.

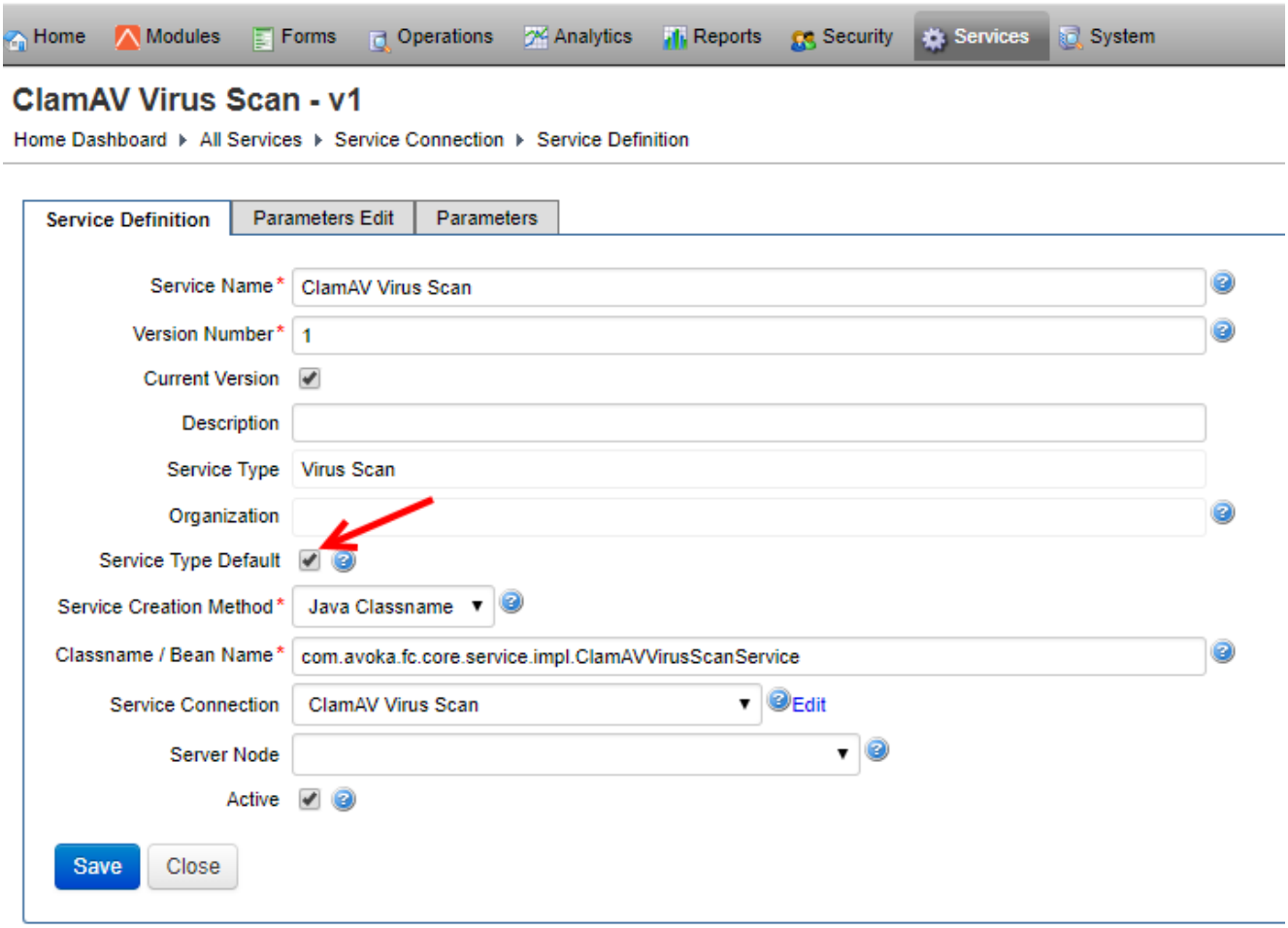
ClamAV Attachment Virus Scanning Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to configure the ClamAV Virus Scanning service to check files for viruses using ClamAV anti-virus software. You can also chose another antivirus software to use depending on your requirements.

To configure the ClamAV Virus Scanning service:

1. Configure **ClamAV**[1] as described in https://www.clamav.net/.
2. Configure the ClamAV Virus Scan service connection.
3. Select **Services > Core Global Services**.
4. Locate the **ClamAV Virus Scan** service and click **Edit.**
5. Select the **Service Definition** tab.



[1]Clam AntiVirus (ClamAV) is a free, cross-platform and open-source antivirus software toolkit able to detect many types of malicious software, including viruses. One of its main uses is on mail servers as a server-side email virus scanner. The application was developed for Unix and has third party versions available for AIX, BSD, HP-UX, Linux, macOS, OpenVMS, OSF (Tru64) and Solaris.

6. Select the **Service Type Default** checkbox to activate this service, which will be used as the default service for the virus scan type.

> There can be only one default service for each service type and only Global Services can be marked as the default ones.

> To deactivate a virus scanning service, clear the **Service Type Default** checkbox

7. Click **Save** to update the changes.

Next, learn how to configure Symantec virus scanning service.

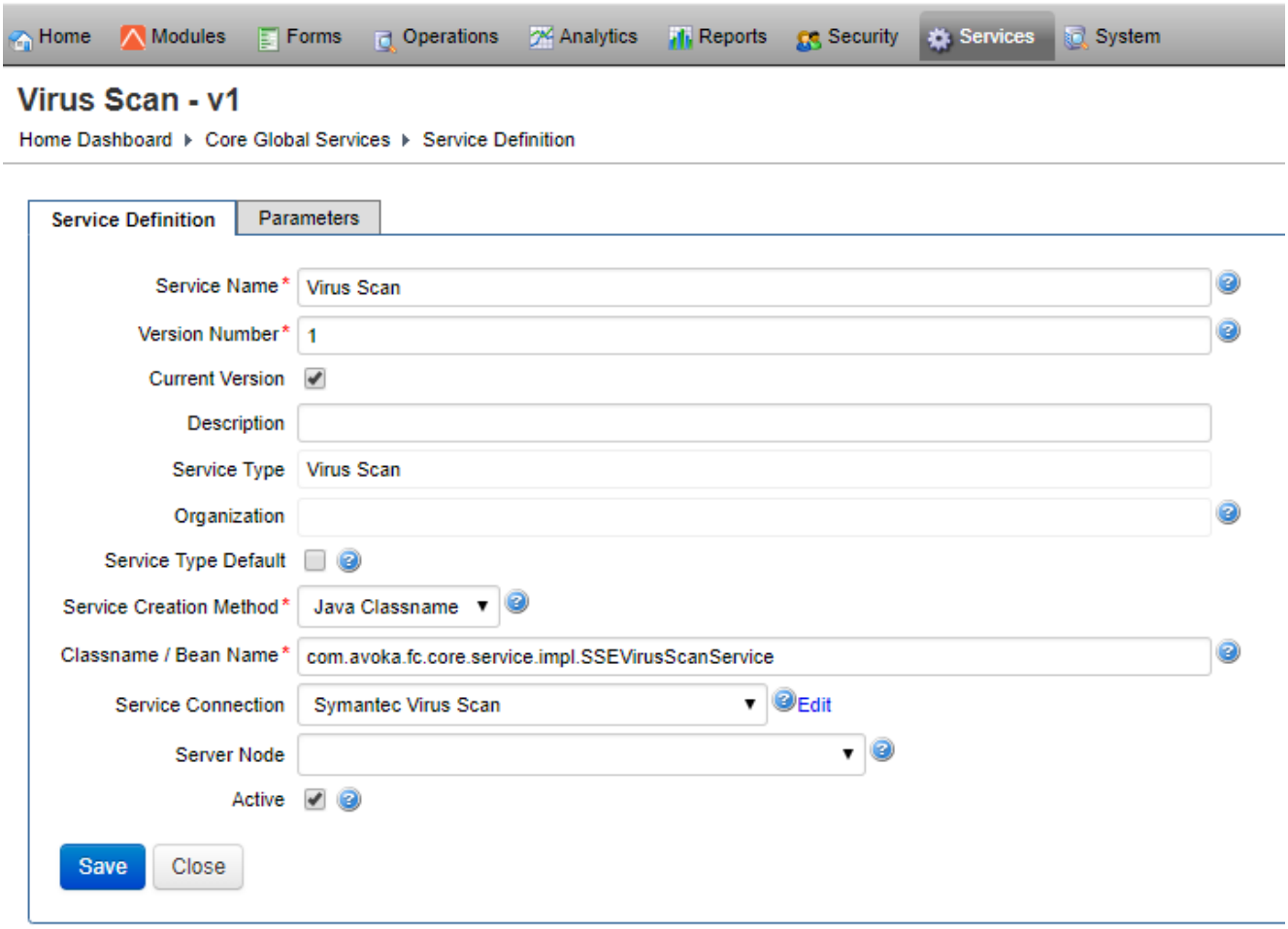
Symantec Attachment Virus Scanning Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to configure the Symantec Virus Scanning service to check files for viruses using Symantec anti-virus software. You can also chose another antivirus software to use depending on your requirements.

To configure a service:

1. Configure the Symantec Protection Engine.
2. Configure the Symantec Virus Scan service connection.
3. Select **Services > Core Global Services**.
4. Locate the **Symantec Virus Scan** service and click **Edit.**
5. Select the **Service Definition** tab.



6. Select the **Service Type Default** checkbox to activate this service, which will be used as the default service for the virus scan type.
7. Click **Save** to update the changes.

Only Global Services can be marked as the default ones.

There can be only one default service for each service type.

To deactivate a virus scanning service, clear the **Service Type Default** checkbox and click **Save**.

Next, learn how to configure ClamAV virus scanning service.

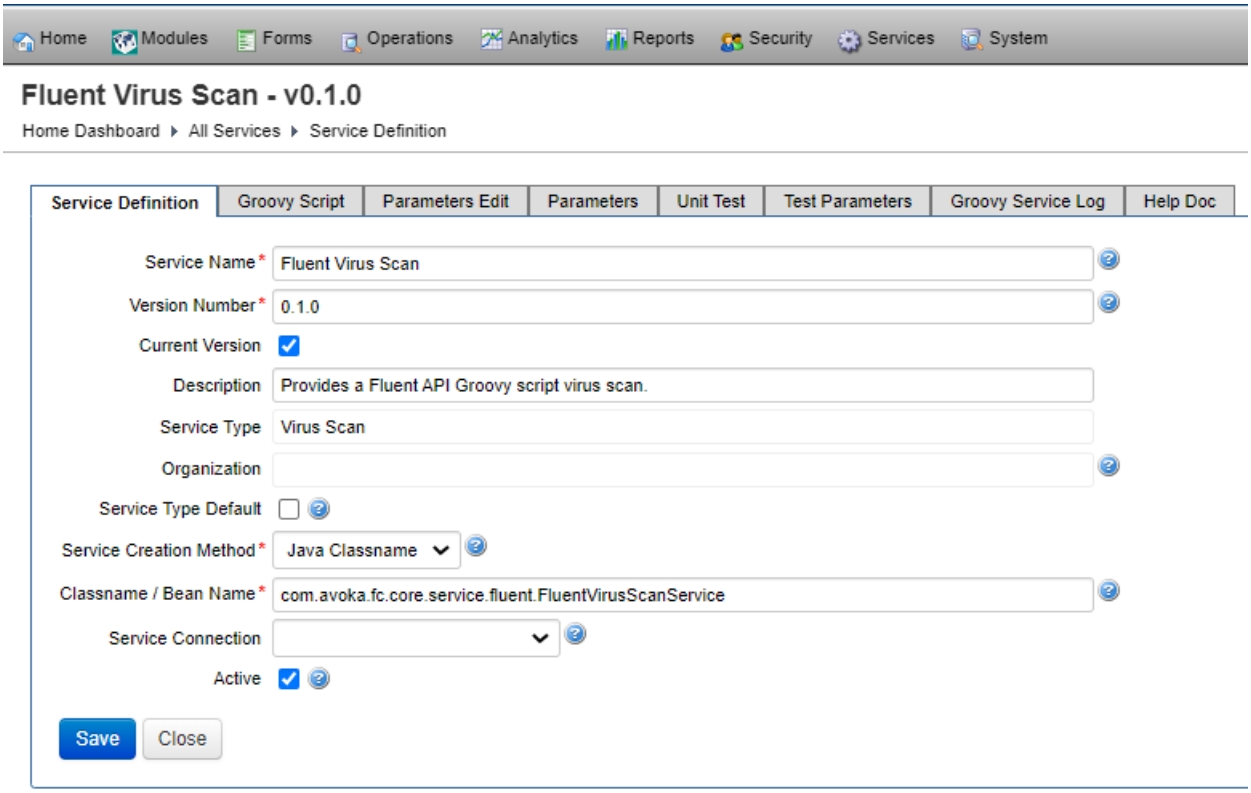
Fluent Attachment Virus Scanning Service

Journey Manager (JM) | System Manager / DevOps | 19.11

Manager allows you to create and configure a Fluent service, which checks files, such as, attachments, for viruses invoking antivirus software of your choice. You can also chose another antivirus software to use depending on your requirements.

To configure the Fluent service:

1. Configure a service connection, if required.
2. Select **Services > Core Global Services**.
3. Locate the Fluent virus scan service and click **Edit.**
4. Select the **Service Definition** tab.



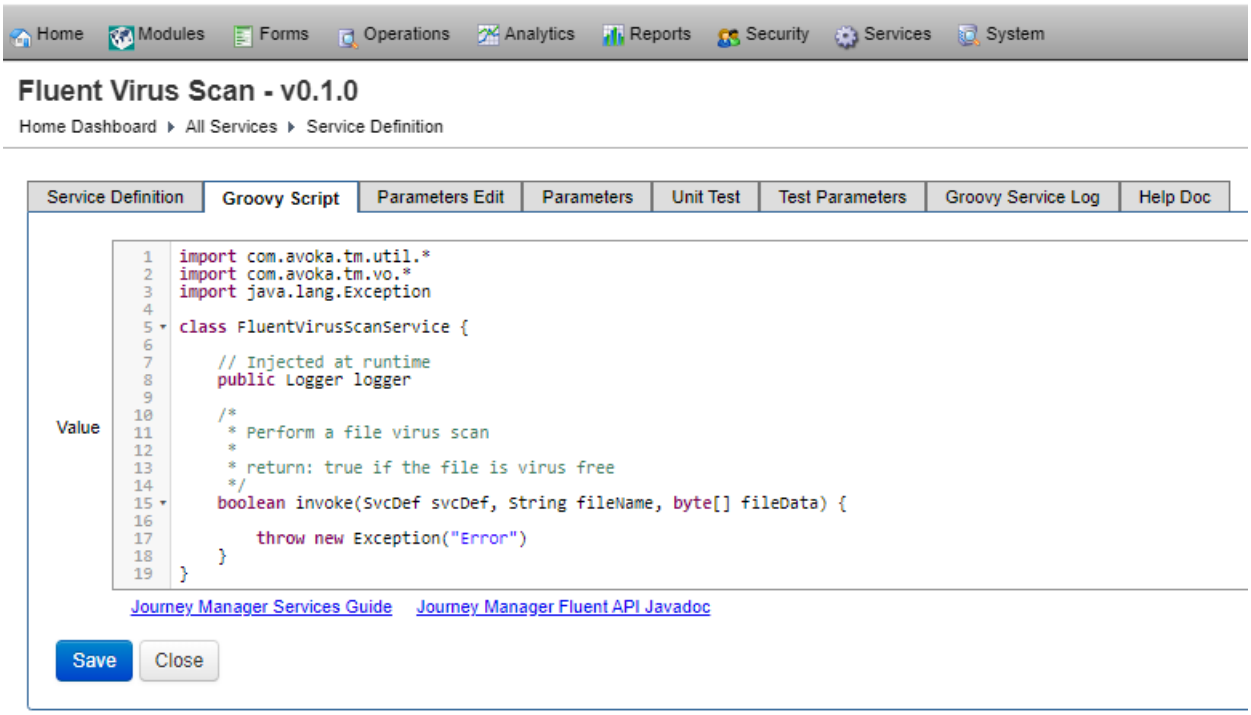
5. Select the **Service Type Default** checkbox to activate this service, which will be used as the default service for the virus scan type.

There can be only one default service for each service type and only Global Services can be marked as the default ones.

To deactivate a virus scanning service, clear the **Service Type Default** checkbox

6. Select the **Groovy Script** tab to configure the Groovy code that performs a file scanning or invokes another service to do the job.

   Check the latest Fluent API to implement your changes.



7. Click **Save** to update the changes.

Next, learn how to configure ClamAV virus scanning service.

Dynamic Data Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

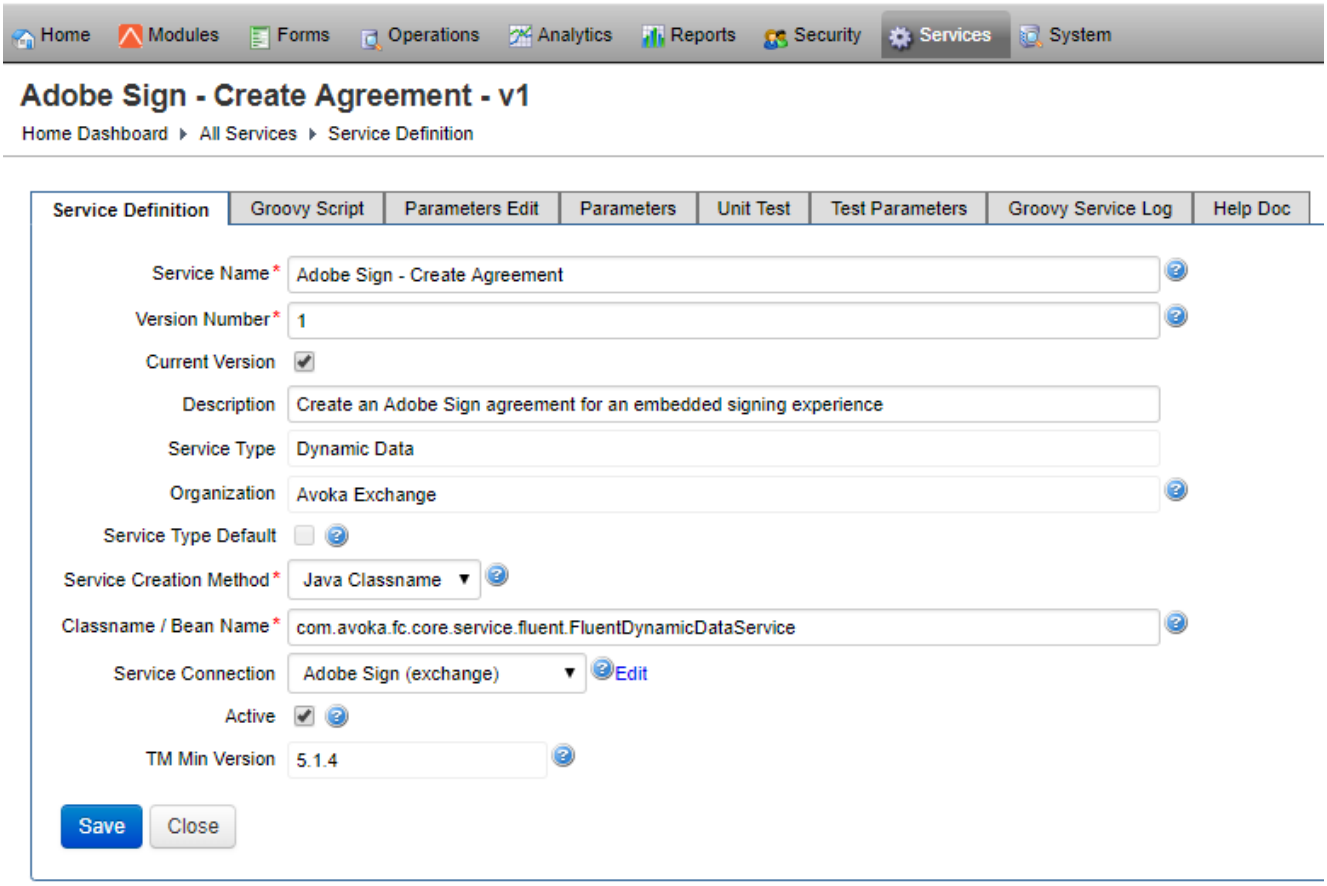
Manager comes with the dynamic data service (DDS) to retrieve required data dynamically, often from external systems. This service can be called by forms to obtain data dynamically while they are being filled in, for example, data lookup or validation using external services. The dynamic data service performs a lookup using a **Groovy**[1] script configured as one of the service's parameters. You can easily change the lookup behavior by modifying this service parameter, without a restart or rebuild of your Manager server.

Maestro forms use DDS as one of the methods to obtain form data dynamically. For more information, see dynamic data in forms.

To configure the dynamic data service:

1. Select **Services > All Services**.
2. Locate a service and click **Edit**.
3. Configure the standard service settings using the **Dynamic Data** service type as the **Service Type** and the **Fluent Dynamic Data** template as the **Service Template**. Select a relevant service connection to use to connect to the external service.

[1] Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.



4. Select the **Groovy Script** tab and update a Groovy script to make use of a service connection. You can use either Transact Fluent API or Core Transact API depending on the service template you have selected. The snippets of both APIs are shown below:

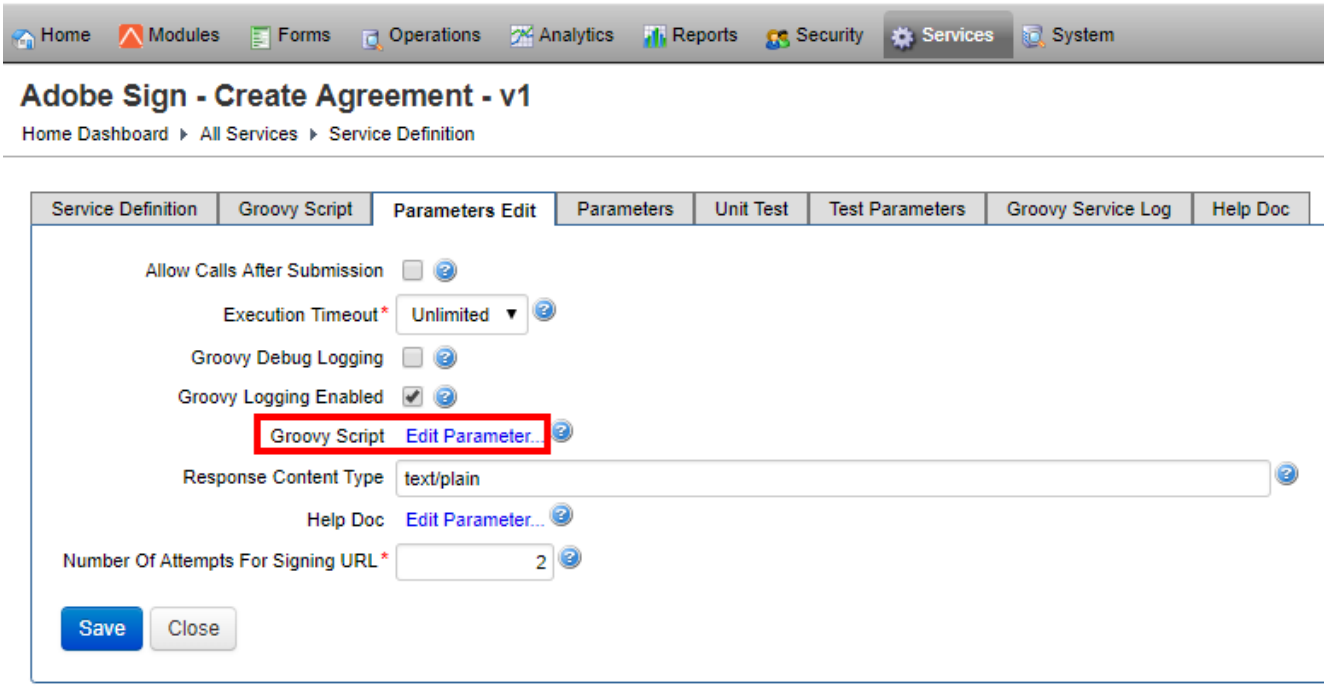
Transact Fluent API

```
String invoke(SvcDef svcDef, Txn txn, HttpServletRequest request, User user) {
       ...
       def endPoint = svcDef.svcConn.endpoint
       ...
}
```

Core Transact API

```
def serviceConnection = serviceDefinition.connection
def endPoint = serviceConnection.endpointValue
```

5. Select the **Parameters Edit** tab to edit the service configuration.
6. Click **Groovy Script Edit Parameter** to create and update parameters used withing the script.



You can retrieve a parameter using the following code:

Transact Fluent API

```
String invoke(SvcDef svcDef, Txn txn, HttpServletRequest request, User user) {
	...
	def zipConnectionName = svcDef
		.paramsMap
		.get('GetInfoByZip Connection Name')
	...
}
```

Core Transact API

```
def zipConnectionName = serviceParameters.get('GetInfoByZip Connection Name')
```

7. Configure the Unit Test and Test Parameters tabs.
8. Click **Save** to update the changes.

DDS services require a request log key, which is session specific, to be provided unless the services run in test mode. This provides some protection against **CSRF**[1]. DDS services are not directly related to forms, so you can call any within your organization unless the setting to enable calling after submission is set the submission related to the requestlog are not be in a finished state. Form Functions has a stronger security model when they are mapped to the form definition. For more information CSRF, see how to configure modules.

Next, learn how to configure a service with multiple endpoints to external services.

[1]Cross-site request forgery (CSRF), also known as one-click attack or session riding, is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the web application trusts.

Submission Completed Processor Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create a service of the submission completed processor type, which is called immediately after a form submission has progressed to form status **Completed**.

To configure the submission completed processor service:

1. Select **Services > All Services**.
2. Locate a service and click **Edit**.
3. Configure the standard service settings.
4. Click **Save** to update the changes.

For more information, see groovy-services-api/submission-completed-processor.html.

Task Cancellation Processor Service

You can create a service of the same type to cancel or abandon a task submission or a task that is a part of a collaboration job. This service will use the following job controller API:

```
/**
* @see IJobController#cancelJob(Job)
* @param job the job instance to cancel (required)
* @return true if the job was cancelled, or false if the job was already finished
*/
public boolean cancelJob(final Job job)
```

To configure the task cancellation processor service:

1. Select **Services > All Services**.
2. Locate a service and click **Edit**.

3. Configure the standard service settings.

4. Select the **Groovy Script** tab and update a **Groovy**[1] script as shown below:

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

```
import com.avoka.component.xml.XmlDoc
import com.avoka.fc.core.dao.DaoFactory
import com.avoka.fc.core.service.ServiceFactory
import com.avoka.fc.core.entity.Job
import com.avoka.fc.core.service.ServiceLocator


def XPATH_TACKING_CODES = '//Get-
tingStarted/TrackingCode/rptTrackingCode/TrackingCode'
def PIPE_SEPARATOR = '||'
def xmlDoc = new XmlDoc(submissionXml)
def submissionDao = DaoFactory.getSubmissionDao()
def jobControllerService = new JobControllerService()
def abandonService = ServiceFactory.getAbandonmentService()
def cancelledTasks = []
def notCancelledTasks = []
def trackingCodeNodes = xmlDoc.getNodeList(XPATH_TACKING_CODES)
// Loop all given tracking codes

for (def node : trackingCodeNodes) {
       def isAbandoned = false
       def taskSubject = ''
       try {
               // Get the submission object
               def trackingCode = node.getTextContent()
               def oSubmission = submissionDao.getSubmissionByTrackingNumber(trackingCode)
               taskSubject = oSubmission.getTaskSubject()
               // Get the job that the submission is part of, if any
               def job = oSubmission.getJob()
               // Check if submission is part of collaboratino job
               if (job != null) {
                       // Submission is part of collaboration job
                       def jobController = ServiceLocator.getJobController(job)
                       isAbandoned = jobController.cancelJob(job)
               } else {
                       // Submission is not part of collaboration job
                       isAbandoned = abandonService.performExplicitAdminAbandonment(oSubmission)
               }
       } catch (Exception e) {

       } finally {
```

```
        if (isAbandoned) {
                cancelledTasks.add(taskSubject)
        } else {
                notCancelledTasks.add(taskSubject)
        }
}
```

5. Configure the service in the Unit Test and Test Parameters tabs.
6. Click **Save** to update the changes.

Next, learn how to view job services.

Job Services Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Job services are specialized services used in Journey Manager to create and process collaboration jobs.

Manager provides different job service types and comes with pre-configured job services and job service templates to help you create new job services.

Job services are defined per an organization. Each organization must have at least one job service configured. There is no limit on the number of job services an organization can have. Each form within the organization can use these job services. You can create new job controllers and job actions to implement your requirements.

Manager provides job service templates, which are enhanced job controller services. Templates allow a user to customize collaboration jobs without having to modify the job service definition. Templates are organization agnostic and you can't edit them.

Next, learn how to view job services.

Job Definitions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to define and configure a job services, which processes collaboration jobs, using a single JSON[1] document. The job definition includes the list of steps that make up the collaboration job. Each of the individual steps has the list of actions to be performed when the step runs, and the routes property, which defines the next step transition by mapping the route name to the next step.

The job definition consists of the following properties:

- jobDetails
- jobGroups
- steps
- actions
- properties
- routes

jobDetails

The **jobDetails** property defines a collaboration job and how it is processed. The snippet of the **jobDetails** is shown below:

```
{
	"jobDetails": {
		"name": "2 Step Review Job",
		"processSubmitImmediate": "true",
		"version": "4.1.0"
	},
	"steps":[
		{ .. step definition ...},
		{ .. step definition ...},
		...
	]
}
```

The **jobDetails** has the following attributes:

Attribute Name	Description

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

name	The name of the collaboration job, which is set by the job controller service when the collaboration job has been saved.
processSubmitImmediate	The collaboration job processing mode. If this attribute is omitted or set to false, the job will use the standard mode. Otherwise, it will use the process immediate mode.
version	The version of Manager that the job was created in.

jobGroups

The **jobGroups** property lists collaboration job groups, so job coordinators can manage the collaboration job. The snippet of the **jobGroups** is shown below:

```
"jobGroups": {}
```

The **jobGroups** has the following attributes:

Attribute Name	Description
name	The name of the job group, which will be displayed in the Job Groups dropdown from the Reviews tab of the form space.
description	The description for the job group.

The job groups can be created in one of the following ways:

1. If a job group defined in a job definition, as in above, has not been created, then the job group will be automatically created the first time a user submits a form associated to this job. This is the preferred way to create job groups.
2. Job groups can be manually created. By manually creating the job group, you must ensure that the job group names and descriptions match with the job definition. Any mismatch will not be subsequently updated to the job groups.

steps

The **steps** property is an JSON array that defines steps. The snippet of the **steps** property is shown below:

```
{
      "name": "Initial Review",
      "type": "",
      "expiryRule": "+5d"
      "expiryServiceName": "An Expiry Service"
      "actions": [
        { ... Action 1 Definition ... },
        { ... Action 2 Definition ... }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" },
        { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "display":
"false" },
        { "name": "Terminate",  "nextStep": "Terminated Initial" }
      ]
},
```

The **steps** has the following attributes:

Attribute Name	Description
name	The name of the step, which is unique within a collaboration job.
type	The type of the step, that can be: • start - the first step for the collaboration job. "type":"start" • endpoint - the processing stops and the job completes. Endpoints do not specify actions, routes or customProperties. "type":"endpoint" • empty string. "type":""
expiryRule	The step scheduled completion time, the step expiry, calculation rule. If specified, the job_step.time_completion_scheduled is set as the step creation time plus the expiry rule value, for example, 2014-03-05 + 5d = 2014-03-10. The valid values are: • "expiryRule" : "+4h" (hours) • "expiryRule" : "+5d" (days) • "expiryRule" : "+4w (weeks) • "expiryRule" : "+4m". This is an optional attribute.

	We recommend setting it up to some default value so an action doesn't stay in the queue forever, if an associated form never abandons and the task is never submitted, even as a transaction expires.
expiryServiceName	Specifies a job expiry service to clean up any items, such as incomplete submissions. This service should specify a route. A route name doesn't have to include a work expiry.

If the **expiryRule** is used, but the **exprityServiceName** is not specified, then expiry route must be provided.

actions

The **actions** property defines the list of actions for each step, which are executed sequentially. Actions are required for all steps except endpoints. The snippet of the **actions** property is shown below:

```
{
    "name": "Create Task",
    "type": "Job Task Assign",
    "properties": [
        { "name": "Process Message Send Email",  "value": "true" },
        { "name": "Process Message Submission Step",  "value": "Start" },
        { "name": "Process Message Text",  "value": "Your ${submission.formName} applic-
ation is at the Initial Review Step" },
}
```

The **actions** has the following attributes:

Attribute Name	Description
name	The name of the action, which is unique within a collaboration job.
type	The action service type, that must be: • listed above in the Standard Actions Services. • groovy action services can have a type of Job Action.
properties	The action specific properties, see the **properties** object definition below. This is an optional attribute.

The standard Action Property classes (**com.avoka.fc.core.service.job.impl**) define a fixed set of Action Properties. The **JobTaskAssignService** defines a broad range of separate properties, so you can add them to your script when creating a new job action. This makes a job action re-useable across a collaboration job and the whole form application. For example, if the job definition has this property:

```
"properties": [
   { "name": "ABC Widget Name",  "value": "Aplle Blue Catapillar" },
   ...
 ]
```

Then, you can retrieve it using the Groovy code below:

```
def abcName = actionStepProperties.getProperty("ABC Widget Name")
```

However, you should consider the following:

- There is no submission associated with the job actions, by default.
- **$formDataMap.name** doesn't resolve as the job action doesn't have any concept of a submission.
- **$func.previousSubmission().formDataMap.name** works as we have used the submission from the previous step.
- **$job.submission(0).name** works as we have used the first submission linked to the collaboration job.
- This is not supported in Fluent job action classes version 17.10.1.

You can use various job functions in a job definition as well.

properties

The **properties** property defines the job definition properties. The snippet of the **properties** property is shown below:

```
"properties": [
    { "name": "Task Assign Group",  "value": "$func.formProperty('Initial Review')" },
    { "name": "Task Type",  "value": "Review" },
]
```

Properties can be used to pass information into an instance of the Action Service. In this case, they are called action properties and can be defined at the action, step or collaboration job level. Properties with the same name defined at an action level override properties defined at the step level. Step override properties of the same name at the job level. This is illustrated in the snippet below:

```
{
  "jobDetails": { ... },
  "properties": [
    { "name": "Task Form Code", "value": "$func.startFormCode()" }
  ],
  "steps":[
    {
      "name": "Initial Review",
      "type": "",
      "actions": [
        {
          "name": "Create Task",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Process Message Text",  "value": "Your ${submission.formName}
application is at the Initial Review Step" },
            { "name": "Task Assign Group",  "value": "$func.formProperty('Initial
Review')" },
            { "name": "Task Message",  "value": "Please perform the initial review of the
${submission.formName} from ${formDataMap.firstName} ${formDataMap.lastName}." },
            { "name": "Task Review Previous Step",  "value": "true" },
            { "name": "Task Subject",  "value": "Initial review of ${submission.formName}
from ${formDataMap.firstName} ${formDataMap.lastName}" },
            { "name": "Task Type",  "value": "Review" },
          ]
        },
        { ... }
      ],
      "properties": [
        { "name": "Task Form Code", "value": "$func.startFormCode()" }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" }
      ],
    },
    { ... }
  ]
}
```

The **properties** has the following attributes:

Attribute Name	Description

name	The name of the property. The predefined action services, such as the job task assign service, have a set number of action properties
value	A property value that can include an Apache Velocity template. The action properties are templates that are evaluated when an action service is run. The templates can be: • static text values, for example, **`{ "name": "Task Email Message", "value": "An Email Message" }`** • substitute values from the form data extract, for example, **`{ "name": "Process Message Text", "value": "Your ${job.submission.formName} application is at the Initial Review Step" }`**
template	It evaluates the value as a velocity template and forces the return value to return a String. This can evaluate nested function calls. This is can be used with property values that start with "$func", these can return an object. For example, **`$func.startUser()`** will return a user. **`{ "name": "Task Email Message", "value": "$func.getSubmission().formXml()", "template": "true" }`**

routes

The **routes** property defines the route mapping between the route name and the next step, which are required for all steps except endpoints. An action can return a route result name, as a string, which is listed in each step. If a valid name is returned, the step routes the collaboration job to the next step. However, if the action returns an invalid route name, an error is thrown. The snippet of the **routes** property is shown below:

```
"routes": [
   { "name": "Approve",  "nextStep": "Application Delivery" },
   { "name": "Decline",  "nextStep": "Applicant Update" },
   { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "display": "false"
},
   { "name": "Terminate",  "nextStep": "Terminated Initial" }
]
```

The **routes** has the following attributes:

Attribute Name	Description
name	The route name that can be any name except the following reserved names:

	• Default - If there is no step found with a route name or if the route name is empty, the default route is used. • Expiry - the route taken when the step has expired. The next step, that can be:
nextStep	• A valid step name contained in the job definition. • A special case is ##PREVIOUS_STEP the job processing assign the next step as the previous step name.
display	The collaboration job provides a list of available routes to the form. However, the **`"display": "false"`** removes the route from the available routes. This is an optional attribute.

You can see the complete example of a job definition here.

Next, learn how to configure job definition.

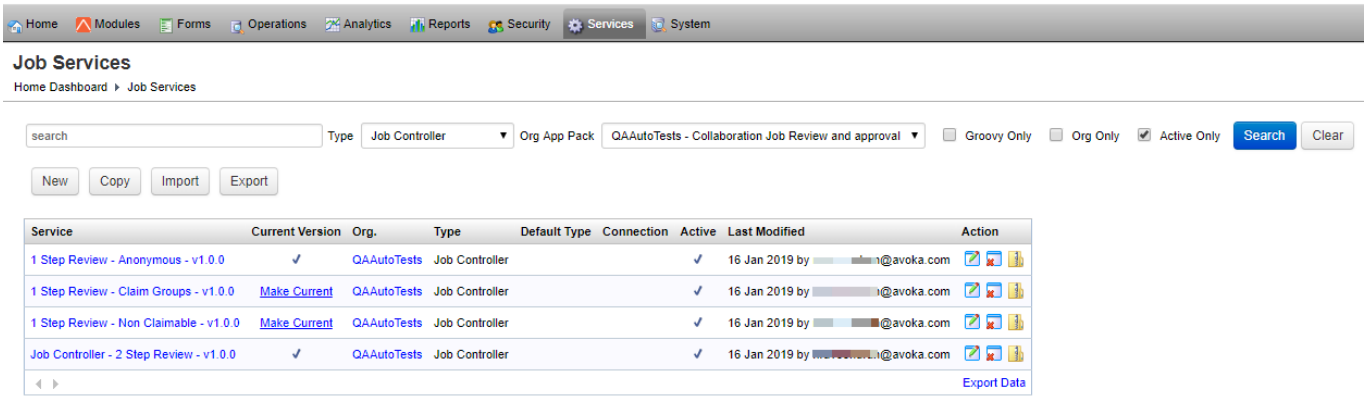
View Job Services

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view all job services, so you can easily find an existing job service.

To view job services, select **Services > Job Services.**

Job services are shown page by page, with the maximum number of records per page configured in the users preferences.



To filter or search the services by one or more criteria, specify the following settings and click **Search**:

- Service name or part of a name
- Service type
- Org App Pack
- Groovy Only to see only **Groovy**[1] services
- Org Only
- Active Only

Click **Clear** to reset the search criteria.

You can check the following service details:

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

- Service name
- Current Version
- Organization
- Type
- Default Type represents whether this service is the default service for this type of service.
- Connection is the service connection associated with this service, if it is configured.
- Active shows whether this service is active or not. It must be active to be available.

> It is recommended to make a service not active rather then remove it when you don't need it any more.

- Last Modified shows the date when this service was last modified and by whom.

You can also perform the following actions:

- Create a service
- View and configure a service
- Copy a service
- Remove a service
- Export services
- Import services

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Configure Job Definition

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to define how a job service processes collaboration jobs by describing an execution flow of steps and actions in the job definition.

To configure a job definition of a service:

1. Locate an existing job service.
2. Select the **Job Definition** tab to review or update the JSON[1] file defining this service.



3. Select the **Validate** checkbox to validate action services, step properties, form codes and custom services defined in the JSON file. It is selected by default, but you can clear it, if you want to add a new action but you haven't created a form or a custom service for this, and you still need to save your changes.

---

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

4. Select the **Auto Format** checkbox to automatically format the job definition and strip unknown elements.
5. Click **Save** to update the changes..

Next, learn how to view all job services of a job controller.

Job Definitions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to define and configure a job services, which processes collaboration jobs, using a single JSON[1] document. The job definition includes the list of steps that make up the collaboration job. Each of the individual steps has the list of actions to be performed when the step runs, and the routes property, which defines the next step transition by mapping the route name to the next step.

The job definition consists of the following properties:

- jobDetails
- jobGroups
- steps
- actions
- properties
- routes

jobDetails

The **jobDetails** property defines a collaboration job and how it is processed. The snippet of the **jobDetails** is shown below:

```
{
	"jobDetails": {
		"name": "2 Step Review Job",
		"processSubmitImmediate": "true",
		"version": "4.1.0"
	},
	"steps":[
		{ .. step definition ...},
		{ .. step definition ...},
		...
	]
}
```

The **jobDetails** has the following attributes:

Attribute Name	Description

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

name	The name of the collaboration job, which is set by the job controller service when the collaboration job has been saved.
processSubmitImmediate	The collaboration job processing mode. If this attribute is omitted or set to false, the job will use the standard mode. Otherwise, it will use the process immediate mode.
version	The version of Manager that the job was created in.

jobGroups

The **jobGroups** property lists collaboration job groups, so job coordinators can manage the collaboration job. The snippet of the **jobGroups** is shown below:

```
"jobGroups": {}
```

The **jobGroups** has the following attributes:

Attribute Name	Description
name	The name of the job group, which will be displayed in the Job Groups dropdown from the Reviews tab of the form space.
description	The description for the job group.

The job groups can be created in one of the following ways:

1. If a job group defined in a job definition, as in above, has not been created, then the job group will be automatically created the first time a user submits a form associated to this job. This is the preferred way to create job groups.
2. Job groups can be manually created. By manually creating the job group, you must ensure that the job group names and descriptions match with the job definition. Any mismatch will not be subsequently updated to the job groups.

steps

The **steps** property is an JSON array that defines steps. The snippet of the **steps** property is shown below:

```
{
      "name": "Initial Review",
      "type": "",
      "expiryRule": "+5d"
      "expiryServiceName": "An Expiry Service"
      "actions": [
        { ... Action 1 Definition ... },
        { ... Action 2 Definition ... }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" },
        { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "display":
"false" },
        { "name": "Terminate",  "nextStep": "Terminated Initial" }
      ]
},
```

The **steps** has the following attributes:

Attribute Name	Description
name	The name of the step, which is unique within a collaboration job.
type	The type of the step, that can be: • start - the first step for the collaboration job. "type":"start" • endpoint - the processing stops and the job completes. Endpoints do not specify actions, routes or customProperties. "type":"endpoint" • empty string. "type":""
expiryRule	The step scheduled completion time, the step expiry, calculation rule. If specified, the job_step.time_completion_scheduled is set as the step creation time plus the expiry rule value, for example, 2014-03-05 + 5d = 2014-03-10. The valid values are: • "expiryRule" : "+4h" (hours) • "expiryRule" : "+5d" (days) • "expiryRule" : "+4w (weeks) • "expiryRule" : "+4m". This is an optional attribute.

	We recommend setting it up to some default value so an action doesn't stay in the queue forever, if an associated form never abandons and the task is never submitted, even as a transaction expires.
expiryServiceName	Specifies a job expiry service to clean up any items, such as incomplete submissions. This service should specify a route. A route name doesn't have to include a work expiry.

If the **expiryRule** is used, but the **exprityServiceName** is not specified, then expiry route must be provided.

actions

The **actions** property defines the list of actions for each step, which are executed sequentially. Actions are required for all steps except endpoints. The snippet of the **actions** property is shown below:

```
{
    "name": "Create Task",
    "type": "Job Task Assign",
    "properties": [
        { "name": "Process Message Send Email",  "value": "true" },
        { "name": "Process Message Submission Step",  "value": "Start" },
        { "name": "Process Message Text",  "value": "Your ${submission.formName} applic-
ation is at the Initial Review Step" },
}
```

The **actions** has the following attributes:

Attribute Name	Description
name	The name of the action, which is unique within a collaboration job.
type	The action service type, that must be: • listed above in the Standard Actions Services. • groovy action services can have a type of Job Action.
properties	The action specific properties, see the **properties** object definition below. This is an optional attribute.

The standard Action Property classes (**com.avoka.fc.core.service.job.impl**) define a fixed set of Action Properties. The **JobTaskAssignService** defines a broad range of separate properties, so you can add them to your script when creating a new job action. This makes a job action re-useable across a collaboration job and the whole form application. For example, if the job definition has this property:

```
"properties": [
   { "name": "ABC Widget Name",  "value": "Aplle Blue Catapillar" },
   ...
 ]
```

Then, you can retrieve it using the Groovy code below:

```
def abcName = actionStepProperties.getProperty("ABC Widget Name")
```

However, you should consider the following:

- There is no submission associated with the job actions, by default.
- **$formDataMap.name** doesn't resolve as the job action doesn't have any concept of a submission.
- **$func.previousSubmission().formDataMap.name** works as we have used the submission from the previous step.
- **$job.submission(0).name** works as we have used the first submission linked to the collaboration job.
- This is not supported in Fluent job action classes version 17.10.1.

You can use various job functions in a job definition as well.

properties

The **properties** property defines the job definition properties. The snippet of the **properties** property is shown below:

```
"properties": [
    { "name": "Task Assign Group",  "value": "$func.formProperty('Initial Review')" },
    { "name": "Task Type",  "value": "Review" },
]
```

Properties can be used to pass information into an instance of the Action Service. In this case, they are called action properties and can be defined at the action, step or collaboration job level. Properties with the same name defined at an action level override properties defined at the step level. Step override properties of the same name at the job level. This is illustrated in the snippet below:

```
{
  "jobDetails": { ... },
  "properties": [
    { "name": "Task Form Code", "value": "$func.startFormCode()" }
  ],
  "steps":[
    {
      "name": "Initial Review",
      "type": "",
      "actions": [
        {
          "name": "Create Task",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Process Message Text",  "value": "Your ${submission.formName}
application is at the Initial Review Step" },
            { "name": "Task Assign Group",  "value": "$func.formProperty('Initial
Review')" },
            { "name": "Task Message",  "value": "Please perform the initial review of the
${submission.formName} from ${formDataMap.firstName} ${formDataMap.lastName}." },
            { "name": "Task Review Previous Step",  "value": "true" },
            { "name": "Task Subject",  "value": "Initial review of ${submission.formName}
from ${formDataMap.firstName} ${formDataMap.lastName}" },
            { "name": "Task Type",  "value": "Review" },
          ]
        },
        { ... }
      ],
      "properties": [
        { "name": "Task Form Code", "value": "$func.startFormCode()" }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" }
      ],
    },
    { ... }
  ]
}
```

The **properties** has the following attributes:

Attribute Name	Description

name	The name of the property. The predefined action services, such as the job task assign service, have a set number of action properties
value	A property value that can include an Apache Velocity template. The action properties are templates that are evaluated when an action service is run. The templates can be: • static text values, for example, **`{ "name": "Task Email Message", "value": "An Email Message" }`** • substitute values from the form data extract, for example, **`{ "name": "Process Message Text", "value": "Your ${job.submission.formName} application is at the Initial Review Step" }`**
template	It evaluates the value as a velocity template and forces the return value to return a String. This can evaluate nested function calls. This is can be used with property values that start with "$func", these can return an object. For example, **`$func.startUser()`** will return a user. **`{ "name": "Task Email Message", "value": "$func.getSubmission().formXml()", "template": "true" }`**

routes

The **routes** property defines the route mapping between the route name and the next step, which are required for all steps except endpoints. An action can return a route result name, as a string, which is listed in each step. If a valid name is returned, the step routes the collaboration job to the next step. However, if the action returns an invalid route name, an error is thrown. The snippet of the **routes** property is shown below:

```
"routes": [
   { "name": "Approve",  "nextStep": "Application Delivery" },
   { "name": "Decline",  "nextStep": "Applicant Update" },
   { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "display": "false"
},
   { "name": "Terminate",  "nextStep": "Terminated Initial" }
]
```

The **routes** has the following attributes:

Attribute Name	Description
name	The route name that can be any name except the following reserved names:

	• Default - If there is no step found with a route name or if the route name is empty, the default route is used. • Expiry - the route taken when the step has expired. The next step, that can be:
nextStep	• A valid step name contained in the job definition. • A special case is ##PREVIOUS_STEP the job processing assign the next step as the previous step name.
display	The collaboration job provides a list of available routes to the form. However, the **`"display": "false"`** removes the route from the available routes. This is an optional attribute.

You can see the complete example of a job definition here.

Next, learn how to configure job definition.

View Job Info

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides you with a read only view of job info of a selected job service.

To check job info:

1. Locate a job service.
2. Select the **Job Info** tab to view job info of this service that includes:
   - Job name
   - Job steps
   - Job Form Data Maps, which lists the names of the parameters used in the form XML data.
3. Click **Close**.

Next, learn how to configure service parameters.

Job Controller Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager uses job controller services to:

- Create a collaboration job instance when a user submits a form or the API is called.
- Process collaboration jobs by executing their steps and actions defined in a job definition.

Manager allows you to create and configure job controller services to implement your customer requirement.

To create a job controller service:

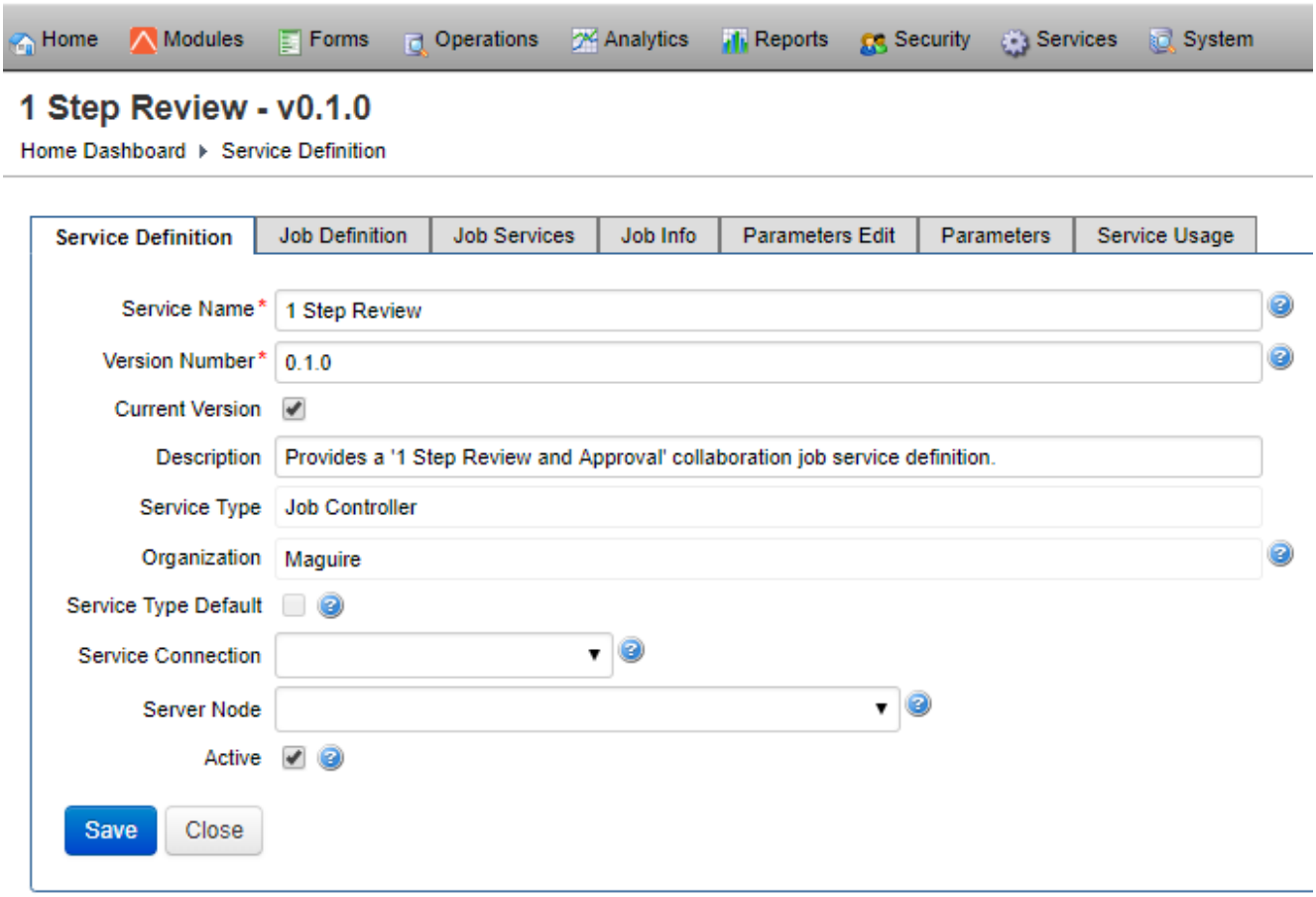
1. Select **Services > Job Services**.
2. Click **New** to create a job service.
3. Select the **Job Controller** as a service type from the **Service Type** dropdown list.
4. Select a service template from the **Service Template** dropdown list, depending on your workflow, for example, **Job Controller - 1 Step Review**.
5. Select an organization from the **Organization** dropdown list.

   You have to assign a job controller to an organization. If you have several organizations that would like to use the same job controller, you need to define one for each organization.

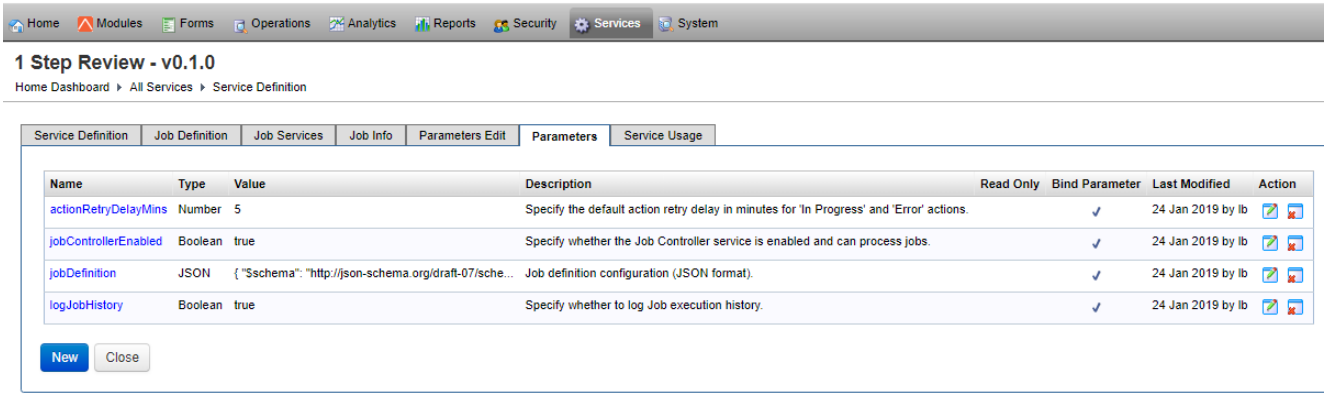
6. Click **Save** to update the changes.

To configure a job controller service:

1. Locate a job service.
2. Configure the service definition.



3. Configure the job definition.
4. Check the job services used in this job controller.
5. Check the job info.
6. Create service parameters required for this job controller to run.



The standard service parameters are:

- **actionRetryDelayMins** - specifies the default action retry delay in minutes for 'In Progress' and 'Error' actions. Default is 5. This is a bind parameter.
- **jobControllerEnabled** - specifies whether the job controller service is enabled and it can process collaboration jobs. This is useful for debugging and test in non-production, and selectively enabling certain collaboration jobs in production environments. Default is true. This is a bind parameter.
- **jobDefinition** - job definition configuration in **JSON**[1] format. This is a bind parameter.
- **logJobHistory** - specifies whether to write collaboration job execution history in a log file or not. Default is true. This is a bind parameter.

  The optional service parameters are:

- **jobDiagram** - contains the collaboration job diagram format created by the external NextStep tool.
- **Form Property Editor Definition** - the job form properties editor definition in JSON format.

7. Check the job controller service usage.

Next, learn how to configure a job controller in a form.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

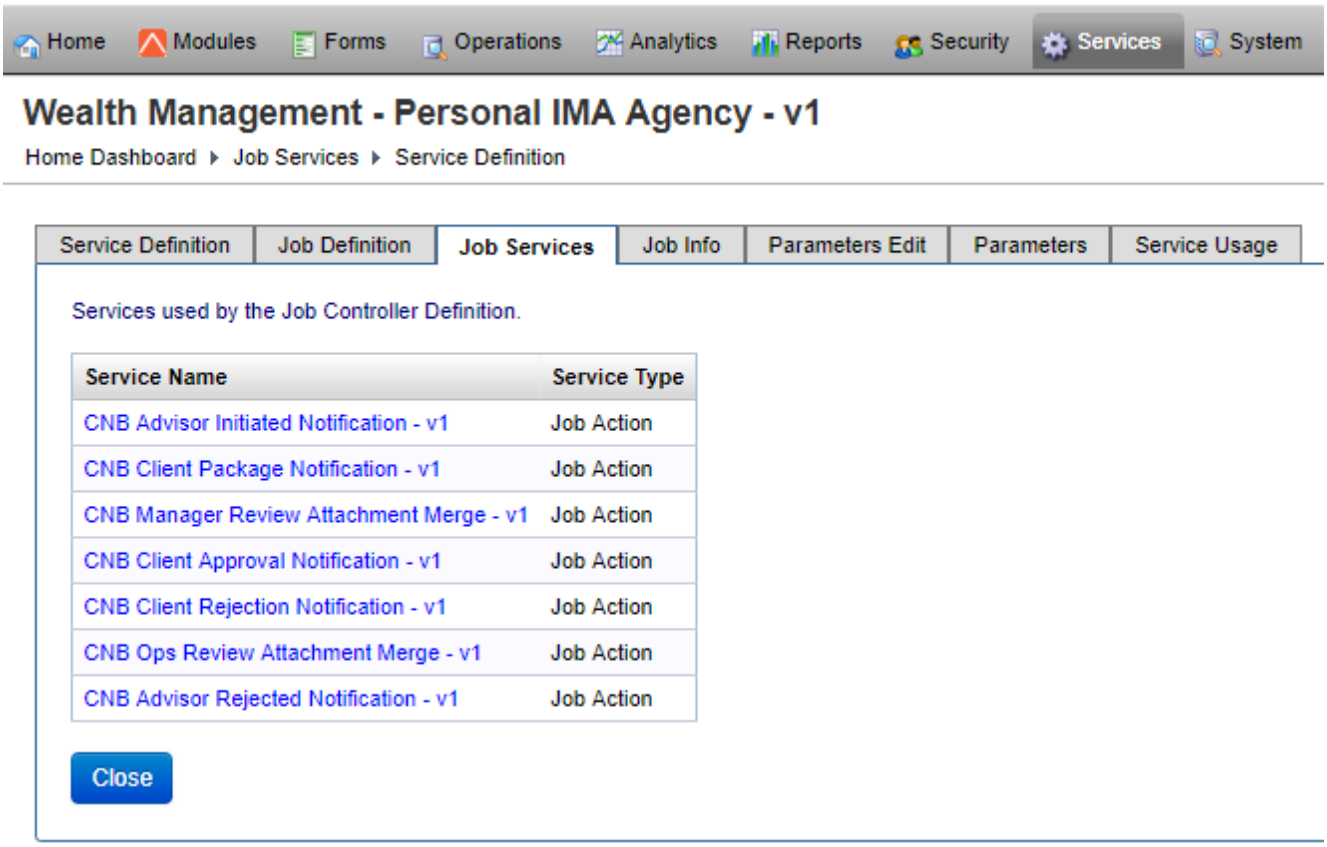
View Job Controller Services

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to viewjob services used by a job controller.

To view the list of job services:

1. Locate an existing job controller.
2. Select the **Job Services** tab to list all job services, which are configured in the job definition. The list is empty if there are no service configured in the JSON[1] file.



3. Click a service name to check its configuration.

Next, learn how to check job info.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

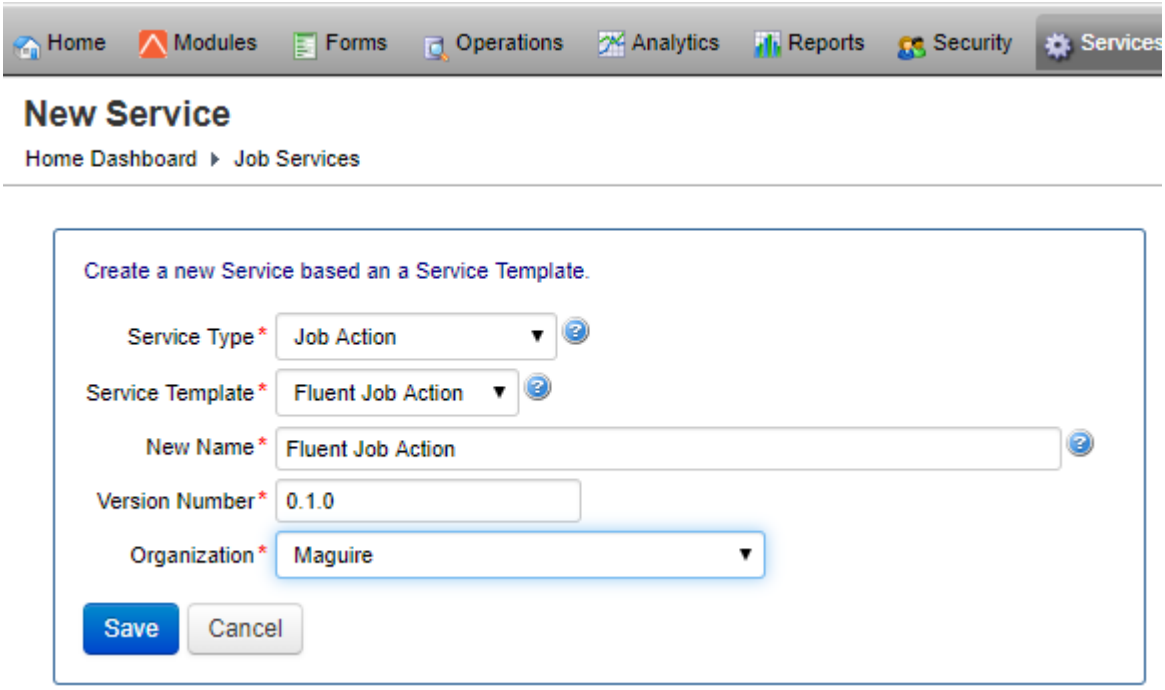
Job Action Services

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure job action services. When a collaboration job's action is processed, an action service is run. A collaboration job usually specifies an action service from one of the available inbuilt action services. This allows a developer to create their own job action and job expiry services using **Groovy**[1] scripts.

To create a job action service:

1. Select **Services > Job Services**.
2. Click **New** to create a job service.



3. Select the **Job Action** as a service type from the **Service Type** dropdown list.
4. Select the **Fluent Job Action** template from the **Service Template** dropdown list.
5. Select an organization from the **Organization** dropdown list.

> You have to assign a job controller to an organization. If you have several organizations that would like to use the same job controller, you need to define one for each organization.

6. Click **Save** to update the changes.

To configure a job action service:

---

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

1. Locate a job action service.
2. Configure the service definition.
3. Select the new service type to assign a Fluent or Groovy job action to a new type, such as **Job Expiry Service** or **Job Task Wait**.
4. Configure the job definition. You can configure an action to call a Groovy service from a job controller, as shown below.

```
"actions": [
  {
      "name": "Individual Tasks",
      "type": "Job Task Assign",
      "properties": [
         { "name": "Task Assign Repeating",  "value": "true" },
         { "name": "Task Assign Repeat Item",  "value": "$func.invoke('Repeating Get
Item', ${formDataMap.itemDelimited}, ${assignRepeatIndex})" },
         { "name": "Task Type",  "value": "Review" }
      ]
   }
```

   The **`$func.invoke`** method executes the **Repeating Get Item** service (invoking it by the name) with two parameters.

   If you want to invoke same job action multiple times as part of single step, give them different job action names, for example, **`"name": "Individual Tasks 2"`**, but define the same service name, for example,**`"serviceName": "Repeating Claimable"`**. If you do not define the service name explicitly, it expects the job action name to match the service name. It's not allowed to have 2 or more job actions with the same name in one job step.

5. Click the **Job Services** tab to view the **Repeating Get Item** service used in the action definition.

6. Click on the service name to open this service's Groovy script.

```
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import groovy.transform.TypeChecked
import javax.servlet.http.*
import org.apache.commons.lang3.StringUtils

@TypeChecked
class FluentGroovyService {

    // Injected at runtime
    public Logger logger

    /*
     * Perform a groovy service invocation
     *
     * return: the result object
     */
    Object invoke(SvcDef svcDef, HttpServletRequest request, User user, Map params)
{
        JobAction jobAction = (JobAction) params.jobAction
        return invoke(svcDef, request, user, jobAction, params)
    }

    Object invoke(SvcDef svcDef, HttpServletRequest request, User user, JobAction
jobAction, Map params) {
        logger.info params
        List<String> args = (List<String>) params.args

        def result = "Not Found"

        if( StringUtils.isNotEmpty(args[0]) &&
           StringUtils.isNotEmpty(args[1]) &&
           args[1].isNumber()) {
            def itemDelimited = args[0]
            // note repeat index from TM is 1 based not 0 based
            def repeatIndex = args[1].toInteger() - 1
            logger.info "itemDelimited: " + itemDelimited
            logger.info "repeatIndex: " + repeatIndex

            def items = itemDelimited.tokenize('|')
            logger.info "items: " + items
```

```
            result = items[repeatIndex.toInteger()]
            logger.info "result Item: " + result
        }

        return result

    }
}
```

7. Click **Save** to update the changes.

You can download the complete code example here.

Next, learn how to view all job services.

Job Service Templates

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides job service templates, which are enhanced job controller services. Templates allow a user to customize collaboration jobs without having to modify the job service definition. Templates are organization agnostic and you can't edit them. Templates are predominantly used for reviews and approvals, so Manager comes with four default review and approval templates:

- Review and Approval - 1 Step Group Review - v1.0.0
- Review and Approval - 1 Step User Review - v1.0.0
- Review and Approval - 2 Step Group Review - v1.0.0
- Review and Approval - Multi Step Group Review - v1.0.0

The job service templates are especially useful for:

- Business users and form builders to configure their forms without a developer.
- Greater reuse of a single job controller.

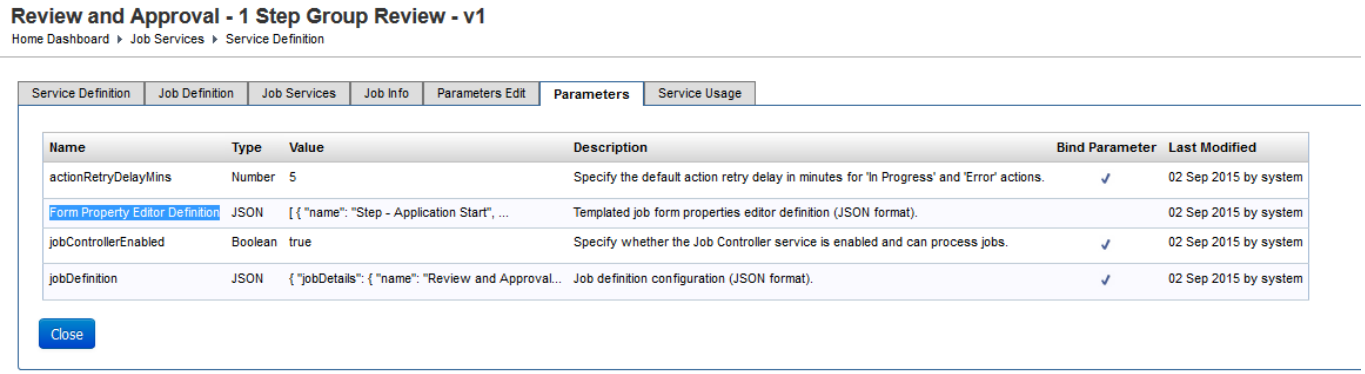
To use a job service template:

1. Locate a form version.
2. Click the **Services** tab to view services configured for this form.
3. Select a job service template from the **Job Controller Service** dropdown list, which has the following available templates:
   - Review and Approval - 1 Step Group Review - v1.0.0
   - Review and Approval - 1 Step User Review - v1.0.0
   - Review and Approval - 2 Step Group Review - v1.0.0
   - Review and Approval - Multi Step Group Review - v1.0.0
4. Click **Save** to update the changes.
5. Select the **Job Properties** tab to configure the collaboration job's properties.
6. Select the **Job Reviewers** from the **Reviewer Group** dropdown list.
7. Select the **Job Example 1** from the **Review Form** dropdown list.
8. Click **Save** to update the changes.

A job service template has the **Form Property Editor Definition** service parameter, which is a **JSON**[1] file containing the form properties meta data.

---

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.



You can create a new custom template by adding the **Form Property Editor Definition** parameter to a standard job controller.

Next, learn about job controller services.

Job Task Wait Action Services

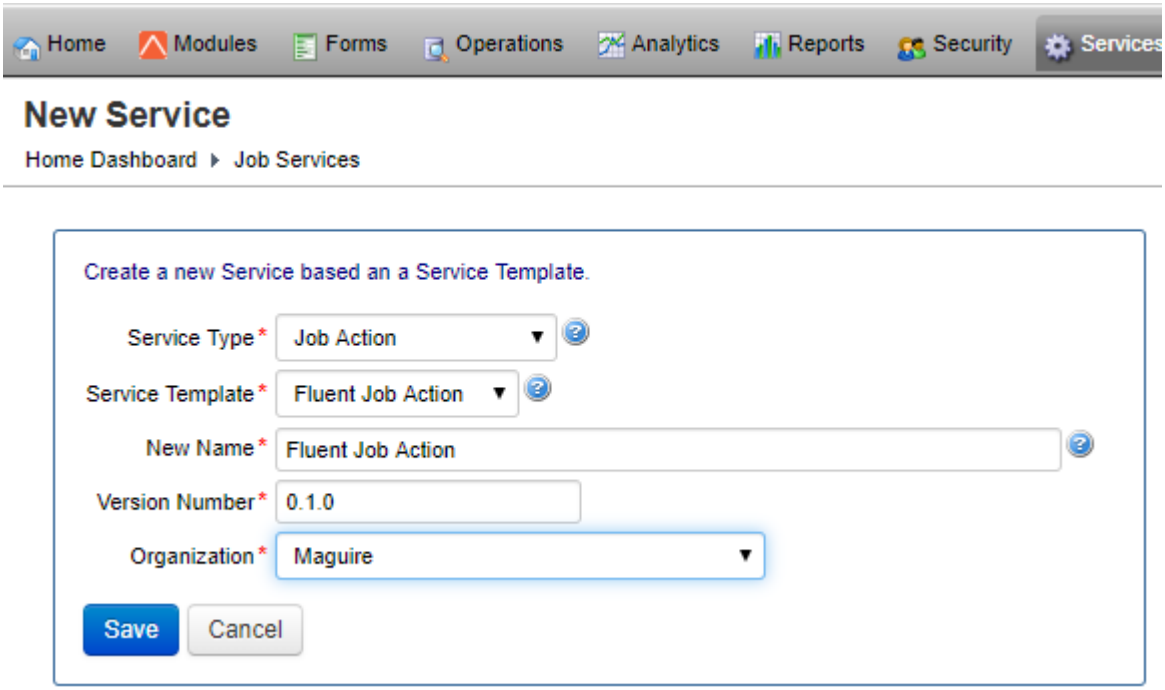
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A job task wait action service is a specialized job action service, which blocks the execution of a collaboration job until a specified event takes place.

Manager allows you to create and configure job task wait action services.

To create a job action service:

1. Select **Services > Job Services**.
2. Click **New** to create a job service.



3. Select the **Job Action** as a service type from the **Service Type** dropdown list.
4. Select the **Fluent Job Action** template from the **Service Template** dropdown list.
5. Select an organization from the **Organization** dropdown list.

> You have to assign a job controller to an organization. If you have several organizations that would like to use the same job controller, you need to define one for each organization.

6. Click **Save** to update the changes.

To configure a job action service:

1. Locate a job action service.
2. Configure the service definition.

3. Select the new service type to assign a Fluent or Groovy job action to a new type, such as **Job Expiry Service** or **Job Task Wait**.
4. Configure the job definition. You can configure an action to call a Groovy service from a job controller, as shown below.

```
"actions": [
        {
          "name": "Individual Tasks",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Task Assign Repeating",  "value": "true" },
            { "name": "Task Assign Repeat Item",  "value": "$func.invoke('Repeating
Get Item', ${formDataMap.itemDelimited}, ${assignRepeatIndex})" },
            ...
            { "name": "Task Type",  "value": "Review" }
          ]
        }
```

   The **`$func.invoke`** method executes the **Repeating Get Item** service (invoking it by the name) with two parameters.

5. Click the **Job Services** tab to view the **Repeating Get Item** service used in the action definition.

6. Click on the service name to open this service's Groovy script.

```
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import groovy.transform.TypeChecked
import javax.servlet.http.*
import org.apache.commons.lang3.StringUtils

@TypeChecked
class FluentGroovyService {

    // Injected at runtime
    public Logger logger

    /*
     * Perform a groovy service invocation
     *
     * return: the result object
     */
    Object invoke(SvcDef svcDef, HttpServletRequest request, User user, Map params)
{
        JobAction jobAction = (JobAction) params.jobAction
        return invoke(svcDef, request, user, jobAction, params)
    }

    Object invoke(SvcDef svcDef, HttpServletRequest request, User user, JobAction
jobAction, Map params) {
        logger.info params
        List<String> args = (List<String>) params.args

        def result = "Not Found"

        if( StringUtils.isNotEmpty(args[0]) &&
           StringUtils.isNotEmpty(args[1]) &&
           args[1].isNumber()) {
            def itemDelimited = args[0]
            // note repeat index from TM is 1 based not 0 based
            def repeatIndex = args[1].toInteger() - 1
            logger.info "itemDelimited: " + itemDelimited
            logger.info "repeatIndex: " + repeatIndex

            def items = itemDelimited.tokenize('|')
            logger.info "items: " + items
```

```
            result = items[repeatIndex.toInteger()]
            logger.info "result Item: " + result
        }

        return result

    }
}
```

7. Click **Save** to update the changes.

Next, learn how to view all job services.

Open the TaskUserWebService in SOAP UI

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to open the TaskUserWebService in SOAP UI. You should read the section 4 of the transact_web_service_guide_v4.2.pdf documentation prior to doing so.

The TaskUserWebService and TaskUserWebServiceV2 WSDL in Manager are located in a secure context path. This wiki article shows the workaround steps to successfully import the WSDL's and call methods in SOAP-UI.

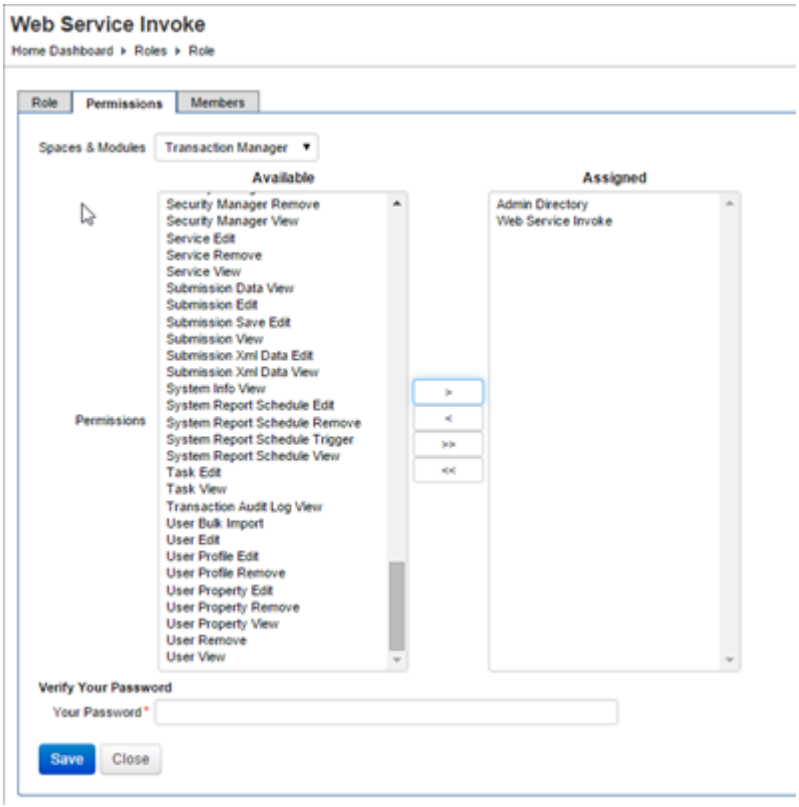
Task Web Services V2 requires you to authenticate using a username and password, It is possible to get the wsdl from the non secure context however you will be unable to call the secured services from the unsecured WSDL

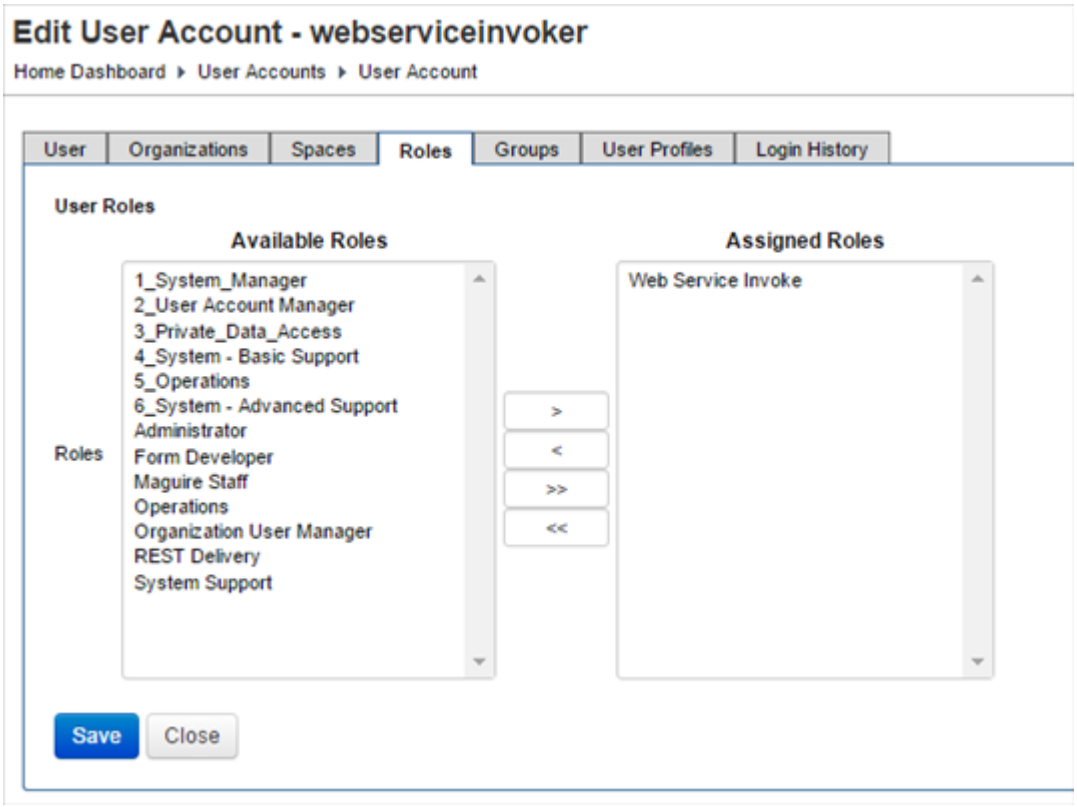
To open the service:

1. WSDL URL- https://{Server Name}/manager/secure/services/TaskUserWebServiceV2?wsdl

> Do not use the unsecure URL https://{Server Name}/manager/fcservices/TaskUserWebServiceV2?wsdl

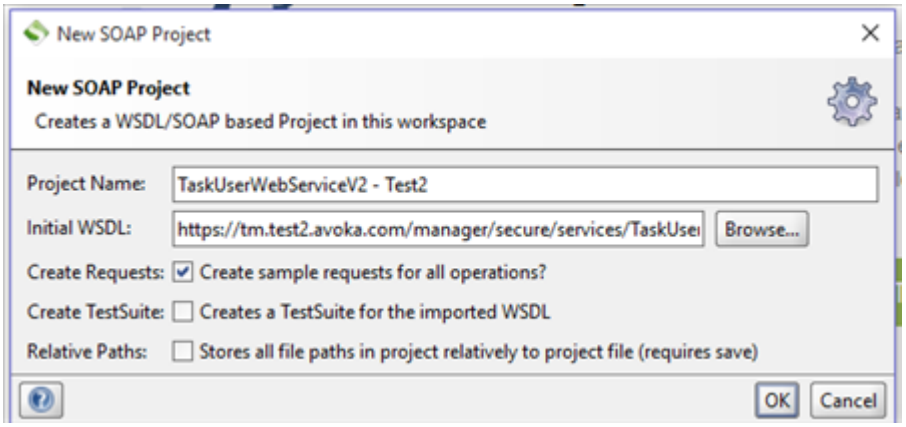
2. Create a service user that has been assigned a Role that has the Web Service Invoke permission.



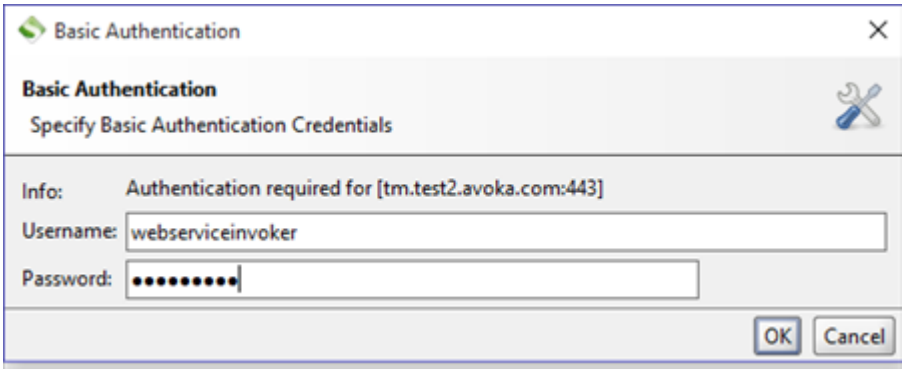


Users with the Adminitrator role have the Web Service Invoke permission.

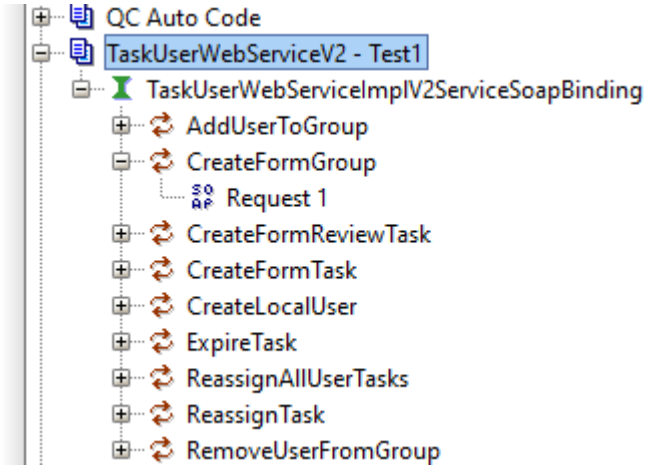
3. Create SOAP-UI project the wsdl in Step 1.



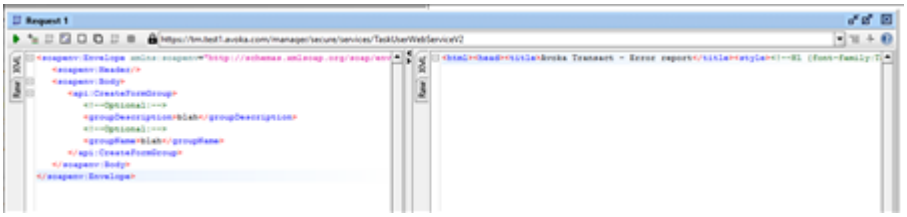
4. Soap-ui will prompt you with the Basic Authentication tab



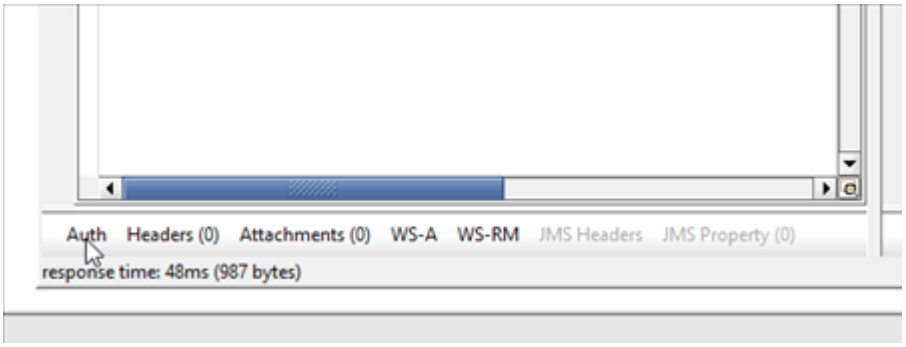
5. You should end up with the following



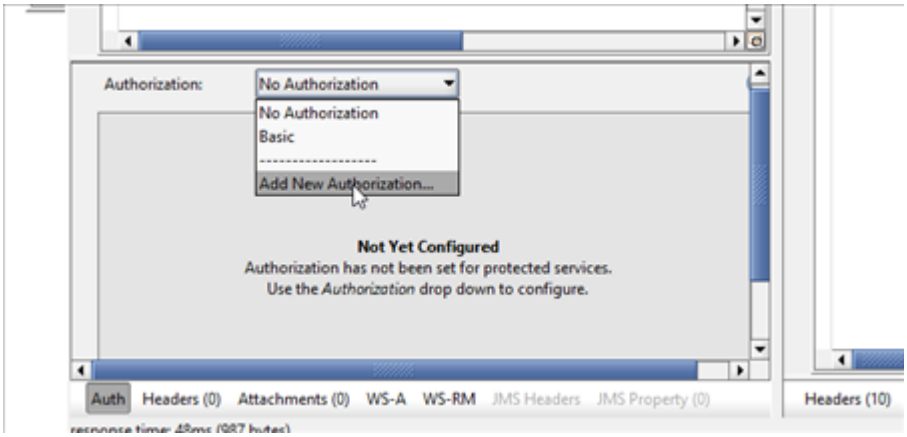
6. Try calling the web services with no authentication. Test Creating a Form Group. The response on the right tell you you are not authenticated.



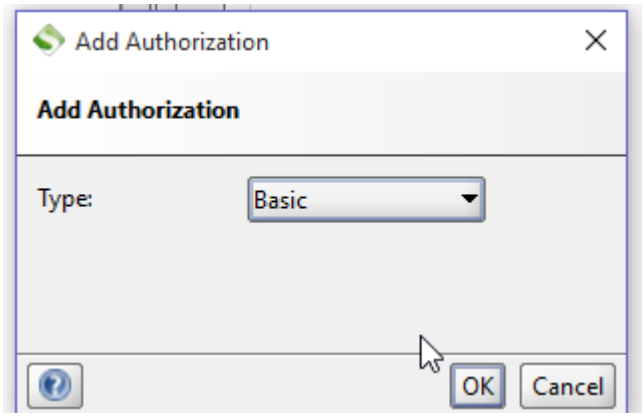
7. Click on the Auth buttong on the bottom left of the Request1 page to configure an Authentication.



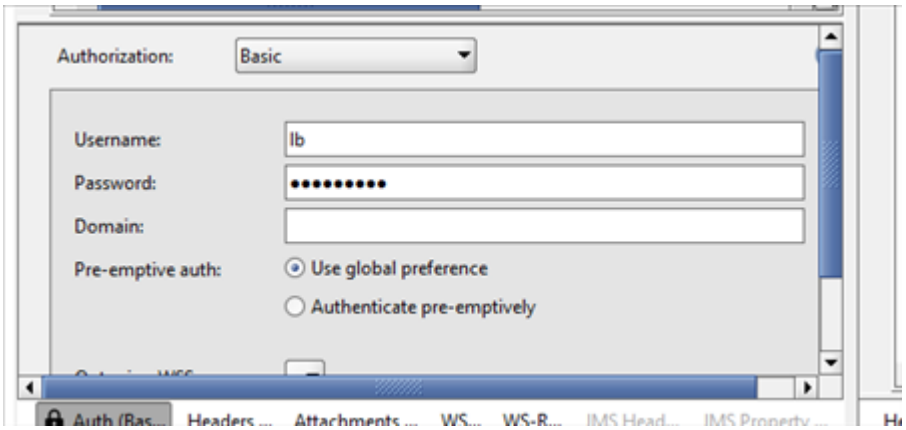
8. Select Add New Authentication.



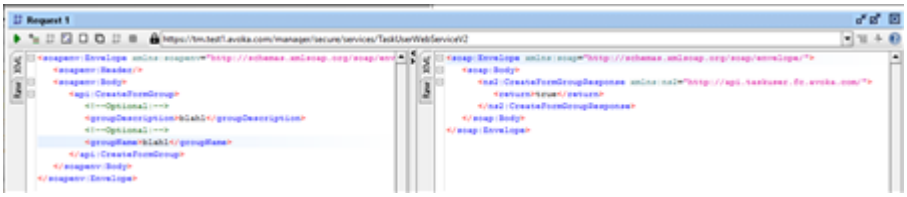
9. Select a type as Basic.



10. Specify a user name and a password.



11. Re-run the request.



Next, learn how to view services.

Scheduled Jobs Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager comes with the number of standard scheduled jobs that run regularly and perform routine operations, such as transaction processor schedule job that delivers transactions and performs data retention purging. A scheduled job invokes a schedule job service, which in turn calls other services that do the actual work.

One of the standard scheduled jobs is the collaboration job controller scheduled job, which processes standard collaboration jobs that are in the ready to be processed state.

In some cases, a particular solution may require custom scheduled jobs, so Manager allows you to create new scheduled jobs and configure them.

All scheduled jobs use a schedule that runs them automatically at regular intervals. You can pause and resume a scheduled job or trigger it manually.

Trigger Types

There are two trigger types - simple and cron.

Simple

A simple trigger type defines a number of times a scheduled job should be run and a time interval between each run. The interval can be set from 15 seconds up to 24 hours.

Cron

A **cron**[1] trigger type allows fine-grained control on when a scheduled job is executed. You must use cron expressions to specify both the interval between the runs and the time to execute the scheduled job. This is particularly useful for maintenance activities to perform overnight, outside the system peak usage times.

Next, learn about standard scheduled jobs.

[1]Cron is a time-based job scheduler in Unix-like computer operating systems. People who set up and maintain software environments use cron to schedule jobs (commands or shell scripts) to run periodically at fixed times, dates, or intervals.

Create a Scheduled Job

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager comes with several default scheduled job types , but you can also create a new scheduled job to suit your needs. Each scheduled job requires a service that will be executed to perform a job, therefore you may need to create a scheduled service first if it doesn't exist.

To create a new scheduled service:

1. Select **Services > All Services**. and click **New**.
2. Select **Scheduled Service** from the **Service Type** dropdown list.
3. Select a service template from the **Service Template** dropdown list, which can be:
   - **Fluent**[1] Scheduled Service - this is the preferred service template.
   - **Groovy**[2] Scheduled Service
   - Scheduled Data Extract Service
4. Update the service name in the **New Name** field.
5. Update the service version in the **Version Number** field.
6. Select an optional organization from the Organization dropdown list.

> If organization is blank, the service will be global, otherwise it will be available to the selected organization only.

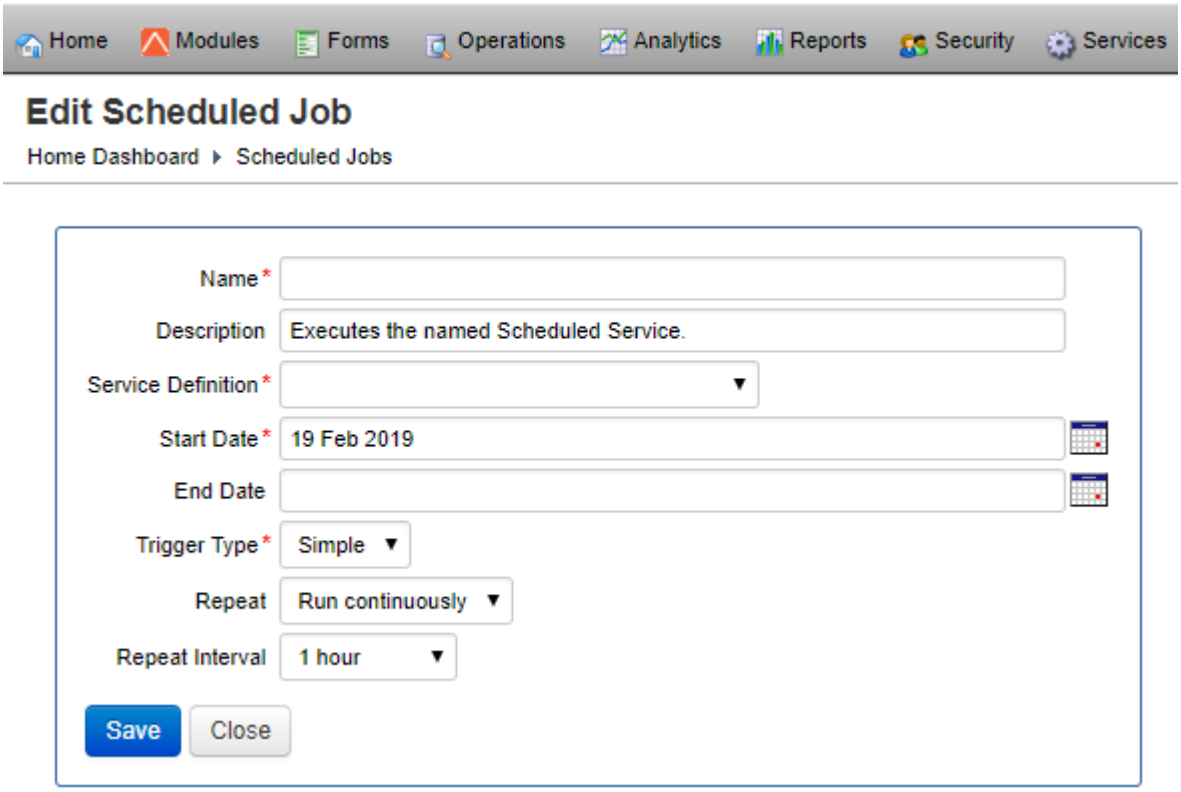
7. Click **Save** to update the changes.

To create a new scheduled job:

1. Select **System > Scheduled Jobs**
2. Click **New Scheduled Service Job** to create a new scheduled job.

---

[1]Fluent is a programming style that can be applied to a wide range of programming languages. Across the Journey platform, fluent is commonly associated with the Fluent SDK. The Fluent SDK is intended for use in developer integrated developer environment's (IDE) and contains libraries and project files for all the different services it supports.
[2]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

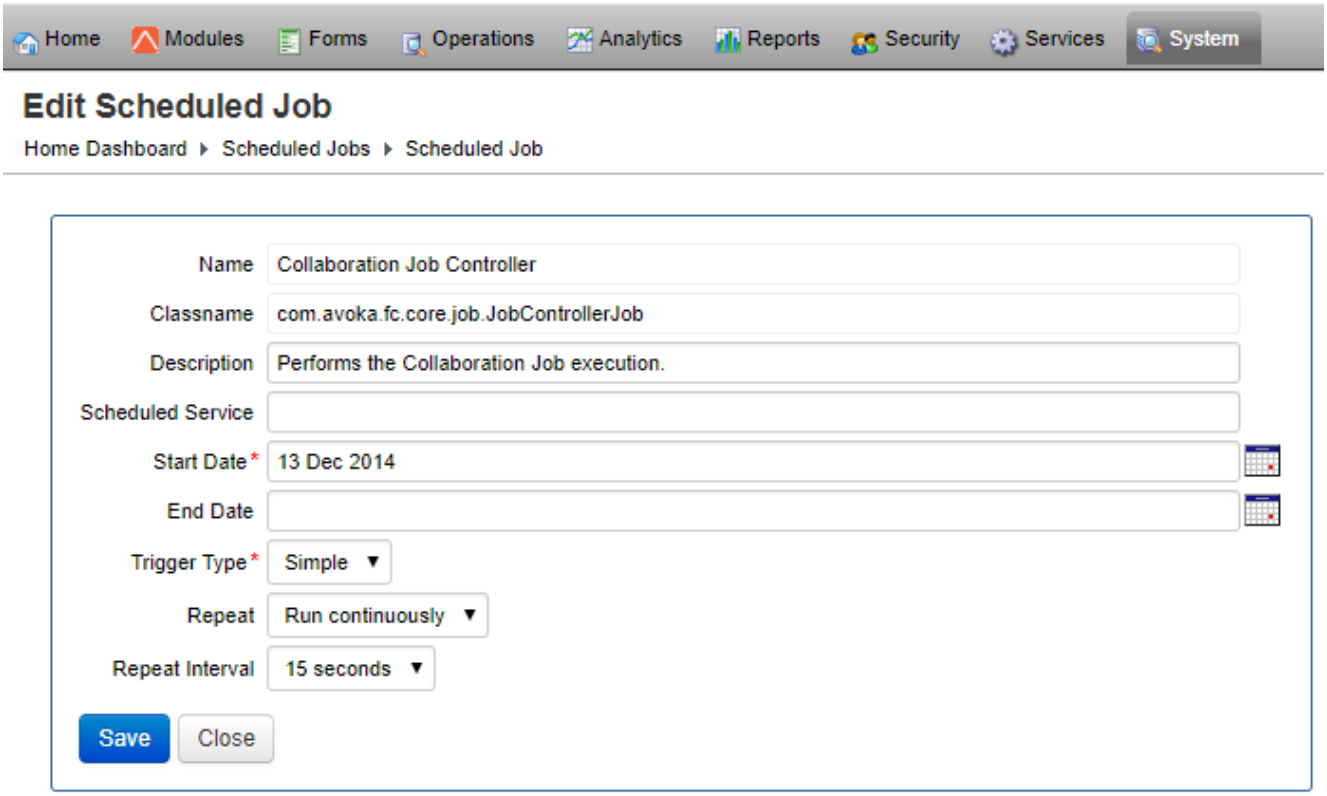


3. Enter the unique name of the scheduled job into the **Name** field.
4. Provide a description of the new scheduled job.
5. Select a service from the **Service Definition** dropdown list, which must be of the Scheduled Service type. If this service doesn't exist, create a new service.
6. Specify the Start Date and End Date.
7. Select a trigger type from the **Trigger Type** dropdown list, which can be **Simple** or **Cron**.
8. Select the repetition period from the **Repeat** dropdown list and repeat interval from the **Repeat Interval** for the simple trigger type.
9. Provide a cron job regular expression in the **Cron Expression** field, which the scheduler will use to know when and how often to invoke this job. For example, `0 0 1 * * ?` will cause the job to run every day at 1 am. However, much more sophisticated schedules are possible.
10. Click **Save** to update the changes..

After scheduled jobs are created, there are only a few configuration changes you can do to maintain them. Most frequently, you will need to update the repeat interval or cron expression of a scheduled job.

To configure a scheduled job:

1. Select **System > Scheduled Jobs**
2. Locate a scheduled job and click **Edit**.



3. Update the Start Date or End Date.
4. Change a trigger type from the **Trigger Type** dropdown list, which can be **Simple** or **Cron**.
5. Select the repetition period from the **Repeat** dropdown list and repeat interval from the **Repeat Interval** for the simple trigger type.
6. Update a cron job regular expression in the **Cron Expression** field.
7. Click **Save** to update the changes..

Next, learn how to view scheduled job history.

Standard Scheduled Jobs

Journey Manager (JM) | System Manager / DevOps | 22.04

Manager comes with a number of preconfigured scheduled jobs that perform a range of important functions and shouldn't be paused or removed without good reasons.

Manager runs the following standard scheduled jobs:

Scheduled Job	Description
Collaboration Job Controller Job	Processes collaboration jobs by calling the job controller service associated with the collaboration jobs.
Data Keystore Rollover Job	Creates new keystores for organization, if automatic rollover is configured.
Data Retention Job	Calls the default Data Retention Management service, which is responsible for purging Manager data such as submissions and error logs. No other scheduled jobs will be run while the data retention job is executing to avoid data contention.
Delivery Escalation Job	Sends escalation emails for submissions delivered via secure email delivery.
Delivery Reminder Job	Sends reminder emails for submissions delivered via secure email delivery.
Email Queue Job	Sends emails placed into the email queue.
HTTP Invoker Job	Invokes a HTTP URL, which is configured as the context parameter of the job.
Payment Reminder Job	Loads the default payment reminder service and calls it. The service sends out a reminder email regarding incomplete payments For more information, see predefined groups to which emails are sent by default.
Run Reports Job	Loops through all report schedules setup in Manager. If it finds one that is due to be sent out, it will run the report, extract the result data and send out the report email.

Scheduled Service Job	Invokes a Scheduled Service Job. These services can be customized using a **Groovy**[1] script and are used to run custom logic on a schedule. The service the job will invoke can be selected when editing the job.
Security Policy Manager Job	Applies security policies such as user role and group expiry. It also removes the accounts of users who never verified their email address.
Status Management Service	Changes form space's, form's, and form version's statuses that are scheduled using Notifications and Status Change. | 22.04
Submission Email Verification Reminder Job	Calls the default submission email verification service, which sends out reminders for submissions where the user must verify their email address.
System Monitoring Job	Monitors other important Manager system activities, such as the transaction processing and email queue. It also sends emails to the Receive System Alerts group if these scheduled jobs don't exist or have not been running as they should.
Template Version Deployment Job	Applies scheduled form version changes.
Transaction Processing Job	Calls the default transaction processor service, which performs submission processing in the following order: abandonment, receipts, delivery, task expiry and transaction history creation.
Virus Scan Job	Loads a list of all attachments that need to be virus scanned and calls the default virus scan service for each attachment, setting the attachment's virus scan status appropriately.

Next, learn how to manage scheduled jobs.

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

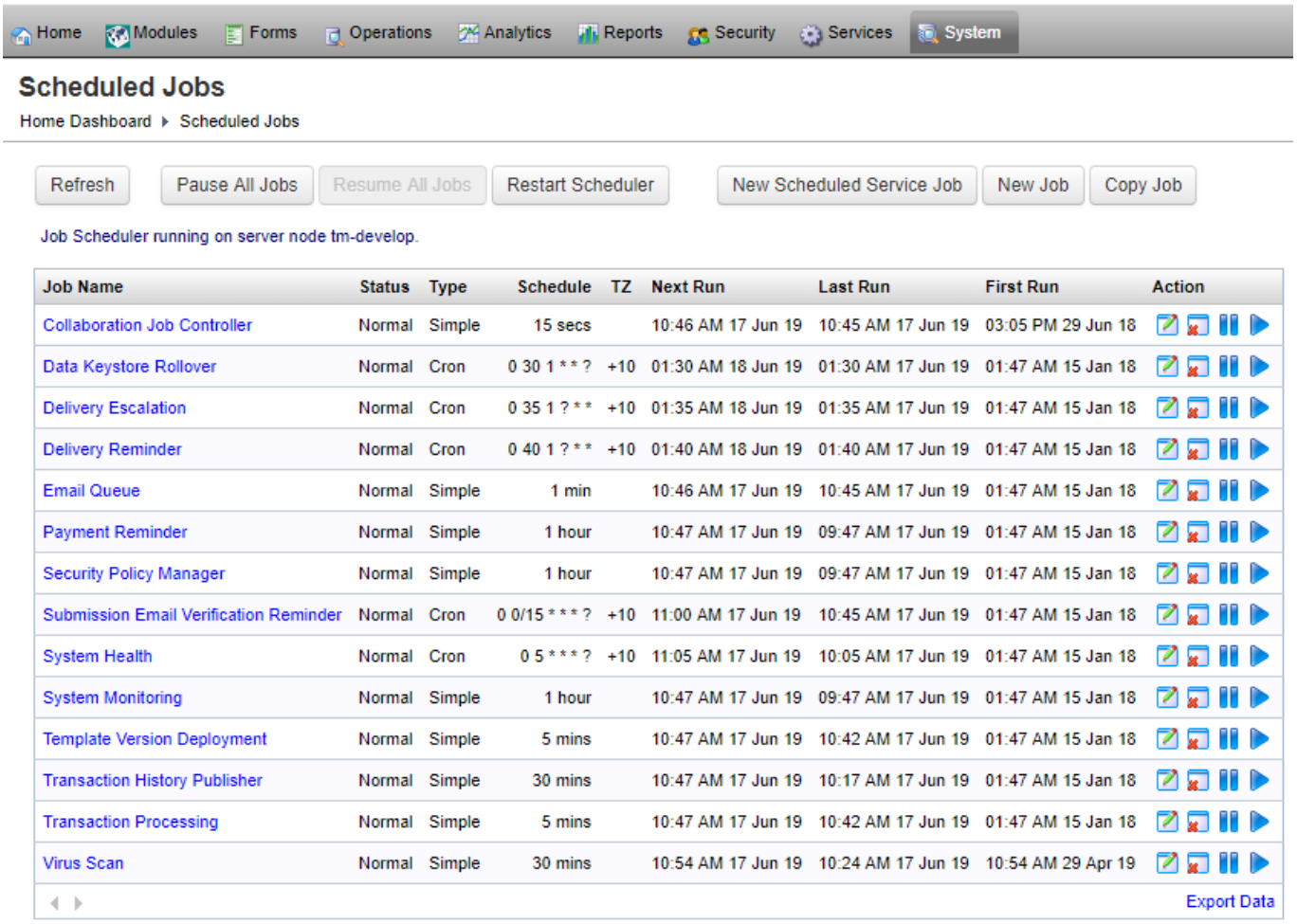
Manage Scheduled Jobs

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view and manage scheduled jobs.

To view scheduled jobs, select **System > Scheduled Jobs**.

Scheduled jobs are shown page by page, with the maximum number of records per page configured in the users preferences.

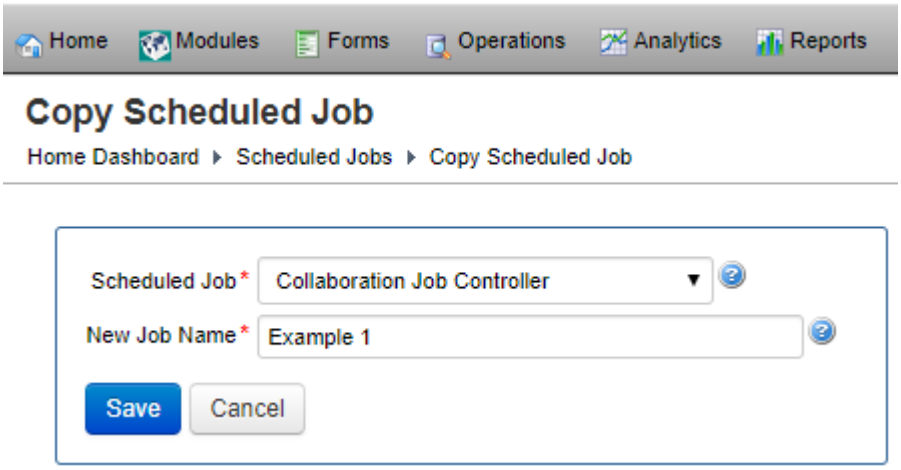


- **Job Name** is a name of a A scheduled job.
- **Status** is a status of the scheduled job that can be **Normal** or **Paused**.
- **Type** is a type of the scheduled job that can be **Simple** or **Cron**.
- **Schedule** is time or interval, depending on a type, when the scheduled job is supposed to run.
- **TZ** is the time zone. Only cron scheduled jobs are associated with a timezone, which generally corresponds to the timezone used n your Manager server. Whenever the server is restarted, it adjusts the timezones for all cron scheduled jobs, if the system time has changed. If you are seeing unexpected timezone values, check the system time settings of the server that Quartz is running on and restart Manager.
- **Next Run** is the time that this scheduled job will run next.
- **Last Run** is the time that this scheduled job was run last .
- **First Run** is the time that this scheduled job was run first .
- **Action** specifies one of the following operations: view and edit, delete, pause, and trigger a scheduled job.
- A server node name is displayed above the main table. In this example, it is the **tm-develop** server node.

If Manager is configured to use multiple server nodes, the job scheduler runs only on a single node. You must log on to the management console on that node to make changes to the scheduled jobs.

You can also perform the following actions:

- Click **Refresh** to refresh the list of scheduled jobs.
- Click **Pause All Jobs** to temporarily suspend all scheduled jobs until you are ready to start them again.
- Click **Resume All Jobs** to resume all scheduled jobs.
- Click **Restart Scheduler** to restart the scheduler. If there are issues when running scheduled jobs, one option is to pause or trigger individual scheduled jobs for debugging purposes. It is also possible to restart the entire job scheduler to reset it without restarting Manager.
- Click **New Scheduled Service Job** or **New Job** to create a new scheduled job.
- Click **Copy Job** to create a new scheduled job as a copy of the existing scheduled job.



- Click **Edit Job** to view or edit a scheduled job. An administrator would typically use this option to change the time interval or cron expression of a scheduled job.
- Click **Delete Job** to remove a scheduled job that is no longer needed.

You can also disable a scheduled job by setting an end date after which will no longer be run by the scheduler.

- Click **Pause Job** to temporarily suspend a scheduled job until you are ready to start it again. You may need to pause it to make sure no transactions or emails queue being processed. The job status changes to **Paused**. Manager won't run the scheduled job until you resume it. Click **Resume Job** to start the job again.
- Click **Trigger Job** to run a job immediately. This is handy when the time for a scheduled job's next invocation has not come yet but you want it to run anyway. The job status changes to **Running**.
- Click **Interrupt Running Job** to stop the job. You may want to terminate a running scheduled job because of bugs in the job or data that the job cannot handle. The job status changes to **Normal**.

Next, learn how to view scheduled job history.

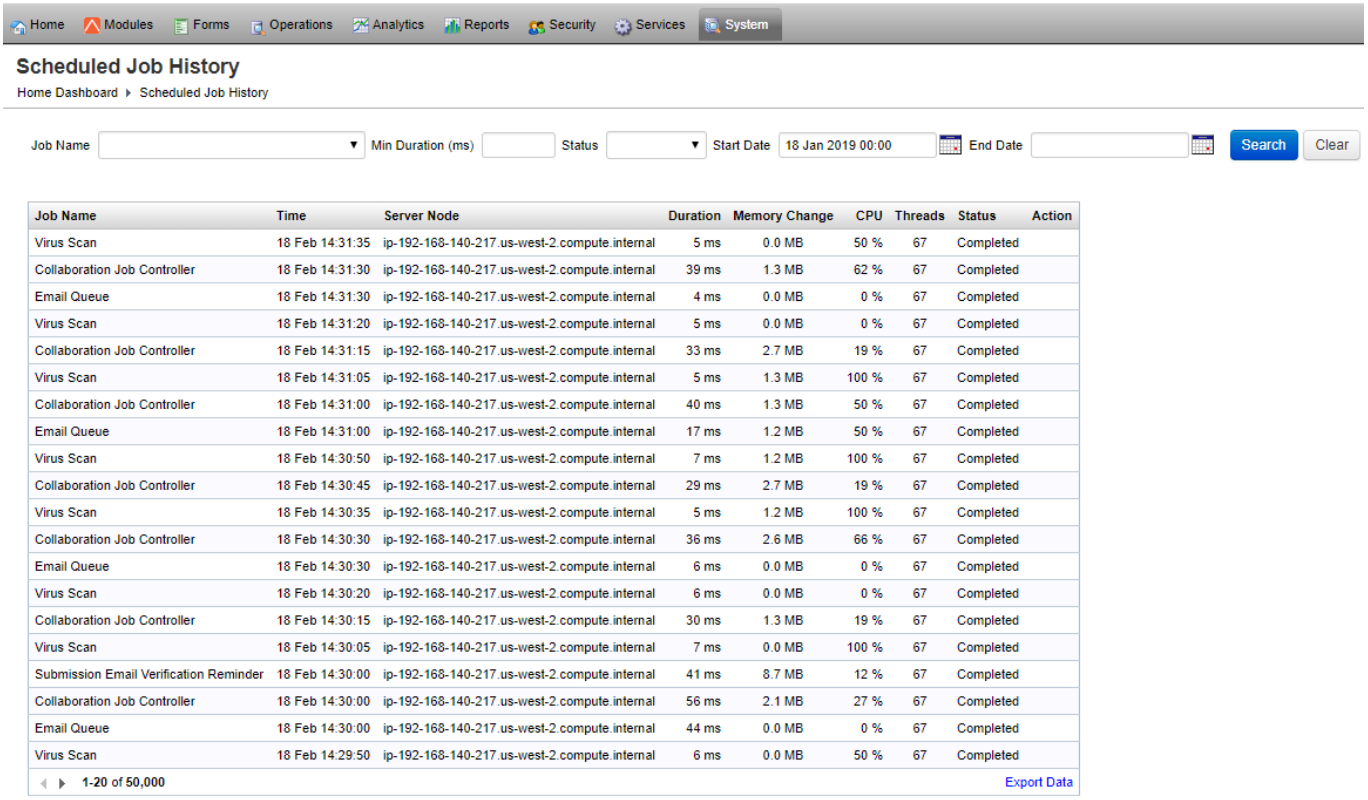
View Scheduled Job History

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Every time a scheduled job runs, Manager logs a history event recording which job ran, the status of the job and some performance metrics. Manager allows you to view and search scheduled jobs history.

Because Manager runs many scheduled jobs as part of normal operations, the scheduled job history can contain a large number of entries. Manager purges the scheduled job history as part of its data retention policies.

To view scheduled job history, select **System > Scheduled Job History**.



The following details are available:

- Job name
- Time the job was run.
- A server node of a Manager server where the job was run.
- Duration of the job.
- Memory Change, which is the amount of the server's memory used to run the job.
- CPU of the server.
- Number of threads.
- The status of this job.

  A scheduled job status can be one of the following:

- Completed- the job has completed.
- Error - the job has failed.

To filter or search the scheduled jobs by one or more criteria, specify the following settings and click **Search**:

- Job name or part of the name.
- Min Duration (ms) it took the server to run a job.
- Status of the job.

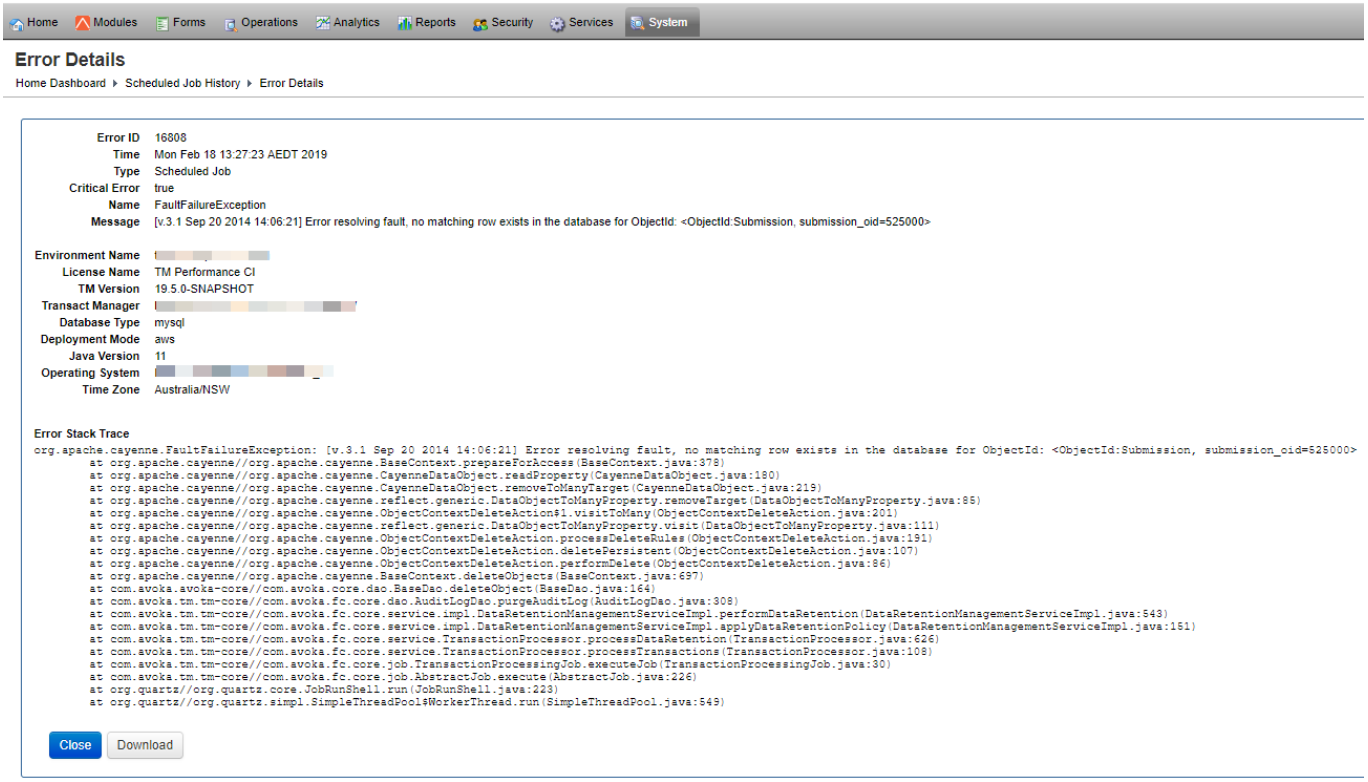
  A scheduled job status can be one of the following:

- Completed- the job has completed.
- Error - the job has failed.
- Start Date and End Date of the job.

Click **Clear** to reset the search criteria.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Click **Error Log** to view the detailed error information, if a job has the error status.



Next, learn how to view scheduled job.

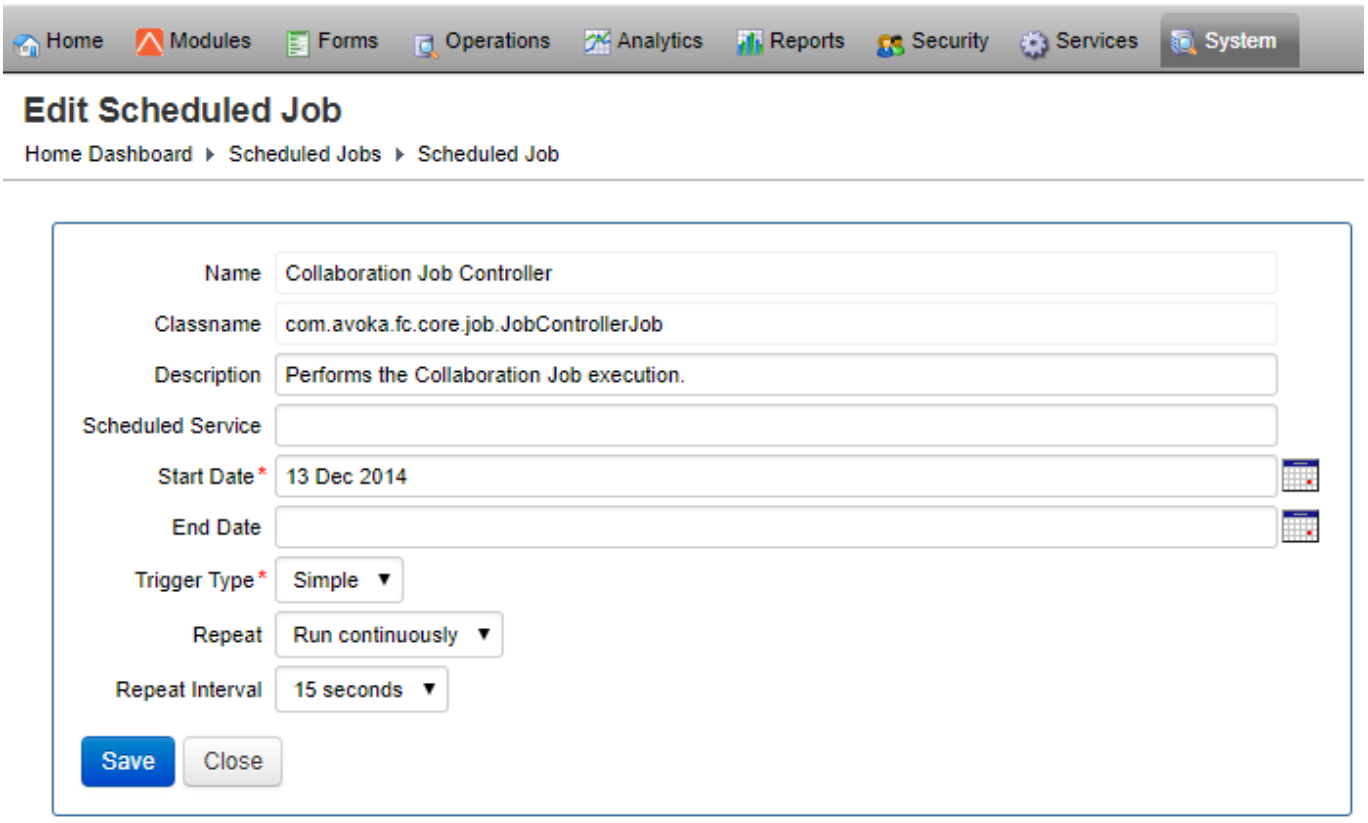
Collaboration Job Controller

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

The collaboration job controller is the scheduled job service, which runs as a single **thread**[1] to processes collaboration jobs. First, it selects a list of collaboration jobs that are available for processing.Then, it goes through each collaboration job in the list and calls an associated job controller service. The job controller service definition is used to process collaboration job's actions specified in each step and go (route) to the next step, until an endpoint is reached. Each collaboration job runs until a wait action is encountered, or an error is thrown during the collaboration job action or the collaboration job successfully completes. The scheduled job terminates when there are no more collaboration jobs in the list to process.

The scheduler only starts one collaboration job thread at a time. A large backlog of work may take more than a minute for the thread to complete. A new collaboration job thread won't be started until the previous one has completed.

The default collaboration job controller configuration is shown below.



[1]A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.

The default interval is 1 minute, which is the value we recommend for production environments. However, it is useful to set this interval to 15 seconds in on a development environment to improve development and testing turnaround.

Next, learn how to manage scheduled jobs

Receipts Overview

Journey Manager (JM) | System Manager / DevOps | 19.11

Journey Manager provides form users with receipts that are read-only statements acknowledging the data they have submitted. Receipts are an essential part of the form submission process. Receipts are viewable PDF forms which are generated, based on form content, for all completed transactions. Each receipt must contain unique receipt numbers, which you can use to identify these transactions. Maestro allows you to create custom receipts for Maestro forms to optimize them for printing or to implement specific customer requirements. In this case, Manager uses the receipt design to issue the PDF receipt.

The Maestro editor allows you to build a form with a separate receipt design when you select the the **Use Separate Receipt Design** checkbox.

Receipts are primarily delivered by email. There are two options available to send receipts to users via email:

- Email a PDF receipt directly to the user as an attachment
- Email a secure link that allows the user to directly download the PDF receipt

Manager still supports downloading PDF receipts from form confirmation pages. However, this method is not recommended due to performance scaling considerations.

Manager allows you to use the following orchestrations for receipt generation:

- **PhantomJS**[1]
- Static PDF
- iText
- **Puppeteer**[2] - | 19.05

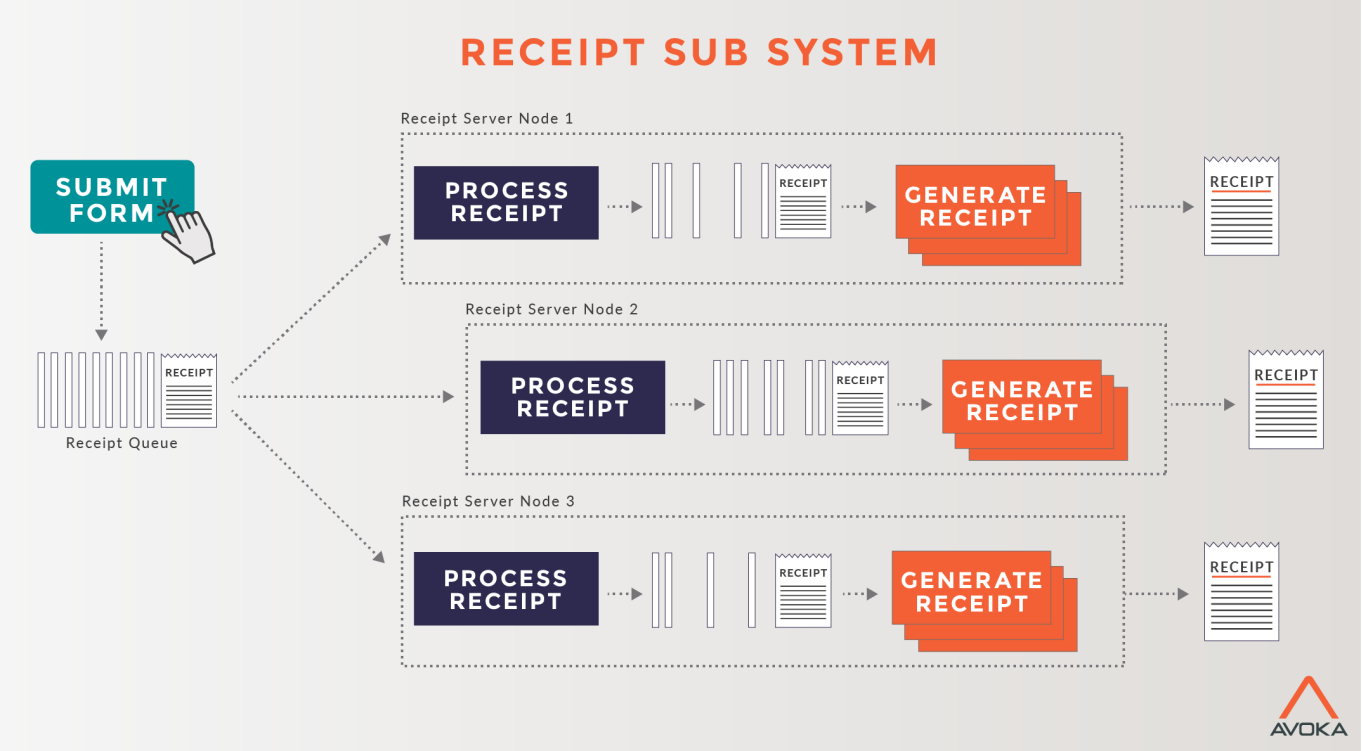
Manager generates user receipts in the background by using dedicated PDF receipt server nodes. This approach offers the following improvements:

---

[1]PhantomJS is a scripted, headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment. PhantomJS is based on WebKit making it a similar browsing environment to Safari and Google Chrome (before Chrome's fork of WebKit evolved into Blink). It is open-source software released under the BSD License.
[2]Puppeteer is a Node library which provides a high-level API to control headless Chrome or Chromium over the DevTools Protocol. It can also be configured to use full (non-headless) Chrome or Chromium.

- PDF receipt generation can be performed on dedicated PDF receipt generation servers that can be scaled independently.
- PDF receipts should only be generated by a background thread to improve server efficiently. This is the default option.

The receipt generation process consists of the sub-systems as shown below.



Each of the sub-system uses the following core global services:

- The Generate Receipt service renders the PDF receipt.
- The Dynamic PDF Receipt service generates a receipt as it appears on a form. This is the legacy receipt render service.
- The Dynamic PDF Receipt 2 service provides more receipt customization options. | 17.10
- The Process Receipt service submits the receipt in the background. This is achieved by using a load balancing mechanism that queues the receipt to a chosen node (in the above diagram, it's a Node 1, Node 2 or Node 3), which can then be processed by one or more threads, depending on the Manager configuration.

Next, learn how to configure receipts server node.

Configure Receipt Numbers and Tracking Codes

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager lets you identify form transactions by using a receipt number or a tracking code. A receipt number is used to identify a submitted transaction in Manager. A tracking code is a unique alphanumerical code that is generated when a transaction is created and is used to track this transaction that is yet to be submitted. Tracking codes are generally shorter than receipt numbers and by their nature are identified before a receipt number is generated. Manager allows you to use the same value for both the receipt number and tracking code of a submission.

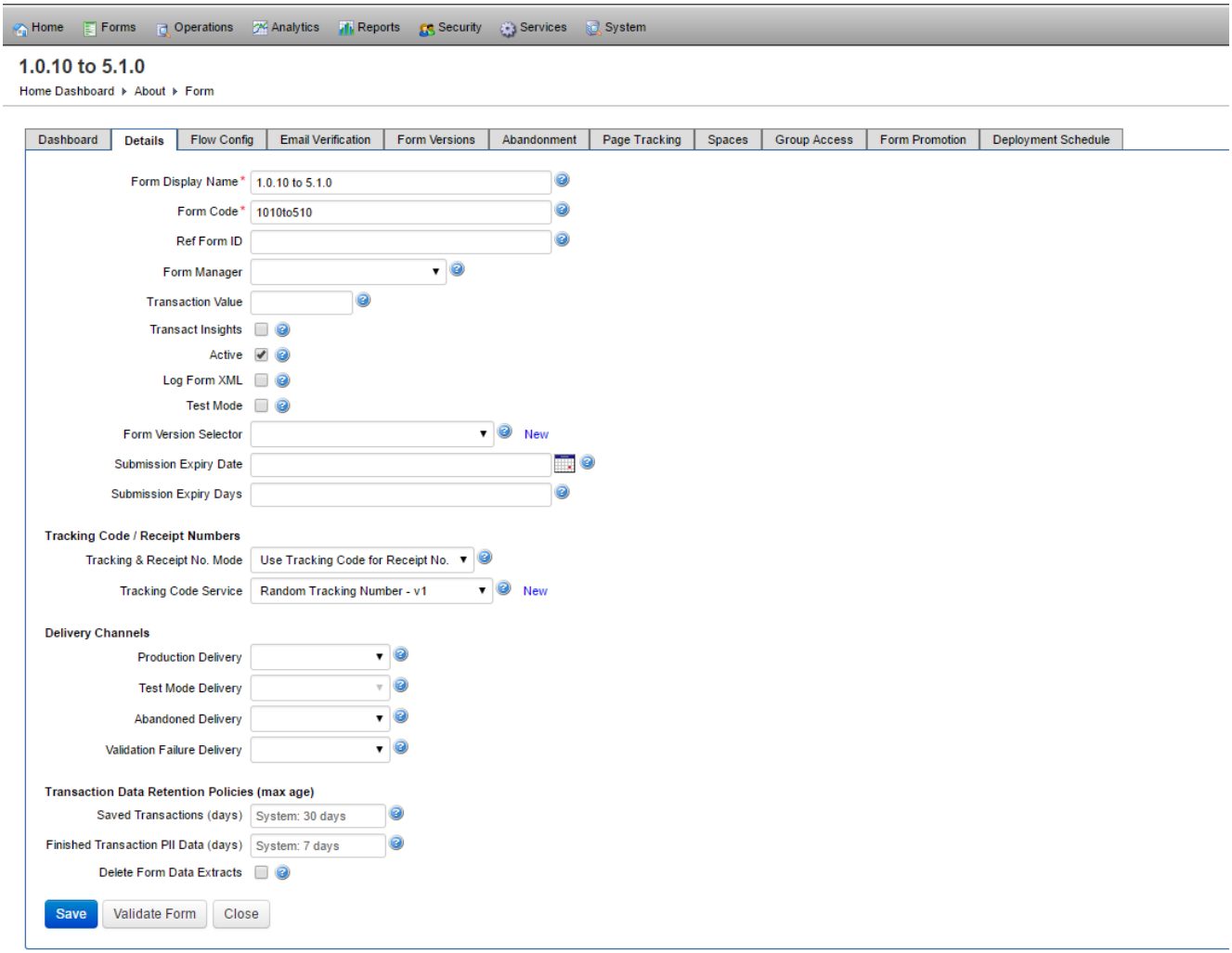
Tracking codes are generated using the **`TrackingCodeBuilder`** class and are 29 characters long. They are stored in a database table, called **`tracking_number`**, to ensure every new tracking code is unique. The table is never purged to make sure the codes are unique for as long as the Manager server runs. In the unlikely event of the table growing too large so it needs to be purged, the uniqueness of tracking codes can be compromised. In reality, it won't happen because transactions are not that long-lived and once a transaction is purged, that tracking code is technically available and can be reused.

> Check Top Table Sizes to see a number of records that the **`tracking_number`** table has. In some installations, this table is not listed.

Tracking codes and receipt numbers are only unique to the environment and hence may be duplicated across systems, such as Prod, Dev, UAT etc. If you need a really unique key, you should use a submission key (**`submitKey`**) which is unique globally. Alternatively, you can prefix or suffix a tracking code with some other piece of unique information, such as day, month and year time stamp.

To configure a form's tracking code or receipt number, follow these steps:

1. Select **Forms > Forms**.
2. Locate a form and click the **Details** tab to view form details.



3. Specify whether to have separate Tracking Code and Receipt Numbers generated for your transactions, or use the same value for both. Select one of the following options from the **Tracking & Receipt No Mode** dropdown list:
   - **Use Tracking Code for Receipt No** – the receipt number will be identical to the tracking code. It's recommended to use the Tracking Code to improve performance and scalability.
   - **Use Receipt No for Tracking Code** – the tracking code will be the same as the receipt number. This option overrides the previously generated tracking code.
4. Select a service to generate form tracking codes from the **Tracking Code Service** dropdown list. You have the following options:
   - Blank - the default Tracking Code service is used
   - Fluent Tracking Number
   - Random Tracking Number
5. Click **Save** to update the changes.

Next, learn how to configure receipt headers and footers.

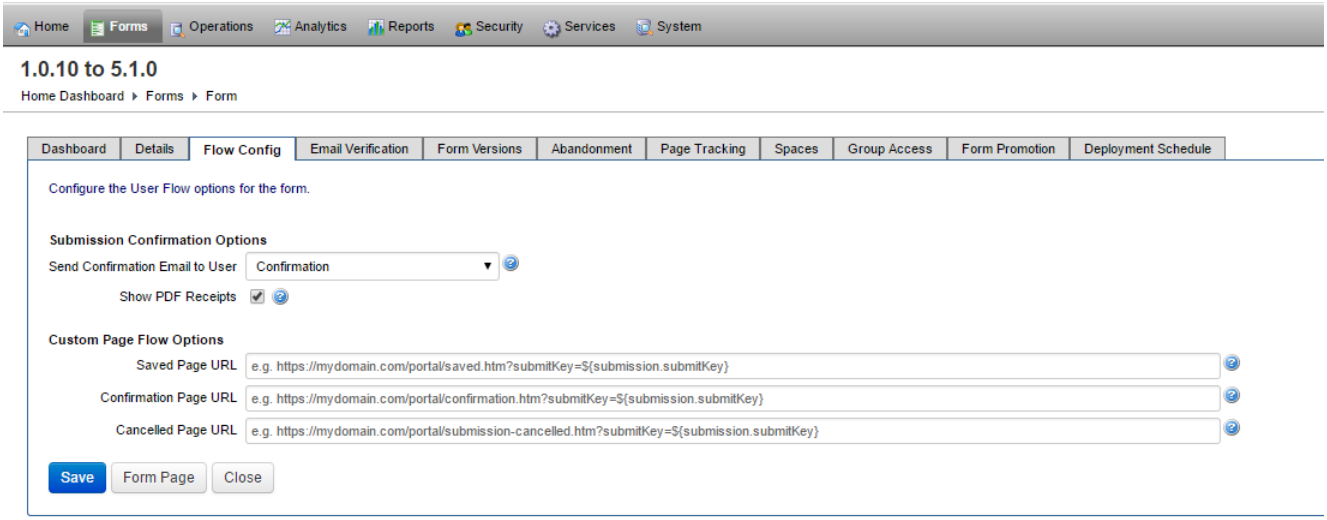
Configure Receipts Delivery Mode

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager automatically sends users a confirmation email with a receipt when they complete submissions.

To customize the receipt delivery:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select the **Flow Config** tab.



4. Select one of the following options from the **Send Confirmation Email to User** dropdown:
   - **None** – doesn't send a confirmation email to the user. Use this option to turn off the auto-confirmation email.
   - **Confirmation** – sends a confirmation email with no attachment or link to the PDF receipt.
   - **Confirmation and Receipt Attachment** – sends a confirmation email including the receipt PDF file.
   - **Confirmation and Receipt Link** – sends a confirmation email containing a secure link to the receipt PDF file.
5. Select the **Show PDF Receipts** checkbox to offer users of the form the opportunity to see the PDF receipt on the Confirmation Page that can be set up to appear as soon as the user submits the form. If no Confirmation Page is set up, then this setting is ignored. If Skip Confirmation Page is selected, this setting is ignored as well.
6. Click **Save** to update the changes.

Next, learn how to configure form flow.

Configure a Receipt Server Node

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager can utilize several server nodes to generate receipts. To optimize receipt generation, create at least one server node that is solely dedicated for receipt generation by selecting the **Transact PDF Receipt Server** check-box only.

Manager has a **Transact Receipts** module, which can be deployed on a dedicated receipt server node.

You can tune a receipt server node performance by:

- Creating more dedicated receipt server nodes that will process the receipt queue quicker.
- Creating more threads on each server node to increase concurrent processing. This is configured in the Receipt Core Global Service.
- The maximum queue length of submissions to process for each node.
- The maximum number of attempts for rendering any PDF receipt before abandoning.

For low throughput systems, receipt rendering can be performed on server nodes that are shared as opposed to dedicated. However, on high throughput systems with complex receipts, dedicating server nodes for receipt generation may be a better option.

With this approach, each receipt server node can be configured to have multiple threads running. The queue for these threads is updated every minute. This provides the ability to spread the receipt rendering over several nodes and dedicating some server nodes to just rendering receipts, effectively building a server farm for receipt rendering. The use of a farm for receipt rendering means these nodes are not affected by user web traffic and other system processes. As a result, they can be tuned to maintain an optimum level of throughput.

Next, learn how to configure receipt delivery mode.

Receipt Headers and Footers Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

When Manager creates a PDF receipt, it uses **phantomjs** to render each header and footer in a separate **HTML**[1] DOM prior to generating the receipt. This has several consequences:

- The pagination process requires a fixed header and footer height in order to correctly size the pages. This means that every page will reserve a fixed amount of space for the header and footer even if there is no content present. You can set this fixed header and footer height by setting the **paperSize.header.height** or **paperSize.footer.height** properties in the **JSON**[2] options file. Content that exceeds this height will be clipped and not show on the receipt.
- Because each header and footer is rendered in a separate HTML DOM, **CSS**[3] classes defined in the form itself will not ordinarily be available to use, but instead, styling must be applied using in-line styles. Render Receipt services, such as Dynamic PDF Receipt 2 Service, allow you to apply these styles with ease.
- You can use the **config.customCSS** property to add additional CSS style definitions to the form.
- If the **config.supportHeaderFooterCSSClasses** is set to true then any CSS classes used in a header or footer template are converted automatically to inline styles. This also allows the use of CSS classes already defined in the form itself, not just those defined in the **config.customCSS** property.

Header and Footer Templates

The **headers** and **footers** properties, stored in a JSON file, allow you to specify template rules that a receipt service runs to determine the content of a header and a footer. Click **Receipt Properties Json** to view a JSON file of your render receipt service, for example Dynamic PDF Receipt 2 Service. Each rule has a **pageFilter** and **template** property.

The **template** property contains the HTML that defines the header and footer content. The tokens **${page}**and **${numPages}** will be replaced with the current page number and total number of pages respectively. Page numbering will start from 1. The content must be a single HTML element so if you need multiple elements, be sure to wrap them in an appropriate ** ** element.

---

[1]Maestro forms are fundamentally built using HTML 5. The structure of the form and the text displayed in the form are all configured and established using HTML5. HTML is the standard markup language for creating Web pages and stands for Hypertext Markup Language. HTML 5 describes the structure of a Maestro form.
[2]JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.
[3]CSS stands for Cascading Styles Sheets and is used to define styles for a Maestro Form, including the design, layout, and variations in the form's display on different devices and screen sizes. The CSS on a Maestro form is primarily taken care of by the Template Designer. The Template Designer will implement the branding and/or styling of the form's assets.CSS can be applied to the template of the form, individual components, and any other asset associated with a Maestro form.

The **pageFilter** property is a string that contains either the page number or a * to match all pages not otherwise filtered.

You can enable logging of the templates used by setting the **config.logHeadersFooters** property to true.

Custom Headers and Footers In a Form

Forms can optionally use **JavaScript**[1] to control the content and styling of headers and footers. You may want to do this for example to inject calculated or dynamic content in the header or footer. To allow a form to control the header and footer behavior first you will need to make sure the **config.disableFormOverrides** property of the JSON options file is set to false.

[1] Javascript (JS) is a scripting languages, primarily used on the Web. It is used to enhance HTML pages and is commonly found embedded in HTML code. JavaScript is an interpreted language. Thus, it doesn't need to be compiled.

You will then need to inject a function into the window object of your form using JavaScript inside your form as in the example below.

```
window.phantomReceiptCallback = function(actionName,data) {
if (actionName=="orientation") {
       // return either "landscape" or "portrait" or fall
       // this action is only invoked once per form
}
if (actionName=="format") {
       // return the paper format eg "A4","letter",etc or fall
       // this action is only invoked once per form
}
if (actionName=="margin") {
       // return a JSON object defining the margin eg {"top": "0.5cm","left": "0.5cm","bottom":
"0.5cm","right":"0.5cm"} or fall
       // this action is only invoked once per form
}
if (actionName=="headerHeight") {
       // return the header height  eg "1cm" or fall
       // this action is only invoked once per form
}
if (actionName=="footerHeight") {
       // return the footer height  eg "1cm" or fall
       // this action is only invoked once per form
}
if (actionName=="header") {
       // return the header template string eg "&lt;p&gt;This is the header&lt;/p&gt;" or fall
       // you can specify the tokens ${page} and ${numPages} in the template you are returning
and they will be replaced appropriately
       // you can access the current page and number of pages programmatically by using
data.pageNum and data.numPages respectively
       // this action is invoked for every page on the form
}
if (actionName=="footer") {
       // return the footer template string eg "&lt;p&gt;Page ${page} of ${numPages}&lt;/p&gt;"
or fall
       // you can specify the tokens ${page} and ${numPages} in the template you are returning
and they will be replaced appropriately
       // you can access the current page and number of pages programmatically by using
data.pageNum and data.numPages respectively
       // this action is invoked for every page on the form
}
// NOTE: always return the value defined in Manager if not explicitly overriden above
return data.defaultValue; // this is the value from the JSON options file
};
```

Here is an example of using a form function to inject the tracking code into the form footer on page 1 for a Composer form. You can for example add this JavaScript to a composer form by dropping a **Business Rule [General Purpose]** with the below script content.

```
window.phantomReceiptCallback = function(actionName,data) {
	if (actionName=="footer") {
		if (data.pageNum==1) {
			var trackingCode = sfc.getPhaseValue("TrackingCode");
			return "&lt;p class='phantom-footer'&gt;Page ${page} of ${numPages} - "+track
ingCode+"&lt;/p&gt;";
		}
		return "&lt;p class='phantom-footer'&gt;Page ${page} of ${numPages}&lt;/p&gt;";
	}
	// NOTE: always return the value defined in Manager if not explicitly overriden above
	return data.defaultValue; // this is the value from the JSON options file
};
```

Next, learn how to configure receipt headers and footers.

Receipt Headers and Footers Customization

Journey Manager (JM) | System Manager / DevOps | | 17.10

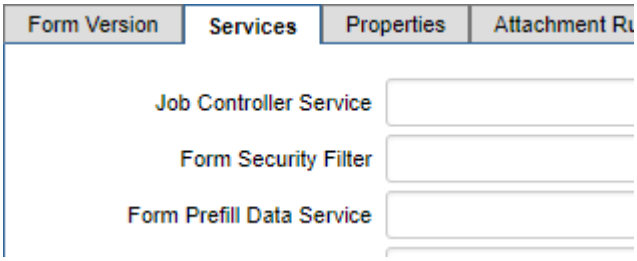
Manager allows you to add and customize the headers and footers of a form's PDF receipt. These headers and footers can include static content, such as HTML, and data from the form, such as tracking codes. Examples are provided throughout this topic.

Dynamic PDF Receipt 2

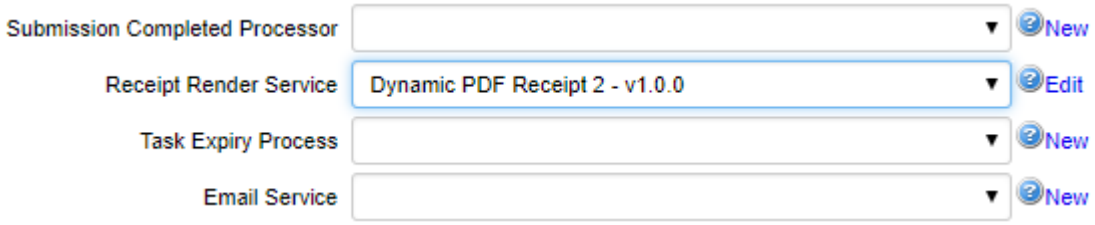
To customize the headers and footers for a receipt, the Dynamic PDF Receipt 2 service needs to be enabled for the form version you're working with. This is not to be confused with the legacy Dynamic PDF Receipt service which has fewer customization options.

To enable this service:

1. Open a form version in Manager.
2. Navigate to the **Services** tab.



3. From the **Receipt Render Service** dropdown, select **Dynamic PDF Receipt 2**.



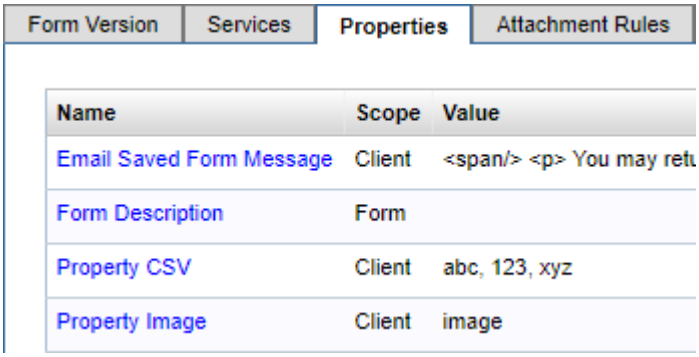
4. Click **Save** to update the changes.

Based on this change, PDF receipts will be now be generated using this service. You won't, however, notice any difference from the default receipt generation until the receipt's properties are modified.

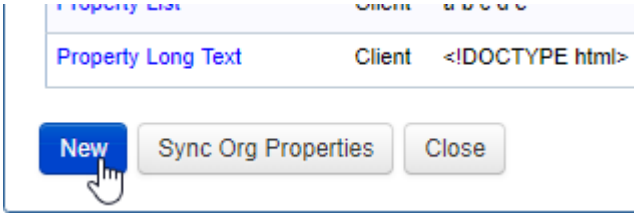
Receipt Properties

When the Dynamic PDF Receipt 2 service is enabled, some default properties are used to define the default functionality of the service. To override these properties:

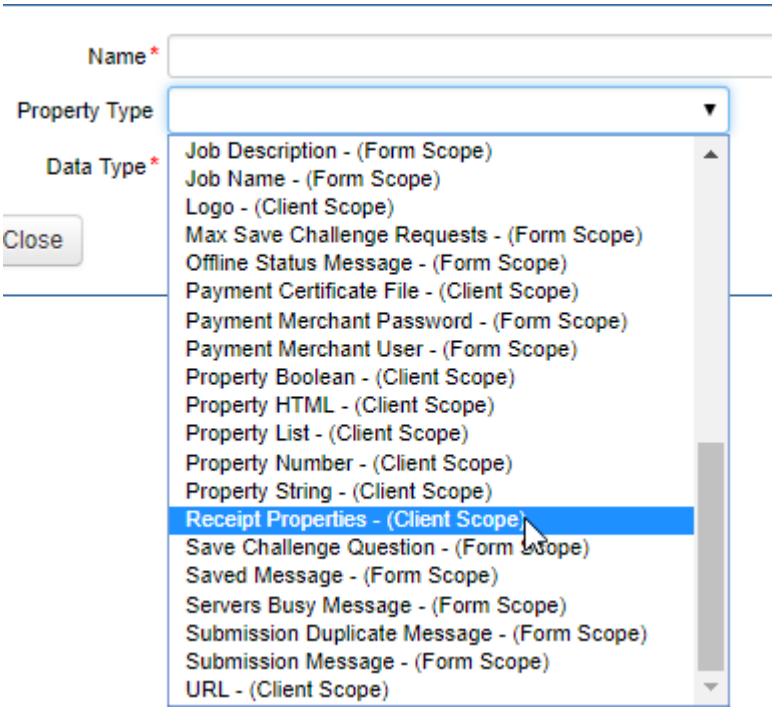
1. Navigate to a form version's **Properties** tab.



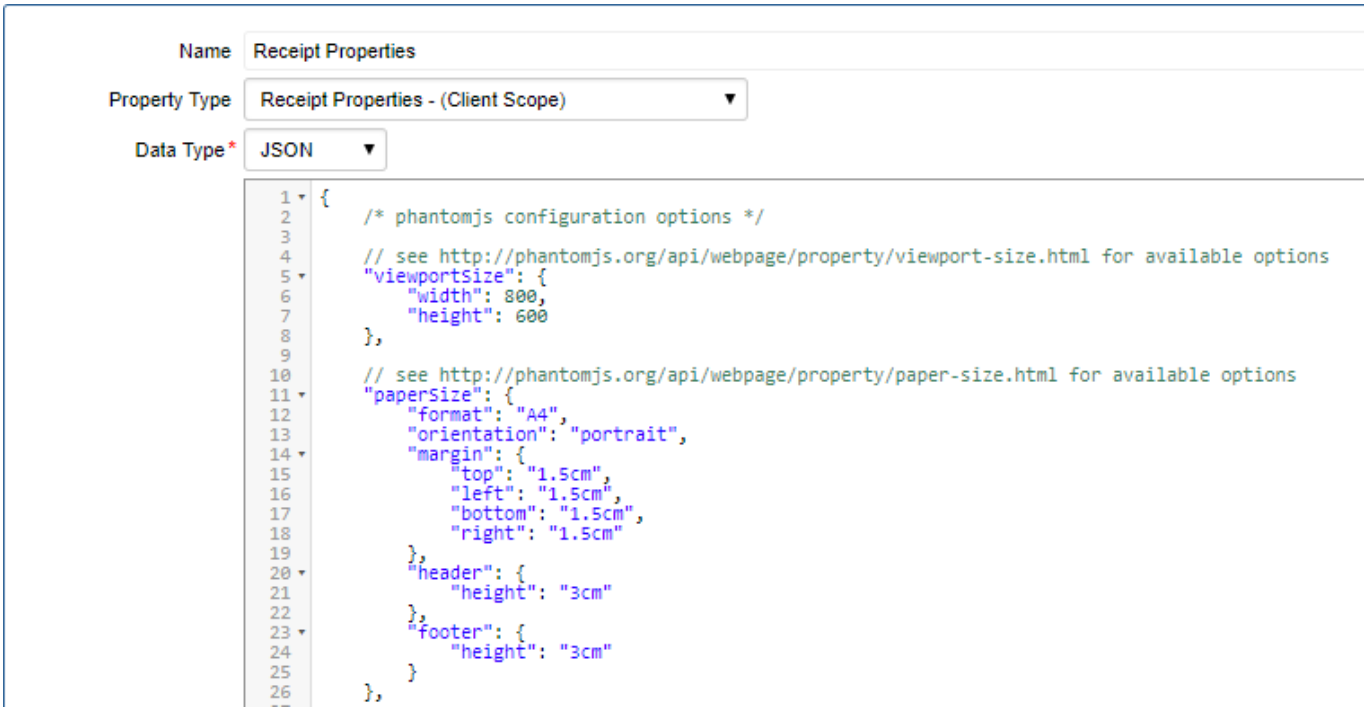
2. Click **New**.



3. From the **Property Type** dropdown, select **Receipt Properties - (Client Scope)**.



The fields will refresh with updated data and the **Value** field will contain JSON that we can modify to affect how the form's receipts are generated.



For example, to enable custom headers and footers for the receipt:

1. Update the **`paperSize`** object's **`height`** properties. By default, these property's values are set to "0cm", which means the header and footer won't be visible. Change this to a larger value, such as "3cm":

```
"header": {
       "height": "3cm"
},
"footer": {
       "height": "3cm"
}
```

2. Uncomment the **`headers`** and **`footers`** properties:

```
"headers": [
	{
		"pageFilter" : "*",
		"template": "<h1 class='phantom-header'>Default Header</h1>"
	},{
		"pageFilter" : "1",
		"template": "<h1 class='phantom-header'>First Page Header</h1>"
	}
],

"footers": [
	{
		"pageFilter" : "*",
		"template": "<p class='phantom-footer'>Page ${page} of ${numPages}</p>"
	}
],
```

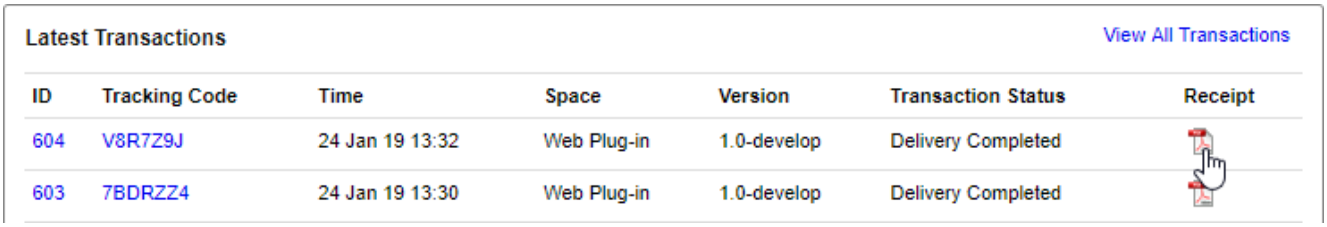
These properties are used to define the default content inside the headers and footers. The **pageFilter** properties can be used to include headers or footers on all pages, or on specific pages.

After making changes to these properties:

1. Click **Save**.
2. Return to the form's **Dashboard** tab.
3. Click the **Form Test Render** icon.



4. Submit some test data via the form.
5. Return to (and refresh) the **Dasboard** tab.
6. Click the **View PDF Receipt** icon for the latest transaction.



You should see that the generated receipt contains the customized header and footer that we defined via the Receipt Properties. Here, for instance, we can see the **h1** element that has been added to the first page of the receipt:



Customizing Receipts from the Form

All of the properties available via the Receipt Properties can be overwritten by the form that generates the receipt. You might, for instance, like to include the reference code of a form in the footer of a receipt. This requires the form to affect how the receipt is generated.

To enable this functionality:

1. In Manager, return to the form version's **Properties** tab.
2. Click **Receipt Properties**.
3. Within the **Value** field, change the **`disableFormOverrides`** property to **`false`**:

```
"disableFormOverrides": false
```

4. Click **Save** to update the changes.

Then, to use this feature, create a form load rule that contains the following function:

```
window.phantomReceiptCallback = function(actionName, data) {
	// code goes here
};
```

This function executes multiple times while a receipt is generated and provides us with two parameters:

1. **`actionName`**
2. **`data`**

Each time the function executes, the **actionName** parameter contains the name of one of the customizable properties, such as **format** or **margin**. Using a conditional, we can return a new value, thereby overriding the value of the property.

Here's a few examples:

1. **Change the receipt orientation to "landscape":**

```
if (actionName === "orientation") {
	return "landscape";
}
```

2. **Change the height of the header to "3cm":**

```
if (actionName === "headerHeight") {
	return "3cm";
}
```

3. **Add the form's tracking code to the footer:**

```
if (actionName === "footer") {
	// Get the tracking code from a Composer form
	// var trackingCode = sfc.getPhaseValue("TrackingCode");
	// Get the tracking code from a Maestro form
	var trackingCode = Form.getSystemData("TrackingCode");
	return "<p>Tracking Code: " + trackingCode + "</p>";
}
```

The **data** parameter is an object with the following structure:

```
{
	"pageNum": 1,
	"numPages": 2,
	"defaultValue": "portrait"
}
```

The **pageNum** property contains the current page number of the receipt. One way to use this property is to display different content on the first page of a receipt:

```
if (actionName === "footer") {
	if (data.pageNum === 1) {
		return "<p>This appears in the footer on the first page.</p>";
	} else {
		return "<p>This appears in the footer on every other page.</p>";
	}
}
```

The **numPages** property contains the total number of pages in the receipt. This is commonly used in the footer, in conjunction with the **pageNum** property:

```
if (actionName === "footer") {
	return "<p>Page " + data.pageNum + " of " + data.numPages + "</p>";
}
```

The **defaultValue** property contains the value of the property that was defined in Manager, via the Receipt Properties. If, for instance, you give the **orientation** property a value of "portrait" via the Receipt Properties, the **defaultValue** property will contain that same value.

This property should always be returned at the bottom of the function:

```
window.phantomReceiptCallback = function(actionName, data) {
	if (actionName === "orientation") {
		return "landscape";
	}
	return data.defaultValue;
};
```

This ensures that the function never returns **undefined**.

For all of the properties, the function expects a string to be returned, except for the margin property, which expects an object with string values:

```
if (actionName === "margin") {
	return {
		top: "1.5cm",
		left: "1.5cm",
		bottom: "1.5cm",
		right: "1.5cm"
	};
}
```

Here's a reference of the available properties and their default values:

Property	Default Value
format	a4
orientation	portrait
margin	`{ top: "1.5cm", left: "1.5cm", bottom: "1.5cm", right: "1.5cm" }`
headerHeight	0cm
footerHeight	0cm
header	<h1 class='phantom-header'>Default Header</h1>
footer	Page ${page} of ${numPages}

After making changes to this function: save the code, build the form, and update the form via Manager. Any submissions from this form will use this function when generating the receipt.

Next, learn more about receipt's headers and footers.

Using Header and Footer Templates

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

The **headers** and **footers** properties of the JSON options file allow you to specify an array of templating rules that are run to determine the content of the header or footer respectively. Each rule has a **pageFilter** and **template** property.

The **template** property contains the HTML that defines the header/footer content. The tokens ${page} and ${numPages} will be replaced with the current page number and total number of pages respectively. Page numbering will start from 1. The content must be a single HTML element so if you need multiple elements, be sure to wrap them in an appropriate element.

The **pageFilter** property is a string that contains either the page number or a * to match all pages not otherwise filtered.

You can enable logging of the templates used by setting the **config.logHeadersFooters** property to true.

How to Control Custom Headers and Footers From a Form

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Forms can optionally use javascript to control the content and styling of headers and footers. You may want to to do this for example to inject calculated or dynamic content in the header or footer. To allow a form to control the header/footer behavior first you will need to make sure the **config.disableFormOverrides** property of the JSON[1] options file is set to false.

You will then need to inject a function into the window object of your form using javascript inside your form as in the example below.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

```
window.phantomReceiptCallback = function(actionName,data) {
if (actionName=="orientation") {
       // return either "landscape" or "portrait" or fall
       // this action is only invoked once per form
}
if (actionName=="format") {
       // return the paper format eg "A4","letter",etc or fall
       // this action is only invoked once per form
}
if (actionName=="margin") {
       // return a JSON object defining the margin eg {"top": "0.5cm","left": "0.5cm","bottom":
"0.5cm","right":"0.5cm"} or fall
       // this action is only invoked once per form
}
if (actionName=="headerHeight") {
       // return the header height  eg "1cm" or fall
       // this action is only invoked once per form
}
if (actionName=="footerHeight") {
       // return the footer height  eg "1cm" or fall
       // this action is only invoked once per form
}
if (actionName=="header") {
       // return the header template string eg "<p>This is the header</p>" or fall
       // you can specify the tokens ${page} and ${numPages} in the template you are returning
and they will be replaced appropriately
       // you can access the current page and number of pages programmatically by using
data.pageNum and data.numPages respectively
       // this action is invoked for every page on the form
}
if (actionName=="footer") {
       // return the footer template string eg "<p>Page ${page} of ${numPages}</p>" or fall
       // you can specify the tokens ${page} and ${numPages} in the template you are returning
and they will be replaced appropriately
       // you can access the current page and number of pages programmatically by using
data.pageNum and data.numPages respectively
       // this action is invoked for every page on the form
}
// NOTE: always return the value defined in Manager, if not explicitly overriden as above
return data.defaultValue;
// this is the value from the JSON options file
```

Here is an example of using a form function to inject the tracking code into the form footer on page 1 for a Composer form. You can, for example, add this javascript to a composer form by dropping on a "Business Rule [General Purpose]" with the below script content.

```
window.phantomReceiptCallback = function(actionName,data) {
if (actionName=="footer") {
	if (data.pageNum==1) {
		var trackingCode = sfc.getPhaseValue("TrackingCode");
		return "<p class='phantom-footer'>Page ${page} of ${numPages} - "+trackingCode+"</p>";
	}
	return "<p class='phantom-footer'>Page ${page} of ${numPages}</p>";
}
// NOTE: always return the value defined in Manager, if not explicitly overriden as above
return data.defaultValue; // this is the value from the JSON options file
```

Understanding How Headers and Footers are Rendered

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

When Manager create a PDF receipt, it uses **phantomjs** to render each header and footer in a separate HTML DOM prior to generating a PDF receipt. This has several consequences:

- The pagination process requires a fixed header and footer height in order to correctly size the pages. This means that every page will reserve a fixed amount of space for the header and footer even if there is no content present. You can set this fixed header and footer height by setting the **paperSize.header.height** or **paperSize.footer.height** properties in the JSON[1] options file. Content that exceeds this height will be clipped and not show on the receipt.
- Because each header and footer is rendered in a separate HTML DOM, CSS classes defined in the form itself would not ordinarily be available to use, but instead, styling would need to be applied using in-line styles. In order to make styling, easier the service provides some additional capabilities to help make styling easier.
- You can use the **config.customCSS** property to add additional CSS style definitions to the form.
- If the **config.supportHeaderFooterCSSClasses** is set to true then any CSS classes used in a header or footer template are converted automatically to inline styles. This also allows the use of CSS classes already defined in the form itself, not just those defined in the **config.customCSS** property.

Next, learn about field overlay receipts using AcroForms.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

Field Overlay Receipts Using AcroForms

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

By default, Manager generates a PDF receipt for every completed transaction. This receipt, however, is based on the design of the completed Maestro form, which might not meet your organization's design or legal requirements. Your organization, for instance, might prefer to use PDF forms they've already designed, such as **AcroForms**[1]. If this is the case, you should consider using Field Overlay Receipts.

Field Overlay Receipts allow organizations to use a PDF form as a receipt template. Then, when a transaction is completed, the customer's data is inserted into the fields of the PDF.

The advantage of this approach is that receipts will look identical to the original PDF. This means:

- They won't need to be re-approved by design or legal departments.
- No one will need to be retrained on how to process the receipts.
- The same form can be used if there is a need to partially complete it, print it out, and complete it with pen and ink.

There are, however, some potential draw-backs:

- Additional development is required to overlay the data fields over the existing PDF.
- Hide and show rules are not honored. All fields are always displayed.
- Repeats are only honored to the extent that they're repeated on the original form. The Form Designer must be careful not to allow more repeats than are presented on the PDF.
- Field lengths must be limited to ensure they don't overflow the space allocated on the PDF.
- The ability to re-design the data entry experience is limited. No new fields can be added, since there is nowhere to place them.

Despite these draw-backs, Field Overlay Receipts are an excellent option for organizations that need to move quickly and integrate Manager with their current processes, rather than adapting to Manager's way of doing things.

Getting Started

Before we continue:

1. Download Adobe Acrobat Pro DC. This is the easiest software for converting a PDF form into an interactive form, which is what Manager needs to insert data into a PDF. The software requires a monthly subscription, but there is a 7-day free trial.

---

[1] AcroForms are the original PDF forms technology that accepts input in both Forms Data Format (FDF) and XML Forms Data Format (XFDF). Many 3rd party vendors support AcroForms.

2. Choose a PDF form to use as a receipt template. If you don't have a PDF form available, either use this example PDF from Adobe or create your own PDF in Microsoft Word.
3. Create a form in Maestro.

In this topic, we'll be using this Maestro form as an example:



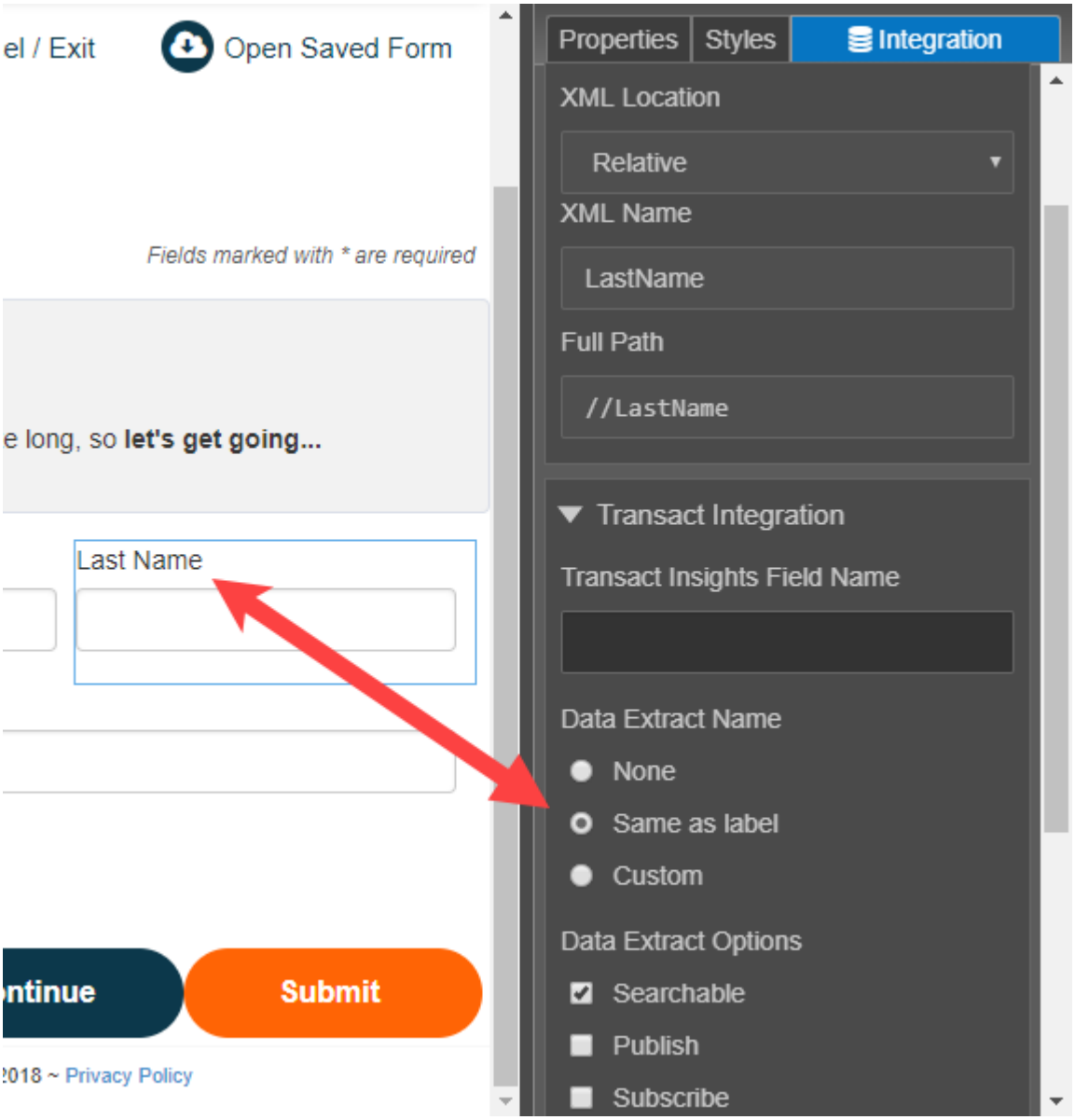
As we can see, this form has various field types, including text fields, a radio button group, and a checkbox. This will allow us to see how different types of data can be represented in a receipt.

Step 1: Add Data Extracts to form fields

Data Extracts allow you to extract data from a completed transaction and integrate that data with external systems. One use of this feature is to transfer data from a form into a PDF file.

To implement Data Extracts:

1. Login to Maestro.
2. Open a form.
3. Select a form field.
4. Open the **Integrations** tab.
5. Expand the **Transaction Integration** panel.
6. For the **Data Extract Name** field, select **Same as Label**.



7. Repeat the previous step for every field that needs to be inserted into the PDF form.
8. Click **Save** to update the changes.

Step 2: Export the form from Maestro

1. Click **Build**.
2. In the modal that appears, click **Build**.
3. Click **Download TM Form Version**.

Step 3: Publish the form to Manager

1. Login to Manager.
2. Navigate to **Forms**.
3. Click **Import Form**.

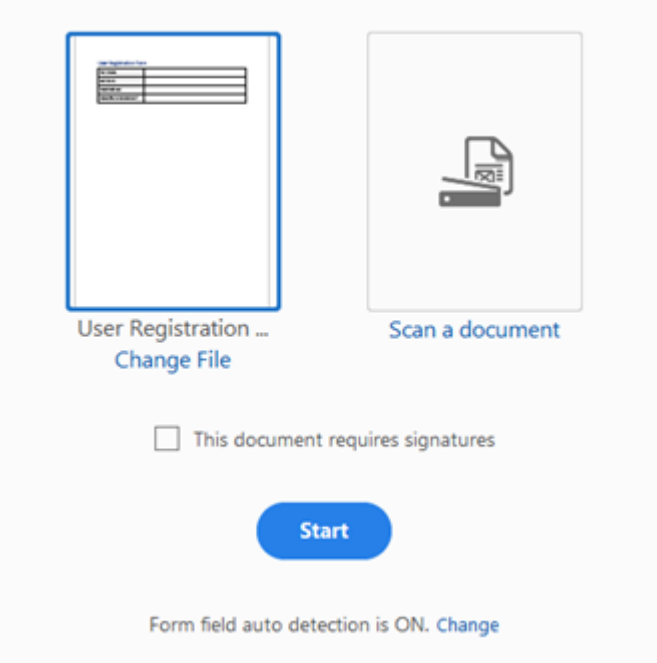
4. Upload the **Form Version Archive File**. This is the **TM Form Version** downloaded in step 2.
5. Select **Create a Form**.
6. Click **Proceed to Import**.
7. Click **Import**.

Step 4: Create a fillable PDF with Adobe Acrobat Pro DC

In this step, we're going to convert a PDF into what Adobe calls a fillable PDF. This will allow Manager to insert data into the PDF file.

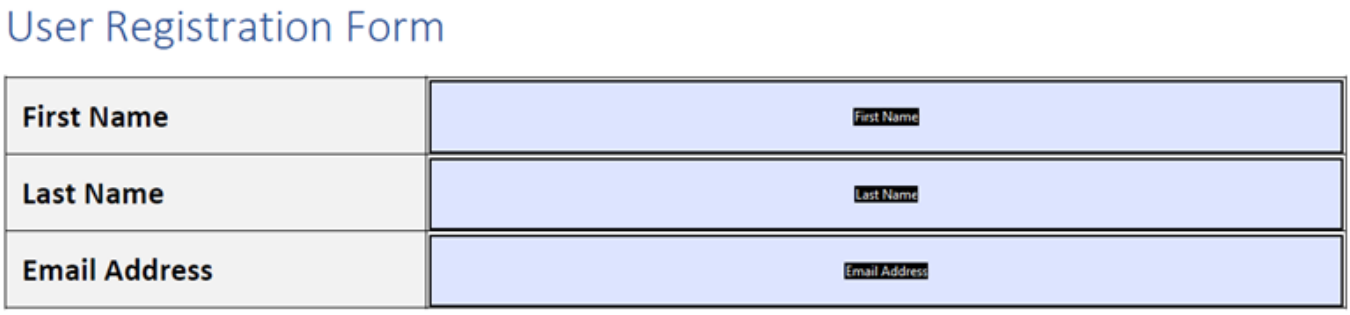
To do this:

1. Open Adobe Acrobat Pro DC.
2. Go to **Tools > Prepare Form**.
3. Click **Select a file**.
4. Select the PDF file.
5. Click **Start**.



During this process, Acrobat will attempt to detect and create fillable form fields inside the PDF. This isn't a seamless process though, so manual adjustments may be necessary.

In this example, Acrobat recognized three text fields:

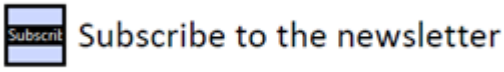


But the radio buttons and checkboxes need to be manually inserted via the toolbar.

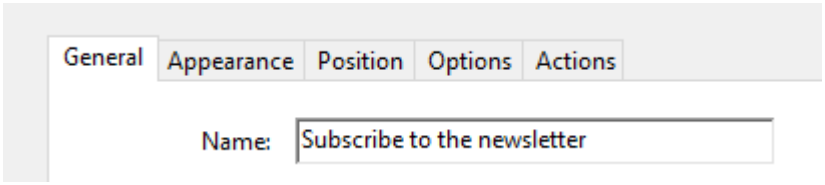


Step 4.1: Insert a checkbox

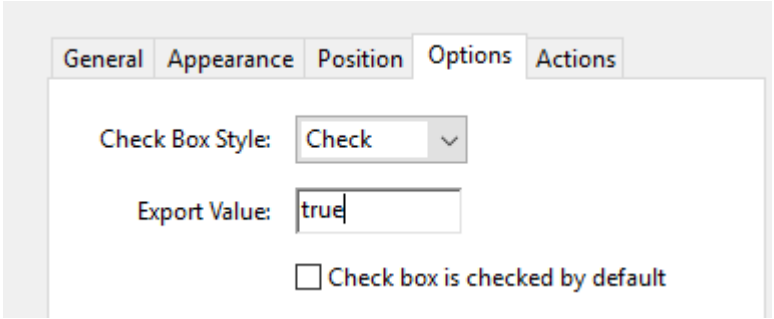
1. Click the **Check Box** icon in the toolbar.
2. Click inside the PDF to place the checkbox.



3. Double-click the checkbox to open the **Check Box Properties** window.
4. On the **General** tab, update the **Name** field. This name should match the field's label that was defined in Maestro. In this example, the label is "Subscribe to the newsletter".

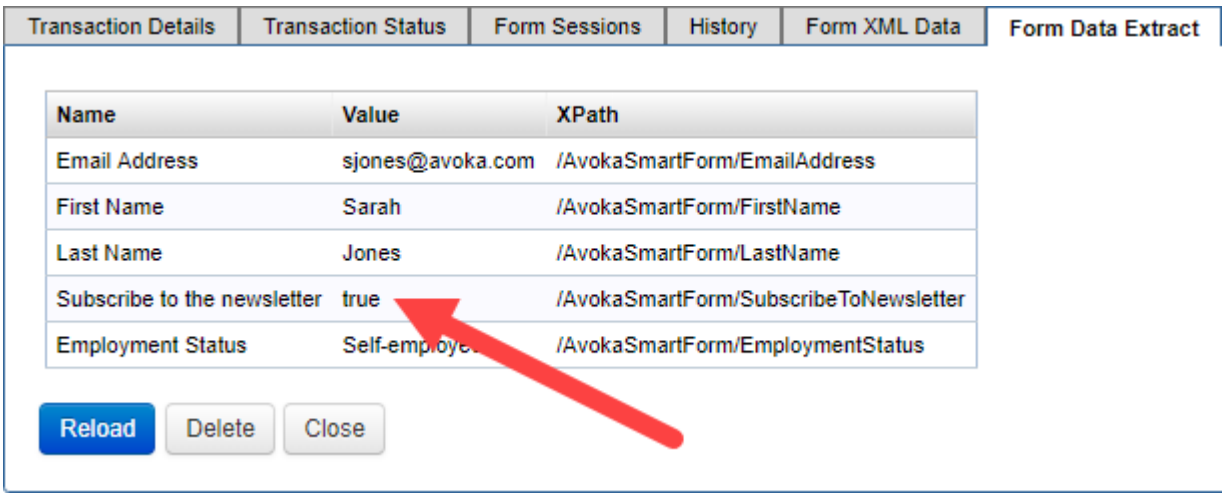


5. On the **Options** tab, set the **Export Value** as "true".



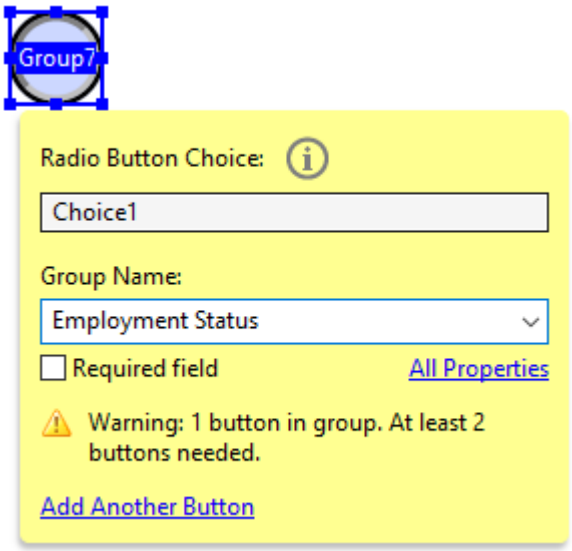
We need to set the value to "true" because this is how the value is stored in Manager. When the **Export**

Value matches the value in Manager, the checkbox will be checked in the PDF.

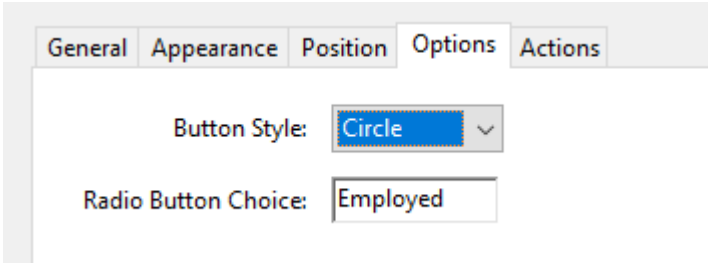


Step 4.2: Insert a radio button

1. Click the **Radio Button** icon in the toolbar.
2. Click inside the PDF to place the radio button.
3. Set the **Group Name** field to the radio button group's label that was defined in Maestro. In this example, the radio button group's label is "Employment Status".

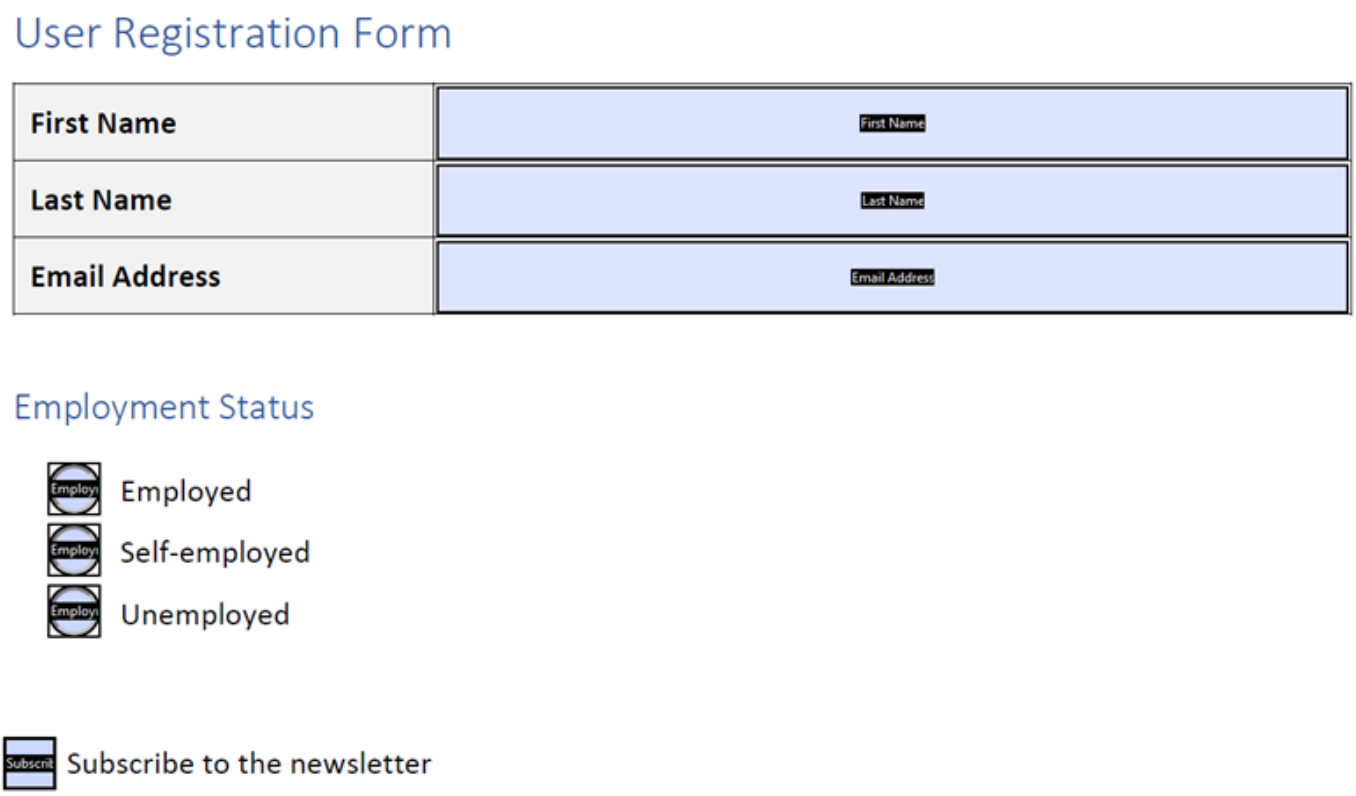


4. Double-click the radio button to open the **Radio Button Properties** window.
5. On the **Options** tab, set the **Radio Button Choice** field to the value of an individual radio button that was defined in Maestro. In this example, the first radio button value is "Employed".

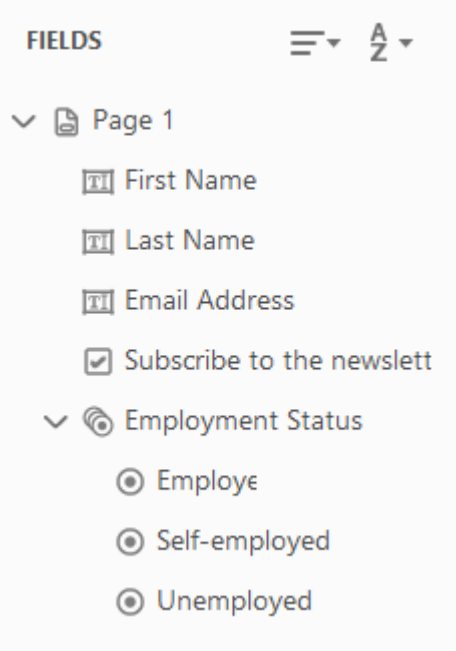


6. Repeat this process for all of the form's radio buttons. In this example, the remaining radio button values are "Self-employed" and "Unemployed".

You should end up with a form that resembles the following:



The structure of the form should also be visible on the right side of the screen:

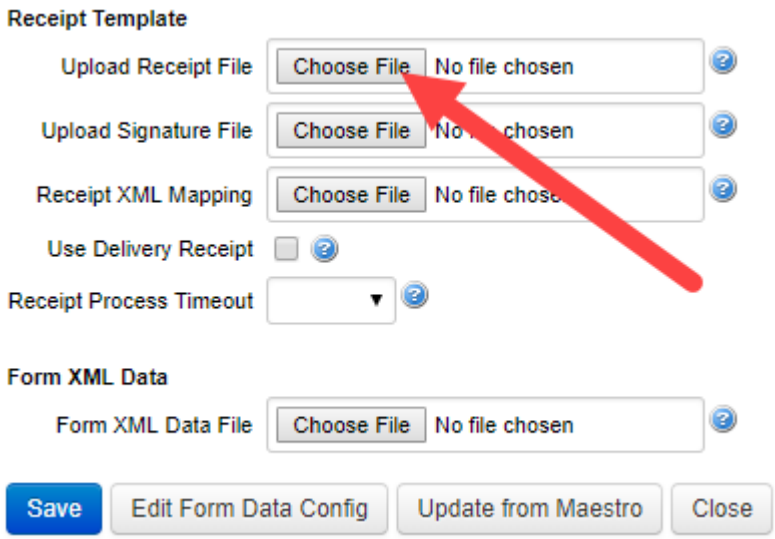


If any of the labels in this list don't match the labels in Maestro, fix them before continuing. If you have any other issues preparing the form, read the official documentation for Acrobat Pro DC.

To end this section, navigate to **File > Save As...** and save the form.

Step 5: Upload the fillable PDF to Manager

1. Return to Manager.
2. Navigate to **Forms**.
3. Open the published form. If you have a lot of published forms, you may need to use the search feature.
4. Click the **Form Versions** tab.
5. Click the **Edit Form Version** icon.
6. Upload the fillable PDF file to the **Upload Receipt File** field.



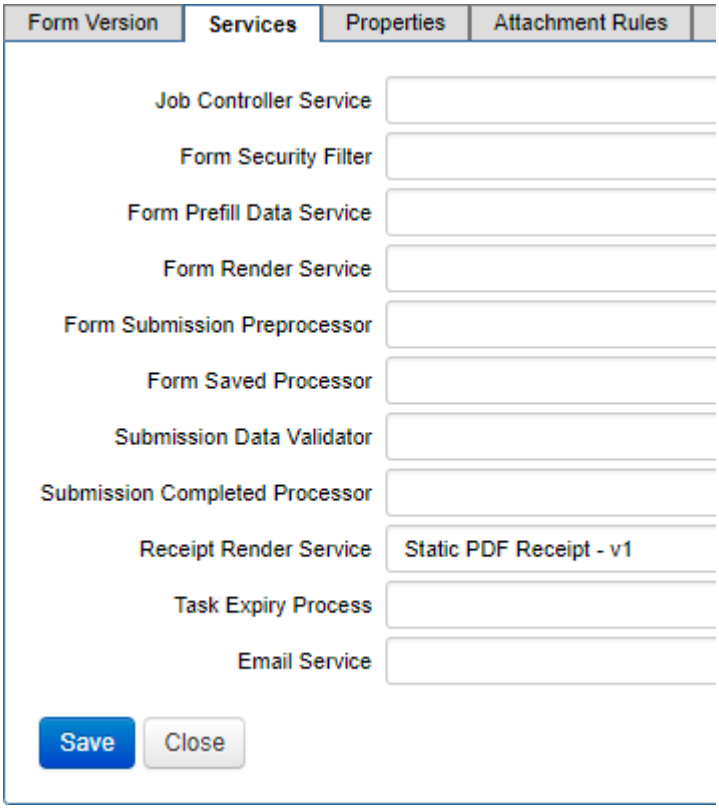
7. Click **Save** to update the changes.

Step 6: Configure the Receipt Render Service

Within Manager, services are used to access a range of business-related functionality on demand. One such service is the Receipt Render Service, which is used to generate PDF receipts.

To enable this service:

1. On the **Form Version** page, click the **Services** tab.
2. From the **Receipt Render Service** dropdown, select **Static PDF Receipt v1**.
3. Click **Save**.

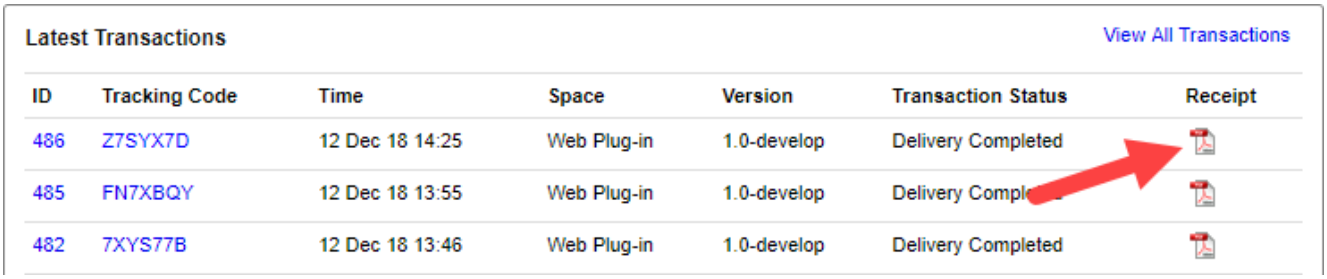


Step 7: Verify the results

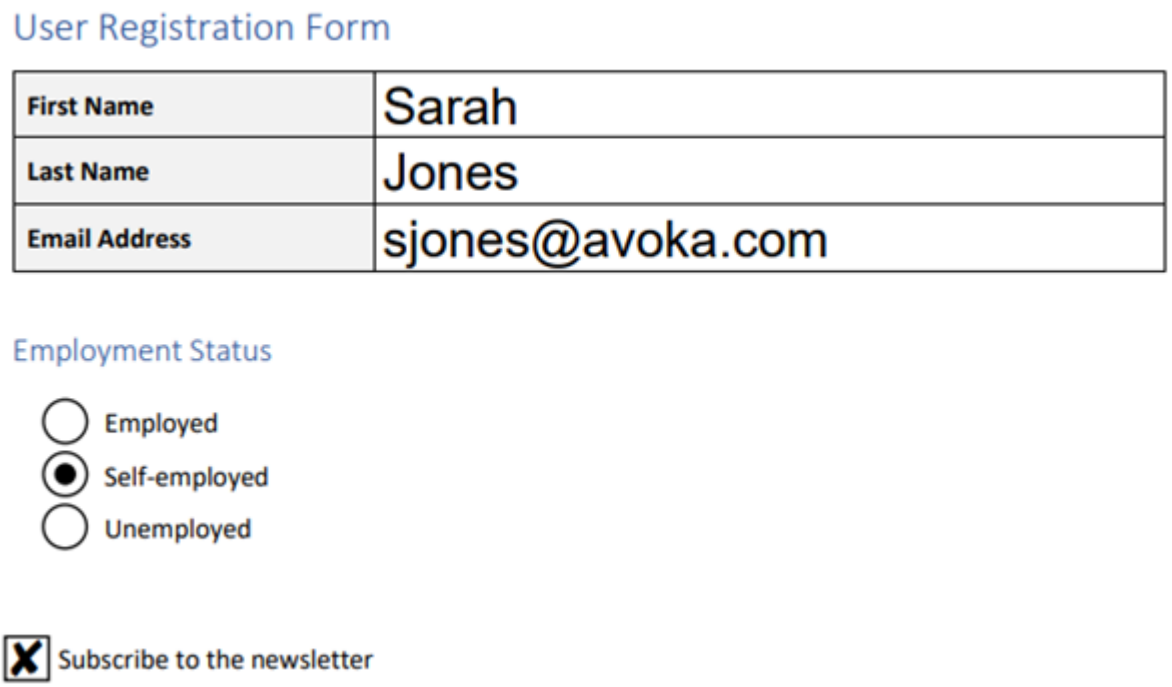
1. Navigate to the **Form Dashboard**.
2. Click the **Test Render** icon for the latest form version.



3. After the form loads, submit some sample data.
4. Refresh the **Form Dashboard**.
5. Click the **View PDF Receipt** icon.



The receipt should contain data from the completed transaction:



Next, learn how headers and footers are rendered.

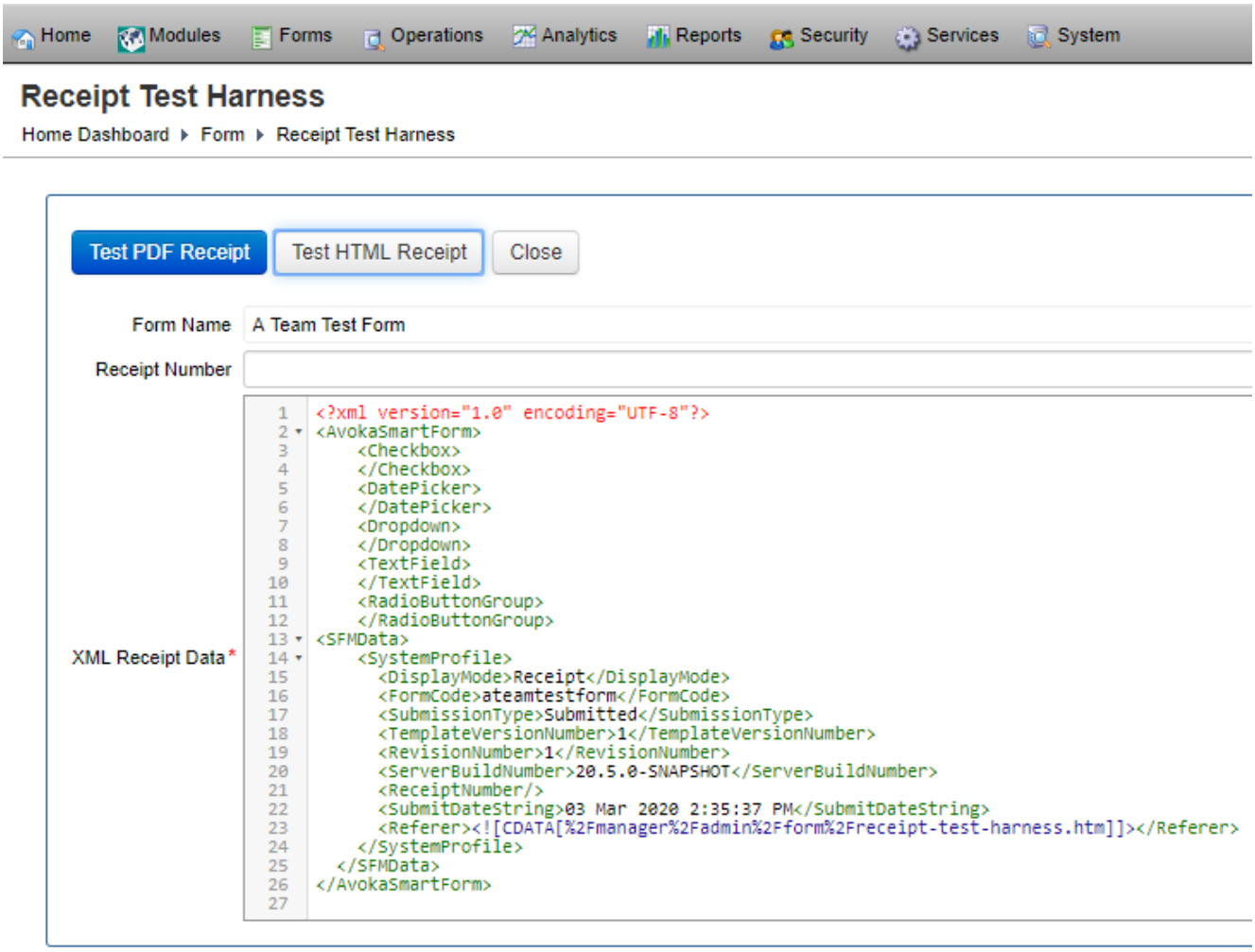
Test a Receipt with Receipt Test Harness

Journey Manager (JM) | Platform Developer | 5.1 & Higher

Manager allows you to manually render a form as a PDF receipt, so you can preview it and debug it. You can generate a receipt as PDF or HTML, which is determined by the form type you have used to create the form. The PDF receipt is suitable for more traditional **AcroForms**[1], whereas the HTML receipt refers to Composer or Maestro forms. However, in either case, the form is rendered as PDF. We recommend you render the form as HTML to test your receipt in a browser and debug issues.

To render a form as a receipt:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Click **Receipt Test Harness** under the **Form Spaces** section.

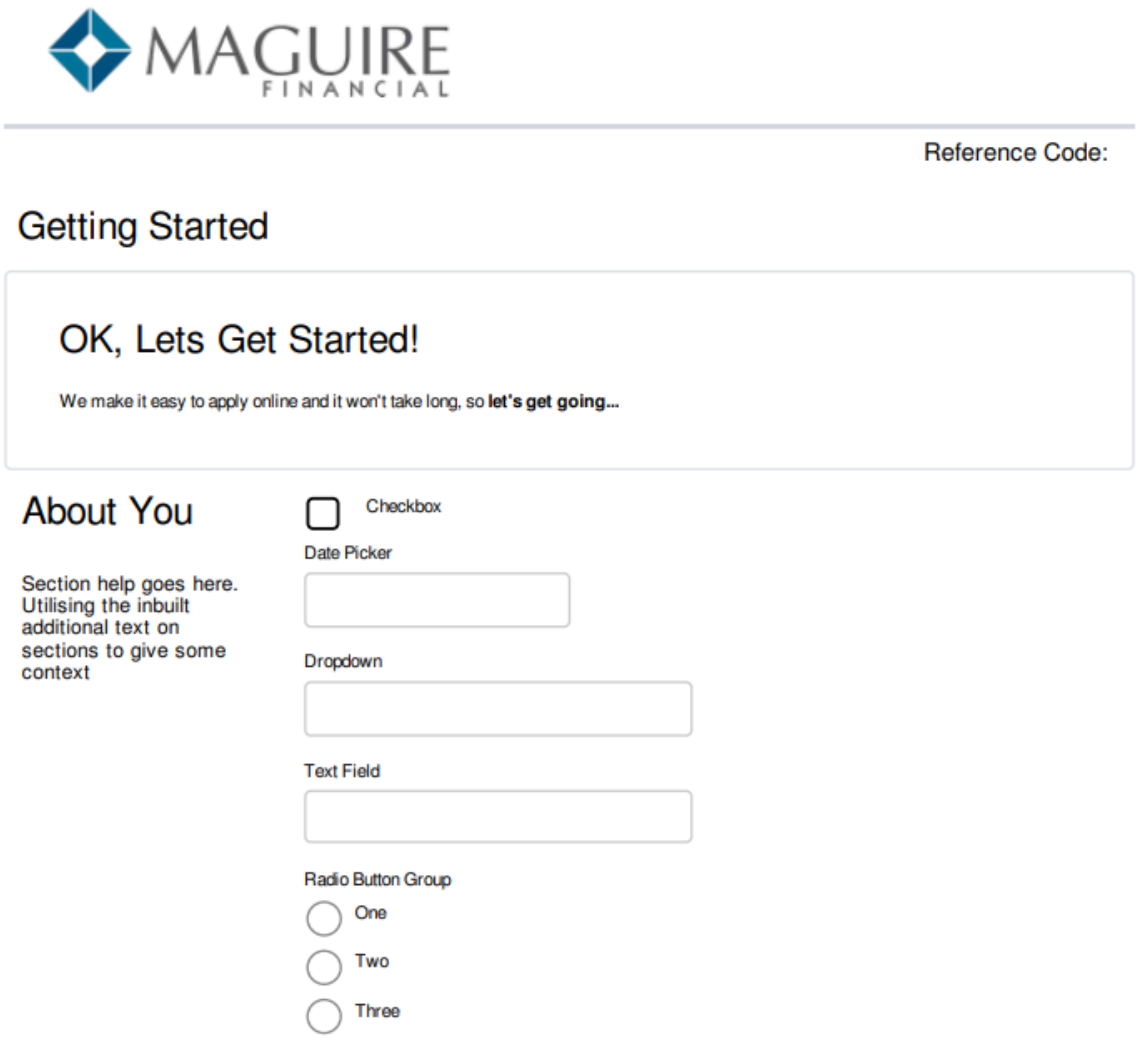


4. Enter a new receipt number in the **Receipt Number** field.

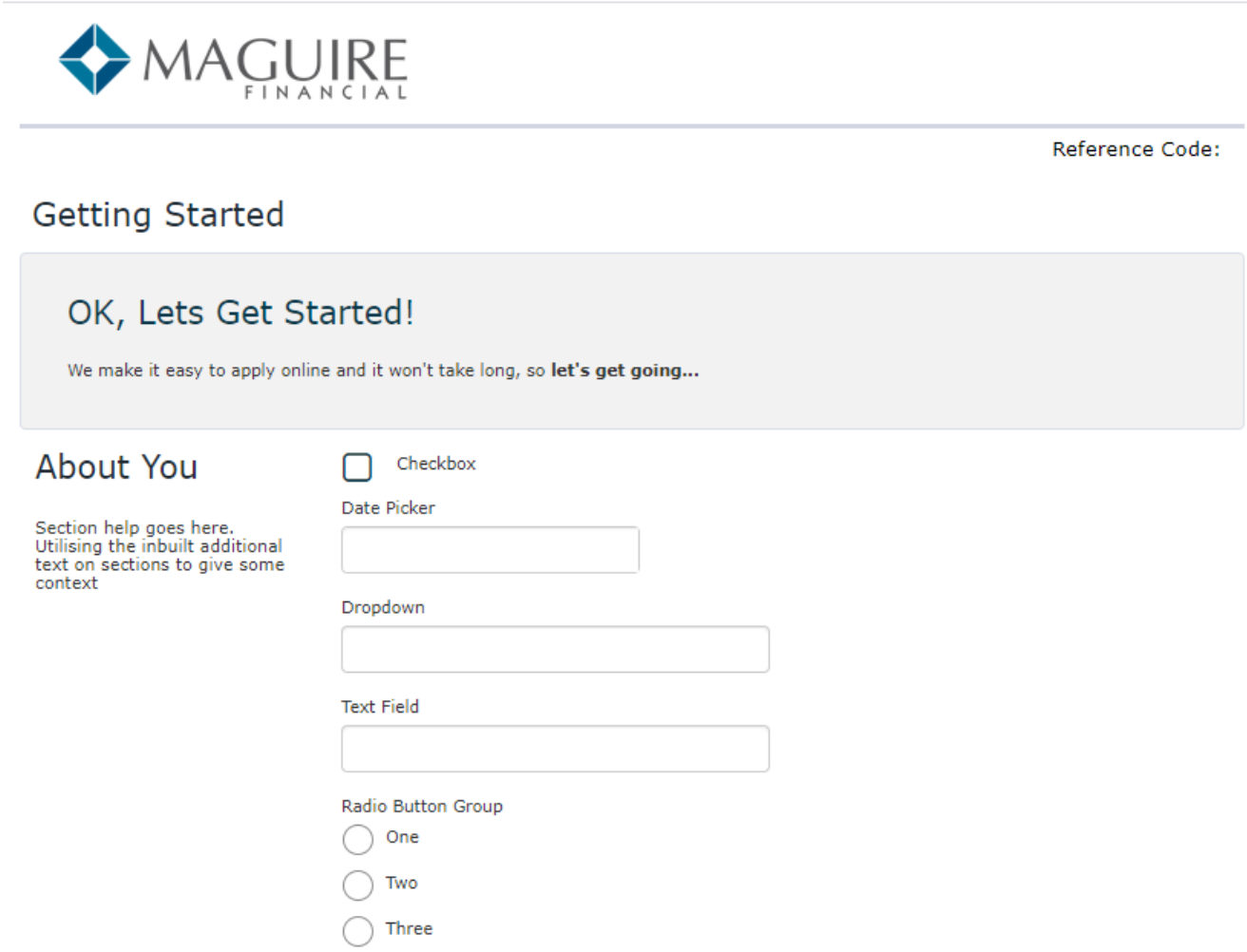
---

[1]AcroForms are the original PDF forms technology that accepts input in both Forms Data Format (FDF) and XML Forms Data Format (XFDF). Many 3rd party vendors support AcroForms.

5. Update the XML data in the **XML Receipt Data** field to test the new changes.
6. Click **Test PDF Receipt** to render a PDF receipt in a new window.



7. Click **Test HTML Receipt** to render a form using **PhantomJS**[1] to produce a receipt as a PDF file. The way this works is by first rendering the Maestro HTML code then 'printing' the result to PDF and returning the PDF data. You should use this functionality to test your receipt in the browser and debug issues.



8. Click **Close**.

Next, learn about the form dashboard.

[1] PhantomJS is a scripted, headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment. PhantomJS is based on WebKit making it a similar browsing environment to Safari and Google Chrome (before Chrome's fork of WebKit evolved into Blink). It is open-source software released under the BSD License.

PDF Receipt Options

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager gives you a number of options to generate a PDF receipt or document of records. The options are:

- Transact PDF Receipts
- XFA Receipts
- Field Overlay Receipts
  - using Acrobat
  - using Livecycle Designer
  - using Texcel
- Custom Receipt Generation

This article does not recommend one of these approaches over another - the choice will depend on the requirements of each situation. Each solution has its own pros and cons. In addition, each tool used has a learning curve, and productivity will be determined by the skills of those using it.

Manager PDF Receipts

Manager is able to create a PDF receipt directly from the form design UI. This receipt is very similar to the interactive form, but with a number of modifications to optimize it for printing. These modifications include:

- Reducing the use of color (users often complain that it uses up too much printer ink)
- Removal of the navigational menu, so that the form becomes a single long multi-page document.
- Removal of all interactive features such as buttons and drop-down lists.

The exact optimizations used can be modified by adjusting a Composer skin.

Pros

- Requires no additional configuration or effort. (It is built into the product.)
- Honors all business rules in the form, such as hide-show logic and repeats.

Cons

- Does not generate PDF/A compatible PDFs. This may be a requirement for some organizations.
- Has limited support for headers, footers, and pagination options.
- Some problems have been reported with layouts, particularly character spacing on Linux.
- Since interactive forms tend to be quite spaced out, this can result in lengthy receipts with multiple pages and low field densities.
- Has a different layout to existing paper forms. (This may be an issue for downstream systems or people. This may also result in additional approval steps.)
- Not suitable for the case where a customer partially completes a form interactively, prints it out, and then completes the rest of the form using ink. (This can be achieved with additional effort in the form design.)
- Since the length and layout of the document may vary based on hide-show rules and repeats, the location of fields on the page cannot be guaranteed. This may impact downstream systems such as OCR, imaging, or digital signatures.

XFA Receipts

Composer is able to generate a PDF in XFA format. This is a format suited to high-fidelity PDF documents.

Pros

- Has strong capabilities for producing high-fidelity PDF documents, including headers, footers, and pagination options.
- Text formatting and layout is good.
- Produces PDF/A receipts.
- Honors all business rules in the form, such as hide-show logic and repeats.

Cons

- Requires Adobe LiveCycle. This incurs additional costs, and additional effort to provision and maintain.
- Since interactive forms tend to be quite spaced out, this can result in lengthy receipts with multiple pages and low field densities.
- Has a different layout to existing paper forms. (This may be an issue for downstream systems or people. This may also result in additional approval steps.)
- Not suitable for the case where a customer partially completes a form interactively, prints it out, and then completes the rest of the form using ink. (This can be achieved with additional effort in the form design.)
- Since the length and layout of the document may vary based on hide-show rules and repeats, the location of fields on the page cannot be guaranteed. This may impact downstream systems such as OCR, imaging, or digital signatures.

Field Overlay Receipts

Instead of generating a complete form, it is possible instead to import an existing (flat) PDF form, and lay the data fields over the top of this existing form. Regardless of the technology used to do this field overlay, there are some pros and cons.

Pros

- High fidelity - the receipt will look the same as the existing PDF.
- No need for additional approval steps, as the receipt is the same as the existing document.
- Information density is high, because the existing PDF has probably already been designing to minimize the number of pages.
- This type of form can be used if there is a need to partially complete the form, print it out, and then complete it with pen and ink.
- Receipt PDF is identical to the existing PDF, which will minimize downstream implications, including retraining. It also eliminates the need for additional approvals.
- PDF/A supported.

Cons

- Additional development effect is required to perform the task of overlaying the data fields over the top of the existing PDF.
- Additional development effort is required to map the XML data fields into a flat field naming structure. This can be done either within Composer, or using a mapping file in Manager.
- Hide or show rules are not honored. All fields are always displayed.
- Repeats are only honored to the extent that they are repeated on the original form. The form designer must be careful not to allow more repeats that are presented on the PDF. Mapping of repeats can be complicated.
- Field lengths must be limited in the interactive form to ensure that they don't overflow the space allocated on the PDF.
- Multi-line text must be controlled carefully to ensure they they don't overflow the space allowed in the PDF.
- Requires very detailed work to accurately size and place fields so that they match the underlying PDF.
- To some extent, the ability to re-design the data entry experience is limited. In particular, no new fields can be added, since there is nowhere to place them.

Field Overlay Receipts Using Adobe Acrobat

Adobe Acrobat has the ability to overlay fields over the top of an existing flat PDF. This uses a technology known as AcroForms.

Pros

- Relatively inexpensive per-seat development license.
- Makes no modifications to underlying PDF, and so the receipt maintains 100% fidelity with the original PDF.
- Manager supports AcroForms technology with no additional provision or configuration.
- Has a field recognition feature, which allows fields to be automatically recognized. This can save a lot of time, particularly when large numbers of PDF forms need to be converted. The effectiveness of this feature varies widely, based on the specifics of particular documents. It is suggested that this feature be tested against a representative sample of documents before being used on large scale.

Cons

- The field creation and editing environment within Acrobat is considered by many to be quite rudimentary, particularly for larger forms, though this is subject to personal preferences.
- No ability to modify the underlying PDF document. For example, to remove explanatory text that may be inappropriate, or to make minor modifications to the form.
- If a new version of the underlying PDF document is produced, then the entire process needs to be repeated on the new document. Copying and pasting fields may help in some cases.

Field Overlay Receipts Using Adobe LiveCycle Designer

Adobe LiveCycle Designer has the ability to overlay fields over the top of an existing PDF document.

Pros

- Relatively inexpensive per-seat development license.
- Makes no modifications to underlying PDF, and so the receipt maintains 100% fidelity with the original PDF.
- The field editing and creation environment is reportedly better than Acrobat, though this is subject to personal preferences.

Cons

- Requires Adobe LiveCycle. This incurs additional costs, and additional effort to provision and maintain.
- No ability to modify the underlying PDF document. For example, to remove explanatory text that may be inappropriate, or to make minor modifications to the form.
- If a new version of the underlying PDF document is produced, then the entire process needs to be repeated on the new document. Copying and pasting fields may help in some cases.

Field Overlay Receipts Using Texcel

Texcel FormBridge is a third-party product that can intelligently recognize fields in an existing PDF document. In many cases, this results in a very quick conversion of existing PDF documents into AcroForms which can then be imported into Manager. Some manual cleanup is almost always required. This is particularly useful when large numbers of PDFs need to be converted.

In addition, Texcel will also produce a list of fields and their attributes. These field definitions can be imported into Composer, and uses as the basis for the form design in Composer. This can accelerate the production of forms in Composer, as well as automate the mapping of fields between Composer and the generated AcroForm.

The effectiveness of the conversion is dependent on the structure of the original PDF documents. It is suggested that this feature be tested against a representative sample of documents before being used on large scale.

Pros

- Automated recognition of fields can save a lot of time.
- The recognized fields are used both to generate a AcroForm PDF, and to import into Composer as a starting point for building the interactive form.
- In addition, the boilerplate text such as instructions, product disclosures and legal copy are also able to be imported into Composer, reducing the amount of effort in doing this.
- Strong and capable editing environment.
- The underlying PDF document is converted into lines and text. This means that the PDF itself is editable, if this is required.

Cons

- Texcel incurs additional cost to convert each page of the original PDF.
- Because the underlying PDF is converted, there may be some differences between the original PDF and the generated one. The differences, if any, will depend on how the original PDF was constructed. Some configuration options in Texcel can be tuned, but ultimately, Texcel should be tested on sample documents.

Custom Receipt Generation

Any third-party tools or systems can be used to generate PDF receipts, as long as these systems have an interface that can be invoked from Manager.

Pros

- No limits as to what can be achieved.

Cons

- The third-party tools systems must be purchased, provisioned and managed.
- A Manager service must be developed that can invoke the receipt service.
- PDF receipts can be quite large. If the system is not hosted close to Manager, this may incur processing delays or bandwidth costs.

Next, learn how to configure receipts server node.

Generate Multiple PDF Receipts from a Single Submission

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Sometimes it is necessary to generate multiple different PDF receipts from a single submission. There are several use cases for this, for example:

- You may want a different PDF receipt depending on some data in the form. For example, a slightly different variation of the receipt depending on which state the applicant is from.
- You may want to generate multiple separate documents, and concatenate them together. For example, a single application form may result in both a credit card and a check account PDF to be generated. You may want to concatenate these two documents together, or perhaps just provide the two separate documents to downstream business processes.

For this particular example, we are going to generate a certificate for a course attendee. When the user fills in the form, they will be able to choose between different PDF receipts, and green receipt, and a receipt with a picture of flowers on it.

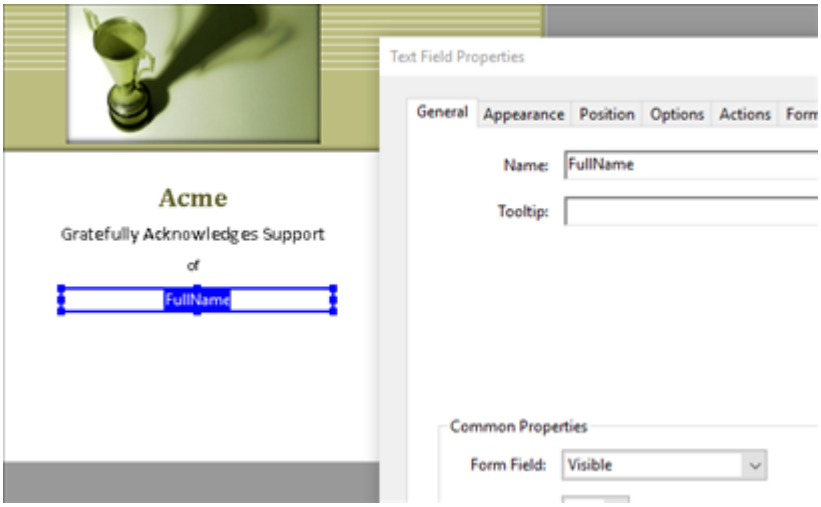




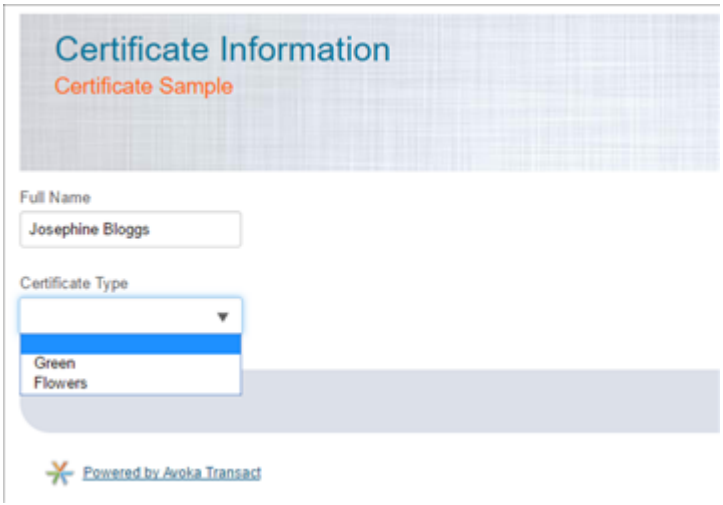
In this case, we are going to generate “pixel perfect” or AcroForm receipts. Please refer to this article for more information about AcroForm Receipts: How to import an Acrobat PDF form as a Transact receipt template (AcroForms).

The code is slightly different if you want to generate other types of receipts, but the principles are similar.

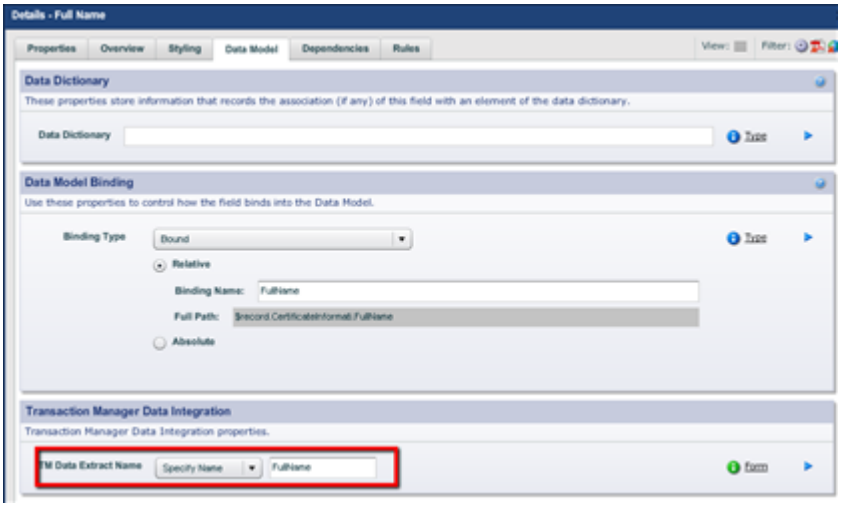
Each of the Acroforms has a single Acroform field placed on it – this field is named “FullName”. For consistency, both forms have the identical field name – but in a real scenario there can be a combination of common fields and form-specific fields. You just need to make sure that the data entry form provides all the necessary fields.



The data entry form has two fields, a field for the person's name, and a dropdown containing the values "Green" and "Flowers".



Each of these fields has a data extract name. We will use the data extract values for binding to the Acroform, as well as for the logic in our script.



The techniques for creating and generating an AcroForm receipt is described in the AcroForm article. But for the multiple receipt example, we have to solve two additional problems.

- We need somewhere to store the two (or more) PDF receipt template files. By default, Transact Manager only stores one receipt template per form.
- We need to create a custom receipt rendering service. By default, the various PDF rendering receipt services that Transact Manager provides only generate a single receipt.

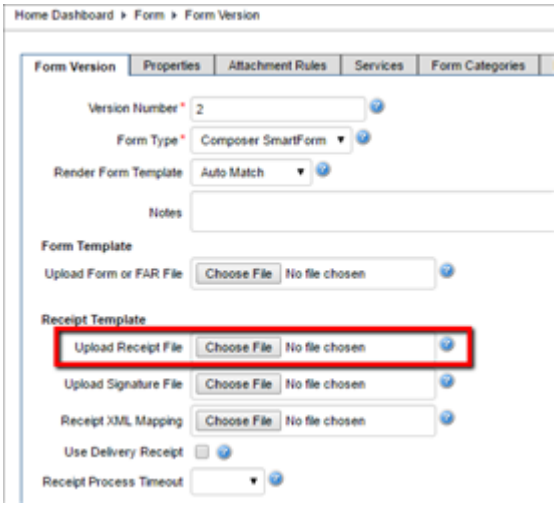
Storing multiple PDF receipts

In order to store multiple receipts, we will create a "dummy" form in Transact Manager for each receipt. This form is not intended for data entry, and it will never be visible externally. It's simply used as a place to store the receipt.

There are two ways to do this. Either:

- Deploy your data entry form twice, under two different form codes. Or
- Deploy your data entry form once. Then go into Transact Manager, and manually create a new form. You will need to upload a data entry form when you create this form, because Transact will not allow a form to be created without a data entry template. However, this can be any form, since the data entry form will never be visible.

Once you have defined your data entry form plus your additional "dummy" forms, you can upload the AcroForm PDF receipts to each of these form codes as shown below:



Custom Rendering Service

The Groovy code is shown below. This is an example of how you might write this script, and of course your needs may vary. The code is hopefully reasonably self-evident for Groovy programmers. You may want to refer to http://i-textpdf.com/ for more details about the iText PDF library which does the work of merging the field data into the AcroForm.

```
/* Render Receipt Groovy script.
   Script parameters include:
       form : com.avoka.fc.core.entity.Form
       request : javax.servlet.http.HttpServletRequest
       submission : com.avoka.fc.core.entity.Submission
       serviceDefinition : com.avoka.fc.core.entity.ServiceDefinition
       serviceParameters : Map<String, String>
   Script return:
       a PDF receipt document of type com.avoka.fc.form.service.DataDocument
*/
import com.avoka.core.groovy.GroovyLogger as logger
import com.avoka.fc.core.dao.DaoFactory
import com.avoka.fc.form.service.DataDocument
import com.itextpdf.text.pdf.AcroFields
import com.itextpdf.text.pdf.AcroFields.Item
import com.itextpdf.text.pdf.PdfReader
import com.itextpdf.text.pdf.PdfStamper
import org.apache.commons.lang.Validate
logger.info "Start render"

final DATA_EXTRACT_CERTIFICATE_TYPE = "Certificate Type"
final FLOWER_FORM = "certificate-flowers"
final GREEN_FORM = "certificate-green"

// Get the Data Extract
Map<String, String> dataExtractMap = submission.dataExtractMap
logger.info "dataExtractMap=" + dataExtractMap

// Using the Certificate Type field data to determine which form code to use
def certificateType = dataExtractMap.get(DATA_EXTRACT_CERTIFICATE_TYPE)
def formCode = ""
if (certificateType == "Green") {
  formCode = GREEN_FORM
} else {
  formCode = FLOWER_FORM
}
logger.info "formCode=" + formCode

// Get the appropriate form object
def form = DaoFactory.getFormDao().getFormByFormCode(formCode.toLowerCase())

// Call the function to render the AcroForm and return the result
def receiptDocument = renderAcroForm(form, dataExtractMap)
logger.info "End render"
```

```
return receiptDocument

// This function uses iText to inject the fields from the data extract map into the
AcroForm receipt
def renderAcroForm(form, dataExtractMap) {
    final CONTENT_TYPE_PDF = "application/pdf"
    Validate.notNull(form, "form parameter is null")
    Validate.notEmpty(dataExtractMap, "dataExtractMap parameter is null")

    def templateVersion = form.getCurrentVersion()
    Validate.notNull(templateVersion, "form.currentVersion parameter is null")

    byte [] templateData = templateVersion.getTemplateVersionData().getReceiptFileData()
    Validate.notNull(templateData, "form.-
currentVersion.templateVersionDate.receiptFileData parameter is null")

    // Create the output stream
    ByteArrayOutputStream stream = new ByteArrayOutputStream(256 * 1024)

    // Setup the PDF receipt template
    PdfReader templatePdfReader = new PdfReader(templateData)
    PdfStamper pdfStamper = new PdfStamper(templatePdfReader, stream)

    // Bind the AcroForm field values
    AcroFields acroFields = pdfStamper.getAcroFields()
    Map<String, Item> fieldMap = acroFields.getFields()
    for (String fieldName : fieldMap.keySet()) {
        if (dataExtractMap.containsKey(fieldName)) {
            acroFields.setField(fieldName, dataExtractMap.get(fieldName))
        }
    }

    // "Flatten" the AcroForm with the bound field data - the fields will no longer be
editable
    pdfStamper.setFormFlattening(true)
    pdfStamper.close()
    return new DataDocument(stream.toByteArray(), CONTENT_TYPE_PDF)
```

Next, learn how to configure receipts server node.

Import an Acrobat PDF Form as a Receipt Template (AcroForms)

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Some organizations have invested heavily in creating glossy PDF forms which are designed to be printed, filled, and posted in, or alternatively opened, filled electronically, and saved and emailed. Sometimes these assets can be re-used to ensure that:

- downstream manual processing processes do not need to be modified, or
- customers are provided with a copy of their application which meets marketing guidelines
- Wet signature processes can be implemented, using copy which has already been ratified by legal

This guide describes how an existing "print & fill" PDF forms can be used by Manager as a pixel-perfect, "artwork PDF" template, without writing any code. This technique requires Adobe Acrobat (additional licenses); Other techniques exist and are not covered by this guide.

What is an AcroForm?

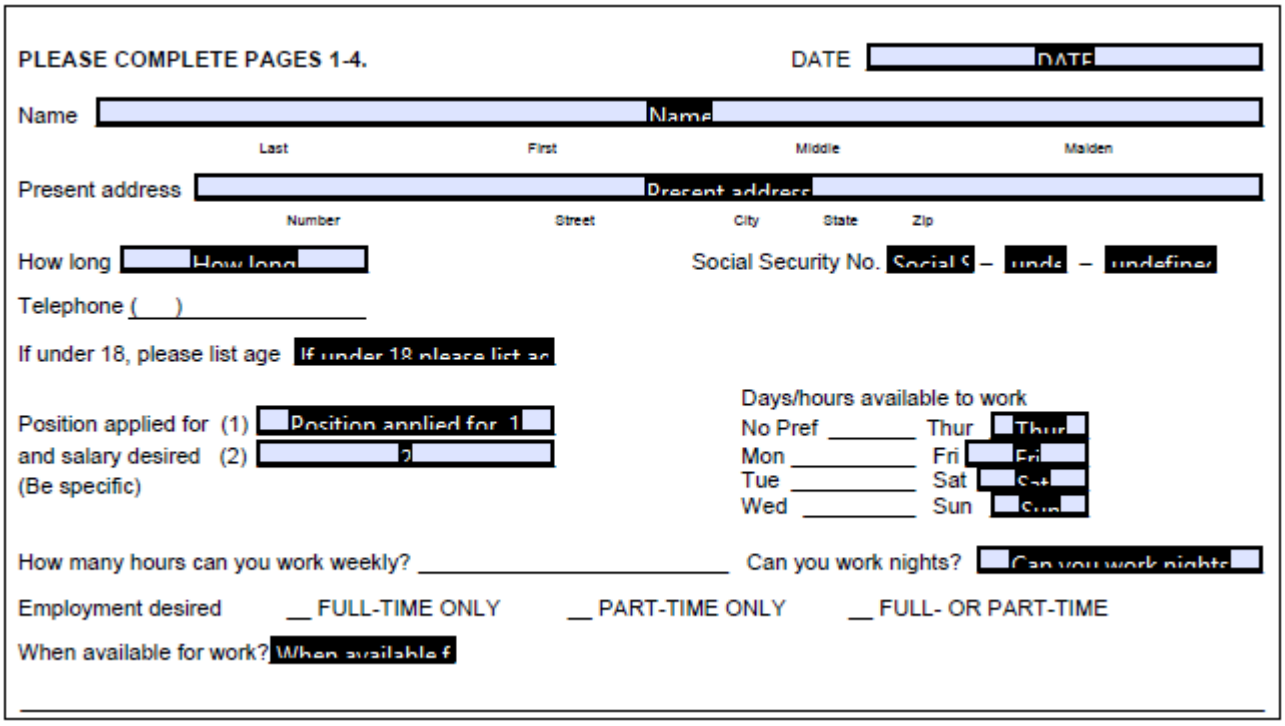
Forms which are created in Acrobat are called "AcroForms"; some PDF readers (Including Acrobat Reader) are capable of rendering these files as interactive PDFs, with features such as fields and radio buttons, and allows saving of data within the file. These should not be confused with interactive XFA forms which function in a specialized Adobe ecosystem (Adobe LiveCycle Designer ES & only Adobe Reader).

Manager re-purposes these PDFs as Templates associated with HTML5 forms on Manager, where HTML5 field data is merged with the template after submission for use as a Document of Record.

High Level Overview of steps required

At a high level, you need to follow these steps

1. Create your form in Composer, using the new form wizard. Drag and drop pages, sections, and fields, and add business rules from the palette, as you normally would. This produces your default dynamic PDF receipt, which will be discarded later. Publish your form to Manager, as you normally would.
2. Open the PDF form in Acrobat, and run the "Prepare Form" wizard. This will create something which looks like this:





3. In Acrobat, correct any issues. You may need to manually add fields, or rename them to make more sense. If you have authorization to make changes to the form, now would be a good time to replace "comb" fields with regular fields, and remove any character place-holders such as parentheses in a telephone field, or slash characters in a date field.

4. In Composer, add Data Extracts to each field. The name of the data extract must match the name of the field declared in Acrobat. Additionally you will need to add some invisible fields (Data Fields) with calculation rules to reformat any dates or other fields which need conversion before printing.
5. In Manager, open the form version, and upload a receipt file. Then under Services, set the Receipt Render Service to "Static PDF Receipt".

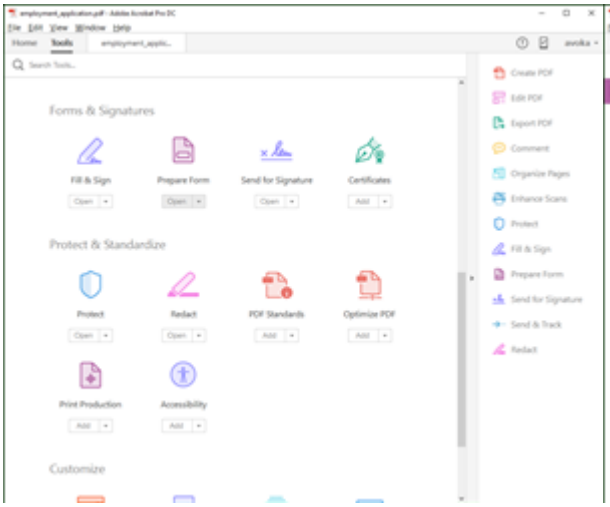
Each of the steps 2-5 is described in detail below:

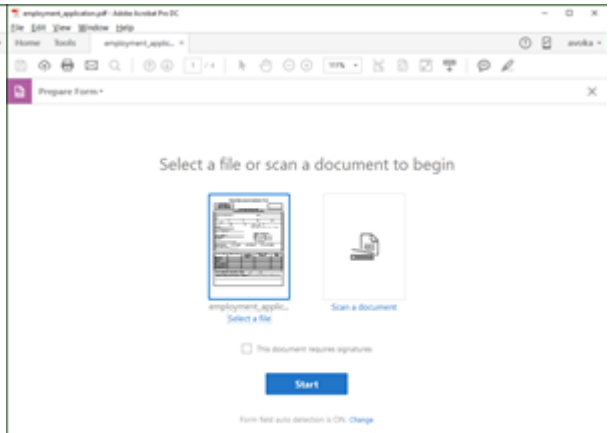
Step 2: Creating AcroForms

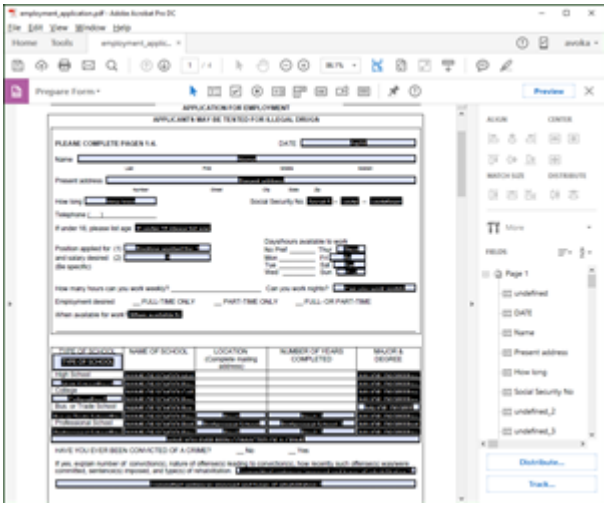
Adobe Acrobat Pro DC (2015 Release)

In order to create your AcroForm using Adobe Acrobat Pro DC (2015 Release) follow these steps:

1. Open the PDF form in Acrobat
2. Go to "Tools", and select "Prepare Form" (Open), under the "Forms & Signatures" heading.
3. Select "Start" (take note that "Form field auto detection is ON" is displayed under the button).
4. You can now "Save As..." and continue to the next section "Prepare Form" in Adobe Acrobat Pro DC.



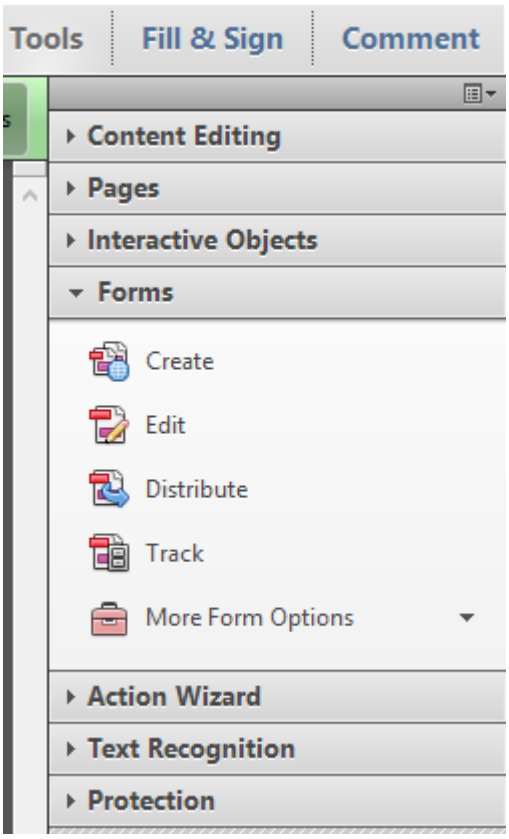


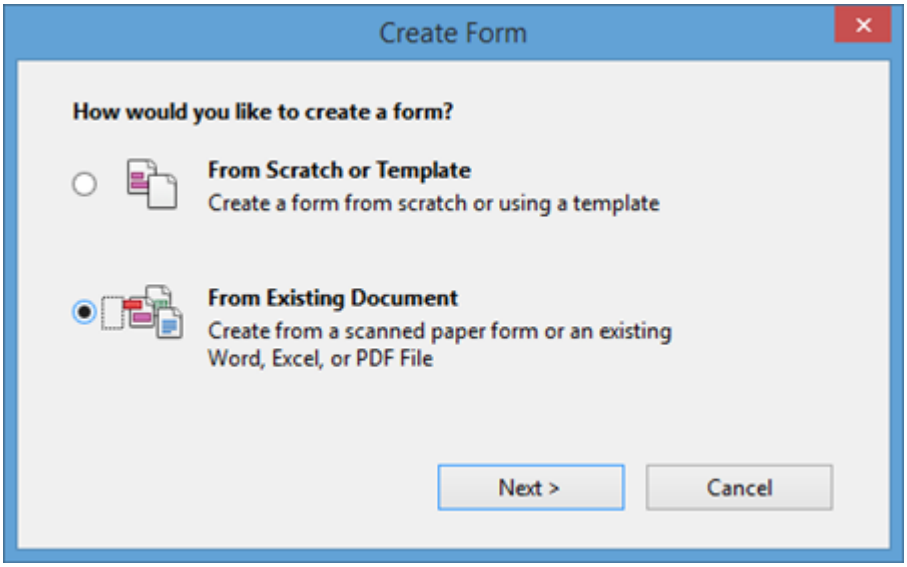


Acrobat XI and below

Acrobat 9 or Acrobat XI users can follow these steps:

1. Start, by accessing the "Forms" menu (under Tools, on the right-hand pane).
2. Select "Create", to start the wizard
3. You can now "Save As..." and continue to the next section.



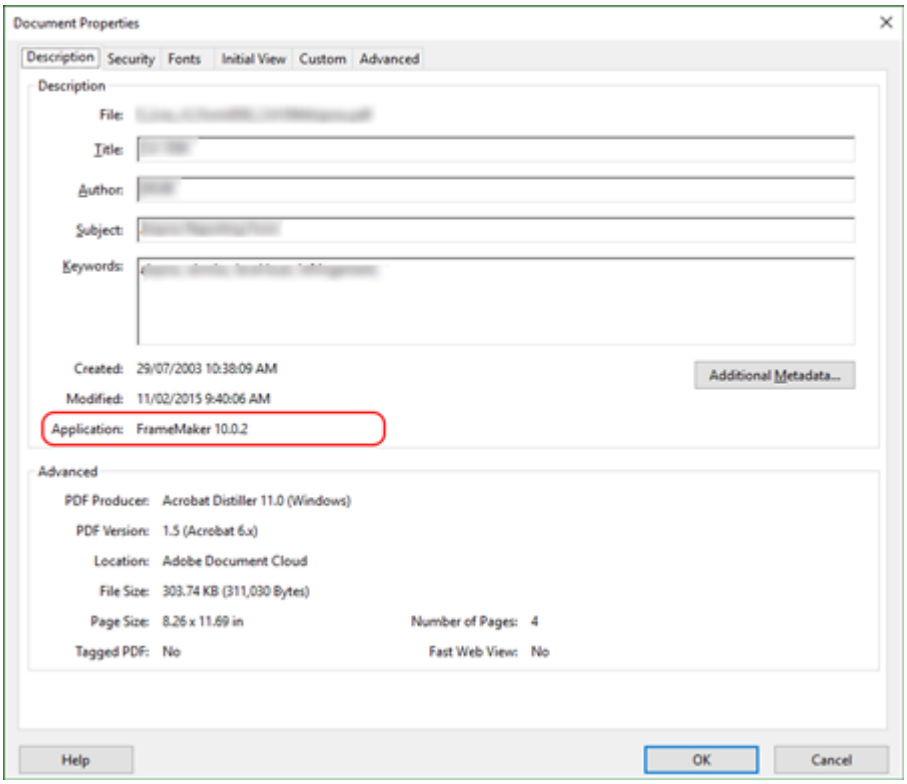


Possible Issues & workarounds

If you have any issues with the above steps, you may need to check what type of PDF you have. There are many types of PDF, some of them may cause some issues. For example:

- Secured PDFs cannot be manipulated in any way without a password.
- XFA PDFs may have a .pdf file extension, but can only be edited with certain Adobe products (Adobe LiveCycle, not covered in this guide)
- Some PDFs may already be converted to an AcroForm, in which case you have less work to do.

The best way to get around any issues is to print the form to PDF using a non-Adobe product such as PDFcreator, CutePDF or your browser's built-in PDF engine. You may need to try some different products to see what works best for you. You can check which application was used to produce it by opening it in Adobe Acrobat (or Reader), and selecting "File" -> "Properties".



Inspecting the file properties in Acrobat

Another way around PDF security or unsupported file types is to either physically print them, or take screenshots, and then use Acrobat's OCR capabilities to scan in the pages.

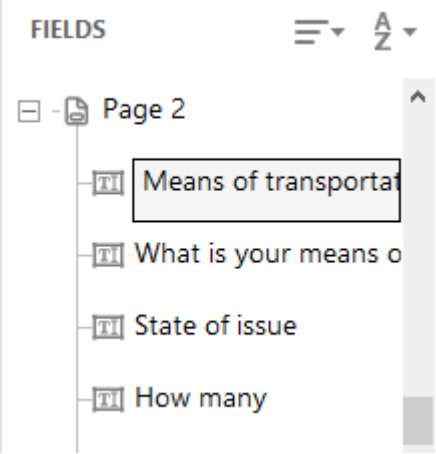
Step 3: Correcting automatic conversion issues

Automatically Detected Fields

For each automatically detected field, you will need to inspect to check:

- Field size, alignment & centring
- Font (Acrobat often chooses large and ugly fonts)
- Field names (it will become helpful to correct any auto-naming of fields to be something that is human readable)

The toolbar on the right will help you quickly rename and reorganise all the interactive objects on the page. You can also bulk edit by selecting more than one field and right-clicking to access the properties panel.

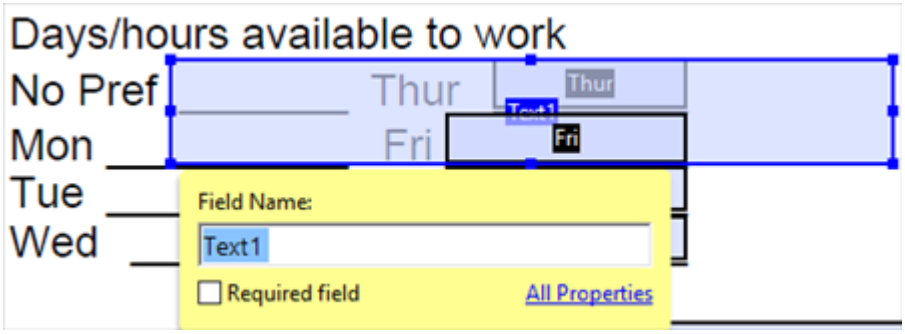


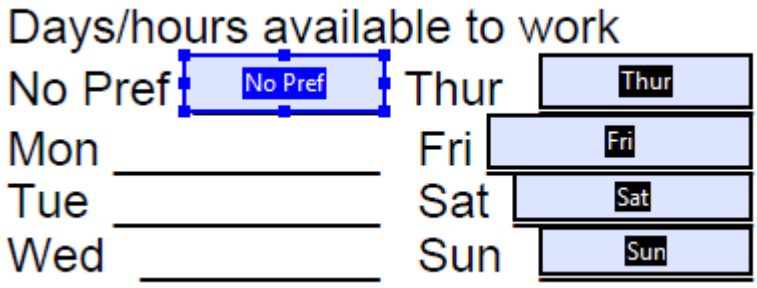
Acrobat Pro DC - Fields Toolbar

Missed Fields

Sometimes the Acrobat forms wizard misses fields, and you will be required to add them manually. In Acrobat DC, there's a toolbar on the top of the page to add fields, checkboxes and radio buttons. Dropdown lists should be added as plain text field.







Adding a missing text field

When you are done, please think about "global" properties such as fonts. You can select all the fields on the form, and edit them in bulk, to ensure consistency.

Dealing with character-separated fields (Dates / phone numbers with area code)

One of the issues that you will encounter is how to deal with character separated fields, such as dates: " / / ". You have two options on how to deal with this: *Override* the PDF design or *re-factor* the data in Composer.

The easiest option and the option with the best result is to add fields with an opaque (coloured) background that cover up the character separators in the original PDF design. This may not be permitted in all circumstances, as the business may mandate that no changes are made to the layout (EG: OCR in downstream processing may fail if you change the exact layout).

If you cannot override the field layout, then you will be required to add 3 separate text boxes for the date (Day, Month, Year), ensure that all values fit in the spaces created.

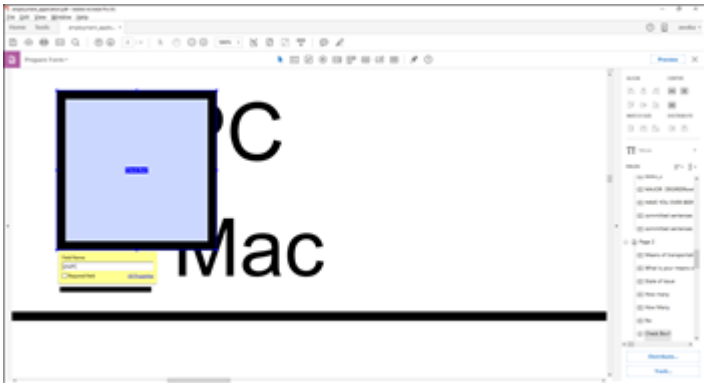
Re-factoring the format of field values

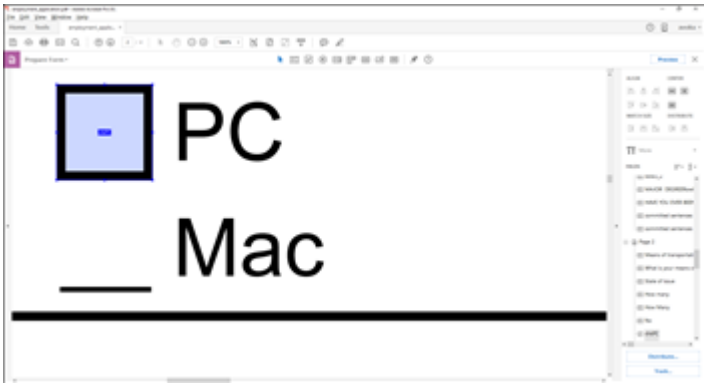
With some fields, you will be required to do some re-factoring of the field values. For example, Manager stores dates in a format of YYYY-MM-DD, and you will want to re-order this to be human readable, or to be displayed in 3 separate fields (pre-formatted character separators). This configuration is performed in Composer, and is described in the next step.

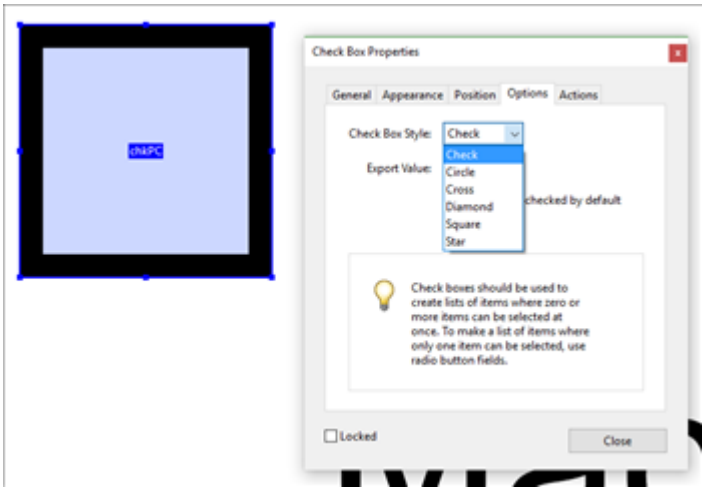
Check Boxes

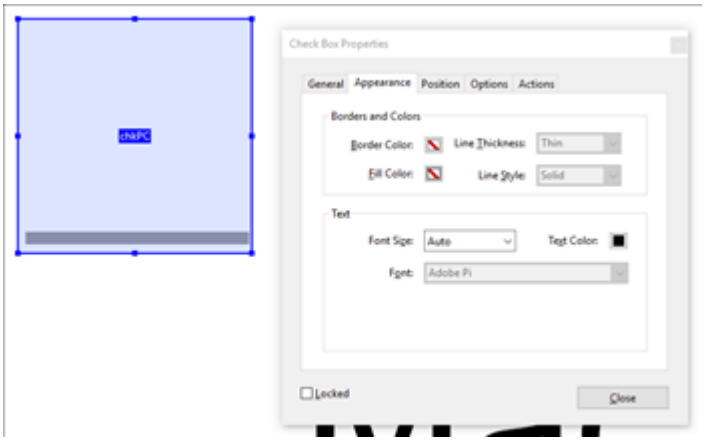
Check Boxes are quite straightforward. Simply add them, style them, and name them appropriately (Choose a convention which makes sense, such as prefixing the name with "chk"). You will need to resize them to fit in the correct area, and the best way to achieve a neat and accurate layout is to zoom right in to the part of the page where you're going to place it. Acrobat allows you to align, centre, match size and distribute objects for perfect layout. By default, Acrobat uses a check (tick) character within a box, but you can change this to a cross by editing the Check Box Properties, Options tab: Change the "Check Box Style" configuration (You may want to "Bulk edit" this setting, by selecting all your check boxes to edit). There are many layout options including border and fill colour.


PC
Mac


Mac


PC
Mac

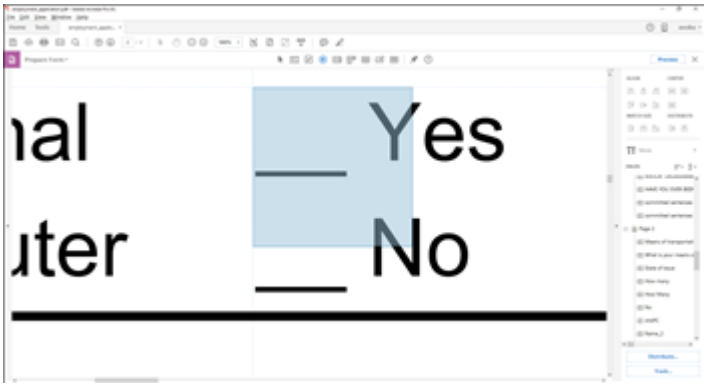

chkPC
Check Box Properties
General
Appearance
Position
Options
Actions
Check Box Style:
Check
Export Value:
Check
Circle
Cross
Diamond
Square
Star
checked by default
Check boxes should be used to create lists of items where zero or more items can be selected at once. To make a list of items where only one item can be selected, use radio button fields.
Locked
Close

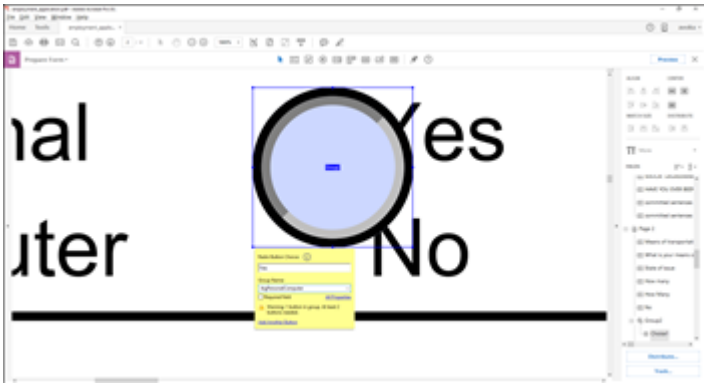


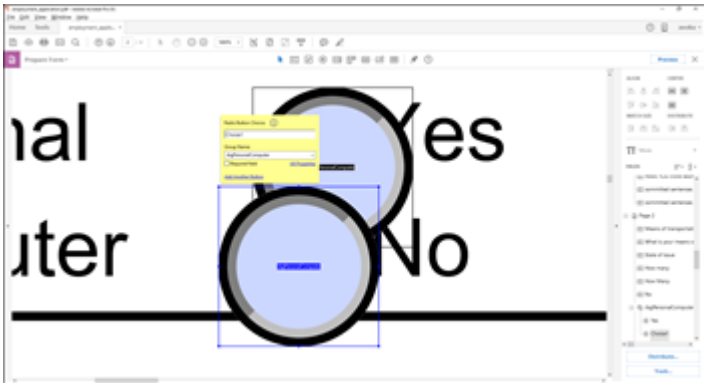
Adding Check Boxes to a form & configuring the appearance

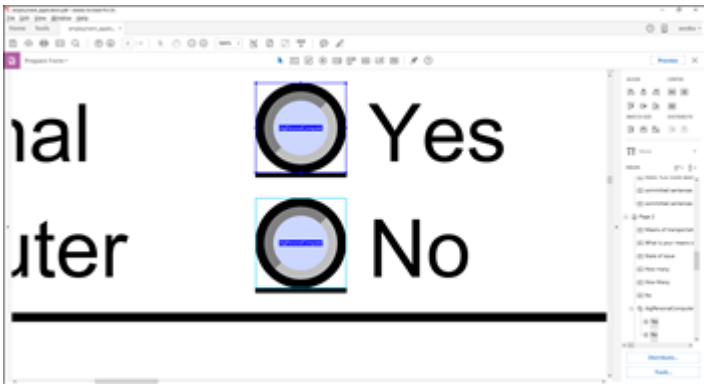
Radio Buttons

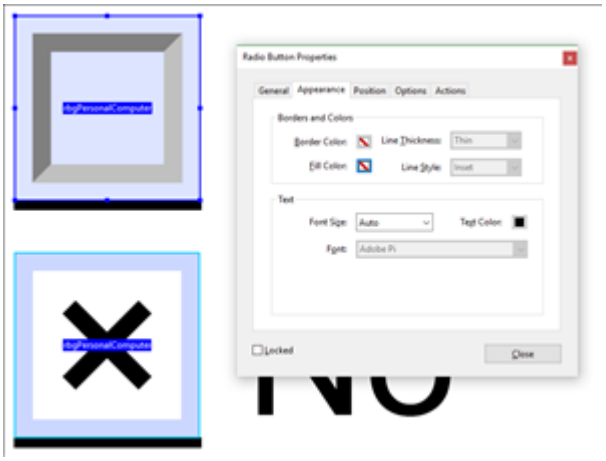
Radio buttons require membership in a radio button group. Acrobat attempts to help you with this by providing a wizard which helps you add multiple options in a list, but you will most likely need to re-factor the configuration to fit into a structure in the "Fields" editor. There is a limitation here: The name of the Radio Button Group should not contain spaces in order to be compatible with Composer.





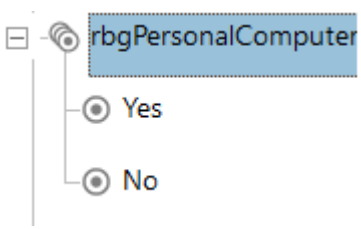






Adding a radio button group and configuring appearance

When you are done, your Fields tab should look something like this:

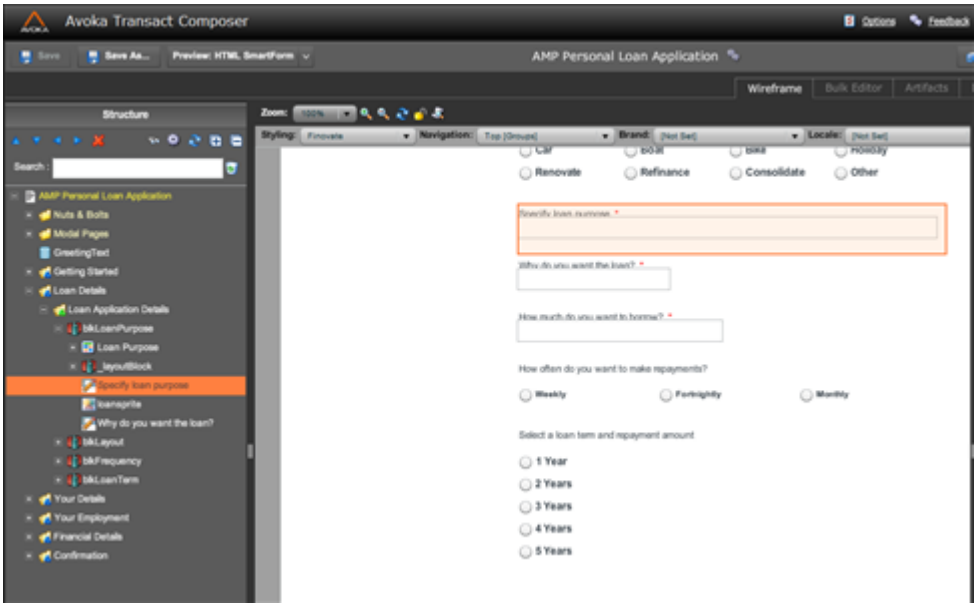


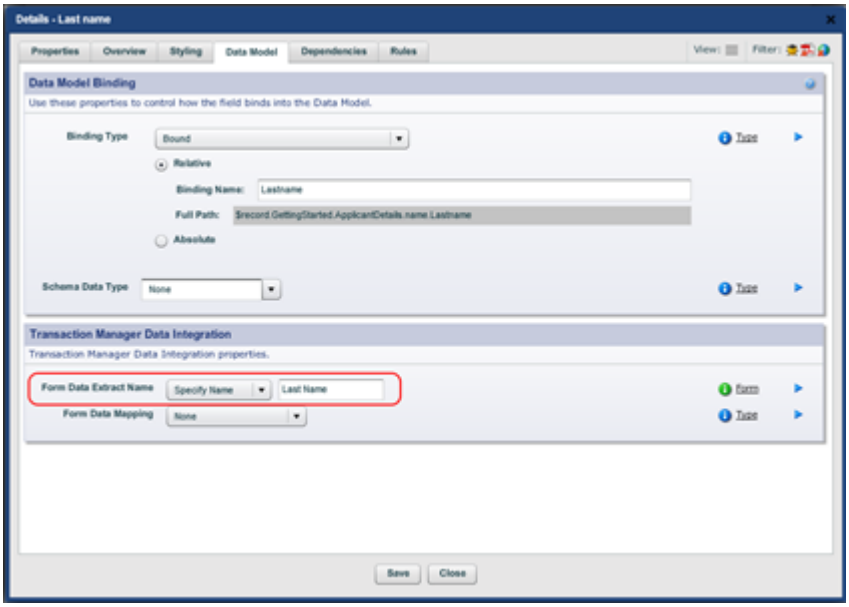
Step 4: Composer Configuration

In Composer, you will be required to perform two tasks:

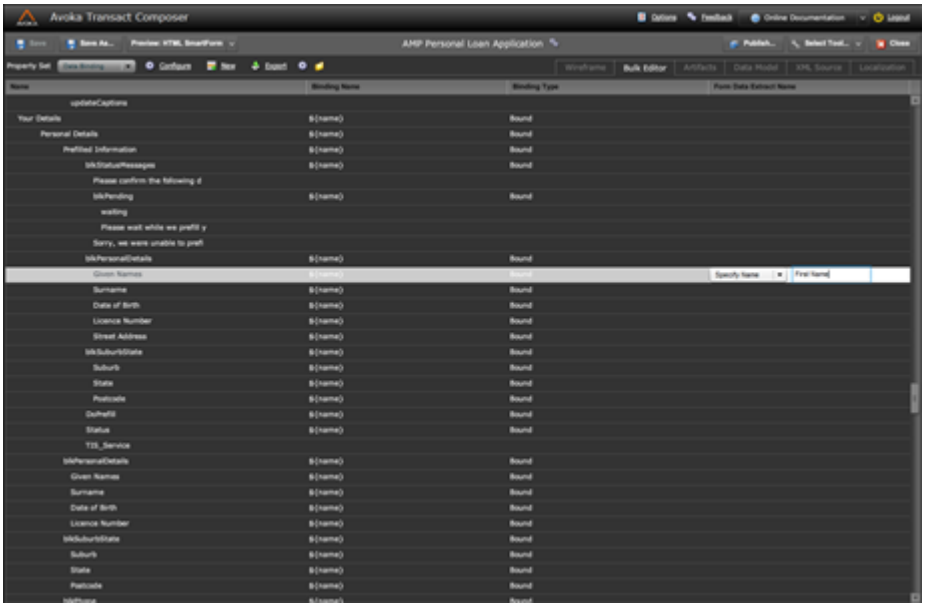
1. Bind Composer fields to PDF fields using Data Extracts
2. Re-factor any formatting of field values (such as dates)

To configure a data extract, you can either inspect the form wireframe/structure, or you can use the bulk editor, which displays the form fields in a table. The bulk editor is a great way of setting Data Extract properties in a tabular interface. To enable it, select the "Bulk Editor" tab, then change the property set to "Data Binding". Scroll down past all the internal fields (at least 10 screen pages), then click in an empty space for the drop-down list to appear.





Setting the Data Extract Name using the wireframe and property editor



Setting the Data Extract Name using the Bulk Editor

To configure data extracts from the wireframe/structure, double click on the field, then go to "Data Model", and set the "Form Data Extract Name".

Whichever process you choose, it is important to:

- Select "Specify Name" in the drop-down list
- Type in the name of the field. This name must match the name defined in the PDF form defined in Step 3 above.

Re-factoring the format of field values

With some fields, you will be required to do some re-factoring of the field values. For example, Manager stores dates in a format of YYYY-MM-DD, and you will want to re-order this to be human readable, or to be displayed in separate fields to be printed in pre-formatted fields with character separators. Follow these steps to ensure data is in the format expected by the PDF form.

1. Add a "Data Field" to your form, adjacent to the field in question. A data field is just an invisible field.
2. Add a scripted calculation rule to the data field.
3. Add some JavaScript to perform to parse and format the date or alternatively add string operations on the dependent field (See sample code below)
4. Add the Data Extract to the Data Field instead of the original user interface element

Reformatting using the Composer library

```
return sfc.parseDate({date},"DD-MM-YYYY").format("DD/MM/YYYY")
```

The above line of code encapsulates several operations in one line. If you are new to JavaScript, you may wish to analyse the sample code below, which achieves the same outcome using string operations

Example code (for beginners)

```
var date = {date};                  // Link to date field as a dependency
var dateArray = date.split("-");    // "YYYY-MM-DD"  ==>  Array [ "YYYY", "MM", "DD" ]

var day   = dateArray[2];           // The 3rd element in the array "DD"
var month = dateArray[1];           // The 2nd element in the array "MM"
var year  = dateArray[0];           // The 1st element in the array "YYYY"

return day + "/" + month + "/" + year;  // "DD/MM/YYYY" (adding strings will concatenate
them)
```

You may wish to extend this example with other types of fields. E.G.: you can use the same technique to re-factor phone numbers (use str.substring(0, 1); to get the first 2 digits in the string).

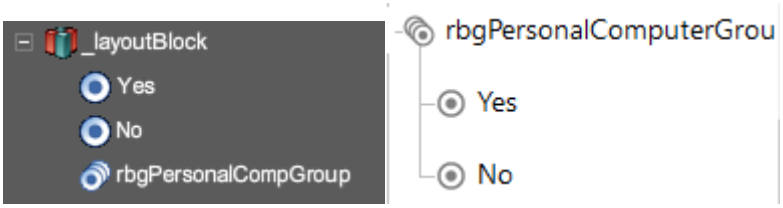
Radio Buttons & Check Boxes

Check boxes in Composer should be configured the same way as text fields: with the Data Extract name matching the AcroForm field name.

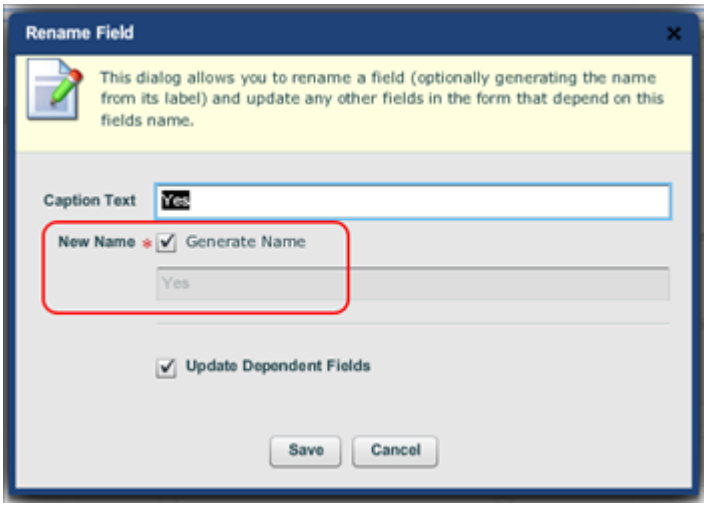
Radio buttons require a little extra care, ensuring that:

- The Composer Radio Button Group **data extract** matches the AcroForm Radio Button Group **name**
- Each Composer Radio Button **name** (not caption) matches the AcroForm Radio Button **names** exactly (check this for each radio button)

It's usually easier to perform modifications in Acrobat, conforming to Composer design.



Composer structure and Acrobat structure with matching names

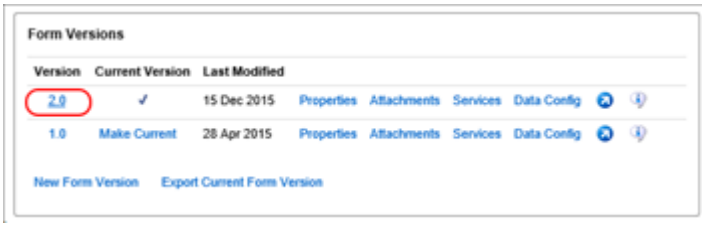


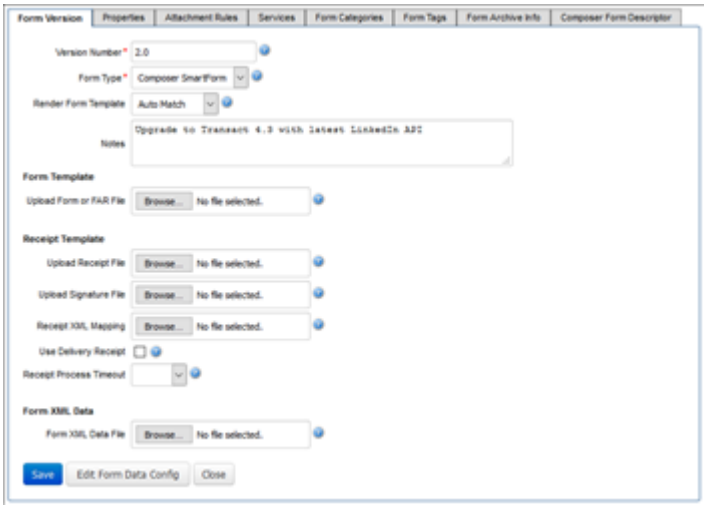
*Note that the **name** of the field does not necessarily match the **caption***

Step 5: Manager Configuration

When you have finished editing the AcroForm, it can be uploaded into Manager for use as a PDF Template for the receipt. First though, please ensure that you have published any changes out of Composer.

In the form dashboard, you can upload a receipt file using the form version dialog as shown below.



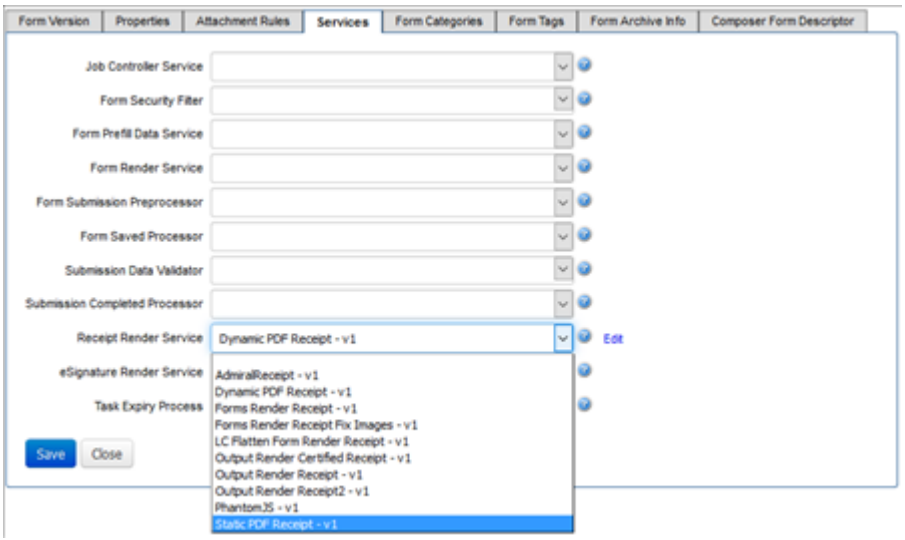


To import the file:

1. Log into Manager.
2. Under Forms, find the form you wish to update.
3. When you select the form you will see the Form Dashboard. On the top-right corner, you will see the Form Versions dashboard element (above)
4. Select the current version (highlighted in red above)
5. This will display the "Form Version" dialog. Under "Receipt Template", select the browse button adjacent to "Upload Receipt File".
6. Attach the AcroForm PDF file and save

Now that the file has been uploaded, it needs to be used by the receipt service:

1. On the form dashboard, in the form versions dashboard area, select the "Services" link adjacent to the latest version.
2. Change the Receipt Render Service to "Static PDF Receipt"



Extending this solution

A code-free solution can be great for rapid prototyping or building business-oriented solutions quickly, without relying on developers to contribute to the project. If you have access to Groovy developers, you can skip step #3, and and build server-side scripts to link form data Xpaths to AcroForm fields. Any data value re-factoring can then be performed in the custom script, rather than in Composer. Developers then have access to an extended suite of capabilities including:

- Concatenation of receipt with other static files (EG: PDS)
- Handling of attachments (as PDF attachments or pages)
- Enrichment of content (adding data from external services)

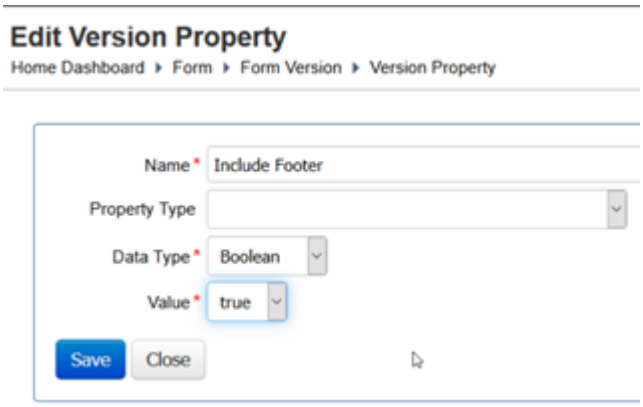
Next, learn how to configure receipts server node.

Insert a Watermark in a PDF Receipt

Journey Manager (JM) | System Manager / DevOps | v4.3 & Higher

You can creating a watermark on every rendered receipt by running a Groovy service, which will render the standard PDF receipt using iText to overlay HTML.

1. Publish your form to Manager, which will perform all the work, so no additional effort is needed in Composer.
2. Go to **Services > Service Definitions**
3. Import this Groovy Service.
4. Add two form version properties called Include Footer and Include Header i.e. Form dashboard > Properties as shown below:



5. Go to the **Form Version > Services** and include the **Receipt Render Service**.
6. Update the following Groovy code to change the position of your watermark:

```
def addHeader = addOverlayContent.curry(
 getTemplate("Header HTML Template"),
   [envName: tmEnv],
   245, 40, 600, 832
);
```

The coordinates above are: lower left X, lower right Y, upper right X, upper left Y

7. Render and submit your form to test the generated receipt or use the receipt test harness. If it is not working, check the system error log.



If you get this error, check that you have created the two form properties.

Next, learn how to configure receipts server node.

Regenerate a Receipt

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

You may need to re-generate a receipt after it has already been generated by Journey Manager. The easiest way to do it using the Fluent SDK code as shown below:

```
new TxnUpdater(param.txn)
  .setReceiptStatusReady()
  .update()
```

This works by setting the receipt status to **Ready**, which means the receipt generation background process will start and regenerate the receipt.

To actually get at the generated receipt, you can use the following code below:

```
//generate pdf receipt
byte[] receiptBytes = new ReceiptSvc()
  .setTxn(txn)
  .renderReceipt();
```

The receipt generation is a background process, and may take some time to execute. You can't assume that the receipt will be generated immediately.

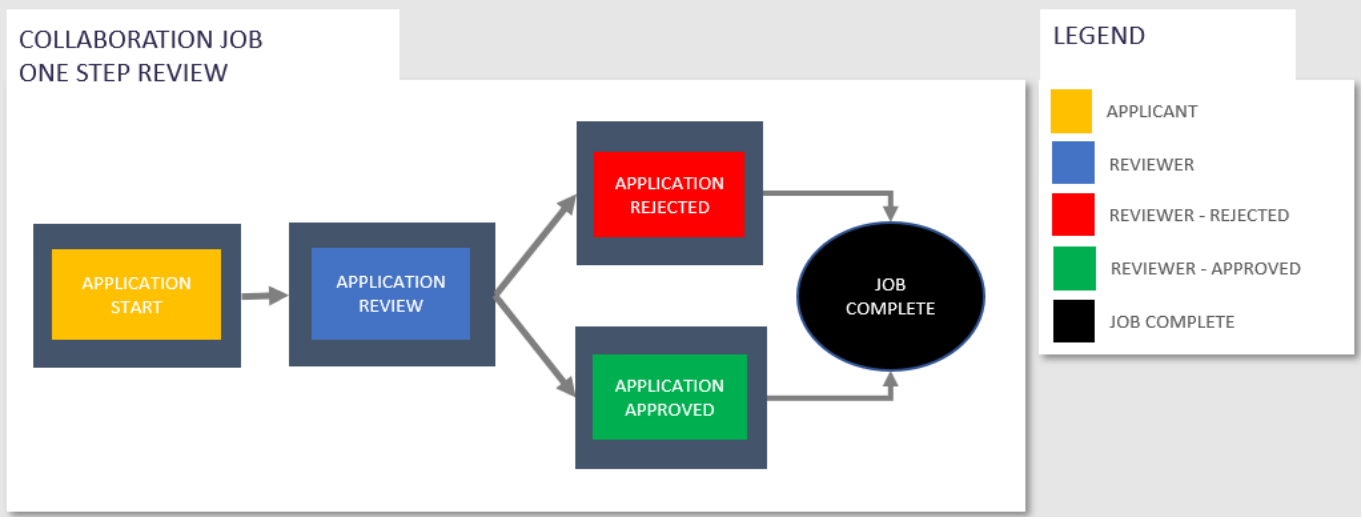
Next, learn how to configure receipts server node.

Collaboration Jobs Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

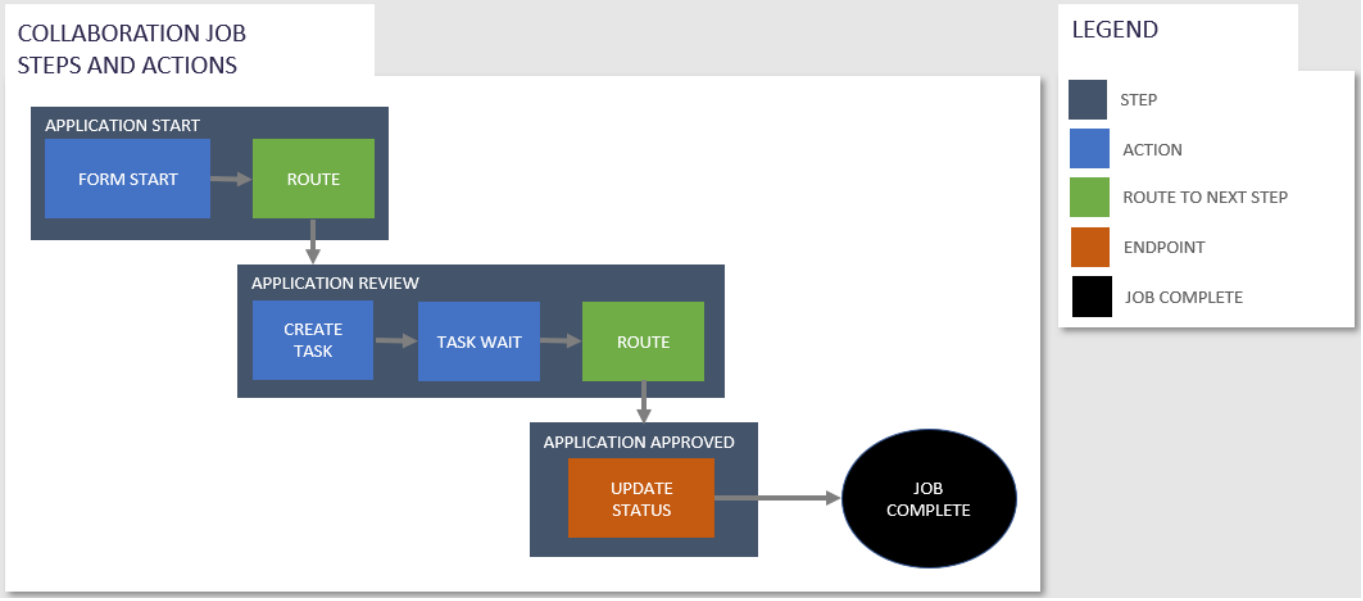
Journey Manager comes with the collaboration job framework for building complex and robust **workflow**[1]s without the need to use **Business Process Modeling**[2] tools. The framework is easy to use, so the collaboration jobs can be configured by non-developers.

A typical workflow can be an application review and approval process, which enable several people to work collaboratively on a user submitted form to either approve or reject it, as shown below.



A collaboration job controls a workflow of transactions that represent users interaction with a form. A collaboration job can be considered as a series of tasks that Manager creates for the users to complete, review or reject the form. A collaboration job consists of several steps with each step containing actions or endpoints. Steps are not ordered, but actions are sequential within each steps. The sequence of steps and actions defining a form approval workflow scenario is shown in the diagram below.

[1]A workflow is progression of steps (tasks, events, interactions) that comprise a work process, involve two or more persons, and create or add value to the organization's activities. In a sequential workflow, each step is dependent on occurrence of the previous step; in a parallel workflow, two or more steps can occur concurrently.
[2]Business Process Modeling Notation (BPMN) is a method of illustrating business processes in the form of a diagram similar to a flowchart.



The example above represents the **happy path**[1] scenario where the instance of a collaboration job is approved. The collaboration job is routed through three of the four steps shown on the step diagram above. An applicant fills and submits a form which starts the collaboration job. The first step being the Application Start, which has one action - the Form Start this links this initial submission by the applicant to the collaboration job. The collaboration job processing completes this action and immediately routes to the next step - the Application Review step - which has two actions - the Create Task, which creates a group task assigned to a review group, and the Task Wait which waits for the task to be submitted by a user. A user in the review group opens the task, review the submission, selects the **Approve** option to approve it, and submits this approval. This completes the Task Wait action which takes the Approve route to the next step - the Application Approved. This step is an endpoint which has one action, the Update Status, that notifies the applicant that there submission has been approved. Once all the actions have completed in an endpoint step, the collaboration job completes.

A collaboration job's workflow is specified in the job definition of a job controller service as a **JSON**[2] document. You can configure the job definition to change its execution flow. The usual way to initiate a collaboration job is to setup a form with the corresponding job controller service. When a form user submits the initial form, it will start a collaboration job, which in turn creates tasks and assigns them to various users, groups or email (anonymous users) in a review and approval flow. Alternatively, tasks can be assigned back to the original submitter to have them fill in new forms which is a form bundles flow.

A collaboration job can be started from an API call without requiring an initial form submission.

---

[1] A happy path is a default scenario featuring no exceptional or error conditions.

[2] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

A collaboration job has two types of execution mode. The standard mode, which is used for reviews and approvals, has the processing kicked off by a standard scheduled job called the Collaboration Job Controller - Scheduled Job, which runs every minute by default. The other mode is used for form bundles and has the job processing kicked of at the end of a submission processing. This is allows for collaboration jobs to run immediately and then present the a newly created form task back in the users browser.

Maestro forms have a number of custom widgets that interact with collaboration jobs. In addition, form fields can be mapped in Maestro to Data Extracts. Collaboration jobs use data extracts in preconditions to determine run a step or an action and also in the displaying of tasks details to the user. Finally, when a task is opened it contains job context information in the form XML, such as Job Ref Number, Current Step and Available routes. This populates widgets as well as controlling the visibility of form sections.

Manager allows you to track collaboration jobs, manage their details and check related transactions. You can also create custom job services.

Typically, there are two groups of Manager users that will work with collaboration jobs - developers and system administrators. You should configure and grant them the following permissions:

- Collaboration Job View
- Collaboration Job Edit
- Collaboration Job Remove

These permissions let developers to view collaboration jobs and enable the **Event** and **Error** tabs, which are useful for testing collaboration jobs. It allows them to search for the test job and drill down to review each step and action.

System administrators should be also able to move forms and collaboration jobs between environments, to maintain and assign users to the correct groups and roles. This will let them to:

- Search on a job and see what step it is up to and then drill down into it for more detail.
- Look at a action task and see who the task is assigned to.
- View form receipts, form XML data for completed submissions.
- Review error status.

Next, learn about collaboration job modes.

Collaboration Job Modes

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager has two processing modes for collaboration jobs: standard and process immediate. Depending on the mode, the form bundles, reviews and approvals workflows are processed differently and they also have different licensing implications. The processing modes are configured in a job definition of each collaboration job's job service.

Let's take a look at these processing modes.

Standard Mode

The standard mode is used to execute the review and approval collaboration jobs. When a user completes and submits a form, Manager starts the standard form submission processing, which calls a job and runs a callback that marks it as ready for processing. This collaboration job runs in a thread, with a delay up to 60 seconds, that executes the callback.

When selecting this mode, consider the following:

- Pausing a scheduler job stops the collaboration job controller from processing all collaboration jobs it has been running.
- Form submissions and scheduled jobs are processed in separate threads so user experience is not affected.
- There is a delay, up to 1 minute, from the time when the submission is completed and the time the submission processing is started. This delay can cause some issues for form bundles when tasks are immediately assigned back to the same user, or during review and approval demo in the non-production environment.
- Easier development and debugging process. If a scheduled job fails during execution of an action, you can change this action configuration and retry it until the action is completed.
- Supports automatic retry and recovery.
- Use it when there are many participants in the collaboration job.

Process Immediate Mode

The process immediate mode is used to execute form bundles, such as customer onboarding forms. For example, a new banking customer fills out an initial form which has some common fields and a list of products. They select the products and submit the form. After that, they are presented with a list of forms to complete, one for each product. A scheduled job runs as a user thread that performs the following:

- The standard submission processing.
- The callback to make the job ready for processing.
- The next part of the collaboration job instead of the collaboration job controller.

When selecting this mode, consider the following:

- Pausing a scheduled job stops the collaboration job controller from processing all collaboration jobs it has been running, but it doesn't stop processing of collaboration jobs running in the immediate mode, so they will be completed.
- Using form bundles where there is a single participant completing all tasks.
- Using collaboration jobs with form chaining, but with only one form per each step.
- Running a demo of review and approval collaboration jobs in non-production environments.
- Debugging is difficult, because a transaction is associated with the whole form submission, which is the user's thread. If an error occurs, steps, actions and collaboration job event logs are rolled back.
- The collaboration job processing adds time to the submission completion. The designer needs to be mindful of how long it takes to run the next part of the job to prevent long waiting times for the form user. Form bundles may take an extra 3 - 10 seconds, because they create form tasks, which is an important step before the tasks can be chained to the next available form.

Next, learn how to view collaboration jobs.

Job Services Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Job services are specialized services used in Journey Manager to create and process collaboration jobs.

Manager provides different job service types and comes with pre-configured job services and job service templates to help you create new job services.

Job services are defined per an organization. Each organization must have at least one job service configured. There is no limit on the number of job services an organization can have. Each form within the organization can use these job services. You can create new job controllers and job actions to implement your requirements.

Manager provides job service templates, which are enhanced job controller services. Templates allow a user to customize collaboration jobs without having to modify the job service definition. Templates are organization agnostic and you can't edit them.

Next, learn how to view job services.

Job Definitions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to define and configure a job services, which processes collaboration jobs, using a single JSON[1] document. The job definition includes the list of steps that make up the collaboration job. Each of the individual steps has the list of actions to be performed when the step runs, and the routes property, which defines the next step transition by mapping the route name to the next step.

The job definition consists of the following properties:

- jobDetails
- jobGroups
- steps
- actions
- properties
- routes

jobDetails

The **jobDetails** property defines a collaboration job and how it is processed. The snippet of the **jobDetails** is shown below:

```
{
	"jobDetails": {
		"name": "2 Step Review Job",
		"processSubmitImmediate": "true",
		"version": "4.1.0"
	},
	"steps":[
		{ .. step definition ...},
		{ .. step definition ...},
		...
	]
}
```

The **jobDetails** has the following attributes:

Attribute Name	Description

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

name	The name of the collaboration job, which is set by the job controller service when the collaboration job has been saved.
processSubmitImmediate	The collaboration job processing mode. If this attribute is omitted or set to false, the job will use the standard mode. Otherwise, it will use the process immediate mode.
version	The version of Manager that the job was created in.

jobGroups

The **jobGroups** property lists collaboration job groups, so job coordinators can manage the collaboration job. The snippet of the **jobGroups** is shown below:

```
"jobGroups": {}
```

The **jobGroups** has the following attributes:

Attribute Name	Description
name	The name of the job group, which will be displayed in the Job Groups dropdown from the Reviews tab of the form space.
description	The description for the job group.

The job groups can be created in one of the following ways:

1. If a job group defined in a job definition, as in above, has not been created, then the job group will be automatically created the first time a user submits a form associated to this job. This is the preferred way to create job groups.
2. Job groups can be manually created. By manually creating the job group, you must ensure that the job group names and descriptions match with the job definition. Any mismatch will not be subsequently updated to the job groups.

steps

The **steps** property is an JSON array that defines steps. The snippet of the **steps** property is shown below:

```
{
      "name": "Initial Review",
      "type": "",
      "expiryRule": "+5d"
      "expiryServiceName": "An Expiry Service"
      "actions": [
        { ... Action 1 Definition ... },
        { ... Action 2 Definition ... }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" },
        { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "display":
"false" },
        { "name": "Terminate",  "nextStep": "Terminated Initial" }
      ]
},
```

The **steps** has the following attributes:

Attribute Name	Description
name	The name of the step, which is unique within a collaboration job.
type	The type of the step, that can be: • start - the first step for the collaboration job. "type":"start" • endpoint - the processing stops and the job completes. Endpoints do not specify actions, routes or customProperties. "type":"endpoint" • empty string. "type":""
expiryRule	The step scheduled completion time, the step expiry, calculation rule. If specified, the job_step.time_completion_scheduled is set as the step creation time plus the expiry rule value, for example, 2014-03-05 + 5d = 2014-03-10. The valid values are: • "expiryRule" : "+4h" (hours) • "expiryRule" : "+5d" (days) • "expiryRule" : "+4w (weeks) • "expiryRule" : "+4m". This is an optional attribute.

	We recommend setting it up to some default value so an action doesn't stay in the queue forever, if an associated form never abandons and the task is never submitted, even as a transaction expires.
expiryServiceName	Specifies a job expiry service to clean up any items, such as incomplete submissions. This service should specify a route. A route name doesn't have to include a work expiry.

If the **expiryRule** is used, but the **exprityServiceName** is not specified, then expiry route must be provided.

actions

The **actions** property defines the list of actions for each step, which are executed sequentially. Actions are required for all steps except endpoints. The snippet of the **actions** property is shown below:

```
{
    "name": "Create Task",
    "type": "Job Task Assign",
    "properties": [
        { "name": "Process Message Send Email",  "value": "true" },
        { "name": "Process Message Submission Step",  "value": "Start" },
        { "name": "Process Message Text",  "value": "Your ${submission.formName} applic-
ation is at the Initial Review Step" },
}
```

The **actions** has the following attributes:

Attribute Name	Description
name	The name of the action, which is unique within a collaboration job.
type	The action service type, that must be: • listed above in the Standard Actions Services. • groovy action services can have a type of Job Action.
properties	The action specific properties, see the **properties** object definition below. This is an optional attribute.

The standard Action Property classes (**com.avoka.fc.core.service.job.impl**) define a fixed set of Action Properties. The **JobTaskAssignService** defines a broad range of separate properties, so you can add them to your script when creating a new job action. This makes a job action re-useable across a collaboration job and the whole form application. For example, if the job definition has this property:

```
"properties": [
   { "name": "ABC Widget Name",  "value": "Aplle Blue Catapillar" },
   ...
 ]
```

Then, you can retrieve it using the Groovy code below:

```
def abcName = actionStepProperties.getProperty("ABC Widget Name")
```

However, you should consider the following:

- There is no submission associated with the job actions, by default.
- **$formDataMap.name** doesn't resolve as the job action doesn't have any concept of a submission.
- **$func.previousSubmission().formDataMap.name** works as we have used the submission from the previous step.
- **$job.submission(0).name** works as we have used the first submission linked to the collaboration job.
- This is not supported in Fluent job action classes version 17.10.1.

You can use various job functions in a job definition as well.

properties

The **properties** property defines the job definition properties. The snippet of the **properties** property is shown below:

```
"properties": [
    { "name": "Task Assign Group",  "value": "$func.formProperty('Initial Review')" },
    { "name": "Task Type",  "value": "Review" },
]
```

Properties can be used to pass information into an instance of the Action Service. In this case, they are called action properties and can be defined at the action, step or collaboration job level. Properties with the same name defined at an action level override properties defined at the step level. Step override properties of the same name at the job level. This is illustrated in the snippet below:

```
{
  "jobDetails": { ... },
  "properties": [
    { "name": "Task Form Code", "value": "$func.startFormCode()" }
  ],
  "steps":[
    {
      "name": "Initial Review",
      "type": "",
      "actions": [
        {
          "name": "Create Task",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Process Message Text",  "value": "Your ${submission.formName}
application is at the Initial Review Step" },
            { "name": "Task Assign Group",  "value": "$func.formProperty('Initial
Review')" },
            { "name": "Task Message",  "value": "Please perform the initial review of the
${submission.formName} from ${formDataMap.firstName} ${formDataMap.lastName}." },
            { "name": "Task Review Previous Step",  "value": "true" },
            { "name": "Task Subject",  "value": "Initial review of ${submission.formName}
from ${formDataMap.firstName} ${formDataMap.lastName}" },
            { "name": "Task Type",  "value": "Review" },
          ]
        },
        { ... }
      ],
      "properties": [
        { "name": "Task Form Code", "value": "$func.startFormCode()" }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" }
      ],
    },
    { ... }
  ]
}
```

The **properties** has the following attributes:

Attribute Name	Description

name	The name of the property. The predefined action services, such as the job task assign service, have a set number of action properties
value	A property value that can include an Apache Velocity template. The action properties are templates that are evaluated when an action service is run. The templates can be: • static text values, for example, **`{ "name": "Task Email Message", "value": "An Email Message" }`** • substitute values from the form data extract, for example, **`{ "name": "Process Message Text", "value": "Your ${job.submission.formName} application is at the Initial Review Step" }`**
template	It evaluates the value as a velocity template and forces the return value to return a String. This can evaluate nested function calls. This is can be used with property values that start with "$func", these can return an object. For example, **`$func.startUser()`** will return a user. **`{ "name": "Task Email Message", "value": "$func.getSubmission().formXml()", "template": "true" }`**

routes

The **routes** property defines the route mapping between the route name and the next step, which are required for all steps except endpoints. An action can return a route result name, as a string, which is listed in each step. If a valid name is returned, the step routes the collaboration job to the next step. However, if the action returns an invalid route name, an error is thrown. The snippet of the **routes** property is shown below:

```
"routes": [
   { "name": "Approve",  "nextStep": "Application Delivery" },
   { "name": "Decline",  "nextStep": "Applicant Update" },
   { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "display": "false"
},
   { "name": "Terminate",  "nextStep": "Terminated Initial" }
]
```

The **routes** has the following attributes:

Attribute Name	Description
name	The route name that can be any name except the following reserved names:

	• Default - If there is no step found with a route name or if the route name is empty, the default route is used. • Expiry - the route taken when the step has expired. The next step, that can be:
nextStep	• A valid step name contained in the job definition. • A special case is ##PREVIOUS_STEP the job processing assign the next step as the previous step name.
display	The collaboration job provides a list of available routes to the form. However, the **`"display": "false"`** removes the route from the available routes. This is an optional attribute.

You can see the complete example of a job definition here.

Next, learn how to configure job definition.

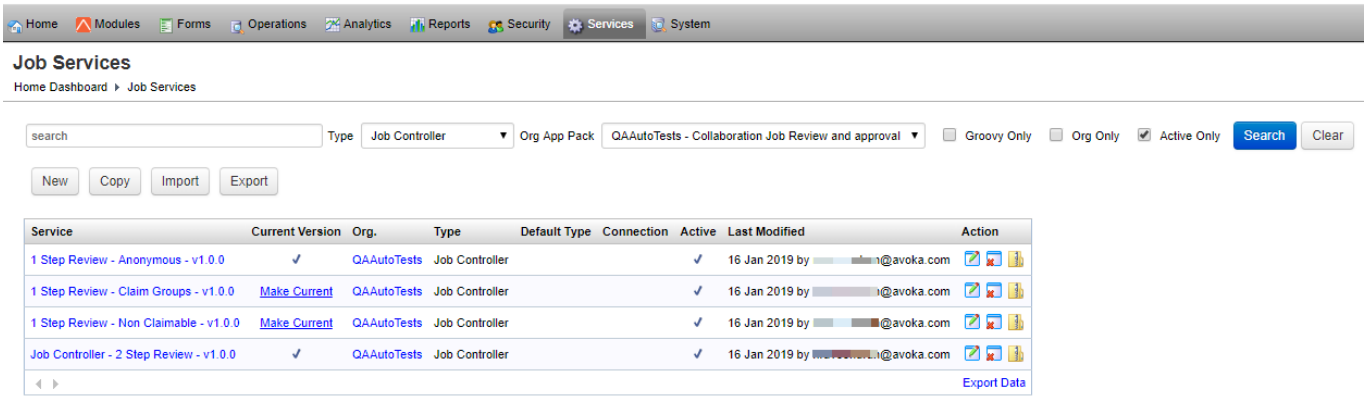
View Job Services

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view all job services, so you can easily find an existing job service.

To view job services, select **Services > Job Services.**

Job services are shown page by page, with the maximum number of records per page configured in the users preferences.



To filter or search the services by one or more criteria, specify the following settings and click **Search**:

- Service name or part of a name
- Service type
- Org App Pack
- Groovy Only to see only **Groovy**[1] services
- Org Only
- Active Only

Click **Clear** to reset the search criteria.

You can check the following service details:

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

- Service name
- Current Version
- Organization
- Type
- Default Type represents whether this service is the default service for this type of service.
- Connection is the service connection associated with this service, if it is configured.
- Active shows whether this service is active or not. It must be active to be available.

> It is recommended to make a service not active rather then remove it when you don't need it any more.

- Last Modified shows the date when this service was last modified and by whom.

You can also perform the following actions:

- Create a service
- View and configure a service
- Copy a service
- Remove a service
- Export services
- Import services

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Configure Job Definition

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to define how a job service processes collaboration jobs by describing an execution flow of steps and actions in the job definition.

To configure a job definition of a service:

1. Locate an existing job service.
2. Select the **Job Definition** tab to review or update the JSON[1] file defining this service.



3. Select the **Validate** checkbox to validate action services, step properties, form codes and custom services defined in the JSON file. It is selected by default, but you can clear it, if you want to add a new action but you haven't created a form or a custom service for this, and you still need to save your changes.

---

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

4. Select the **Auto Format** checkbox to automatically format the job definition and strip unknown elements.
5. Click **Save** to update the changes..

Next, learn how to view all job services of a job controller.

Job Definitions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to define and configure a job services, which processes collaboration jobs, using a single JSON[1] document. The job definition includes the list of steps that make up the collaboration job. Each of the individual steps has the list of actions to be performed when the step runs, and the routes property, which defines the next step transition by mapping the route name to the next step.

The job definition consists of the following properties:

- jobDetails
- jobGroups
- steps
- actions
- properties
- routes

jobDetails

The **jobDetails** property defines a collaboration job and how it is processed. The snippet of the **jobDetails** is shown below:

```
{
        "jobDetails": {
                "name": "2 Step Review Job",
                "processSubmitImmediate": "true",
                "version": "4.1.0"
        },
        "steps":[
                { .. step definition ...},
                { .. step definition ...},
                ...
        ]
}
```

The **jobDetails** has the following attributes:

Attribute Name	Description

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

name	The name of the collaboration job, which is set by the job controller service when the collaboration job has been saved.
processSubmitImmediate	The collaboration job processing mode. If this attribute is omitted or set to false, the job will use the standard mode. Otherwise, it will use the process immediate mode.
version	The version of Manager that the job was created in.

jobGroups

The **jobGroups** property lists collaboration job groups, so job coordinators can manage the collaboration job. The snippet of the **jobGroups** is shown below:

```
"jobGroups": {}
```

The **jobGroups** has the following attributes:

Attribute Name	Description
name	The name of the job group, which will be displayed in the Job Groups dropdown from the Reviews tab of the form space.
description	The description for the job group.

The job groups can be created in one of the following ways:

1. If a job group defined in a job definition, as in above, has not been created, then the job group will be automatically created the first time a user submits a form associated to this job. This is the preferred way to create job groups.
2. Job groups can be manually created. By manually creating the job group, you must ensure that the job group names and descriptions match with the job definition. Any mismatch will not be subsequently updated to the job groups.

steps

The **steps** property is an JSON array that defines steps. The snippet of the **steps** property is shown below:

```
{
      "name": "Initial Review",
      "type": "",
      "expiryRule": "+5d"
      "expiryServiceName": "An Expiry Service"
      "actions": [
        { ... Action 1 Definition ... },
        { ... Action 2 Definition ... }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" },
        { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "display":
"false" },
        { "name": "Terminate",  "nextStep": "Terminated Initial" }
      ]
},
```

The **steps** has the following attributes:

Attribute Name	Description
name	The name of the step, which is unique within a collaboration job.
type	The type of the step, that can be: • start - the first step for the collaboration job. "type":"start" • endpoint - the processing stops and the job completes. Endpoints do not specify actions, routes or customProperties. "type":"endpoint" • empty string. "type":""
expiryRule	The step scheduled completion time, the step expiry, calculation rule. If specified, the job_step.time_completion_scheduled is set as the step creation time plus the expiry rule value, for example, 2014-03-05 + 5d = 2014-03-10. The valid values are: • "expiryRule" : "+4h" (hours) • "expiryRule" : "+5d" (days) • "expiryRule" : "+4w (weeks) • "expiryRule" : "+4m". This is an optional attribute.

	We recommend setting it up to some default value so an action doesn't stay in the queue forever, if an associated form never abandons and the task is never submitted, even as a transaction expires.
expiryServiceName	Specifies a job expiry service to clean up any items, such as incomplete submissions. This service should specify a route. A route name doesn't have to include a work expiry.

If the **expiryRule** is used, but the **exprityServiceName** is not specified, then expiry route must be provided.

actions

The **actions** property defines the list of actions for each step, which are executed sequentially. Actions are required for all steps except endpoints. The snippet of the **actions** property is shown below:

```
{
    "name": "Create Task",
    "type": "Job Task Assign",
    "properties": [
        { "name": "Process Message Send Email",  "value": "true" },
        { "name": "Process Message Submission Step",  "value": "Start" },
        { "name": "Process Message Text",  "value": "Your ${submission.formName} applic-
ation is at the Initial Review Step" },
}
```

The **actions** has the following attributes:

Attribute Name	Description
name	The name of the action, which is unique within a collaboration job.
type	The action service type, that must be: • listed above in the Standard Actions Services. • groovy action services can have a type of Job Action.
properties	The action specific properties, see the **properties** object definition below. This is an optional attribute.

The standard Action Property classes (**com.avoka.fc.core.service.job.impl**) define a fixed set of Action Properties. The **JobTaskAssignService** defines a broad range of separate properties, so you can add them to your script when creating a new job action. This makes a job action re-useable across a collaboration job and the whole form application. For example, if the job definition has this property:

```
"properties": [
   { "name": "ABC Widget Name",  "value": "Aplle Blue Catapillar" },
   ...
 ]
```

Then, you can retrieve it using the Groovy code below:

```
def abcName = actionStepProperties.getProperty("ABC Widget Name")
```

However, you should consider the following:

- There is no submission associated with the job actions, by default.
- **$formDataMap.name** doesn't resolve as the job action doesn't have any concept of a submission.
- **$func.previousSubmission().formDataMap.name** works as we have used the submission from the previous step.
- **$job.submission(0).name** works as we have used the first submission linked to the collaboration job.
- This is not supported in Fluent job action classes version 17.10.1.

You can use various job functions in a job definition as well.

properties

The **properties** property defines the job definition properties. The snippet of the **properties** property is shown below:

```
"properties": [
    { "name": "Task Assign Group",  "value": "$func.formProperty('Initial Review')" },
    { "name": "Task Type",  "value": "Review" },
]
```

Properties can be used to pass information into an instance of the Action Service. In this case, they are called action properties and can be defined at the action, step or collaboration job level. Properties with the same name defined at an action level override properties defined at the step level. Step override properties of the same name at the job level. This is illustrated in the snippet below:

```
{
  "jobDetails": { ... },
  "properties": [
    { "name": "Task Form Code", "value": "$func.startFormCode()" }
  ],
  "steps":[
    {
      "name": "Initial Review",
      "type": "",
      "actions": [
        {
          "name": "Create Task",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Process Message Text",  "value": "Your ${submission.formName}
application is at the Initial Review Step" },
            { "name": "Task Assign Group",  "value": "$func.formProperty('Initial
Review')" },
            { "name": "Task Message",  "value": "Please perform the initial review of the
${submission.formName} from ${formDataMap.firstName} ${formDataMap.lastName}." },
            { "name": "Task Review Previous Step",  "value": "true" },
            { "name": "Task Subject",  "value": "Initial review of ${submission.formName}
from ${formDataMap.firstName} ${formDataMap.lastName}" },
            { "name": "Task Type",  "value": "Review" },
          ]
        },
        { ... }
      ],
      "properties": [
        { "name": "Task Form Code", "value": "$func.startFormCode()" }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" }
      ],
    },
    { ... }
  ]
}
```

The **properties** has the following attributes:

Attribute Name	Description

name	The name of the property. The predefined action services, such as the job task assign service, have a set number of action properties
value	A property value that can include an Apache Velocity template. The action properties are templates that are evaluated when an action service is run. The templates can be: • static text values, for example, **`{ "name": "Task Email Message", "value": "An Email Message" }`** • substitute values from the form data extract, for example, **`{ "name": "Process Message Text", "value": "Your ${job.submission.formName} application is at the Initial Review Step" }`**
template	It evaluates the value as a velocity template and forces the return value to return a String. This can evaluate nested function calls. This is can be used with property values that start with "$func", these can return an object. For example, **`$func.startUser()`** will return a user. **`{ "name": "Task Email Message", "value": "$func.getSubmission().formXml()", "template": "true" }`**

routes

The **routes** property defines the route mapping between the route name and the next step, which are required for all steps except endpoints. An action can return a route result name, as a string, which is listed in each step. If a valid name is returned, the step routes the collaboration job to the next step. However, if the action returns an invalid route name, an error is thrown. The snippet of the **routes** property is shown below:

```
"routes": [
   { "name": "Approve",  "nextStep": "Application Delivery" },
   { "name": "Decline",  "nextStep": "Applicant Update" },
   { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "display": "false"
},
   { "name": "Terminate",  "nextStep": "Terminated Initial" }
]
```

The **routes** has the following attributes:

Attribute Name	Description
name	The route name that can be any name except the following reserved names:

	• Default - If there is no step found with a route name or if the route name is empty, the default route is used. • Expiry - the route taken when the step has expired. The next step, that can be:
nextStep	• A valid step name contained in the job definition. • A special case is ##PREVIOUS_STEP the job processing assign the next step as the previous step name.
display	The collaboration job provides a list of available routes to the form. However, the **`"display": "false"`** removes the route from the available routes. This is an optional attribute.

You can see the complete example of a job definition here.

Next, learn how to configure job definition.

View Job Info

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides you with a read only view of job info of a selected job service.

To check job info:

1. Locate a job service.
2. Select the **Job Info** tab to view job info of this service that includes:
   - Job name
   - Job steps
   - Job Form Data Maps, which lists the names of the parameters used in the form XML data.
3. Click **Close**.

Next, learn how to configure service parameters.

Job Controller Service

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager uses job controller services to:

- Create a collaboration job instance when a user submits a form or the API is called.
- Process collaboration jobs by executing their steps and actions defined in a job definition.

Manager allows you to create and configure job controller services to implement your customer requirement.

To create a job controller service:

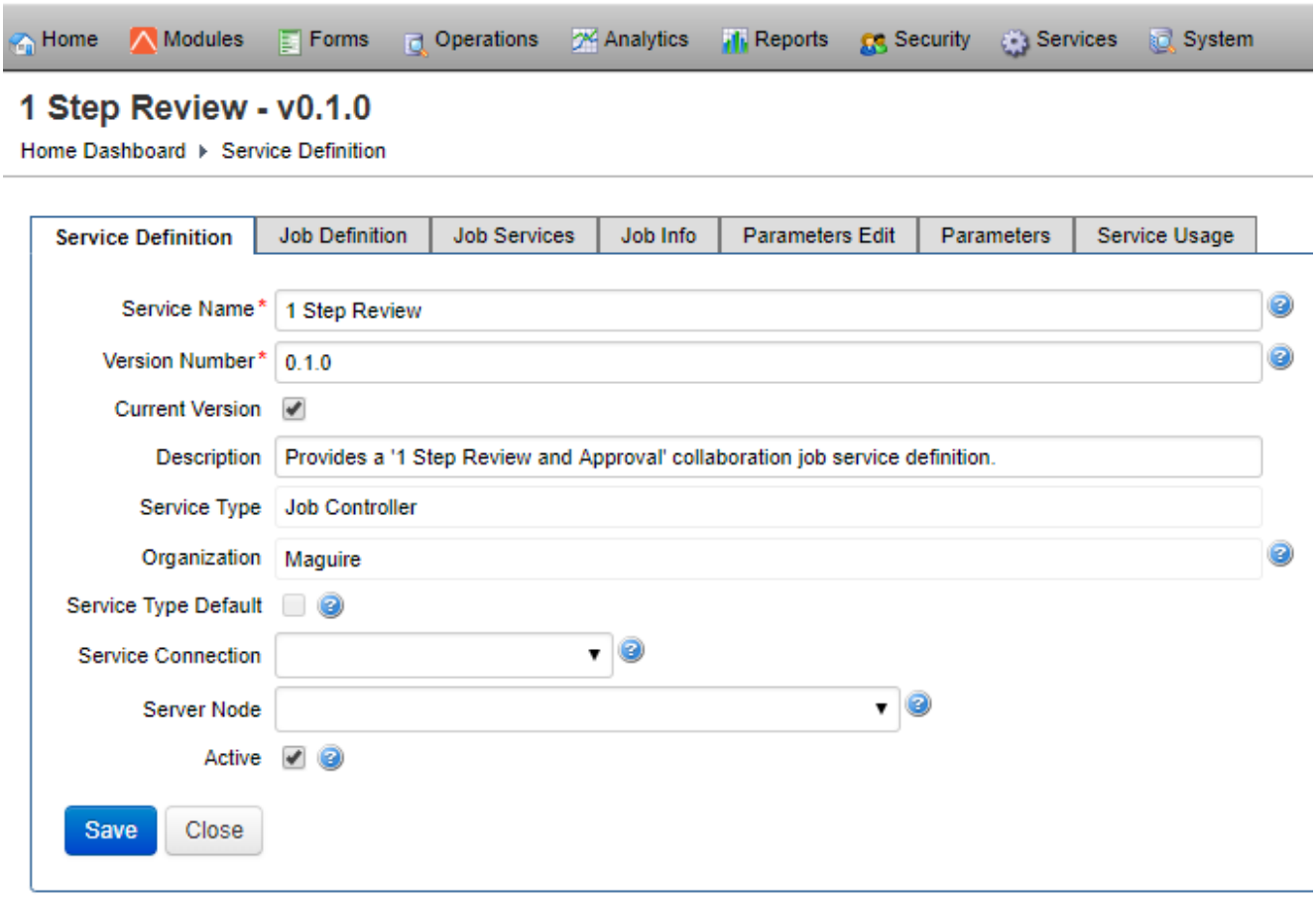
1. Select **Services > Job Services**.
2. Click **New** to create a job service.
3. Select the **Job Controller** as a service type from the **Service Type** dropdown list.
4. Select a service template from the **Service Template** dropdown list, depending on your workflow, for example, **Job Controller - 1 Step Review**.
5. Select an organization from the **Organization** dropdown list.

   You have to assign a job controller to an organization. If you have several organizations that would like to use the same job controller, you need to define one for each organization.

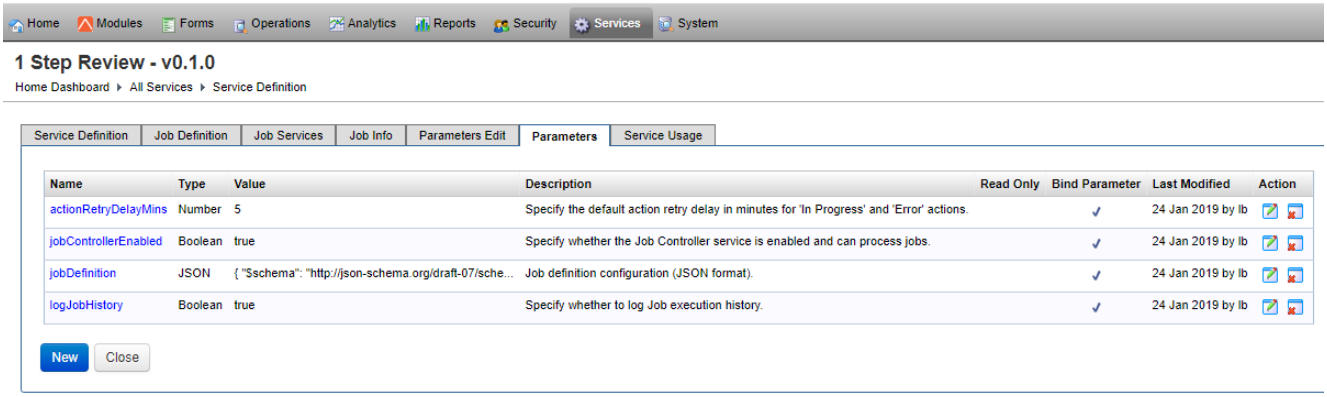
6. Click **Save** to update the changes.

To configure a job controller service:

1. Locate a job service.
2. Configure the service definition.



3. Configure the job definition.
4. Check the job services used in this job controller.
5. Check the job info.
6. Create service parameters required for this job controller to run.



The standard service parameters are:

- **actionRetryDelayMins** - specifies the default action retry delay in minutes for 'In Progress' and 'Error' actions. Default is 5. This is a bind parameter.
- **jobControllerEnabled** - specifies whether the job controller service is enabled and it can process collaboration jobs. This is useful for debugging and test in non-production, and selectively enabling certain collaboration jobs in production environments. Default is true. This is a bind parameter.
- **jobDefinition** - job definition configuration in **JSON**[1] format. This is a bind parameter.
- **logJobHistory** - specifies whether to write collaboration job execution history in a log file or not. Default is true. This is a bind parameter.

  The optional service parameters are:

- **jobDiagram** - contains the collaboration job diagram format created by the external NextStep tool.
- **Form Property Editor Definition** - the job form properties editor definition in JSON format.

7. Check the job controller service usage.

Next, learn how to configure a job controller in a form.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

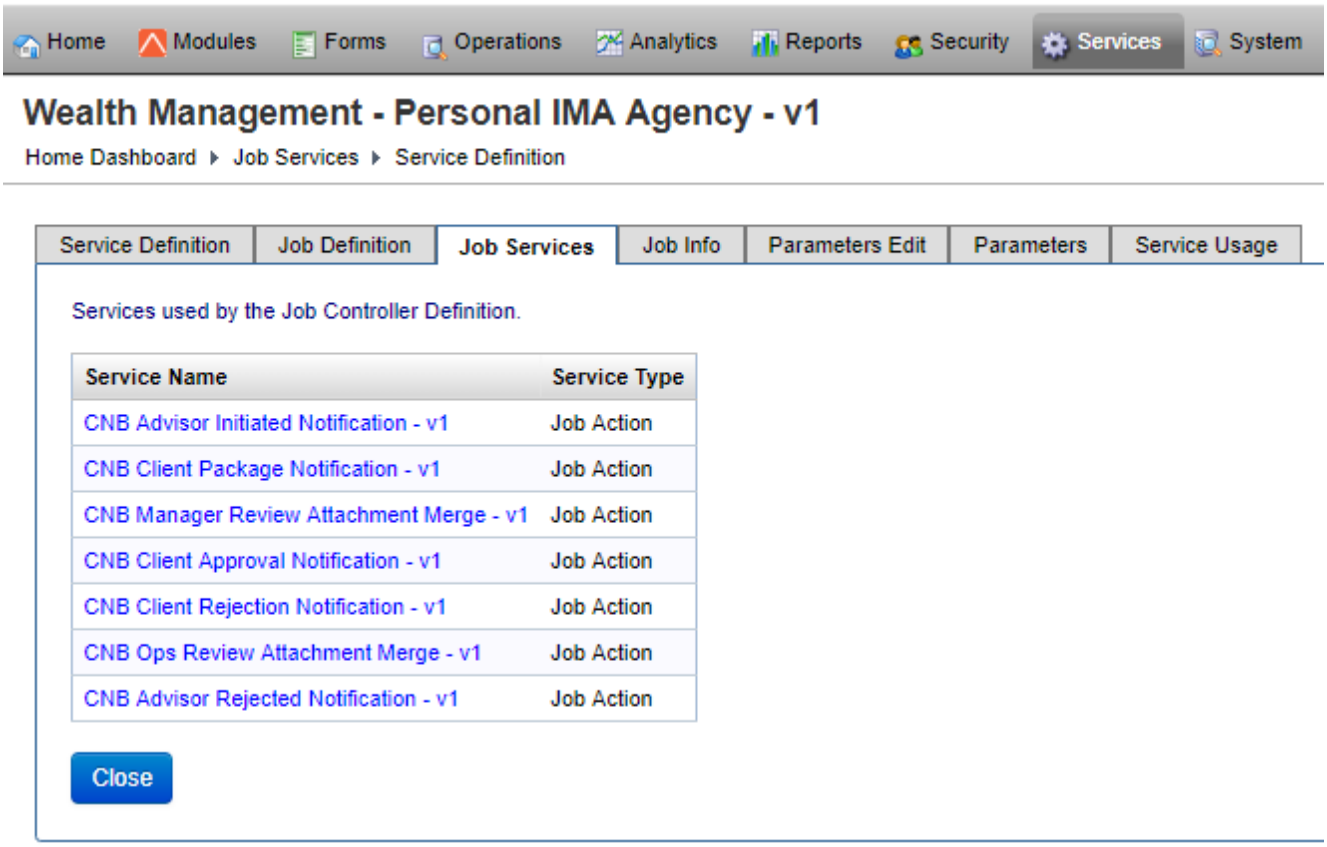
View Job Controller Services

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to viewjob services used by a job controller.

To view the list of job services:

1. Locate an existing job controller.
2. Select the **Job Services** tab to list all job services, which are configured in the job definition. The list is empty if there are no service configured in the JSON[1] file.



3. Click a service name to check its configuration.

Next, learn how to check job info.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

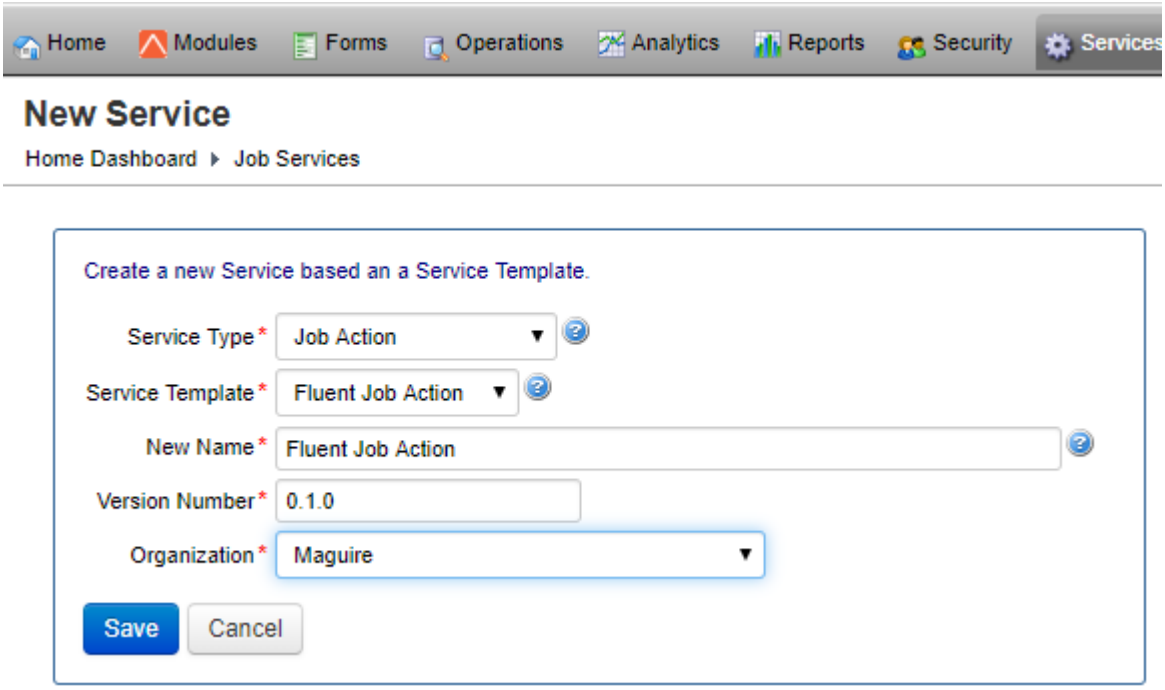
Job Action Services

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure job action services. When a collaboration job's action is processed, an action service is run. A collaboration job usually specifies an action service from one of the available inbuilt action services. This allows a developer to create their own job action and job expiry services using **Groovy**[1] scripts.

To create a job action service:

1. Select **Services > Job Services**.
2. Click **New** to create a job service.



3. Select the **Job Action** as a service type from the **Service Type** dropdown list.
4. Select the **Fluent Job Action** template from the **Service Template** dropdown list.
5. Select an organization from the **Organization** dropdown list.

> You have to assign a job controller to an organization. If you have several organizations that would like to use the same job controller, you need to define one for each organization.

6. Click **Save** to update the changes.

To configure a job action service:

---

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

1. Locate a job action service.
2. Configure the service definition.
3. Select the new service type to assign a Fluent or Groovy job action to a new type, such as **Job Expiry Service** or **Job Task Wait**.
4. Configure the job definition. You can configure an action to call a Groovy service from a job controller, as shown below.

```
"actions": [
  {
      "name": "Individual Tasks",
      "type": "Job Task Assign",
      "properties": [
         { "name": "Task Assign Repeating",  "value": "true" },
         { "name": "Task Assign Repeat Item",  "value": "$func.invoke('Repeating Get
Item', ${formDataMap.itemDelimited}, ${assignRepeatIndex})" },
         { "name": "Task Type",  "value": "Review" }
      ]
   }
```

   The **`$func.invoke`** method executes the **Repeating Get Item** service (invoking it by the name) with two parameters.

   If you want to invoke same job action multiple times as part of single step, give them different job action names, for example, **`"name": "Individual Tasks 2"`**, but define the same service name, for example,**`"serviceName": "Repeating Claimable"`**. If you do not define the service name explicitly, it expects the job action name to match the service name. It's not allowed to have 2 or more job actions with the same name in one job step.

5. Click the **Job Services** tab to view the **Repeating Get Item** service used in the action definition.

6. Click on the service name to open this service's Groovy script.

```
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import groovy.transform.TypeChecked
import javax.servlet.http.*
import org.apache.commons.lang3.StringUtils

@TypeChecked
class FluentGroovyService {

    // Injected at runtime
    public Logger logger

    /*
     * Perform a groovy service invocation
     *
     * return: the result object
     */
    Object invoke(SvcDef svcDef, HttpServletRequest request, User user, Map params)
{
        JobAction jobAction = (JobAction) params.jobAction
        return invoke(svcDef, request, user, jobAction, params)
    }

    Object invoke(SvcDef svcDef, HttpServletRequest request, User user, JobAction
jobAction, Map params) {
        logger.info params
        List<String> args = (List<String>) params.args

        def result = "Not Found"

        if( StringUtils.isNotEmpty(args[0]) &&
           StringUtils.isNotEmpty(args[1]) &&
           args[1].isNumber()) {
            def itemDelimited = args[0]
            // note repeat index from TM is 1 based not 0 based
            def repeatIndex = args[1].toInteger() - 1
            logger.info "itemDelimited: " + itemDelimited
            logger.info "repeatIndex: " + repeatIndex

            def items = itemDelimited.tokenize('|')
            logger.info "items: " + items
```

```
            result = items[repeatIndex.toInteger()]
            logger.info "result Item: " + result
        }

        return result

    }
}
```

7. Click **Save** to update the changes.

You can download the complete code example here.

Next, learn how to view all job services.

Job Service Templates

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides job service templates, which are enhanced job controller services. Templates allow a user to customize collaboration jobs without having to modify the job service definition. Templates are organization agnostic and you can't edit them. Templates are predominantly used for reviews and approvals, so Manager comes with four default review and approval templates:

- Review and Approval - 1 Step Group Review - v1.0.0
- Review and Approval - 1 Step User Review - v1.0.0
- Review and Approval - 2 Step Group Review - v1.0.0
- Review and Approval - Multi Step Group Review - v1.0.0

The job service templates are especially useful for:

- Business users and form builders to configure their forms without a developer.
- Greater reuse of a single job controller.

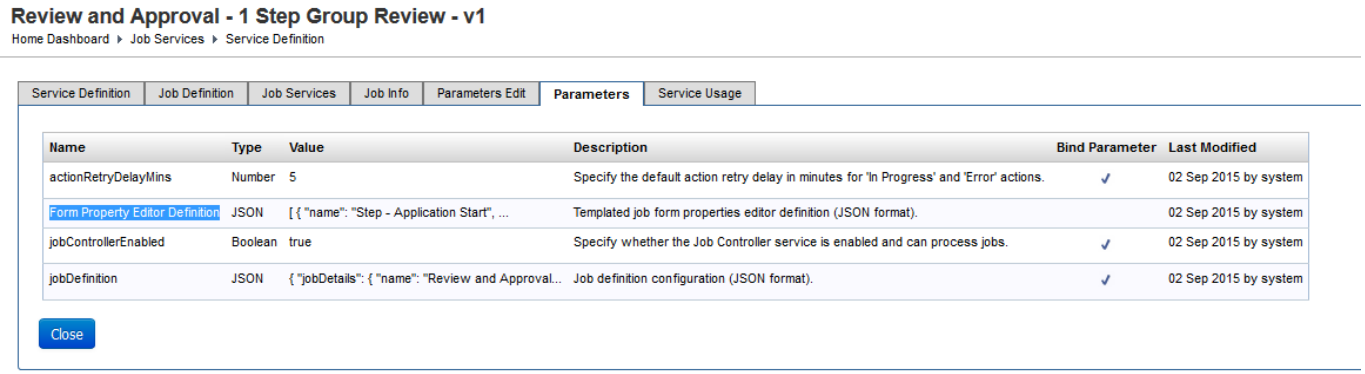
To use a job service template:

1. Locate a form version.
2. Click the **Services** tab to view services configured for this form.
3. Select a job service template from the **Job Controller Service** dropdown list, which has the following available templates:
   - Review and Approval - 1 Step Group Review - v1.0.0
   - Review and Approval - 1 Step User Review - v1.0.0
   - Review and Approval - 2 Step Group Review - v1.0.0
   - Review and Approval - Multi Step Group Review - v1.0.0
4. Click **Save** to update the changes.
5. Select the **Job Properties** tab to configure the collaboration job's properties.
6. Select the **Job Reviewers** from the **Reviewer Group** dropdown list.
7. Select the **Job Example 1** from the **Review Form** dropdown list.
8. Click **Save** to update the changes.

A job service template has the **Form Property Editor Definition** service parameter, which is a **JSON**[1] file containing the form properties meta data.

---

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.



You can create a new custom template by adding the **Form Property Editor Definition** parameter to a standard job controller.

Next, learn about job controller services.

Job Task Wait Action Services

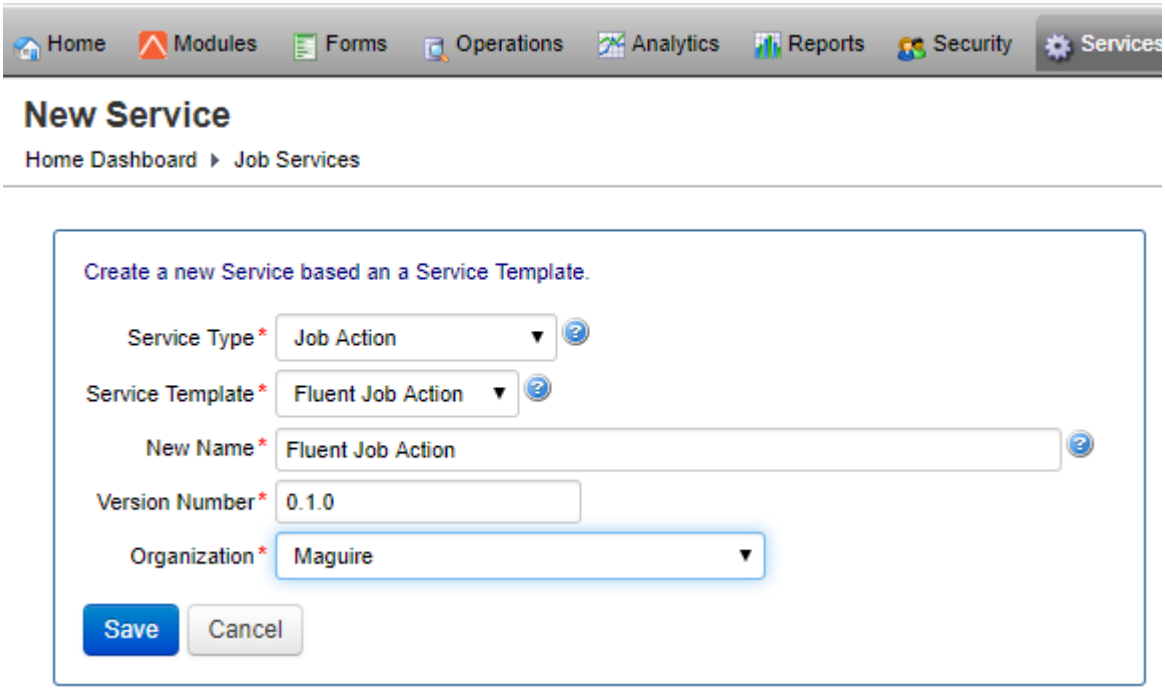
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A job task wait action service is a specialized job action service, which blocks the execution of a collaboration job until a specified event takes place.

Manager allows you to create and configure job task wait action services.

To create a job action service:

1. Select **Services > Job Services**.
2. Click **New** to create a job service.



3. Select the **Job Action** as a service type from the **Service Type** dropdown list.
4. Select the **Fluent Job Action** template from the **Service Template** dropdown list.
5. Select an organization from the **Organization** dropdown list.

> You have to assign a job controller to an organization. If you have several organizations that would like to use the same job controller, you need to define one for each organization.

6. Click **Save** to update the changes.

To configure a job action service:

1. Locate a job action service.
2. Configure the service definition.

3. Select the new service type to assign a Fluent or Groovy job action to a new type, such as **Job Expiry Service** or **Job Task Wait**.
4. Configure the job definition. You can configure an action to call a Groovy service from a job controller, as shown below.

```
"actions": [
        {
          "name": "Individual Tasks",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Task Assign Repeating",  "value": "true" },
            { "name": "Task Assign Repeat Item",  "value": "$func.invoke('Repeating
Get Item', ${formDataMap.itemDelimited}, ${assignRepeatIndex})" },
            ...
            { "name": "Task Type",  "value": "Review" }
          ]
        }
```

   The **`$func.invoke`** method executes the **Repeating Get Item** service (invoking it by the name) with two parameters.

5. Click the **Job Services** tab to view the **Repeating Get Item** service used in the action definition.

6. Click on the service name to open this service's Groovy script.

```
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import groovy.transform.TypeChecked
import javax.servlet.http.*
import org.apache.commons.lang3.StringUtils

@TypeChecked
class FluentGroovyService {

    // Injected at runtime
    public Logger logger

    /*
     * Perform a groovy service invocation
     *
     * return: the result object
     */
    Object invoke(SvcDef svcDef, HttpServletRequest request, User user, Map params)
{
        JobAction jobAction = (JobAction) params.jobAction
        return invoke(svcDef, request, user, jobAction, params)
    }

    Object invoke(SvcDef svcDef, HttpServletRequest request, User user, JobAction
jobAction, Map params) {
        logger.info params
        List<String> args = (List<String>) params.args

        def result = "Not Found"

        if( StringUtils.isNotEmpty(args[0]) &&
           StringUtils.isNotEmpty(args[1]) &&
           args[1].isNumber()) {
            def itemDelimited = args[0]
            // note repeat index from TM is 1 based not 0 based
            def repeatIndex = args[1].toInteger() - 1
            logger.info "itemDelimited: " + itemDelimited
            logger.info "repeatIndex: " + repeatIndex

            def items = itemDelimited.tokenize('|')
            logger.info "items: " + items
```

```
            result = items[repeatIndex.toInteger()]
            logger.info "result Item: " + result
        }

        return result

    }
}
```

7. Click **Save** to update the changes.

Next, learn how to view all job services.

Open the TaskUserWebService in SOAP UI

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to open the TaskUserWebService in SOAP UI. You should read the section 4 of the transact_web_service_guide_v4.2.pdf documentation prior to doing so.

The TaskUserWebService and TaskUserWebServiceV2 WSDL in Manager are located in a secure context path. This wiki article shows the workaround steps to successfully import the WSDL's and call methods in SOAP-UI.

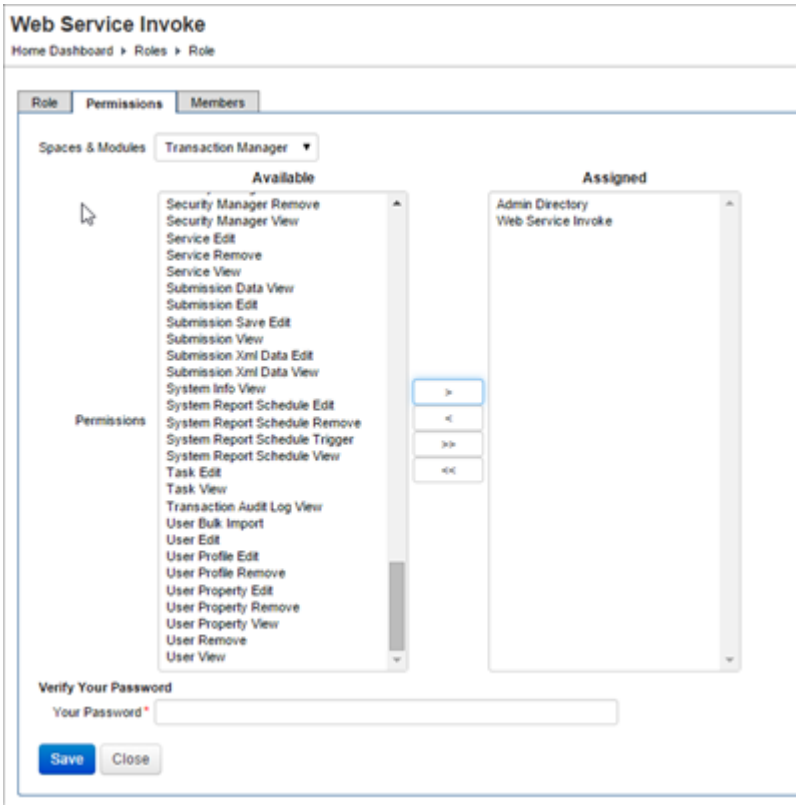
Task Web Services V2 requires you to authenticate using a username and password, It is possible to get the wsdl from the non secure context however you will be unable to call the secured services from the unsecured WSDL

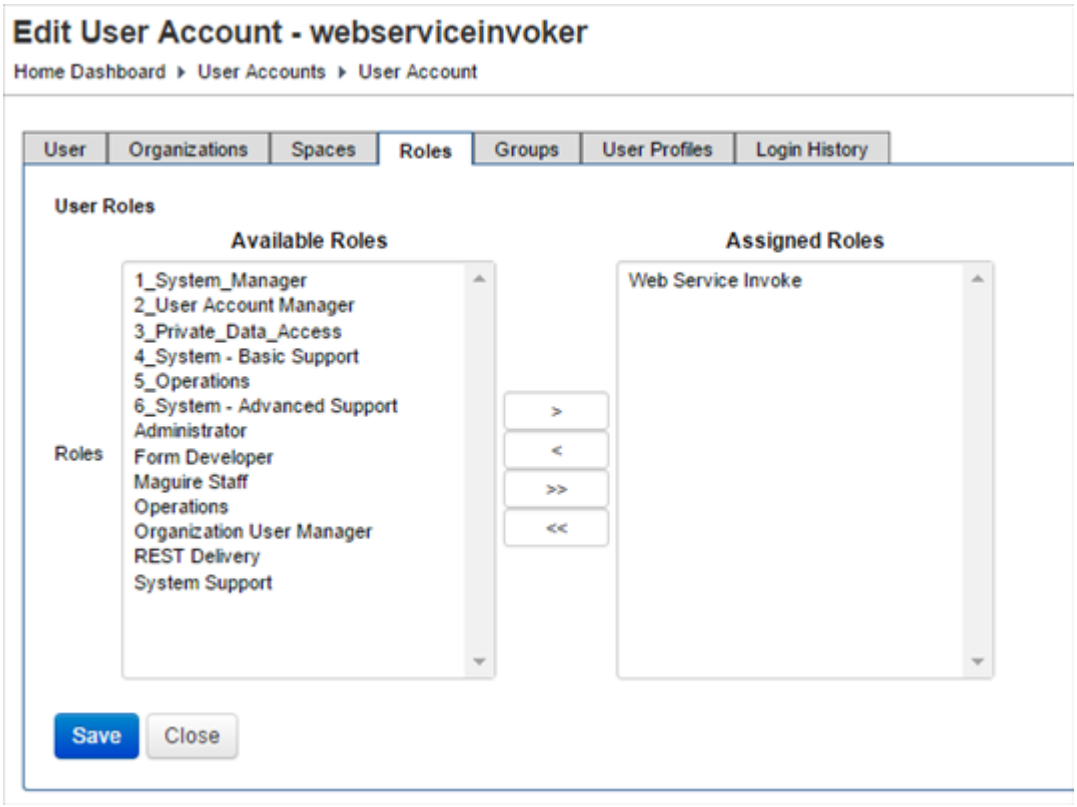
To open the service:

1. WSDL URL- https://{Server Name}/manager/secure/services/TaskUserWebServiceV2?wsdl

> Do not use the unsecure URL https://{Server Name}/manager/fcservices/TaskUserWebServiceV2?wsdl

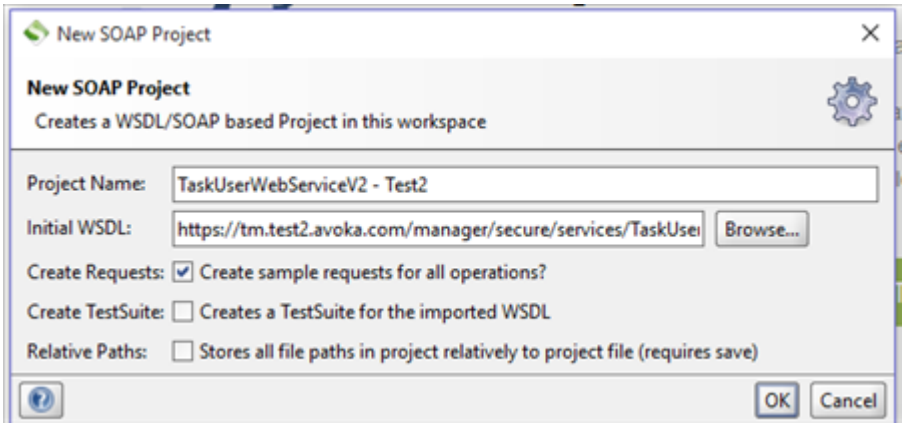
2. Create a service user that has been assigned a Role that has the Web Service Invoke permission.



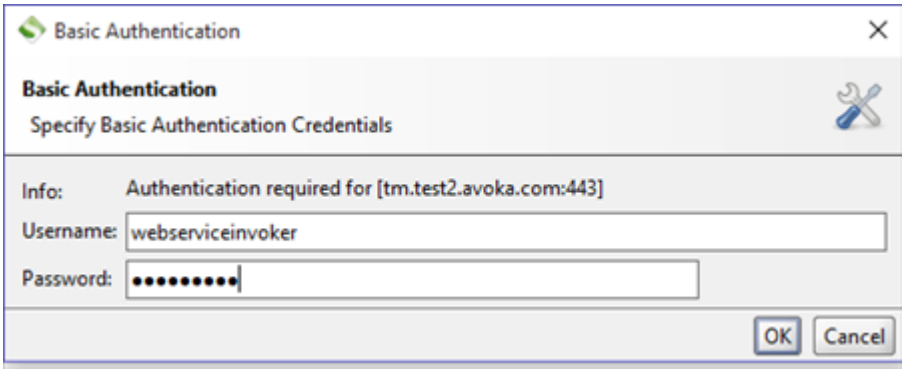


Users with the Adminitrator role have the Web Service Invoke permission.

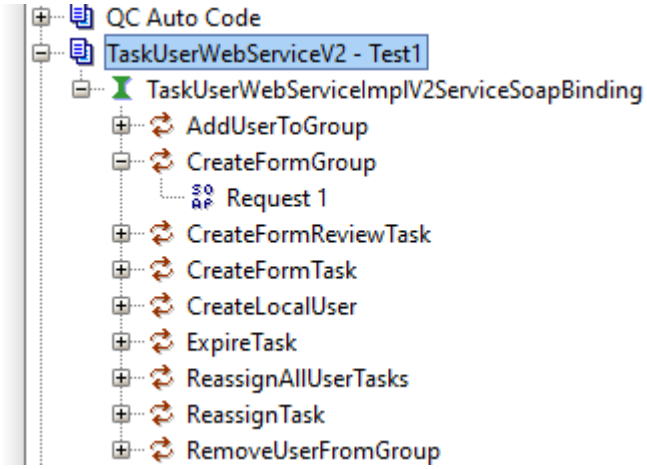
3. Create SOAP-UI project the wsdl in Step 1.



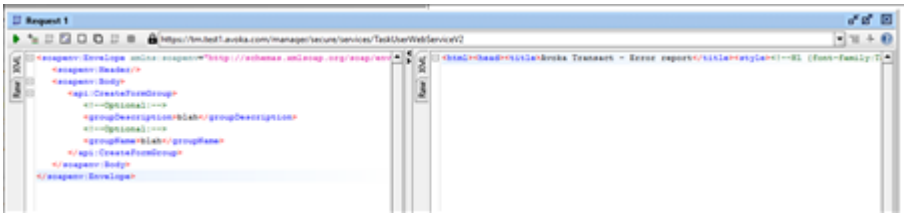
4. Soap-ui will prompt you with the Basic Authentication tab



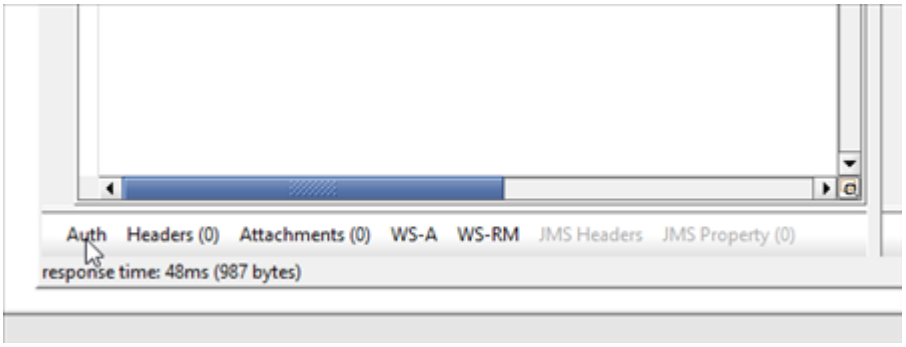
5. You should end up with the following



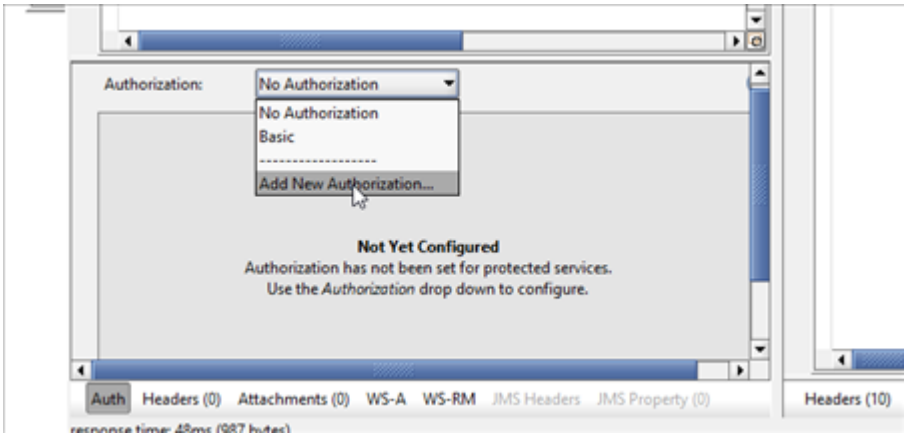
6. Try calling the web services with no authentication. Test Creating a Form Group. The response on the right tell you you are not authenticated.



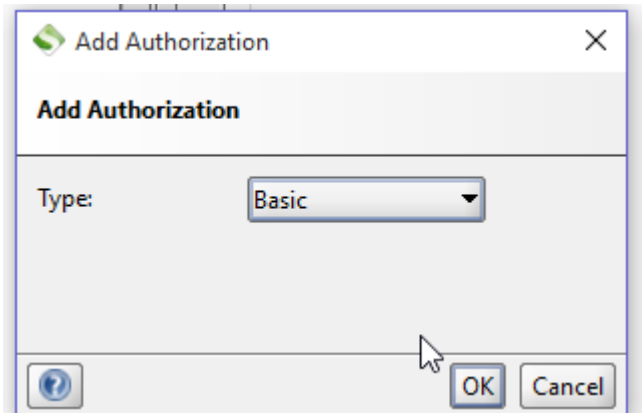
7. Click on the Auth buttong on the bottom left of the Request1 page to configure an Authentication.



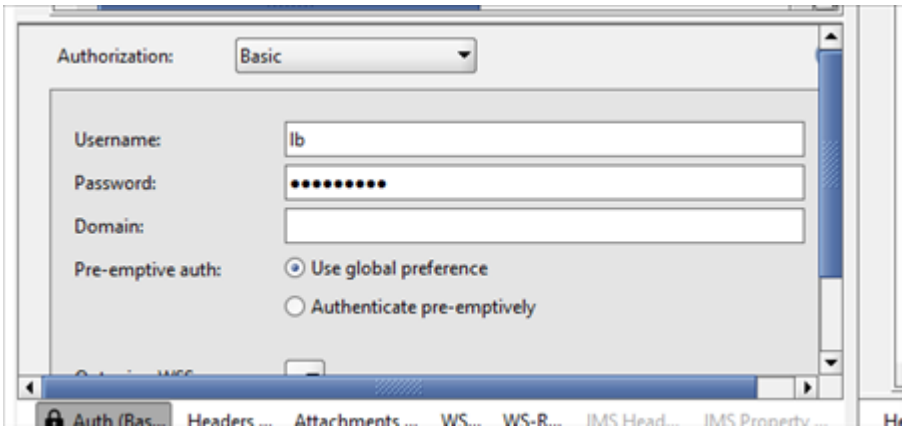
8. Select Add New Authentication.



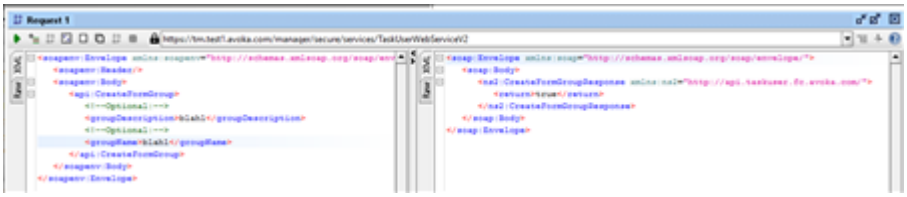
9. Select a type as Basic.



10. Specify a user name and a password.



11. Re-run the request.



Next, learn how to view services.

Steps

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A collaboration job consists of individual steps executed by Journey Manager to complete the collaboration job. The definition of which step and how they are linked together resides in the job definition of the job controller service, which processes collaboration jobs. The order of steps within the job definition is not important because they are linked with each other using the **nextStep** instruction of the **routes** element. Each step contains one or more actions. Each action completes a part of the workflow, usually by calling a service that does the work. Actions are sequential within each step.

> The order of steps are not important, but the order of actions is.

A step has several pre-defined statuses depending on the completion of actions.

A job step status can be one of the following:

- In Progress- the step is in progress now.
- Completed - the step is completed.
- Expired - the step is expired.
- Cancelled - the step has been manually cancelled by an administrator.

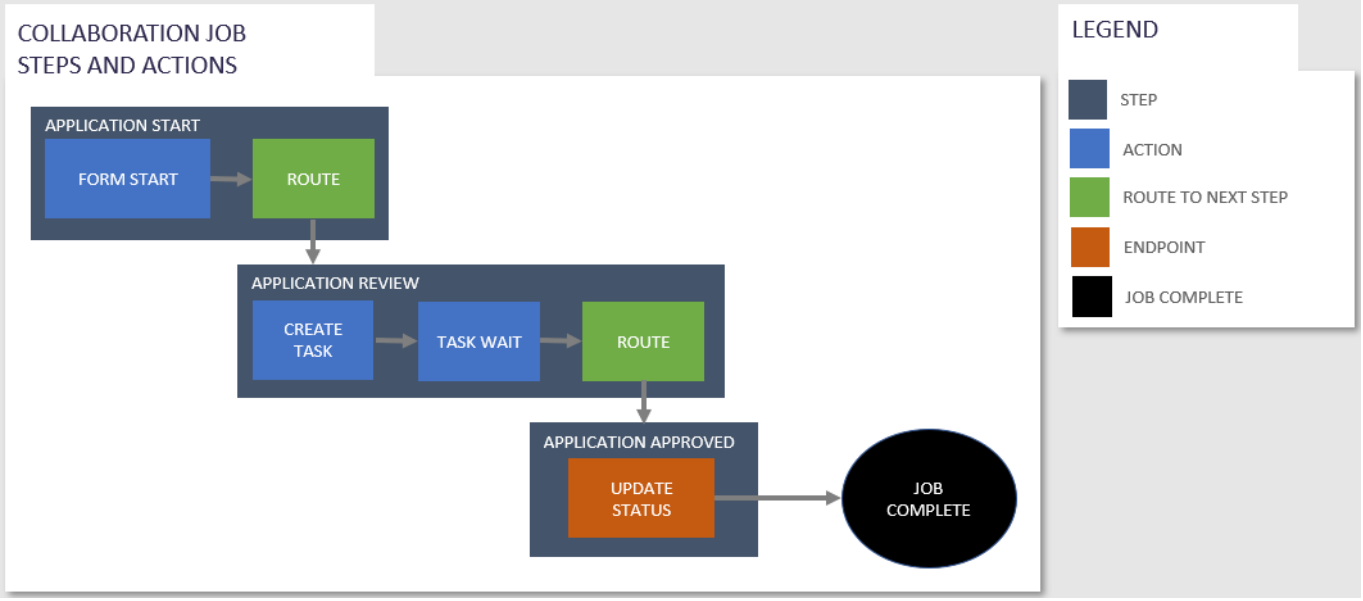
A step is completed when all its actions have executed in specified order without any exceptions, so the next step can be run. The next step is defined by either:

- The last action that returns a route name, which, in many cases, is selected by a user while completing a form.
- A step's **routes** element pointing to the next step.

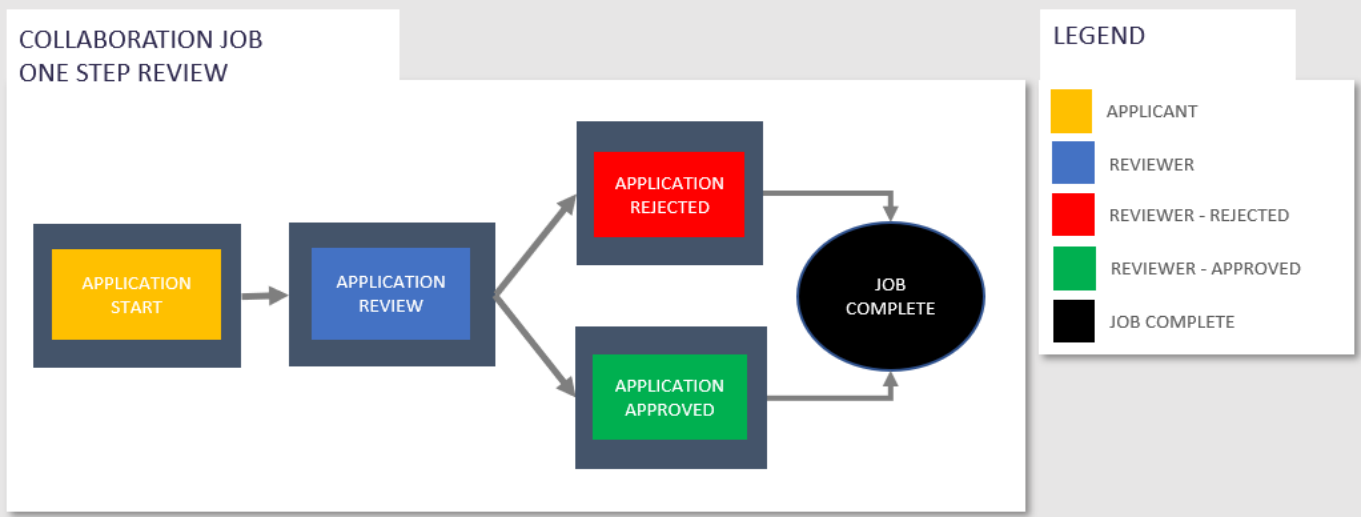
A step can be a standard or a start step. A standard step consists of:

- A list of actions which are executed in the specified order.
- An optional expiry action.
- One or more routes.

A start step is a first standard step that is executed during a collaboration job processing. The start step has the same elements as the standard step, with an extra **start** type assigned to it. You must define one of the steps in the collaboration job as the **start step**. Usually, a collaboration job is started by a form submission, so the first action in the start step is a Job Form Start action.



A collaboration job can have one or more **endpoint** steps. Each endpoint can execute none, one or more actions, such as update a status or send an email. When all actions complete, this makes this step complete, so the collaboration job can complete as well. For example, the **Application Approved** and **Application Rejected** steps are the two endpoints that make the collaboration job complete.



A step definition for the **One Step Review** is shown below.

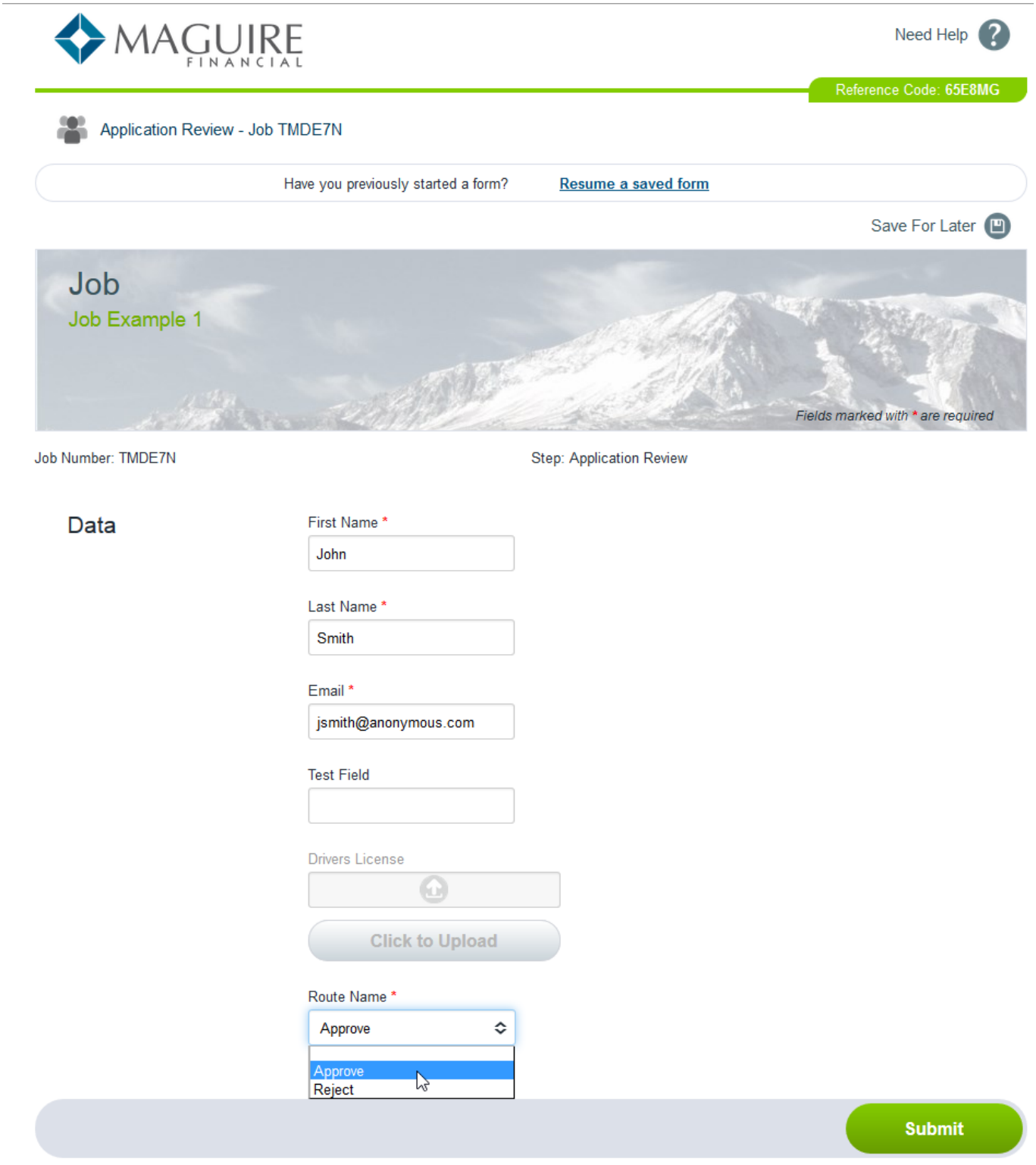
```
{
 "$schema": "http://json-schema.org/draft-07/schema#",
 "$id": "http://example.com/product.schema.json",
 "jobDetails": {
  "name": "1 Step Review",
  "processSubmitImmediate": false,
  "version": "4.1.0"
 },
 "jobGroups": {},
 "properties": {},
 "steps": {
  "Application Start": {
   "type": "start",
   "actions": {
    "Accept Quote": {
     "name": "Accept Quote",
     "type": "Job Form Start",
     "properties": {
         "Process Message Send Email": {
          "value": "true",
          "template": false
         },
         "Process Message Text": {
          "value": "Thank you ${formDataMap.firstName} ${formDataMap.lastName} your ${sub-
mission.formName} is being processed.",
          "template": false
         }
        }
    }
  },
  "routes": {
   "Default": {
    "nextStep": "Application Review"
   }
  },
  "properties": {}
  },
  "Application Review": {
   "type": "",
   "actions": {
    "Assign Review": {
        "name": "Assign Review",
        "type": "Job Task Assign",
        "properties": {
```

```
          "Task Assign Groups": {
           "value": "Job Reviewers, Job Managers",
           "template": false
          },
          "Task Form Code": {
           "value": "$func.startFormCode()",
           "template": false
          },
          "Task Message": {
           "value": "Please review the ${submission.formName} by ${formDataMap.firstName}
${formDataMap.lastName}.",
           "template": false
          },
          "Task Review Previous Step": {
           "value": "true",
           "template": false
          },
          "Task Subject": {
           "value": "Review ${submission.formName} by ${submission.contactEmailAddress}.",
           "template": false
          },
          "Task Type": {
           "value": "Review",
           "template": false
          }
         }
        },
        "Review Wait": {
         "name": "Review Wait",
         "type": "Job Task Wait",
         "properties": {}
        }
    },
    "routes": {
     "Approve": {
        "nextStep": "Application Delivery"
    },
    "Reject": {
     "nextStep": "Application Rejected"
    }
   },
   "properties": {}
  },
  "Application Delivery": {
```

```
                "type": "",
                "actions": {
                        "Application Delivery": {
                                "name": "Application Delivery",
                                "type": "Job Delivery",
                                "properties": {}
                        },
                        "Application Delivery Wait": {
                                "name": "Application Delivery Wait",
                                "type": "Job Delivery Wait",
                                "properties": {}
                        }
                },
                "routes": {
                        "Default": {
                                "nextStep": "Application Completed"
                        }
                },
                "properties": {}
        },
        "Application Completed": {
                "type": "endpoint",
                "actions": {
                        "Process Message": {
                                "name": "Process Message",
                                "type": "Job Process Message",
                                "properties": {
                                        "Process Message Send Email": {
                                                "value": "true",
                                                "template": false
                                        },
                                        "Process Message Submission": {
                                                "value": "$func.startSubmission()",
                                                "template": false
                                        },
                                        "Process Message Text": {
                                                "value": "Thank you ${formDataMap.firstName} ${formDataM
nission.formName} has been Approved.",
                                                "template": false
                                        }
                                }
                        }
                },
                "properties": {}
```

```
        },
        "Application Rejected": {
                "type": "endpoint",
                "actions": {
                        "Process Message": {
                                "name": "Process Message",
                                "type": "Job Process Message",
                                "properties": {
                                        "Process Message Send Email": {
                                                "value": "true",
                                                "template": false
                                        },
                                        "Process Message Submission": {
                                                "value": "$func.startSubmission()",
                                                "template": false
                                        },
                                        "Process Message Text": {
                                                "value": "Sorry ${formDataMap.firstName} ${formDataMap.l
nission.formName} has been declined.",
                                                "template": false
                                        }
                                }
                        }
                },
                "properties": {}
                }
        }
}
```

The **Application Review** step has two possible next steps: **Application Approved** and **Application Rejected**. The first action of the Application Review step creates and assigns a Group Task. A user who is a member of that group can open the form associated with the task as shown below.



The user has an option of approving or rejecting the form by selecting a corresponding route from the **Route Name** dropdown list before submitting the form. If the user selects Approve, the step is routed to the Application Approved step. If the user selects **Reject** , it is routed to the Application Rejected step.

You can view the list of all steps for each collaboration job.

Next, learn about collaboration job actions.

Actions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A collaboration job consists of a number of steps and actions. Each action executes an instance of an action service or a job function. There are a number of predefined actions that you can use. You can also create **Groovy**[1] actions and expiry services to implement customer requirements. You can view all actions associated with the collaboration job. A Action can have one or more action properties.

An action can be assigned to a different user. However, more often a single user is assigned to work on several actions. To achieve a seamless transition between the actions, the **`"redirectNext": true`** property is used to show next action to the same user upon the completion of the previous action.

An action can be anonymous. In this case, you should set the **`"name": "Task Type", "value": "Anonymous"`** property.

An action status can be one of the following:

- In Progress - the action is active and in progress. The action will be retried (looping), if needed.
- Completed - the action has been completed.
- Expired - the action is expired when a step is automatically expired by the system. Any actions that are In Progress (Pending, Assigned, Invoked, Ready or In Progress) will be marked as Expired. This include action associated with tasks that are in the Assigned state or job actions that are in the Invoked state awaiting a callback from a 3rd party system.
- Cancelled - the action is manually canceled by an administrator, who cancelled a job or transaction, or a user, who clicked cancel on a form task.
- Error - the action has a critical or unrecoverable error and requires an operator's or administrator's intervention before it can be run again.
- Invoked - the action that is a part of an Asynchronous execution where you have a Job Action and a Job Action Wait pair. The Job Actions processing (invoking a 3rd party service) has completed successfully, so it is awaiting a callback from a 3rd party system and, therefore, it can move to the completed status.
- Pending - or paused, the action is waiting for an external event to move it forward.
- Assigned - the Job Task Assign Action process has successfully created and assigned a task to a user, group or email (anonymous). The task is awaiting user submission to be completed which then executes a callback on the collaboration job.
- Ready - the action will be executed next time the scheduled job runs.

---

[1] Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

Manager allows you to use various actions, such as a simple, auto retry, or wait actions, that are processed differently.

Simple Actions

A job controller processes a simple action as described below:

Normal Case

An action service completes an action and sets the its status as **Completed**. Then, a job controller proceeds to the next action or a step.

Error Case

An action service may stop processing an action, depending on errors:

- Transient Errors - the errors that can be retried automatically, for example, in case a web service is down, by configuring the retry time interval of action services.
- Hard Errors - the errors that can't be fixed by retrying, for example, a bug in the source code. In this case, you may want to stop a collaboration job until the error is fixed and then run the collaboration job manually.

Polling Case

An action service can implement polling or looping logic to check if a certain event has occurred, for example, whether a receipt has been generated. If the event has occurred, the service sets the action status as **Completed** and the job controller proceeds to the next action or step. If the event has not occurred, the service sets the action status as **In Progress**.

Expiry Case

You can configure an action to expire by setting an expiry time, so the action status will be set as **Expired**. You can also configure a step to expire.

Auto Retry Actions

An action with the **Error** or **In Progress** statuses can implement the retry (looping) behavior, when it is retried each time a job controller service runs. By default, the job controller service tries to retry an action after a five minute delay, but you can change it by updating the **`actionRetryDelayMins`** service parameter. To illustrate it with an example, let's assume your action calls a web service. If the web service is down for a couple of minutes, the action instance status will be set as **Error** and the next time the action will be run is set as the **current time plus 5 minutes**. The next time your action is executed, the web service is running, so the action completes and its status is set as **Completed**. This concludes the life cycle of your action.

You can always write a custom job action in Groovy script to specify the next time it runs.

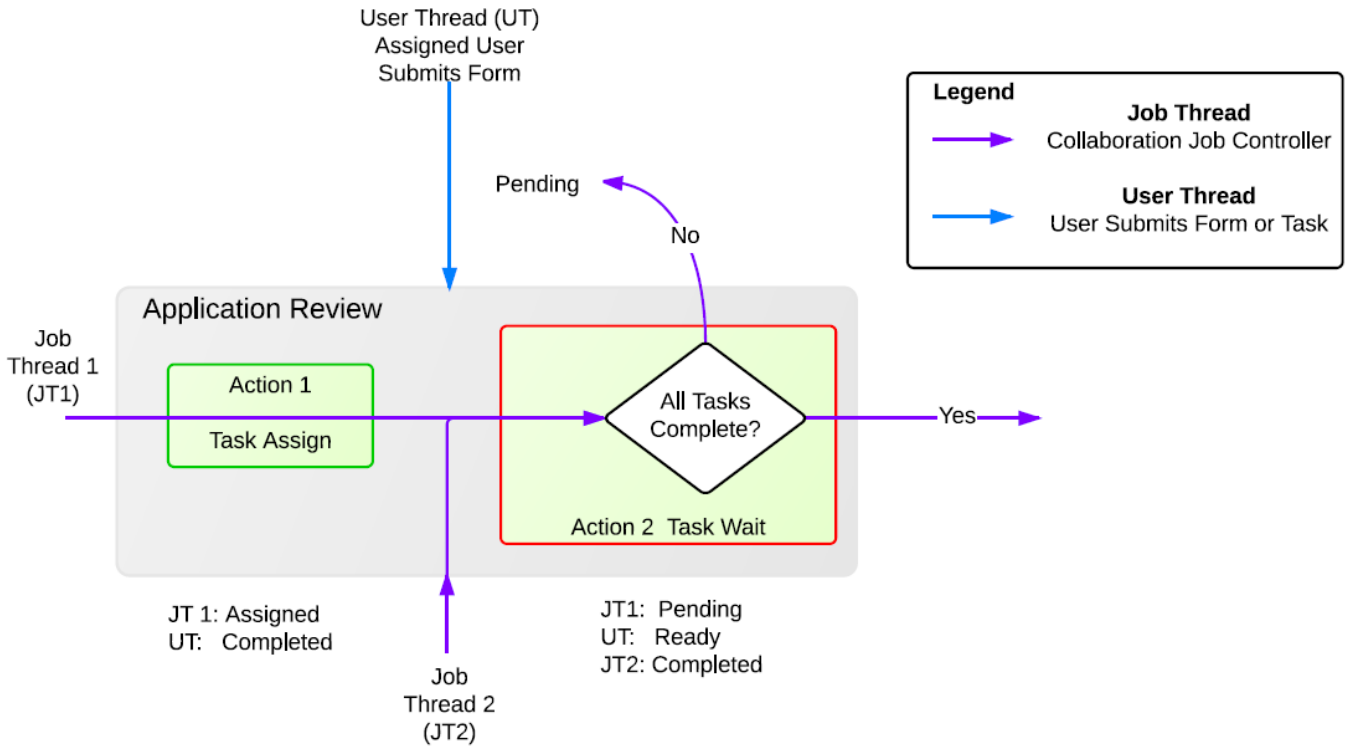
Wait Actions

A wait action implements a callback to handle asynchronous events, such as a user completing a task. A Job Task Wait Action service is paired with a single step having one or more Job Task Assign Action. The Job Task Assign Action creates a task and assigns it to a user or a group. The user completes the task by submitting it, which could be within hours, days or even weeks after the task was assigned. The submission triggers a callback, which enables the collaboration job to complete.

The examples below show how to use different actions within various steps.

Example 1 - Task Step - Standard Mode Processing

The state changes associated with the actions in a task step are illustrated in the diagram below.



Job Thread 1

1. Runs the Job Task Assign action immediately.
2. Creates and assigns the task submission.
3. Links the task submission to the action instance.
4. Sets the Task Assign Action status to **Assigned**.
5. Runs the Task Wait processing. As the Action 1 task isn't complete the Action Status of the Task Wait to Pending and Job Thread 1 terminates. No processing is done on the collaboration job where the current action is in Pending status.

Assigns User Action 1 - Task Submission Thread

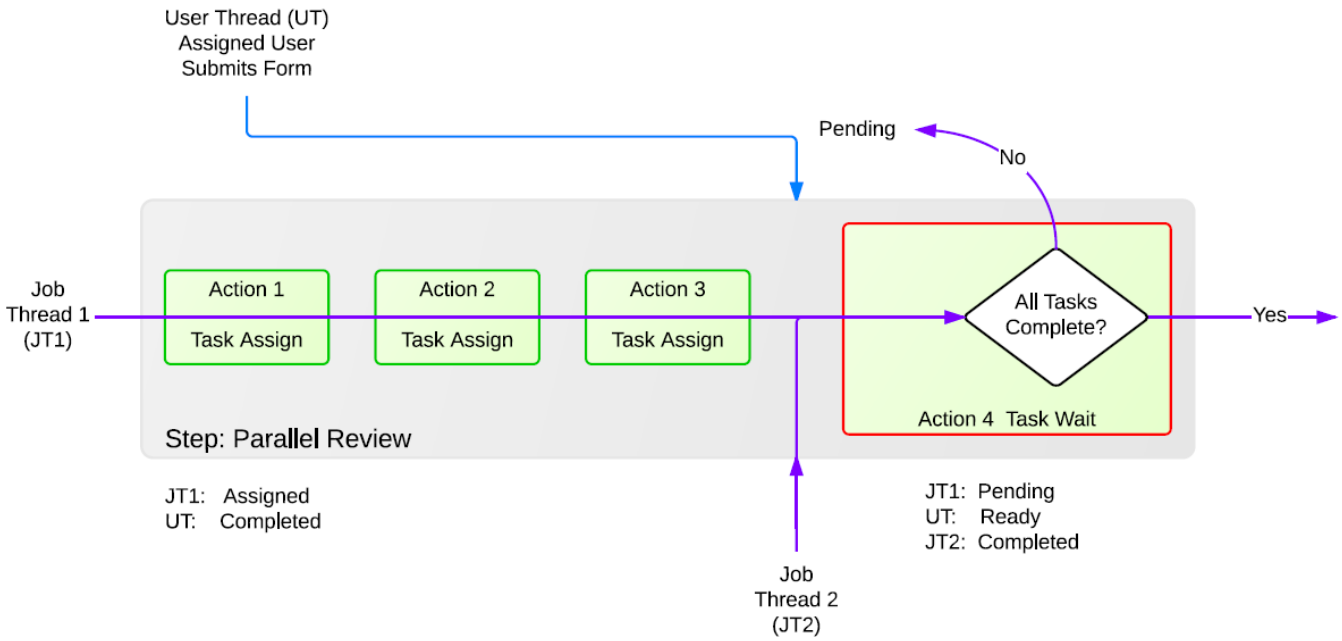
1. It generates a callback.
2. Changes the Task Assign action status from **Assigned** to **Completed**.
3. Changes the Task Wait action status from **Pending** to **Ready**.
4. As a result, the Task Wait is now available for processing.

Job Thread 2

1. Completes the Task Wait processing.
2. Changes the Task Wait status to **Completed**.
3. Completes the application review step.
4. Routes to the next step to be processed.

Example 2 - Multiple Tasks Step - Standard Mode Processing

The Job Task Wait Action service can wait on more than one Job Task Assign action services to complete. The diagram below shows a step with three assigned tasks that run in parallel. This can be created using the same form assigned to different users or groups.



Job Thread 1

1. Creates all of the action instances and the 3 associated tasks are created and are given an action status of Assigned.
2. Stops when it gets to the Task Wait and sets the status to Pending.

Assigned User Action 1 - Task Submission Thread

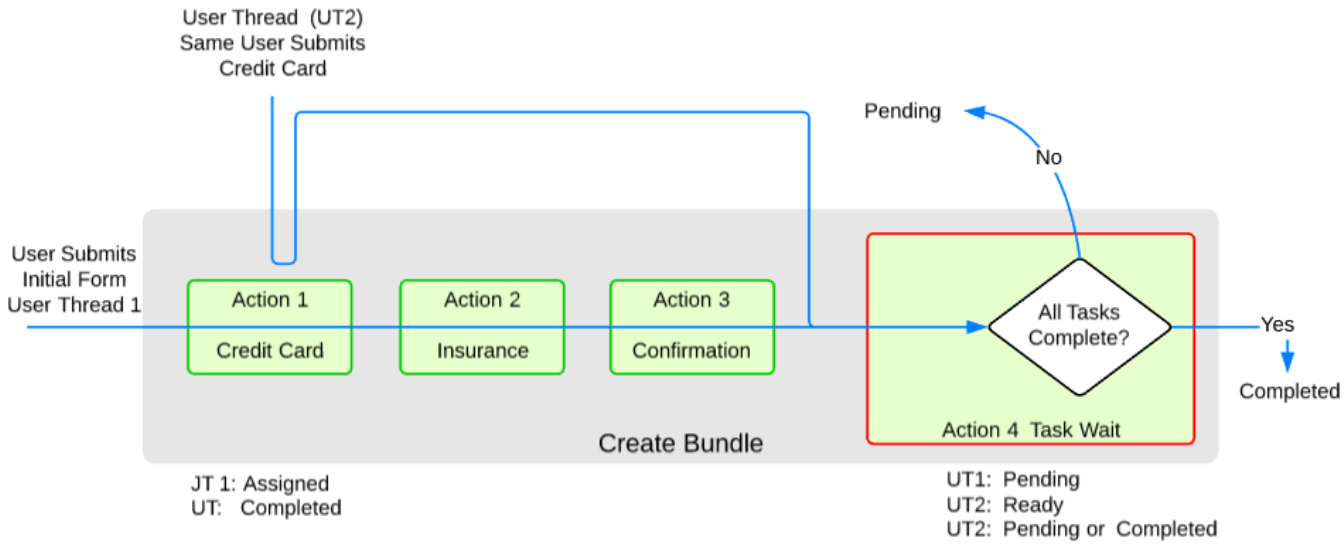
1. Callback triggered that changes the Action 1 status to Completed and Changes the Task Wait status to Ready

Job Thread 2

1. The Task Wait Processing is run which check to check whether Tasks Complete or not.
2. If complete, the Task Wait action status is set to **Completed**. The step is completed and processing the next step is done.
3. If not complete, the Task Wait action status is set to **Pending**.

Example 3 - Form Bundle - Process Immediate Mode

Form bundles can create multiple Task Assign action instances that are assigned back to the original submitter.



User Thread 1 - User Submits Initial Form

1. Creates all of the action instances and the 3 associated tasks are created and assigned to the same user, they are given an action status of Assigned.
2. Stops when it gets to the Task Wait and sets the status to Pending.

User Thread 2 - Same User Submits Credit Card:

1. Callback triggered that changes the Action 1 status to Completed and Changes the Task Wait status to Ready
2. Immediately Runs Task Wait processing is run which check to see Are Tasks Complete:
3. No: The action status for the Task Wait is set to Pending.
4. Yes: The action status for the Task Wait is set to Completed. The step is completed and processing the next step starts.

Actions can assign a group to a collaboration job. The example below shows how to assign the Job Reviewers group to a collaboration job.

```
"actions": [
    {
        "name": "Create Task",
        "type": "Job Task Assign",
        "properties": [
            ...
            { "name": "Task Assign Group",  "value": "Job Reviewers" },
            ...
        ]
    }
]
```

Next, learn how to view collaboration job steps.

Action Properties

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A Action can have a set of properties, such as the Job Task Assign service properties, which are associated with:

1. Status Updates and Email: This service can add a submission processing status messages to the specified Submission. These appear in the user's submission history. It can also send a status update via email
2. Task Assignment: These properties are used to create and assign user and group tasks. Tasks can be categorized as:

- Standard Tasks, which are for an authenticated user or group.
- Review Tasks, which are the similar to the standard tasks with attachments copied from a previous submission.
- Anonymous Save when a submission record is saved but not assigned to a user in the Manager database. They are usually assigned to a email address, so the user is sent a notification email with a link. They click on a link and enter a challenge response before they have access to the task form.

You can use the following job action properties with the **Job Task Assign** service.

Name	Description	Example
Task Type	A type of task to create, which can be an Anonymous, Form, or Review.	`{ "name": "Task Type", "value": "Review" }`
Task Form XML Data	A form XML data, such as a schema seed.	`{ "name": "Task Form XML Data", "value": "$func.startSubmissionXml()" }`
Task Input XML Prefill	An input pre-fill data XML which is mapped into the form XML data using prefill input XPath mappings. This property value is ignored for the Anonymous task type.	`{ "name": "Task Input XML Prefill", "value": "$func.invoke('Job Prefill Service', $job)" }`
Task Form Code	A form code to assign in this task. Review tasks don't need to specify a Task Form Code, in which case the form code and form version from the previous submission are used.If a form code is specified without a form version, then the current form version is used.	`{ "name": "Task Form Code", "value": "ABC123" }{ "name": "Task Form Code", "value": "$func.startFormCode()" }`
Task Form Version	This explicitly sets the form version of the form assigned in Task Form Code. This requires the Task Form Code to be specified.	`{ "name": "Task Form Version", "value": "1" }`
Task Review Previous Step	Specifies whether to review a form submission of the previous step [true | false]. If true this will override the value in Task Review Submission step. This property is	`{ "name": "Task Review Previous Step", "value": "true" }`

	only valid with the Review task type.	
Task Review Submission Step	Specifies the name of the form submission step to review. Ignored if Task Review Previous Step is set to true. This property is only valid with the Review task type.	`{ "name": "Task Review Submission Step", "value": "Application Start" }`
Task Attachments Previous Step	Specifies whether to copy of attachments from the submission of the previous step [true | false]. If true, it overrides the value in the Task Attachments Submission step. This property is ignored for a Review task which automatically copies attachments form the reviewed submission.	`{ "name": "Task Attachments Previous Step", "value": "true" }`
Task Attachments Submission Step	Specifies the name of the form submission step to copy the attachments from. It's ignored if the Task Attachments Previous step is set to true. This property is also ignored for a Review task which automatically copies attachments form the reviewed submission.	`{ "name": "Task Attachments Submission Step", "value": "Application Start" }`
Task Assign Group	This property is only valid with the Form or Review task types. The assigned task group name. A function can also specify the assigned group.	`{ "name": "Task Assign Group", "value": "Initial Review Group" }{ "name": "Task Assign Group", "value": "$func.formProperty('Manager Review Group')" }`
Task Assign Groups	This property is only valid with the Form or Review task types. The assigned task groups. A function can also specify the assigned groups.	`{ "name": "Task Assign Groups", "value": "Blue-Reviewers, Red-Reviewers" }{ "name": "Task Assign Groups", "value": "$func.formProperty('Reviewer Groups')" }`
Task Assign Group Create	Enables form groups to be automatically created if they don't already exist, and create submission group association.	`{ "name": "Task Assign Groups", "value": "$formDataMap.assignGroups" }{ "name": "Task Assign Group Create", "value": "true" } { "name": "Task Assign Repeating", "value": "true" }`
Task Assign User	This property is only valid with the Form or Review task types. The assigned task user's login name, who must be an active user in Manager.	`{ "name": "Task Assign User", "value": "johnsmith" }{ "name": "Task Assign User", "value": "$func.startUser()" }`

		{ "name": "Task Assign User", "value": "$formDataMap.-managerLoginName" }{ "name": "Task Assign User", "value": "$func.userForEmail ($formDataMap.managerEmail)" } { "name": "Task Assign User", "value": "$func.formProperty ('Manager Login')" } { "name": "Task Assign Email", "value": "john.-smith@drdaves.com" }{ "name": "Task Assign Email", "value": "$formDataMap.-doctorEmailAddress" }
Task Assign Email	This property is only valid with the Anonymous task type. The assigned task email for anonymous user email assignment.	
Task Assign Repeating	This property specifies whether the Task Assignment is repeating, and a task assignments should be performed to each assignment member, either Email, Group or User. This example assigns separate tasks to the **App QA** and **App Reviewers** groups.	{ "name": "Task Assign Group", "value": "App QA|App Reviewers" }{ "name": "Task Assign Repeating", "value": "true" }
Task Assign Repeat Item	This property specifies the Task Assign Repeat Item, and its value is made available during the evaluation of other properties, such as **${assignRepeatItem}**, and is stored in the JobAction record. This value can be used to differentiate task assignments during a repeating assignment. This data is also populated into the XML form data element, such as **//SystemProfile/Job/AssignRepeatItem**.	{ "name": "Task Assign Group", "value": "App QA|App Reviewers" }{ "name": "Task Assign Repeating", "value": "true" }{ "name": "Task Assign Repeat Item", "value": "$formDataMap.-divisionChoice" }
Task Enable Claiming	Enables group tasks to be claimed or assigned to individual users belonging to the task's groups.	{ "name": "Task Enable Claiming", "value": "true" }
Task Save Challenge	This property is only valid with the Anonymous task ype and is associated with the Anonymous save and the Task Assign Email properties. The save challenge question is configured in the form version.	{ "name": "Task Save Challenge", "value": "${formDataMap.email}" }
Task Assign Portal	A form space name to be used for the task assignment, for anonymous user email assignment (mandatory) for users or group assignment (optional). If not specified, it uses	{ "name": "Task Assign Portal", "value": "Maguire" }{ "name": "Task Assign Portal",

	the first form space assigned to the form.	`"value": "$func.formProperty ('Review Portal')" }`
Task User Deletable	Specifies whether the task can be deleted by the assignee [true | false]. If it is not set, the value is false.	`{ "name": "Task User Deletable", "value": "true" }`
Task Scheduled	The task's scheduled time in the following format: **yyyy-[m]m-[d]d.**	`{ "name": "Task Scheduled", "value": "2014-04-10" }`
Task Expiry	The task's expiry time in the following format: **yyyy-[m]m-[d]d.**	`{ "name": "Task Expiry", "value": "2014-04-10" }`
Task Menu Label	The task menu label value for use in application bundle navigation menus.	`{ "name": "Task Menu Label", "value": "Credit Card" }{ "name": "Task Menu Category", "value": "Products" }`
Task Menu Category	The task menu category for use in application bundle navigation menus to group related tasks into menus	`{ "name": "Task Menu Label", "value": "Credit Card" }{ "name": "Task Menu Category", "value": "Products" }`
Task Subject	The task's subject and title.	`{ "name": "Task Email Subject", "value": "Application Review Job" }{ "name": "Task Email Subject", "value": "$job.referenceNumber for $formDataMap.firstName $formDataMap.lastName" }`
Task Message	The task descriptive message.	`{ "name": "Task Email Message", "value": "$job.referenceNumber for $formDataMap.firstName $formDataMap.lastName }`
Task Sequence	The sorted or ordered task's sequence number.	`{ "name": "Task Sequence", "value": "1" }`
Task Address	The task location address.	`{ "name": "Task Address", "value": "1a Rialto Lane, Manly, New South Wales" }`
Task Latitude	The task location latitude in decimal degrees.	`{ "name": "Task Latitude", "value": "-33.79833" }`
Task Longitude	The task location longitude in decimal degrees.	`{ "name": "Task Longitude", "value": "151.28723" }`
Task Send Email	Specifies whether a notification email should be sent to task assignees **[true | false]**.	`{ "name": "Task Send Email", "value": "true" }`

Task Email Message	The task email notification email message body template. The following model is used: submission (Submission): the previous submissionformDataMap: Map<String, String> the form data map for the submissiontask (Submission): the task submissionformat(EmailFormat): the email formatterportal (Portal): the Space (Portal), if the portal is specified or resolvedtaskUrl (String) the link to the newly created task, if portal specified or resolvedenvironmentName (String): the environment name deployment propertysupportEmail (String): the email for the default senderNote:Most of the time this property should be removed in which case it will use the "Email Task Notification Message" template property will be looked up in this order: **form -> organization -> form space -> deployment property**. Adding large templates into the JobDefinition JSON can be problematic, so it's better to set an **Email Task Notification Message** property.	`{ "name": "Task Email Message", "value": "$job.referenceNumber for $formDataMap.firstName $formDataMap.lastName" }{ "name": "Task Email Message", "value": '$job.referenceNumber for $formDataMap.firstName $formDataMap.lastName-clickhereto open your task' }{ "name": "Task Email Message", "value": "$func.formProperty ('Assign review email message')" }`

Task Email Subject	The task email notification subject template.The following model is used: submission (Submission): the previous submission formDataMap: Map<String, String> the form data map for the submission task (Submission): the task submission format(EmailFormat): the email formatter portal (Portal): the Space (Portal), if the portal is specified or resolvedtaskUrl (String) the link to the newly created task, if portal specified or resolvedenvironmentName (String): the environment name deployment propertysupportEmail (String): the email for the default senderNote:Most of the time this property should be removed in which case it will use the "Email Task Notification Subject" template property will be looked up in this order: **form -> organization -> space (portal) -> deployment property**. Adding large templates into the JobDefinition JSON can be problematic. It may be better to set an **Email Task Notification Subject** property.	`{ "name": "Task Email Subject", "value": "Application Review Job" } { "name": "Task Email Subject", "value": "$job.referenceNumber for $formDataMap.firstName $formDataMap.lastName" } { "name": "Task Email Subject", "value": "$func.formProperty ('Assign review email subject')" }`

For more information, see **`JobTaskAssignService`** core API.

Task Types

The Task Type action properties specify either the Anonymous, Form or Review task. Form and Review tasks are both require a user to be an authenticated user, a user Task Assign User or group(s) Task Assign Group or Task Assign Groups must be specified.

Anonymous tasks don't require a user authentication, but an email address, such as Task Assign Email, and a portal, such as Task Assign Portal. When specifying the form XML and Attachments for a Review task reference has to be given to a previous submission. The Task creation process will automatically copy the XML and Attachments for you from the previous submission. However, for Form and Anonymous tasks, we don't know about previous tasks, so the XML and attachments needs to be explicitly specified.

The XML requires either the Task Form XML Data or Task Input XML Prefill to be configured.

The attachments can be copied from a previous submission using the Task Attachments Previous step or the Task Attachments Submission step

The following table summarizers the action property required by a task type. Where a single letter is repeated in two cells, only one of the action properties should be used.

Action Property Name	Form	Review	Anonymous
Task Form XML Data	A		B
Task Input XML Prefill	A		B
Task Review Previous Step		C	
Task Review Submission Step		C	
Task Attachments Previous Step	D		D
Task Attachments Submission Step	D		D
Task Assign User	E*	F*	
Task Assign Group	E	F	
Task Assign Groups	E	F	
Task Assign Email			G
Task Assign Portal	Optional	Optional	H

Note: * A user can be assigned at the same time as a claimable group. This is the same as a user claiming the Task. The user must belong to one of the assigned groups.

A Form Task job definition is shown below.

```
{
  "jobDetails": {
    "name": "1 Step Review - Form Task with Attach",
    "version": "4.2.0"
  },
  "steps": [
    {
      "name": "Application Start",
      "type": "start",
      "actions": [
        {
          "name": "Accept Quote",
          "type": "Job Form Start"
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Application Review" }
      ]
    },
    {
      "name": "Application Review",
      "type": "",
      "actions": [
        {
          "name": "Assign Review",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Task Assign User",  "value": "roger" },
            { "name": "Task Form Code",  "value": "$func.startFormCode()" },
            { "name": "Task Message",  "value": "blah" },
            { "name": "Task Attachments Previous Step",  "value": "true" },
            { "name": "Task Form XML Data",  "value": "$func.stepOrStartSubmissionXml()"
},
            { "name": "Task Review Previous Step",  "value": "true" },
            { "name": "Task Send Email",  "value": false },
            { "name": "Task Subject",  "value": "Review this" },
            { "name": "Task Type",  "value": "Form" }
          ]
        },
        {
          "name": "Review Wait",
          "type": "Job Task Wait"
        }
      ],
```

```
      "routes": [
        { "name": "Approve",  "nextStep": "Application Completed" },
        { "name": "Reject",  "nextStep": "Application Rejected" }
      ]
    },
    {
      "name": "Application Completed",
      "type": "endpoint"
    },
    {
      "name": "Application Rejected",
      "type": "endpoint"
    }
  ]
}
```

An Anonymous Task job definition is shown below.

```
{
  "jobDetails": {
    "name": "1 Step Review - Form Task with Attach",
    "version": "4.2.0"
  },
  "steps": [
    {
      "name": "Application Start",
      "type": "start",
      "actions": [
        {
          "name": "Accept Quote",
          "type": "Job Form Start"
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Application Review" }
      ]
    },
    {
      "name": "Application Review",
      "type": "",
      "actions": [
        {
          "name": "Assign Review",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Task Assign Email",  "value": "${formDataMap.email}" },
            { "name": "Task Form Code",  "value": "$func.startFormCode()" },
            { "name": "Task Message",  "value": "Please review the ${submission.formName}
by ${formDataMap.firstName} ${formDataMap.lastName}." },
            { "name": "Task Subject",  "value": "Review ${submission.formName} by ${sub-
mission.contactEmailAddress}." },
            { "name": "Task Assign Portal",  "value": "Maguire" },
            { "name": "Task Form XML Data",  "value": "$func.stepOrStartSubmissionXml()"
},
            { "name": "Task Attachments Submission Step",  "value": "Application Start"
},
            { "name": "Task Send Email",  "value": "true" },
            { "name": "Task Email Message",  "value": "$func.formProperty('Anonymous Task
Email Message')" },
            { "name": "Task Email Subject",  "value": "$func.formProperty('Anonymous Task
Email Subject')" },
            { "name": "Task Type",  "value": "Anonymous" }
```

```
          ]
        },
        {
          "name": "Review Wait",
          "type": "Job Task Wait"
        }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Reject",  "nextStep": "Application Rejected" }
      ]
    }    {
      "name": "Application Completed",
      "type": "endpoint"
    },
    {
      "name": "Application Rejected",
      "type": "endpoint"
    }
  ]
}
```

Job Functions

You can use **$func** as an Action Property value. For more information, see JobFunctions Core API.

For example, if you need to fetch a submission property value within a collaboration job controller JSON definition, which to be injected into a job action property value, you should use the **$func.startSubmission.getSubmissionPropertyValue('propertyName')** call.

Next, learn about job functions.

Collaboration Job Basics

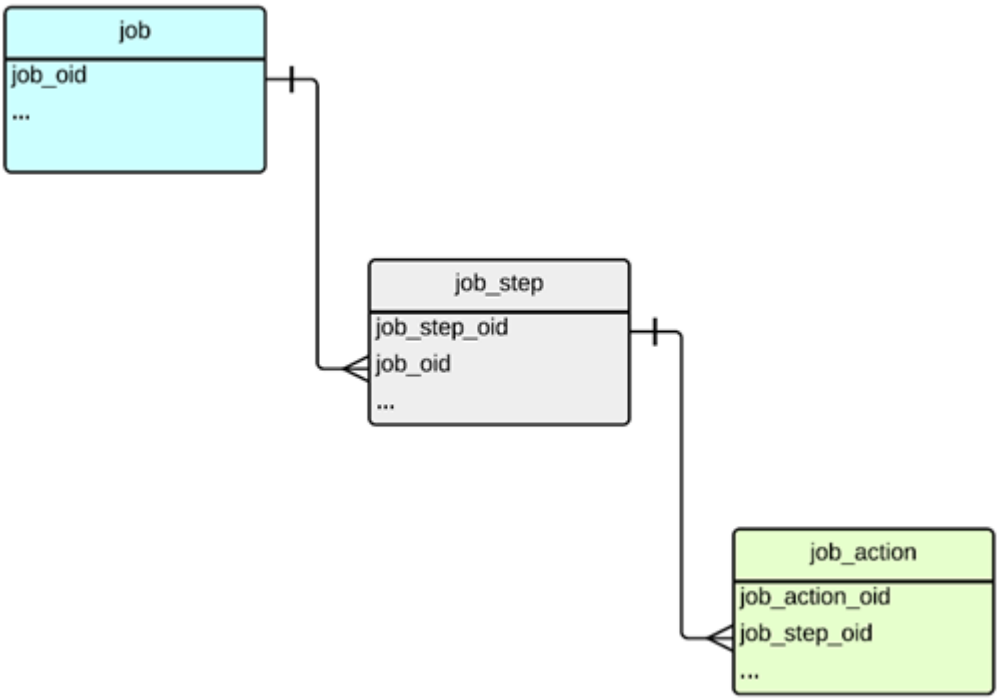
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A collaboration job contains of two or more steps, each step contains one or more actions. This hierarchy flows down into the design of the job database entities and the structure of a job definition.

Manager stores collaboration jobs' entities in the following database tables:

- job - job instances
- job_step - step instances
- job_action- action instances

The tables relationship is shown in the diagram below.



A collaboration job has a status.

A job status can be one of the following:

- In Progress - the collaboration job is active and in progress.
- Completed - the collaboration job has been completed. This happens when a endpoint step has completed.
- Expired - this happens when an endpoint expiry collaboration job has been automatically expired by Journey Manager.
- Cancelled - the collaboration job has been manually canceled by an administrator.
- Error - the collaboration job has a critical or unrecoverable error that requires an administrator intervention before it can be run again.
- Paused - the collaboration job has been paused temporarily.

Next, learn how to.

View Collaboration Jobs

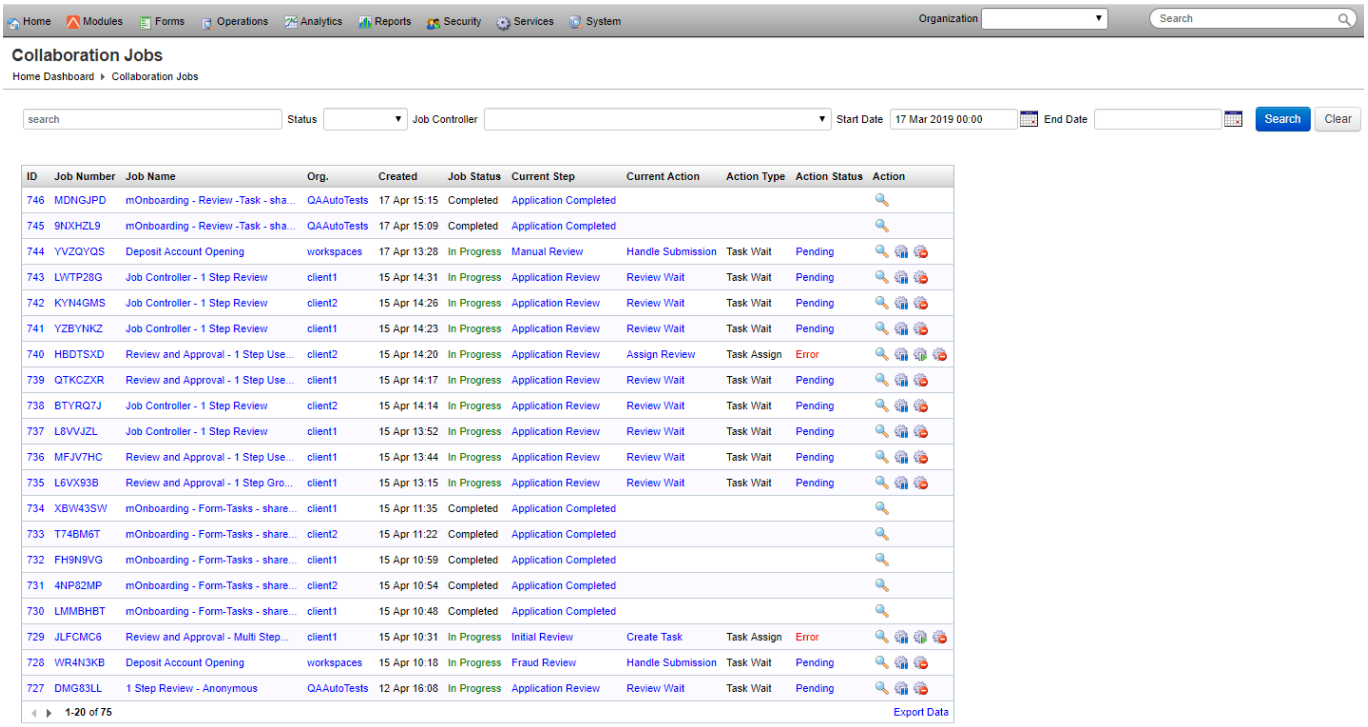
Journey Manager (JM) | System Manager / DevOps | 19.05

Manager allows you to search and view collaboration jobs that are in progress or have completed but have not been purged yet. You can also narrow it down to drill into complex workflows to see which step and action each collaboration job is.This is especially useful in troubleshooting transactions.

> You have to be a Job Coordinator or an Administrator to use this functionality.

To view collaboration jobs, select **Operations > Collaboration Jobs**.

Collaboration jobs are shown page by page, with the maximum number of records per page configured in the users preferences.



The collaboration job list displays the following details for each job:

- **ID** is an internal table ID of a collaboration job, which uniquely identifies it in Manager.
- **Job Number** (**Job Reference Number**) is a unique code, which is generated on the first submission of a form and assigned to a collaboration job using:
  - A Job Number, which is the default behavior of a collaboration job created on a form submission, The job number is shown on a form as a reference code (a transaction tracking code), so a form user (the form applicant) can use to reference the status of their application.
  - A Job Reference Number, when a collaboration job is created using the Fluent API, such as `JobBuilder`.

  > Each subsequent job task has a different tracking code to the initial submitted form. However, they may show the Job Number in a separate field on the form.

- **Job Name** is the name of the job controller service this collaboration job use.
- **Org** is an organization this collaboration job belongs to.
- **Created** is day and time when this collaboration job was created.
- **Job Status** is a status of a collaboration job, which can be in one of the several states depending on executions of its steps and actions.

  A job status can be one of the following:

  - In Progress - the collaboration job is active and in progress.
  - Completed - the collaboration job has been completed. This happens when a endpoint step has completed.
  - Expired - this happens when an endpoint expiry collaboration job has been automatically expired by Journey Manager.
  - Cancelled - the collaboration job has been manually canceled by an administrator.
  - Error - the collaboration job has a critical or unrecoverable error that requires an administrator intervention before it can be run again.
  - Paused - the collaboration job has been paused temporarily.

- **Current Step** is the step the collaboration job is at.
- **Current Action** is the action the collaboration job is at.
- **Action Type** is a type of the action.

An action type can be one of the following:

- Action
- Input
- Output
- Task Assign
- Task Wait

- **Action Status** is a status of the action.

  An action status can be one of the following:

  - In Progress - the action is active and in progress. The action will be retried (looping), if needed.
  - Completed - the action has been completed.
  - Expired - the action is expired when a step is automatically expired by the system. Any actions that are In Progress (Pending, Assigned, Invoked, Ready or In Progress) will be marked as Expired. This include action associated with tasks that are in the Assigned state or job actions that are in the Invoked state awaiting a callback from a 3rd party system.
  - Cancelled - the action is manually canceled by an administrator, who cancelled a job or transaction, or a user, who clicked cancel on a form task.
  - Error - the action has a critical or unrecoverable error and requires an operator's or administrator's intervention before it can be run again.
  - Invoked - the action that is a part of an Asynchronous execution where you have a Job Action and a Job Action Wait pair. The Job Actions processing (invoking a 3rd party service) has completed successfully, so it is awaiting a callback from a 3rd party system and, therefore, it can move to the completed status.
  - Pending - or paused, the action is waiting for an external event to move it forward.
  - Assigned - the Job Task Assign Action process has successfully created and assigned a task to a user, group or email (anonymous). The task is awaiting user submission to be completed which then executes a callback on the collaboration job.
  - Ready - the action will be executed next time the scheduled job runs.

To filter or search the collaboration jobs by one or more criteria, specify the following settings and click **Search**:

- Enter a collaboration job's ID, job number, job name or part of a name, for example, **claimable**.
- Select a collaboration job status from the **Status** dropdown list.
- Select a job controller from the **Job Controller** dropdown list.

> It only includes job controllers for organizations that the administrator has access to, so the search results only show the job instances linked to those organizations.

- Specify the Start Date
- Specify the End Date
- Select an organization from the Organization dropdown list. It applies to a global administrator, or an organizational administrator that has been allocated to more than one organization.

Click **Clear** to reset the search criteria.

You can also perform the following actions:

- Click **View** to view collaboration job details.
- Click **Pause Job** to temporarily pause collaboration job processing.
- Click **Process Job** to start collaboration job processing immediately, which is useful for developers to troubleshoot an action service as they can make a change to the service and execute it to see if it works or not right away. An administrator can re-run a failed action once it is fixed.
- Click **Cancel Job** to permanently cancel a job that is in progress.
- Click **Remove Job** to permanently remove the collaboration job after it was canceled. This is useful for cleaning out test collaboration jobs created during development.

> You can't manually create a collaboration job, because it is based on a form submission where a form is associated with a job controller service.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

> You can also view a collaboration job workflow in Journey Analytics.

Next, learn how to check collaboration job details.

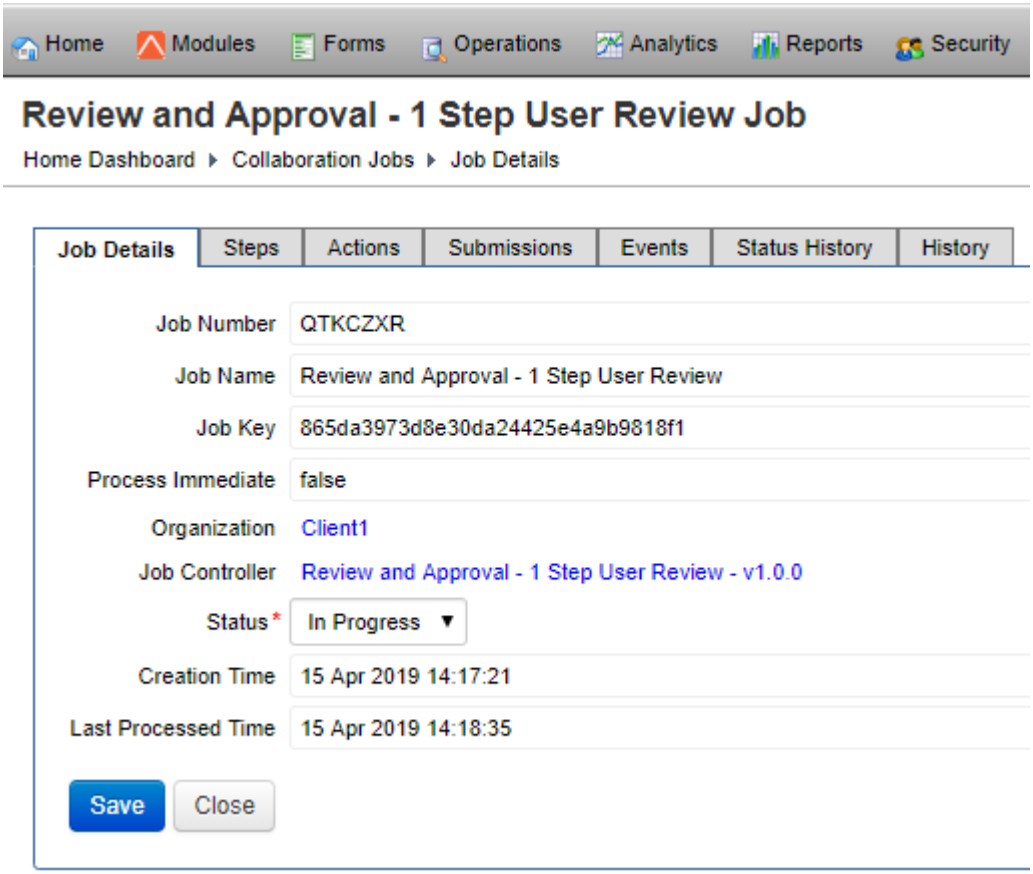
Collaboration Job Details

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to check details of a collaboration job and update its status as required by your workflow or to resolve an issue related to this collaboration job.

To view the details of a collaboration job:

1. Select **Operations > Collaboration Jobs**.
2. Locate a collaboration job and click **View.**



3. Check the collaboration job's information in the **Job Details** tab, which is the same as described in view collaboration jobs with few extra fields:

- **Process Immediate** indicates how this collaboration job is processed by a scheduled job: true - the collaboration job is processed right after it is created at the end of the submission, false - it's queued for a scheduled job.
- **Last Processed Time** shows the time the collaboration job was last processed.

4. Select a new collaboration job status from the **Status** dropdown list, which can be **In Progress**, **Paused**, **Cancelled**.

You should understand statuses and transitions between them, as some transitions are irreversible. If you change the status from **In Progress** to **Cancelled**, you can't undo it. However, if you change the status from **In Progress** to **Paused**, you can change it back to **In Progress**.

5. Click **Save** to update the changes. It saves the updated status only.

Next, learn how to view collaboration job steps.

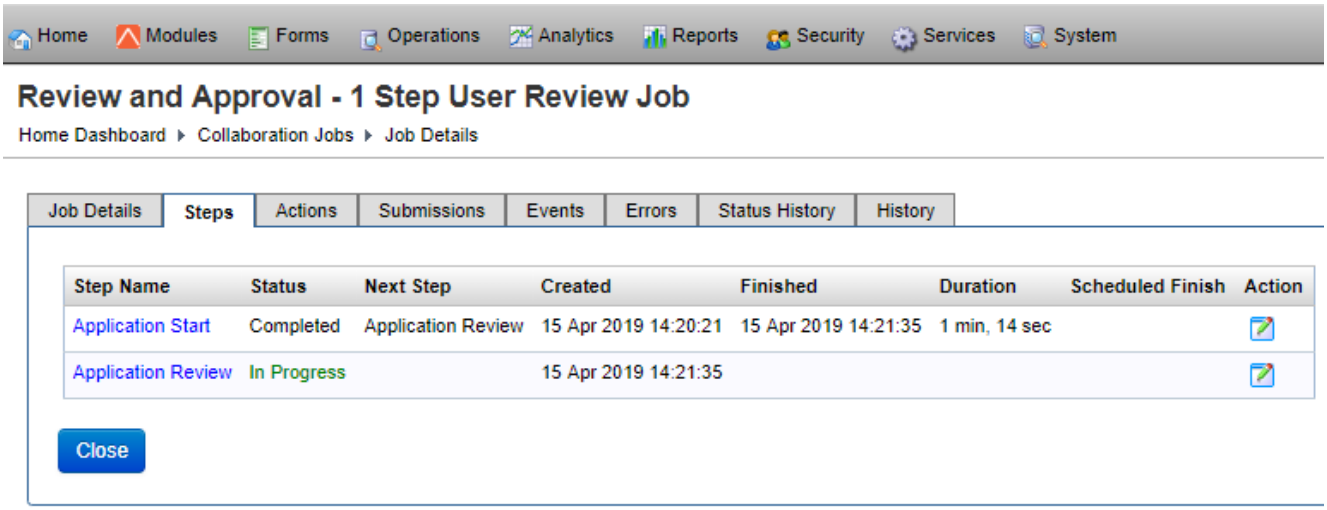
View Collaboration Job Steps

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view steps comprising a collaboration job.

To view the steps of a collaboration job:

1. Select **Operations > Collaboration Jobs**.
2. Locate a collaboration job and click **View**.
3. Select the **Steps** tab to see the collaboration job's steps and their information, such as:

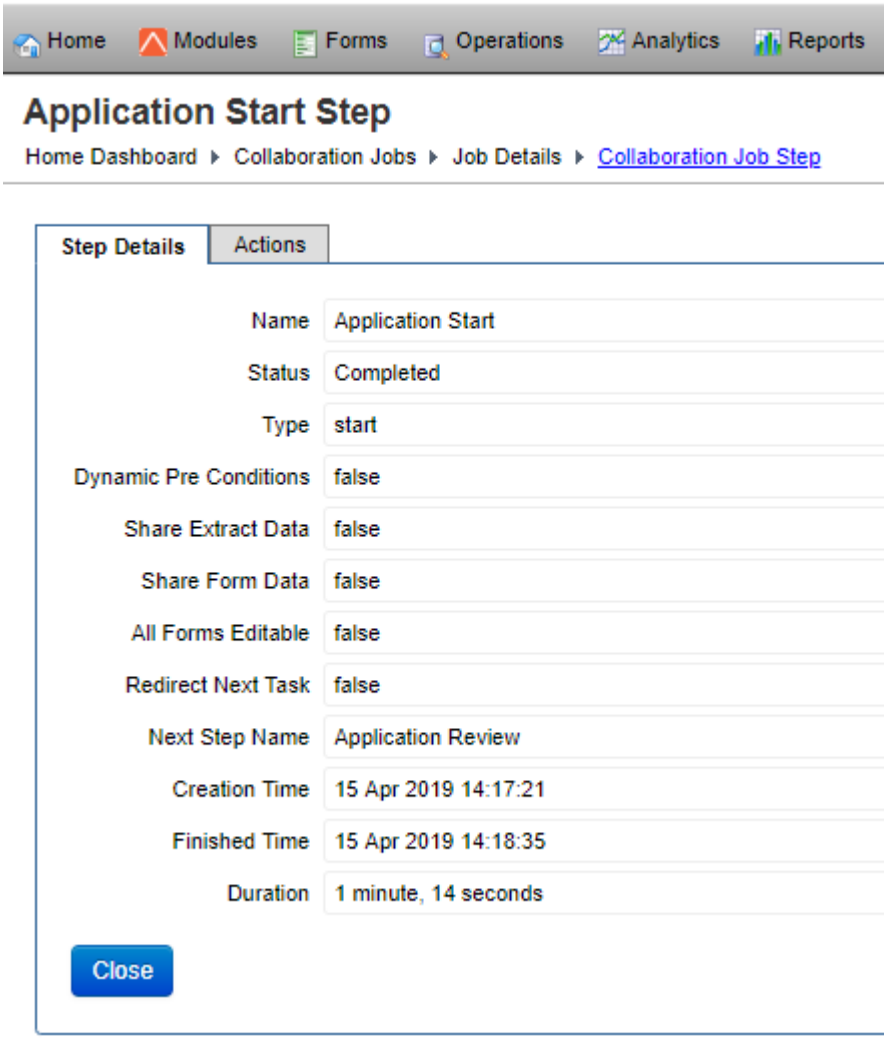


- **Step Name** is a name of a step, which is a part of a job definition of the collaboration job.
- **Step Status** is a current status of a step.

  A job step status can be one of the following:

- In Progress- the step is in progress now.
- Completed - the step is completed.
- Expired - the step is expired.
- Cancelled - the step has been manually cancelled by an administrator.
- **Next Step** is a name of a next step as per the job definition.
- **Created** is the day and time the step's execution started.
- **Finished** is the day and time the step's execution finished.
- **Duration** is the time it has taken to complete the step.
- **Scheduled Finish** is the day and time the step's expected to complete.

4. Click a step name to view its extra information, such as:



- **Type** is a type of the step. The type **start** marks the step as being the first to be executed. Otherwise, the type field is not shown.
- **Dynamic Pre Conditions** specifies whether the step has dynamic preconditions or not.
- **Share Extract Data** specifies whether the step has a share extract data or not.
- **All Forms Editable** specifies whether the step has all forms editable or not.
- **Redirect Next Task** specifies whether the step has a redirect next task or not.

5. Select the **Actions** tab to see actions associated for this step.
6. Click **Close**.

Next, learn how to view collaboration job actions.

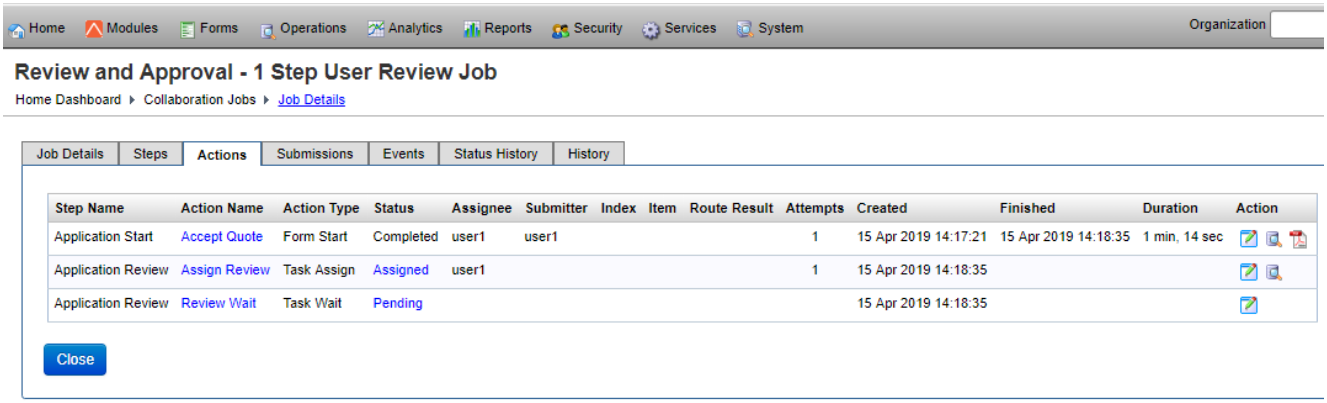
View Collaboration Job Actions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view actions of each step comprising a collaboration job, so you can see at a glance where each step is at and what action service it invokes.

To view the actions of a collaboration job:

1. Select **Operations > Collaboration Jobs**.
2. Locate a collaboration job and click **View**.
3. Select the **Actions** tab to see the collaboration job's actions and their information, such as:



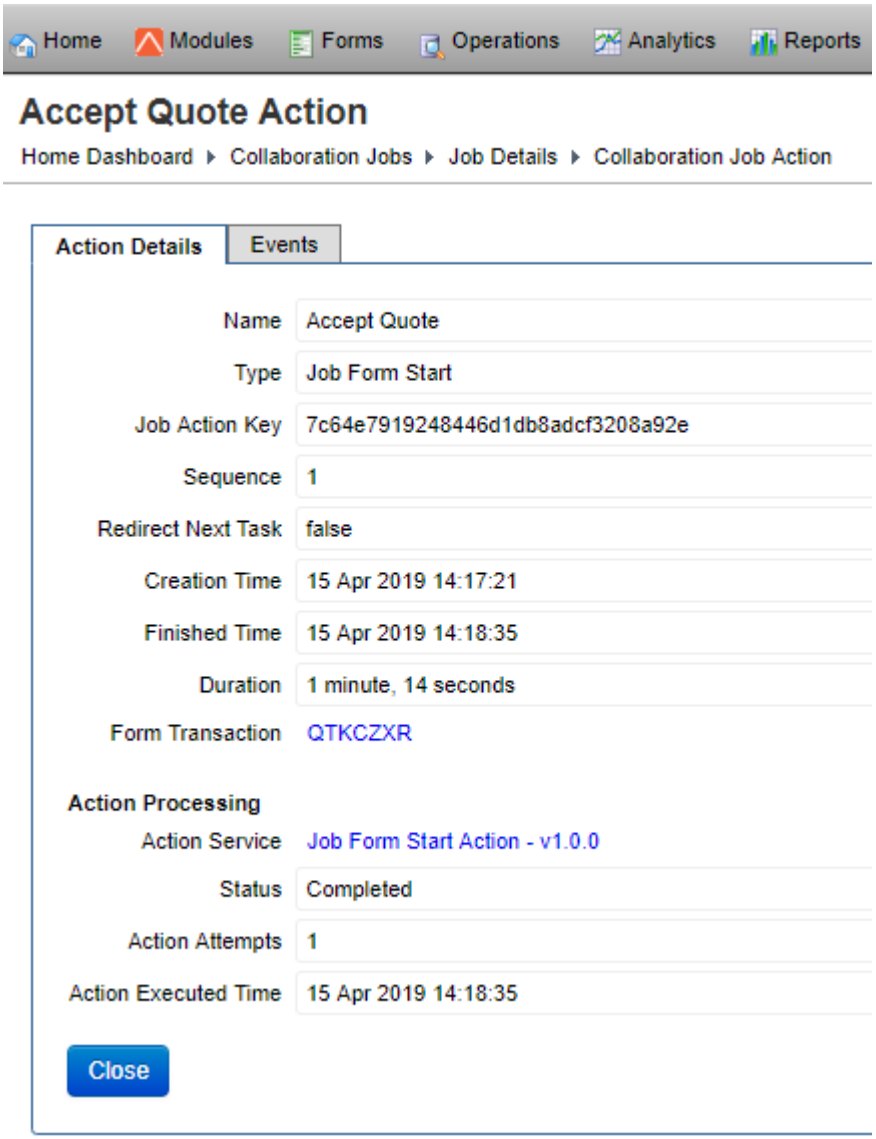
- **Step Name** is a name of a step, which is a part of a job definition of the collaboration job.
- **Action Name** is a name of an action, which is a part of the job definition.
- **Action Type** is a type of the action.
- **Status** is a status of the collaboration job.

  An action status can be one of the following:

  - In Progress - the action is active and in progress. The action will be retried (looping), if needed.
  - Completed - the action has been completed.
  - Expired - the action is expired when a step is automatically expired by the system. Any actions that are In Progress (Pending, Assigned, Invoked, Ready or In Progress) will be marked as Expired. This include action associated with tasks that are in the Assigned state or job actions that are in the Invoked state awaiting a callback from a 3rd party system.
  - Cancelled - the action is manually canceled by an administrator, who cancelled a job or transaction, or a user, who clicked cancel on a form task.
  - Error - the action has a critical or unrecoverable error and requires an operator's or administrator's intervention before it can be run again.
  - Invoked - the action that is a part of an Asynchronous execution where you have a Job Action and a Job Action Wait pair. The Job Actions processing (invoking a 3rd party service) has completed successfully, so it is awaiting a callback from a 3rd party system and, therefore, it can move to the completed status.
  - Pending - or paused, the action is waiting for an external event to move it forward.
  - Assigned - the Job Task Assign Action process has successfully created and assigned a task to a user, group or email (anonymous). The task is awaiting user submission to be completed which then executes a callback on the collaboration job.
  - Ready - the action will be executed next time the scheduled job runs.

- **Assignee** is a group, user, or email (an anonymous user) this collaboration job is assigned to.
- **Submitter** is a user, or email (an anonymous user) who submitted a form.
- **Index** is an **`AssignRepeatIndex`**, if the task assign repeat action is used with tasks.
- **Item** is an **`AssignRepeatItem`**, if the task assign repeat action is used with tasks.
- **Route Result** is a **`route`** as per the collaboration job's job definition.
- **Attempts** is the number of attempts for the action.
- **Created** is the day and time the action's execution started.
- **Finished** is the day and time the action's execution finished.
- **Duration** is the time it has taken to complete the action.

4. Click an action name to view extra information, such as:



- **Type** is a type of the action.
- **Job Action Key** is a unique action code
- **Sequence** is a sequence of the action as per the job definition
- **Redirect Next Task** specifies if the action has a redirect next task or not.
- **Form Transaction** shows the tracking code of the transaction associated with this action, which only appears for Job Task Assign or Job Form Start actions.
- **Action Service** shows the name and the version of a action service, which is invoked during this action.
- **Action Executed Time** is the time it has taken to complete the action.

5. Click the **Events** tab to view all the events related to this action.
6. Click **Edit** to check or update a service, which is invoked during this job step.
7. Click **View Transaction Details** to check the details of the transaction associated with this action.
8. Click **View PDF Receipt** to view a form as PDF receipt.

9. Click **Close**.

Next, learn how to view collaboration job submissions.

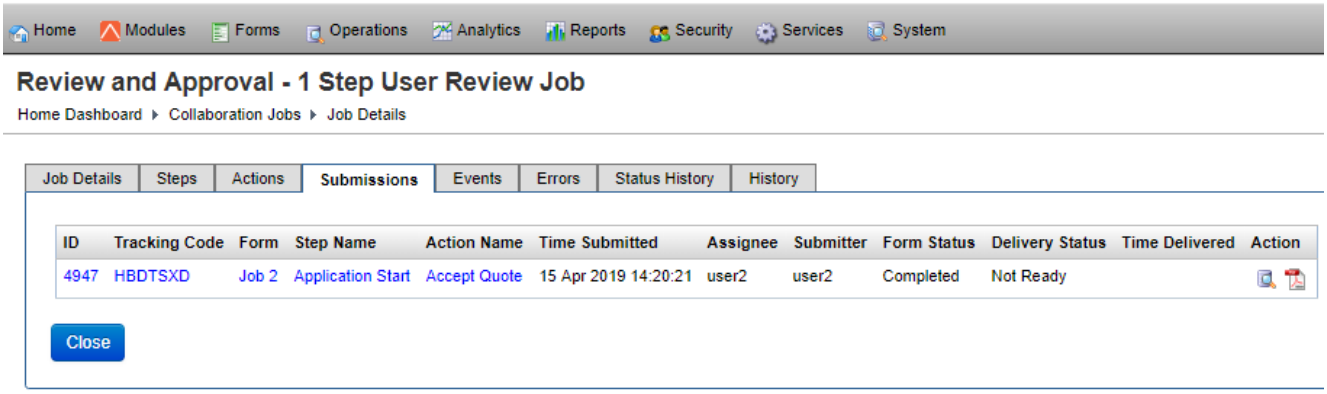
View Collaboration Job Submissions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view transactions created for each action executed within a collaboration job.

To view the transactions within a collaboration job:

1. Select **Operations > Collaboration Jobs**.
2. Locate a collaboration job and click **View**.
3. Select the **Submissions** tab to see the collaboration job's transactions and their basic information such as:



- A transaction ID and a tracking code.
- Form
- Step name
- Action name
- Time submitted
- Assignee
- Submitter
- Form status

  A form status can be one of the following:

- Abandoned - transactions that have been abandoned by the user, or based on the data retention policy.
- Assigned - a task which is waiting to be completed by a user.
- Completed - transactions that have been successfully submitted and contain all required data.
- Opened - transactions that have been started by a user. When a transaction is created by a user (by starting a new form), the initial form status is set to opened. This status will be changed once the user saves, cancels, or submits the transaction.
- Saved - transactions that have been saved by the user or automatically by the system.
- Submitted - transactions that have been submitted but may require additional information (attachments, payment, etc.). Once the required data is collected, the transaction status will change to Completed. This status is not common for Maestro forms as all information, such as attachments and payments, are collected prior to the user submitting the form. Most Maestro forms would skip the Submitted status and be marked as Completed after the form is submitted by the user. This is a legacy feature.
- Expired - a form used by this transaction has expired and is no longer available.
- Delivery status
- Time delivered

4. Click **ID**, **Tracking Code** or **View Transaction Details** to check the details of the transaction.
5. Click **View PDF Receipt** to view a form as PDF receipt.

Next, learn how to view collaboration jobs events.

View Collaboration Job Events

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view events related to each collaboration job. Events are added when the job actions are executed or processed, including:

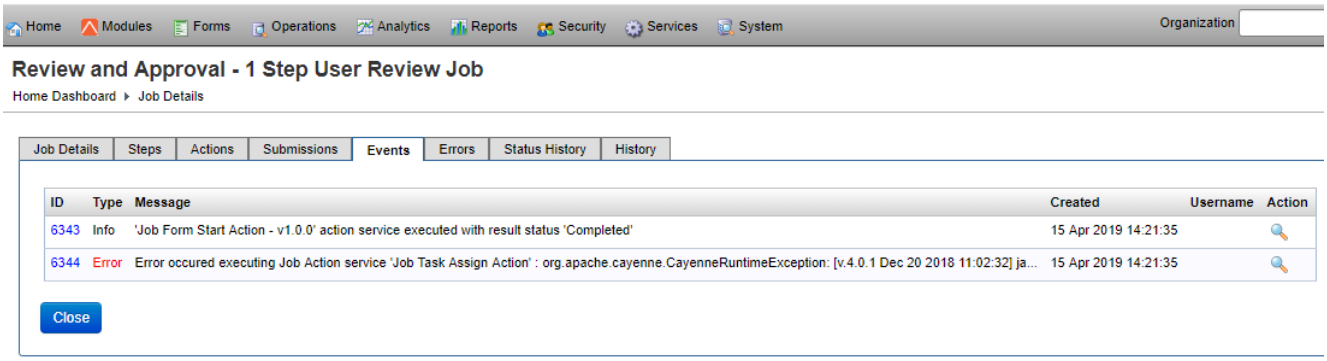
- **Info** events are generated when the action proceeds normally. The action status is Completed, Invoked, Assigned, Pending, and so on.
- **Error** events are generated where and exception is thrown by the action processing or the result of the action processing is determined to be an error. Errors will also appear in the Errors tab.
- Custom Groovy Job Actions can write either an error or info event to the JobEventLog Server.

Various services, invoked during a collaboration job execution, can write to the event log using the following Groovy code:

```
// logEvent closure
def logEvent = { msg ->
        if ( isLogEvents ) {
                // Logs to Job Action Event Tab
                jobEventLogService.logInfoEventWithAction((String) msg, jobAction)
        }
}
```

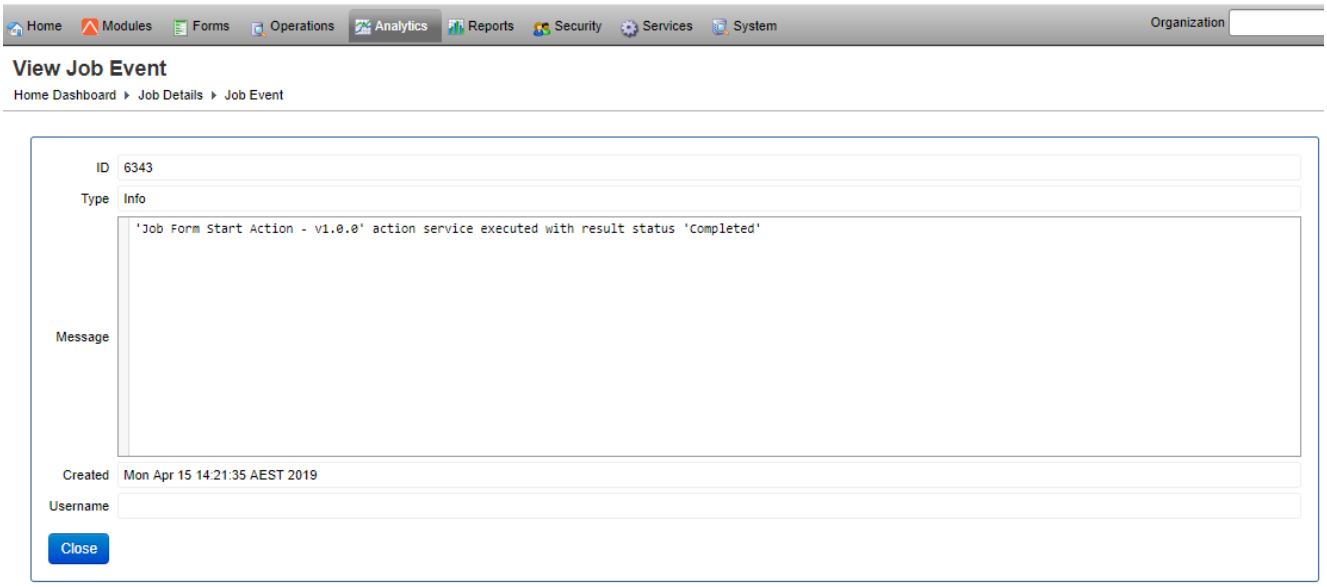
To view the events related to a collaboration job:

1. Select **Operations > Collaboration Jobs**.
2. Locate a collaboration job and click **View**.
3. Select the **Events** tab to see the collaboration job's events and their basic information such as:



- **ID** is the event ID. Click **ID** to see the job event details.
- **Type** is the type of the event, which can be **info** or **error**.
- **Message** is a meaningful description of the event.
- **Created** is a timestamp of the event.
- **Username** is a user of the event.

4. Click **View** to see the job event details.



5. Click **Close**.

Manager stores collaboration job events in a separate table, so you can't find them in the event log.

You can also see errors that occurred during a collaboration job execution. Generally, errors are of greater severity than events.

Next, learn how to view collaboration job errors.

View Collaboration Job Errors

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

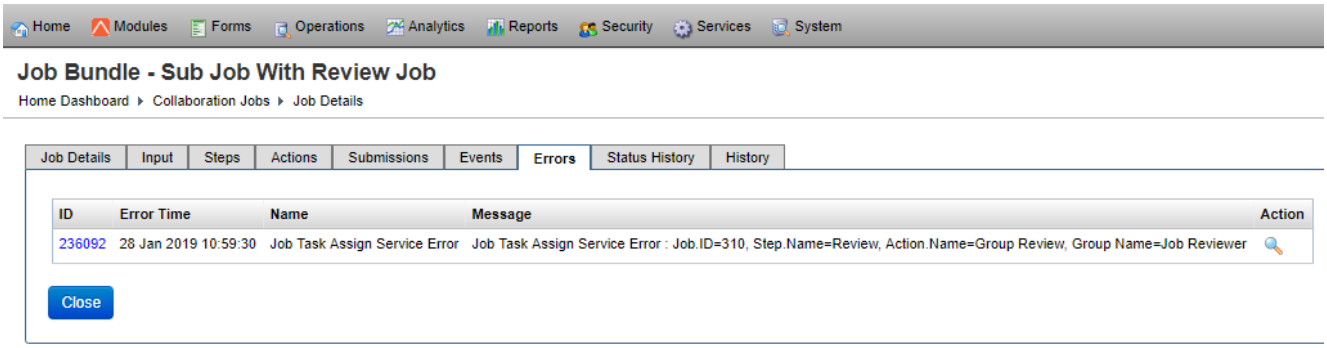
Manager allows you to view errors that occurred during execution of each collaboration job.

> If a collaboration job doesn't have errors, this tab is not shown.

To view the errors related to a collaboration job:

1. Select **Operations > Collaboration Jobs**.
2. Locate a collaboration job and click **View**.
3. Select the **Errors** tab to see the collaboration job's errors.

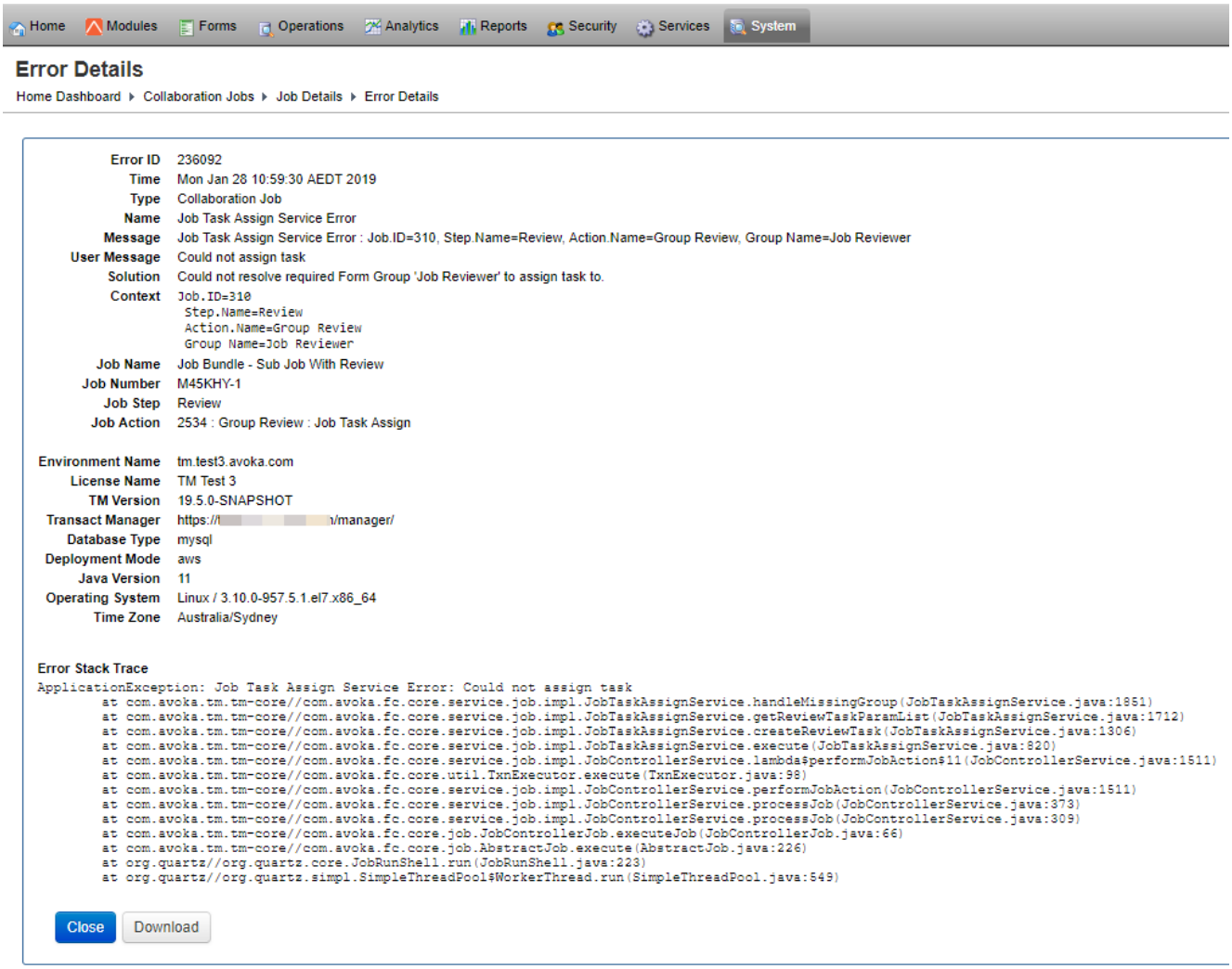
> This tab is not visible if the collaboration job doesn't have any errors.



- **ID** is the error ID. Click **ID** to see the job error details.
- **Name** is a meaningful description of the error.
- **Message** is a detailed error message.

4. Click **View** to see the error details including error stack trace, which is especially useful in problem

troubleshooting.



Manager stores collaboration job errors along with all other errors, so you can find them in the error log.

You can also see events that occurred during a collaboration job execution. Generally, errors are of greater severity than events.

Next, learn how to view collaboration job status history.

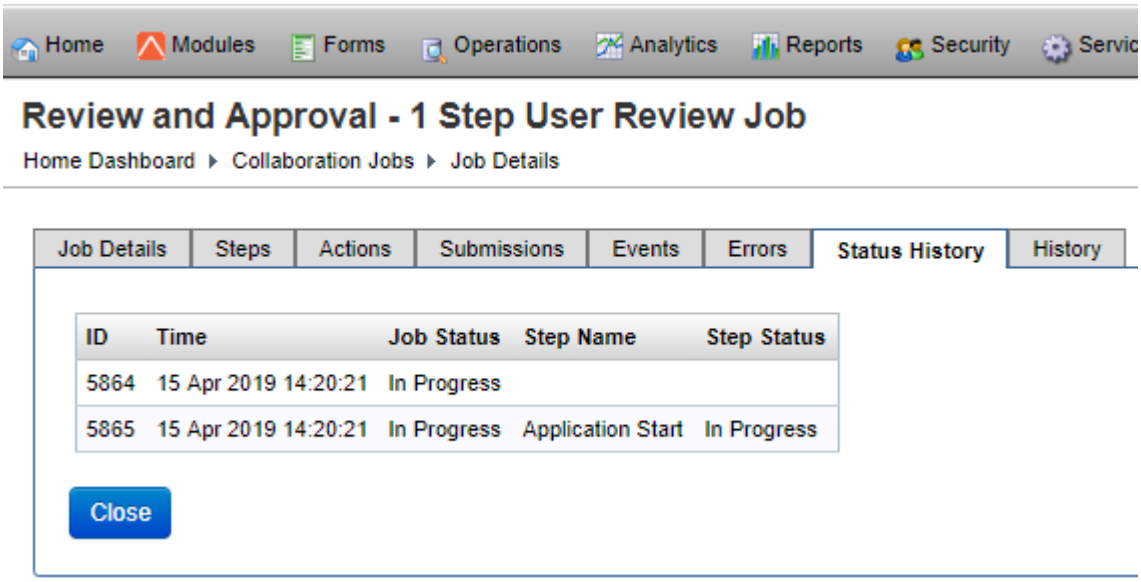
View Collaboration Job Status History

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view status history of each collaboration job.

To view the status history of a collaboration job:

1. Select **Operations > Collaboration Jobs**.
2. Locate a collaboration job and click **View**.
3. Select the **Status History** tab to see the collaboration job's status history and its information, such as:



- **ID** is a unique status ID.
- **Time** is the day and time the job or step status was changed.
- **Job Status** is a status of a collaboration job.

  A job status can be one of the following:

  - In Progress - the collaboration job is active and in progress.
  - Completed - the collaboration job has been completed. This happens when a endpoint step has completed.
  - Expired - this happens when an endpoint expiry collaboration job has been automatically expired by Journey Manager.
  - Cancelled - the collaboration job has been manually canceled by an administrator.
  - Error - the collaboration job has a critical or unrecoverable error that requires an administrator intervention before it can be run again.
  - Paused - the collaboration job has been paused temporarily.

- **Step Name** is a name of a step, which is a part of a job definition of the collaboration job.
- **Step Status** is a current status of a step.

  A job step status can be one of the following:

  - In Progress- the step is in progress now.
  - Completed - the step is completed.
  - Expired - the step is expired.
  - Cancelled - the step has been manually cancelled by an administrator.

- Click **Close**.

Next, learn how to view collaboration job history.

View Collaboration Job History

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

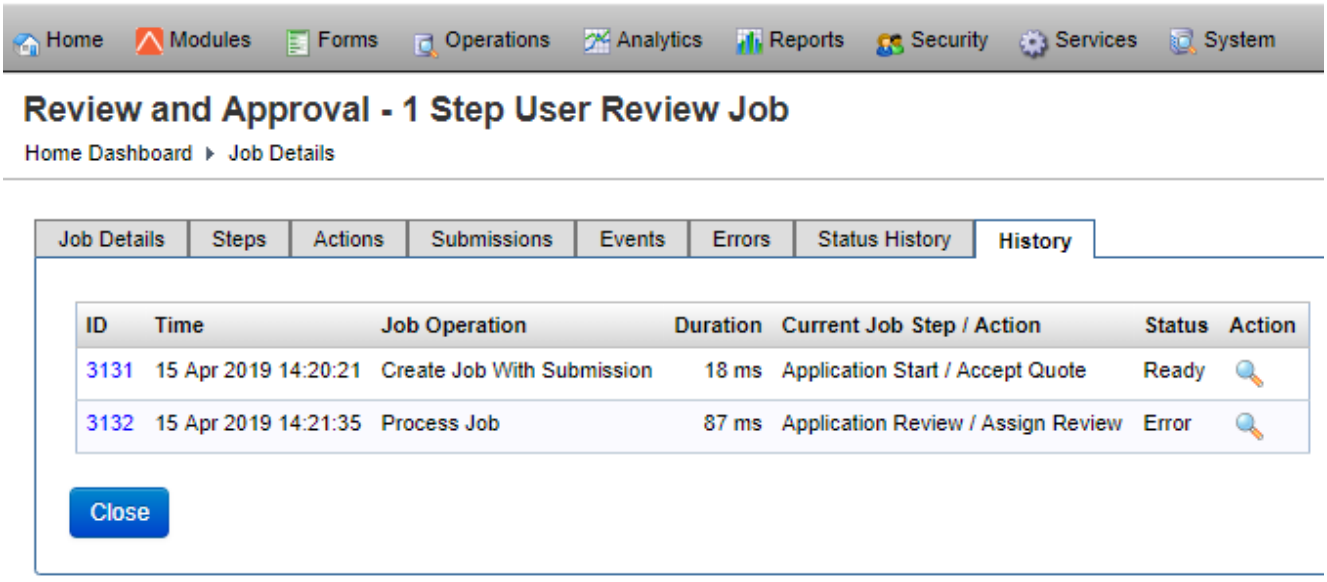
Manager allows you to view history of each collaboration job.

A collaboration job is a state machine that can be represented with a graph consisting of steps and actions. A collaboration job has life cycle events, such as action processing and submission's callbacks, that take the collaboration job from one state to another. The Action Processing may execute one or more actions in one Job Processing cycle. Similarly, a callback, such as a task submission, changes the collaboration job's action status for the task assign from **Assigned** to **Completed** and the **Task Wait** status to **Ready**.

> A collaboration job that is to be processed immediately true will have less history state changes because the callback state changes and action processing are combined.

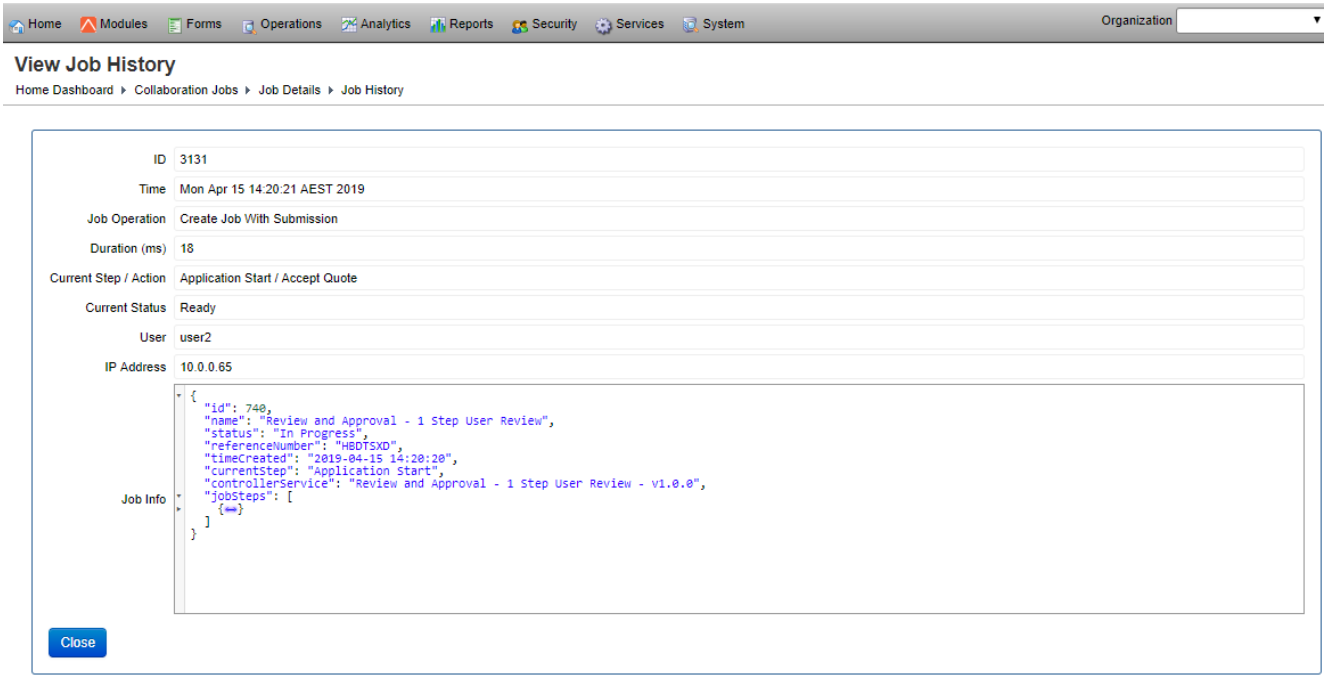
To view the history of a collaboration job:

1. Select **Operations > Collaboration Jobs**.
2. Locate a collaboration job and click **View**.
3. Select the **History** tab to see the collaboration job's history:



- **ID** is a unique job operation ID which can be a step or an action.
- **Time** is the day and time the job operation was invoked.
- **Job Operation** is the description of the step or the action executed as a part of the collaboration job.
- **Duration** is the time it has taken to complete the job operation.
- **Current Job Step / Action** is the name of the step and the action executed as a part of the collaboration job.
- **Status** is the status of the job operation that can be the status of the step or the action.

4. Click **View** to see the detailed information about a selected job operation.



The **Job Info** shows the collaboration job graph as an object in **JSON**[1] format.For more information, see %=TransactVariables.JobDefinitionLowCase%].

[1]JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

```
{
	"id": 740,
	"name": "Review and Approval - 1 Step User Review",
	"status": "In Progress",
	"referenceNumber": "HBDTSXD",
	"timeCreated": "2019-04-15 14:20:20",
	"currentStep": "Application Start",
	"controllerService": "Review and Approval - 1 Step User Review - v1.0.0",
	"jobSteps": [
		{
			"id": 2584,
			"name": "Application Start",
			"status": "In Progress",
			"type": "start",
			"timeCreated": "2019-04-15 14:20:20",
			"currentAction": "Accept Quote",
			"jobActions": [
				{
					"id": 4257,
					"name": "Accept Quote",
					"status": "Ready",
					"type": "Job Form Start",
					"actionService": "Job Form Start Action - v1.0.0",
					"timeCreated": "2019-04-15 14:20:20",
					"submission": {
						"id": 4947,
						"trackingCode": "HBDTSXD",
						"formStatus": "Completed",
						"receiptStatus": "Ready",
						"deliveryStatus": "Not Ready",
						"versionNumber": "18.5-Branch",
						"formSpace": "Work Spaces",
						"timeRequest": "2019-04-15 14:20:18",
						"timeFormCompleted": "2019-04-15 14:20:20",
						"contactEmailAddress": "test@test.com",
						"userLoginName": "user2"
					}
				}
			]
		}
	]
}
```


Next, learn how to view collaboration jobs.

Collaboration Job Examples

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager comes with several collaboration job examples that you can use to create your new workflows. Every example contains forms, form spaces and services configurations available for you to download and run in Manager.

You can start with the basic examples, including:

- One step review collaboration jobs
- Multi step group review collaboration jobs
- Customer onboarding collaboration jobs - form bundle
- Maestro customer onboarding collaboration jobs - form bundle

Then, you can check the advanced examples:

- Dynamic customer onboarding collaboration jobs
- Step expiry
- Anonymous collaboration jobs
- Asynchronous step execution
- Task assign repeat action

Before you can use these examples, you must create users with specific roles as explained below.

To create the users:

1. Log on to Manager as an administrator.

2. Copy the **Groovy**[1] script into the Groovy console.

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

```
/**
 * Adds Users For the Collaboration Jobs Tests
 * Copy and Paste into the Groovy Console
 * Developed for Transaction Manager 4.3.0
 */

import com.avoka.fc.core.dao.ClientDao
import com.avoka.fc.core.dao.DaoFactory
import com.avoka.fc.core.dao.GroupDao
import com.avoka.fc.core.dao.PortalDao
import com.avoka.fc.core.dao.RoleDao
import com.avoka.fc.core.dao.UserAccountDao
import com.avoka.fc.core.entity.Client
import com.avoka.fc.core.entity.ClientUser
import com.avoka.fc.core.entity.Group
import com.avoka.fc.core.entity.Portal
import com.avoka.fc.core.entity.PropertyType
import com.avoka.fc.core.entity.Role
import com.avoka.fc.core.entity.UserAccount
import com.avoka.fc.core.entity.UserRole
import com.avoka.fc.core.security.ISecurityManagerService
import com.avoka.fc.core.service.ServiceFactory
import com.avoka.fc.core.service.UserService

final EMAIL_TESTTEAM = "testemailgroup@yourcompany.com"
final PASSWORD_COMMON = "replace with password"
final JOB_APPLICANT = "Job Applicants"
final JOB_REVIEWER = "Job Reviewers"
final JOB_MANAGER = "Job Managers"
final MAGUIRE = "Maguire"

PortalDao portalDao = DaoFactory.getPortalDao()
Portal portal = portalDao.getPortalByName(MAGUIRE)

UserAccountDao userAccountDao = DaoFactory.getUserAccountDao()

ISecurityManagerService securityManagerService = Ser-
viceFactory.getSecurityManagerService(portal)
GroupDao groupDao = DaoFactory.getGroupDao()
UserService userService = ServiceFactory.getUserService(portal)

def userList = [
```

```
        ["firstName": "Joe", "lastName": "Public"],
        ["firstName": "Anne", "lastName": "Applicant", "group": JOB_APPLICANT],
        ["firstName": "Roger", "lastName": "Reviewer", "group": JOB_REVIEWER],
        ["firstName": "Mike", "lastName": "Applicant", "group": JOB_MANAGER],
        ["firstName": "Cat", "lastName": "Coordinator", "group": JOB_MANAGER]
]


for (Map userMap : userList) {
    UserAccount userAccount = userAccountDao.getUserAccountForLogin((String) user-
Map.firstName)

    if ( userAccount==null) {

        Map<String, String> profileMap = new HashMap<String, String>();
        profileMap.put((String) PropertyType.USER_Property_Given_Name, (String) user-
Map.firstName);
        profileMap.put((String) PropertyType.USER_Property_Family_Name, (String)
UserMap.lastName);
        profileMap.put((String) PropertyType.USER_Property_Email, (String) EMAIL_
TESTTEAM);



        userAccount = securityManagerService.createUserAndProfile((String) user-
Map.firstName,
                (String) EMAIL_TESTTEAM,
                (String) PASSWORD_COMMON,
                (String) userMap.firstName,
                (String) userMap.lastName,
                "",
                profileMap,
                "",
                UserAccount.USER_TYPE_LOCAL,
                false)

        if (userMap.firstName == "Cat") {
            ClientDao clientDao = DaoFactory.getClientDao()
            Client client = clientDao.getClientByName(MAGUIRE)

            RoleDao roleDao = DaoFactory.getRoleDao()
            Role role = roleDao.getRoleForName("Maguire Staff")
```

```
            ClientUser clientUser = new ClientUser()
            clientUser.setClient(client)
            clientUser.setUser(userAccount)

            UserRole userRole = new UserRole()
            userRole.setUser(userAccount)
            userRole.setRole(role)
        }

        Group group = groupDao.getGroupForName((String) userMap.group)
        if (group != null) {
            List<String> groupIdList = new ArrayList<String>()
            groupIdList.add(""+group.id)
            userService.updateGroups(userAccount.id, groupIdList)
        }     else {
            println "Cannot link group User: " + userMap.firstName + " Group: " +
userMap.group
        }

        userService.commitChanges()
        println "Create User: " + userMap.firstName

    } else {
        println "User: " + userMap.firstName + " already exists"
    }
}
```

3. Update the script's two lines below with your test email address and password.

```
final EMAIL_TESTTEAM = "testemailgroup@yourcompany.com"
final PASSWORD_COMMON = "replace with password"
```

4. Click **Execute Script** to create the users.

Next, learn about one step review collaboration jobs.

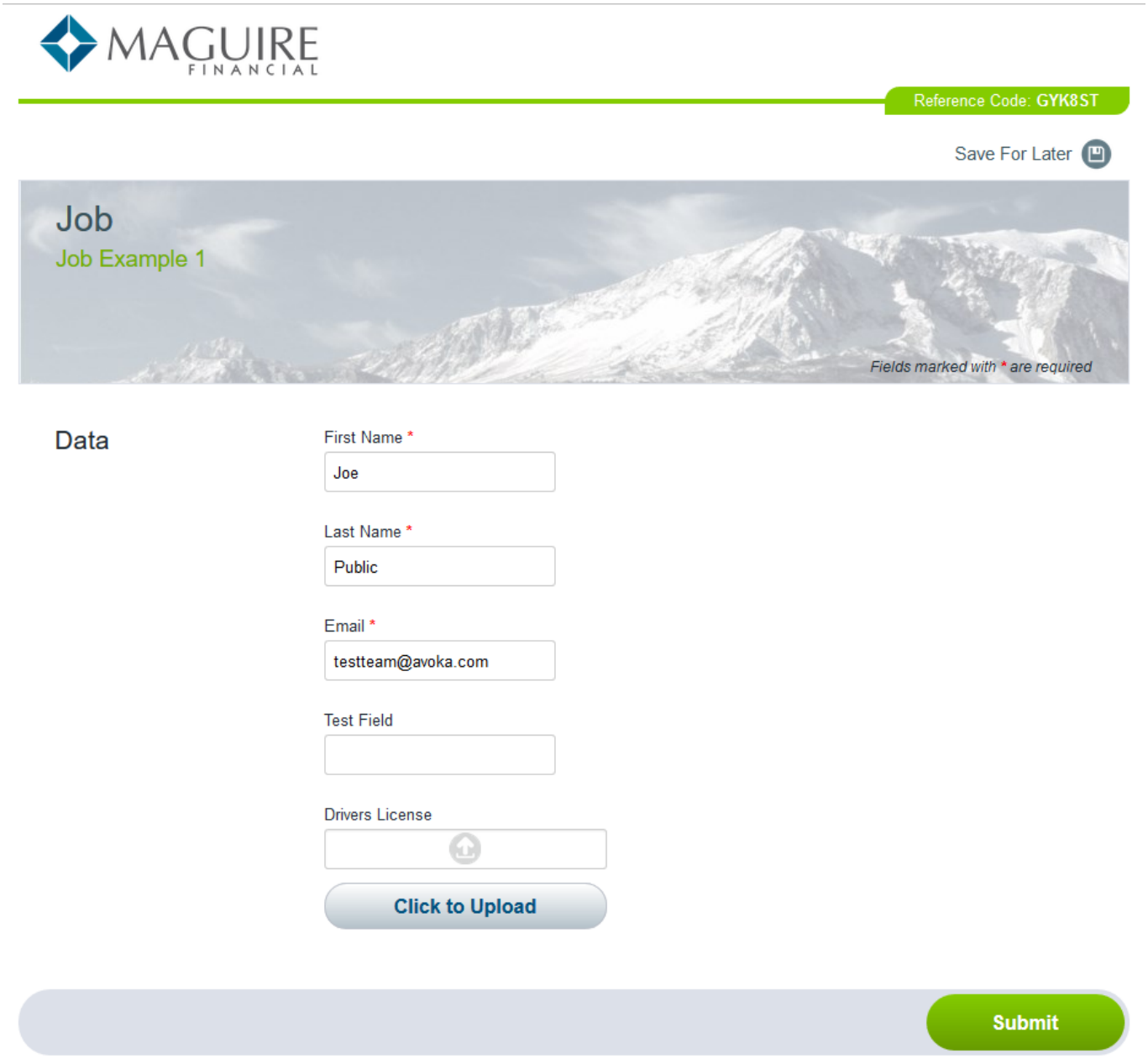
Configure Forms for Use with Collaboration Jobs

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

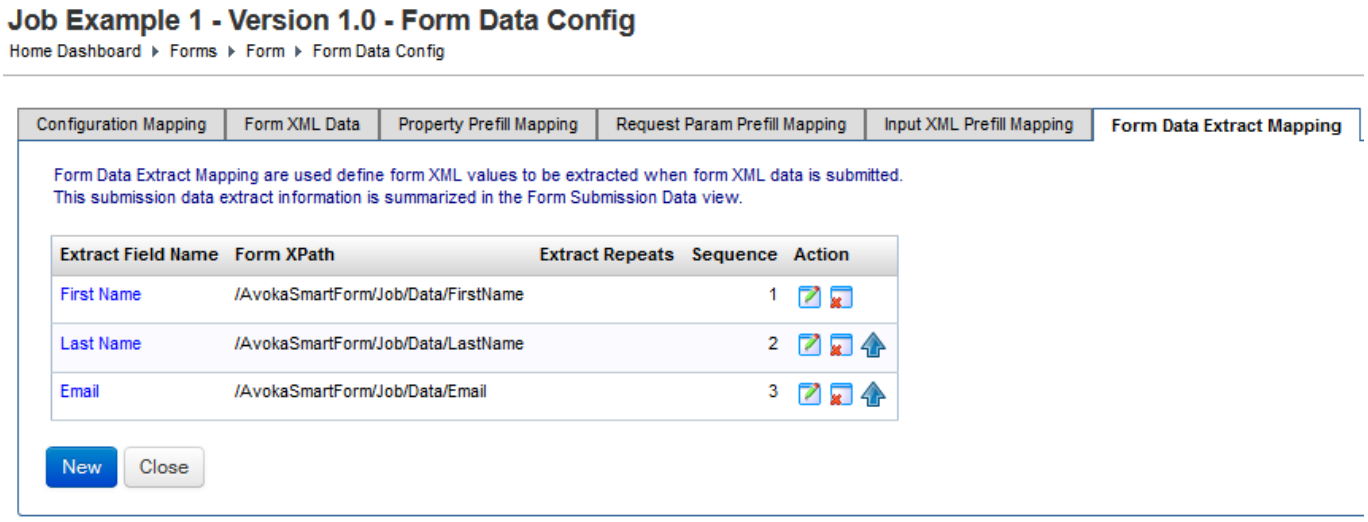
Before you can use your forms with collaboration jobs, you have to configure them in Journey Manager and Journey Maestro. The collaboration jobs use data extracts to reference data used in submitted forms. You can configure data extracts during form design time, which is the recommended approach, or you can map them manually in Manager.

> Data extracts that are configured during form design will overwrite data extracts with the same name in Manager, which happens every time the form is deployed into Manager, so the best practice is to configure them in the form.

Let's explore the data extracts configuration implemented in the Maestro form **Job Example 1**, shown below.



Next, you can chec whether data extracts from this form have been automatically populated and mapped in Manager. To do so, select **Form Dashboard > Data Config > Form Data Extract Mapping** as shown below.



The job definitions of form data extracts are shown below, including field names, how they are referenced in the action properties, and usage examples.

Extract Name Field	Extract Reference	Example
First Name	`$formDataMap.firstName`	`{ "name": "Task Subject", "value": "Job Number: <b>$job.referenceNumber</b> $submission.form.formName From $formDataMap.firstName $formDataMap.lastName" }`
Last Name	`$formDataMap.lastName`	`{ "name": "Task Subject", "value": "Job Number: <b>$job.referenceNumber</b> $submission.form.formName From $formDataMap.firstName $formDataMap.lastName" }`
Email	`$formDataMap.email`	`{ "name": "Task Subject", "value": "Job Number: <b>$job.referenceNumber</b> From $formDataMap.email" }`

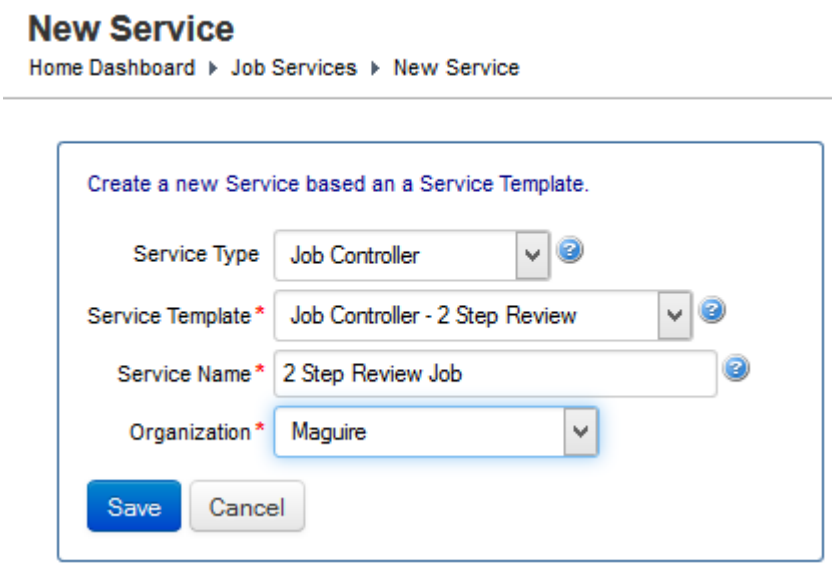
Click on each field to see the data extract mapping information. For example, the **Emain Address** has the **/AvokaSmartForm/GettingStarted/AboutYou/contact/EmailAddress** XPath. Manager passes job context information to the form via the **SFMData.SystemProfile.Job** XML node when the form is rendered.

After you have imported the form, you can configure the following in Manager:

- Form properties to reference them in the job definition for form specific settings.
- Form groups to control access to the form.
- Form delivery methods to enable the job controller to complete collaboration jobs.

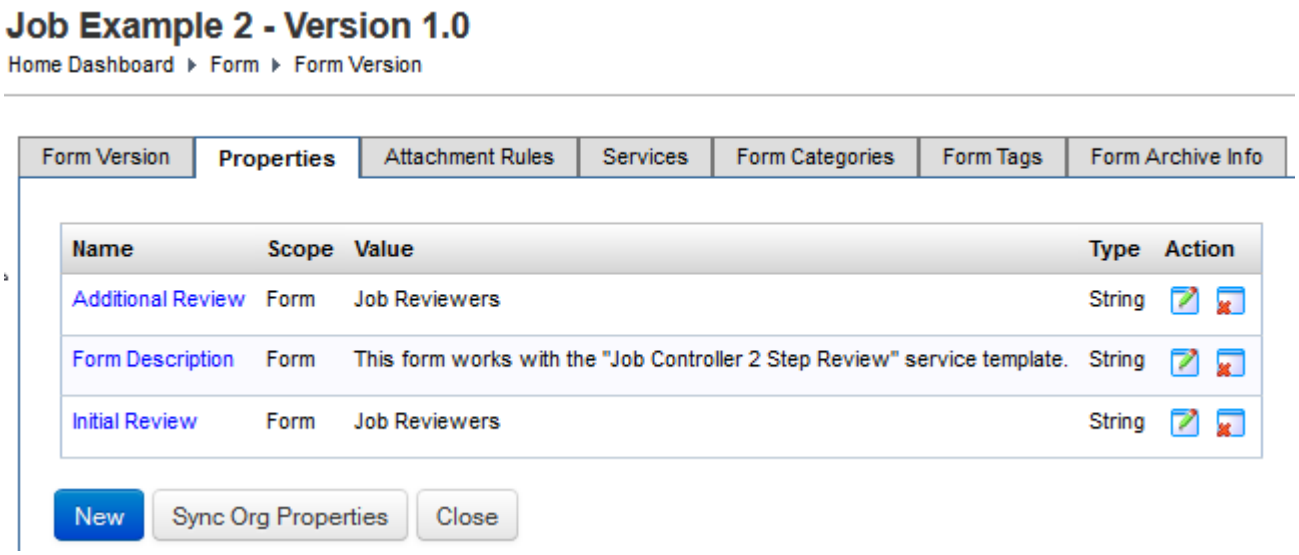
To configure a form:

1. Create a new job controller service instance from available service templates, for example, the **Job Controller - 2 Step Review**.

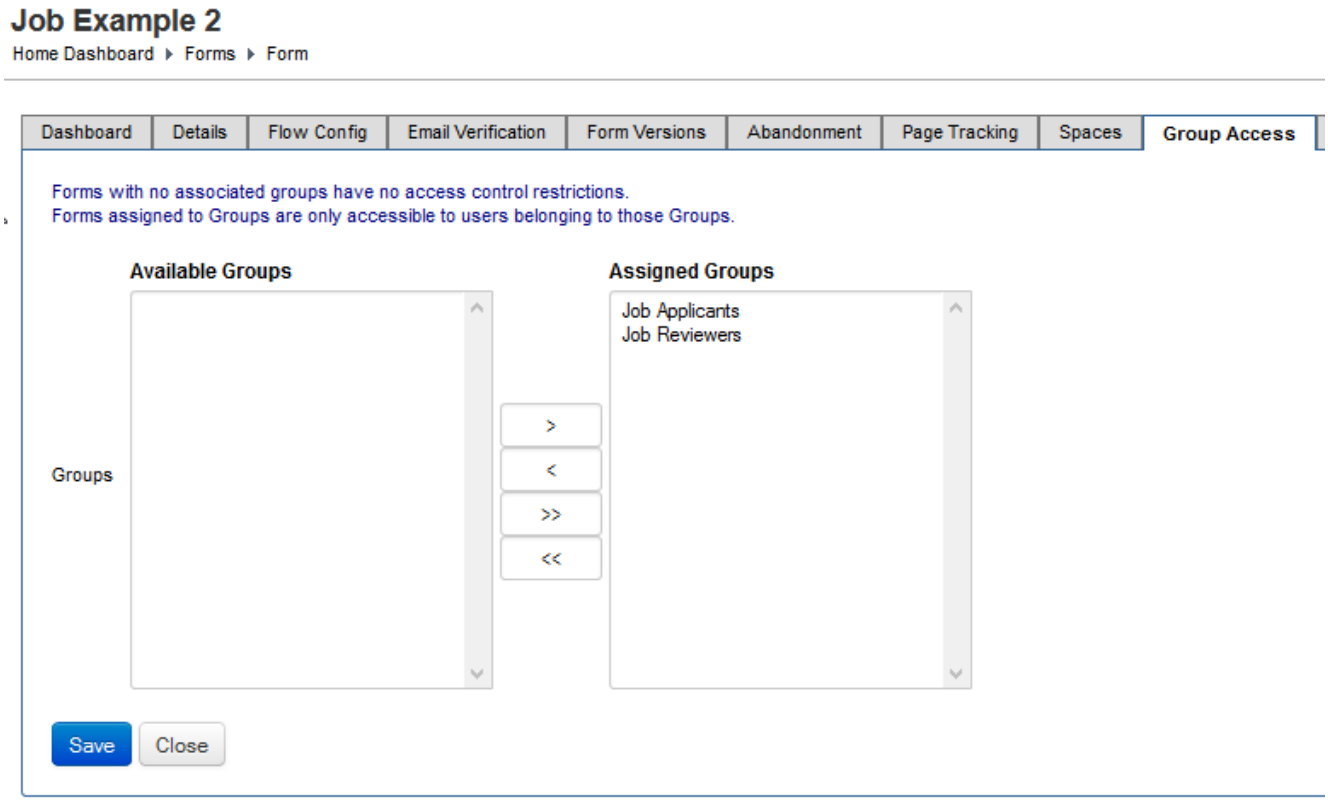


2. Import a form into Manager.
3. Open a form and click **Services**.
4. Select a form prefill data service from the **Form Prefill Data Service** dropdown list, for example, **2 Step Review Job**.
5. Select the **Property** tab to link the reviewer for **Initial Review** and **Additional Review Steps** to be assigned to a form group.
6. Set the **Additional Review** and **Initial Review** property values.

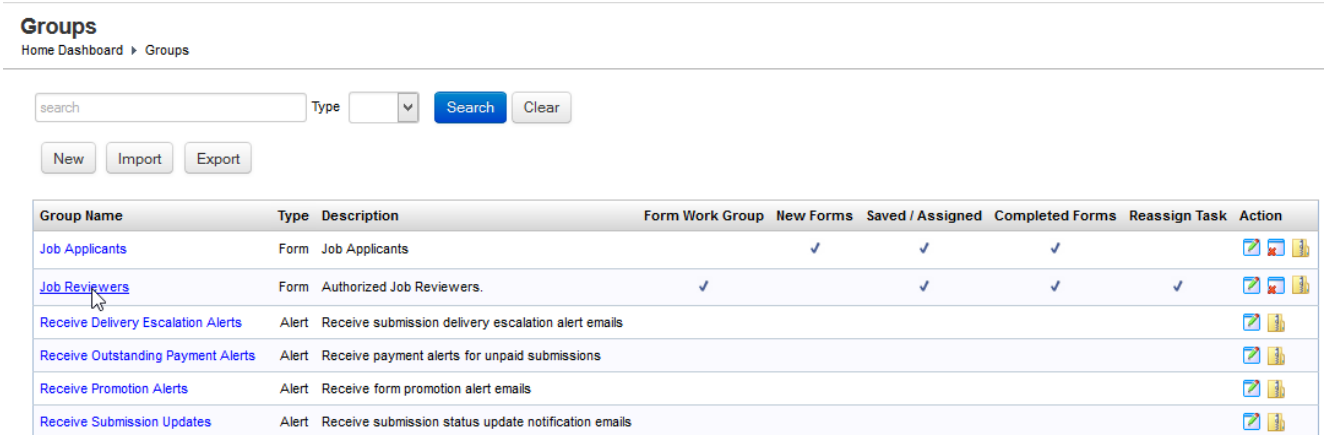
> By default, both will be mapped to the same **Job Reviewers** form group.



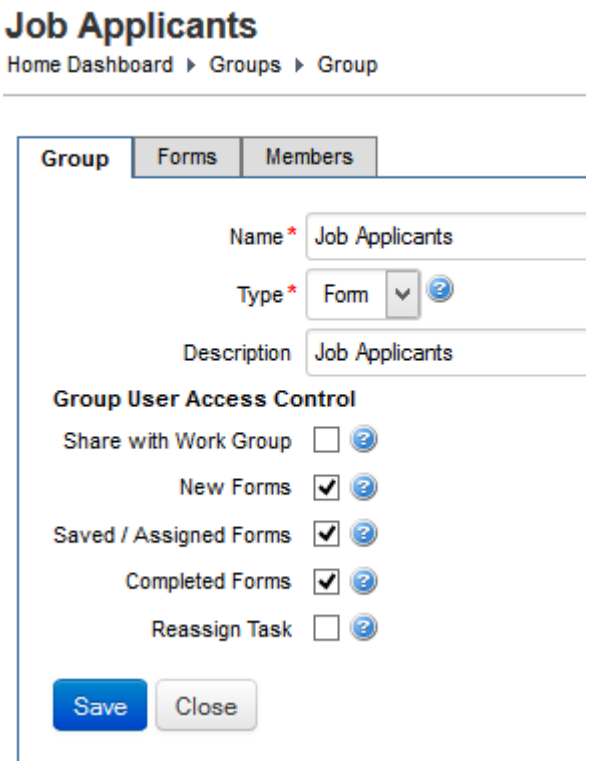
7. Open a form dashboard and select the **Group Access** tab to assign the form groups. Form groups are used to control access the following access to a form. By default, if a form group is not assigned to a form, it will be available to any users belonging to the organization.



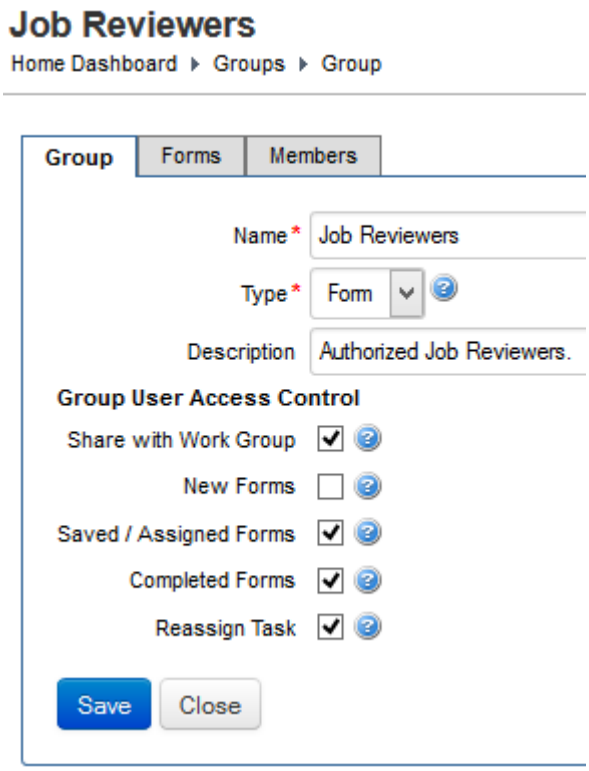
8. Assign a user that you log in with to the **Job Reviewers** form group to review collaboration jobs at the **Initial** and **Additional Review** steps.



9. Select the **Job Applicants** group to restrict users who can start the form, because the **Job Reviewers** can't initiate the form, but they can participate in reviewing the submitted forms.
10. Select the **New Forms** checkbox to allow members of this group to start new forms. Form more information, see configure a group.



11. Assign your user in the **Members** tab.
12. Click the **Job Reviewers** group and select the **Share with Work Group** check box to allow members of this group to participate in tasks, but not start new forms.



13. Select the **Members** tab and assign your user to this group.
14. Assign a job group to the applicable job coordinators' user accounts.
15. Configure a delivery channel, with a default delivery method, for your organization or your form to allow the collaboration job to run and deliver form submissions.
16. Now, your form is configured for collaboration jobs.

A form configuration can be quite extensive and it varies depending on a tool it was built with, as shown below:

- Configure Maestro forms for use with collaboration jobs
- Configure Composer forms for use with Collaboration Jobs

Next, learn how to configure a form.

Configure Maestro Forms for Use with Collaboration Jobs

Maestro | Form Builder | 5.1 & Higher

Maestro forms require configuration before you can use them with collaboration jobs, including:

- How to configure data extracts.
- How to use the collaboration job components that are included in Maestro by default.
- How to set form rules, such as visibility and editability rules, based on collaboration job's steps.
- How to preview your form in Maestro and simulate the collaboration job behavior.

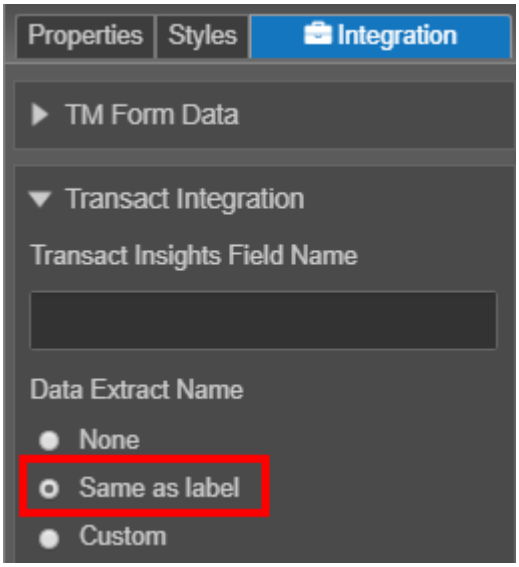
In this example, we use the Maestro form configured with the **Review and Approval – Multi Step Group Review** job controller service in Manager.

Data Extracts

Some form components are required to be saved as data extracts so they can be used with collaboration jobs. For example, the forms used in this example where an applicant enters a loan amount. If the reviewer approves the loan, the **Loan Amount** on the **Request** tab must be checked to validate if it exceeds the threshold of $20,000. In this case, the loan must be also approved by a manager. To achieve this, we will pass the entered loan amount from the form to the job controller service in Manager using the form of data extracts.

To include a component into the data extracts:

1. Select a component to be extracted from the **Structure** panel or directly from the form editor.
2. Click the **Integration** panel.
3. Expand the **Transact Integration** section.
4. Select the **Same as label** radio button.



5. Alternatively, if you want the data extract's name to be different from the form component's name, select the **Custom** radio button and enter a new name in the text field. This depends on the name of the variable used within Manager such as in the job controller service. For more information, see submission data extract.

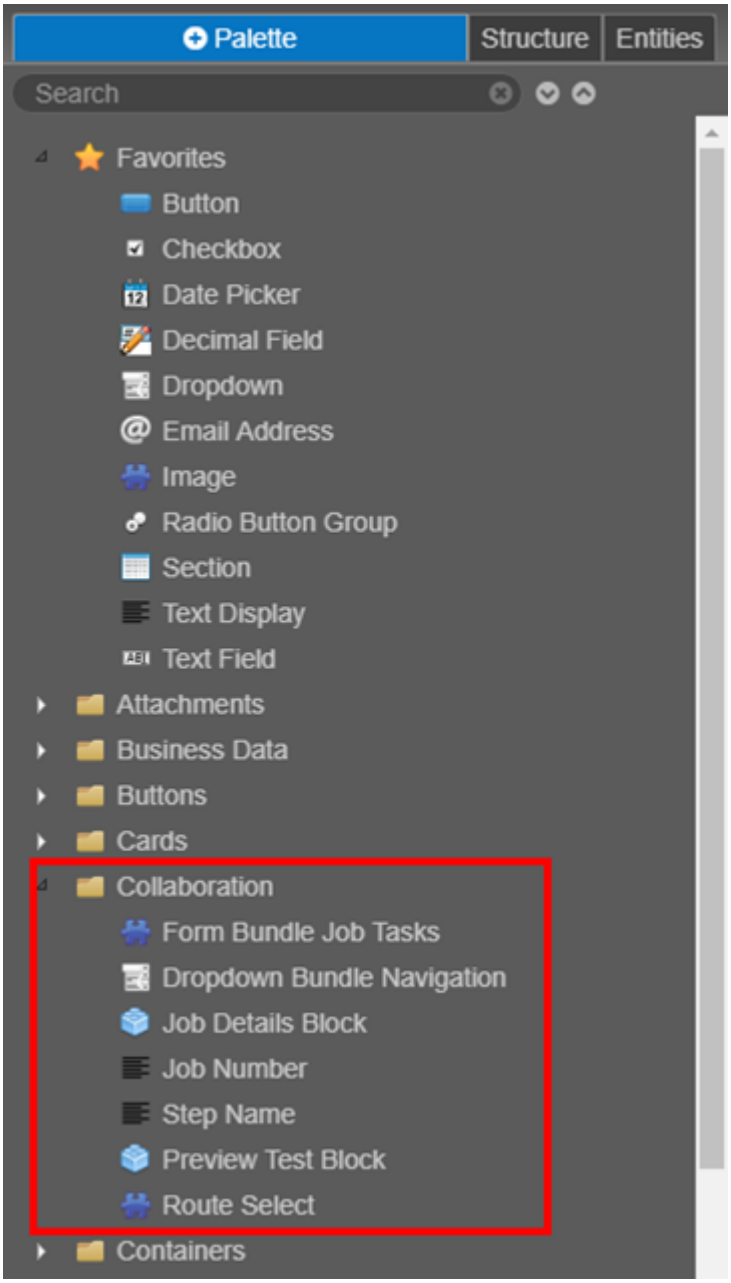
Collaboration Job Components

Collaboration job components have been created to assist the form builder in combining the collaboration jobs features into their forms.

To use a collaboration job component in Maestro:

1. Go to the **Palette** panel.
2. Expand the **Collaboration** folder.

3. Drag and drop a components you need on to your form.



The collaboration job components can be one of the following:

Form Bundle Job Tasks

This component is used for form bundles as a way of navigating and keeping track of forms that are still to be completed. It appears as a set of links for navigating to the applications for products previously selected. It has extra features such as allowing a user to click on only the applications that have not been completed and allowing to download a copy of any completed application.

Dropdown Bundle Navigation

It provides a similar feature as the form bundle job tasks, but it is presented as a simple dropdown.

Job Details Block

This component is for displaying collaboration job details on the form.

To display the collaboration job details:

1. Expand **User Experience.**
2. Click **Form Options.**
3. Selecting the **Include Collaboration Job Details**.
4. Use the **Job Details Block** to customize the location of the collaboration job details within you form or template.

Job Number

This component displays the collaboration job number.

Step Number

This component displays the step's name the collaboration job is currently at.

Preview Test Block

This provides a section in your form that can be used to test the form without having to render the form in Transact Manager. The block is used for testing purposes and can be viewed from the preview mode in the Maestro editor. This allows the form builder to preview the form and make sure that the rules are set up correctly without having to leave Maestro. That is, without the form being first deployed to Manager and associated with a job controller. It is important to note that the **Preview Test Block** will never display in a built / rendered form as it only appears when using the **Maestro Preview** mode.

The following components may be entered:

- Job Reference will be shown in the relevant job details or job number fields. That is, it will appear in the Job Number and Job Details components where they appear on your form, or in the job details in the heading if you have ticked the Include Collaboration Job Details in the Form Options.
- Step Name is quite useful for testing the rules set up for presenting the correct form depending on which step name the form is up to. As this is a dropdown you must first configure the options to mimic the step name values of the job controller, which is also used in the visibility and editability rules configured for this form. To do this:

1. Select the **Properties** panel.
2. Expand the **Dropdown** section.
3. Click **Options**.
4. Enter each of the correct step name values for each row under the **Label** column. Make sure that the first one is left blank which is used for testing when the applicant first submits the application.
5. Click **Save**.

- Available Routes is used for the Route Select dropdown component. It will plug in the correct values in the Route Select dropdown, which is the same as when passed from the job controller. The values must be entered as a pipe delimited string. You will also need to select the suitable step name from the Step Name checkbox above.

Route Select

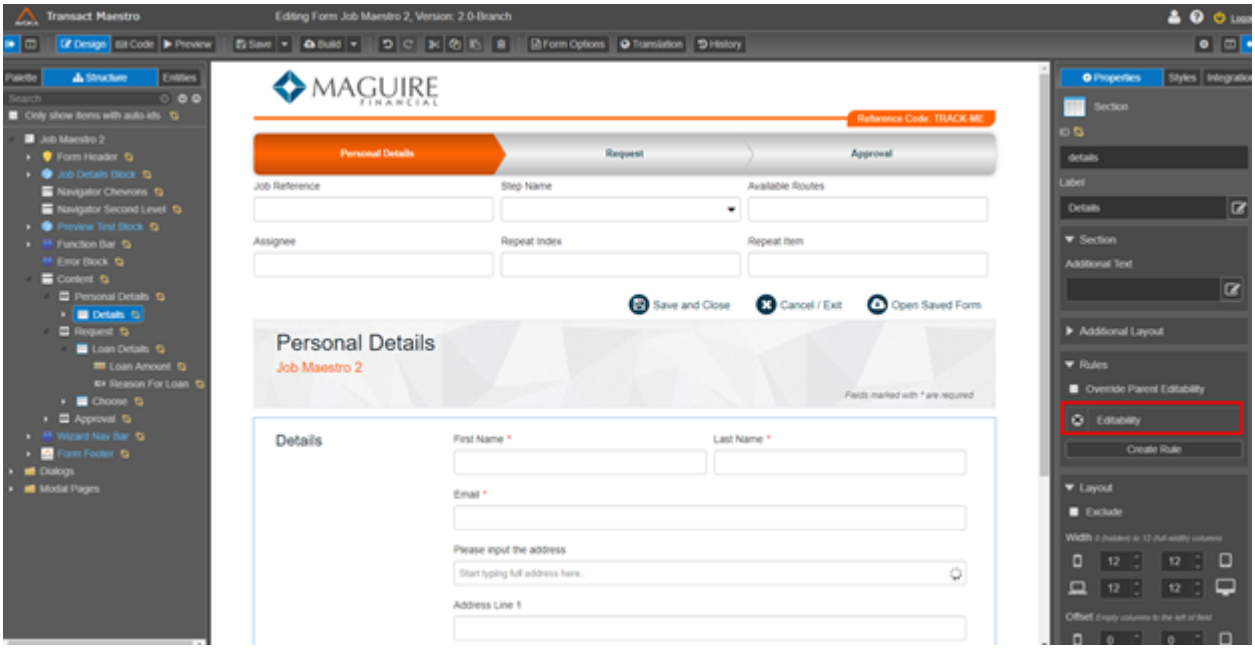
This component is a dropdown that will present to a reviewer or manager reviewing the application a list of options. A review or manager will choose an option to indicate their decision. This component will dynamically plug-in the values for the options from the job controller associated with the form. This may be useful but there has been a preference towards using radio button group component instead. For more information, see below.

Form Rules

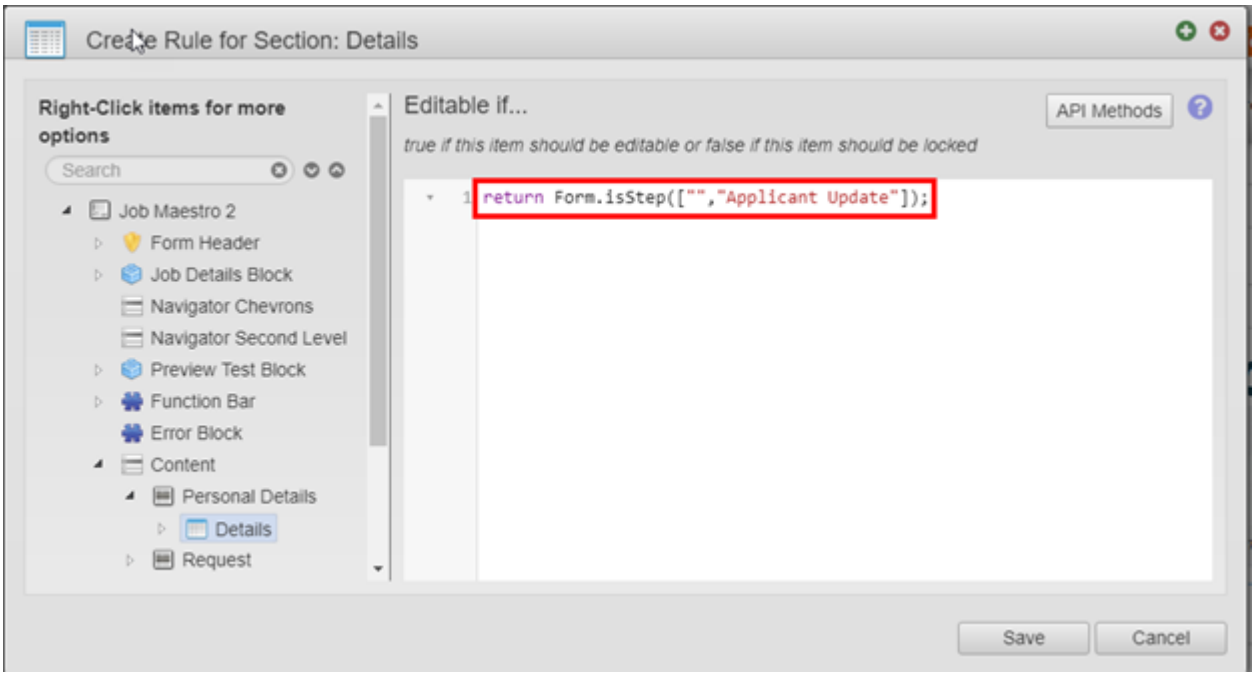
A collaboration job workflow is often required to be shared amongst the relevant parties, such as an applicant, a reviewer and a manager. For the form to be presented suitably to each user, you need to specify the form rules, such as:

- Applicants can enter their personal details but can't approve their own applications. However, they can update their initial request if it was declined.
- Reviewers can indicate their decision. They can view but not update any of the applicant's personal or form request details.
- Managers can indicate their decision which is an additional decision to the reviewer decision. Managers can view but not update any of the applicant's personal or form request details. Managers are like reviewers except they can only view but not update the reviewer's decision.

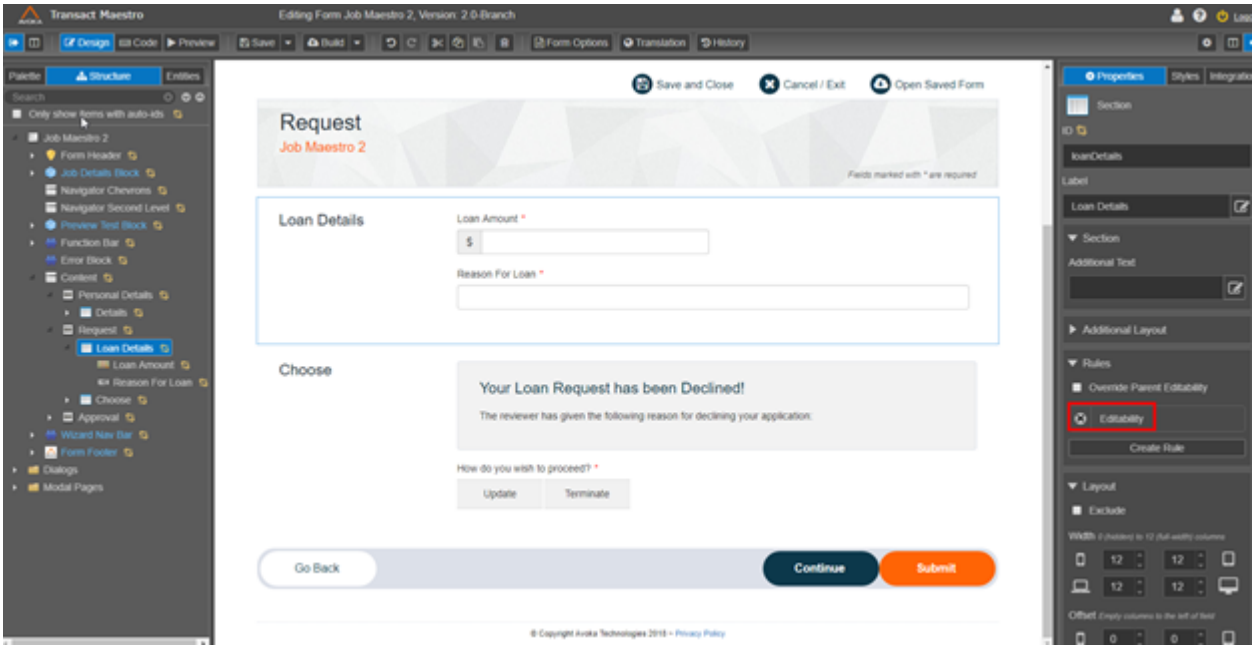
To allow only the applicant to enter and update their personal or form request details, you must create editability rules in your form as shown below.



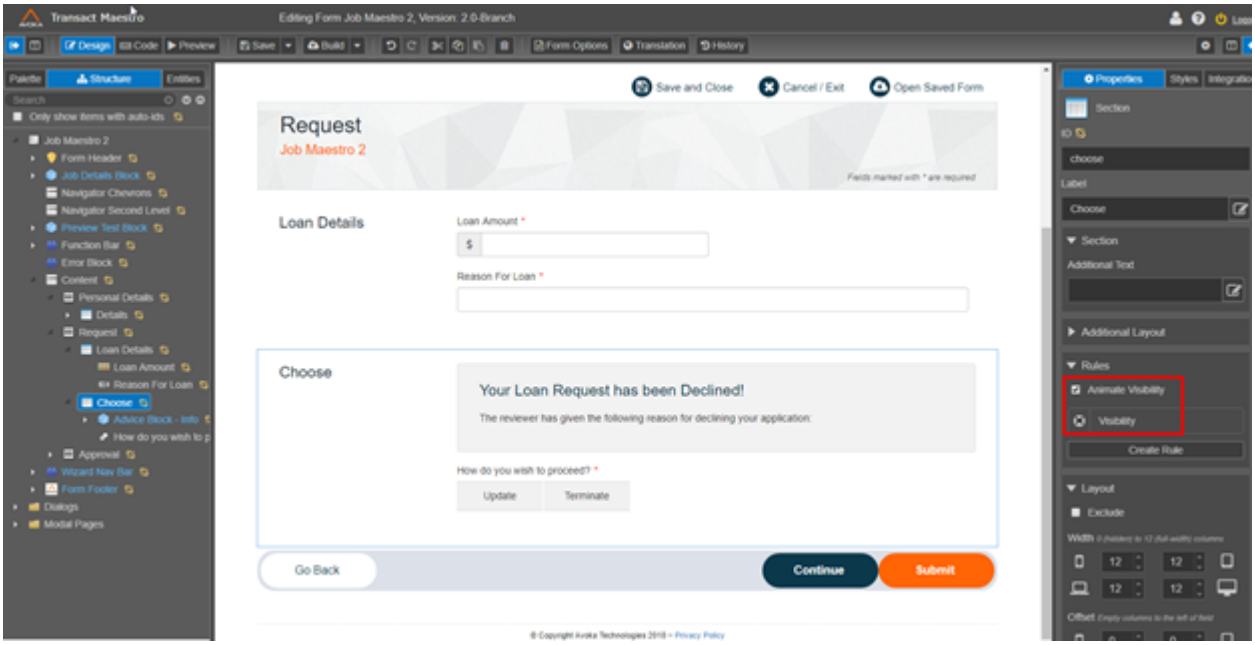
Create the editability rule with the second parameter matching the step name in your intended job controller service configured in Manager. In this example, the step name requesting the applicant to update their details is the **Applicant Update** step.



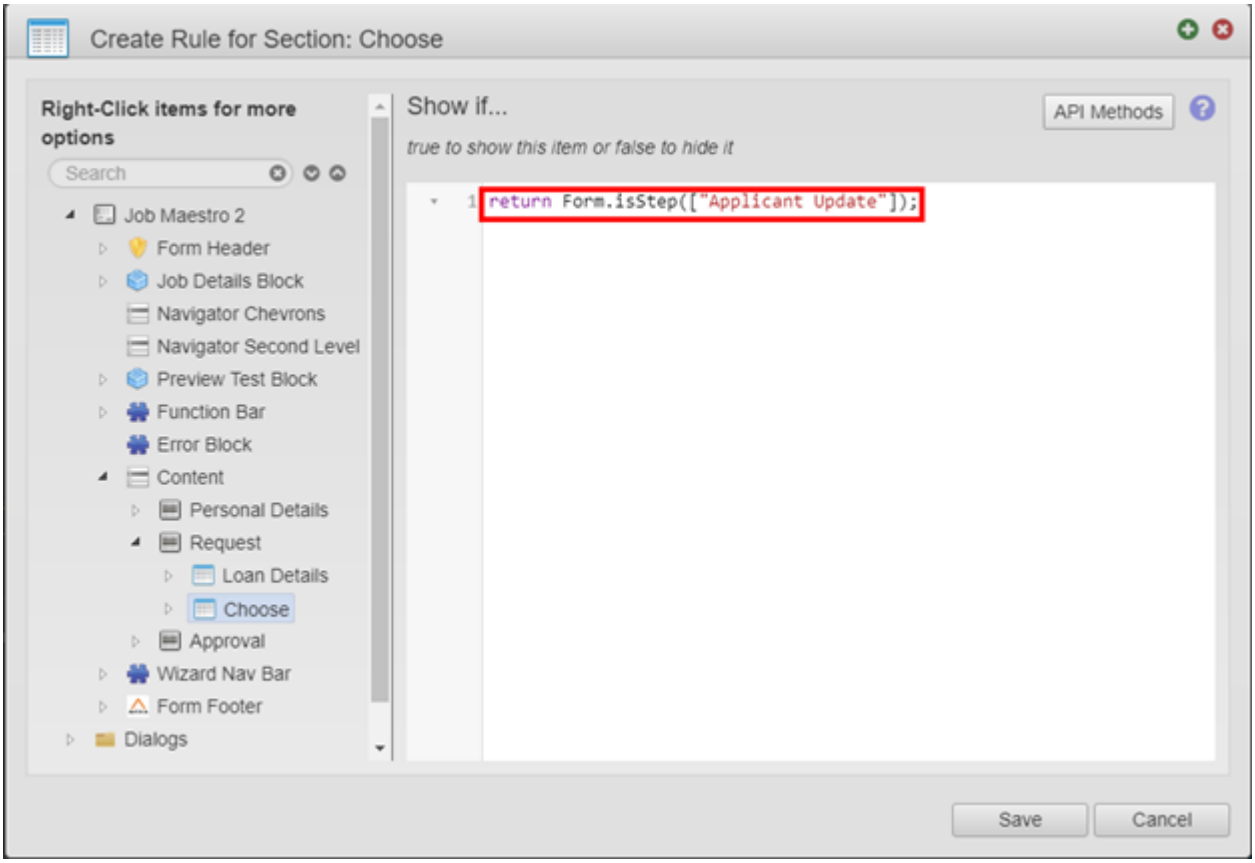
Add this same editability rule to the applicant's request.



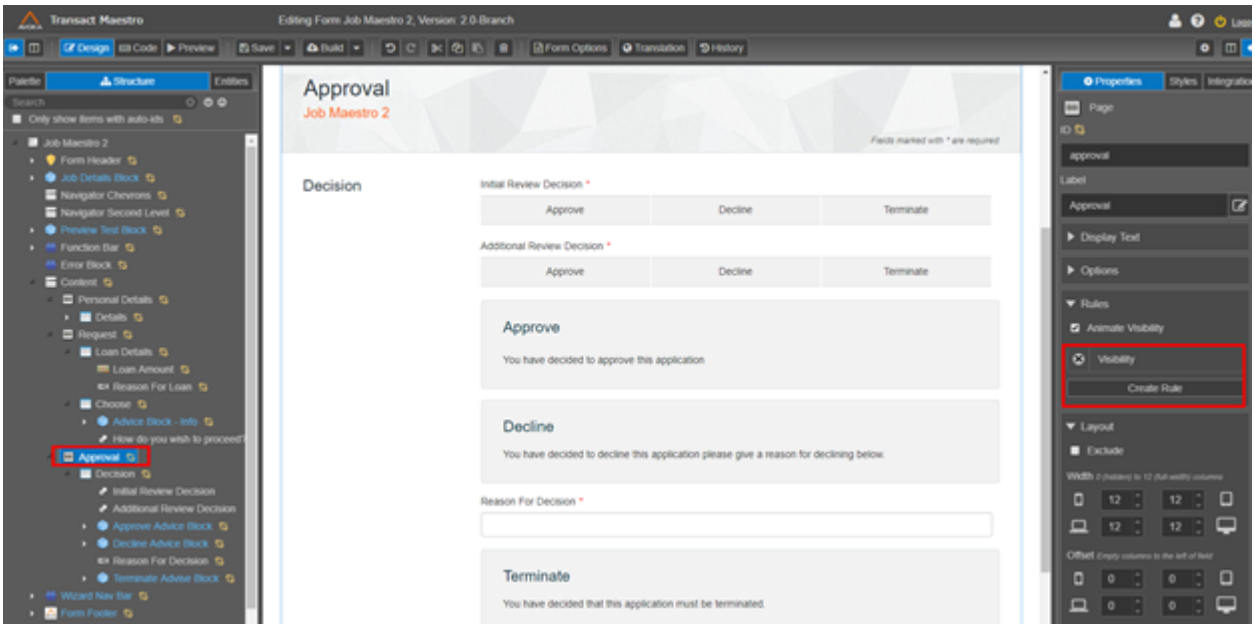
Create the following visibility rule to present the relevant information to the applicants when their application was declined. This rule allows them to choose whether to update or terminate their loan application.



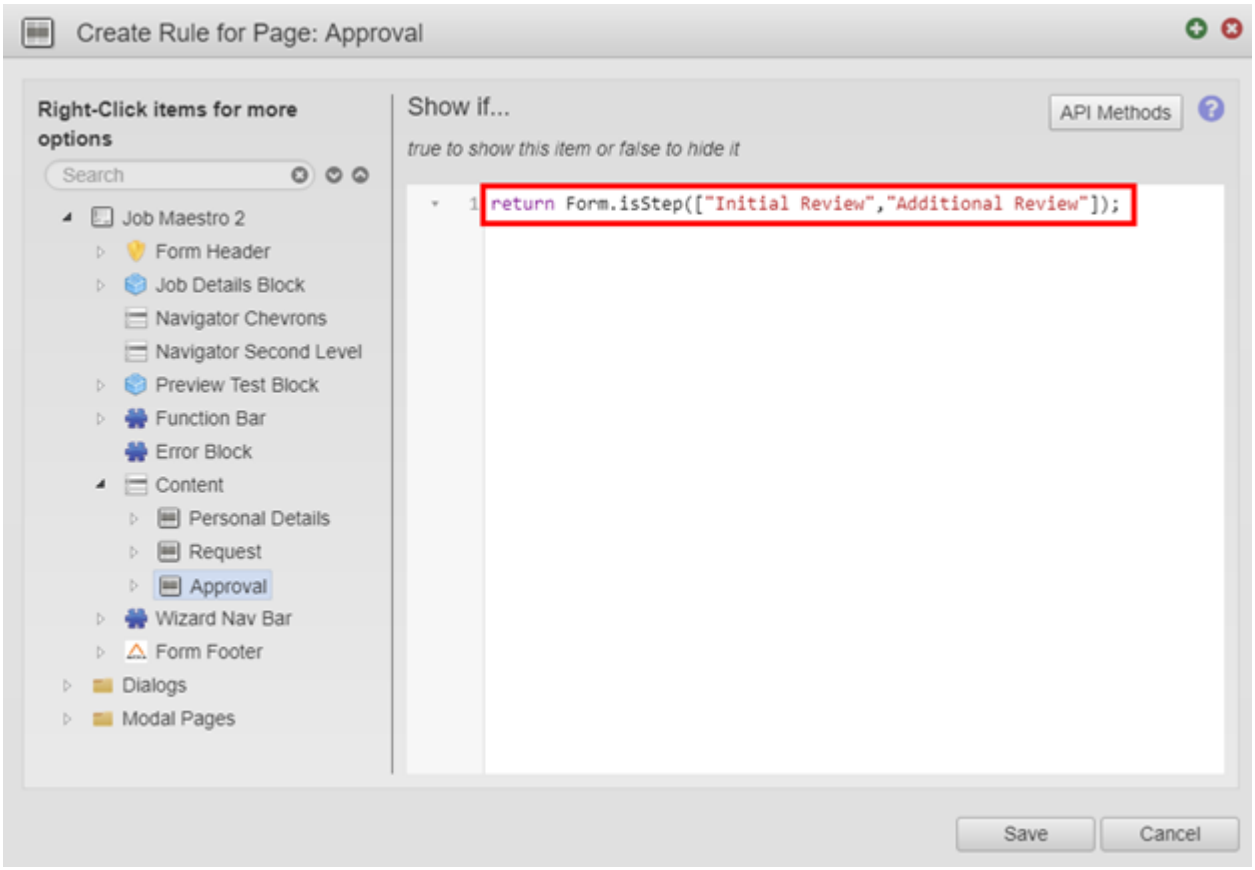
This rule is only relevant for this section to appear when the application was previously declined.



A visibility rule is created to only show this tab to both reviewers and managers.



Create a new rule as shown below.



Using a Radio Button Group for the Reviewer and Manager Decision

Both the reviewer and the manager need to be able to indicate their decision on the form. Previously, we covered the **Route Select** component, which dynamically loads each of the possible decisions for a reviewer and a manager, using the job controller service. However, a radio button group is much better choice because:

- It looks nicer than a dropdown list.
- Feedback and decision can be clearer.
- The form receipt indicates the decision that was made.
- It accommodates for additional instructions or feedback to the reviewer and manager.

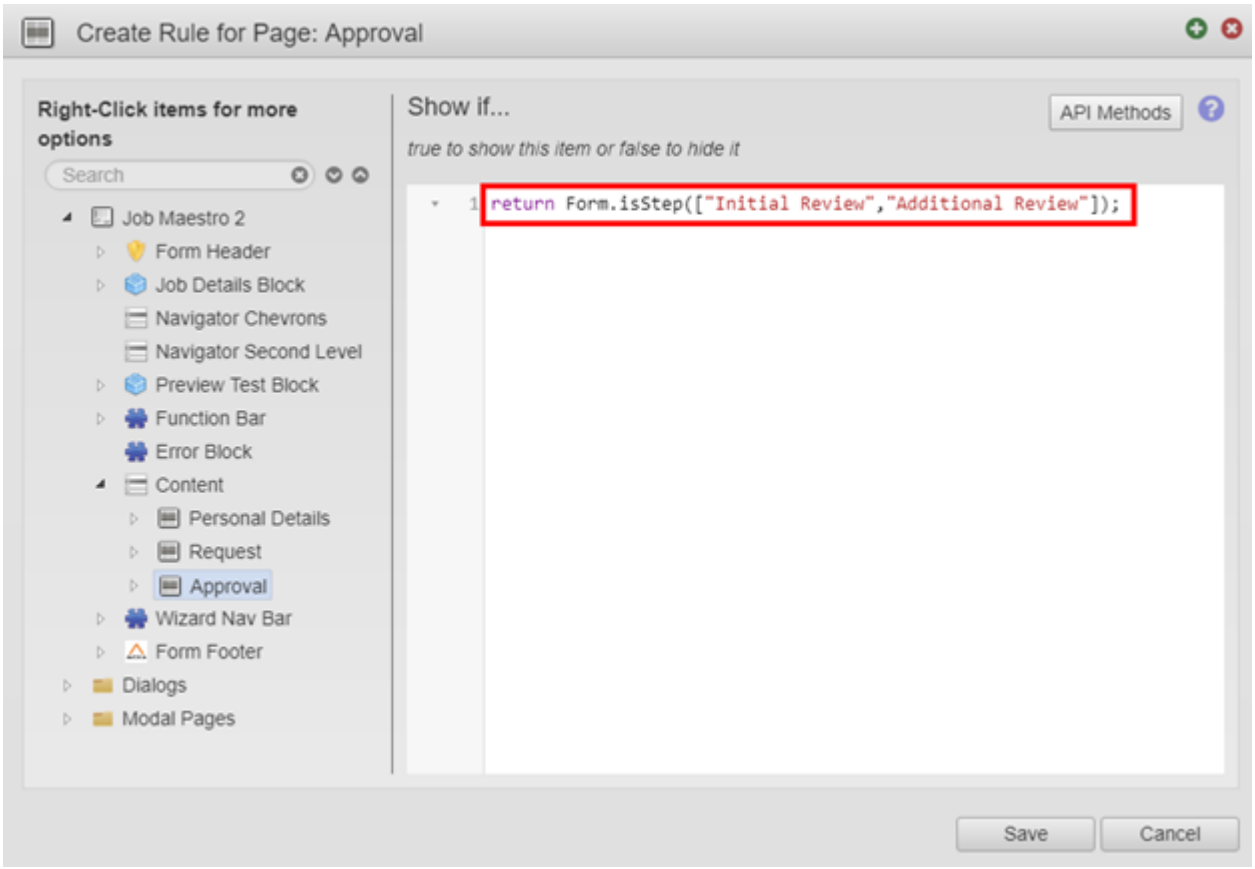
The only negative side is that the radio button group doesn't allow for the dynamic load of the radio button options based on the job definition service. You must hard code the possible decisions as radio button options on the radio buttons group.

In this example, the reviewer and the manager are both involved in the collaboration job, so we will use the **Review and Approval – Multi Step Group Review** job controller service.

As this decision feature is not available for the applicant, it makes sense to add a separate tab, the **Approval** tab, to your form, which can be shared amongst the reviewer and manager, so it is only visible to them.

To create a new visibility rule:

1. Select the **Approval** tab.
2. Select the **Properties** panel and click **Create Rule**.
3. Add the following rule.



4. Click **Save**.

Being a shared tab, we need to manage the form interaction correctly for each of the parties involved. For example, the following rules apply:

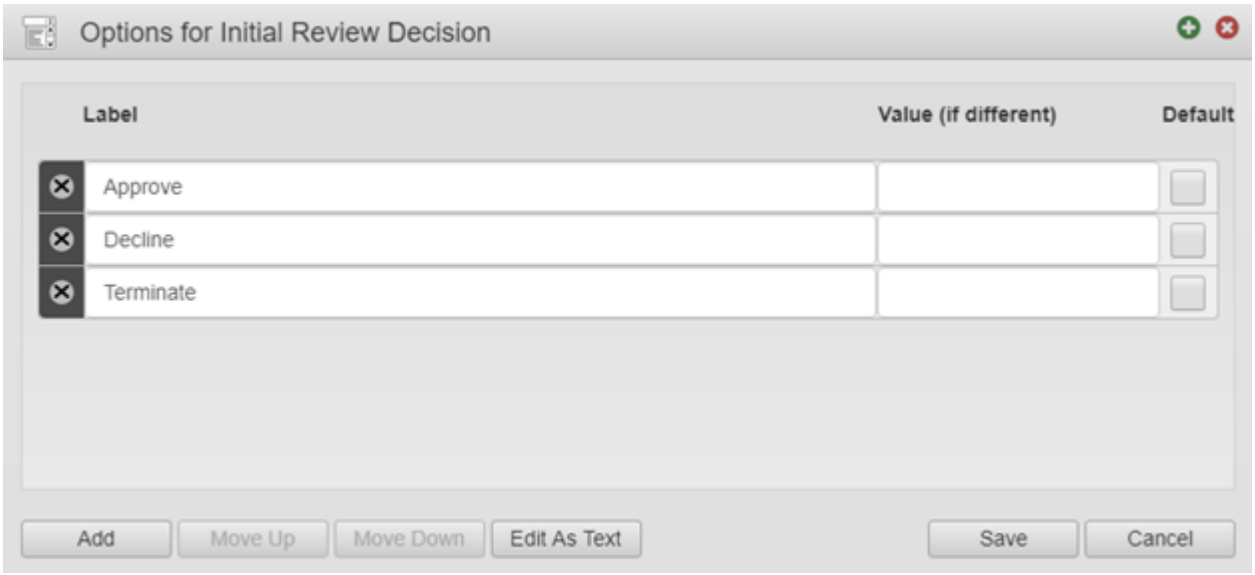
- A reviewer can indicate their decision, but can't see the manager's decision.
- A manager can indicate their decision, which is an additional decision to the reviewer decision. A manager is like a reviewer except they can also view, but not update the reviewer's decision.

To keep it simple, we will cover some of the rules here, but you will need to determine all the rules that apply for your form.

To create a radio button group for the reviewer decision:

1. Open your form in Maestro.
2. Add a new tab, the **Approval**, to capture the reviewer and manager decision.
3. Drag and drop the **Radio Button Group** component onto your form and name it **Initial Review Decision**.

4. Select the **Properties** panel.
5. Expand the **Button Group** section.
6. Click **Options**.
7. Enter each of the decision options under the **Label** column, for example, **Approve**, **Decline** and **Terminate**.



8. Click **Save**.
9. Expand the **Rule** section and select the **Mandatory** checkbox to make the decision mandatory to minimize ambiguity.

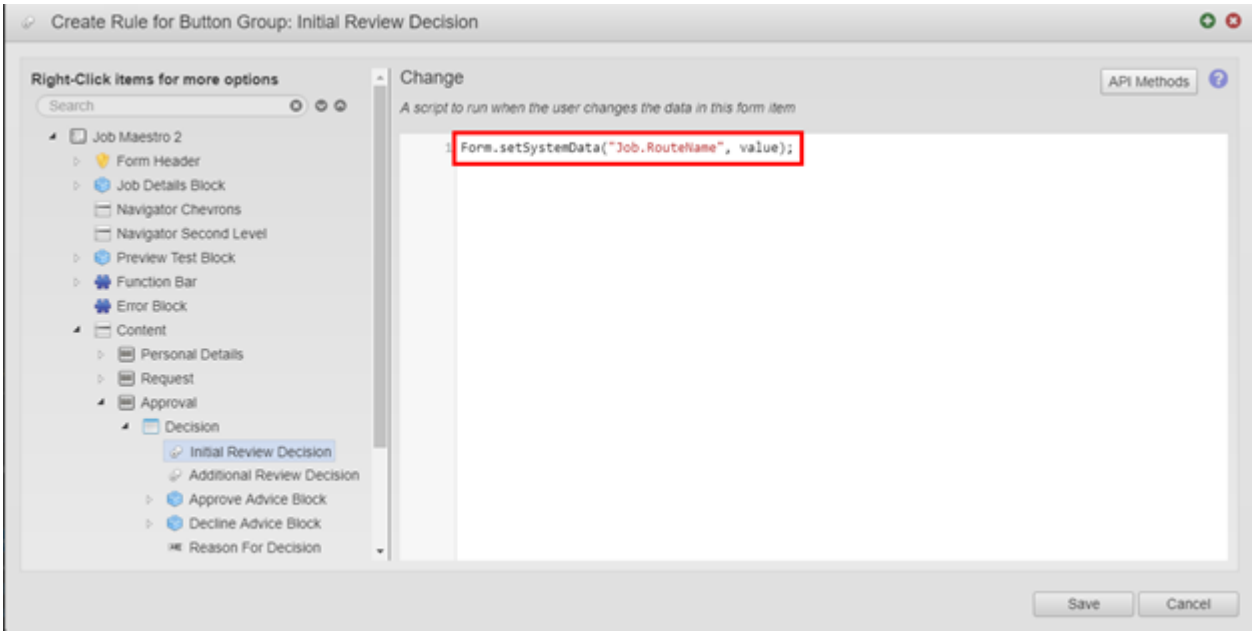
To create a radio button group for the manager decision, repeat the above steps 2 to 8 and give it a suitable name, for example **Additional Review Decision**.

As explained above, the two radio button groups require different rules, so for the reviewer, the **Initial Review Decision** radio button group, you need to create the following two rules:

Change Rule

An added complexity exists with sharing this form regarding which decision must be used for the job controller to decide what to do next. For this, you need to create a change rule which plugs the reviewer decision to the system profile route name, Job.RouteName, only when the reviewer has changed this. This means that, on change, the reviewer decision will be used by the job controller. To do this:

1. Select the **Properties** panel and click **Create Rule**.
2. Add the following rule.

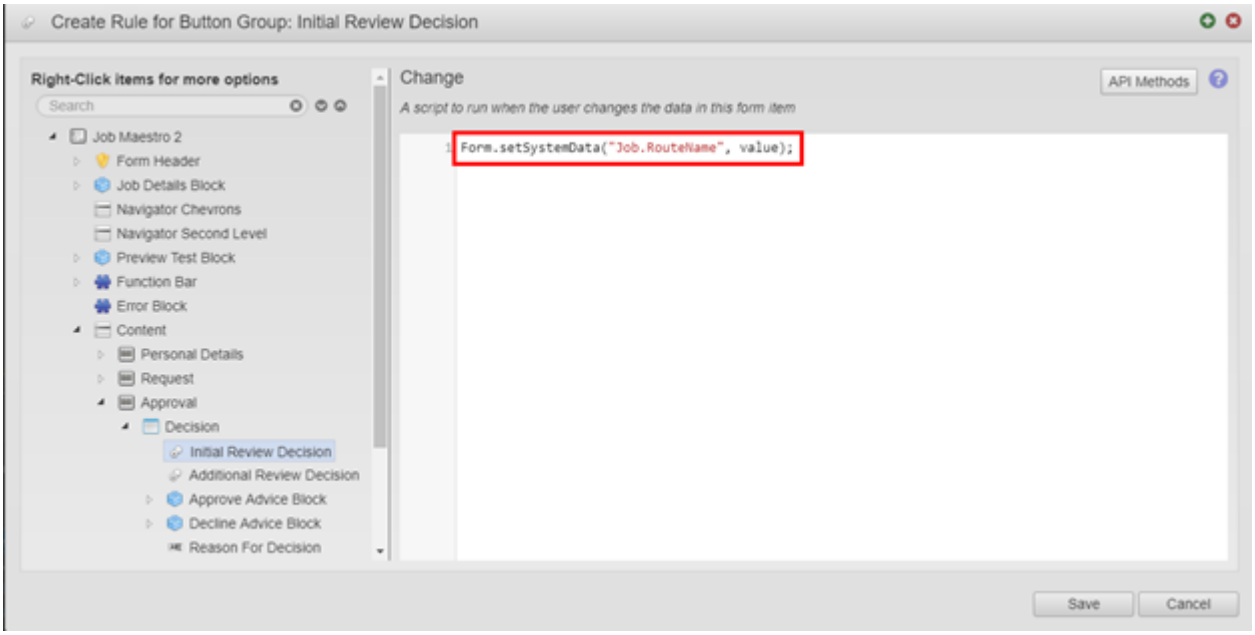


3. Click **Save**.

Editability Rule

As the reviewer can also be seen by the manager we need to make sure that only the reviewer can update this. To do this:

1. Select the **Properties** panel and click **Create Rule**.
2. Add the following rule.



3. Click **Save**.

For the manager, the **Additional Review Decision** radio button group, we need to create the following two rules:

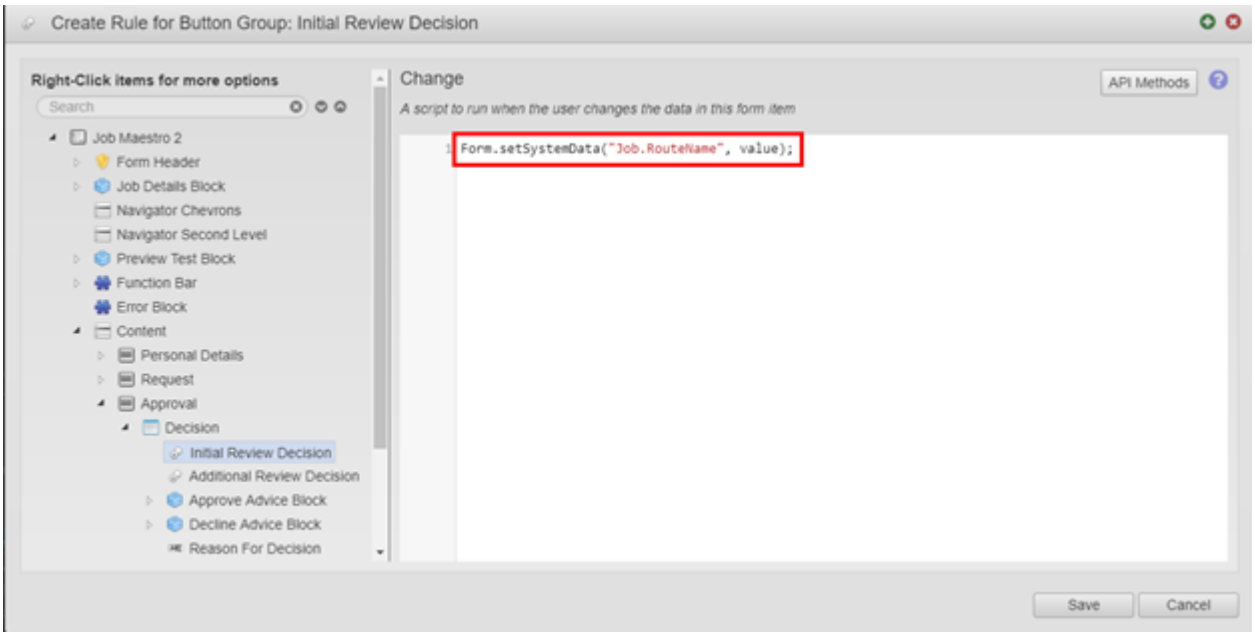
Change Rule

The same change rule as for the reviewer needs to be added here to allow the manager's decision to be used for directing what happens next when this has been changed.

Visibility Rule

This radio button group must only be visible to a manager. To do this:

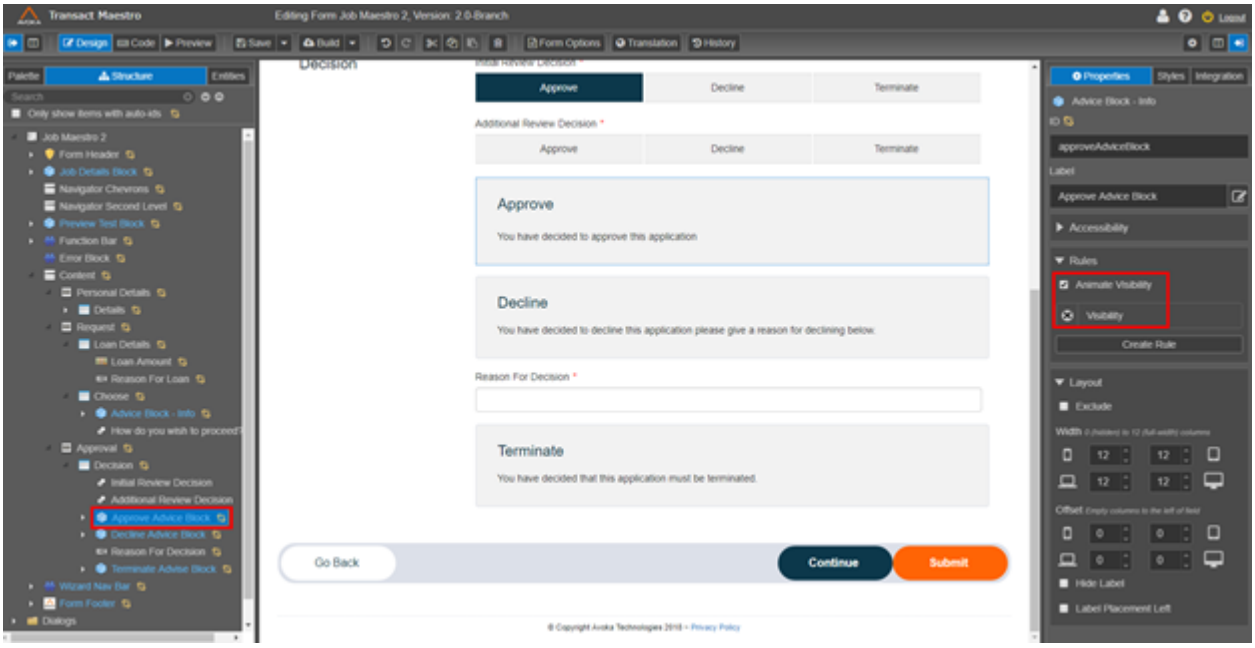
1. Select the **Properties** panel and click **Create Rule.**
2. Add the following rule.



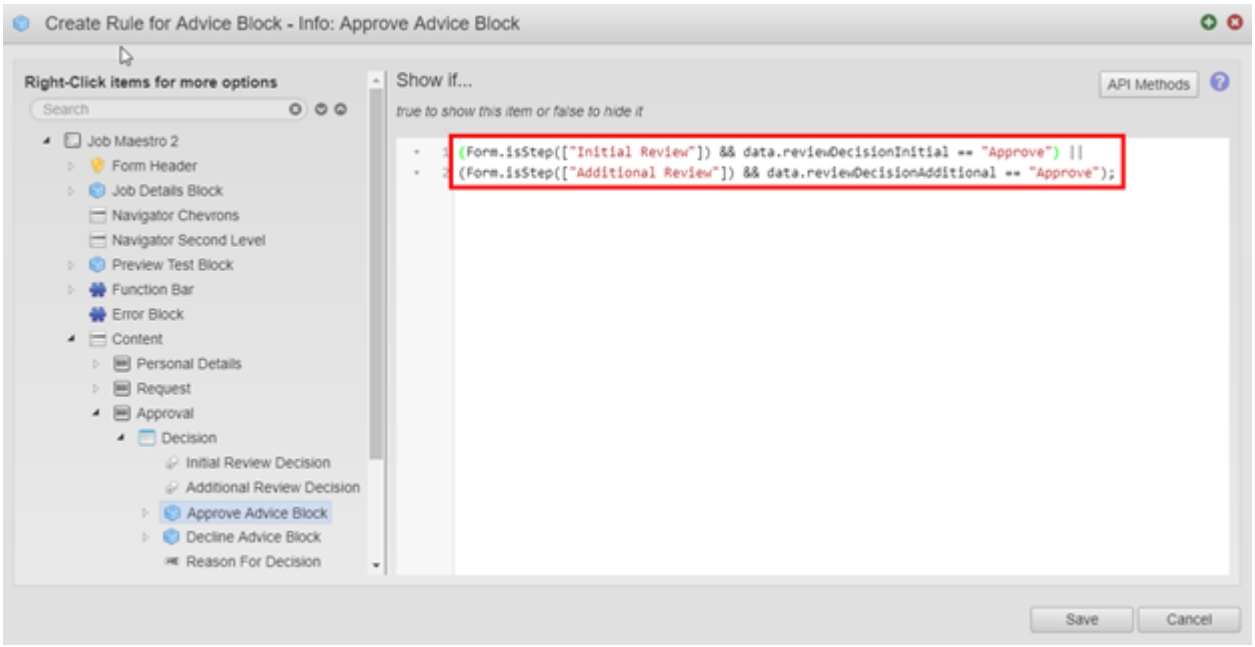
3. Click **Save**.

Some other rules that may be relevant are:

In this case an Approve Advice Block pops up when the reviewer or manager clicks the Approve radio button option, to highlight the decision they are about to submit.

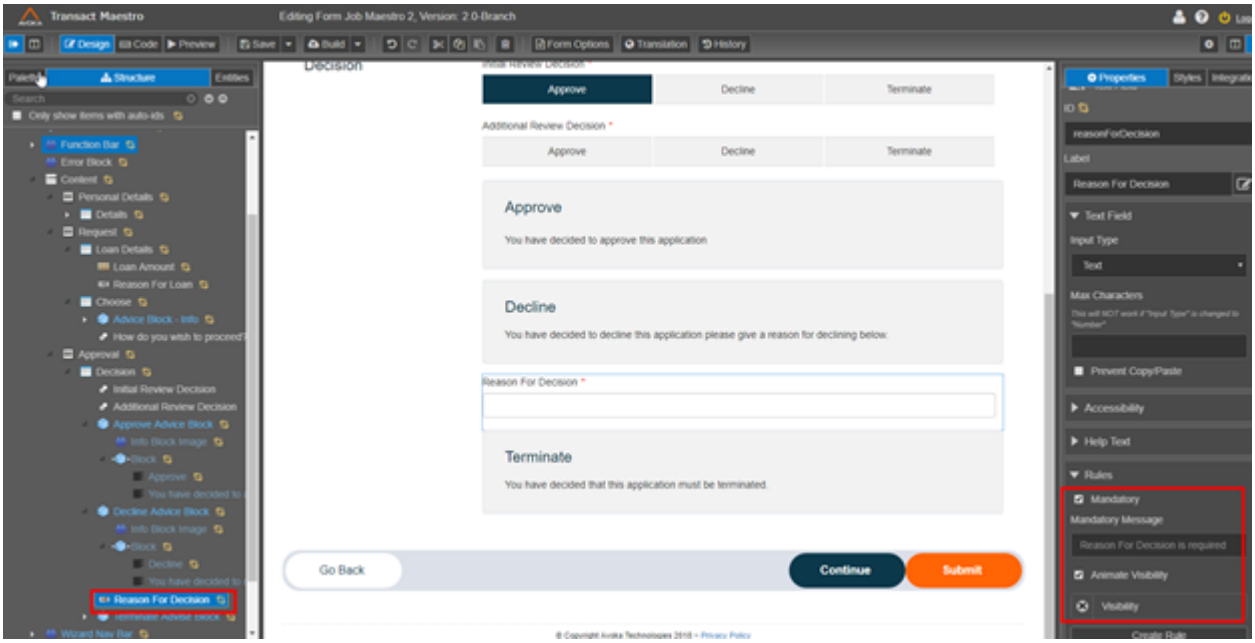


For this the visibility rule is:

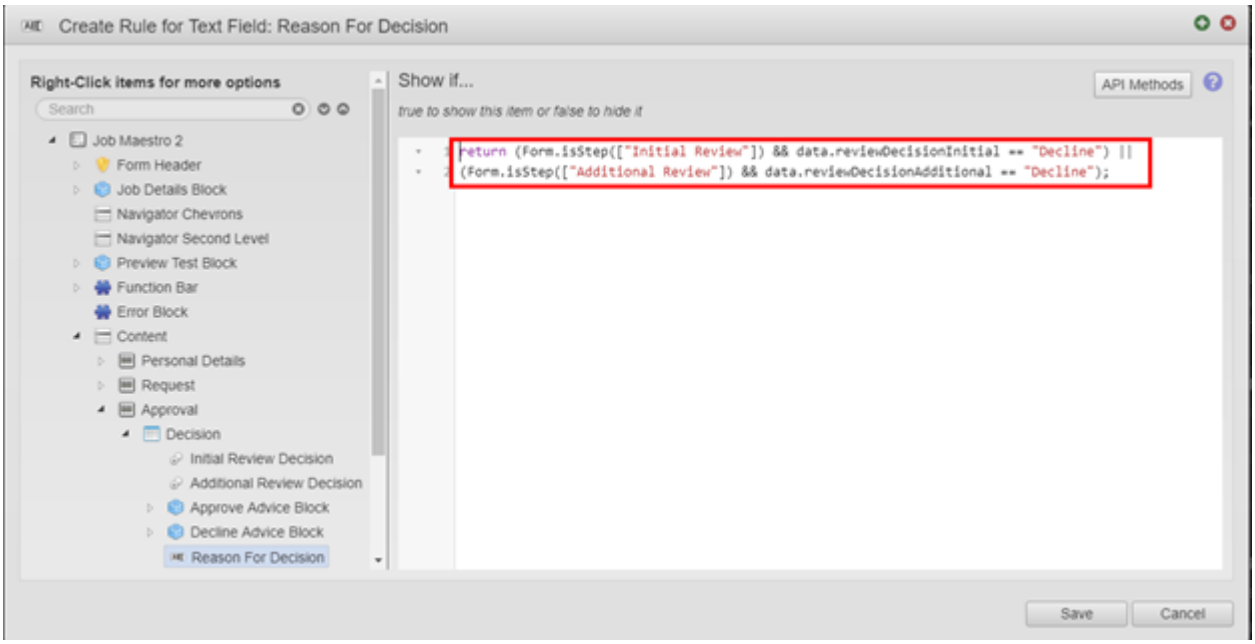


When either the reviewer or the manager declines the application, they must enter a reason for their decision.

The following visibility rule is required for the Reason for Decision field which is ticked as Mandatory.



The visibility rule is:



Now, your Maestro form is ready to be used with collaboration jobs.

Next, learn how to configure Composer forms for use with collaboration jobs.

Configure Composer Forms for Use with Collaboration Jobs

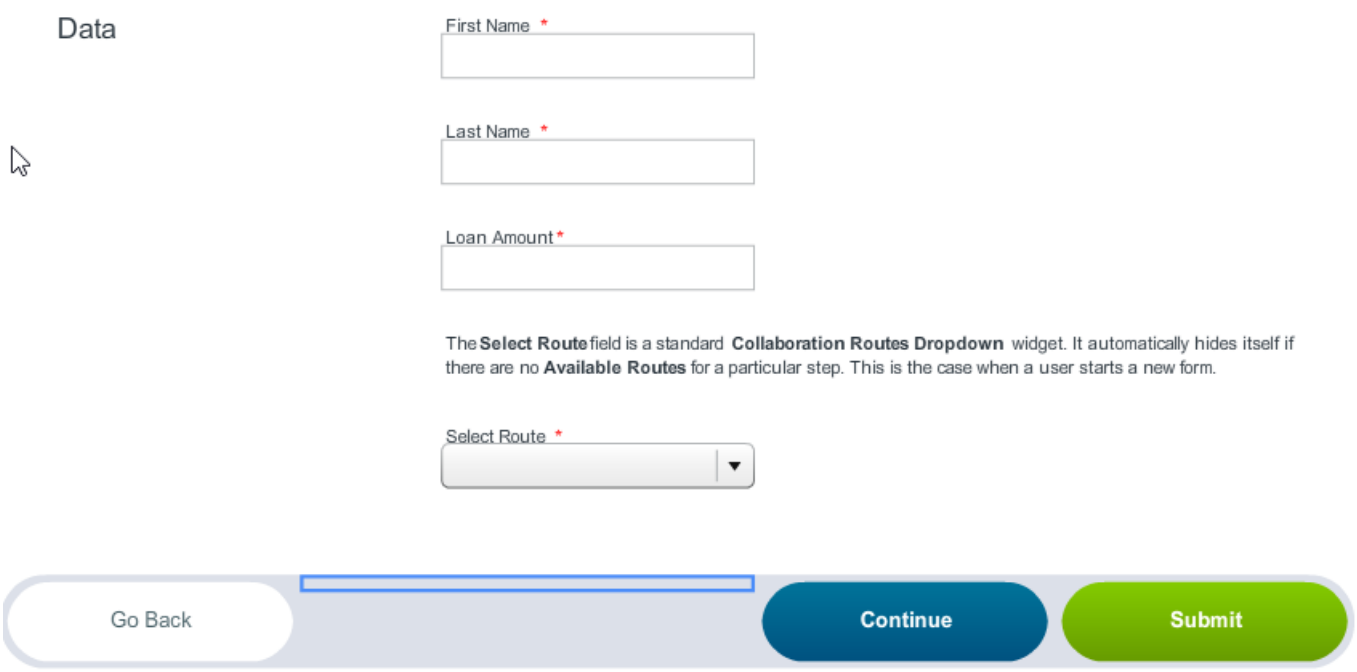
Composer | Form Builder | 5.1 & Higher

Composer forms require configuration before you can use them with collaboration jobs.including:

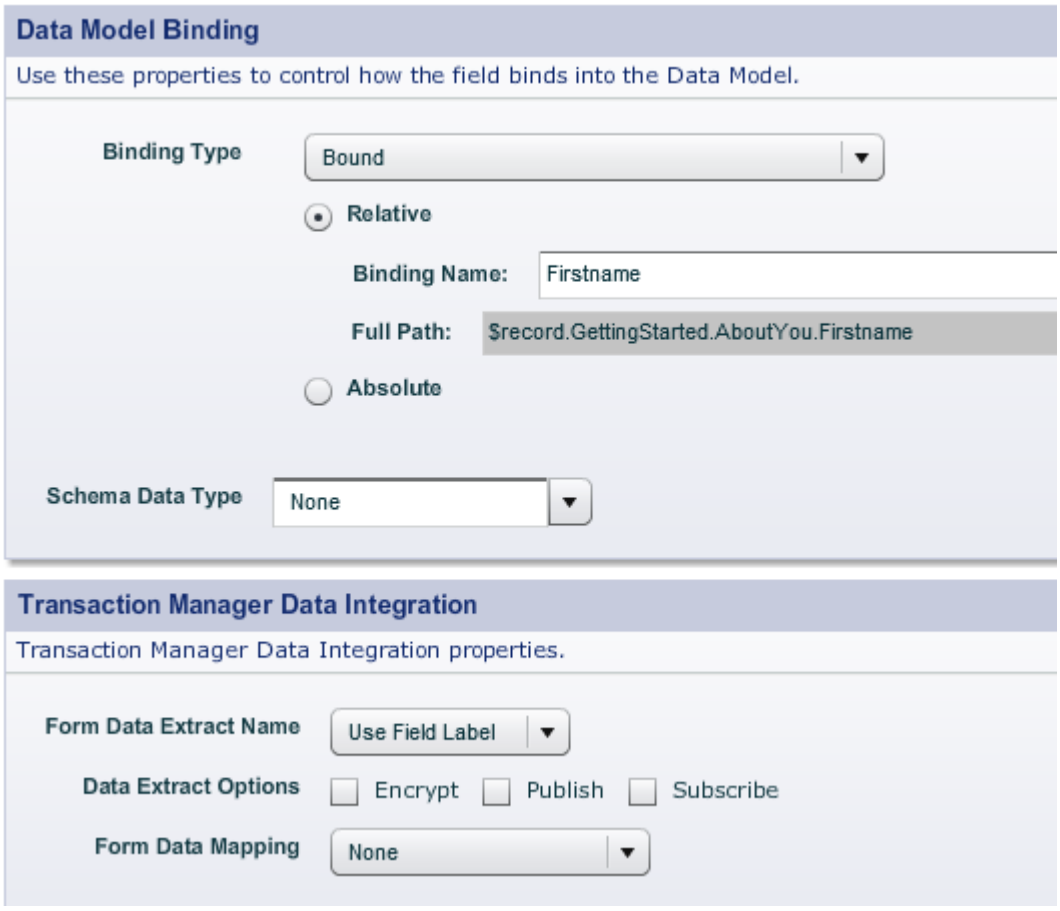
- How to configuredata extracts.
- What collaboration jobs widgets are.
- How to set rules, such as the section visibility and editabilty rules.

Data Extracts

Data extracts are used by collaboration jobs to access form data. In this example, we use the Composer form **Job Example 2 Minimal**.



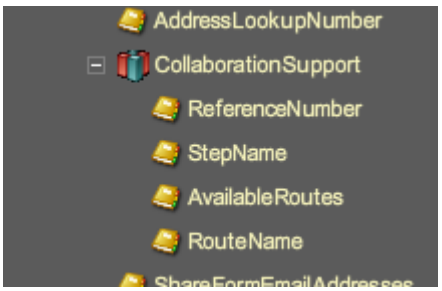
A collaboration job will put this into the **`$formDataMap.routeName`** values automatically. Data extracts are configured in Composer in the **Data Model** tab of the field's properties dialogue as shown below.



The easiest way to setup the data extract is to use the field label. However, you can use an alternative name. Composer version 4.3.0 and above has the **Encrypt**, **Publish** and **Subscribe** checkboxes to specify data extract options.

Form Data Mappings

The collaboration components described in the section below rely on the 4 Job data elements that are read into form variables. These can be found in **Nuts & Bolts -> Transaction Manager Support -> Collaboration Support** as shown below.



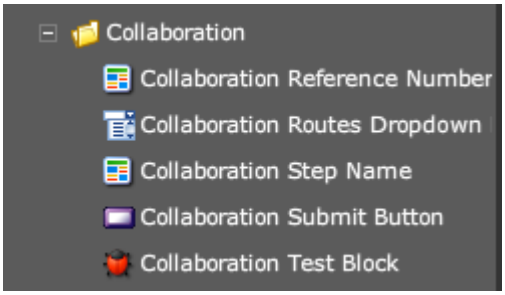
The job controller sets the following XML data elements for each task:

- ReferenceNumber, which is a Job Number
- StepName
- AvailableRoutes, which is a pipe delimited list of the current steps route names

The Composer form sets the **RouteName** on **Task Submission** via a widget or business rule.

Collaboration Widgets and Components

Composer has a number of widgets that can be used to build forms for collaboration jobs as shown below.



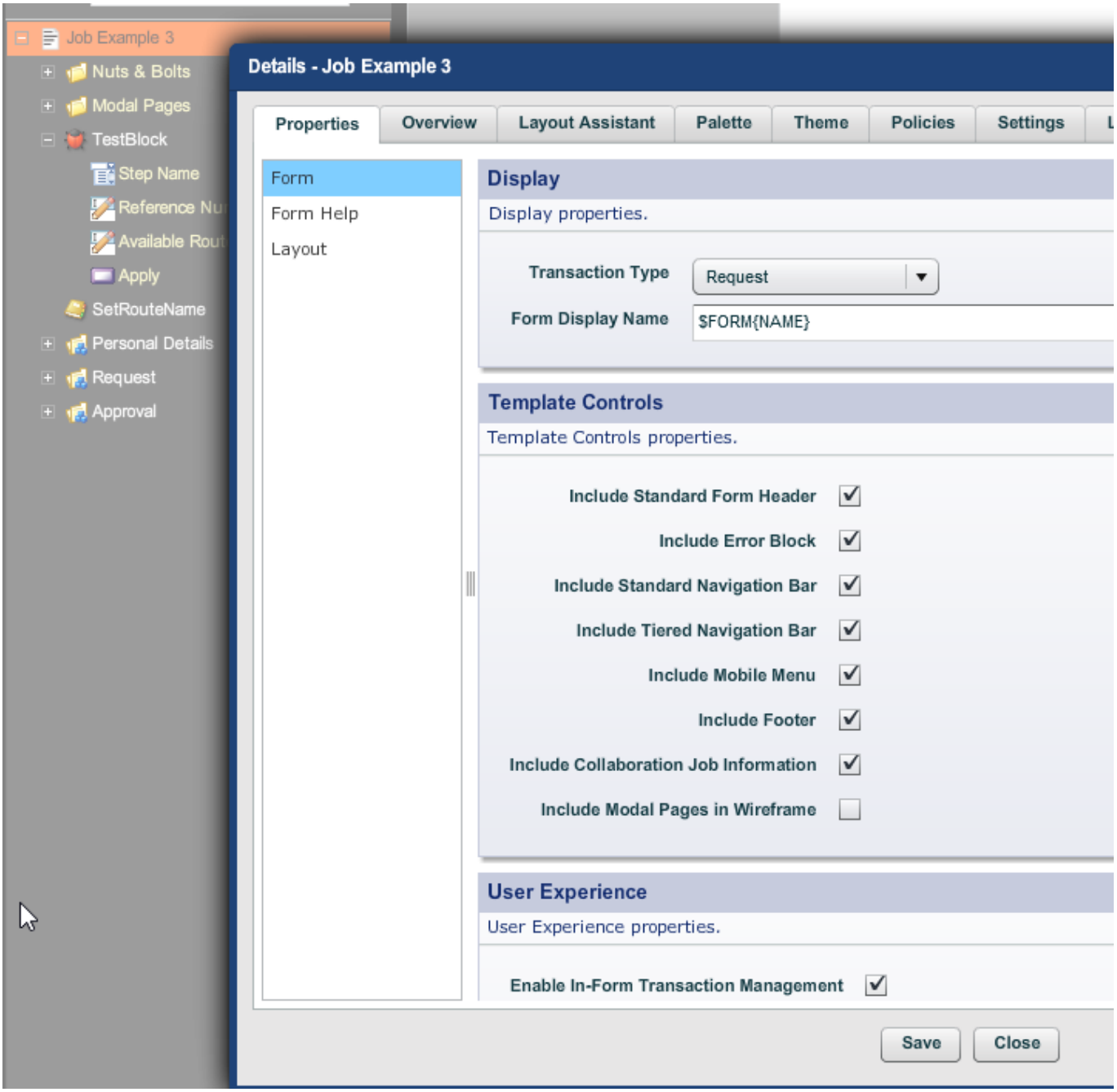
The Maguire template has additional collaboration support to display the step name and job number in the form heading.

Collaboration Reference Number and Collaboration Step Name

The Collaboration Reference Number (Job Number) and the Collaboration Step Name are Rich Text Fields to display the Job Number and Step Name on the form. Both fields can specify a prefix. The Minimal form contains Both fields above the data fields. We don't recommend that these fields be used as they are now both incorporated into standard Maguire 4.1 Template as shown below.



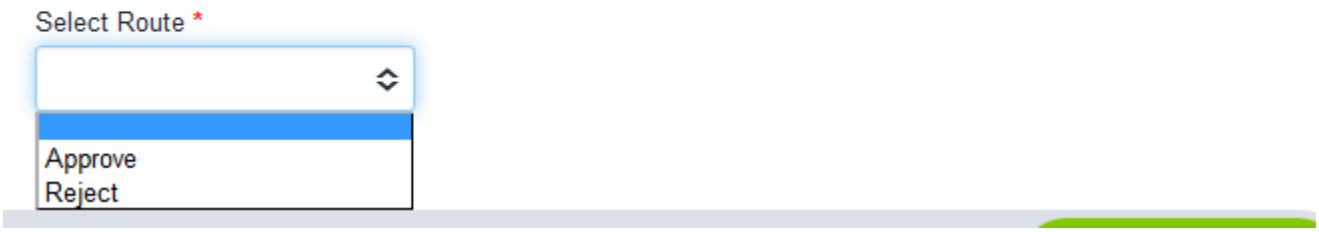
The template hides the Step - Job when they are blank such as when the applicant first fills out the form. To turn on this Maguire Feature double click on the forms root node, under the properties tab select the Include Collaboration Job Information check box.



Collaboration Routes Dropdown List

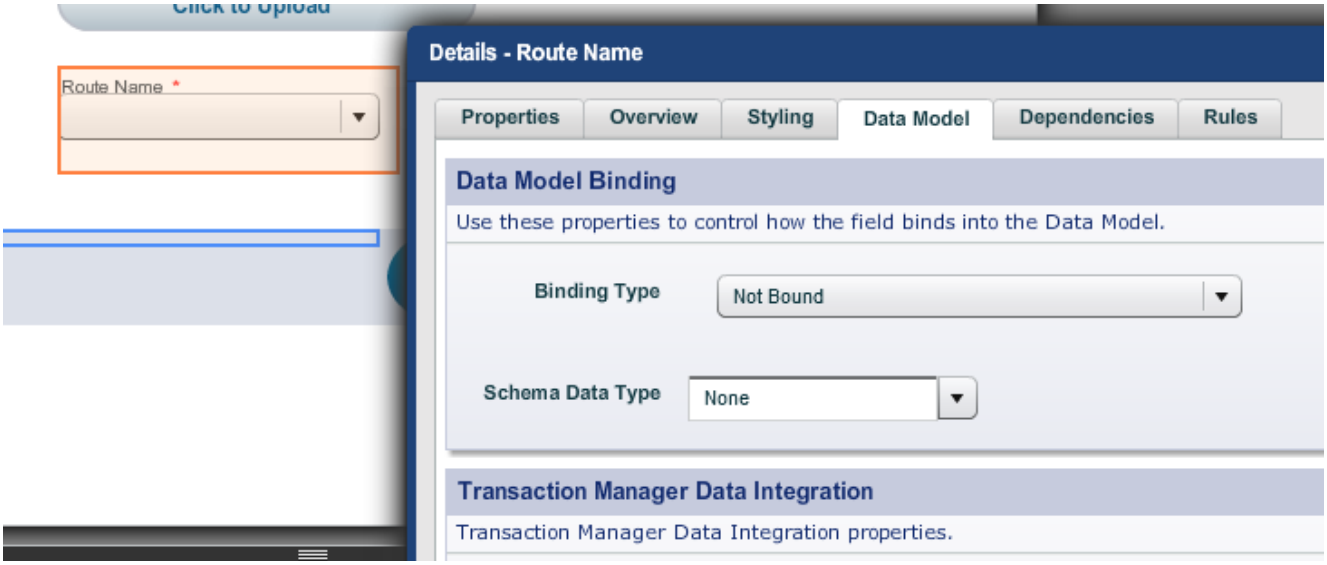
The Collaboration Routes dropdown List is the easiest way to make a user select the route to the next step.

The **Select Route** field is a standard **Collaboration Routes Dropdown** widget. It automatically hides itself if there are no **Available Routes** for a particular step. This is the case when a user starts a new form.



To ensure this item doesn't pick up the route of the previous step, its strongly recommended that the binding be changed to **Not Bound**. To do this:

1. Position it on the form.
2. Select **Properties > Data Model > Binding Type**
3. Select **Not Bound**.



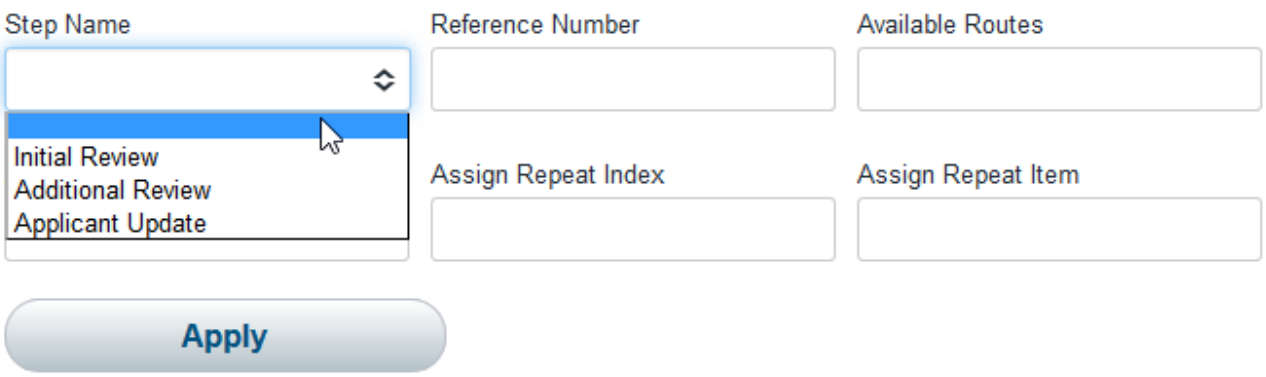
Collaboration Submit Button

Don't use the **Collaboration Submit Button** widget, because the form using this button can fail to return the route. It was initially prototyped to create a submit button that sets it's label as a route.

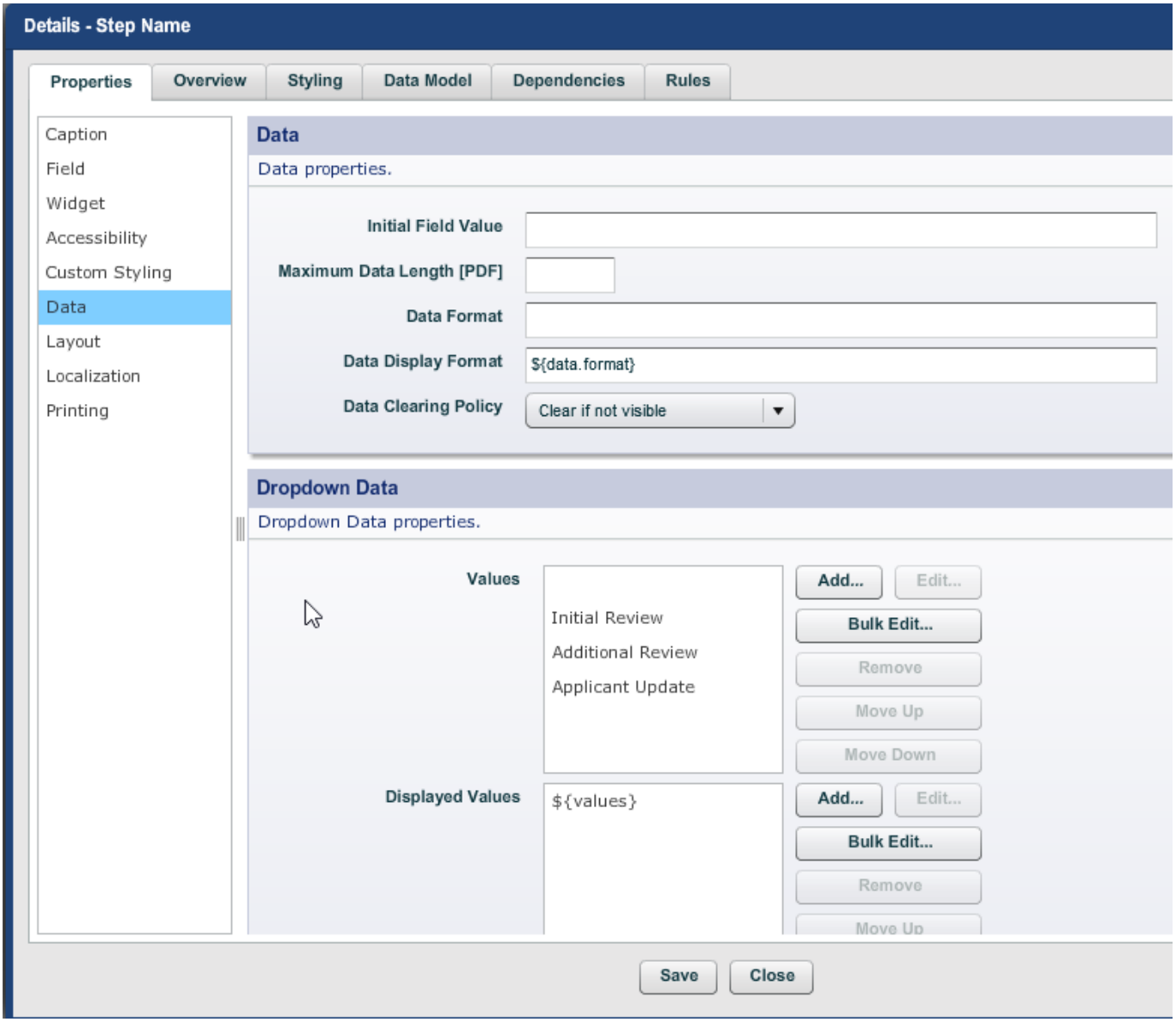
We recommend that you either use the **Collaboration Routes** dropdown or the standard **Radio Button Group** as described below.

Collaboration Test Block

The **Collaboration Test Block** is used to test a form in the Composer preview. It has the Step Name, Job Number and the Available Routes. Composer version 4.2.0 and above has 3 fields that are used to test **Task Assign Repeat** tasks.



Select the collaboration job's step name from the **Step Name** dropdown list. Don't forget to include an empty row to test the field's behavior in the form before its submitted.



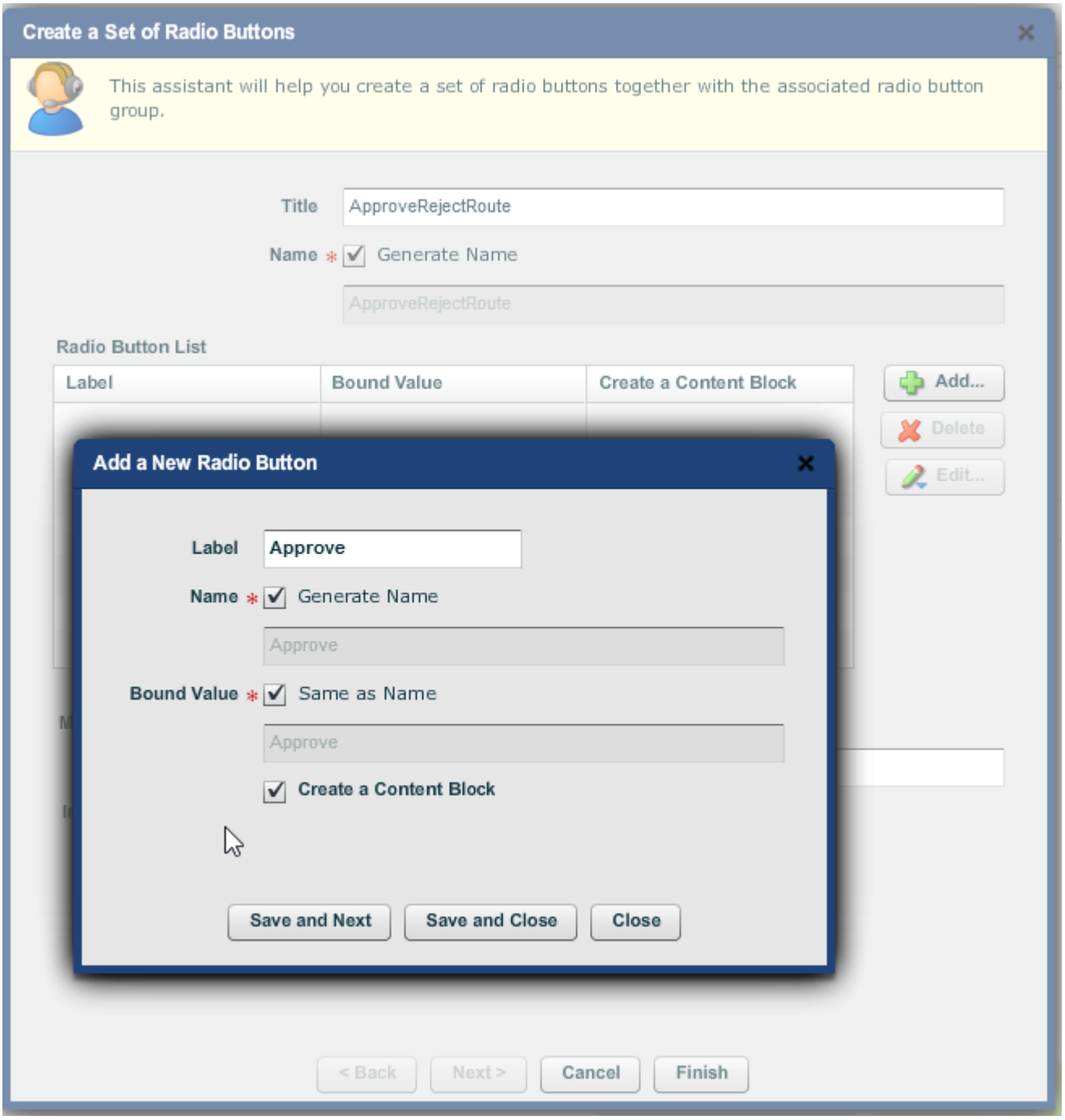
Routing Using the Standard Radio Button Group

This is the preferred method to use with the collaboration submit button.However, it doesn't dynamically create the radio buttons, so you have to hard code the values in your form. The advantages of using the radio button groups are:

- They look nicer than the drop down.
- The give feed back to the user as to what they intend to do.
- The route choice appears on the form receipt.
- They can specify an optional content block with additional user's instructions, which is useful for new form users. The content block can also hold addition fields that give feedback to the next review step.

The following example creates a single radio button group that is used for form approval. To do this:

1. Log onto Composer.
2. Drag the **Radio Button Assistant** widget into you form hierarchy. This will open up the **Create a Set of Radio Buttons** wizard dialog.
3. Add the route names as **Radio Button Labels**.

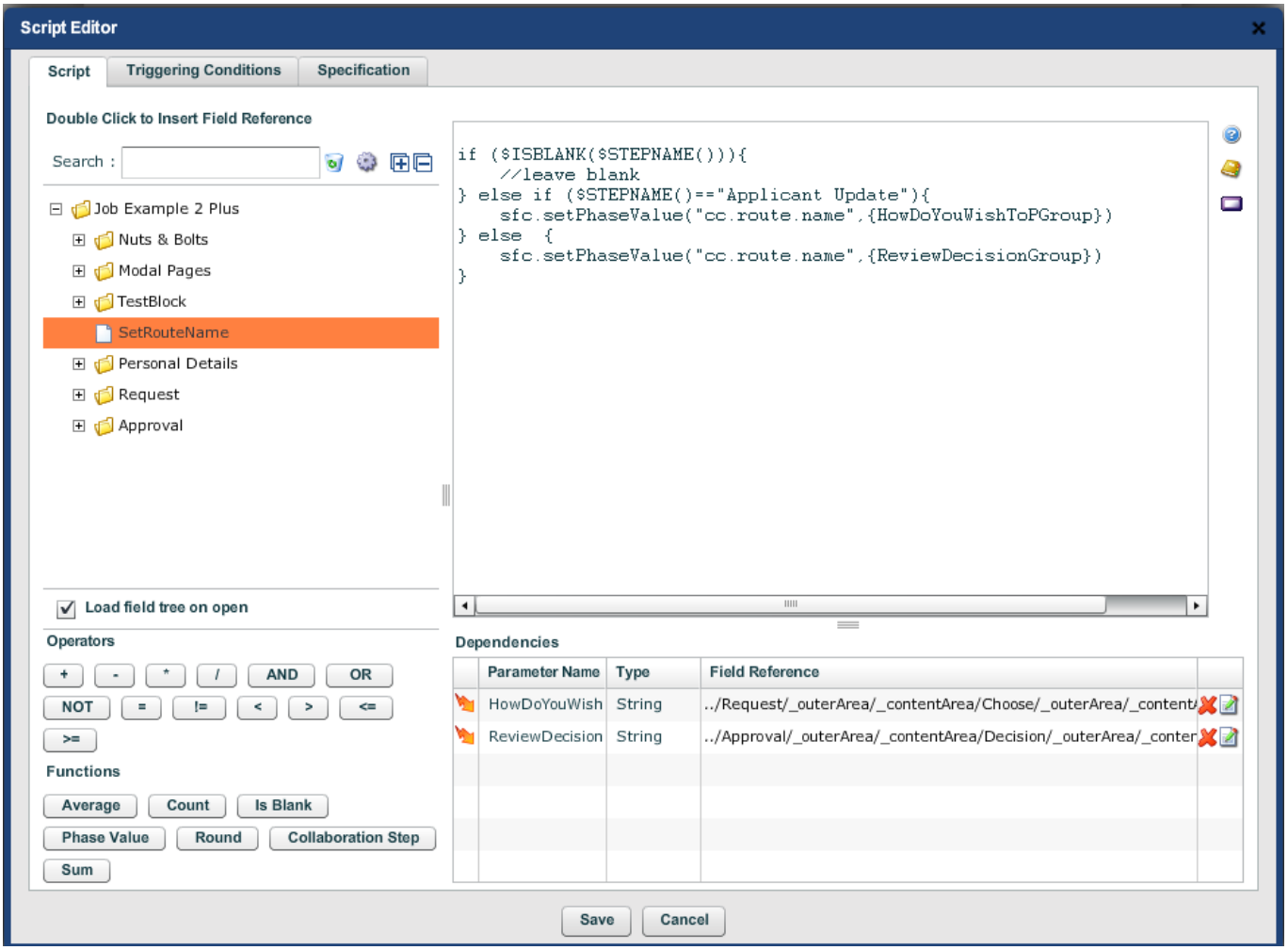


4. Make the button group mandatory.



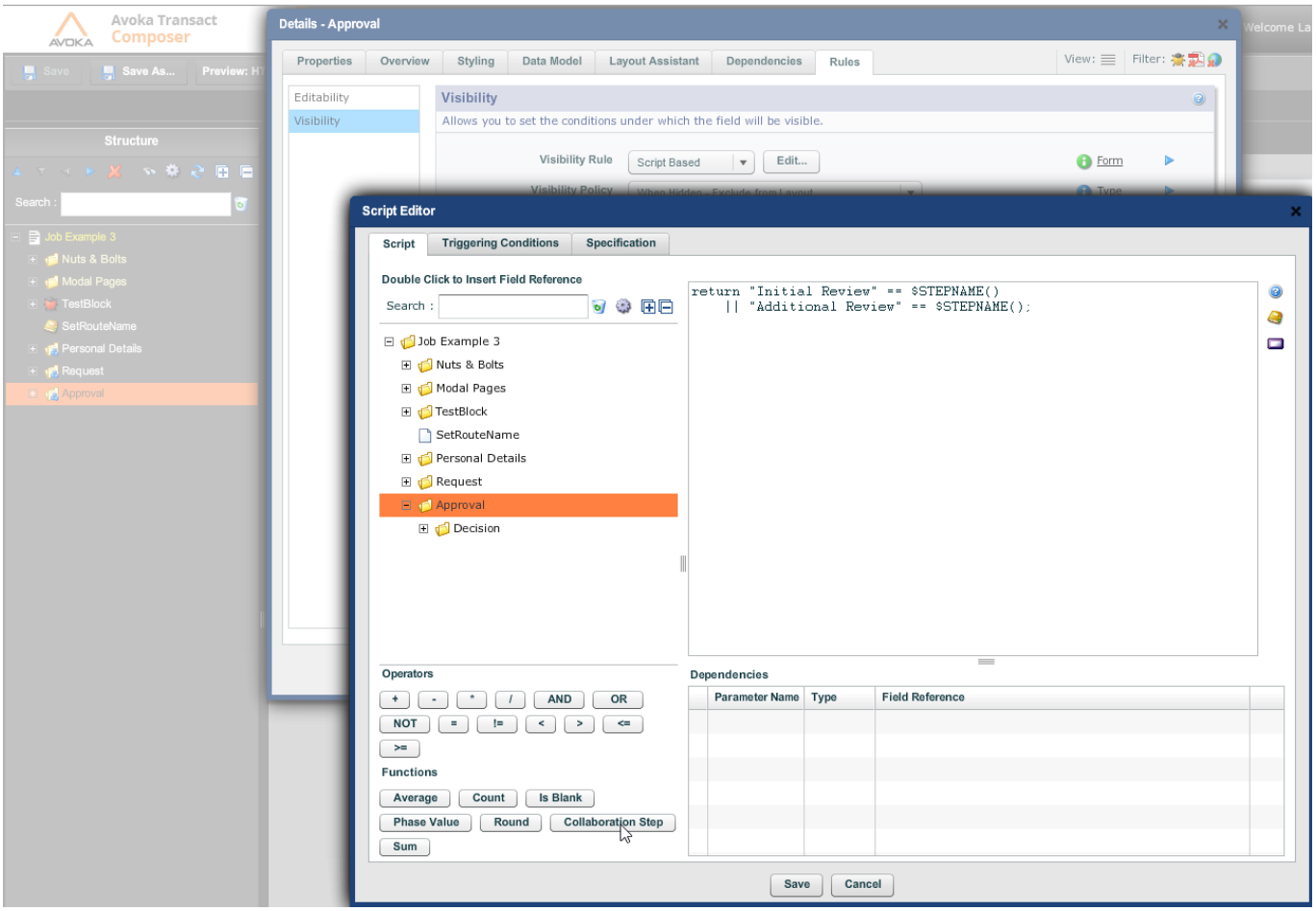
Business rules can be used to set the route. Your form has steps with different route choices like the **Job Example 2 (Multi Step Review Job)** with the business rule shown below.

```
if ($ISBLANK($STEPNAME())){
       //leave blank
} else if ($STEPNAME()=="Applicant Update"){
       sfc.setPhaseValue("cc.route.name",{HowDoYouWishToPGroup})
} else  {
       sfc.setPhaseValue("cc.route.name",{ReviewDecisionGroup})
}
```

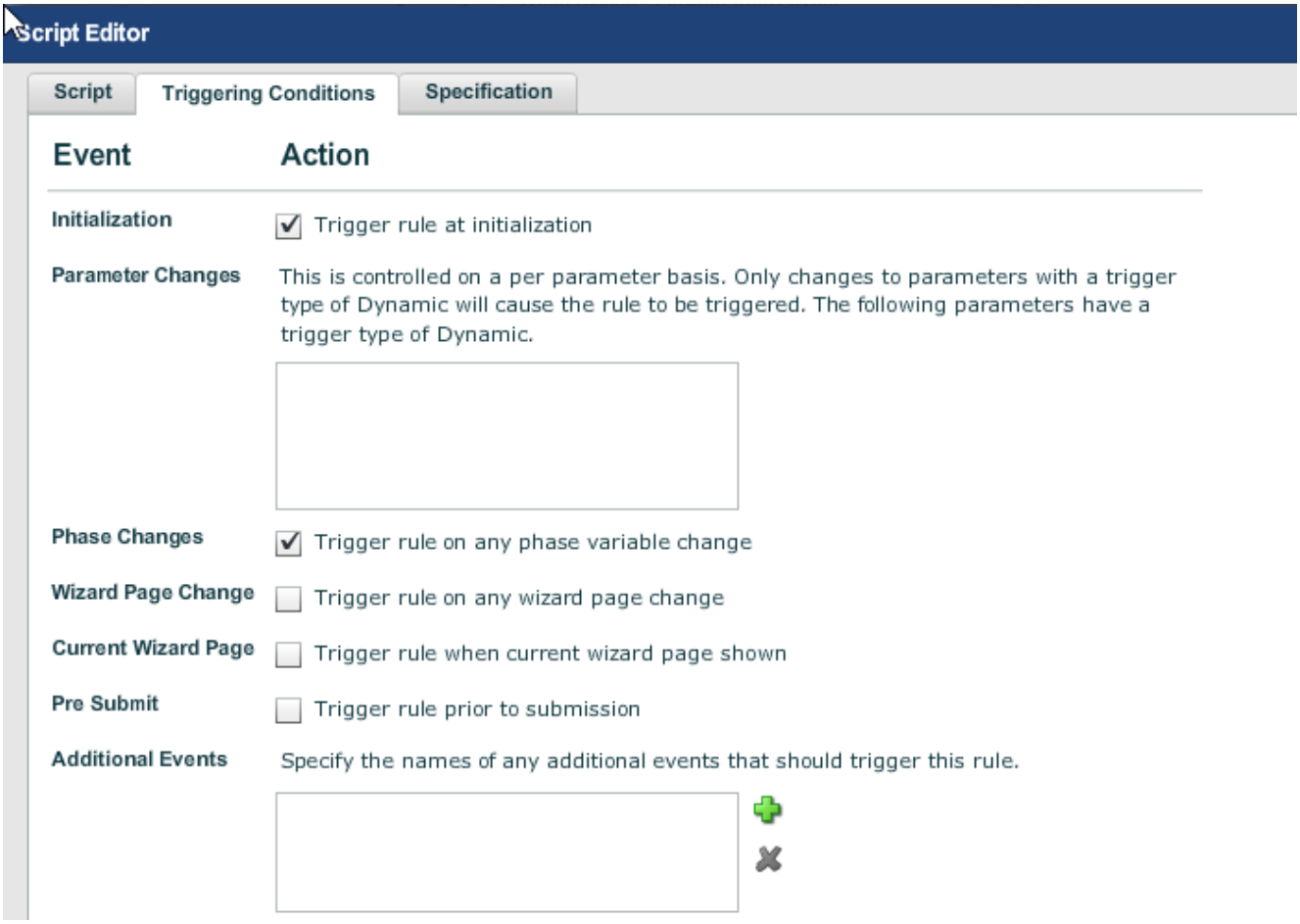


Visibility and Editability Rules

Scripts are used to setup visibility and editability rules for sections based on the Step Name. This can be used to hide the office use only section from the applicant or make the section of form that the applicant fills out read only. When configuring a script rule, use the collaboration step button that inserts a **$STEPNAME** reference to the step name into your script.



1. Select the **Triggering Conditions** tab.
2. Select the **Phase Changes** checkbox.

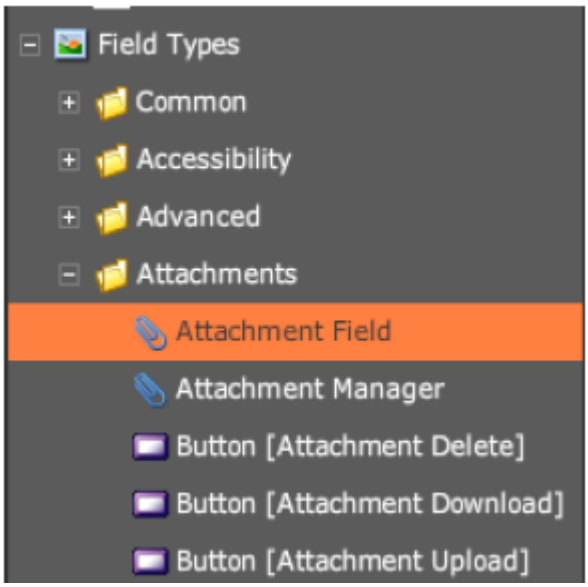


3. Click **Save**.

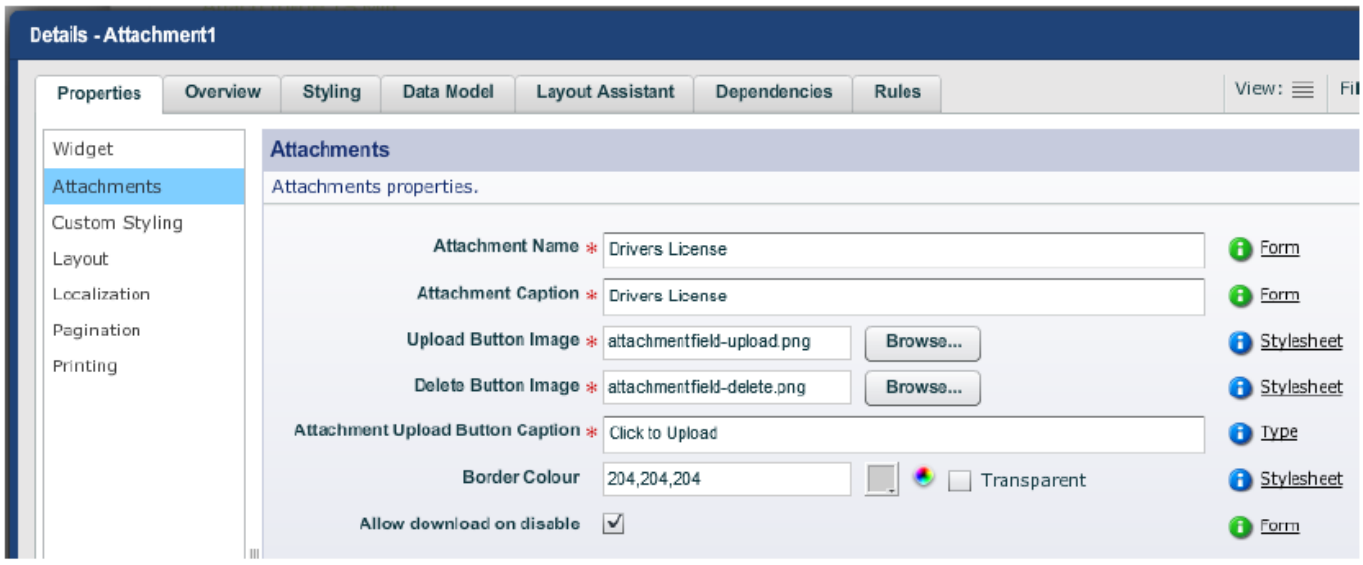
The step name is blank when this form is first submitted. The Visibility Rule that would show a section that is only visible when a user fills out the form for the first time return: **`isBlank($STEPNAME())`**

Attachments

The **Attachment Field** is available with Manager version 4.1 and above that uses the Maguire Template 4.1 as shown below.



This widget will allow a file to be uploaded. A user may want to confirm the file is correct and can download the form by clicking on the file link. They can delete the file by clicking on the X. The field works as expected when a user saves the form any attachments that have been uploaded are available when the form is next opened. This field uses the same Editability and Visibility rules as discussed above. However, if the editability rules determine that the widget is not editable at a particular step, the attached file can't be uploaded, downloaded or deleted. There is an Allow download on disabled check box as shown below. This is useful where a reviewer may want to look at the form but not delete or modify it. This is on by default.



The widget has an internal field **Attachment Manager**, which must not be altered when using with Manager task assignments and the job controller.

Configure Routes with Business Rules

It is possible to set the route from within the form using a business rule, which is how the pre-built **Collaboration Widgets** and **Components** work. Using a simple script, you can set the route phase value, which allows for logical changes to the collaboration route based on form data.

To configure the route:

1. Add a business rule to your form.
2. Edit the business rule script with the following code.

```
if (condition) {
       sfc.setPhaseValue("cc.route.name", "Deliver");
} else {
       sfc.setPhaseValue("cc.route.name", "Review");
}
```

You can set a number of the values associated with the collaboration job, including the step name (**`cc.step.name`**), reference number (**`cc.reference.number`**), available route (**`cc.available.routes`**), and assignee (**`cc.assignee`**), using this method.

Next, learn how to configure Maestro forms for use with collaboration jobs.

Customer Onboarding Collaboration Jobs

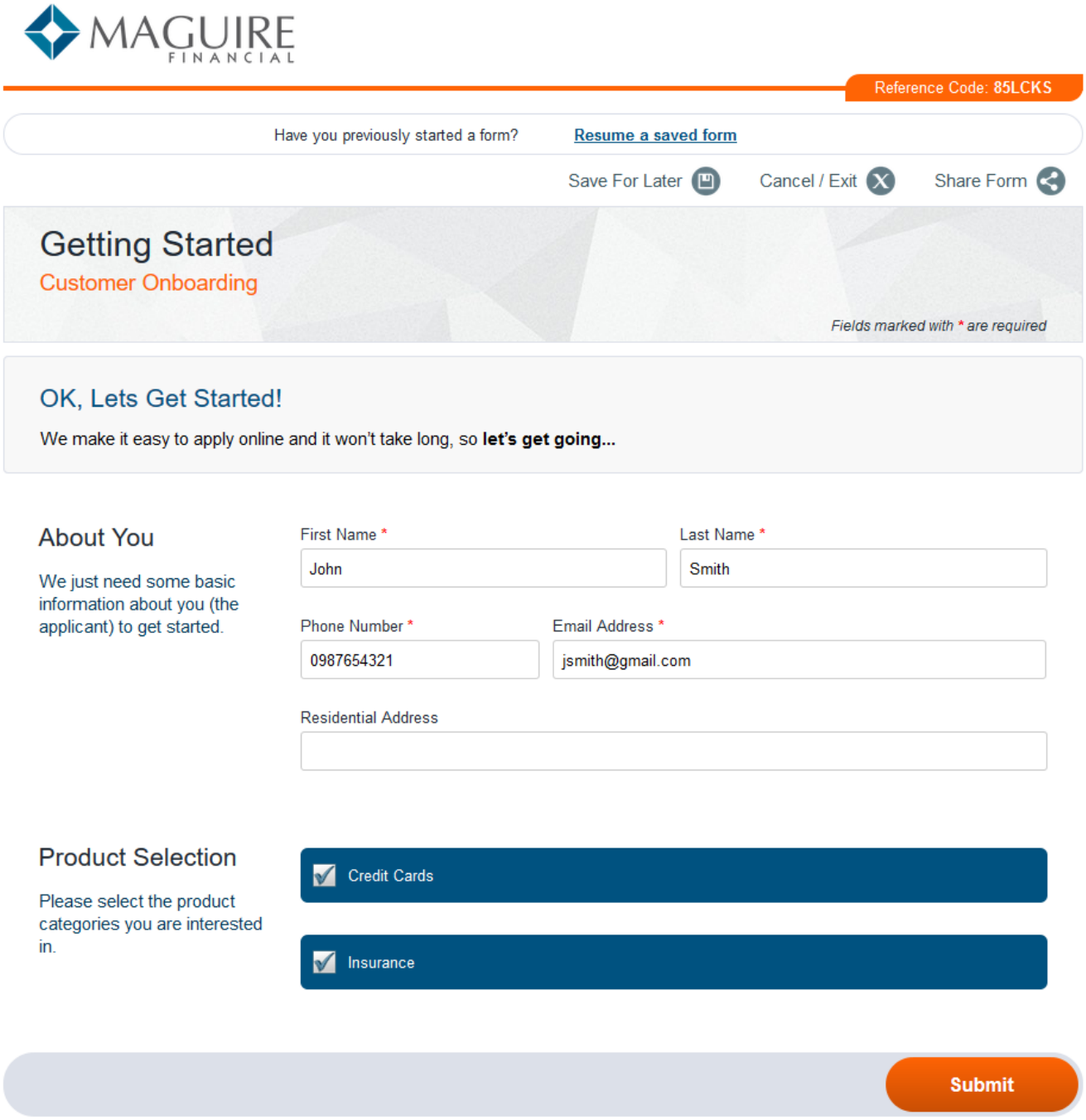
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A customer onboarding collaboration job implements a simple customer onboarding form bundle. The bundle consists of a number of separate forms that are filled out by a single user in the one step. We will use this as an example to illustrate the following:

- Anonymous users.
- Preconditions to determine which forms to use.
- Form bundle navigation.
- Redirect to a next form functionality or form chaining.
- Confirmation on the **Signature** page.
- Consolidation of form receipts.
- Dynamic preconditions.
- Shared form modal using the entire form shareFormData XML or shareExtractData for publish / subscribe only.

The Workflow

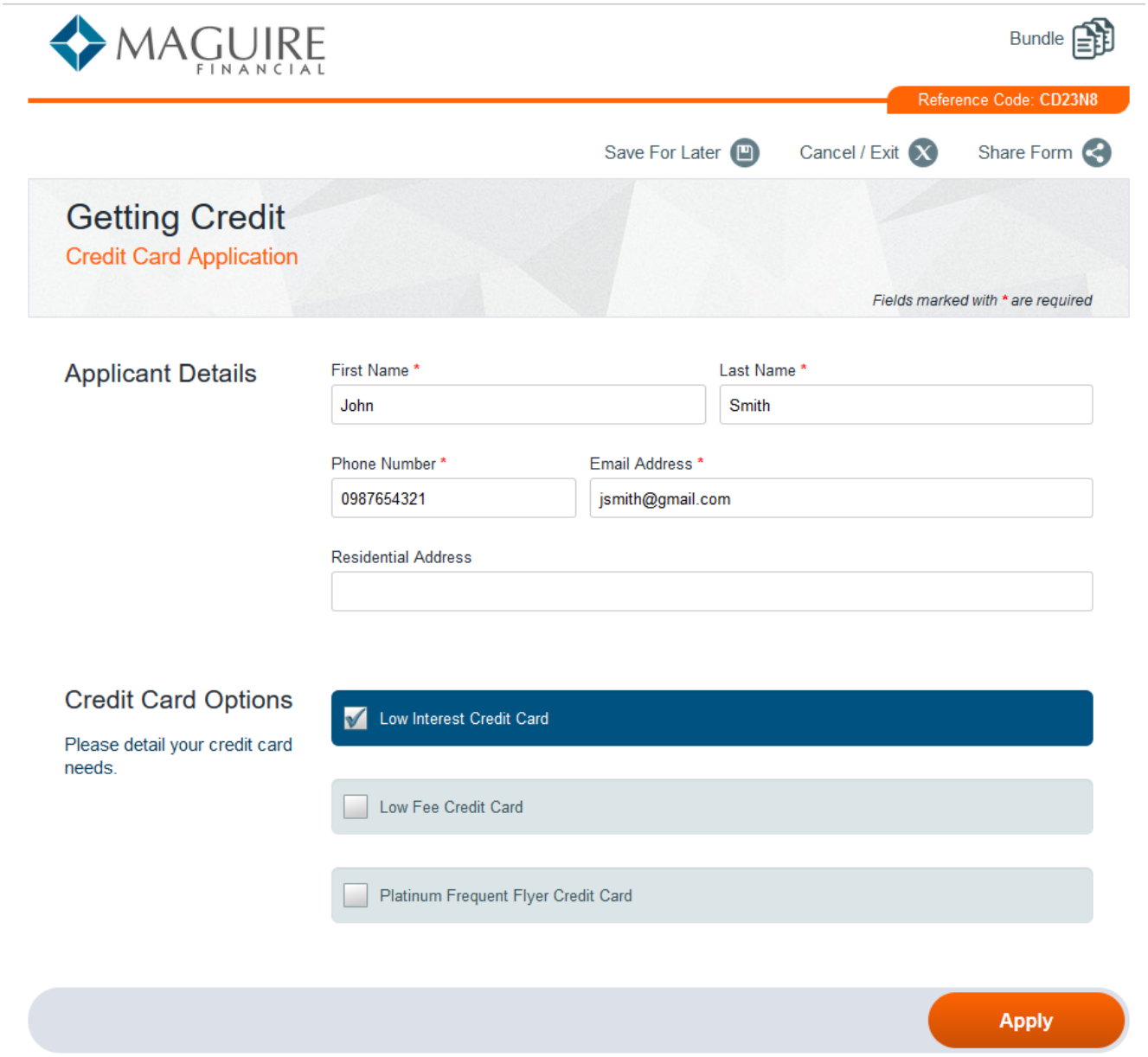
A user opens the customer onboarding triage form and fills the required details, such as the name, phone number and email address.



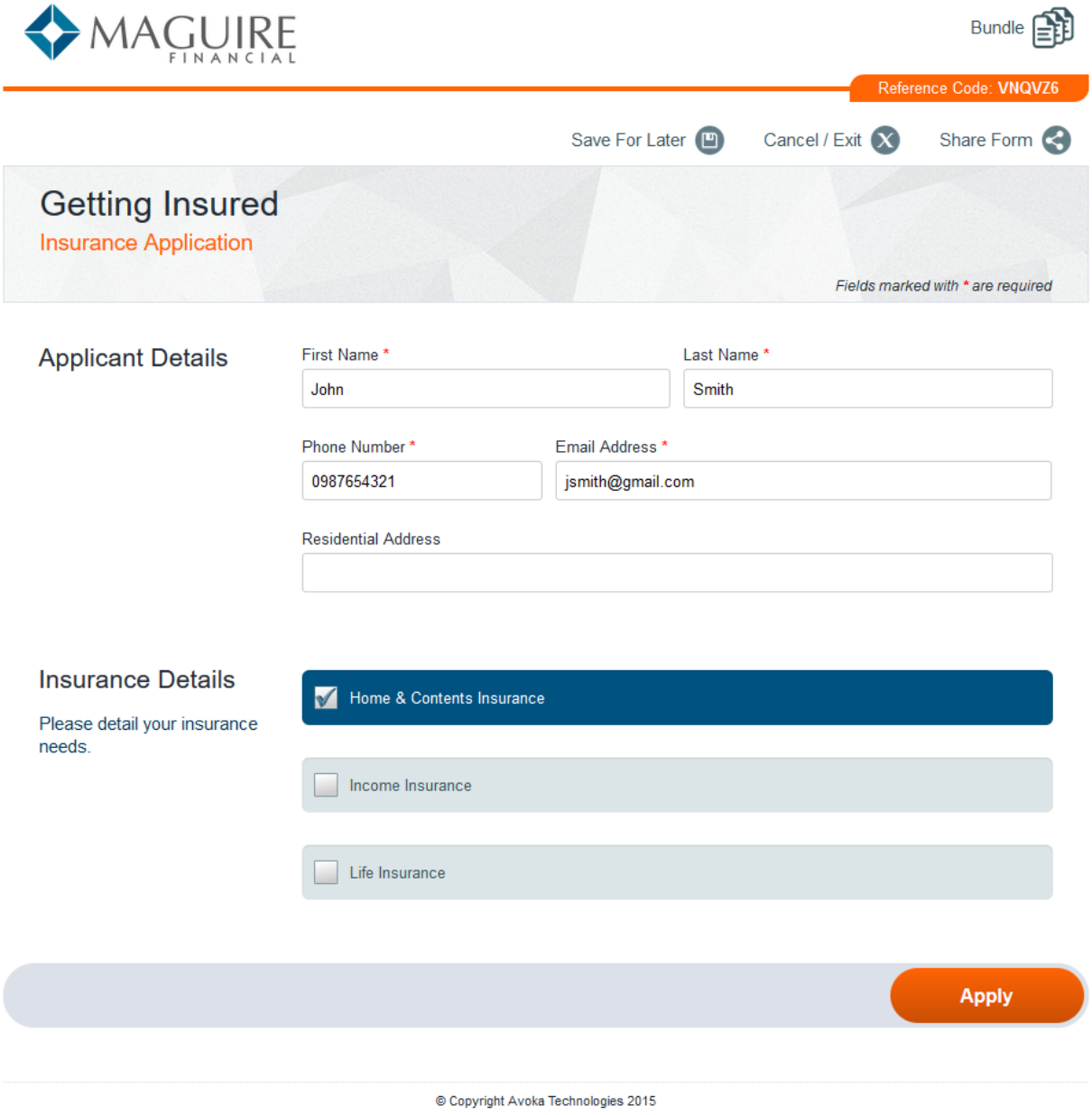
The user selects one or several available products, such as **Credit Cards**, **Insurance** or both, and clicks **Submit**. Manager starts a new collaboration job, which is immediately processed and goes to the **Additional Products** step.

Manager generates the following tasks:

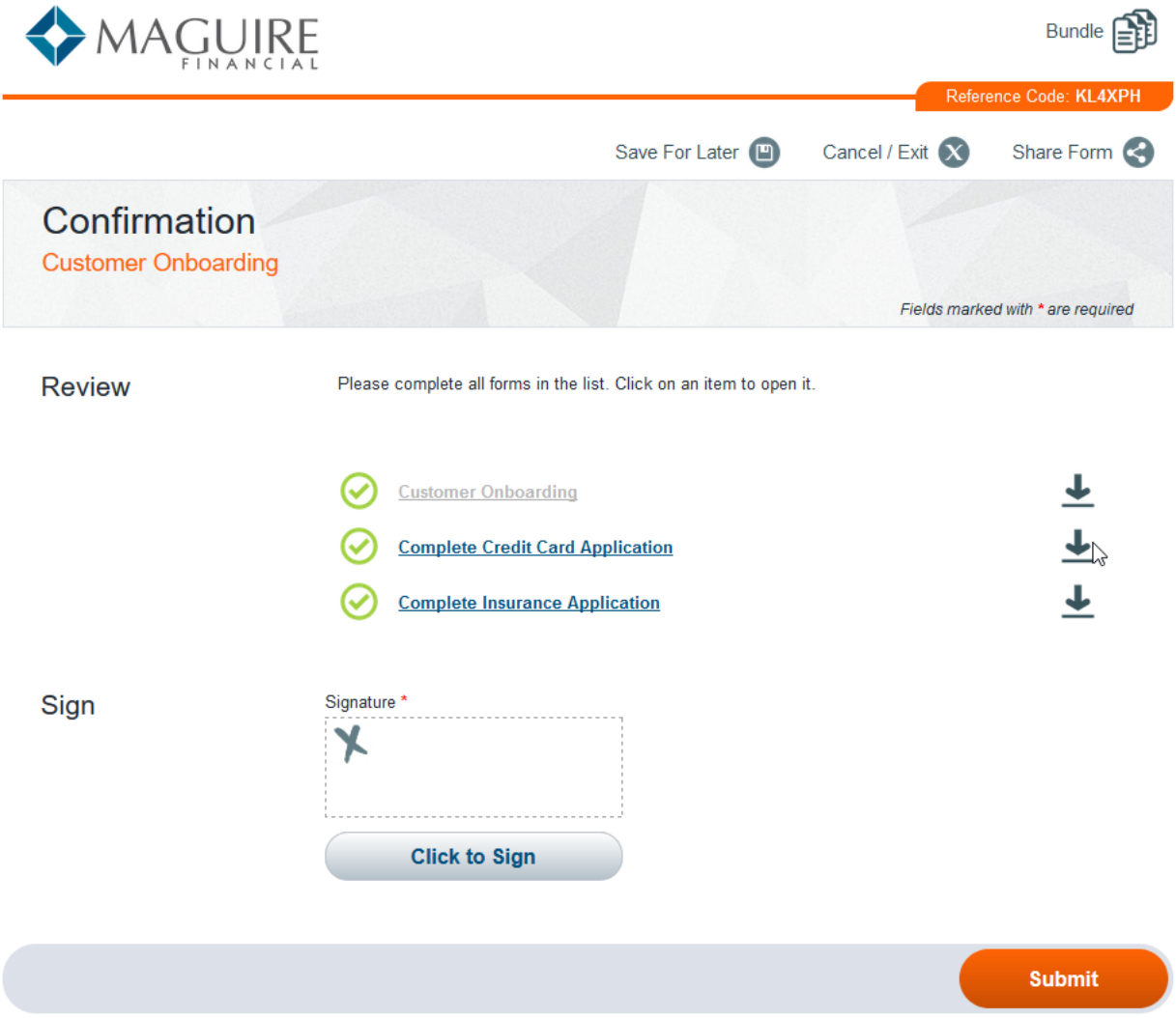
1. The Customer Onboarding - Credit Card form, if the user selected the **Credit Card** checkbox. In this case, the precondition is **`"preCondition": "$formDataMap.productCreditCards == 'true'"`**.



2. The Customer Onboarding - Insurance form, if use user selected the Insurance checkbox. In this case, the pre-condition is **`"preCondition": "$formDataMap.productInsurance == 'true'"`**.

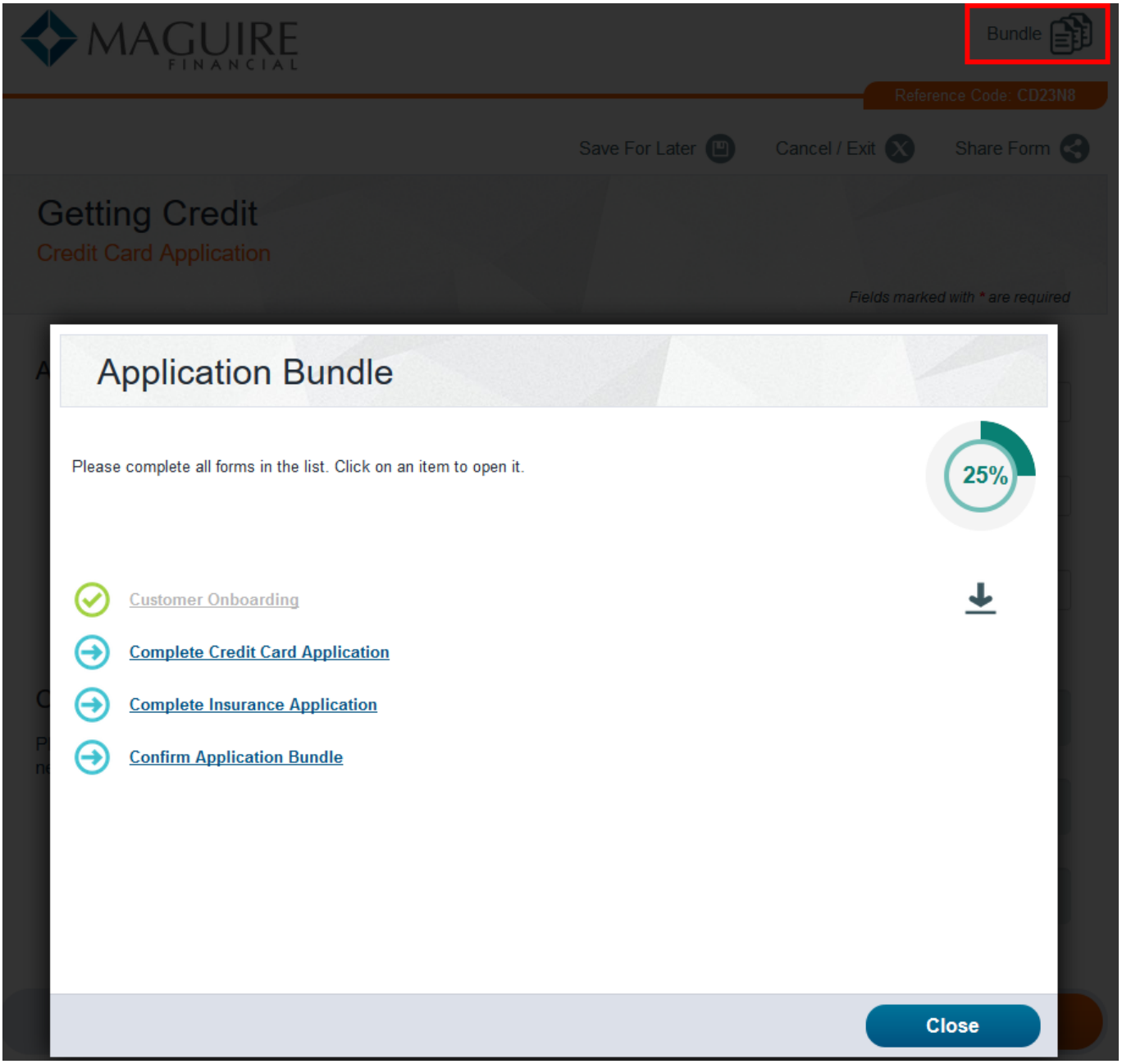


3. The Customer Onboarding - Confirmation form to allow the user to complete and confirm the application with a signature.

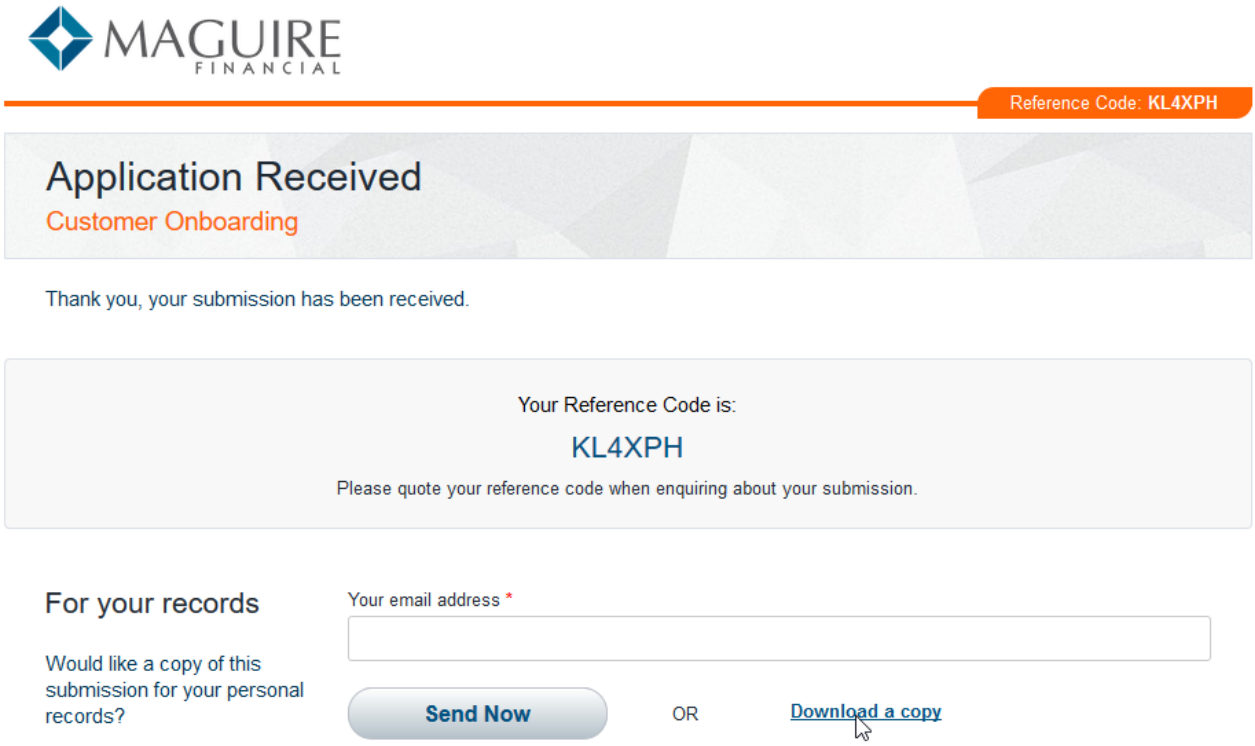


The Redirect Next functionality takes the user to the next form or task that has not be completed yet. A happy day scenario opens the credit card form, and after submitting it, the next form will be the insurance form, followed with the confirmation page.

The Maguire template has a form bundle option, the **Bundle** icon, which you can tun on or off. Click **Bundle** to open the **Application Bundle** navigation window to move to other forms in the bundle in a different to the Redirect Next order.



The confirmation page list the other forms (triage, credit card and insurance) so a user can download and review the receipts. At this stage they click on the link and go back and edit / modify any of the other forms in the Additional Product steps, such as the credit card and insurance. Once the user is happy with the forms they can sign and submit. The Additional products step has completed and the forms cannot be edited. The Customer Onboarding - Confirmation forms job action does not have redirectNext set. It goes to Application Received screen where they can download a consolidated receipt containing the Triage and Additional Product forms.



Click download a copy to view the PDF report.

For this example, you need to configure forms with the following form codes:

- Customer Onboarding : onboard-customer
- Customer Onboarding - Confirmation : onboard-confirmation
- Customer Onboarding - Credit Cards: onboard-credit-cards
- Customer Onboarding - Insurance : onboard-insurance

You must create a Customer Onboarding Job - v1 collaboration job and link it to the Customer Onboarding form, which is the start or triage form. The job definition is shown below:

```
{
  "jobDetails": {
    "name": "Customer Onboarding Job",
    "processSubmitImmediate": true,
    "version": "4.3.0"
  },
  "steps": [
    {
      "name": "Customer Onboarding",
      "type": "start",
      "actions": [
        {
          "name": "Customer Submission",
          "type": "Job Form Start",
          "redirectNext": true
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Additional Products" }
      ]
    },
    {
      "name": "Additional Products",
      "type": "",
      "shareExtractData": true,
      "allFormsEditable": true,
      "showPreviousForms": true,
      "redirectNext": true,
      "actions": [
        {
          "name": "Credit Cards Application",
          "type": "Job Task Assign",
          "preCondition": "$formDataMap.productCreditCards == 'true'",
          "redirectNext": true,
          "properties": [
            { "name": "Task Assign Email",  "value": "$formDataMap.emailAddress" },
            { "name": "Task Form Code",  "value": "onboard-credit-cards" },
            { "name": "Task Message",  "value": "Please complete the Credit Card Applic-
ation form." },
            { "name": "Task Subject",  "value": "Complete Credit Card Application" },
            { "name": "Task Input XML Prefill",  "value": "$func.startSubmissionXml()" },
            { "name": "Task Type",  "value": "Anonymous" }
          ]
        },
```

```
        {
          "name": "Insurance Application",
          "type": "Job Task Assign",
          "preCondition": "$formDataMap.productInsurance == 'true'",
          "redirectNext": true,
          "properties": [
            { "name": "Task Assign Email",  "value": "$formDataMap.emailAddress" },
            { "name": "Task Form Code",  "value": "onboard-insurance" },
            { "name": "Task Message",  "value": "Please complete the Insurance Applic-
ation form." },
            { "name": "Task Subject",  "value": "Complete Insurance Application" },
            { "name": "Task Input XML Prefill",  "value": "$func.startSubmissionXml()" },
            { "name": "Task Type",  "value": "Anonymous" }
          ]
        },
        {
          "name": "Appliction Confirmation",
          "type": "Job Task Assign",
          "preCondition": "$formDataMap.productInsurance == 'true' || $formDataMap.-
productCreditCards == 'true'",
          "properties": [
            { "name": "Task Assign Email",  "value": "$formDataMap.emailAddress" },
            { "name": "Task Form Code",  "value": "onboard-confirmation" },
            { "name": "Task Message",  "value": "Please sign to complete Application
bundle." },
            { "name": "Task Subject",  "value": "Confirm Application Bundle" },
            { "name": "Task Input XML Prefill",  "value": "$func.startSubmissionXml()" },
            { "name": "Task Type",  "value": "Anonymous" }
          ]
        },
        {
          "name": "Review Wait",
          "type": "Job Task Wait"
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Application Delivery" }
      ]
    },
    {
      "name": "Application Delivery",
      "type": "",
      "actions": [
        {
```

```
          "name": "Application Delivery",
          "type": "Job Delivery",
          "properties": [
            { "name": "Delivery Mode",  "value": "All Submissions" }
          ]
        },
        {
          "name": "Application Delivery Wait",
          "type": "Job Delivery Wait"
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Application Completed" }
      ]
    },
    {
      "name": "Application Completed",
      "type": "endpoint"
    }
  ]
}
```

You can download the complete code example here.

Next, learn more about dynamic customer onboarding collaboration jobs.

Dynamic Customer Onboarding Collaboration Jobs

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A dynamic customer onboarding collaboration job implements a simple customer onboarding dynamic form bundle. The bundle consists of a number of separate forms that are filled out by a single user in the one step. This is the enhancement of the standard customer onboarding form bundle with dynamic preconditions.

The standard form bundle allows a user to fill a triage form and submits it. On submit, form data extracts are created and used to evaluate preconditions when the bundle step is created. Dynamic bundles allow the preconditions to be re-evaluated after each form in the bundle is submitted.

Next, learn about step expiry.

Maestro Customer Onboarding Collaboration Jobs

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A customer onboarding collaboration job implements the simplest customer onboarding form bundle using a Journey Maestro form. It uses the same configuration and functionality as described in customer onboarding Collaboration Jobs.

You can download the complete code example here.

The forms require Maestro version 5.1 and higher. Forms and collaboration jobs should run on Manager version 5.0 and higher. To start the application, you can run the **Maestro Onboarding - Triage** form.

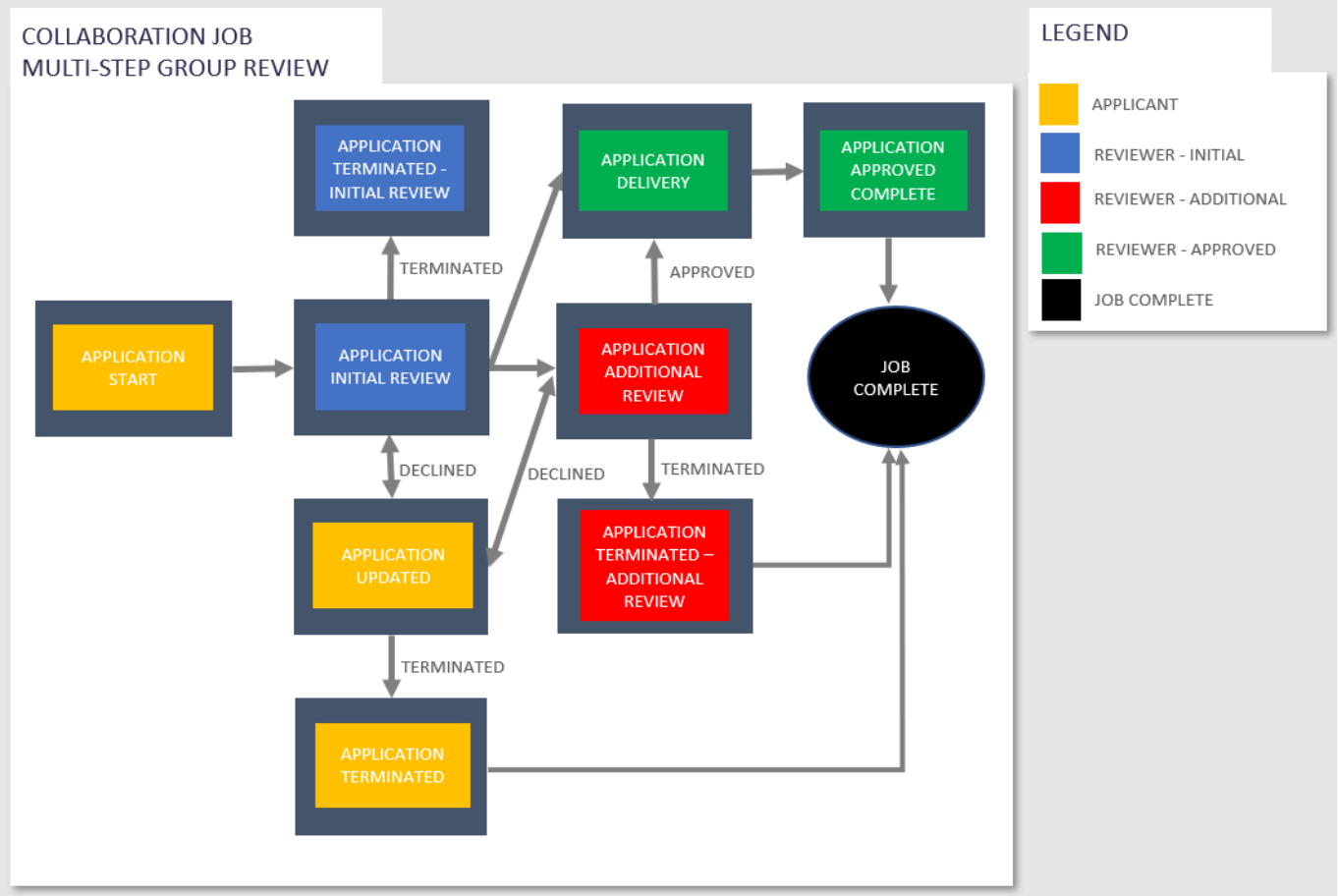
Next, learn more about dynamic customer onboarding Collaboration Jobs.

Multi-Step Group Review Collaboration Jobs

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

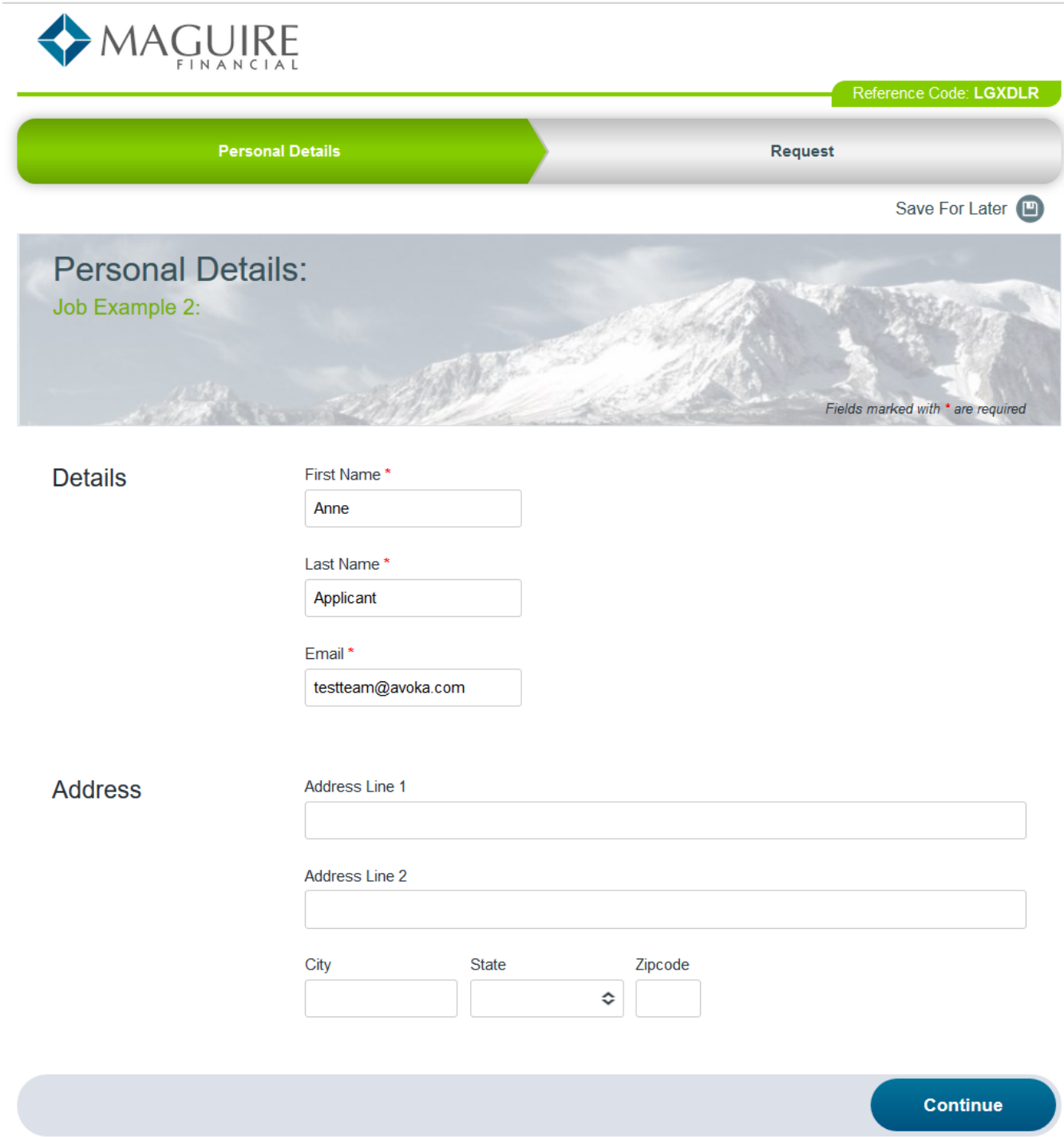
After a user submits an application form, it has to be reviewed and a decision has to be made whether to approve or decline the application. A review workflow can be as simple as one step review or a multi-step group review. The later is a collaboration job that implements the simplest group review process, so we will use to illustrate how a collaboration job works. The **Job Example 2** implements the review workflow using the **Form - Job Example 2** form and the **Review and Approval - Multi Step Group Review** job controller, which is the job service template.

The review workflow is shown below.



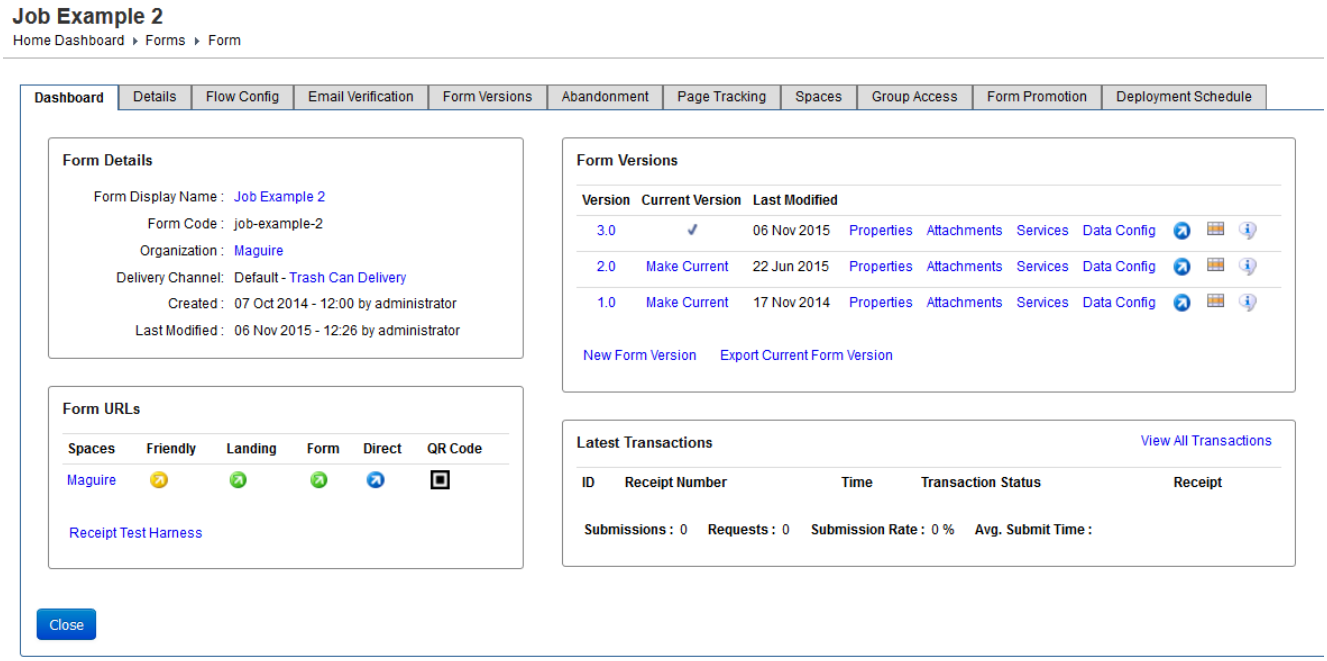
Configure a Form

Below is the **Job Example 2** form, which is built using our Composer product.

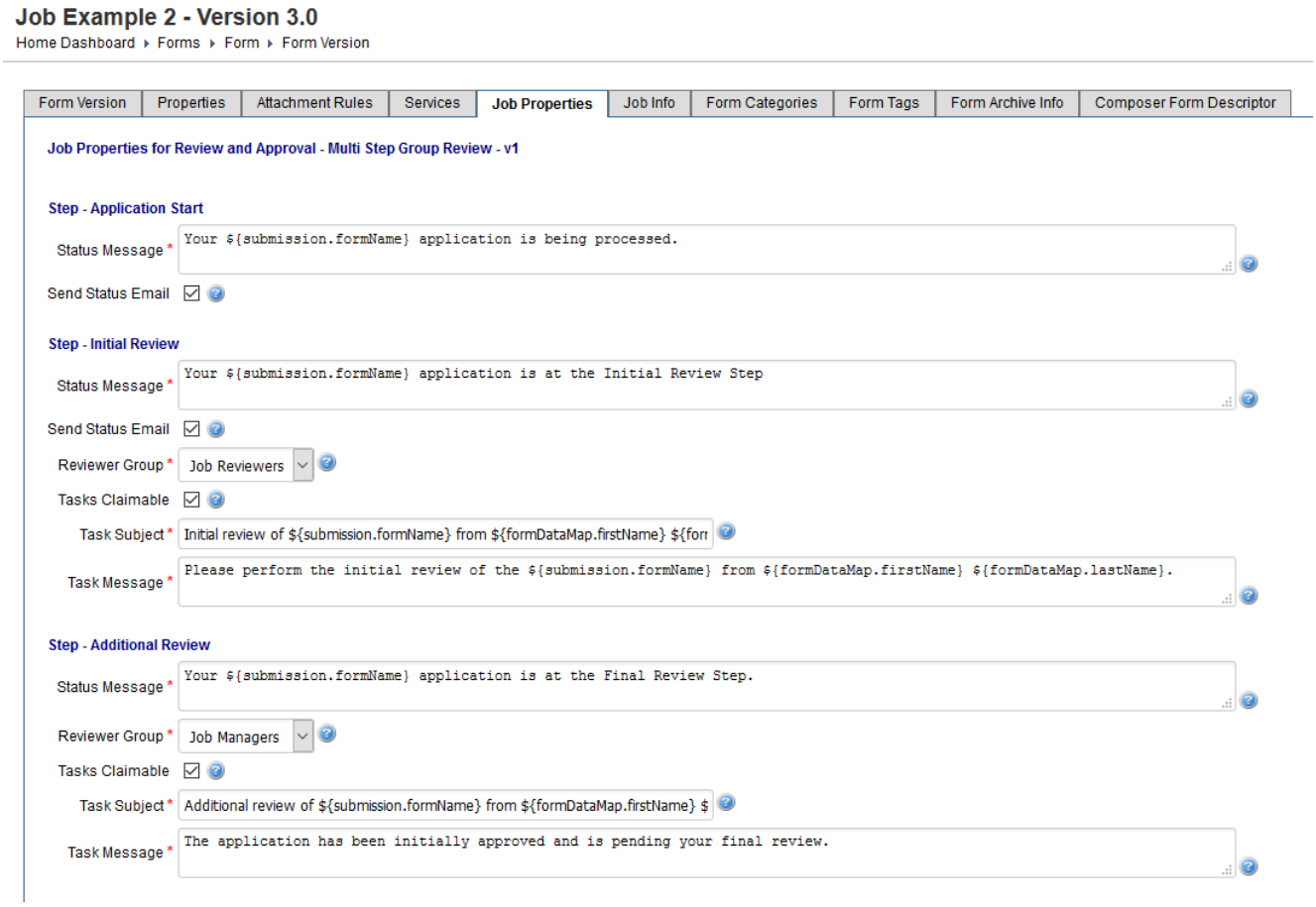

MAGUIRE
FINANCIAL
Reference Code: LGXDLR
Personal Details
Request
Save For Later
Personal Details:
Job Example 2:
Fields marked with * are required
Details
First Name *
Anne
Last Name *
Applicant
Email *
testteam@avoka.com
Address
Address Line 1
Address Line 2
City
State
Zipcode
Continue

Configure Manager

1. Open the **Job Example 2** form.



2. Select the form version 3 because it has the following features:
   - The visibility and editability rules based on the collaboration job step.
   - The radio buttons.
   - The job service template.
3. Click **Services** of the latest form version to configure its job controller service and select the **Review and Approval - Multi Step Group Review - v1** from the **Job Controller Service** dropdown list as the template.
4. Select the **Job Properties** tab to configure the **Reviewer Groups** as shown below. Properties can use data extracts and other model elements.



5. Check the **Review and Approval - Multi Step Group Review** job controller's definitions, as shown below.

```
{
  "jobDetails": {
    "name": "Review and Approval - Multi Step Group Review",
    "version": "4.2.0"
  },
  "properties": [
    { "name": "Task Form Code",  "value": "$func.startFormCode()" }
  ],
  "steps": [
    {
      "name": "Start",
      "type": "start",
      "actions": [
        {
          "name": "Handle Submission",
          "type": "Job Form Start",
          "properties": [
            { "name": "Process Message Send Email",  "value": "$func.formProperty
('Start Send Email')" },
            { "name": "Process Message Text",  "value": "$func.formProperty
('Start Process Message')" }
          ]
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Initial Review" }
      ]
    },
    {
      "name": "Initial Review",
      "type": "",
      "actions": [
        {
          "name": "Create Task",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Process Message Send Email",  "value": "$func.formProperty
('Initial Review Send Email')" },
            { "name": "Process Message Submission Step",  "value": "Start" },
            { "name": "Process Message Text",  "value": "$func.formProperty('Ini-
tial Review Process Message')" },
            { "name": "Task Assign Group",  "value": "$func.formProperty('Initial
```

```
Review Group')" },
            { "name": "Task Enable Claiming",  "value": "$func.formProperty('Ini-
tial Review Enable Claiming')" },
            { "name": "Task Message",  "value": "$func.formProperty('Initial
Review Task Message')" },
            { "name": "Task Review Previous Step",  "value": "true" },
            { "name": "Task Subject",  "value": "$func.formProperty('Initial
Review Task Subject')" },
            { "name": "Task Type",  "value": "Review" }
          ]
        },
        {
          "name": "Handle Submission",
          "type": "Job Task Wait",
          "properties": [
            { "name": "Conditional Route Name",  "value": "#if (
$formDataMap.routeName == 'Approve' && $formDataMap.loanAmount >= 20000 ) Exceeds
Threshold #end" }
          ]
        }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" },
        { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "dis-
play": "false" },
        { "name": "Terminate",  "nextStep": "Terminated Initial" }
      ]
    },
    {
      "name": "Additional Review",
      "type": "",
      "actions": [
        {
          "name": "Create Task",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Process Message Submission Step",  "value": "Start" },
            { "name": "Process Message Text",  "value": "$func.formProperty('Addi-
tional Review Process Message')" },
            { "name": "Task Assign Group",  "value": "$func.formProperty('Addi-
tional Review Group')" },
```

```
            { "name": "Task Enable Claiming",  "value": "$func.formProperty('Addi-
tional Review Enable Claiming')" },
            { "name": "Task Message",  "value": "$func.formProperty('Additional
Review Task Message')" },
            { "name": "Task Review Previous Step",  "value": "true" },
            { "name": "Task Subject",  "value": "$func.formProperty('Additional
Review Task Subject')" },
            { "name": "Task Type",  "value": "Review" }
          ]
        },
        {
          "name": "Handle Submission",
          "type": "Job Task Wait"
        }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" },
        { "name": "Terminate",  "nextStep": "Terminated Additional" }
      ]
    },
    {
      "name": "Applicant Update",
      "type": "",
      "actions": [
        {
          "name": "Create Task",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Process Message Submission",  "value": "$fun-
c.startSubmission()" },
            { "name": "Process Message Text",  "value": "$func.formProperty
('Applicant Update Process Message')" },
            { "name": "Task Assign User",  "value": "$func.startUser()" },
            { "name": "Task Message",  "value": "$func.formProperty('Applicant
Update Task Message')" },
            { "name": "Task Review Previous Step",  "value": "true" },
            { "name": "Task Subject",  "value": "$func.formProperty('Applicant
Update Task Subject')" },
            { "name": "Task Type",  "value": "Review" }
          ]
        },
```

```
        {
          "name": "Handle Submission",
          "type": "Job Task Wait"
        }
      ],
      "routes": [
        { "name": "Update",  "nextStep": "##PREVIOUS_STEP" },
        { "name": "Terminate",  "nextStep": "Terminated Applicant" }
      ]
    },
    {
      "name": "Application Delivery",
      "type": "",
      "actions": [
        {
          "name": "Application Delivery",
          "type": "Job Delivery"
        },
        {
          "name": "Application Delivery Wait",
          "type": "Job Delivery Wait"
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Approved Complete" }
      ]
    },
    {
      "name": "Approved Complete",
      "type": "endpoint",
      "actions": [
        {
          "name": "Process Message",
          "type": "Job Process Message",
          "properties": [
            { "name": "Process Message Send Email",  "value": "$func.formProperty
('Approved Complete Send Email')" },
            { "name": "Process Message Submission Step",  "value": "Start" },
            { "name": "Process Message Text",  "value": "$func.formProperty
('Approved Complete Process Message')" }
          ]
        }
```

```
      ]
    },
    {
      "name": "Terminated Initial",
      "type": "endpoint",
      "actions": [
        {
          "name": "Process Message",
          "type": "Job Process Message",
          "properties": [
            { "name": "Process Message Send Email",  "value": "$func.formProperty
('Terminated Initial Send Email')" },
            { "name": "Process Message Submission Step",  "value": "Start" },
            { "name": "Process Message Text",  "value": "$func.formProperty('Ter-
minated Initial Process Message')" }
          ]
        }
      ]
    },
    {
      "name": "Terminated Additional",
      "type": "endpoint",
      "actions": [
        {
          "name": "Process Message",
          "type": "Job Process Message",
          "properties": [
            { "name": "Process Message Send Email",  "value": "$func.formProperty
('Terminated Additional Send Email')" },
            { "name": "Process Message Submission Step",  "value": "Start" },
            { "name": "Process Message Text",  "value": "$func.formProperty('Ter-
minated Additional Process Message')" }
          ]
        }
      ]
    },
    {
      "name": "Terminated Applicant",
      "type": "endpoint",
      "actions": [
        {
          "name": "Process Message",
```

```
          "type": "Job Process Message",
          "properties": [
            { "name": "Process Message Send Email",  "value": "$func.formProperty
('Terminated Applicant Send Email')" },
            { "name": "Process Message Submission Step",  "value": "Start" },
            { "name": "Process Message Text",  "value": "$func.formProperty('Ter-
minated Applicant Process Message')" }
          ]
        }
      ]
    }
  ]
}
```

Configure Users and Groups

This example uses the following users, which are the actors in the job definition above.

Group	Login	Name	Description
Job Applicants	anne	Anne Applicant	Anne is the applicant.
Job Reviewers	roger	Roger Reviewer	Roger looks after the Initial Review step.
Job Managers	mike	Mike Manager	Mike looks performs the Additional Review.
Job Coordinators	cat	Cat Coordinator	Cat looks after the collaboration job as a whole and needs to track where collaboration jobs are up to using the Reviews page.

Route Options

The review workflow has the following route options at each user's step.

Step	Group	Login	Description
Start	Job Applicants	anne	Anne starts the collaboration job (Start step) by the initial submission.
Initial Review	Job Reviewers	roger	Roger can either: **Approve**: A Conditional Route Rule checks if the loan amount is:

			• Below or equal to threshold the route stays as Approved and is routed to Application Delivery then to the Approved Complete endpoint. • Above the route is changes to Exceeds Threshold and is routed to the Additional Review. **Decline**: Routed to the Applicant Update Step **Terminate**: Routed to the Terminated Initial endpoint
Additional Review	Job Managers	mike	Mike can either **Approve**: Routed to Application Delivery step then to the Approved Complete endpoint, **Decline**: The reviewer enters feedback and the job is routed to the Applicant Update Step **Terminate**: Routed to the Terminated Additional endpoint
Applicant Update	Job Applicants	anne	Anne can read the feedback from the reviewer, plus she can: **Update**: Anne can update the form and the route it back to the review step that declined the application. **Terminate**: Routed to the Terminated Applicant endpoint.

You can see that the Roger's **Approve** action, the Conditional Route Rule, is the Conditional Route Name. The following action property belongs to the Task Wait Service, which defines a rule that is applied with the collaboration job on the Manager server side. The rule is shown below:

```
{ "name": "Conditional Route Name",  "value": "#if ( $formDataMap.routeName == 'Approve'
&& $formDataMap.loanAmount >= 20000 ) Exceeds Threshold #end" }
```

The following example shows the listed routes where the Exceeds Threshold has the display attribute set to false, so this route is not put in the Available Routes, as shown below:

```
"routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Decline",  "nextStep": "Applicant Update" },
        { "name": "Exceeds Threshold",  "nextStep": "Additional Review",  "display":
"false" },
        { "name": "Terminate",  "nextStep": "Terminated Initial" }
      ]
```

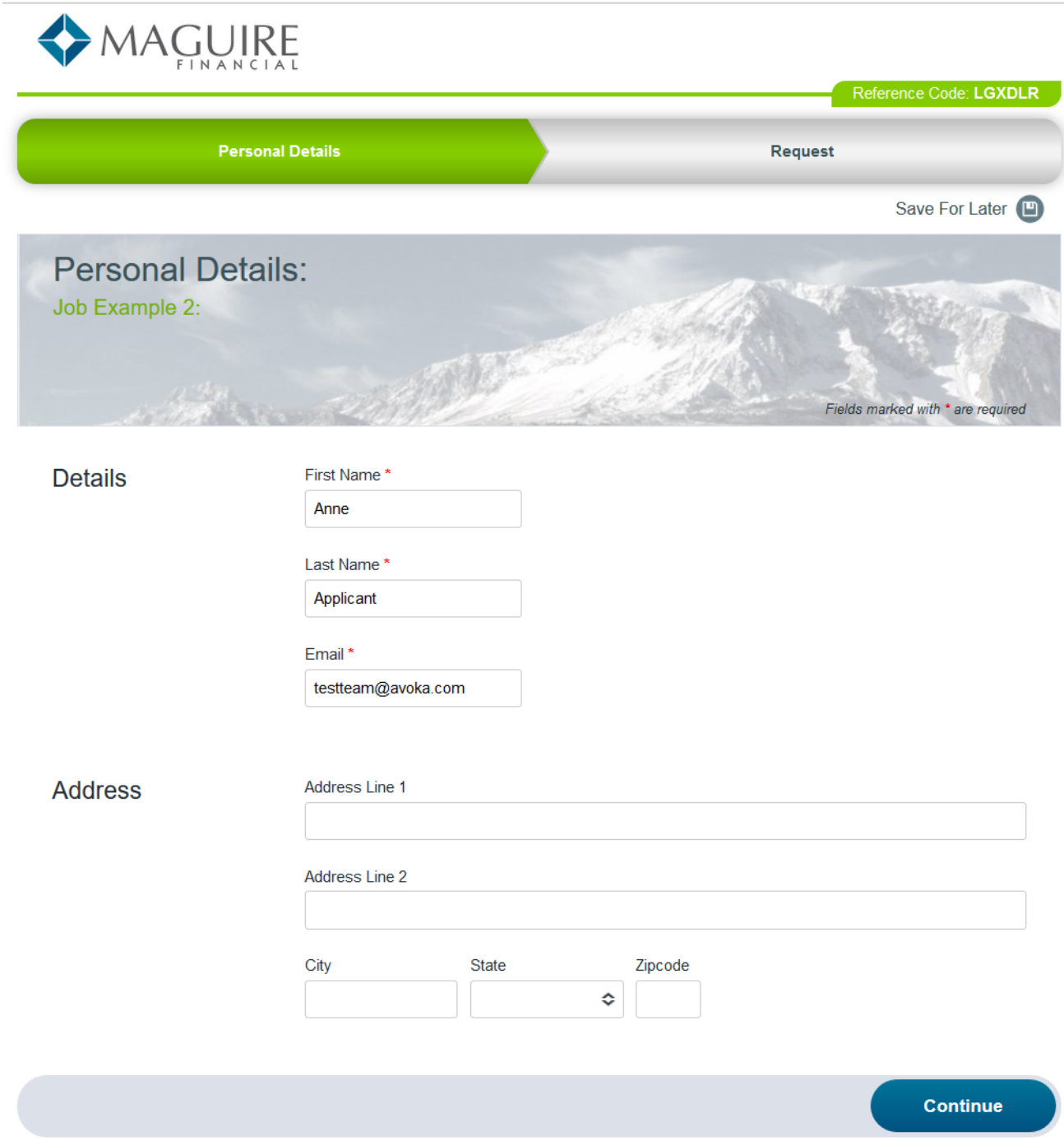
The Anne's **Update** action is a previous step. The following shows the routes for the Applicant Update step. When the Initial Review declines it will be sent back to the Initial Review, and same with the Additional Review.

```
"routes": [
        { "name": "Update",  "nextStep": "##PREVIOUS_STEP" },
        { "name": "Terminate",  "nextStep": "Terminated Applicant" }
      ]
```

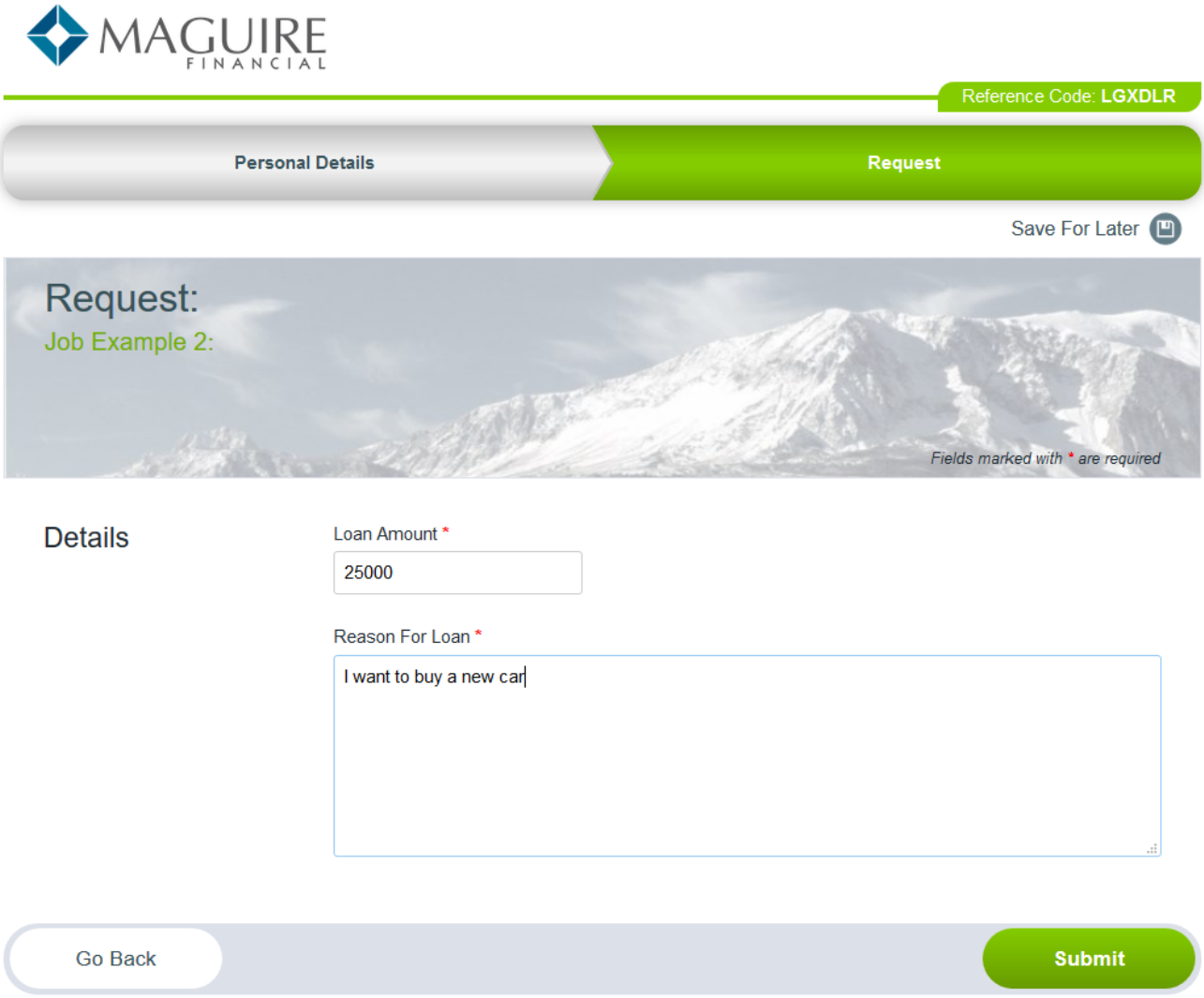
Now, you can run the example and see how collaboration job works.

Form - Anne

1. Log on to Work Space as Anne.
2. Fills in the personal details.



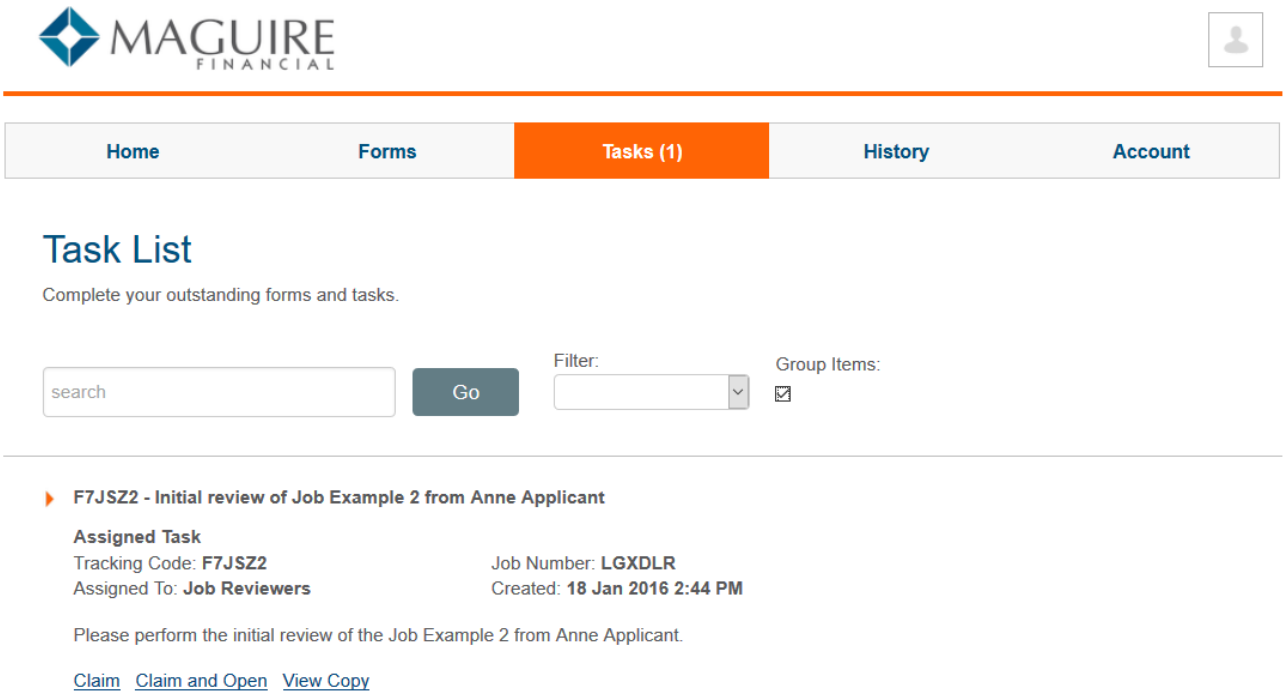
3. Enter the desired loan amount and the reason for the loan.



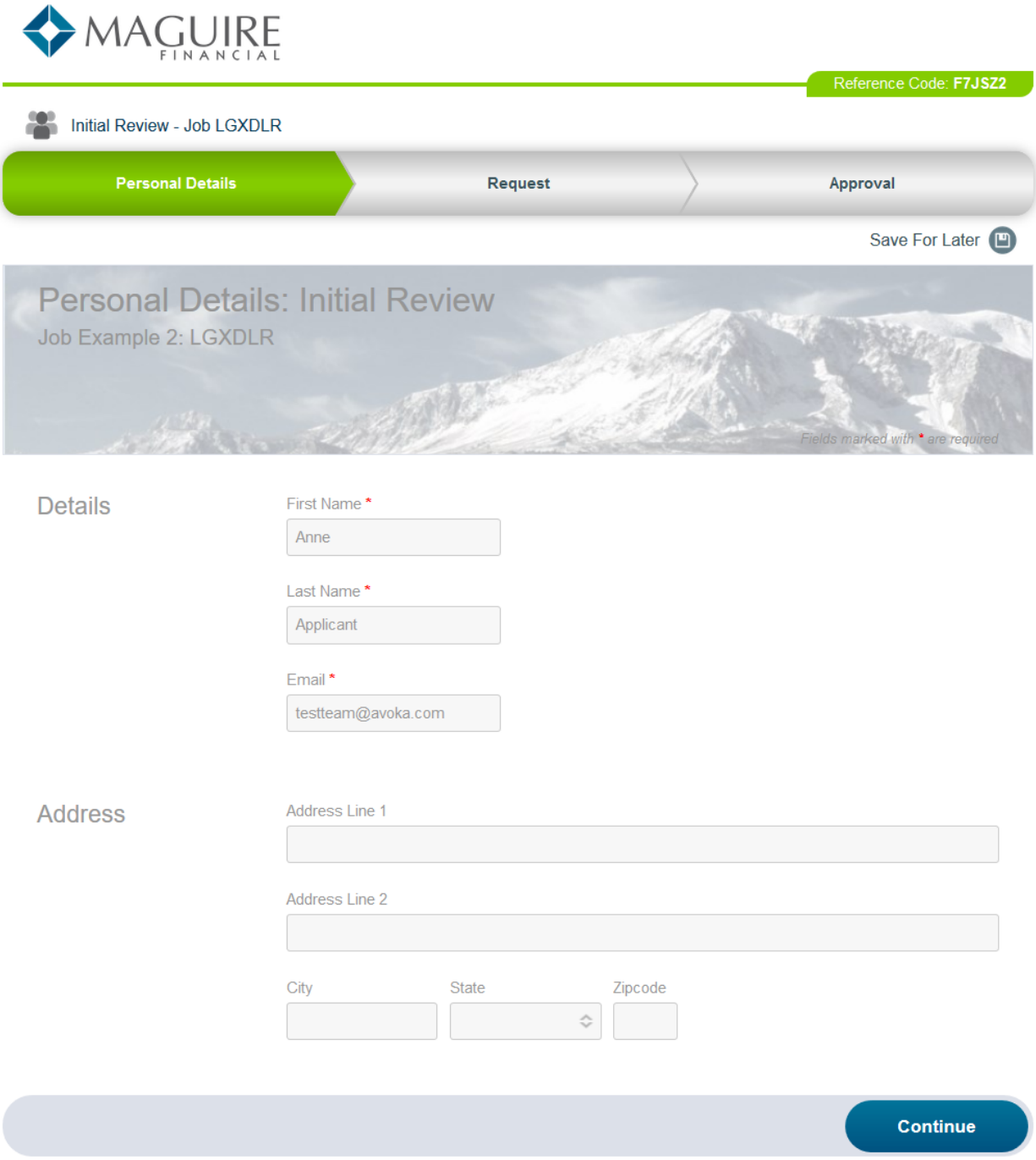
4. Click **Submits**.

Initial Review - Roger

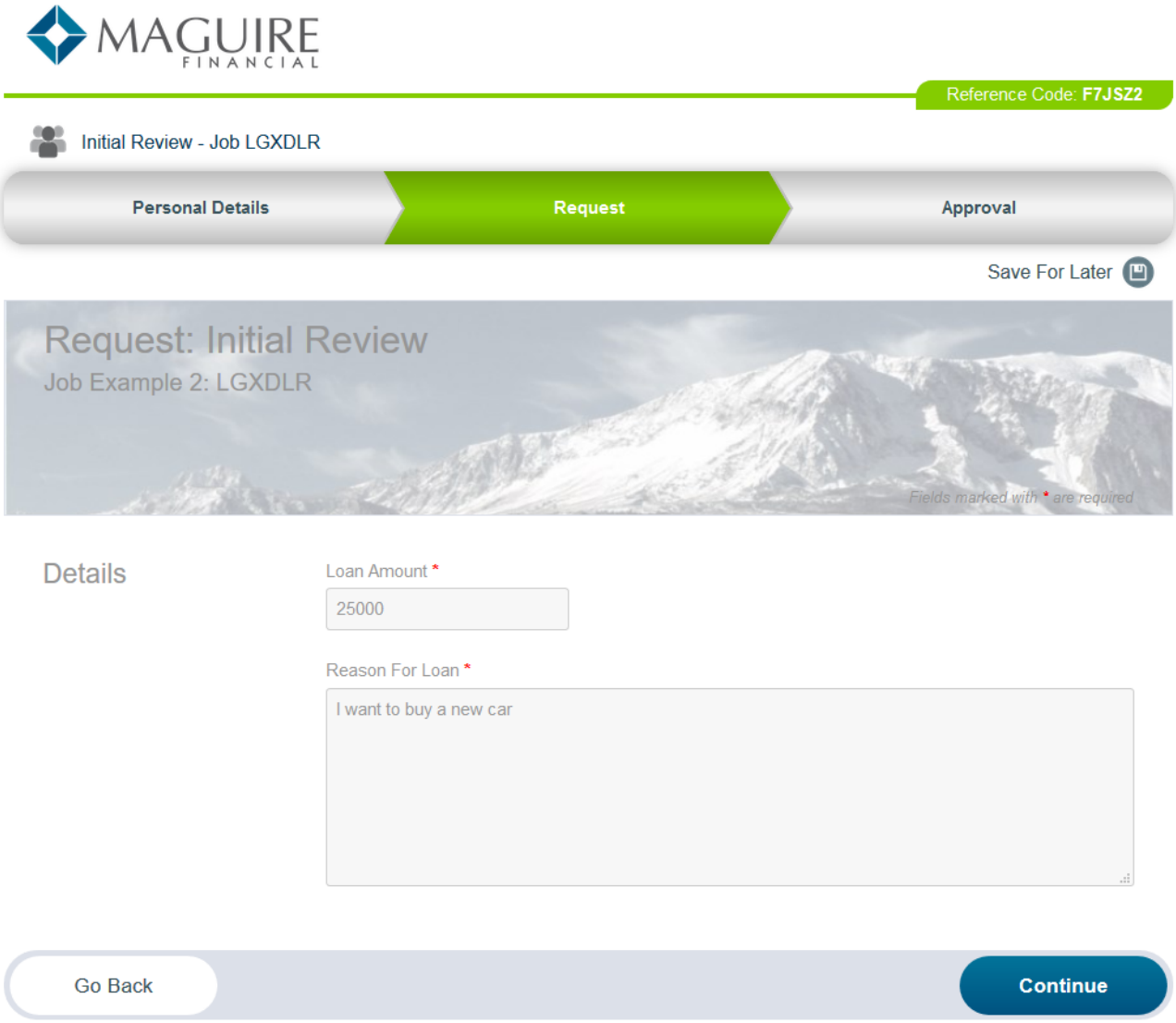
1. Log on to Work Space as Roger.
2. Check the group task, claim it and open it for the review.



3. Review the applicant details' section.



4. Review the application loan amount and the reason for the loan.



5. The application doesn't have sufficient information to justify the loan, so Roger decides to decline the application. He selects the **Decline** radio button and adds a comment for Anne "What car do you want?".

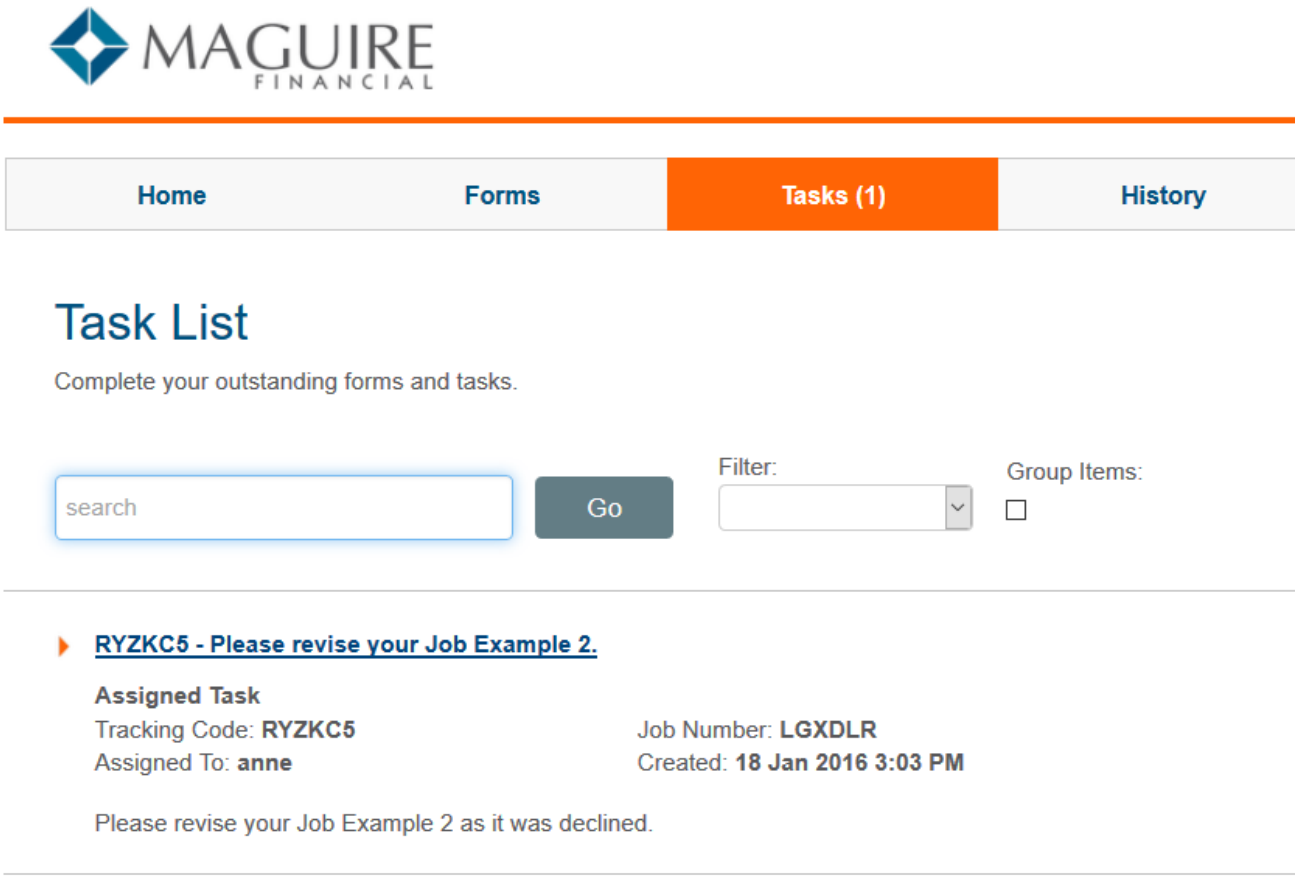


6. The application goes back to Anne, so she can update or terminate (cancel) it.

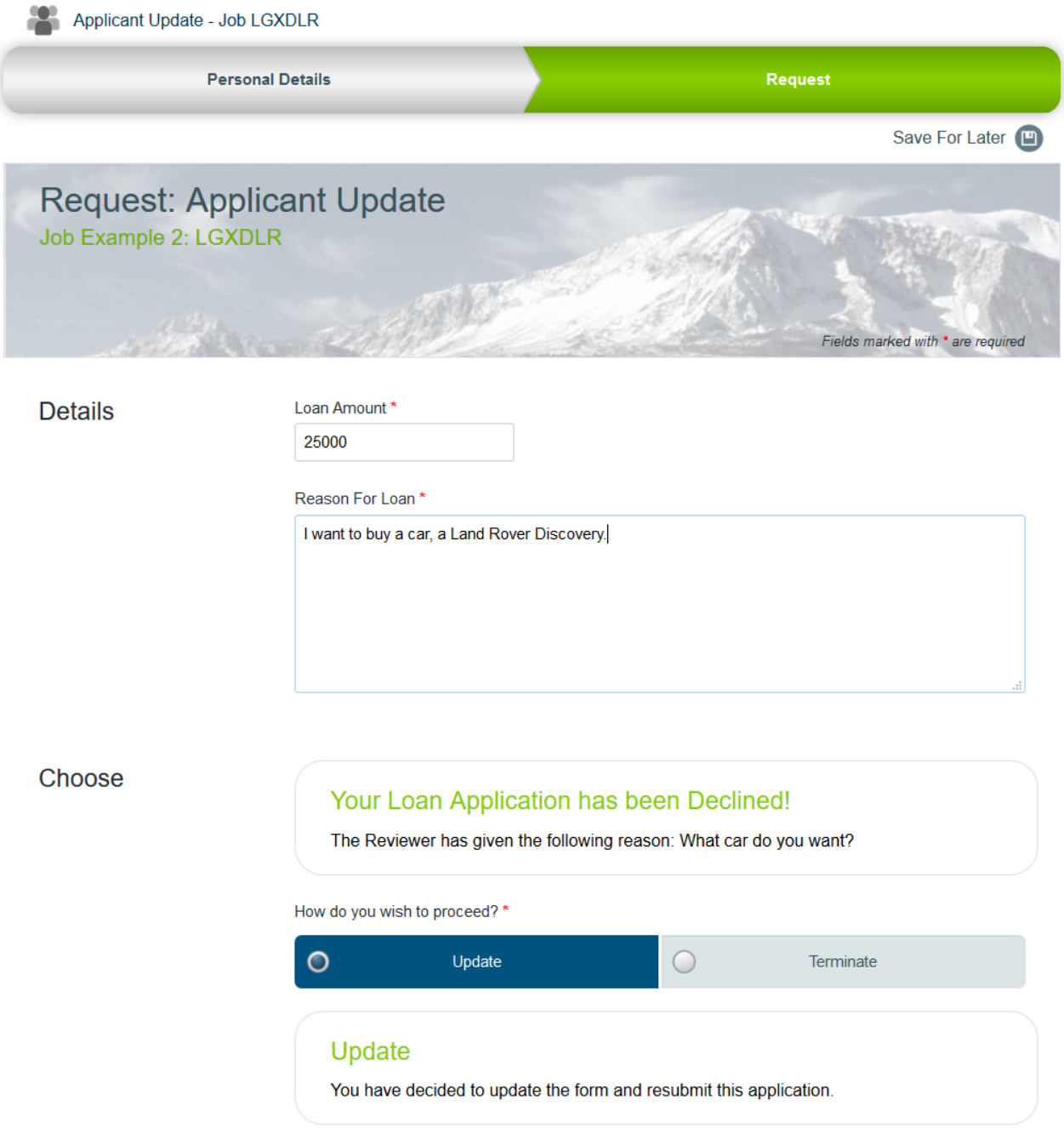
The routes in the form use radio buttons. If you use more that one radio button groups to control the Route Name, see collaboration job configuration in Composer.

Applicant Update - Anne

1. Log on to Work Space as Anne.
2. Check the tasks assigned to her.



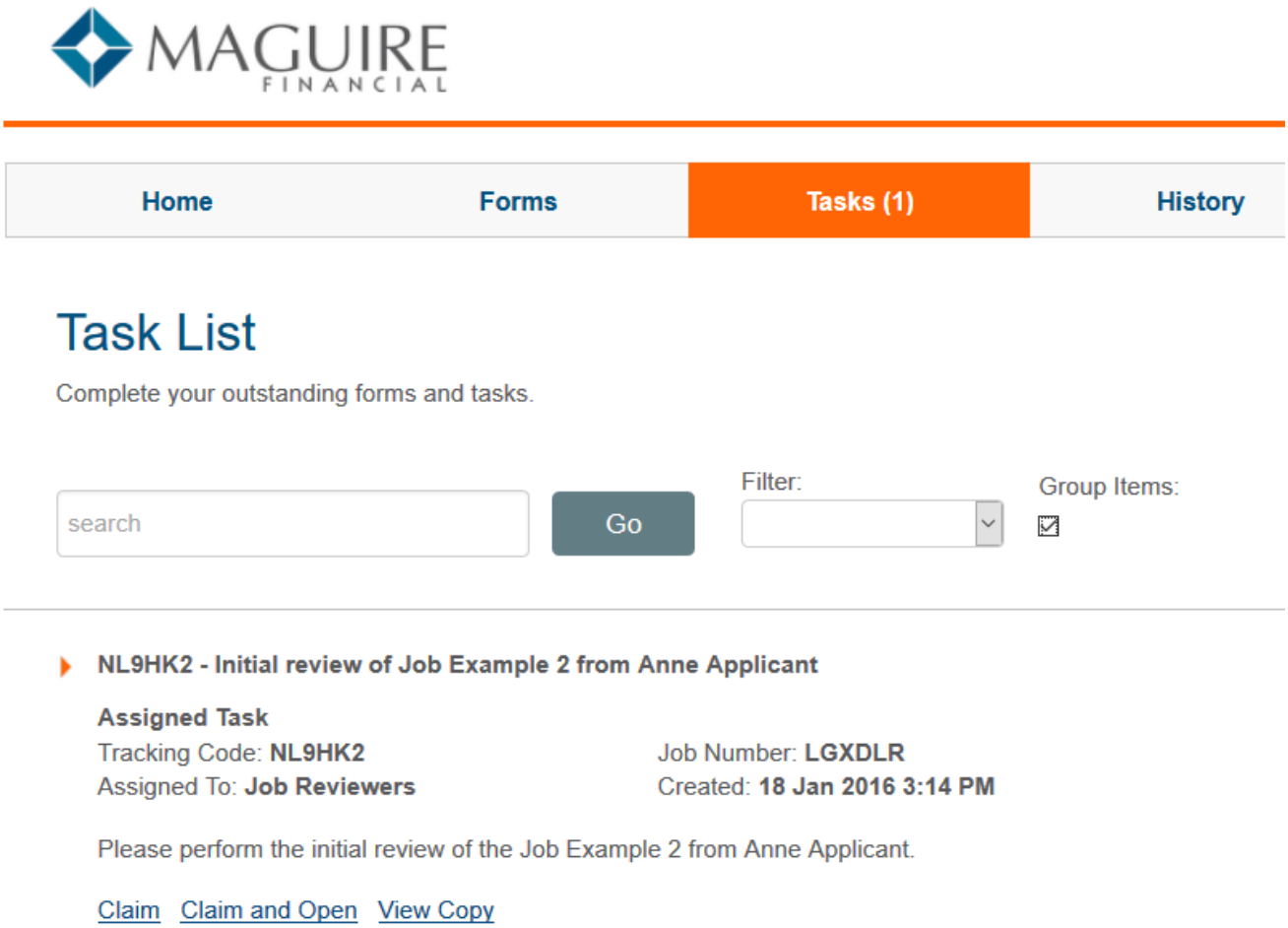
3. Open the form and select the **Request** page that now has the **Choose** section so Anne can either update or terminate (cancel) the application.



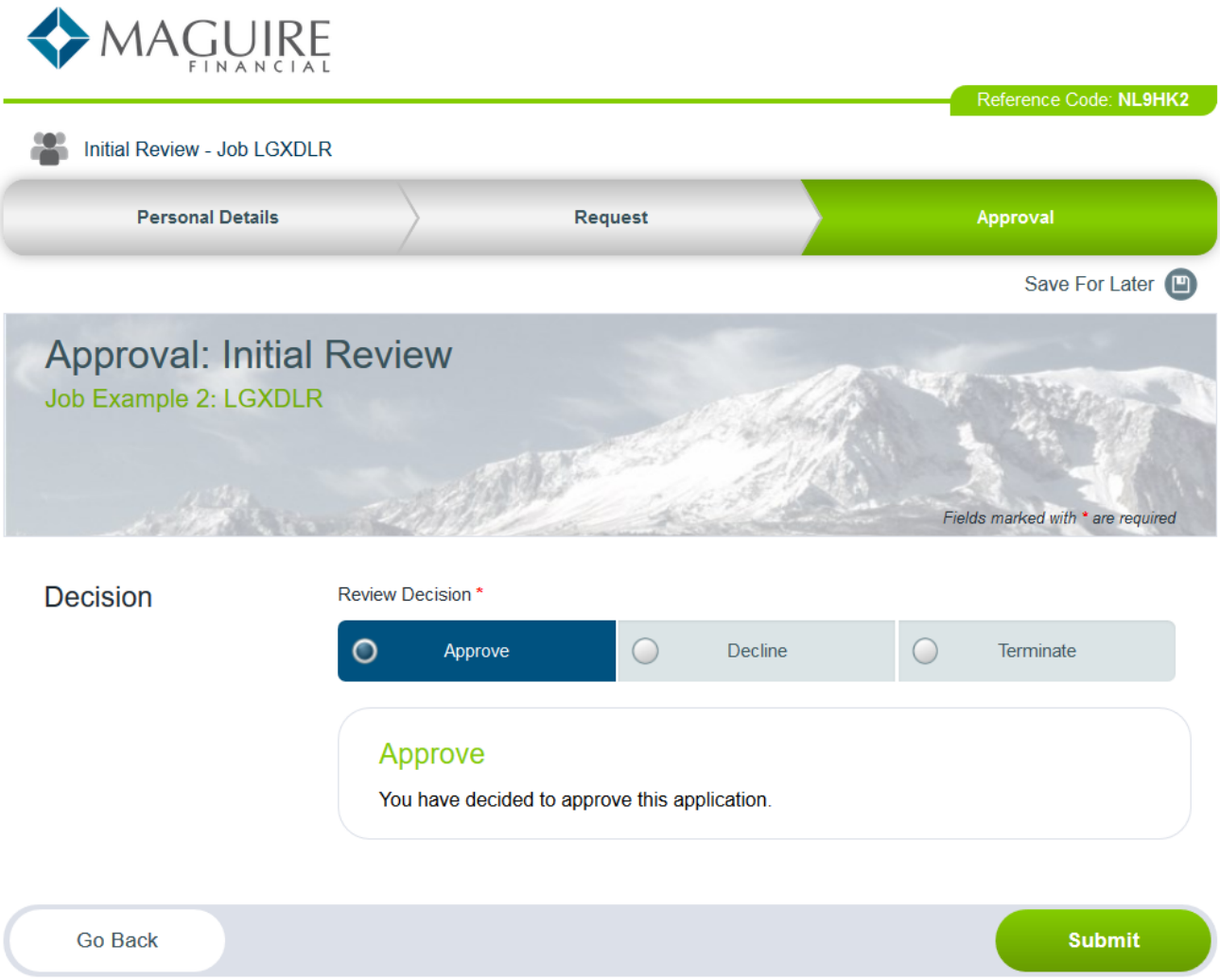
4. Select the **Update** radio button to provide the required additional information, such as the new car make in the **Reason For Loan** field.
5. Clicks **Submit.**

Initial Review - Roger

1. Log on to Work Space as Roger.
2. View the updated task.



3. Click **Claim and Open** to view the **Request - Reason For Loan** section. The updated comment is sufficient, so the application can be approved.

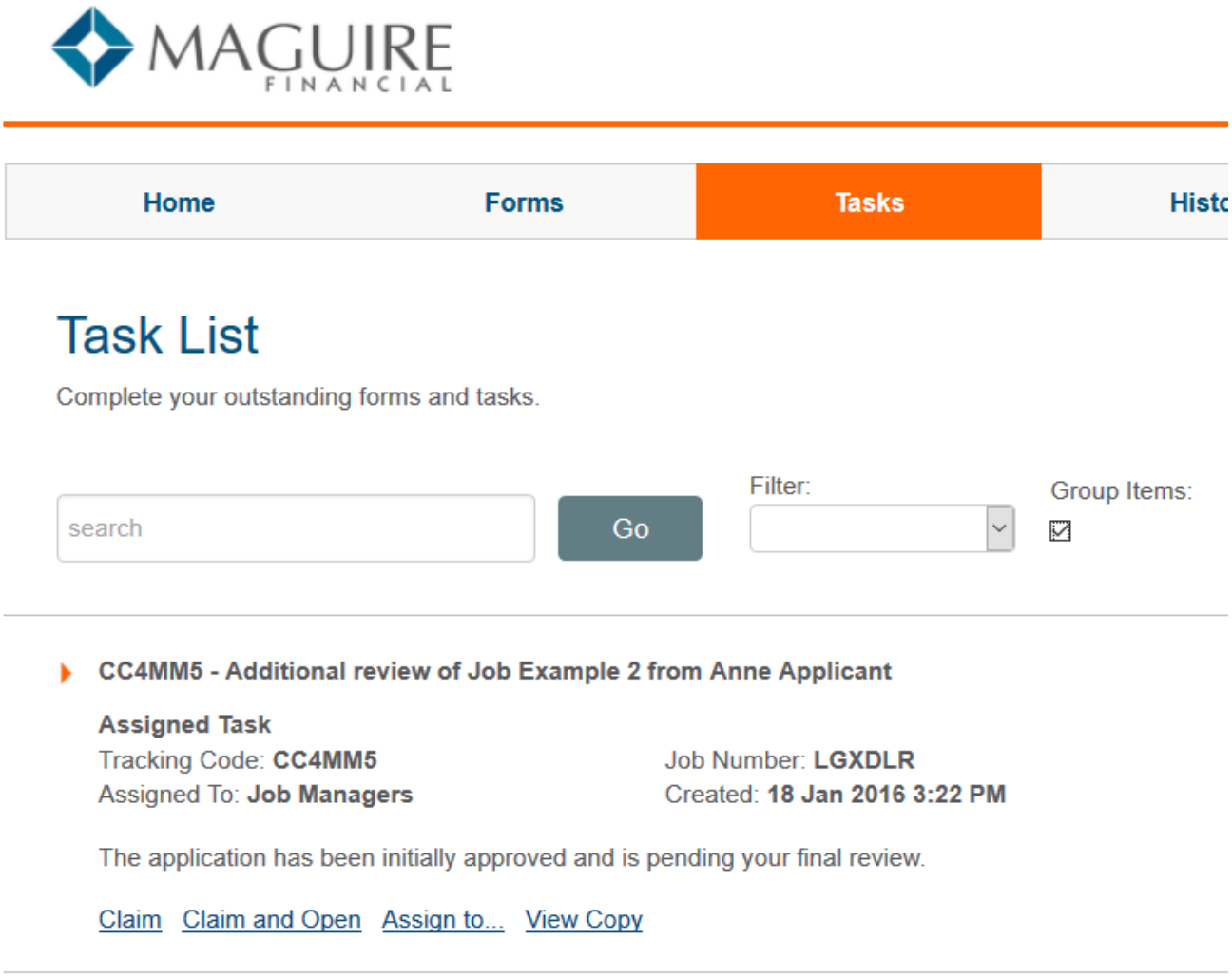


4. Select the **Approve** radio button to approve the application.

Additional Review - Mike

Because the loan amount was greater that $20,000, the collaboration job is routed to Mike.

1. Log on to Work Space as Mike.
2. View the task.



3. Click **Claim and Open** and review the **Personal Details** and **Request** sections.
4. Navigate to the **Approval** section and select the **Approve** radio button.
5. The application is approved now so the applicant loan will be served.

Below is the summary of what is visible and editable in each application form section.

Section	Start	Initial Review	Additional Review	Application Update
User Details	Editable	Read Only	Read Only	Editable
Request	Editable	Read Only	Read Only	Editable
Request - Choose	Hidden	Hidden	Hidden	Visible
Approve	Hidden	Visible	Visible	Hidden

This concludes the example.

You can download the complete code example here.

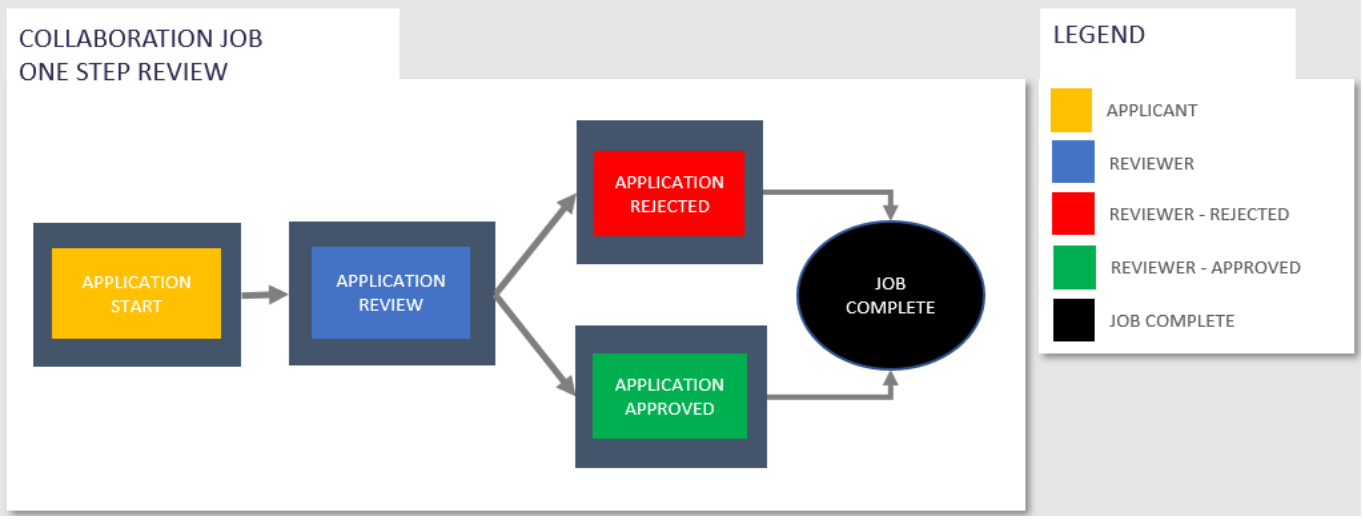
Next, learn about customer onboarding collaboration jobs.

One Step Review Collaboration Job

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

After a user submits an application form, it has to be reviewed and a decision has to be made whether to approve or decline the application. A one step review collaboration job implements the simplest review process, so we will use to illustrate how a collaboration job works. The **Job Example 1** implements the review workflow using the **Form - Job Example 1** form, which contains only the fields that are used in the collaboration job. It also explains how attachments are reviewed using the **In Form Attachment** widget.

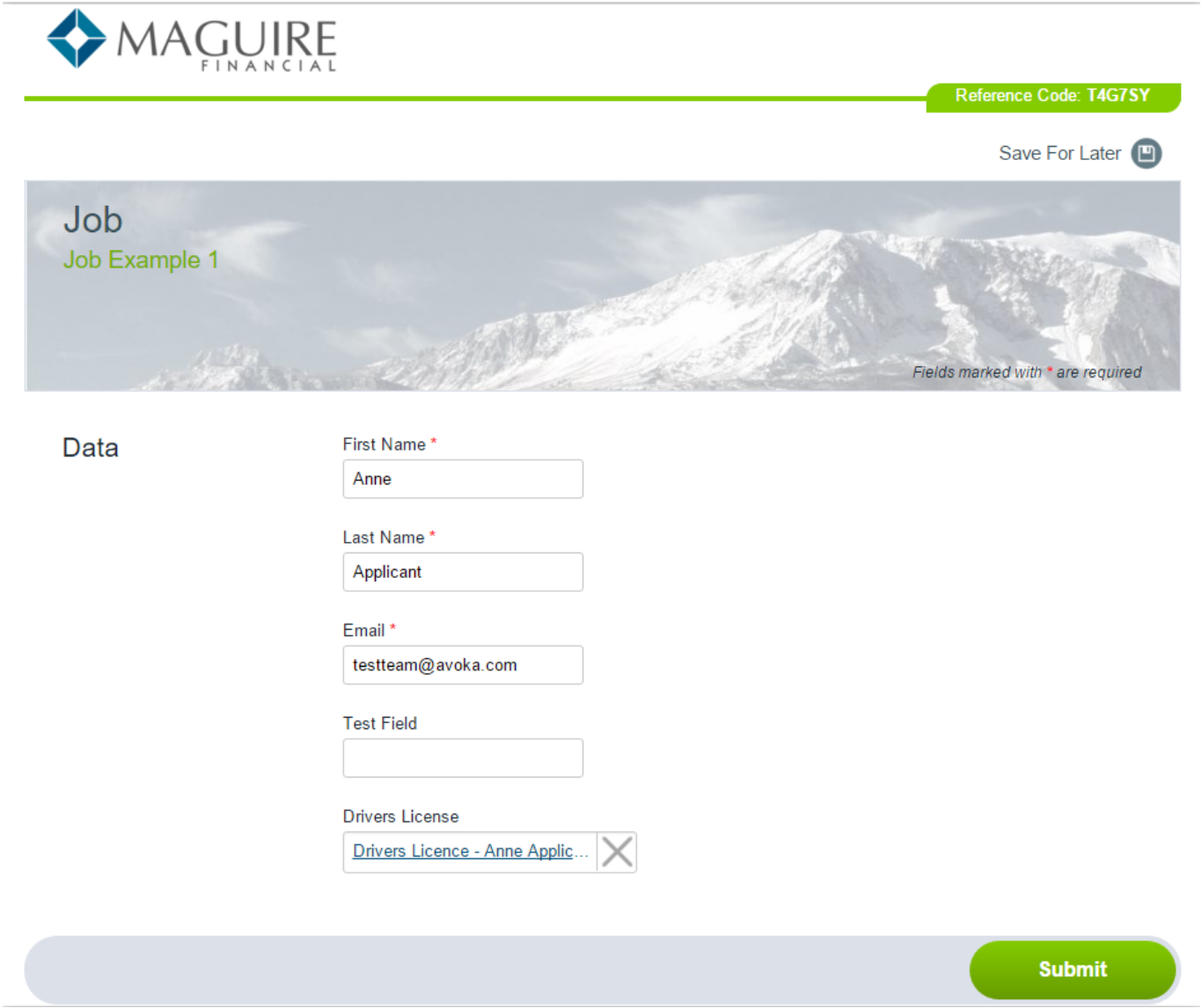
The review workflow is shown below.



1. A user fills in an application form and submits it.
2. The form goes to the **Application Review** step where it is reviewed by a member of the Job Reviewer group.
3. The reviewer may **Accept** or **Reject** the application:
   - If it is rejected, the collaboration job moves to the **Application Rejected** endpoint.
   - If it is accepted, the collaboration job progresses to the **Application Approved** endpoint and the form is delivered (this step is not shown) .

Configure a Form

Below is the **Job Example 1** form, which is built using our Composer product.



The form has the following mandatory fields, which are data extracts, which are populated when an authenticated user is logged into the form space. Forms can use the same Job Controller as long as they have the same data extract fields.:

- First Name - the Text Field
- Surname - the Text Field
- Email - the Email Field

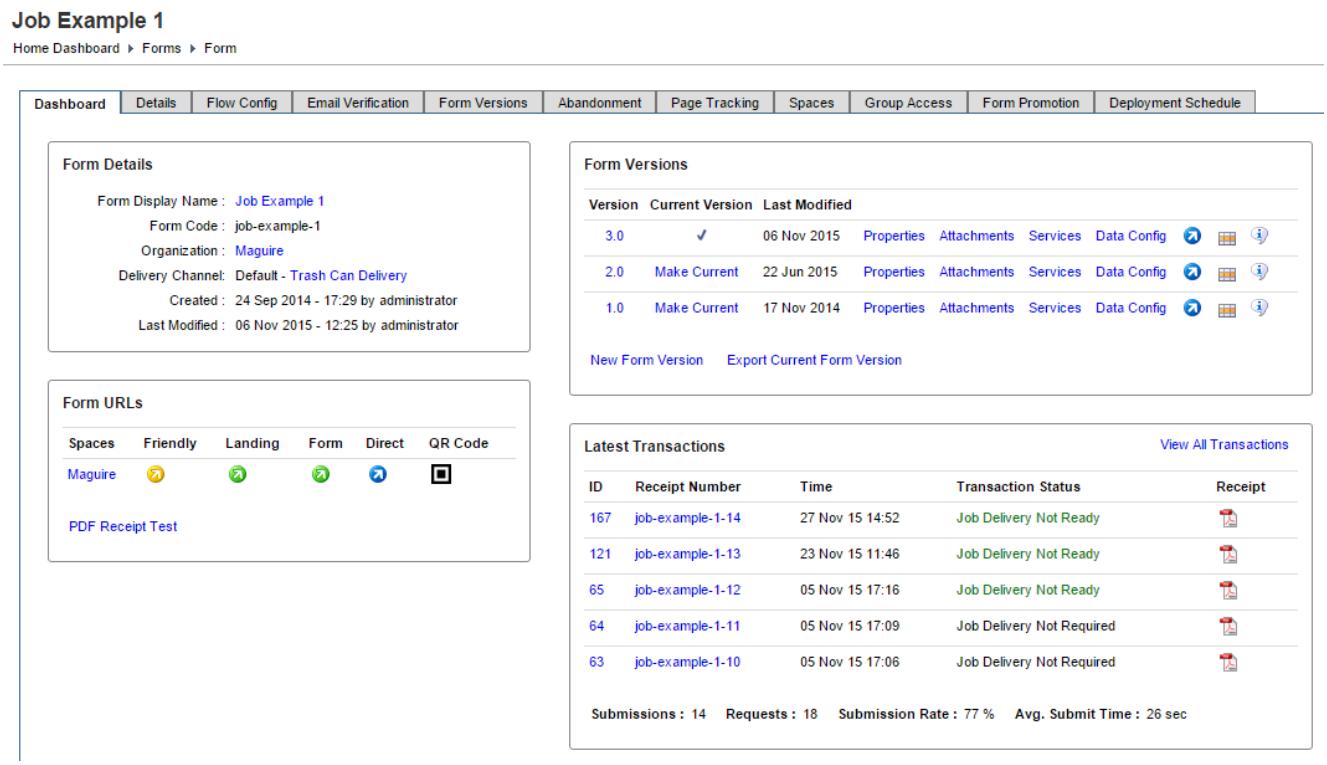
The form also has the optional fields:

- Test Field - the Text Field.
- Drivers License - the Inform Attachment Field, which is configured as read-only in the Application Review step.
- Route Name - the Collaboration Routes Dropdown list.

When the form is opened by the Applicant the Route Name Dropdown is hidden.This is because the Job hasn't started until this form has been submitted. As the form is not a Task that part of a Job and the job context isn't populated in the form data and hence the Available Routes Names are empty. The Route Name Dropdown field will appear when the form is a Task at the Application Review step Available Routed Names Approve and Reject associated with the step are listed in the form data.

Configure Manager

1. Open the Job Example 1 form.



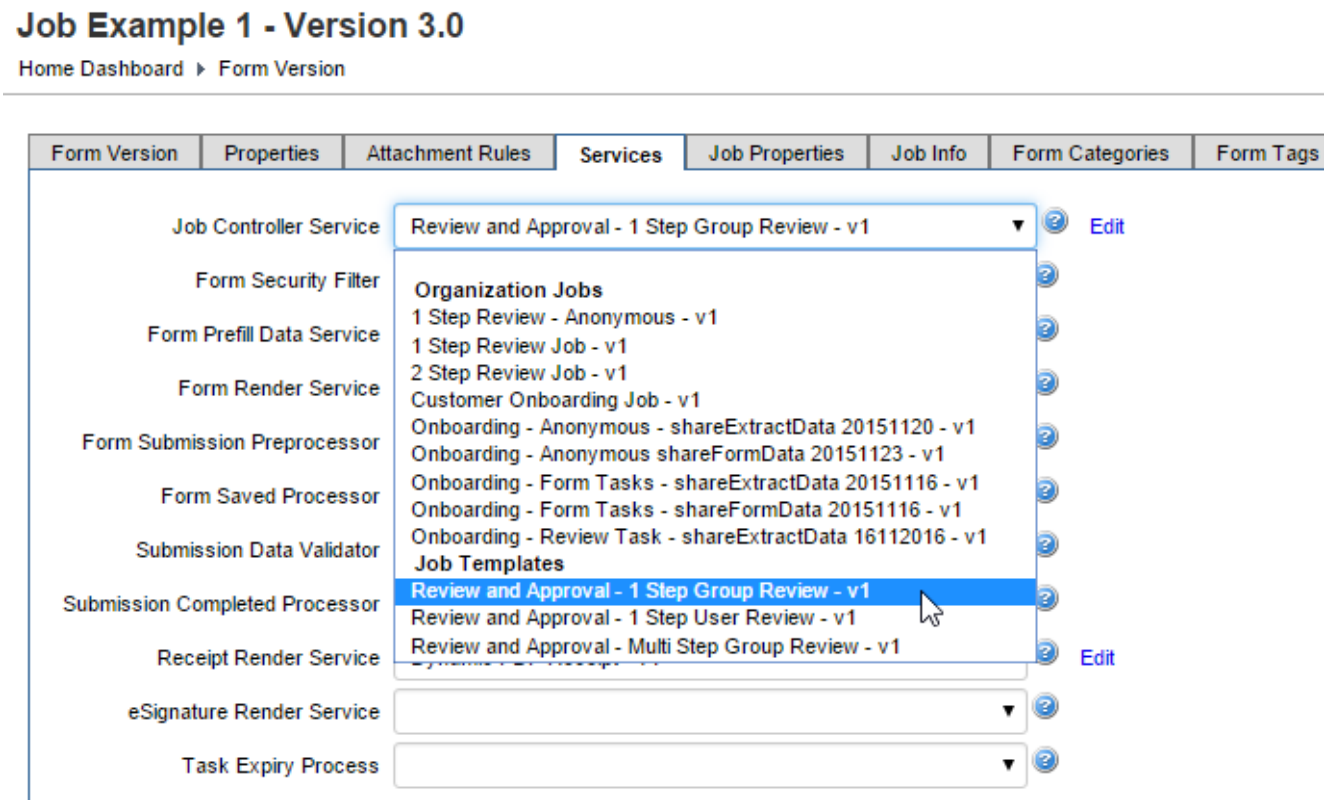
2. Confirm that the delivery channel in the **Form Details Delivery Channel** is set to the **Trash Can Delivery**.



3. Click **Services** of the latest form version to configure its job controller service.

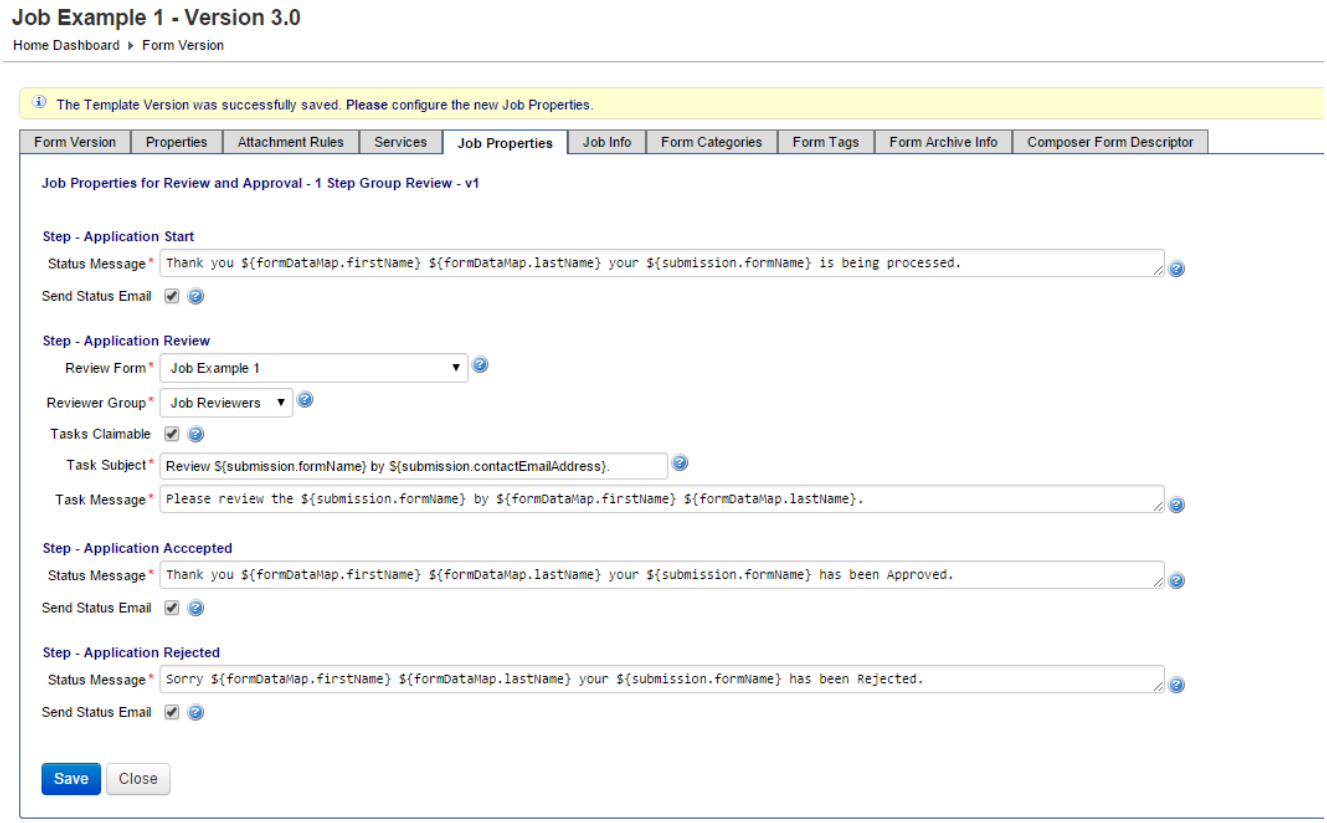


4. Select the **Review and Approval - 1 Step Group Review** service as a job template from the **Job Controller Service** dropdown list and click **Save**.



The **Job Example 1** form also works with the **Review and Approval - 1 Step User Review** job controller.

5. Select the **Job Properties** tab, which is available for job templates, to customize the job controller under the **Step - Application Review** section. Properties can use data extracts and other model elements.



6. Select the **Job Example 1** from the **Review Form** dropdown list.
7. Select the **Job Reviewers** from the **Reviewer Group** dropdown list.
8. Click **Save** to update the changes.

9. Check the **Review and Approval - 1 Step Group Review** job controller's definitions.

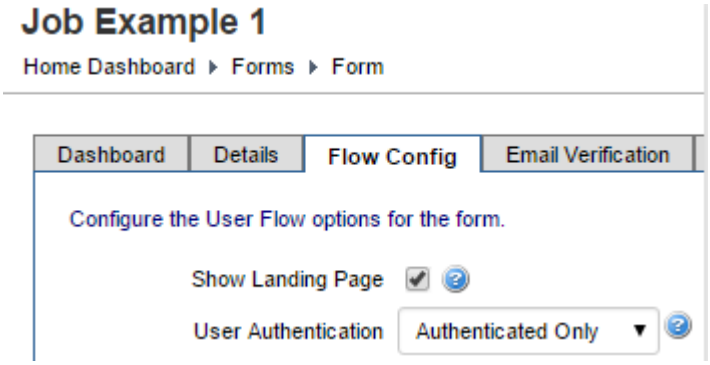
Configure Users and Groups

This example uses the following users, which are the actors in the job definition. The users have access to the Maguire form space and are in the Maguire organization.

Group	Login	Name	Description
Job Applicants	anne	Anne Applicant	Anne is the applicant who submits the application.
Job Reviewers	roger	Roger Reviewer	Roger is the reviewer who reviews the application and belongs to the **Job Reviewers** group.

However, you can substitute existing users for Anne and Roger. The form can be submitted as an public anonymous user or an authenticated user, but the reviewer has to be an authenticate user.

The **Job Example 1** form pre-populates data for an authentication user only. You can customize your form flow by selecting the **Authenticated Only** from the **User Authentication** dropdown list, as shown below.

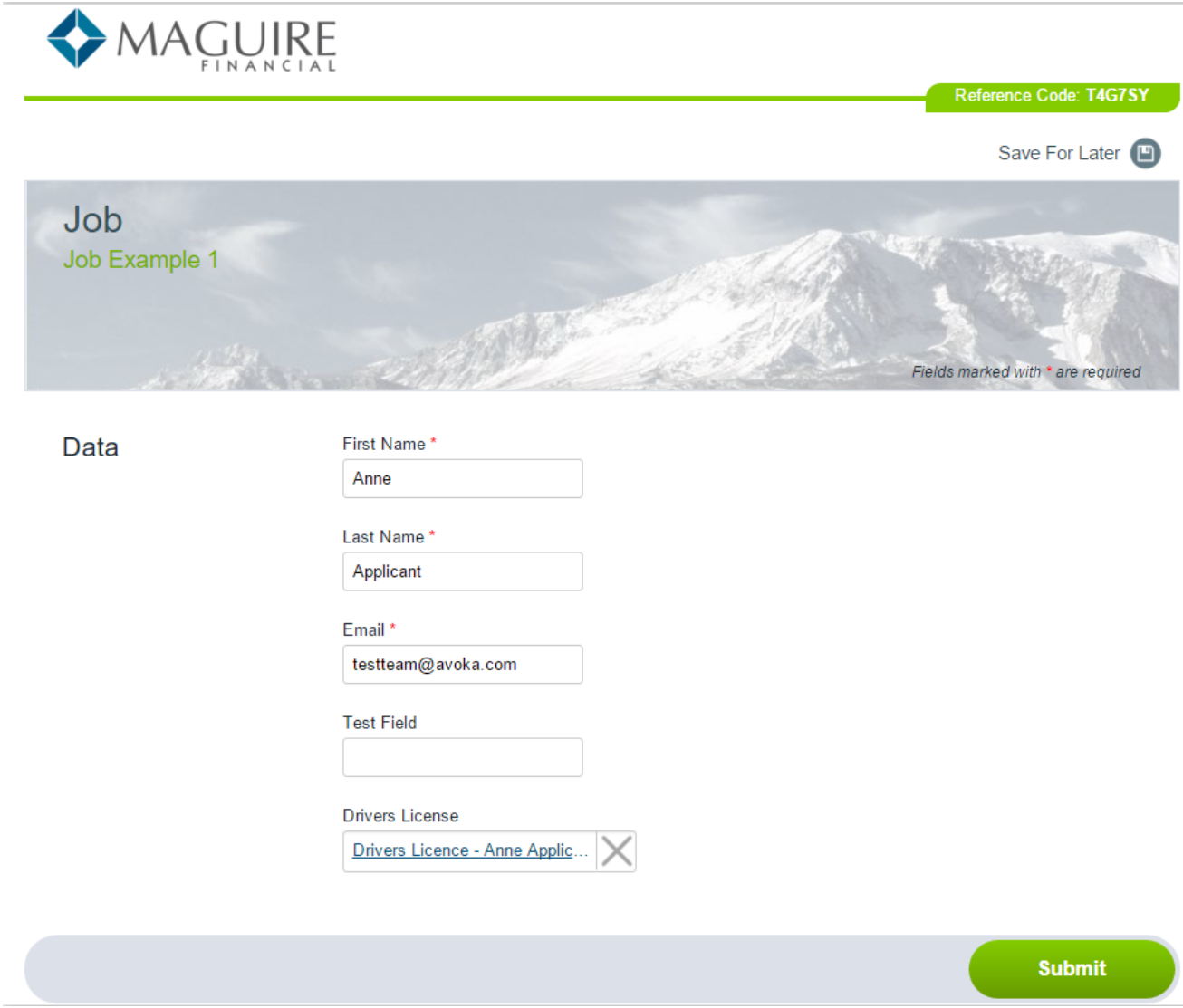


You should also configure the Job Reviewers group by adding Roger to the **Group Members** in the **Members** tab.

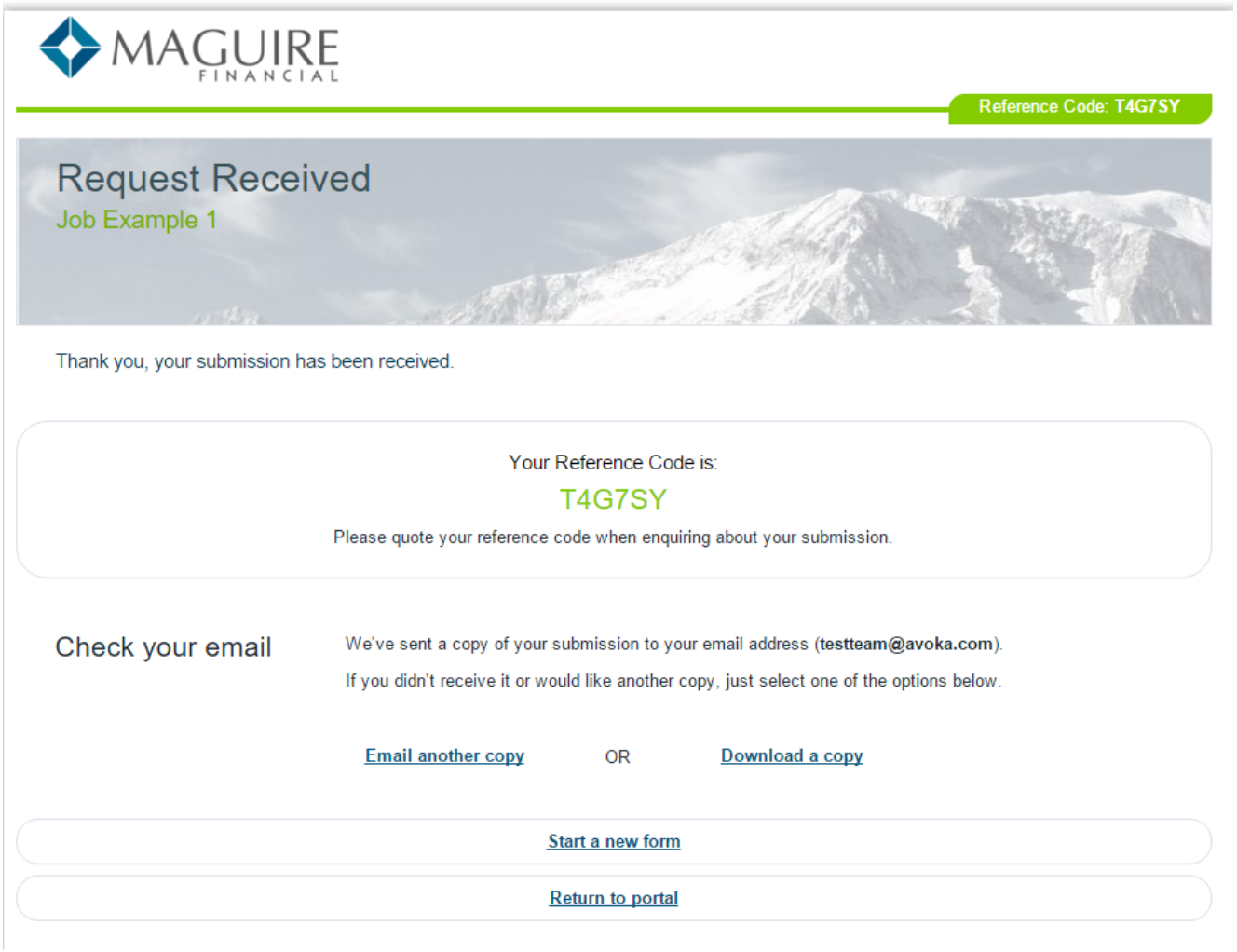
Now, you can run the example and see how collaboration job works.

Form - Anne

1. Log on to Manager as Anne.
2. Open the Maguire form space and select the **Forms** tab.
3. Locate the **Job Example 1** form and click **Direct** to display the form in a form space in a new browser tab. If the form requires an authenticated user, It will first redirect to the portal login page. After logging in with the correct credentials, it will redirect you to the form page itself.



4. Fill in the **First Name**, **Last Name** and **Email** fields.
5. Upload a copy of a driver license as an image file using the **Drivers License** field browser and click **Submit**.



6. Take a note of the **Reference Code** of the form submission, T4G7SY in this case, so you can later use as the **Job Number**.
7. Click **Return to Portal** to return to your form space.

Manager - Administrator

This submission creates a collaboration job, which you can check in Manager:

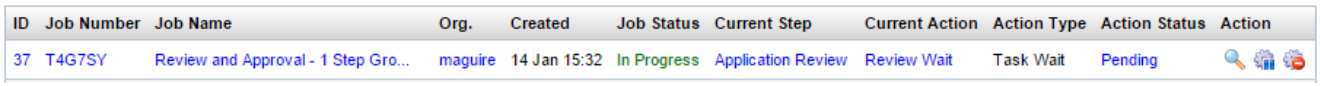
1. Log on to Manager as an administrator.
2. Locate the created collaboration job.

ID	Job Number	Job Name	Org.	Created	Job Status	Current Step	Current Action	Action Type	Action Status	Action
37	T4G7SY	Review and Approval - 1 Step Gro...	maguire	14 Jan 15:32	In Progress	Application Start	Accept Quote	Form Start	Ready	

3. Check the collaboration job state, which should be:
   - Current Step - Application Start
   - Current Action - Accept Quote
   - Action Status - Ready.

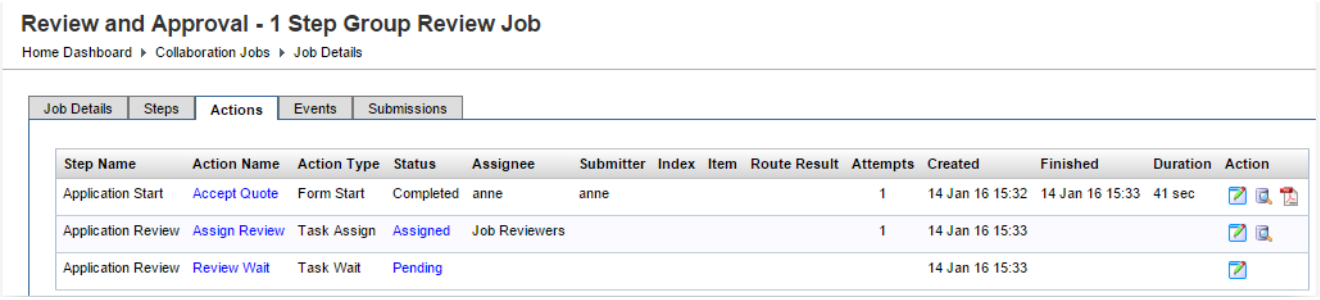
This means the collaboration job is available and is waiting to be processed. Form more information, see collaboration job modes.

4. Click **Process Job** to execute this collaboration job, so you can test it right away. Manager processes the collaboration job and moves it to the Application Review step, as shown below.



ID	Job Number	Job Name	Org.	Created	Job Status	Current Step	Current Action	Action Type	Action Status	Action
37	T4G7SY	Review and Approval - 1 Step Gro...	maguire	14 Jan 15:32	In Progress	Application Review	Review Wait	Task Wait	Pending	

5. Click **View** and select the **Actions** tab for this collaboration job instance to see each action and its status.



Review and Approval - 1 Step Group Review Job

Home Dashboard ▸ Collaboration Jobs ▸ Job Details

Job Details | Steps | Actions | Events | Submissions

Step Name	Action Name	Action Type	Status	Assignee	Submitter	Index	Item	Route Result	Attempts	Created	Finished	Duration	Action
Application Start	Accept Quote	Form Start	Completed	anne	anne				1	14 Jan 16 15:32	14 Jan 16 15:33	41 sec	
Application Review	Assign Review	Task Assign	Assigned	Job Reviewers					1	14 Jan 16 15:33			
Application Review	Review Wait	Task Wait	Pending							14 Jan 16 15:33			

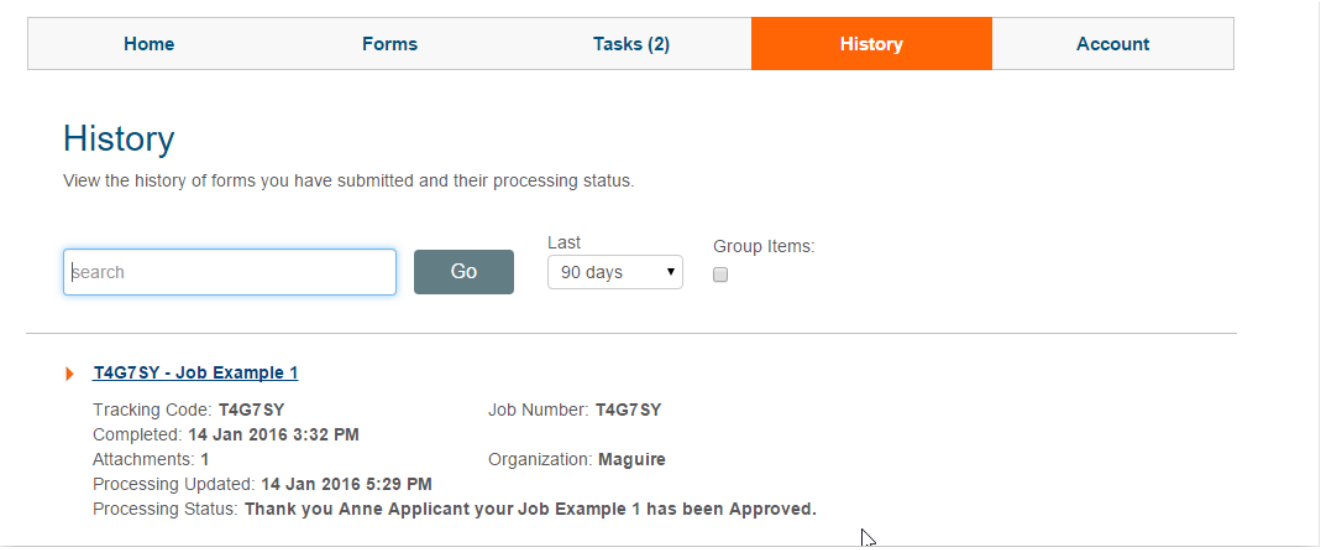
The Application Review - Assign Review entry is assigned to Job Reviewers.

6. Click **Close**.

Form Space - Anne

You can see the collaboration job in your Work Space:

1. Log on to Work Space as Anne.
2. Click **History** tab to see the **Processing Status** of the collaboration job.



Home | Forms | Tasks (2) | History | Account

History

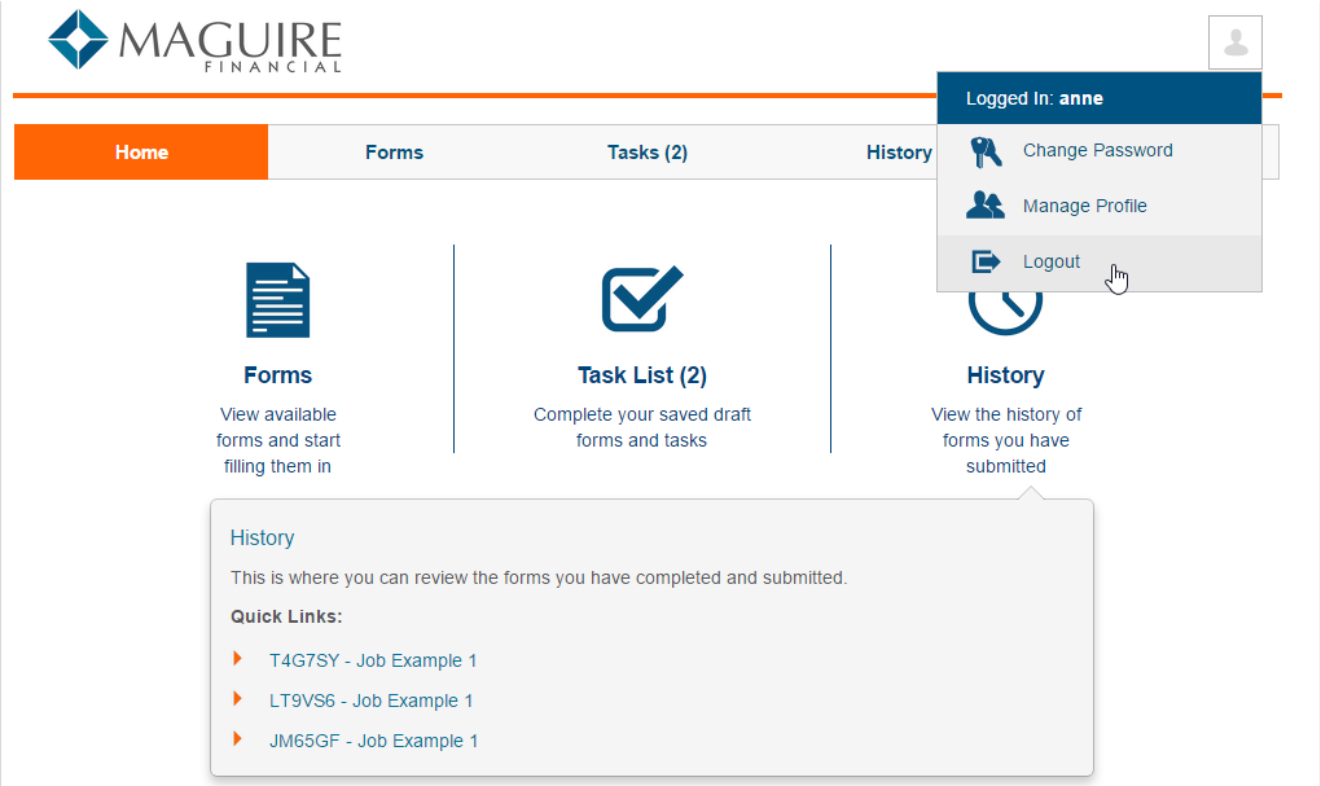
View the history of forms you have submitted and their processing status.

search | Go | Last 90 days | Group Items:

T4G7SY - Job Example 1

Tracking Code: **T4G7SY** Job Number: **T4G7SY**
Completed: **14 Jan 2016 3:32 PM**
Attachments: **1** Organization: **Maguire**
Processing Updated: **14 Jan 2016 5:29 PM**
Processing Status: **Thank you Anne Applicant your Job Example 1 has been Approved.**

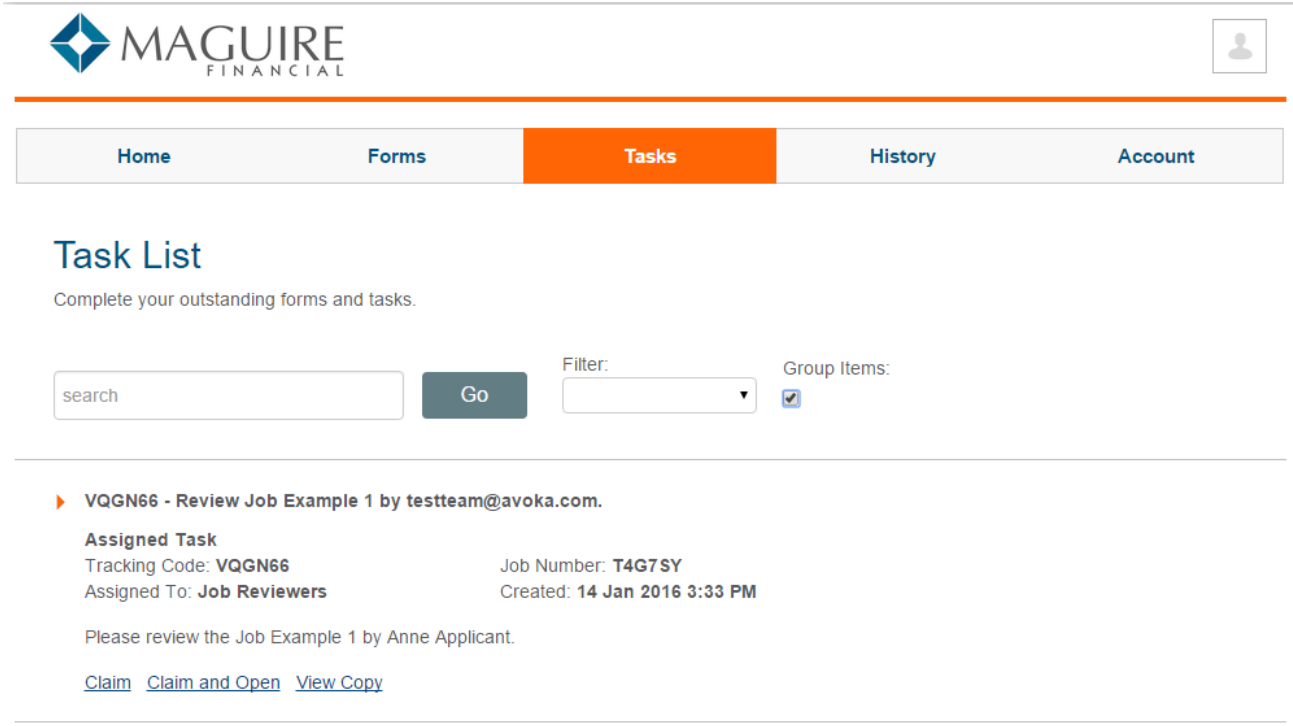
3. Log out as Anne



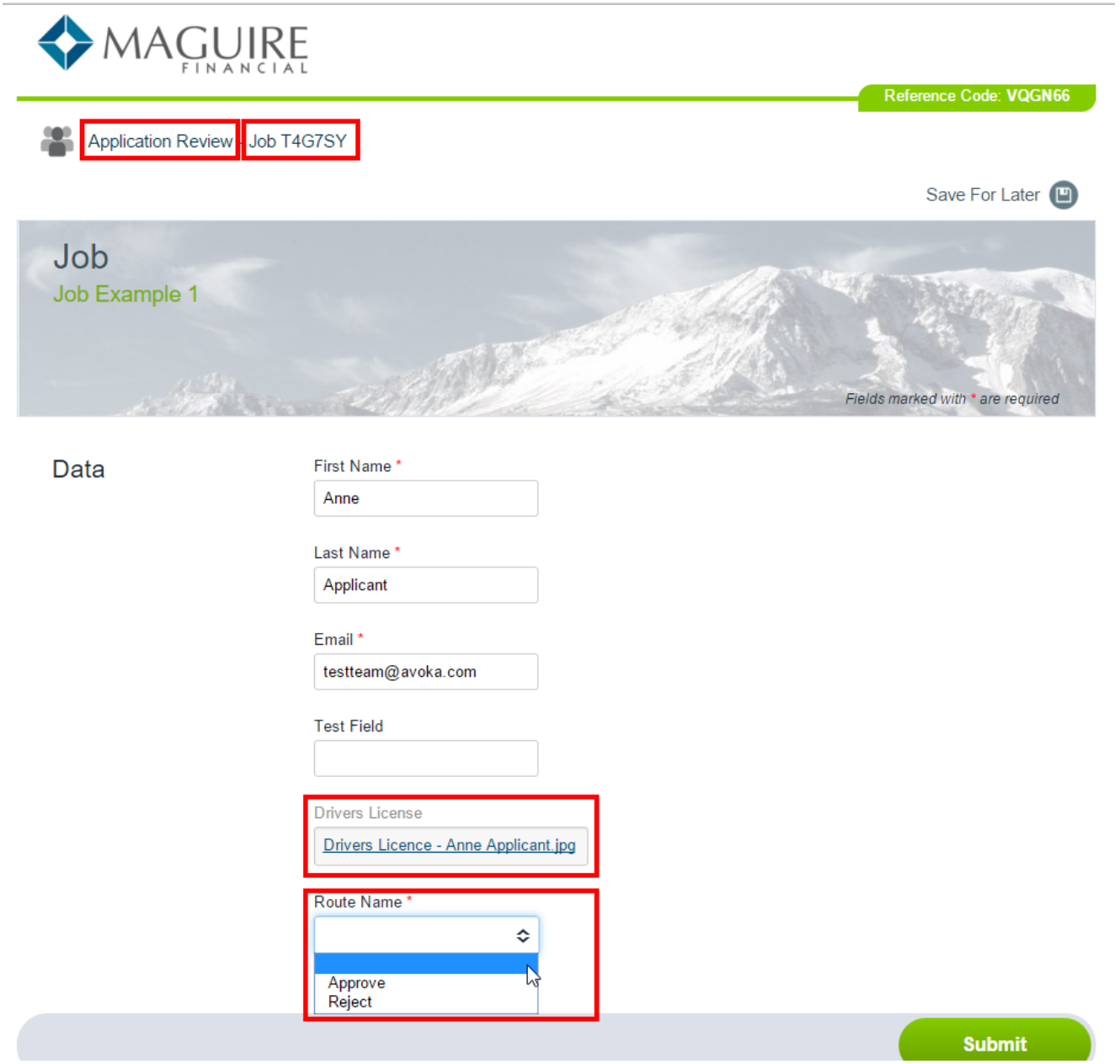
Form Space - Roger

You can review the collaboration job and decide whether to approve or reject it.

1. Log on to the Maguire form space as Roger.
2. Click the **Tasks** tab and select the **Group Items** checkbox to see the **Group Task** details as shown below.



3. Click **Claim and Open** to open the form and check the **Step Name** and **Job Number**. The **Drivers License** field, which is the inform attachment, is read only. The **Route Name** dropdown is now populated with the following choices: **Approve** or **Reject**.



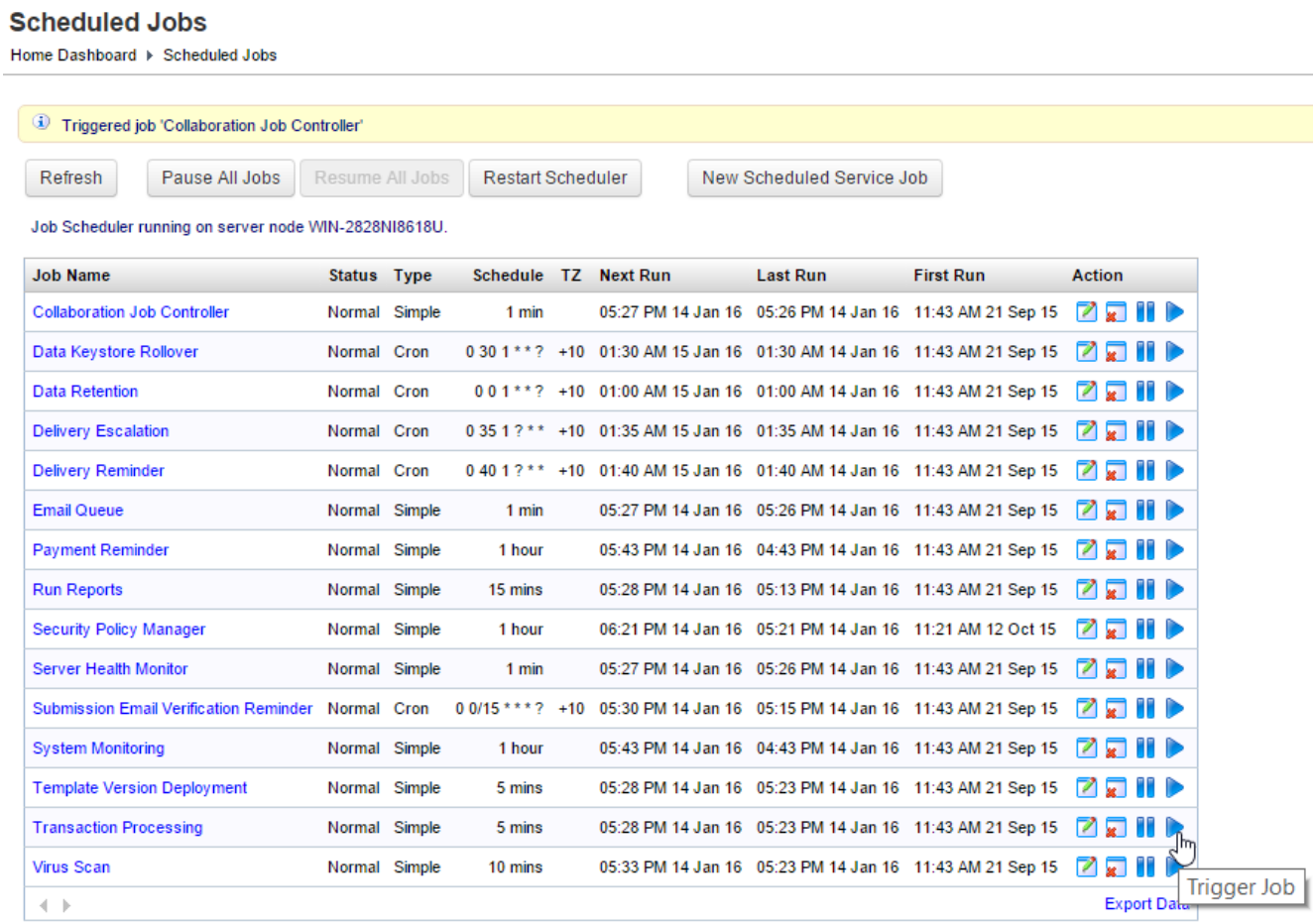
4. Select **Approve** from the **Route Name** dropdown list and click **Submit**.
5. Click **Return to Portal** on the confirmation page and click **Logout**.

Manager - Administrator

1. Locate the collaboration job and click **Press Process Job**, if its still at the **Application Review** step.
2. Check that the collaboration job moves to the **Application Delivery** step where it waits for the delivery to be completed.

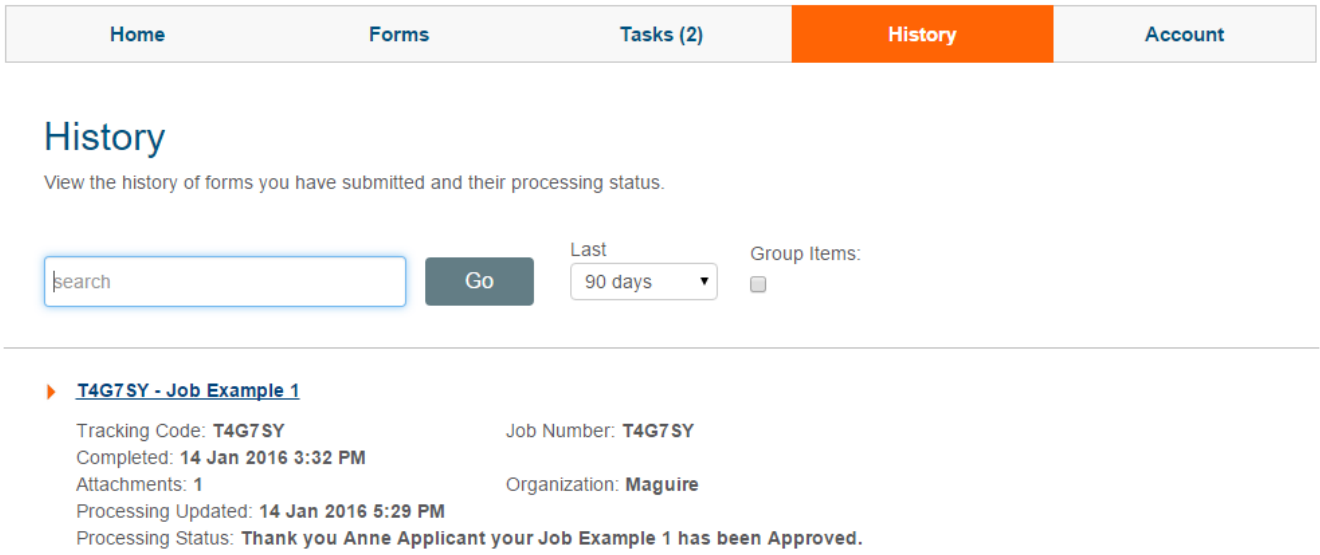
ID	Job Number	Job Name	Org.	Created	Job Status	Current Step	Current Action	Action Type	Action Status	Action
37	T4G7SY	Review and Approval - 1 Step Gro...	maguire	14 Jan 15:32	In Progress	Application Delivery	Application Delivery	Delivery	In Progress	

3. A scheduled job performs the delivery, which runs every 5 minutes by default. However, you can click **Trigger Job** to execute it right away, as shown below.



Form Space - Anne

1. Log on to the Maguire form space as Anne.
2. Select the **History** tab to see that the application has now been approved now.



3. Close the form space.

This concludes the example.

You can download the complete code example here.

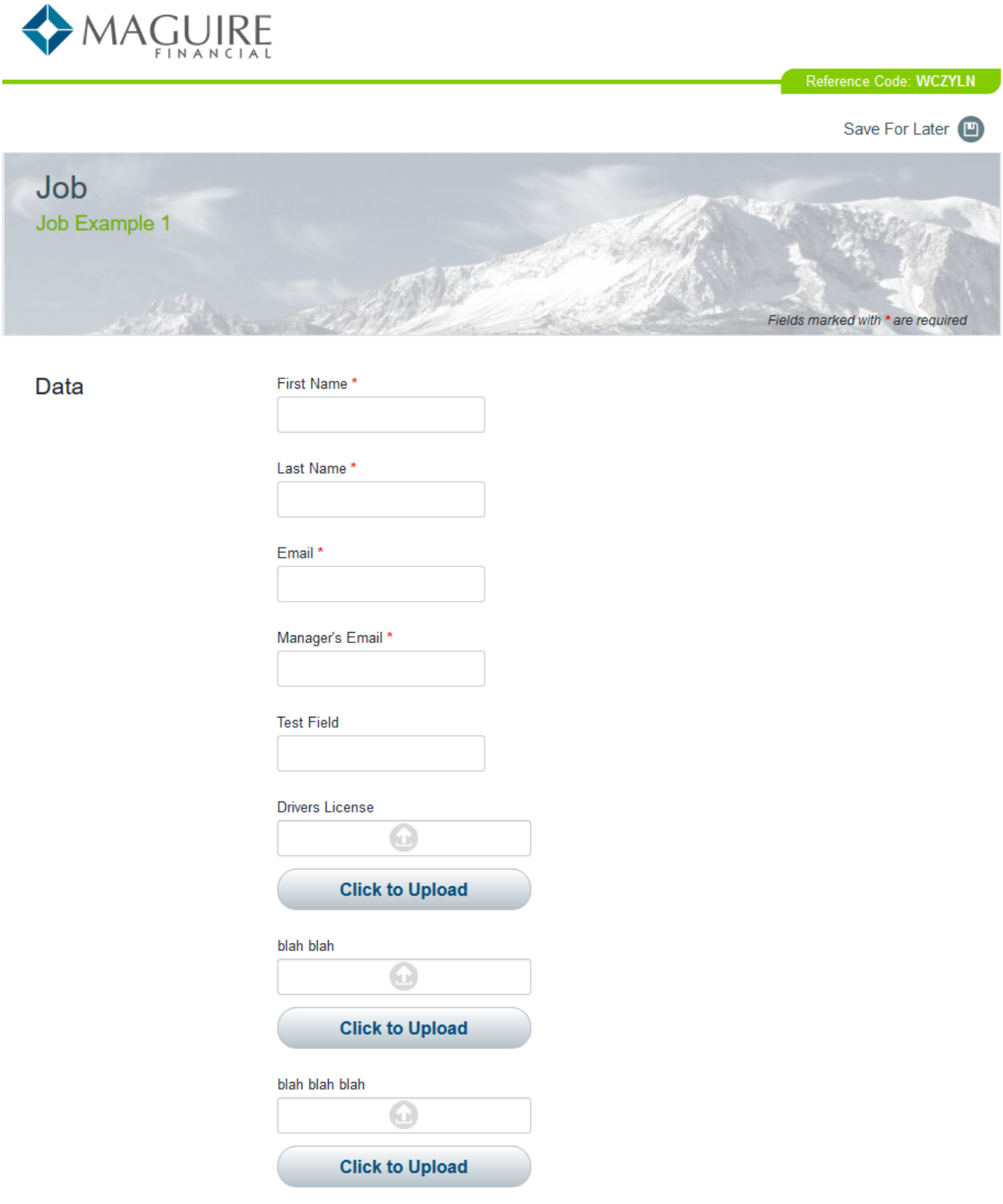
Next, learn about multi-step group review collaboration jobs.

One Step Anonymous Review Collaboration Jobs

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A one step anonymous review collaboration job is similar to a one step review collaboration job but with anonymous tasks.

The application form is shown below.



The form has the extra field, Managers Email, which is who the **Job Step Application Review** is assigned to. The first inform attachment field drivers license is read only at the review step. The other two inform attachment fields have no rules applied.

The job definition of the **1 Step Review - Anonymous** is shown below.

```
{
  "jobDetails": {
    "name": "Anonymous 1 Step",
    "processSubmitImmediate": "false",
    "version": "SNAPSHOT"
  },
  "steps": [
    {
      "name": "Application Start",
      "type": "start",
      "actions": [
        {
          "name": "Accept Quote",
          "type": "Job Form Start",
          "properties": [
            { "name": "Process Message Send Email",  "value": "true" },
            { "name": "Process Message Text",  "value": "Thank you
${formDataMap.firstName} ${formDataMap.lastName} your ${submission.formName} is being pro-
cessed." }
          ]
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Application Review" }
      ]
    },
    {
      "name": "Application Review",
      "type": "",
      "actions": [
        {
          "name": "Assign Review",
          "type": "Job Task Assign",
          "properties": [
            { "name": "Task Assign Email",  "value": "${formDataMap.managersEmail}" },
            { "name": "Task Form Code",  "value": "$func.startFormCode()" },
            { "name": "Task Message",  "value": "Please review the ${submission.formName}
by ${formDataMap.firstName} ${formDataMap.lastName}." },
            { "name": "Task Subject",  "value": "Review ${submission.formName} by ${sub-
mission.contactEmailAddress}." },
            { "name": "Task Assign Portal",  "value": "Maguire" },
            { "name": "Task Form XML Data",  "value": "$func.startSubmissionXml()" },
            { "name": "Task Attachments Submission Step",  "value": "Application Start"
},
```

```
            { "name": "Task Send Email",  "value": "true" },
            { "name": "Task Email Message",  "value": "$func.formProperty('Anonymous Task
Email Message')" },
            { "name": "Task Email Subject",  "value": "$func.formProperty('Anonymous Task
Email Subject')" },
            { "name": "Task Type",  "value": "Anonymous" }
          ]
        },
        {
          "name": "Review Wait",
          "type": "Job Task Wait"
        }
      ],
      "routes": [
        { "name": "Approve",  "nextStep": "Application Delivery" },
        { "name": "Reject",  "nextStep": "Application Rejected" }
      ]
    },
    {
      "name": "Application Delivery",
      "type": "",
      "actions": [
        {
          "name": "Application Delivery",
          "type": "Job Delivery"
        },
        {
          "name": "Application Delivery Wait",
          "type": "Job Delivery Wait"
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Application Completed" }
      ]
    },
    {
      "name": "Application Completed",
      "type": "endpoint",
      "actions": [
        {
          "name": "Process Message",
          "type": "Job Process Message",
          "properties": [
            { "name": "Process Message Send Email",  "value": "true" },
```

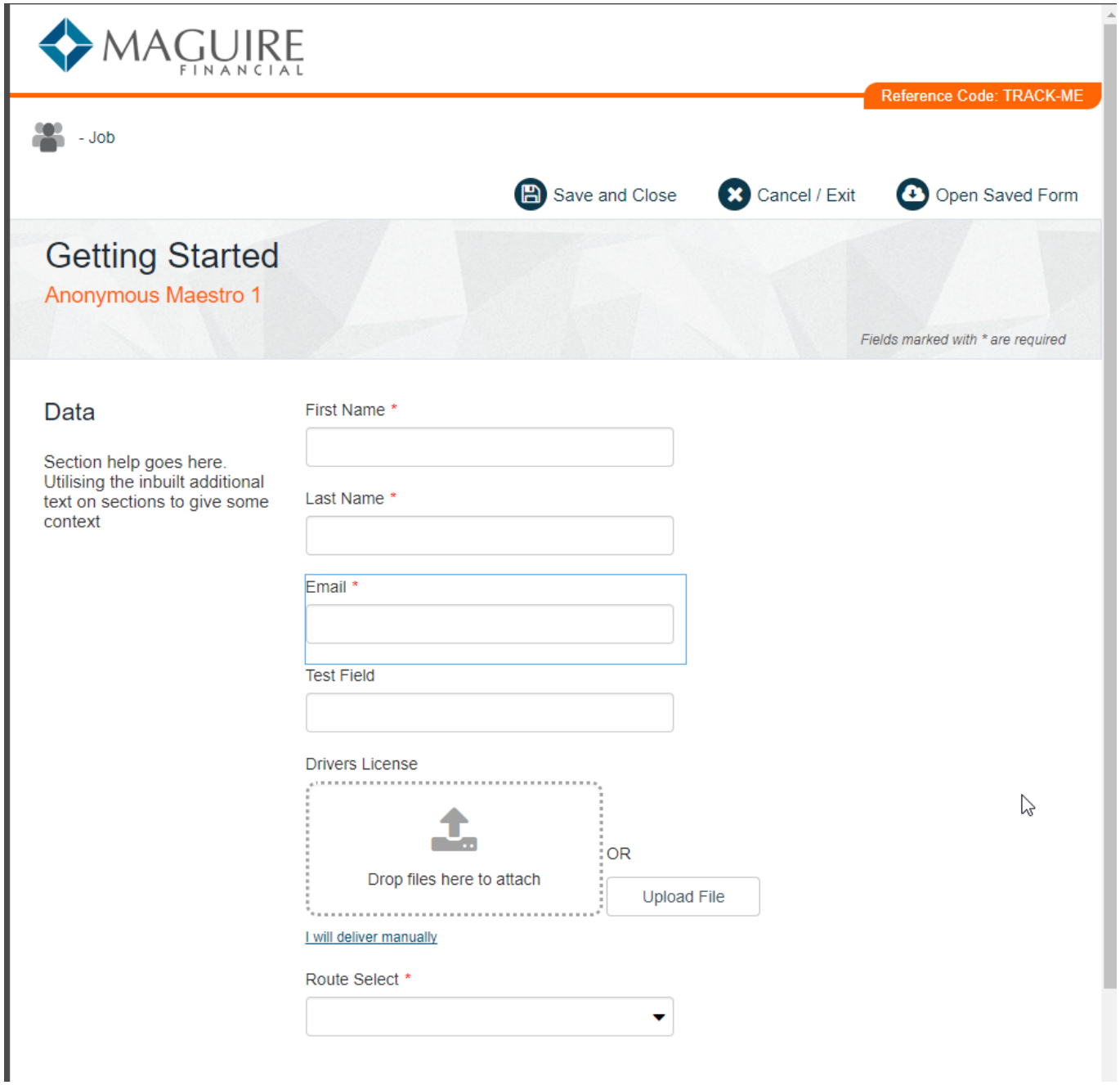
```
            { "name": "Process Message Submission Step",  "value": "Application Start" },
            { "name": "Process Message Text",  "value": "Thank you
${formDataMap.firstName} ${formDataMap.lastName} your ${submission.formName} has been
Approved." }
          ]
        }
      ]
    },
    {
      "name": "Application Rejected",
      "type": "endpoint",
      "actions": [
        {
          "name": "Process Message",
          "type": "Job Process Message",
          "properties": [
            { "name": "Process Message Send Email",  "value": "true" },
            { "name": "Process Message Submission Step",  "value": "Application Start" },
            { "name": "Process Message Text",  "value": "Sorry ${formDataMap.firstName}
${formDataMap.lastName} your ${submission.formName} has been declined." }
          ]
        }
      ]
    }
  ]
```

You can download the complete code example here.

Cancel an Anonymous Task in Maestro Forms

You can cancel an authenticated task in the same way as per the one step review described above.

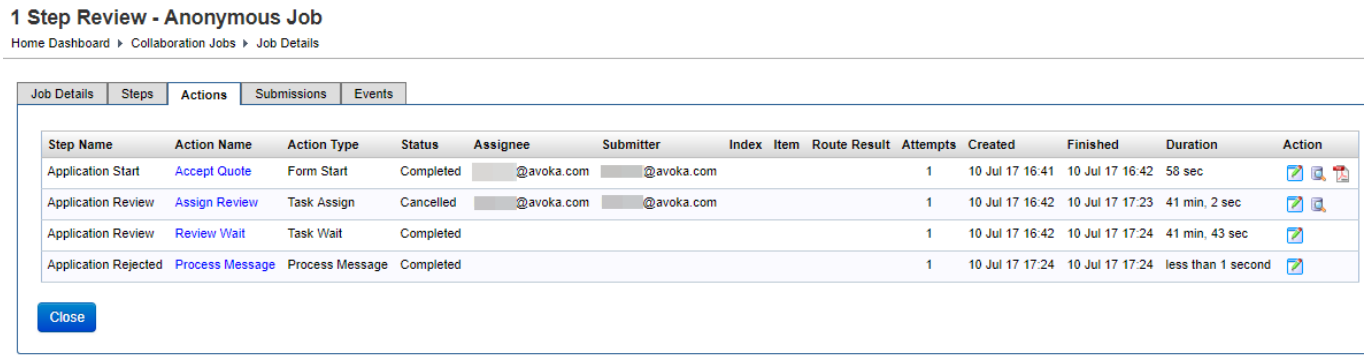
.



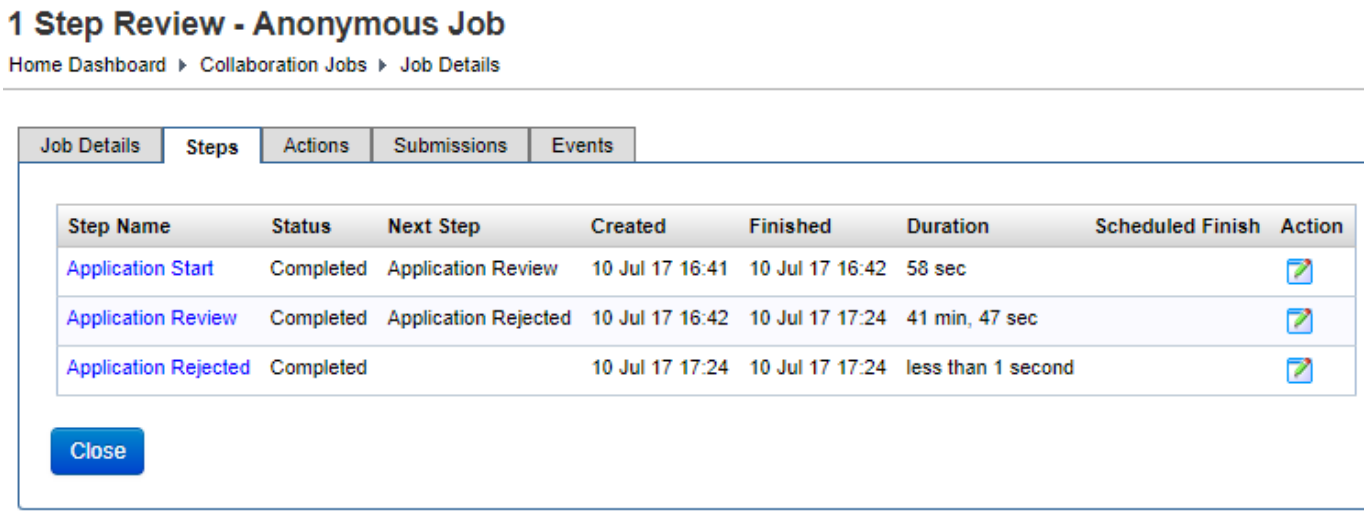
When we initially submit the form we enter our details. The task that is created will get assigned back to the address in the email field. You will get notified of a saved email with a link to open the form.

You have opened the form associated with the Application Review - Assign Review action. Then click on the Cancel Exit button. Cancelling doesn't return any form.xml and hence the route is empty. The empty route is picked up by the new Default route which send to the next step Application rejected.

Here we can see the actions that have been executed. We can see the status of the Application Review has been changed to cancelled and the route result is blank.



Here we can see that when the default route is routing to the next step Application Rejected.



The cancel doesn't submit any form data resulting in the empty **`routeName`**, so you needed to add the Default route to the application Review step of a job definition of the collaboration job as shown below.

```
{
  "name": "Application Review",
  "type": "",
  "actions": [
    {
      "name": "Assign Review",
      "type": "Job Task Assign",
      "properties": [
        { "name": "Task Assign Email",  "value": "${formDataMap.email}" },
        { "name": "Task Form Code",  "value": "$func.startFormCode()" },
        { "name": "Task Message",  "value": "Please review the ${submission.formName} by
${formDataMap.firstName} ${formDataMap.lastName}." },
        { "name": "Task Subject",  "value": "Review ${submission.formName} by ${sub-
mission.contactEmailAddress}." },
        { "name": "Task Assign Portal",  "value": "Web Plug-in" },
        { "name": "Task Form XML Data",  "value": "$func.startSubmissionXml()" },
        { "name": "Task Attachments Submission Step",  "value": "Application Start" },
        { "name": "Task Send Email",  "value": "true" },
        { "name": "Task Email Message",  "value": "$func.formProperty('Anonymous Task
Email Message')" },
        { "name": "Task Email Subject",  "value": "$func.formProperty('Anonymous Task
Email Subject')" },
        { "name": "Task Type",  "value": "Anonymous" }
      ]
    },
    {
      "name": "Review Wait",
      "type": "Job Task Wait"
    }
  ],
  "routes": [
    { "name": "Approve",  "nextStep": "Application Delivery" },
    { "name": "Reject",  "nextStep": "Application Rejected" },
    { "name": "Default",  "nextStep": "Application Rejected" }
  ]
},
```

This concludes the example.

You can download the complete code example here.

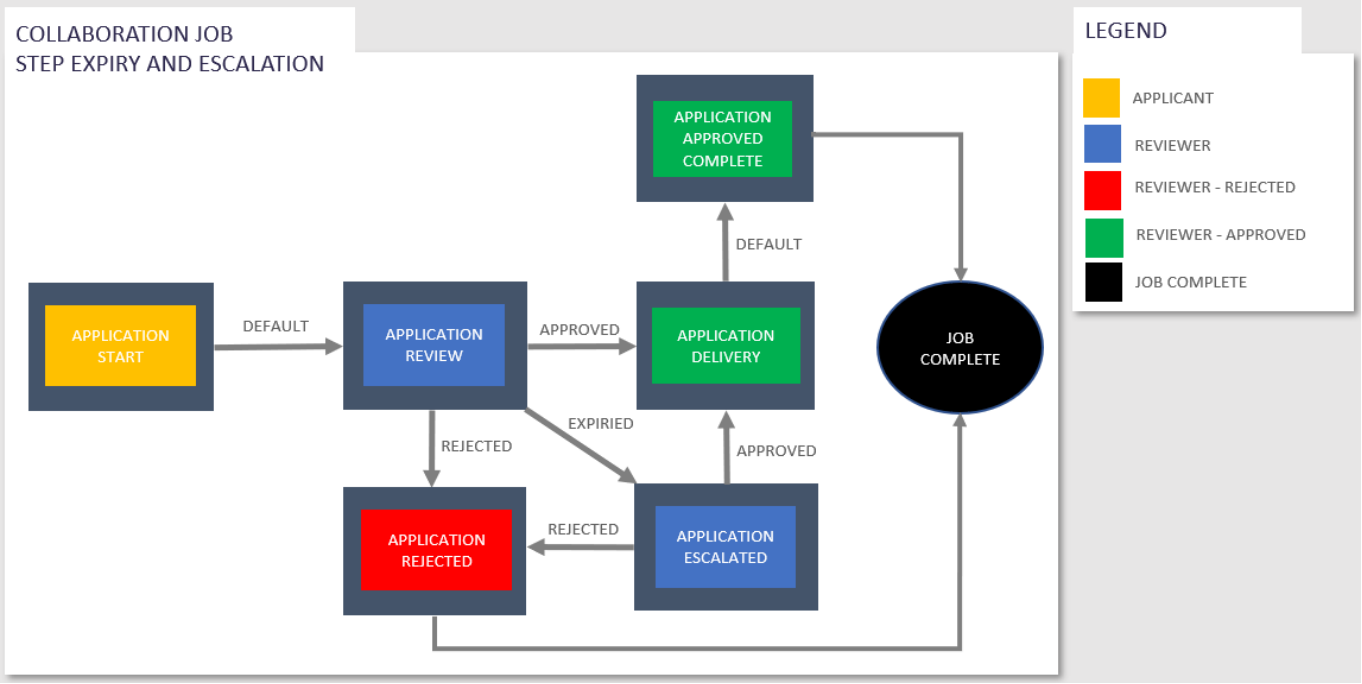
Next, learn about asynchronous step execution.

Step Expiry

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

You can configure steps of a collaboration job to expire depending on customer requirements, such as a **Service Level Agreement**[1]. For example, you can implement a Review step to assign a task to a group, such as Reviewers group, so they have two days to action the task. Otherwise, it expires and will be routed to an Escalation step for a manager to review it.

The step expiry workflow is shown below.



You can implement step expiration using one of the following mechanisms:

1. The step's **`expiryRule`** attribute to specify the period of time, from its creation date, when the step will expire.
2. Any action in a step can control the step expiry by returning an action result that sets an action expiry or returns the **Expired** status.

You can also configure the Expiry Action, see Job Expiry Service, within a step. This allows for the clean-up code to be run. Steps that expire must define an Expiry Route; that is, a route with the **Route Name Expiry**. When the step expires, it is routed to the **Next Step** defined in the **Expiry Route**.

You can also download the application package with this example here.

---

[1] A service-level agreement (SLA) is a commitment between a service provider and a client. Particular aspects of the service – quality, availability, responsibilities – are agreed between the service provider and the service user.

Configure a Form

This example uses the **AVT-3547 Test Expiry** form, which is a copy of Job Example 1 form, initiates a job controller. You can download the example below.

Configure a Job Controller

You can update an existing job controller by adding an **Application Escalated** step, so when the **Application Review** step expires, it will be routed to the **Application Escalated** step.

The **AVT-3547 Test Expiry B** job controller is a modified copy of the **1 Step Review** job controller, where the version B introduces a new Group Task Step.

The Application Review step is modified to:

- Expire the step after 1 minute. | v4.3 & Higher
- Call the job expiry service.
- Call the expiryServiceName service.
- Expiry routes to the net step Application Escalated.
- Expiry route added with display set to false.

The Application Review and Application Escalated steps definition is shown below.

```
{
  "name": "Application Review",
  "type": "",
  "expiryRule": "+1m",
  "expiryServiceName": "Job Expiry - Expire Step Tasks",
  "actions": [
    {
      "name": "Assign Review",
      "type": "Job Task Assign",
      "properties": [
        { "name": "Task Assign Groups",  "value": "Job Reviewers" },
        { "name": "Task Form Code",  "value": "$func.startFormCode()" },
        { "name": "Task Message",  "value": "Please review the ${submission.formName} by
${formDataMap.firstName} ${formDataMap.lastName}." },
        { "name": "Task Review Previous Step",  "value": "true" },
        { "name": "Task Subject",  "value": "Review ${submission.formName} by by
${formDataMap.firstName} ${formDataMap.lastName}" },
        { "name": "Task Type",  "value": "Review" }
      ]
    },
    {
      "name": "Review Wait",
      "type": "Job Task Wait"
    }
  ],
  "routes": [
    { "name": "Approve",  "nextStep": "Application Delivery" },
    { "name": "Reject",  "nextStep": "Application Rejected" },
    { "name": "Expiry",  "nextStep": "Application Escalated",  "display": "false" }
  ]
},
{
  "name": "Application Escalated",
  "type": "",
  "actions": [
    {
      "name": "Assign Review",
      "type": "Job Task Assign",
      "properties": [
        { "name": "Task Assign Groups",  "value": "Job Managers" },
        { "name": "Task Form Code",  "value": "$func.startFormCode()" },
        { "name": "Task Message",  "value": "The ${submission.formName} by
${formDataMap.firstName} ${formDataMap.lastName} has been escalated." },
        { "name": "Task Review Previous Step",  "value": "true" },
```

```
        { "name": "Task Subject",  "value": "Review Escalated ${submission.formName} by
${formDataMap.firstName} ${formDataMap.lastName}." },
        { "name": "Task Type",  "value": "Review" }
      ]
    },
    {
      "name": "Review Wait",
      "type": "Job Task Wait"
    }
  ]
```

Configure a Job Expiry Service

This service cleans up an task that hasn't been submitted and returns the Expiry route name. If you are expiring a step that has tasks that have not been completed, then you will need to run a Job Expiry Action that cleans these up, as shown in the code below.

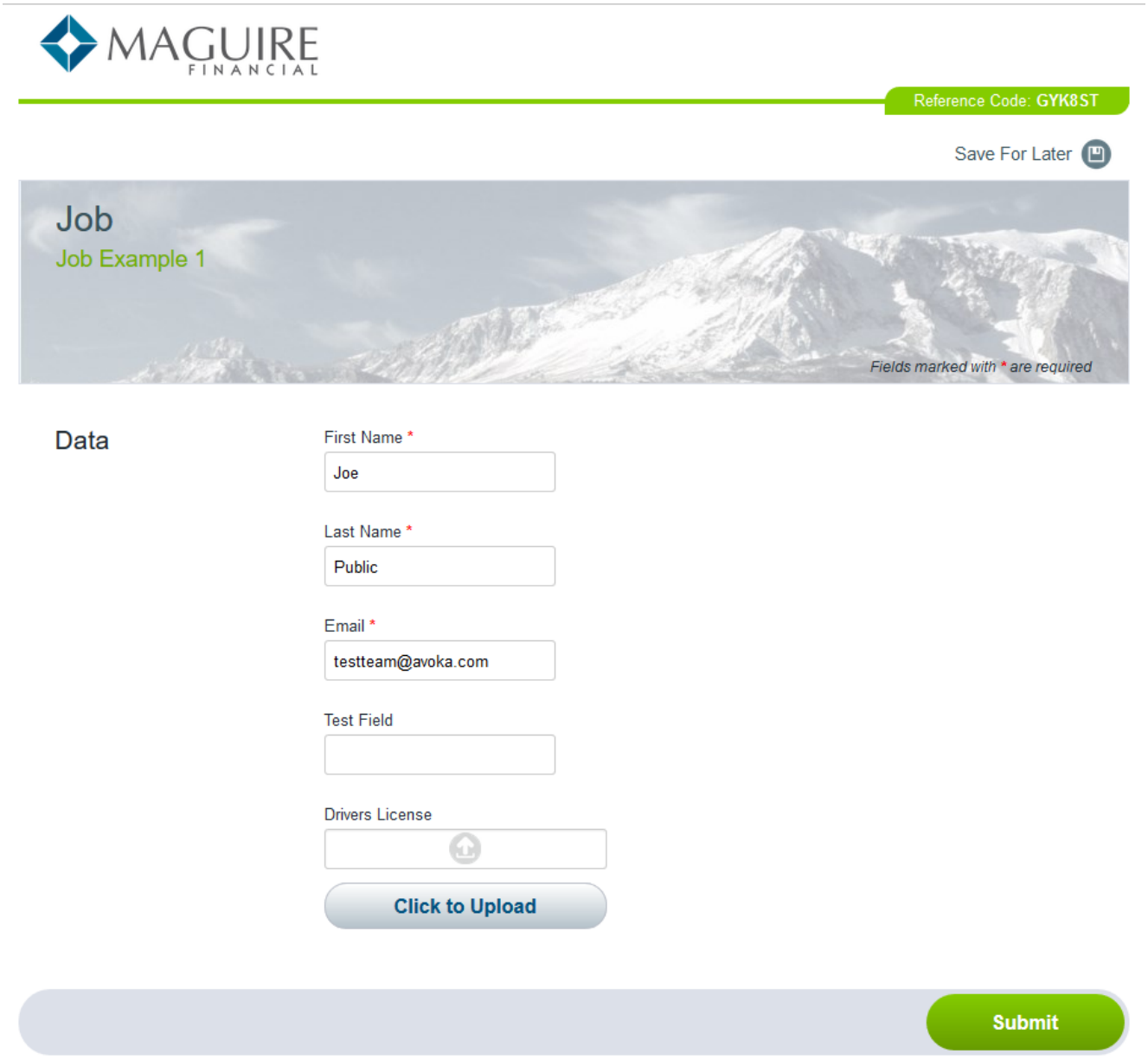
```
/* Groovy Job Expiry action service is a special type of job action service that is
executed when a step expires.
   It is used to clean up items such as tasks created by the step runs its actions.
   Script parameters include:
       actionContext : com.avoka.fc.core.service.job.ActionContext
       actionStepProperties :  com.avoka.fc.core.service.job.ActionStepProperties
       serviceDefinition : com.avoka.fc.core.entity.ServiceDefinition
       serviceParameters : Map<String, String>
       job : com.avoka.fc.core.entity.Job
       submission : com.avoka.fc.core.entity.Submission
       formDataMap : Map<String, String>
   Script return:
       actionResult : com.avoka.fc.core.service.job.ActionResult
*/

import com.avoka.fc.core.dao.DaoFactory
import com.avoka.fc.core.entity.Job
import com.avoka.fc.core.entity.JobAction
import com.avoka.fc.core.entity.JobStep
import com.avoka.fc.core.entity.Submission
import com.avoka.fc.core.service.AbandonmentService
import com.avoka.fc.core.service.job.ActionResult
import com.avoka.fc.core.service.job.ActionResult.Status
import com.avoka.core.groovy.GroovyLogger as logger
final EXPIRY_ROUTE_NAME = "Expiry"
Date now = new Date()

JobAction expiryJobAction = actionContext.getJobAction()
JobStep step = expiryJobAction.jobStep
List<JobAction> jobActions = step.getOrderedJobActions();
jobActions.remove(expiryJobAction)

// the msg can't be longer than 2000 characters.
StringBuilder msg = new StringBuilder();
msg.append("The job step associated with this task has expired. ref Job: ")
        .append(job.referenceNumber)
        .append(" Step: ")
        .append(step.name)

for (JobAction jobAction in jobActions) {
    // Fix Job Actions that have not completed
    if (jobAction.isStatusInProgress()
            || jobAction.isStatusAssigned()
            || jobAction.isStatusInvoked()
```

```
            || jobAction.isStatusPending()
            || jobAction.isStatusReady()) {
        jobAction.setStatus(JobAction.STATUS_EXPIRED);
        jobAction.setTimeFinished(now);
    }

    // Fix Tasks / Submissions in progress
    Submission submission = jobAction.getSubmission()
    if (submission != null) {
        // Set For all Tasks in this step as the step has Expired we don't want to
deliver the
        submission.setDeliveryStatus(Submission.STATUS_NotRequired);
        if (submission.isFormOpened()
                || submission.isFormAssigned()
                || submission.isFormSaved()
                || submission.isFormSubmitted()) {
            submission.setFormStatus(Submission.STATUS_Abandoned)
            submission.setAbandonmentType(AbandonmentService.ABANDONMENT_TYPE_SYSTEM)
            submission.setAbandonmentTimestamp(now)
            submission.setAbandonmentReason(msg.toString())

        } else {
            // For Tasks that were submitted
            submission.setDeliveryTime(now);
            submission.setAbandonmentReason(msg.toString())
        }

        if (submission.isDeliveryNotRequired()) {
            if (submission.getPurgeTimeDataDelivered() == null) {
                DaoFactory.submissionDao.setPurgeDatesForDeliveredAbandonedForms(sub-
mission);
            }
        }
    }
}
ActionResult actionResult = new ActionResult(Status.COMPLETED)
actionResult.setRoute(EXPIRY_ROUTE_NAME)
return actionResult
```

Now, you can run the example and see how collaboration job's step expiry works.

Form - Start

1. Joe click on the AVT-3547 Step Expiry Form which opens.

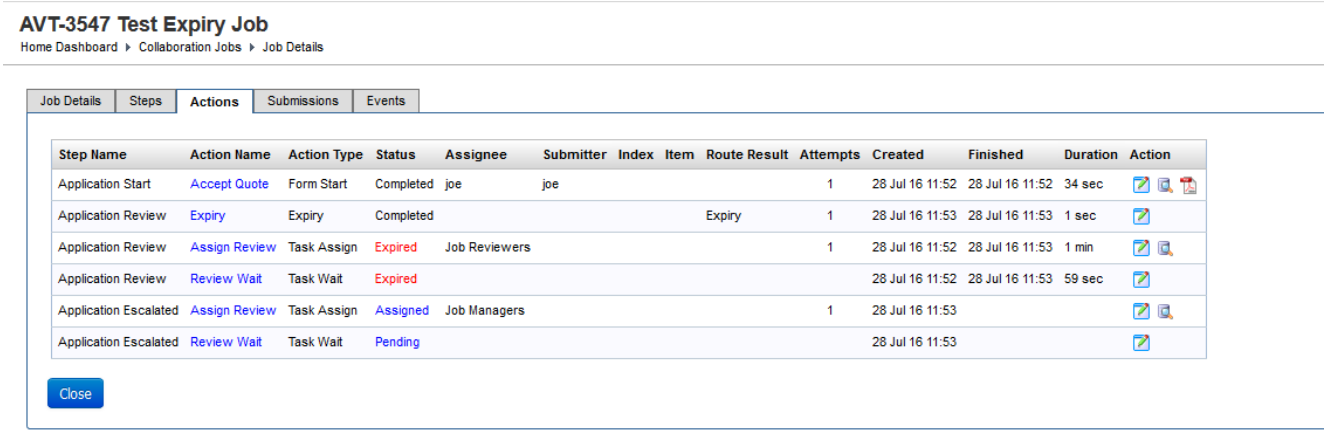


2. Joe's detail are prefilled so he clicks **Submit**.

Manager - Action Review Expiry

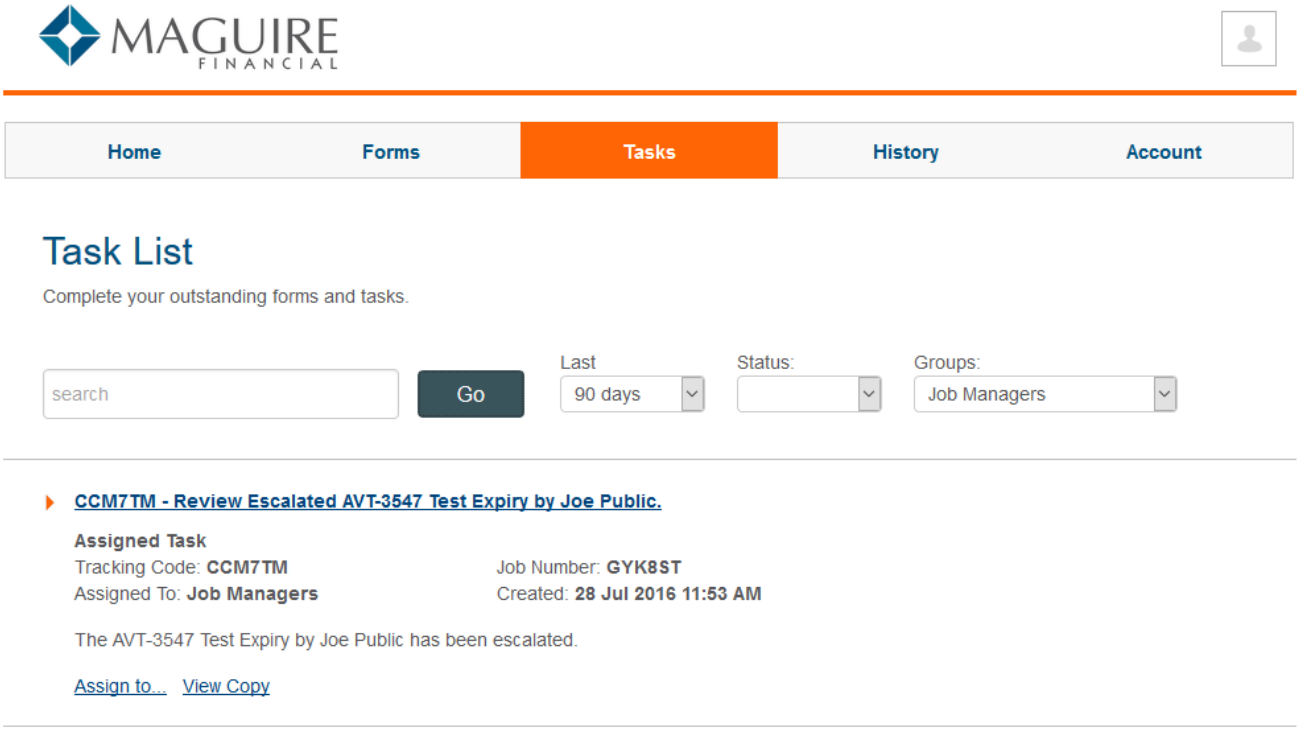
1. The collaboration job creates a **Application Review** step with an **Review Task Assign** action link to a sub-mission assigned to the Job Reviewers and Review. There is also a task Review Task Wait Actions.
2. When the step expires these 2 actions are expired. The Application Review - Expiry action (Custom Groovy

Service) runs to clean up the submissions for this step. For more details see the Job Expiry Service below.

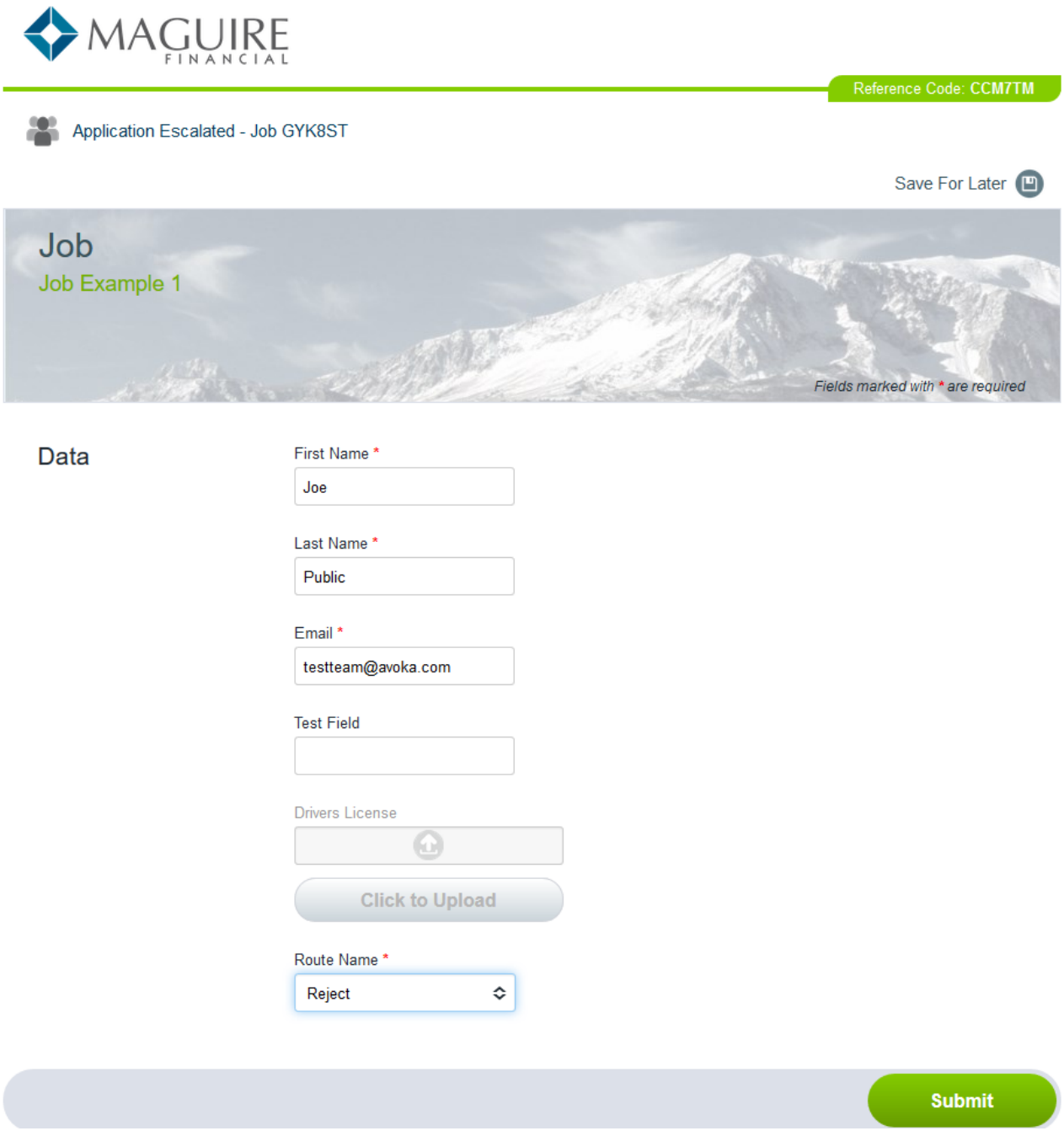


Form Space - Application Escalated

1. Log in as Mike, who is a Job Manager.
2. Select the **Tasks** tab.
3. Select the **Job Managers** from the **Groups** dropdown list and click **Go** to view the following task list.



4. Click the **CCM7TM** task.



5. Select the **Approve** or **Reject** from the **Route Name** dropdown list and click **Submit**.

This concludes the example.

You can also download the application package with this example here.

Next, learn about one step anonymous review collaboration jobs.

Task Assign Repeat Action

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A collaboration job can have a task assign repeat action that allows you to do the following in a single step:

- Create a form bundle, with multiple tasks assigned to the same user, from a single form.
- Assign the Reviews and Approval workflow with multiple tasks to different users so an application can be reviewed in parallel.

The collaboration job must provide context back to the form to differentiate tasks. The context is represented with the following properties (node names):

Assignee

It's used by Parallel tasks, in a form as a read-only parameter. The repeat processing iterates over a pipe delimited list of assignees. This shows the current assignee which can be a Username, Group or Email Address (Anonymous Tasks).

The example Job Definition - Task Assign - Job Action.

```
{
 "name": "Parallel Tasks",
 "type": "",
 "actions": [
 {
  "name": "Create Repeat Tasks",
  "type": "Job Task Assign",
  "properties": [
   { "name": "Task Assign Repeating", "value": "true" },
   { "name": "Task Assign Repeat Item", "value": "$func.invoke('FCT-3097 Get Repeat
Item', ${formDataMap.itemDelimited}, ${assignRepeatIndex})" },
   { "name": "Task Assign User",  "value": "$formDataMap.assignDelimited" },
   { "name": "Task Form Code",  "value": "fct-3097-composer" },
   { "name": "Task Message",  "value": "Please approve or reject" },
   { "name": "Task Subject",  "value": "${assignRepeatItem} Complete Credit Card Applic-
ation." },
   { "name": "Task Input XML Prefill",  "value": "$func.startSubmissionXml()" },
   { "name": "Task Type",  "value": "Form" }
  ]
 },]
}
```

To specify users, you must apply the **Task Assign User** action property with a pipe delimited list of usernames, for example, **`Task Assign User = $formDataMap.assignDelimited`**. This data extract can be evaluated as **`"peter|paul|mary"`**. The first task is created for peter with the Assignee as peter. The second task is created for paul with the assignee as paul, and so on.

To specify groups, you must apply the Task Assign Groups action property with a pipe delimited list of groups.

To specify anonymous users, you must apply the action or Email Address anonymous tasks that an individual item is assigned to.

AssignRepeatIndex

It's used by Parallel tasks. It's used in a form as a read-only parameter. The repeat processing creates a is the index (Starting at 1) of the task that was created. A Forms can use this to look up a repeat section and hide show values on the form. In the example above 3 tasks will be created:

- (Assignee: peter, AssignRepeatIndex: 1)
- (Assignee: paul, AssignRepeatIndex: 2)
- (Assignee: mary, AssignRepeatIndex: 3)

AssignRepeatItem

It's used by Parallel tasks. It's used in a form as a read-only parameter. A repeating section in the form may be broken down by something other than a user or group. For example, a student fills out what sports they want to play: cricket, basketball or rugby

Sport	Coach	Season
Basketball	Al	All Year
Cricket	John	Summer
Rugby	John	Winter

A student Ginger Meggs wants to do Cricket, Basketball and Rugby, the issue is that the coach John is the same for Cricket and Rugby. How do we present the unique item value? It can be done in repeat form XML, as shown below.

```
<Sports>
  <Sport><Name>Cricket</Name><Selected>True<Selected><Coach>john</Coach></Sport>
  <Sport><Name>Basketball</Name><Selected>True<Selected><Coach>al</Coach></Sport>
  <Sport><Name>Rugby</Name><Selected>True<Selected><Coach>john</Coach></Sport>
</Sports>
```

The Job Task Assign properties are like this

```
{ "name": "Task Assign Repeating", "value": "true" },
{ "name": "Task Assign Repeat Item", "value": "$func.invoke('Get Repeat Item From Sport
Name', ${assignRepeatIndex})" },
{ "name": "Task Assign User", "value": "$formDataMap.assignDelimited" },{ "name": "Task
Subject",  "value": "${assignRepeatItem} application from ${assignee}" }

$formDataMap.assignDelimited = "john|al|john"
```

In this example 3 tasks are created:

- (Assignee: john, AssignRepeatIndex: 1, AssignRepeatItem: Cricket) and the Task Subject will be "Cricket application by Ginger Meggs"
- (Assignee: al, AssignRepeatIndex: 2, AssignRepeatItem: Basketball) and the Task Subject will be "Basketball application by Ginger Meggs"
- (Assignee: john, AssignRepeatIndex: 3, AssignRepeatItem: Rugby) and the Task Subject will be "Rugby application by Ginger Meggs"

Next, learn about task assign actions.

Collaboration Jobs Reuse

Journey Manager (JM) | System Manager / DevOps | 19.05

Configuring collaboration jobs from scratch can be a tedious task, so Manager lets you **re-use**[1] the following components to make the configuration easier and increase your productivity:

- Existing collaboration jobs.
- Action services, which can be reused between collaboration jobs and within the same collaboration job by providing different attributes and action properties.
- Common services when you can balance isolation of services between organizations.
- Common collaboration job actions.
- Forms when a collaboration job can be associated with many forms.
- Form code, when it uses the task, for example:`{"name": "Task Form Code", "value": "$func.startFormCode()" }`
- Form properties, when it stores the name of the form, for example: `{"name": "Task Assign Group", "value": "$func.formProperty('Initial Review')" }`

Next, learn how to view job services.

[1]Code reuse aims to save time and resources and reduce redundancy by taking advantage of assets that have already been created in some form within the software product development process. The key idea in reuse is that parts of a computer program written at one time can be or should be used in the construction of other programs written at a later time.

Form Bundles Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager comes with form bundles, which allows you to implement customer on-boarding solutions where a user can select several products. A form bundle contains two or more forms grouped together in a one step, so they can be used instead of building a single monolithic form. Breaking a large form into more manageable chunks makes it easier for you to develop and maintain. Form bundles also enable you to develop parts of a form concurrently.

Manager has the following form bundle features:

- Authenticated form bundles
- Anonymous form bundles
- Share form data
- Form bundle context
- Dynamic preconditions
- Form chaining
- shareExtractData

A sample job definition of a form bundle collaboration job is shown below.

```
{
  "jobDetails": {
    "name": "Customer Onboarding Job",
    "processSubmitImmediate": true,
    "version": "4.3.0"
  },
  "steps": [
    {
      "name": "Customer Onboarding",
      "type": "start",
      "actions": [
        {
          "name": "Customer Submission",
          "type": "Job Form Start",
          "redirectNext": true
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Additional Products" }
      ]
    },
    {
      "name": "Additional Products",
      "type": "",
      "shareExtractData": true,
      "allFormsEditable": true,
      "showPreviousForms": true,
      "redirectNext": true,
      "actions": [
        {
          "name": "Credit Cards Application",
          "type": "Job Task Assign",
          "preCondition": "$formDataMap.productCreditCards == 'true'",
          "redirectNext": true,
          "properties": [
            { "name": "Task Assign Email",  "value": "$formDataMap.emailAddress" },
            { "name": "Task Form Code",  "value": "onboard-credit-cards" },
            { "name": "Task Message",  "value": "Please complete the Credit Card Applic-
ation form." },
            { "name": "Task Subject",  "value": "Complete Credit Card Application" },
            { "name": "Task Input XML Prefill",  "value": "$func.startSubmissionXml()" },
            { "name": "Task Type",  "value": "Anonymous" }
          ]
        },
```

```
        {
          "name": "Insurance Application",
          "type": "Job Task Assign",
          "preCondition": "$formDataMap.productInsurance == 'true'",
          "redirectNext": true,
          "properties": [
            { "name": "Task Assign Email",  "value": "$formDataMap.emailAddress" },
            { "name": "Task Form Code",  "value": "onboard-insurance" },
            { "name": "Task Message",  "value": "Please complete the Insurance Applic-
ation form." },
            { "name": "Task Subject",  "value": "Complete Insurance Application" },
            { "name": "Task Input XML Prefill",  "value": "$func.startSubmissionXml()" },
            { "name": "Task Type",  "value": "Anonymous" }
          ]
        },
        {
          "name": "Appliction Confirmation",
          "type": "Job Task Assign",
          "preCondition": "$formDataMap.productInsurance == 'true' || $formDataMap.-
oroductCreditCards == 'true'",
          "properties": [
            { "name": "Task Assign Email",  "value": "$formDataMap.emailAddress" },
            { "name": "Task Form Code",  "value": "onboard-confirmation" },
            { "name": "Task Message",  "value": "Please sign to complete Application
oundle." },
            { "name": "Task Subject",  "value": "Confirm Application Bundle" },
            { "name": "Task Input XML Prefill",  "value": "$func.startSubmissionXml()" },
            { "name": "Task Type",  "value": "Anonymous" }
          ]
        },
        {
          "name": "Review Wait",
          "type": "Job Task Wait"
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Application Delivery" }
      ]
    },
    {
      "name": "Application Delivery",
      "type": "",
      "actions": [
        {
```

```
          "name": "Application Delivery",
          "type": "Job Delivery",
          "properties": [
            { "name": "Delivery Mode",  "value": "All Submissions" }
          ]
        },
        {
          "name": "Application Delivery Wait",
          "type": "Job Delivery Wait"
        }
      ],
      "routes": [
        { "name": "Default",  "nextStep": "Application Completed" }
      ]
    },
    {
      "name": "Application Completed",
      "type": "endpoint"
    }
  ]
}
```

A typical form bundle asks users to provide basic user details, such as a name and address, and products they want to sign up for in a triage form.

When a user submits the triage form, Manager performs the following:

1. Executes the next section of a collaboration job, which is the bundle step.
2. Associates a task with a job action.
3. Assigns a task to the user who submitted the triage form.
4. Redirects a browser to load the next available Form Task.

It makes the whole user experience looks like they have just moved to the next page of a bigger form.

A form bundle provides the following additional features:

Sharing Data Between Tasks

There are three valid options that can be taken to share data between forms in a form bundle:

No Sharing

This sets the step attribute **`shareFormData`** is false, and **`shareExtractData`** is false, so no data is copied between the forms.

Shared XML Data Model

This sets the step attribute **shareFormData** is true, and **shareExtractData** is false. When a Form Task is submitted the data is automatically written to the other forms. To use this , all forms must use the same data model. This is the most common way to share data between forms.

Share Extract Data

This sets the step attribute **shareFormData** is false, and **shareExtractData** is true. This can be used where the forms don't have a common data model. This type of sharing is also called **Publish - Subscribe** sharing of data extracts.

Let's demonstrate it with a simple example where a form A sets a field as a data extract Demo Type and a form B subscribes to the same data extract Demo Type. The form A is opened, the Demo Type data extract value is changed to a new value and then the form A is submitted. The Demo Type data extract is published on the save action. Because the form B has subscribed to the value, it picks its data extract and form XML is updated with the new value.

Show Previous Forms

The form bundle context is populated in the task when the step attribute **showPreviousForms** is true. You can see the content of the data that is returned to the form and how the form and widget can use it to display the previous forms in the form bundle.

Forms in the Same Step Editable

Forms in a bundle step are editable when the step attribute **allFormsEditable** is true and when the step has still not completed. This happens even if the form tasks in the step have previously been submitted.

You can use the **Collaboration Job Bundle Navigation** widgets to list form links to previous forms. Click on a link to open the selected form as you will be redirected to the form. You can also click on a link to download a receipt for a previous submitted form.

Preconditions

All the Job Task Assign actions in form bundle steps can have optional preconditions, such as **"preCondition": "$formDataMap.productCreditCards == 'true'"**. The precondition is triggered, if the velocity template on the right evaluates to true, so the action instance and associated task are created.

By default, the preconditions are evaluated against a start submission. If the previous submission isn't the start submission, its possible that the data extract fields are populated in the previous submission, but are missing from the start submission. In this case, the precondition may not fire.

To fix this, you must specify the **Precondition Submission** step to be used for evaluating preconditions. You should replace the "Triage Step Name" with the actual step name that contains triage form submission.

```
{
  "name": "A Form Bundle",
  "type": "",
  "dynamicPreConditions": true,
  "shareFormData": true,
  "showPreviousForms": true,
  "redirectNext": true,
  "actions": [ ... ],
  "properties": [
    {"name": "Pre Condition Submission Step", "value" : "Triage Step Name"}
  ],
  "routes": [ ... ]
}
```

Dynamic Preconditions

Normal or static preconditions are evaluated once when the step is first run using the data extracts for the pre-condition submission. Dynamic preconditions fire after the submission of any task in a form bundle. The data extracts are taken from the form bundle and are submitted and evaluated against the actions whose pre-conditions have not already fired.

Actions that have already been created, as a result of the previous preconditions being true, are not affected, if their preconditions are evaluated to false now. To configure **`dynamicPreconditions`** in a form bundlestep, add the **`"dynamicPreConditions": true`** step's attribute:

Dynamic Precondition and Action Order

The order that the forms are made available in a form bundle is taken from the order they are listed in a job definition of a collaboration job. With dynamic preconditions the next form loaded is the highest one on the list that hasn't been submitted yet.

Let's use the Additional Products step of a collaboration job as an example. A user can select the insurance product option only in the triage form. The result is the insurance application and the application confirmation are created.

The next available form is the Insurance Application so it's opened next, so the user can now select the Credit Card product. The dynamic precondition is triggered and the Credit Card Application form is created.

The next available form is the Credit Card Application, because it is at the top of the list. The user submits this form.

As the Insurance Application was already submitted, the form chaining opens the next available form that is the Application Confirmation.

The Application Confirmation lists all the forms in order that they appear in the collaboration job, as shown below:

```
{
  "name": "Additional Products",
  "type": "",
  "dynamicPreConditions": true,
  "shareFormData": true,
  "showPreviousForms": true,
  "redirectNext": true,
  "actions": [
    {
      "name": "Credit Cards Application",
      "type": "Job Task Assign",
      "preCondition": "$formDataMap.productCreditCards == 'true'",
      ...
    },
    {
      "name": "Insurance Application",
      "type": "Job Task Assign",
      "preCondition": "$formDataMap.productInsurance == 'true'",
      ...
    },
    {
      "name": "Appliction Confirmation",
      "type": "Job Task Assign",
      ...
    },
    ...
  ],
  "routes": [ ... ]
}
```

User Authentication

Form bundles support both anonymous and authenticated flows, which is based on the Form or Review task. For more information on what action properties are available to the task, see action properties.

> Mixing of task types, such as Anonymous, Form and Review, in the same form bundle step is tot supported.

Next, learn about a form bundle context.

Form Bundles Context

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A collaboration job provides additional context information to form bundles created with Maestro, Composer, or the Maguire template. This information is automatically used by forms and widgets to provide a list of all the other forms in the form bundle, the completion status and a way to navigate to the chosen form as well as print a receipt. A form bundle content includes **`allFormsEditable`** option that enables users to open any previously completed forms of their application bundle in a step.

The form XML shown below is the part of the customer onboarding collaboration job where the data is created in the **`//SystemProfile/Job/StepTasks`** XML node, which is a list of the previous and current step tasks for the form bundle.

```
<SystemProfile>
    <DisplayMode>Entry</DisplayMode>
    <HostContext>Page</HostContext>
    <ReceiptNumber></ReceiptNumber>
    <SubmitDateString></SubmitDateString>
    <SubmissionMessage></SubmissionMessage>
    <FormDataSer-
viceURL>ht-
tps://tm.maestro.avoka.com/maguire/servlet/FormDynamicDataServlet</FormDataServiceURL>
    <RequestLogKey>ab45894a127f2e26a3e516ba4291080a</RequestLogKey>
    <TrackingCode>PT3F6P</TrackingCode>
    <Job>
        <ReferenceNumber>QF3FJK</ReferenceNumber>
        <StepName>Additional Products</StepName>
        <AvailableRoutes>Default</AvailableRoutes>
        <RouteName></RouteName>
        <Assignee>jsmith@gmail.com</Assignee>
        <AssignRepeatIndex></AssignRepeatIndex>
        <AssignRepeatItem></AssignRepeatItem>
        <Description></Description>
        <StepTasks>
            <Task>
                <StepName>Customer Onboarding</StepName>
                <MenuLabel>Customer Onboarding</MenuLabel>
                <MenuCategory>Customer Onboarding</MenuCategory>
                <TaskSubject>Customer Onboarding</TaskSubject>
                <FormCompleted>true</FormCompleted>
                <FormEditable>false</FormEditable>
                <FormName>Customer Onboarding</FormName>
                <ReceiptURL>ht-
tps://t-
m.maes-
tro.a-
voka.-
com/maguire/ser-
vlet/FormRe-
ceipt.pdf?submitKey=dfe15aab5345ee23505021f889ac0ff1&amp;renderMode=file</ReceiptURL>
                <CurrentForm>false</CurrentForm>
                <TaskMessage></TaskMessage>
                <ScheduledTime></ScheduledTime>
                <ExpiryTime></ExpiryTime>
                <FormURL></FormURL>
            </Task>
            <Task>
```

```
                <StepName>Additional Products</StepName>
                <MenuLabel>Complete Credit Card Application</MenuLabel>
                <MenuCategory>Additional Products</MenuCategory>
                <TaskSubject>Complete Credit Card Application</TaskSubject>
                <TaskMessage>Please complete the Credit Card Application form.</TaskMes-
sage>
                <FormCompleted>false</FormCompleted>
                <FormEditable>true</FormEditable>
                <FormName>Customer Onboarding - Credit Cards</FormName>
                <FormURL>ht-
tps://t-
n.maestro.avoka.com/maguire/form.htm?submitKey=2ef6da250d9d1dbee74749172f64e3e4</FormURL>
                <CurrentForm>true</CurrentForm>
                <ScheduledTime></ScheduledTime>
                <ExpiryTime></ExpiryTime>
                <ReceiptURL></ReceiptURL>
            </Task>
            <Task>
                <StepName>Additional Products</StepName>
                <MenuLabel>Complete Insurance Application</MenuLabel>
                <MenuCategory>Additional Products</MenuCategory>
                <TaskSubject>Complete Insurance Application</TaskSubject>
                <TaskMessage>Please complete the Insurance Application form.</TaskMes-
sage>
                <FormCompleted>false</FormCompleted>
                <FormEditable>true</FormEditable>
                <FormName>Customer Onboarding - Insurance</FormName>
                <FormURL>ht-
tps://t-
n.maestro.avoka.com/maguire/form.htm?submitKey=ff13980c03dccd2963ba8c6be469e248</FormURL>
                <CurrentForm>false</CurrentForm>
                <ScheduledTime></ScheduledTime>
                <ExpiryTime></ExpiryTime>
                <ReceiptURL></ReceiptURL>
            </Task>
            <Task>
                <StepName>Additional Products</StepName>
                <MenuLabel>Confirm Application Bundle</MenuLabel>
                <MenuCategory>Additional Products</MenuCategory>
                <TaskSubject>Confirm Application Bundle</TaskSubject>
                <TaskMessage>Please sign to complete Application bundle.</TaskMessage>
                <FormCompleted>false</FormCompleted>
                <FormEditable>true</FormEditable>
                <FormName>Customer Onboarding - Confirmation</FormName>
```

```
                <FormURL>ht-
tps://t-
m.maestro.avoka.com/maguire/form.htm?submitKey=8e6b6352eb284b03b10ea36fa0b454f2</FormURL>
                <CurrentForm>false</CurrentForm>
                <ScheduledTime></ScheduledTime>
                <ExpiryTime></ExpiryTime>
                <ReceiptURL></ReceiptURL>
            </Task>
        </StepTasks>
    </Job>
```

Next, learn about Job Definition elements for a Form Bundle configuration.

Configure Form Bundles

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to configure form bundles using the following pre-defined set of job definition elements that are part of step and action attributes or properties.

Step Attributes For Form Bundles with More Than Two Tasks

The following are ignored if there is only one task in a step.

dynamicPreConditions -re-evaluates action preconditions for forms with the multiple step, if set to true. Otherwise, they are only evaluated on the triage form.

shareFormData - copies the entire form XML data between form tasks within a form bundle step.

shareExtractData - copies data between form tasks within a form bundle step using a publish subscribe method.

allFormsEditable - if set to true, the form tasks within a form bundle are mutable, until the step is completed.

Step Attributes For Form Bundles with One or More Tasks

showPreviousForms - shows the previous forms.

redirectNext - allows form chaining along with the action attribute redirect next.

Step Properties

This is used to specify the triage step and is only required if triage form is not the start form.

```
"properties": [
    {"name": "Pre Condition Submission Step", "value" : "Triage Step Name"}
  ],
```

Action Properties

preCondition - allows you to use the **Apache velocity templates**[1].

[1]Apache Velocity Templates are used extensively throughout Journey platformto define html page layout, email body and subject. A template is combined with model containing a number or objects to produce a String. For more information see http://velocity.apache.org/engine/devel/user-guide.html

`redirectNext` - allows form chaining along with the step attribute redirect next.

Next, learn about anonymous Form Bundles.

Anonymous Form Bundles

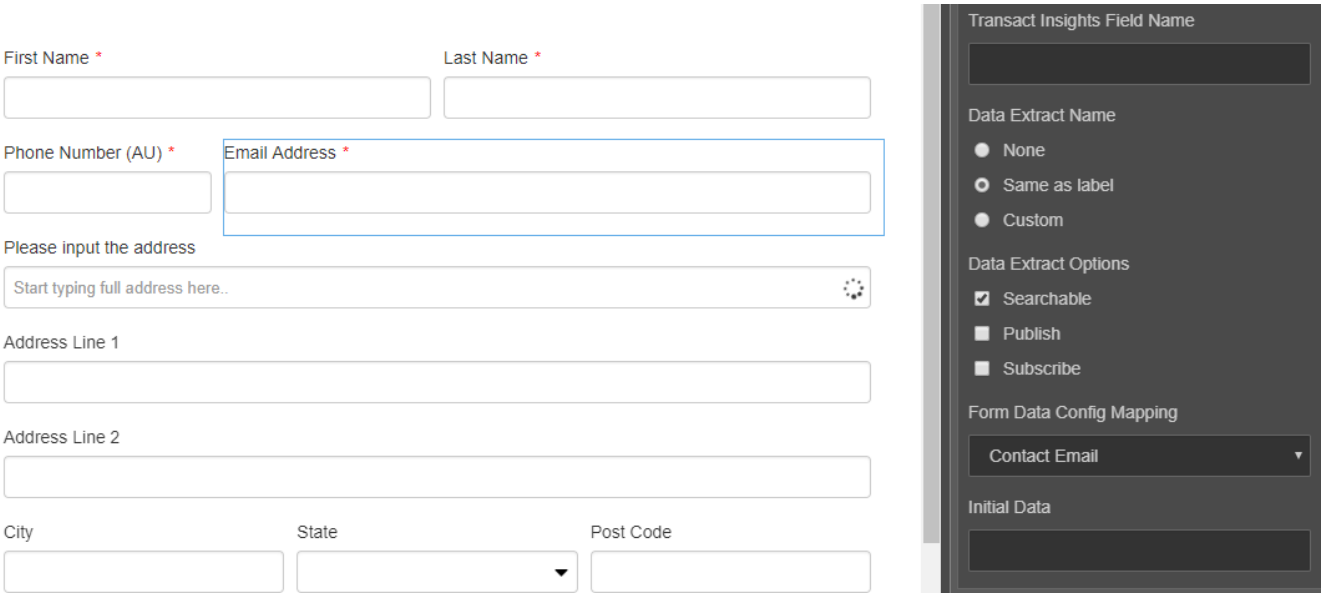
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Anonymous bundles are form bundles that require the use of an email address to allow form users to partially fill out the forms and then complete them later. For example, a triage form can ask a user to enter an email address, so after the form is filled out and submitted, the user can be sent an email with a link to open and continue editing this form bundle. You can use a hard coded contact email in the form, but you must decide how much information the user should re-enter, in case they are not able to complete the form in one go. Regardless of the email address, an anonymous form bundle must be always protected with a save challenge.

We will demonstrate anonymous form bundles with the Maestro Onboarding Job and four Maestro forms examples.

Configure a Form

Below is the **Maestro Onboarding - Triage** form including the **Email Address** field. This form starts the anonymous collaboration job.



1. Select the **Same as label** checkbox under the **Data Extract Name**, so you can reference it in the collaboration job as **`$formDataMap.emailAddress`**.
2. Select the **Contact Email Address** from the **Form Data Config Mapping** dropdown list.

> This email address can't be blank, so the field must be mandatory where the email address is to be entered by the form user.

You can hard code an email address, such as onboarding@company.com, in the form by doing one of the following:

- Setting the initial data value to onboarding@company.com.
- Creating a Maestro form rule to hide the email address or to make it read-only.

However, we don't recommend using a hard-coded address.

Job Definition - Additional Products

The job definitions for additional products shows the configuration bundle step below:

```
{
	"name": "Additional Products",
	"type": "",
	"dynanicPreConditions": true,
	"shareExtractData": false,
	"shareFormData": true,
	"allFormsEditable": true,
	"showPreviousForms": true,
	"redirectNext": true,
	"actions": [
		{
		"name": "Credit Cards Application",
		"type": "Job Task Assign",
		"preCondition": "$formDataMap.productCreditCards == 'true'",
		"redirectNext": true,
		"properties":[
			{"name": "Task Assign Email", "value": "$formDataMap.enailAddress"},
			{"name": "Task Form Code", "value": maestroonboardingcre},
			{"name": "Task Message", "value": "Please complete the Credit Card Application
form."},
			{"name": "Task Subject", "value": "Complete Credit Card Application"},
			{"name": "Task Input XML Prefil", "value": "$func.startSubmissionXml()"},
			{"name": "Task Type", "value": "Anonymous"},
			{"name": "Task Save Challenge", "value": "$formDataMap.phoneNumber"}
		]
		},
		{
		"name": "Insurance Application",
		"type": "Job Task Assign",
		"preCondition": "$formDataMap.productInsurance == 'true'",
		"redirectNext": true,
		"properties": [
		{"name": "Task Assign Email", "value": "$formDataMap.enailAddress"},
			{"name": "Task Form Code", "value": maestroonboardingins},
			{"name": "Task Message", "value": "Please complete the Insurance Application for
			{"name": "Task Subject", "value": "Complete Insurance Application"},
			{"name": "Task Input XML Prefil", "value": "$func.startSubmissionXML()"},
			{"name": "Task Type", "value": "Anonymous"},
			{"name": "Task Save Challenge", "value": "$formDataMap.phoneNumber"}
		]
		},
...	]
}
```

The following action properties are typically used for an anonymous form bundle:

- `{ "name": "Task Type", "value": "Anonymous" }`
- `{ "name": "Task Assign Email", "value": "$formDataMap.emailAddress" }`. Anonymous form bundles require an email to be set. If the value is blank, the Job Task Assign service throws an error. The email address can be set to a value or it can be read from a data extract of the triage form.
- `{ "name": "Task Input XML Prefill", "value": "$func.startSubmissionXml()" }`. The Anonymous form task gets the XML data from the previous submission. You can also use **`$func.stepOrStartSubmissionXml()`** instead. This is required if the step uses the **`dynamicPreConditions`**. For more information, see job functions.
- `{ "name": "Task Save Challenge", "value": "${formDataMap.phoneNumber}" }`. The save challenge data is set for the form.

> It is possible for anonymous tasks to copy attachments from a previous task, but sharing of attachments is not supported in form bundles. For more information, see action properties.

Example 1

The Job Controller below has been extracted from the **mOnboarding - Anonymous - shareExtractData** job definition JSON above with the following step level attributes: **`dynamicPreConditions`**, **`shareExtractData`**, as well as the standard ones: **`allFormsEditable`**, **`showPreviousForms`** and **`redirectNext`**.

```
"dynanicPreConditions": true,
"shareExtractData": true,
"shareFormData": false,
"allFormsEditable": true,
"showPreviousForms": true,
"redirectNext": true,
```

Example 2

In this example, the job controller is extracted from the **mOnboarding - Anonymous - shareFormData** job definition above with the following step attributes:

- **`dynamicPreConditions`**, **`ShareFormData`**
- **`allFormsEditable`**,
- **`showPreviousForms`**
- **`redirectNext`**.

```
"dynanicPreConditions": true,
"shareExtractData": true,
"shareFormData": false,
"allFormsEditable": true,
"showPreviousForms": true,
"redirectNext": true,
```

Next, learn about authenticated form bundles.

Authenticated Form Bundles

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

You can enforce an authenticated access to form bundles by using authenticated tasks that are assigned to a group. Authenticated form bundles can be built based on a form and Review Tasks types. However, we recommend you use the Form Task types.

Authenticated tasks can be one of the following:

User Task

The current user is assigned to a new task:

```
{"name": "Task Assign User", "value": "$func.startUser()" }
```

Alternatively

```
{"name": "Task Assign User", "value": "$func.previousUser()" }
```

Non Claimable Tasks

This task is assigned to a group the current user belongs to, for example, the accounts group:

```
{"name": "Task Assign Group", "value": "accounts" }
```

It can be assigned to two groups, if there is a supervisor group for the task:

```
{"name": "Task Assign Groups", "value": "accounts,supervisor" }
```

Claimable Group Tasks

This is also required to be claimed by the current user, which is required for the `redirectNext` to work:

```
{"name": "Task Assign User", "value": "$func.startUser()" }
```

```
{"name": "Task Assign Group", "value": "accounts" }
```

If there is a supervisor group for the task:

```
{"name": "Task Assign User", "value": "$func.previousUser()" }
```

```
{"name": "Task Assign Groups", "value": "accounts,supervisor" }
```

A hard-coded address is not recommended.

Job Definition - Additional Products

The job definition for additional products shows the form bundle configuration below:

```
{
	"name": "Additional Products",
	"type": "",
	"dynanicPreConditions": true,
	"shareExtractData": true,
	"shareFormData": false,
	"allFormsEditable": true,
	"showPreviousForms": true,
	"redirectNext": true,
	"actions": [
		{
			"name": "Credit Cards Application",
			"type": "Job Task Assign",
			"preCondition": "$formDataMap.productCreditCards == 'true'",
			"redirectNext": true,
			"properties":[
				{"name": "Task Assign User", "value": "$formDataMap.startUser()"},
				{"name": "Task Form Code", "value": maestroonboardingcred2},
				{"name": "Task Message", "value": "Please complete the Credit Card Appli
form."},
				{"name": "Task Subject", "value": "Complete Credit Card Application"},
				{"name": "Task Input XML Prefil", "value": "$func.stepOrStartSubmissionX
				{"name": "Task Type", "value": "Form"},
			]
		},
		{
			"name": "Insurance Application",
			"type": "Job Task Assign",
			"preCondition": "$formDataMap.productInsurance == 'true'",
			"redirectNext": true,
			"properties": [
				{"name": "Task Assign User", "value": "$formDataMap.startUser()"},
				{"name": "Task Form Code", "value": maestroonboardinginsu2},
				{"name": "Task Message", "value": "Please complete the Insurance Applica
				{"name": "Task Subject", "value": "Complete Insurance Application"},
				{"name": "Task Input XML Prefil", "value": "$func.stepOrStartSubmissionX
				{"name": "Task Type", "value": "Form"},
			]
		},
...	]
}
```

Example 1

The job controller below has been extracted from the **mOnboarding - Form Tasks - shareExtractData** job definition. Its features setup in the with step level attribute flags **dynamicPreConditions**, **shareExtractData**, as well as the standard **allFormsEditable**, **showPreviousForms** and **redirectNext**.

All form tasks in the form bundle require action properties, such as:

- `{ "name": "Task Type", "value": "Form" }`
- `{ "name": "Task Input XML Prefill", "value": "$func.stepOrStartSubmissionXml()" }` or `{ "name": "Task Input XML Prefill", "value": "$func.stepOrPreviousSubmissionXml()" }`

> **$func.stepOrStartSubmission** or **$func.stepOrPreviousSubmissionXml()** are required for dynamic preconditions to work. They also work for the static bundles.

Example 2

This example uses a form task bundle configured with the **shareFormData** attributes:

```
"shareExtractData": true,
"shareFormData": false
```

Review Task

Review tasks are primarily used for review and approval processes that use the same form to review as was submitted by the user. They are configured by specifying a previous step or submission. The review task will copy the XML data, attachments from the triage form. You need to specify a form code so the review task doesn't use the triage form. Review tasks should be only configured with the **shareExtractData** option.

Example 3

This example is extracted from **mOnboarding - Review Task - shareExtractData**. The attributes are the same as in the job definition of the Example 1.

The Task Type should be Review:

`{ "name": "Task Type", "value": "Review" }`

The following code snippet retrieves the review data from a previous step:

`{ "name": "Task Review Previous Step", "value": "true" }`

```
"dynanicPreConditions": true,
"shareExtractData": true,
"shareFormData": false,
"allFormsEditable": true,
"showPreviousForms": true,
"redirectNext": true
```

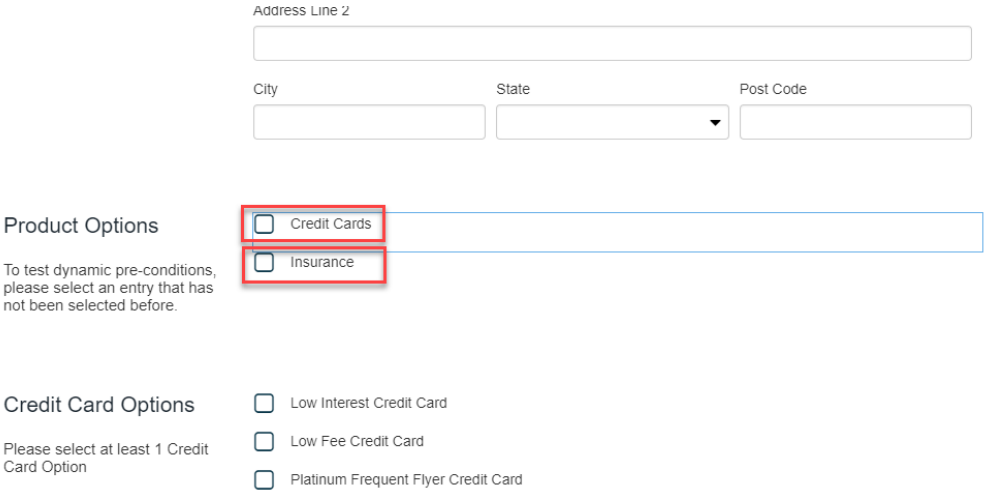
Next, learn about anonymous form bundles.

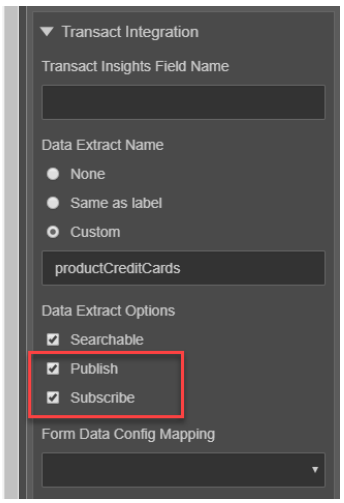
Form Bundle Flow Example

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

The following example illustrates the simple usage of a form bundle as an anonymous form bundle flow that uses nearly all **redirectNext**, **sharedExtractData** (from a users perspective the **shareFormData** looks the same), **dynamicPreConditions**, **preconditions**, **allFormsEditable**.

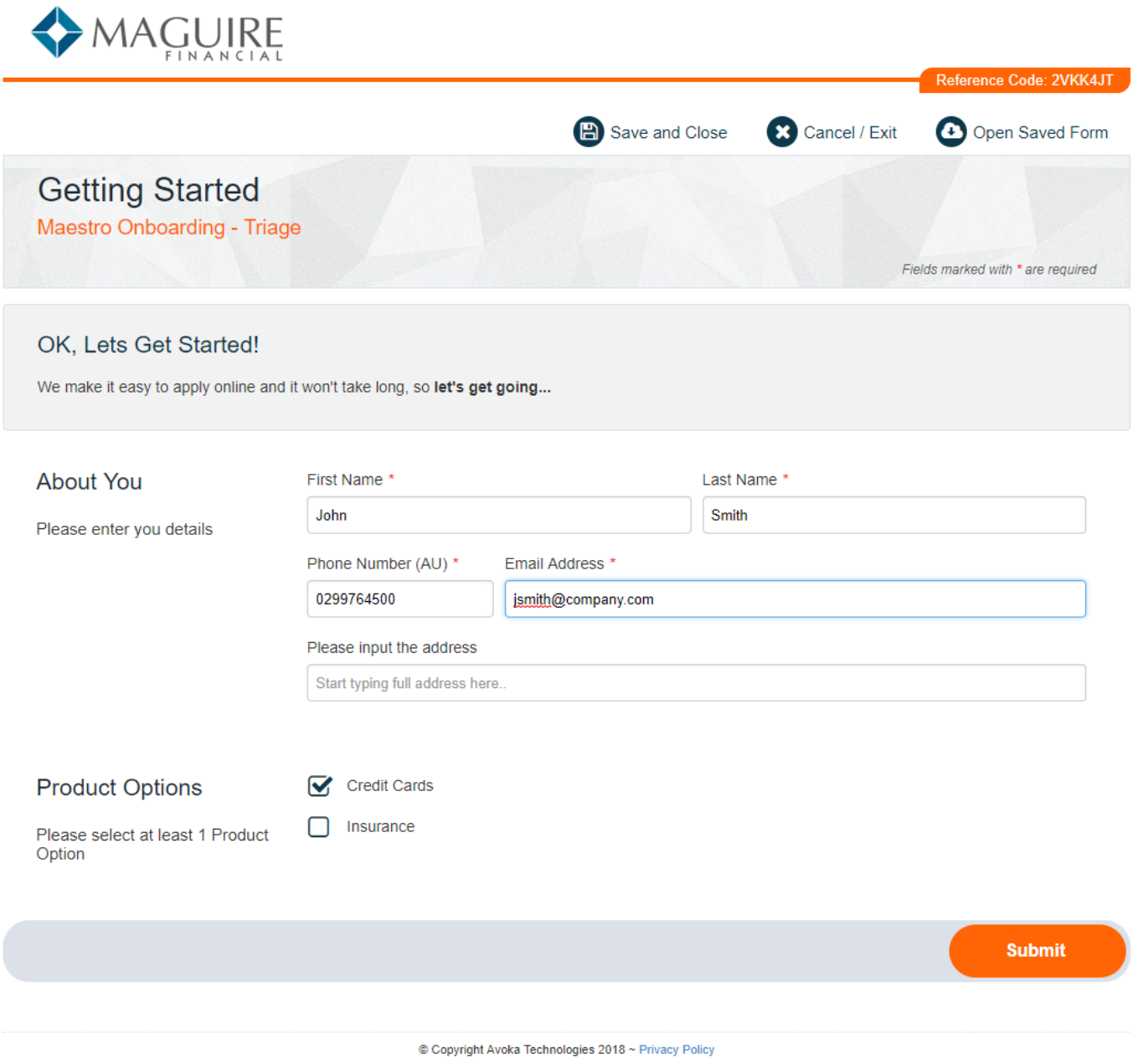
The test cases extend the Maestro onboarding example where the Credit Card Application and Insurance Application forms have been replaced with forms that include the Product Options to allow you to test the **dynamicPreCondition** attribute. The Product options - Credit Card and insurance check boxes are set to both Publish and Subscribe. This allows you to test the **shareExtractData** step attribute. The Maestro build form is shown below.





The Flow

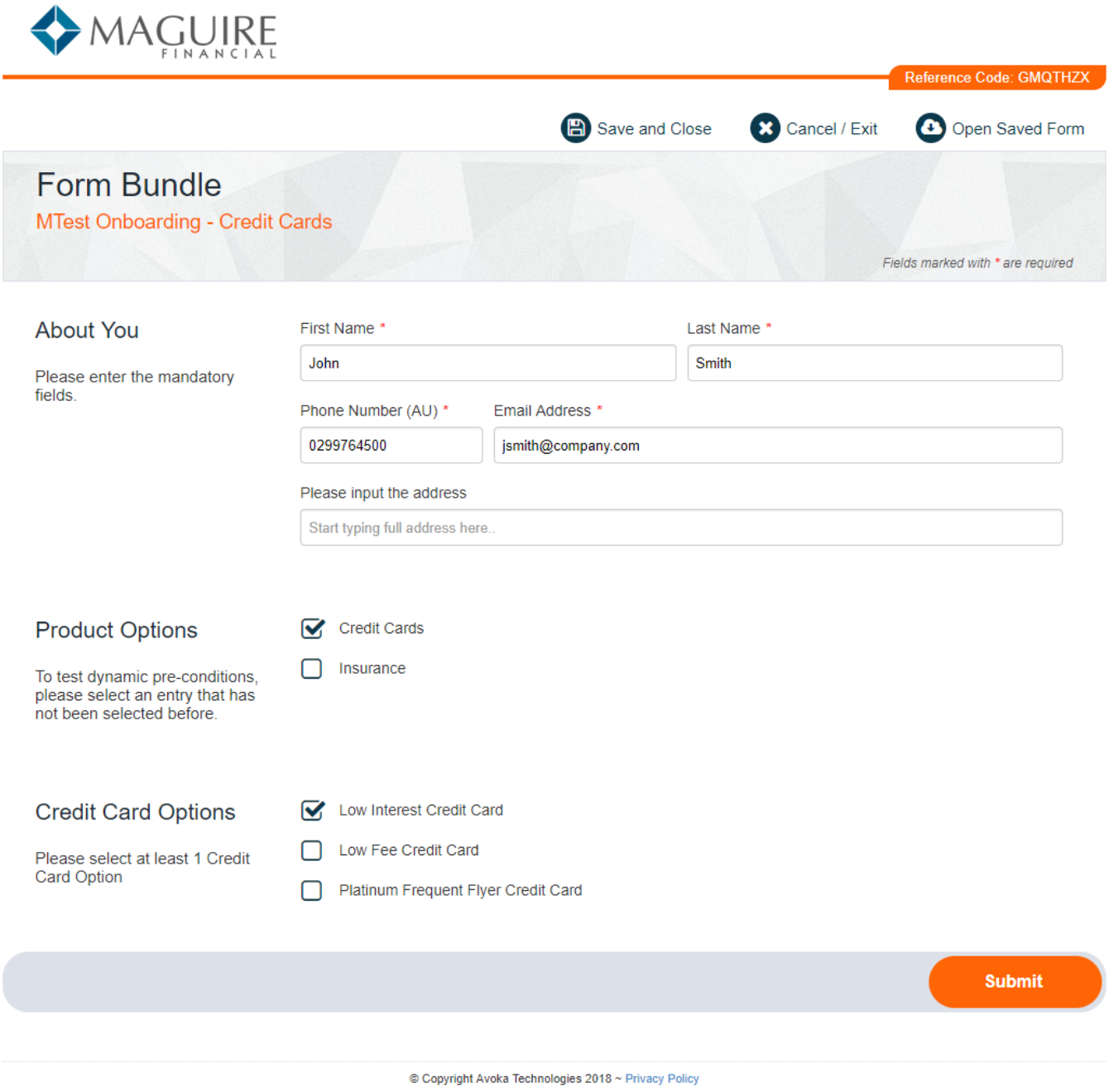
A form user fills out the Triage form and submits it.



Manager executes the Additional Package step. Preconditions match for Credit Cards and the Confirmation Page and tasks are created.

The browser is redirected to the next available task, which is the Credit Card.

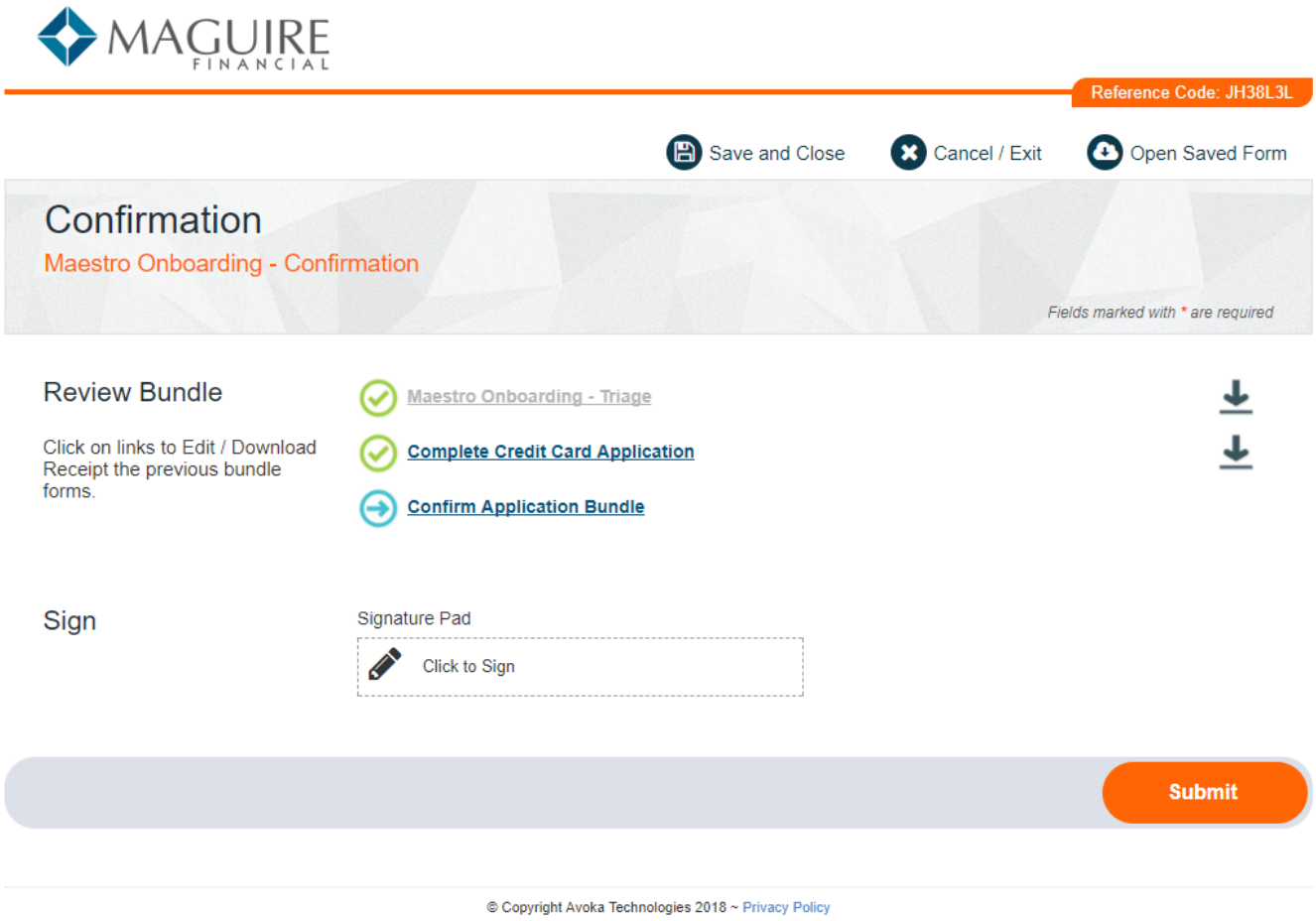
The user selects the first credit card application and clicks **Submit**.



The browser is redirected to the Application Confirmation page, so you can see the Triage Form (Grayed out - it is outside the step and is read only, for example, print receipt)

You can see the Credit card for has been completed.

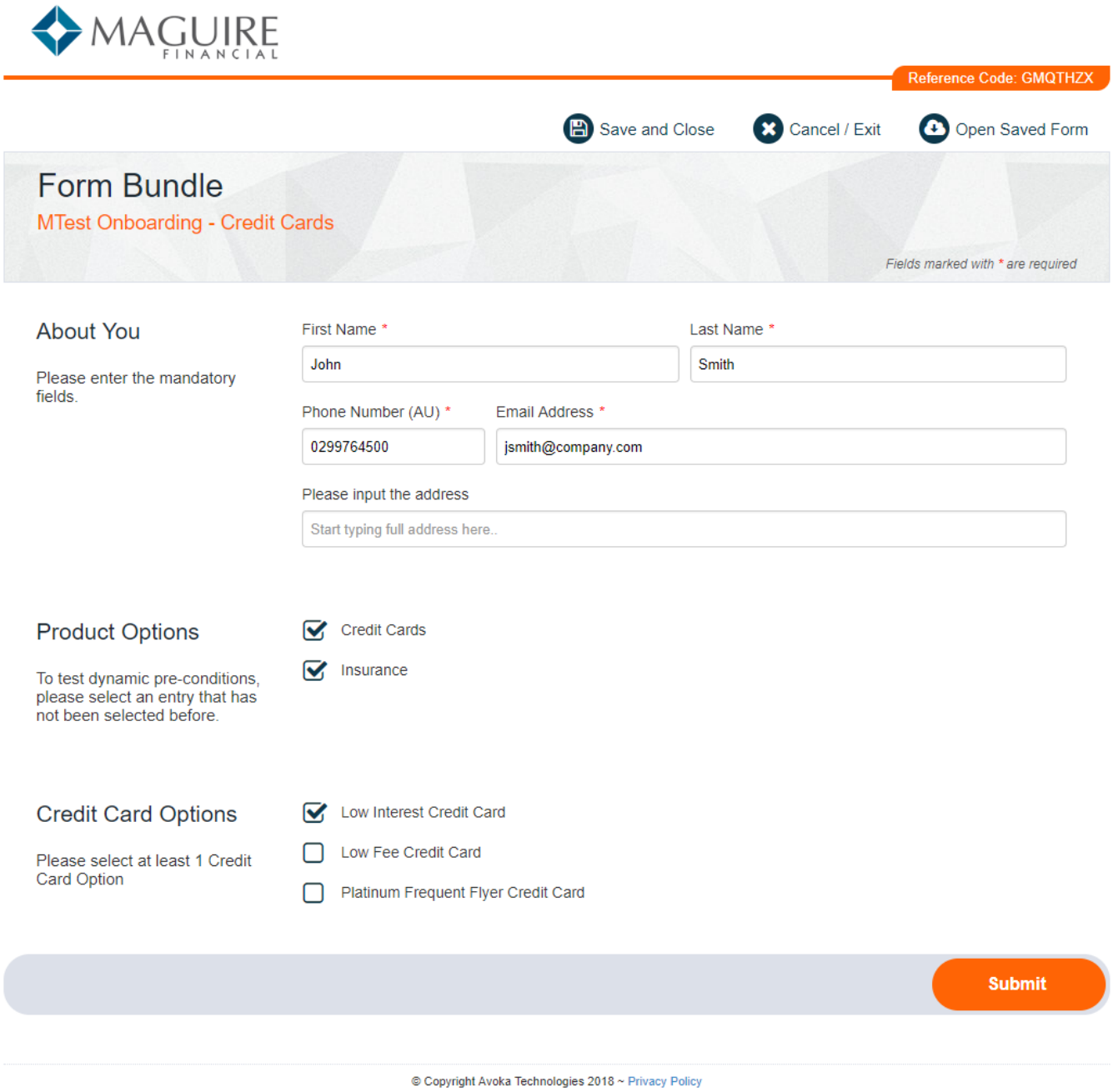
Click on the link to edit it. The **`allFormsEditable`** is set to true in the Additional Products step.



The browser redirects back to the Credit Card Page. This shows the behavior of **`dynamicPreConditions`**.

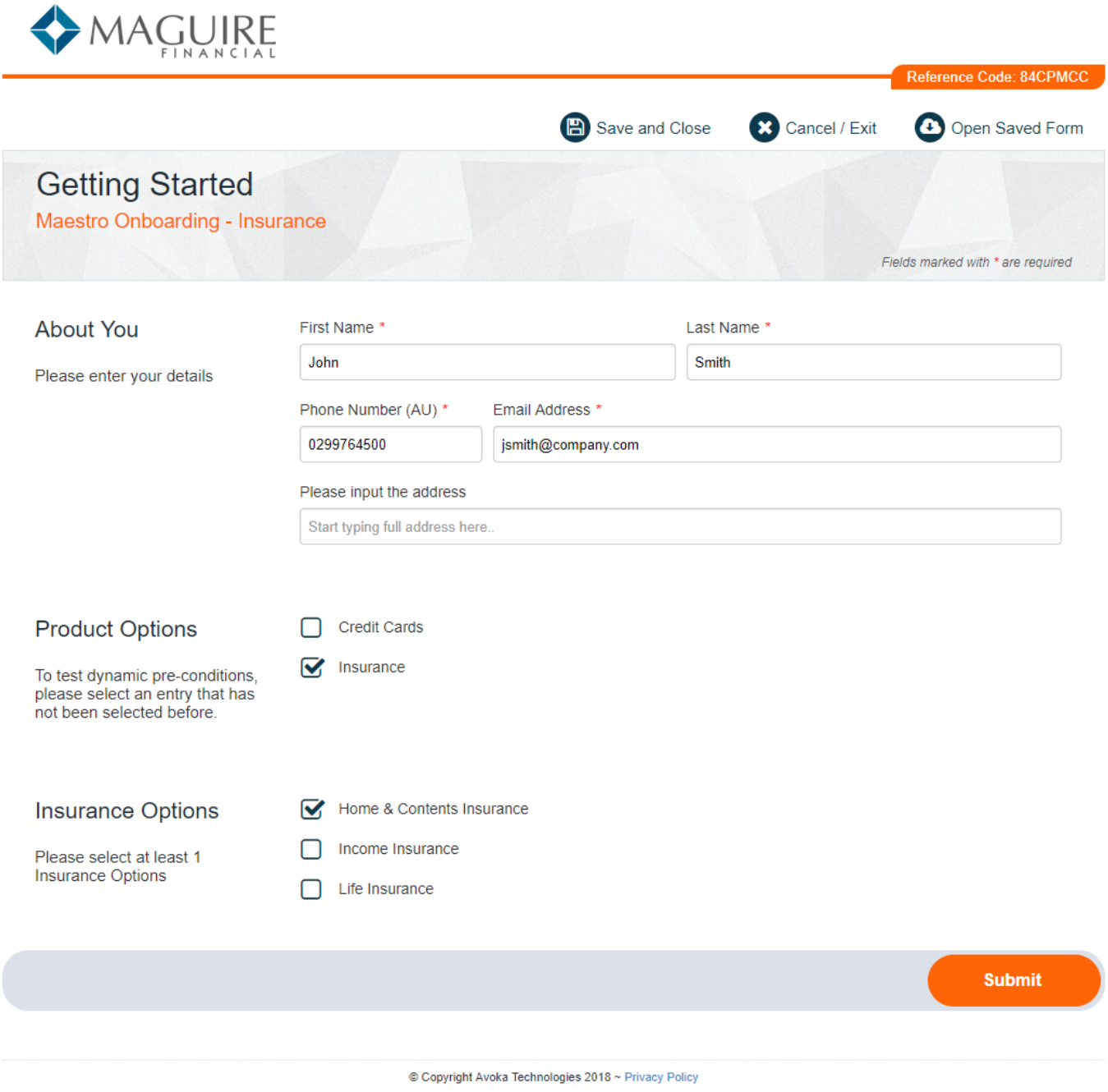
Select Product Options: Insurance and click **Submit**.

The data extracts for the Product Options insurance are set to true and the precondition for the insurance task is evaluated and an Insurance Task created.



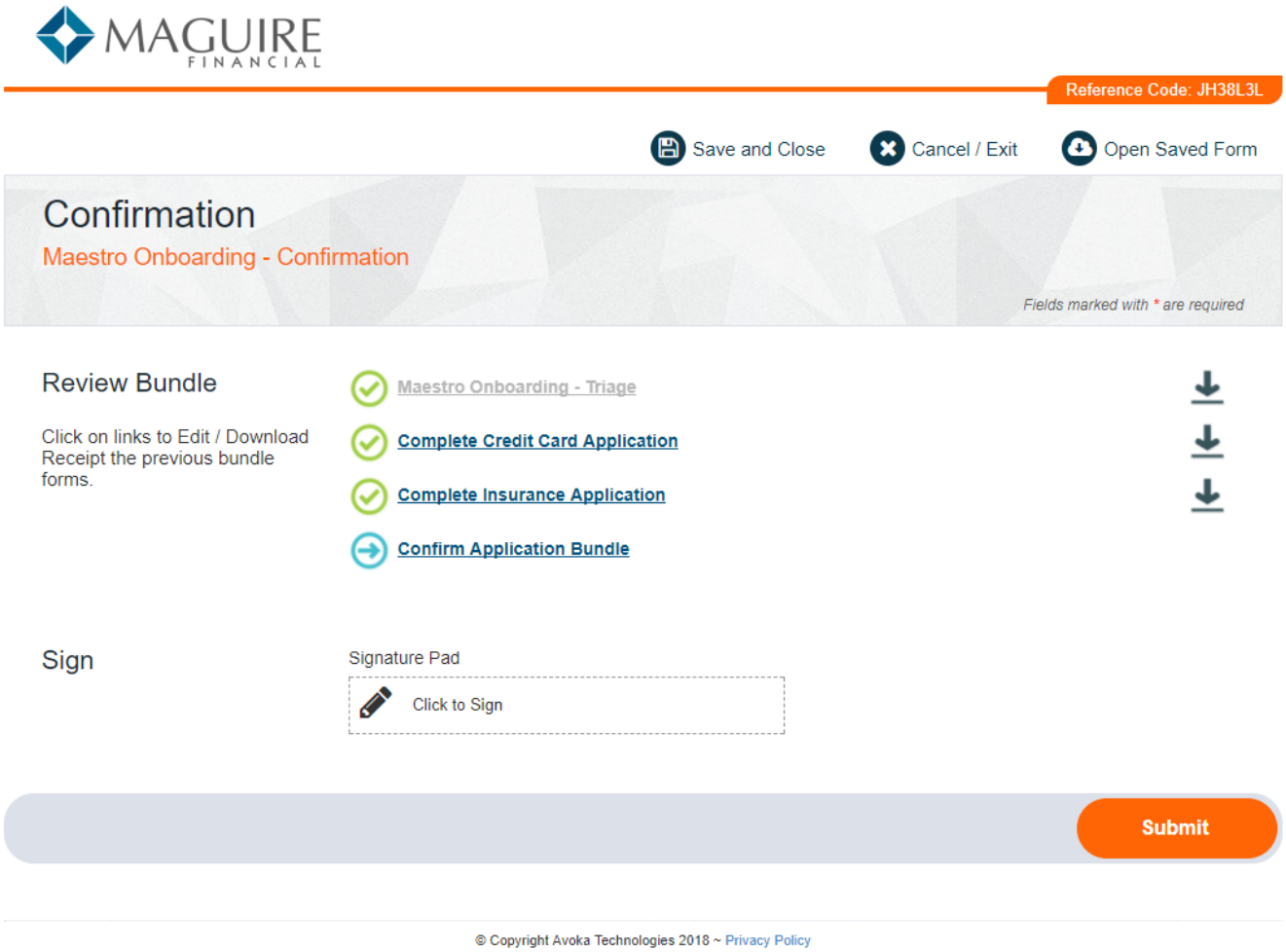
The browser redirects next to the Insurance Page. This is because the order of the Insurance is above the confirmation in the Job Action List

Clear the Product Options - Credit Card, Check the Home Contents and Insurance then submit



The browser is redirected to the confirmation page, which shows the list of the forms and the insurances.

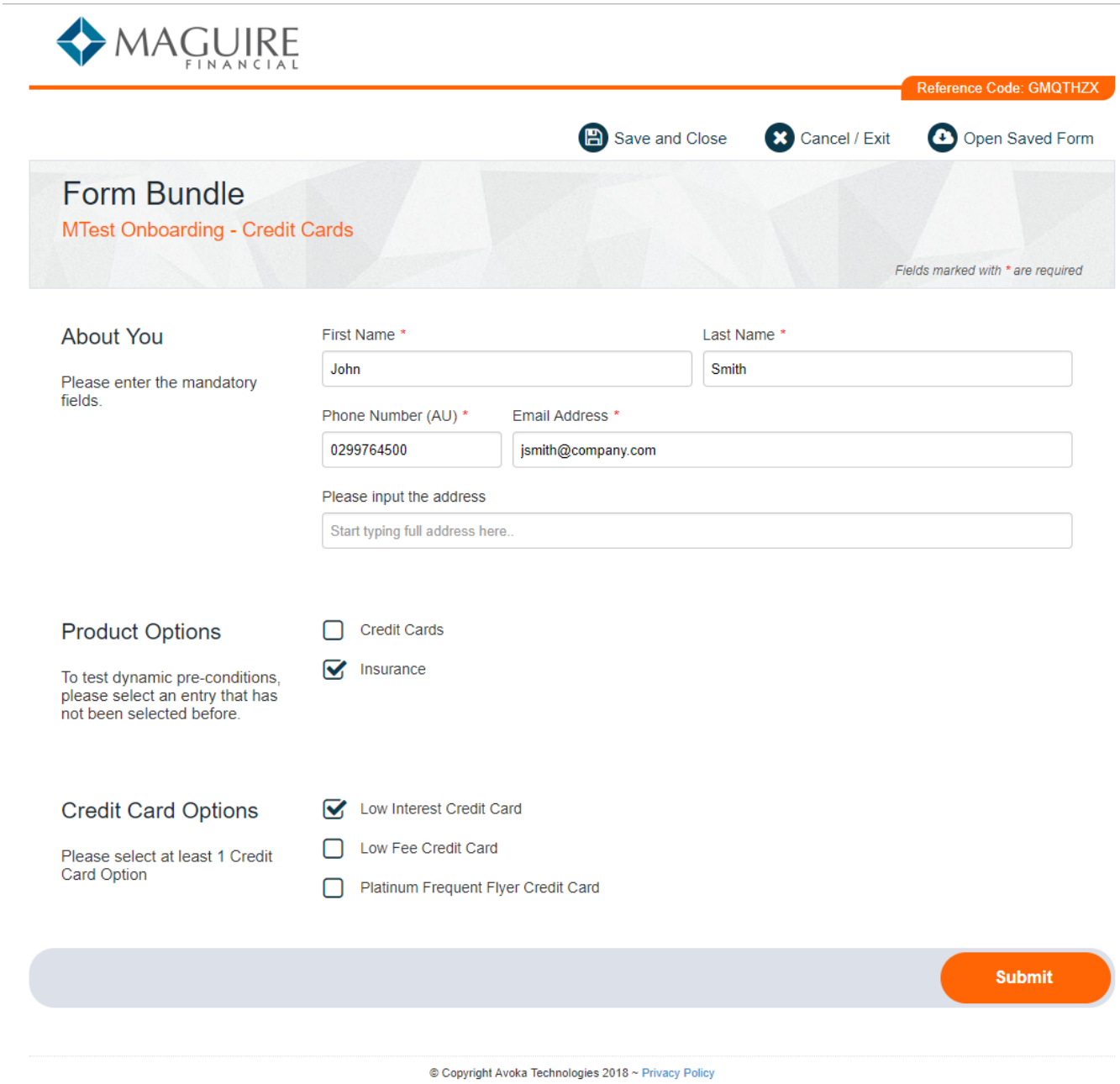
Click **Complete Credit Card Application**.



The browser is redirected to the Credit Card form. The Credit Card product option is cleared. The is done by the **`shareFormData`** or **`shareExtractData`**.

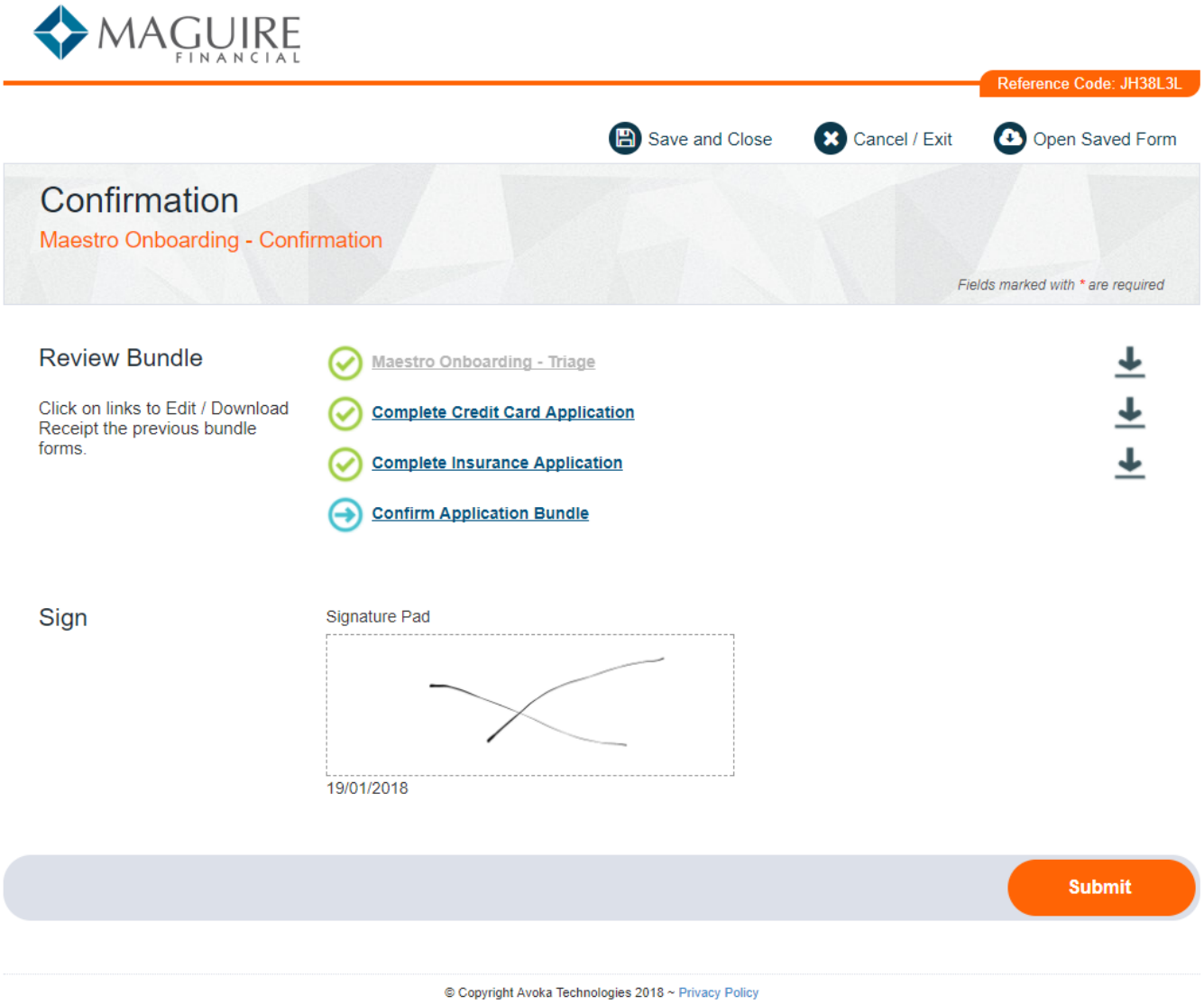
The credit card form isn't removed. The precondition is only evaluated once. A form user can click **Cancel** to remove them.

Click **Submit**



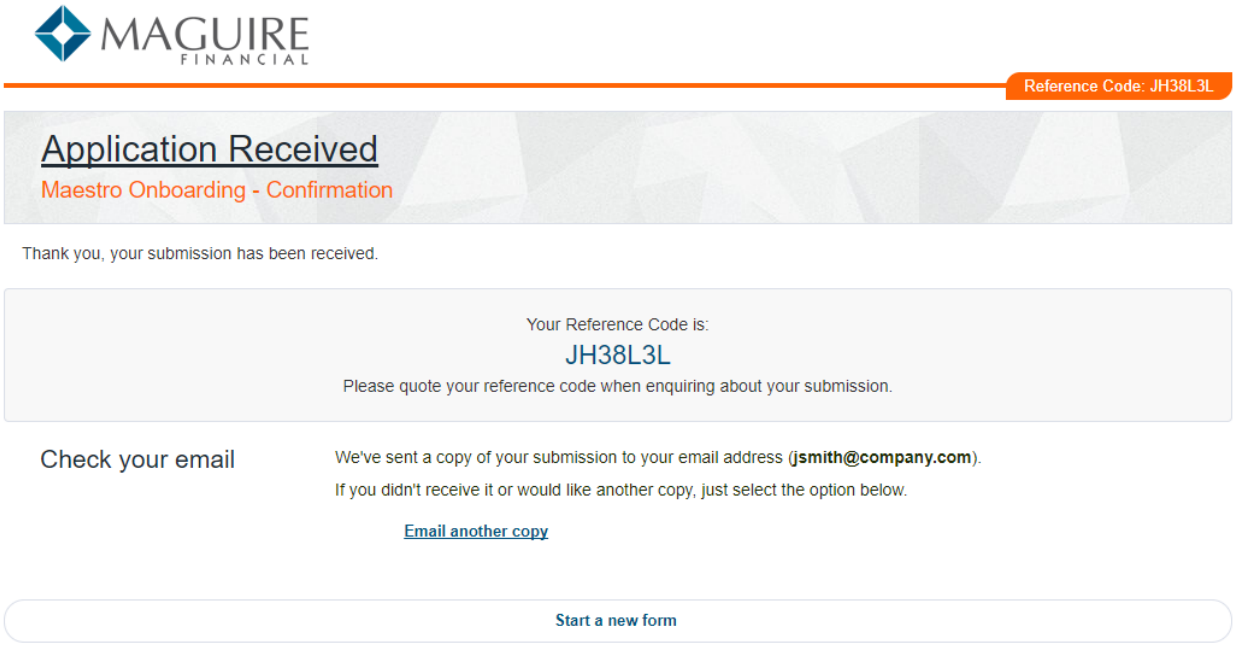
The browser is redirected back to the Confirmation page

Sign the form and click **Submit**.



The form goes to the **Application Received** page, which informs the user that a receipt will be sent via email.

This is standard form processing where form chaining was turned off because the **`redirectNext`** attribute is not set on the Maestro Onboarding - confirmation page.



The form flow above can be bundled up to make it look like it's one form by applying the following minor changes:

- Change the submit button label to **Next** on all but the final **Application Confirmation** form.
- Add a back button to go to the previous form.
- Use the **Custom Navigator Maestro Widgets** that allow a user to easily jump to any part of the form.

This example uses the following forms, which are part of the standard Maestro Onboarding example:

- Maestro Onboarding - Confirmation
- Maestro Onboarding - Credit Cards
- Maestro Onboarding - Insurance
- Maestro Onboarding - Triage

These are two forms that substitute the equivalent standard form:

- MTest Onboarding - Credit Cards
- MTest Onboarding - Insurance

These are used, instead of the standard triage form, to start authenticated bundle tests of type Form and Review:

- MTest Onboarding - Triage Form
- MTest Onboarding - Triage Review

The example uses the following services:

- Maestro Onboarding - v17.10.0 - The standard job controller, Anonymous with shareFormData, Not Dynamic
- Maestro Onboard-Anony-SharedExtractData - v17.10.0 - Anonymous with shareExtractData, Not Dynamic
- mOnboarding - Anonymous - Dynamic - shareExtractData - v17.10 - Anonymous with shareExtractData, Dynamic
- mOnboarding - Anonymous - Dynamic - shareFormData - v17.10.0 - Anonymous with shareFormData, Not Dynamic
- mOnboarding - Anonymous - No Dynamic - v17.10.0 - Anonymous No sharing, Not Dynamic
- mOnboarding - Form Tasks - shareExtractData - v17.10.0 - Authenticated Form Task, shareExtractData, Dynamic
- mOnboarding - Form Tasks - shareFormData - v17.10.0 - Authenticated Form Task, shareFormData, Dynamic
- mOnboarding - Review Task - shareExtractData - v17.10.0 - Authenticated Review Task, shareExtractData, Dynamic

You can download the complete code example here.

Next, learn about authenticated form bundles

Form Chaining Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

The type of UI flow mentioned in form bundles is often referred to as form chaining. Its possible to configure a collaboration job to have one task created per one step and still implement chaining between the steps. The **redirectNext** is both a step and an action attribute. If you want to the form chaining to be used within steps, set the step attribute **redirectNext** to **true**. To allow the form chaining from one action to another, set the action attribute **redirectNext** to **true**.

Let's look at the customer onboarding collaboration job. The job definition's additional products step configuration is shown below.

```
{
  "name": "Additional Products",
  "type": "",
  "shareExtractData": true,
  "allFormsEditable": true,
  "showPreviousForms": true,
  "redirectNext": true,
  "actions": [
    {
      "name": "Credit Cards Application",
      "type": "Job Task Assign",
      "preCondition": "$formDataMap.productCreditCards == 'true'",
      "redirectNext": true,
      "properties": [
        { "name": "Task Assign Email",  "value": "$formDataMap.emailAddress" },
        { "name": "Task Form Code",  "value": "onboard-credit-cards" },
        { "name": "Task Message",  "value": "Please complete the Credit Card Application
form." },
        { "name": "Task Subject",  "value": "Complete Credit Card Application" },
        { "name": "Task Input XML Prefill",  "value": "$func.startSubmissionXml()" },
        { "name": "Task Type",  "value": "Anonymous" }
      ]
    },
    {
      "name": "Insurance Application",
      "type": "Job Task Assign",
      "preCondition": "$formDataMap.productInsurance == 'true'",
      "redirectNext": true,
      "properties": [
        { "name": "Task Assign Email",  "value": "$formDataMap.emailAddress" },
        { "name": "Task Form Code",  "value": "onboard-insurance" },
        { "name": "Task Message",  "value": "Please complete the Insurance Application
form." },
        { "name": "Task Subject",  "value": "Complete Insurance Application" },
        { "name": "Task Input XML Prefill",  "value": "$func.startSubmissionXml()" },
        { "name": "Task Type",  "value": "Anonymous" }
      ]
    },
    {
      "name": "Appliction Confirmation",
      "type": "Job Task Assign",
```

```
      "preCondition": "$formDataMap.productInsurance == 'true' || $formDataMap.-
productCreditCards == 'true'",
      "properties": [
        { "name": "Task Assign Email",  "value": "$formDataMap.emailAddress" },
        { "name": "Task Form Code",  "value": "onboard-confirmation" },
        { "name": "Task Message",  "value": "Please sign to complete Application bundle."
},
        { "name": "Task Subject",  "value": "Confirm Application Bundle" },
        { "name": "Task Input XML Prefill",  "value": "$func.startSubmissionXml()" },
        { "name": "Task Type",  "value": "Anonymous" }
      ]
    },
    {
      "name": "Review Wait",
      "type": "Job Task Wait"
    }
  ],
  "routes": [
    { "name": "Default",  "nextStep": "Application Delivery" }
  ]
},
```

Here, you can see the steps has "redirectNext": true. The first 2 Task Actions,Credit Card Application and Insurance Application Forms, have redirect next "redirectNext": true. The last Action Application Confirmation does not have a redirectNext attribute or "redirectNext": false. The Triage form is filled in, depending upon the user selection in the triage form and the either or both Credit Card and Insurance preCondition will evaluate to true.

Lets assume that the both products selected. The user submits the triage form it chains to the Credit Card. The order for the next available form is determined by the order of the Action in the Step that is Triage Form > Credit Card > Insurance > Confirmation. The form chaining continues to the confirmation page. However at the confirmation page it breaks the chaining flow and goes to the Application Confirmation form's - Submission Confirmation page. At this page the user can download a combined PDF receipt.

Change of User

Form chaining can happen only when the current user has access to newly created tasks:

1. Anonymous Forms the email address needs to stay the same.
2. Authenticated task either:
   - the current user is assigned to the new task
   - the task is assigned to a group that the current user belongs to
   - group tasks are claimable are also required to be claimed by the current user.

When there is a change of assignment that does not meet the conditions above, additional tasks cannont be openned by the current user. The current form stops chaining and goes to the forms submission confirmation page.

Legacy Mode

There is an legacy UI flow that was in earlier revisions of form bundles 4.2 did not load the next available form, that is did not perform form chaining.

When a user submits the submission the user is then presented with a list of forms on the forms submission confirmation page, one for each form that they were to completed.

> We recommend to use the form chaining because it provides a better user experience.

Next, learn about a form bundle.

Transact Functions Overview

Journey Manager (JM) | Form Builder Platform Developer | 17.10

Journey Manager allows developers to create services as Transact functions, which can be called from Maestro forms.

Consider calling Transact functions from a Maestro form to:

- Use the rich Transact Fluent API, as opposed to Java script calls.
- Implement complex business logic that requires extensive coding, so Maestro forms can only use simple JavaScripts.
- Make code more secure because it resides on the server side, not on the client side, which is a Maestro form.
- Provide better integration with back end, such as database, because it resides on the server side.

Manager defines the following Transact functions:

- Fluent Function (aka Form Function)
- Delivery Function

Both Fluent and Delivery Transact functions are Fluent Functions because they implement **`com.avoka.fc.core.service.fluent.FluentFunctionService`** Fluent Service, see Transact Fluent API reference.

Fluent Functions

Fluent Functions, also known as Form Functions, provide an improved programming model for client applications to call server-side business functions. Form Functions are designed for applications to make call to other system for doing remote data lookup or executing some secure business logic on the server. They provide the replacement for Form Dynamic Data Services with an improved programming and security model. Fluent Functions provide server-side function event handlers for a variety of Form event triggers. The Fluent Function programming interface is the same across a variety of function triggers enabling you to re-uses function code in a variety of contexts. For example, you could a single function for doing both data pre-fill when a form is first opened and when a saved form is resumed.

Delivery Functions

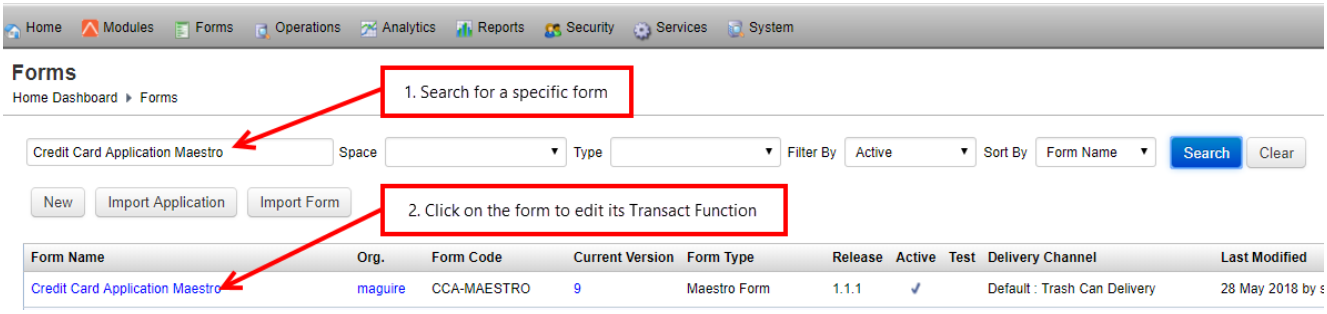
Delivery Functions are used to deliver user entered form data when a transaction is submitted. This form data includes the submitted form XML, the PDF receipt document and many also include user uploaded file attachments.

How It Works

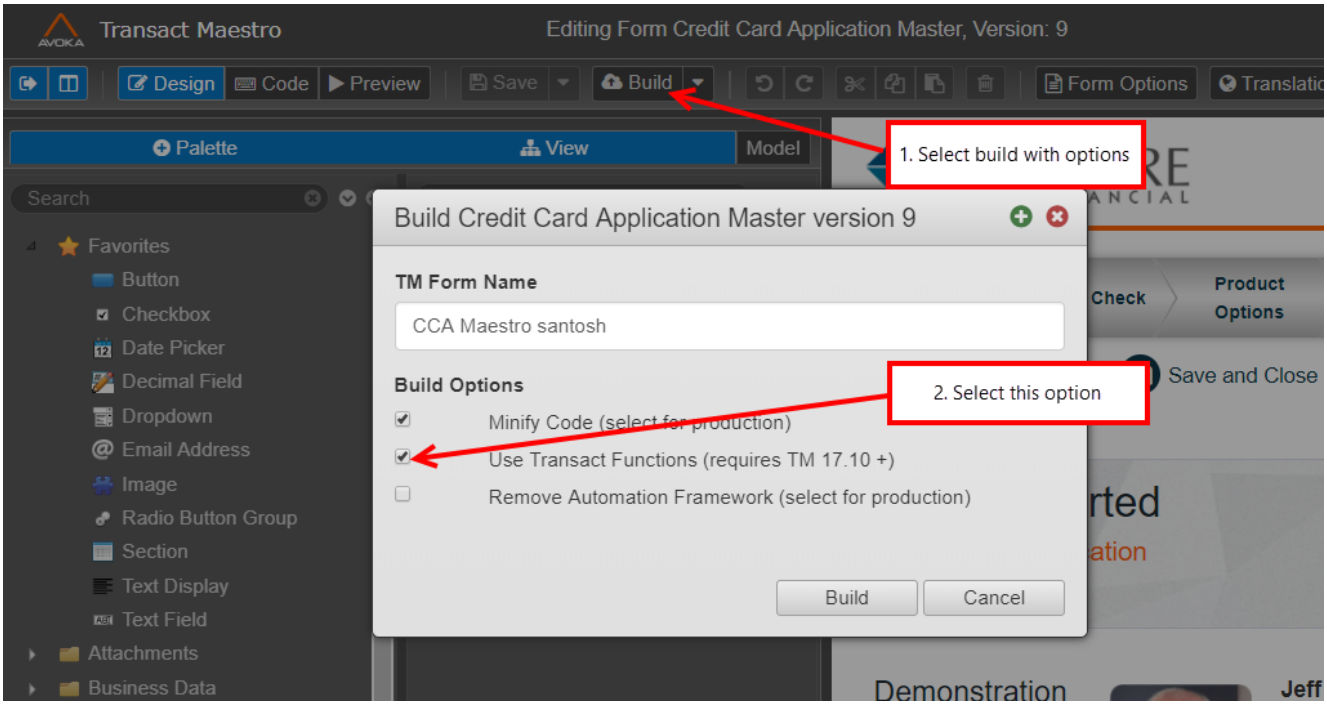
Transact functions are configured in Manager and they can be used from Maestro forms.

To update a Transact function:

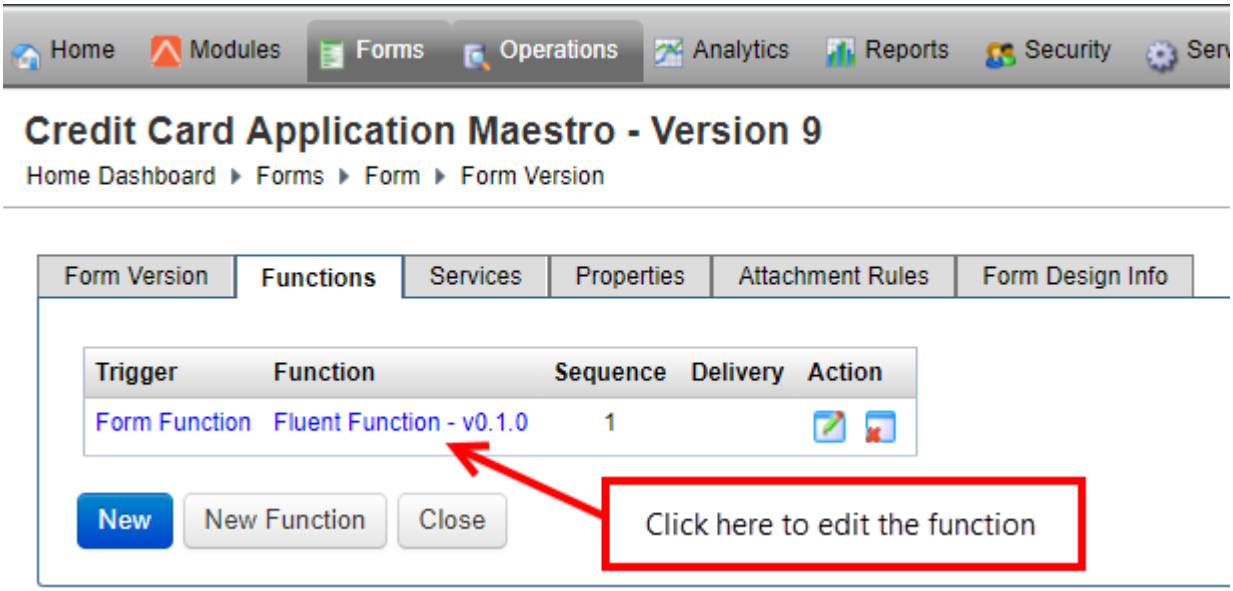
1. Select **Forms > Forms** and search for a form.



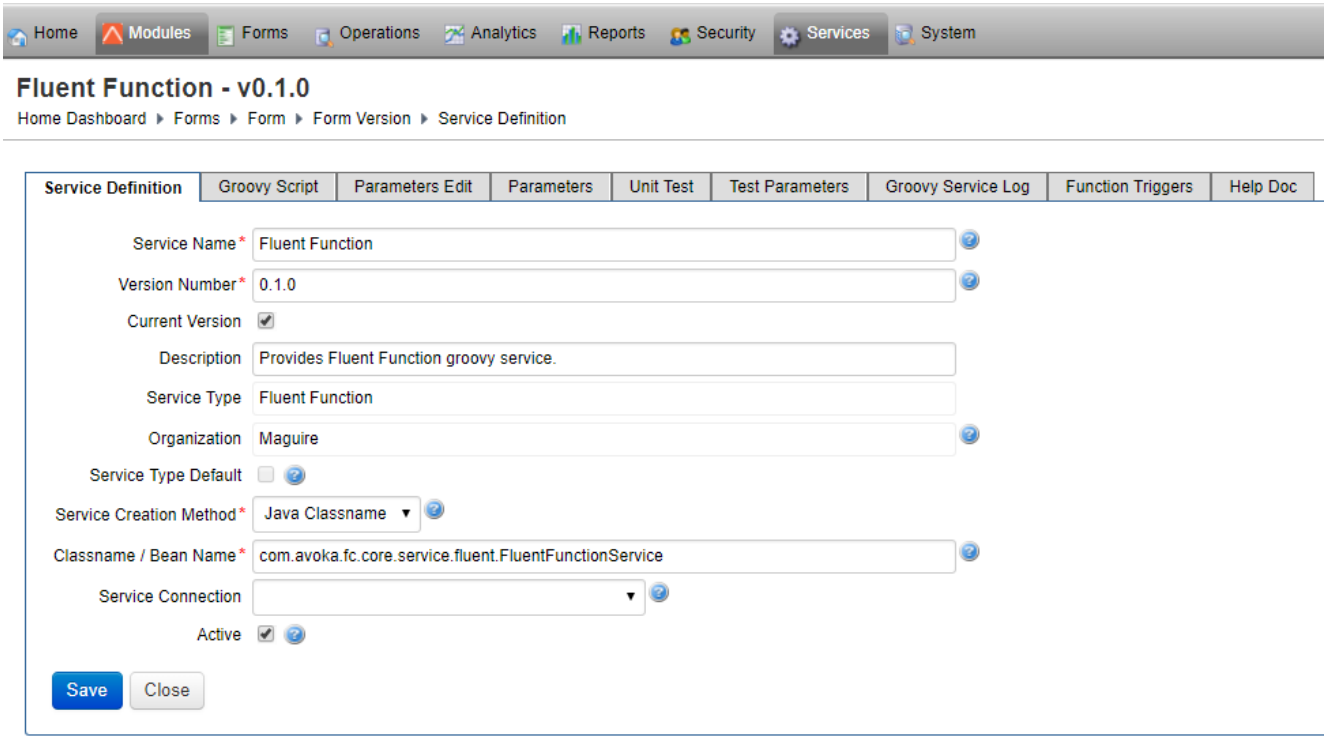
2. Click the **Functions** link to configure a Transact function.
3. If the form does not have a function associated to it, you need to re-build it in Journey Maestro with the **Use Transact Functions** option.



4. Refresh your form view in Manager. You should see the Functions link available now.
5. Click the link to continue the configuration. You will be presented with the Form Versions tab from where you can configure the Transact function.



6. Complete the Transact function configuration.



Next, learn how to view Transact functions.

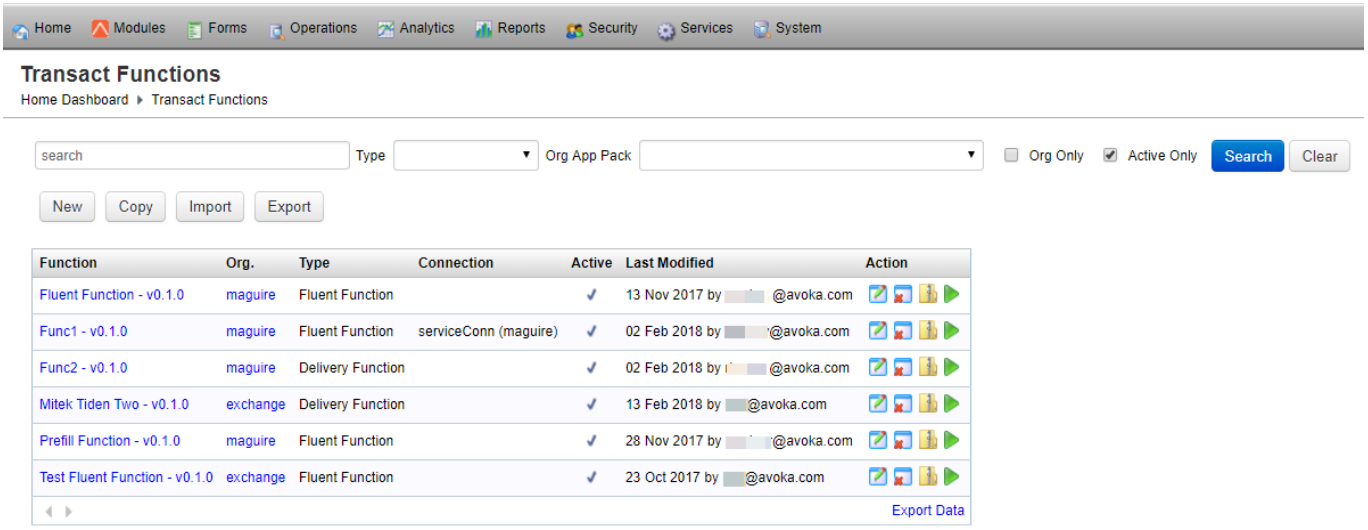
View Transact Functions

Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to view, search, create, copy, configure, and remove Transact functions. You can use this functionality to manage the Transact functions life cycle.

To view all Transact functions, select **Services > Transact Functions.**

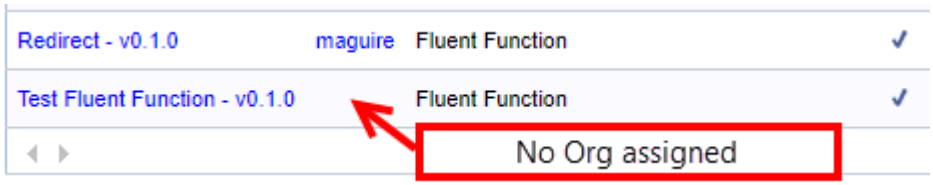
Transact functions are shown page by page, with the maximum number of records per page configured in the users preferences.



Searching and filtering of Transact functions become important when there are many functions created in Manager.

To filter or search Transact functions by one or more criteria, specify the following settings and click **Search**:

- Name or part of the name of the Transact function.
- Type of a Transact function.
  - ATransact function type can be one of the following:
    - Delivery Function
    - Fluent Function
- Org App Pack to filter functions by organization application pack.
- Org Only. Select this checkbox to list Transact functions that have their organizations configured. Otherwise, everything is listed.



- Active Only to list Transact functions that are active.

> Do not remove a Transact function, make it inactive instead. You can also use this setting in the Transact function rollback process.

Click **Clear** to reset the search criteria.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

> You can list all Fluent services using a Groovy script and run it in the Groovy console.

You can also import and export Transact functions.

Create a Transact Function

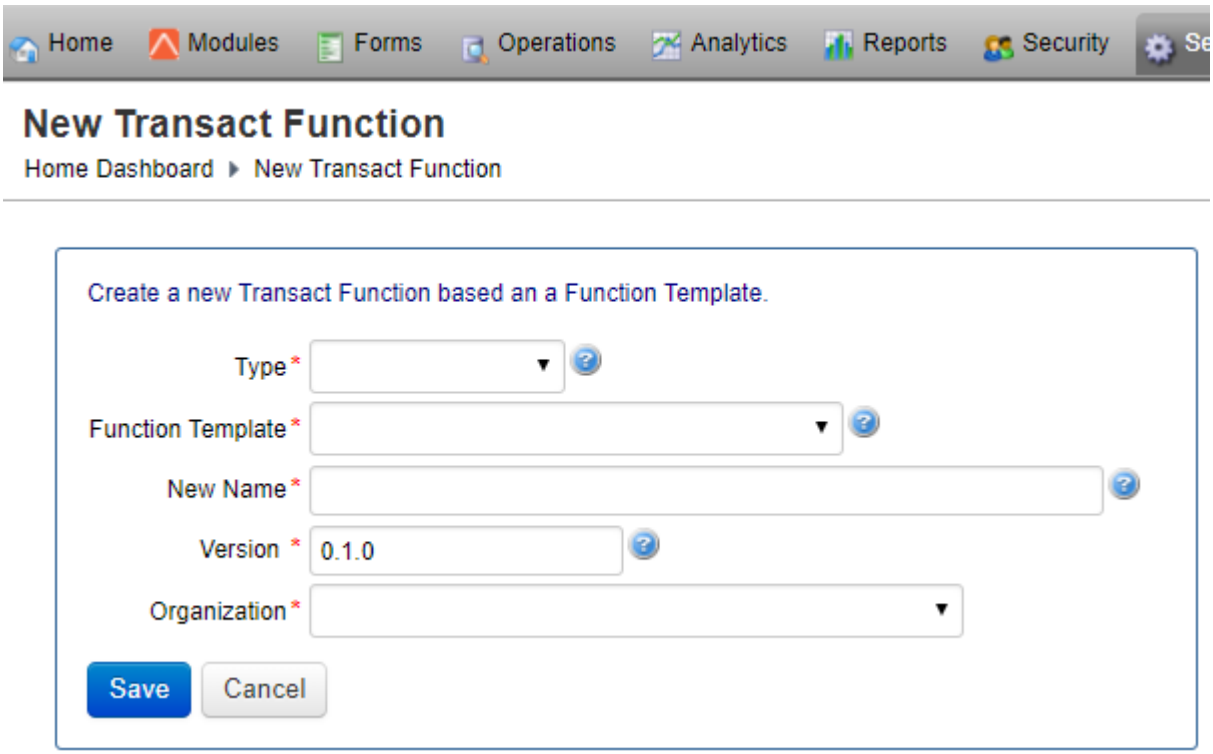
Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to create a new Transact function to:

- Implement a new customer's requirement.
- Implement a new version of the existing business logic.

To create a new Transact function:

1. Select **Services > Transact Functions** and click **New**.
2. Select the **Type** of the Transact function from the following options:
   - Fluent Function
   - Delivery Function



3. Select the **Function Template** from the following options:
   - Fluent Function
   - Delivery Function
4. Provide a unique function name in the **New Name** field.
5. Specify the function version in the **Version** field.
6. Select the **Organization** from the dropdown menu or leave it empty.
7. Click **Save** to update the changes.

Next, learn how configure the new Transact function .

Copy a Transact Function

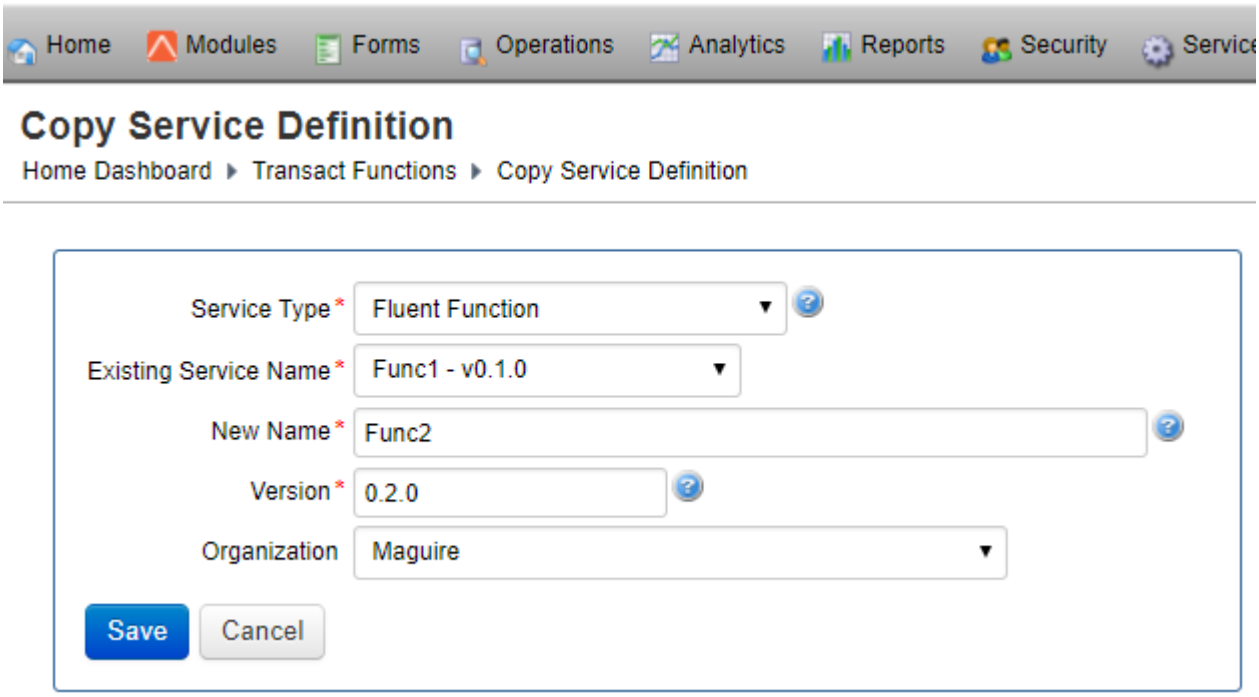
Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to copy a Transact function. You can use this functionality to:

- Create a new Transact function as a copy of another Transact function, which implements a different business logic.
- Create a new version of the existing Transact function.

To copy a Transact function:

1. Select **Services > Transact Functions** and click **Copy**.



2. Select a Transact function from the **Service Type** dropdown list:
   - Fluent Function
   - Delivery Function
3. Select a function name you want to copy from the **Existing Service Name** dropdown list, for example **Func1 - v0.1.0**.
4. Provide a unique function name in the **New Name** field.
5. Specify the function version in the **Version** field.
6. Select the organization from the **Organization** dropdown menu or leave it blank.
7. Click **Save** to update the changes.

Next, learn how to configure a Transact function.

Import a Transact Function

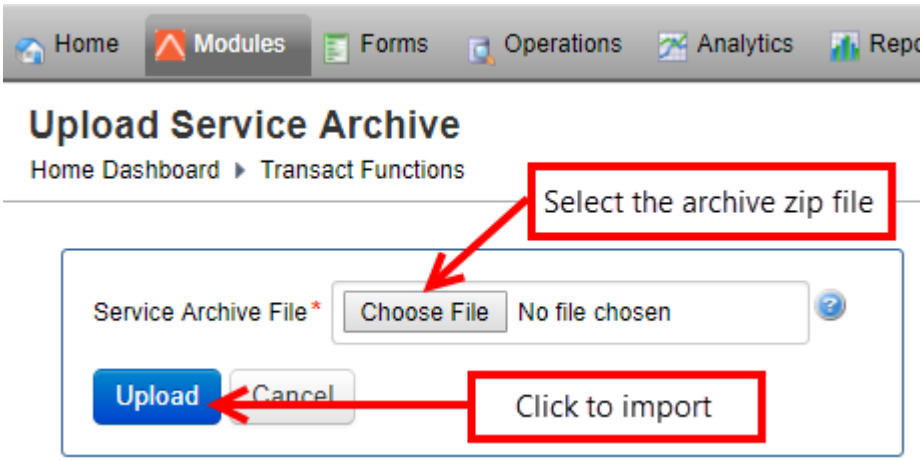
Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to import Transact functions. Use this functionality to:

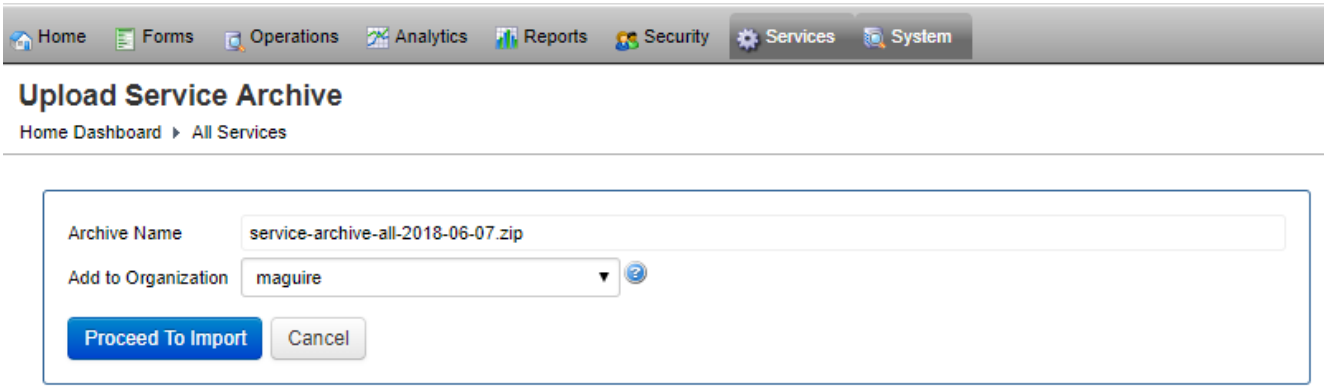
- Rollback Transact functions.
- Re-create or promote Transact functions in another environment.

To import one or several Transact functions:

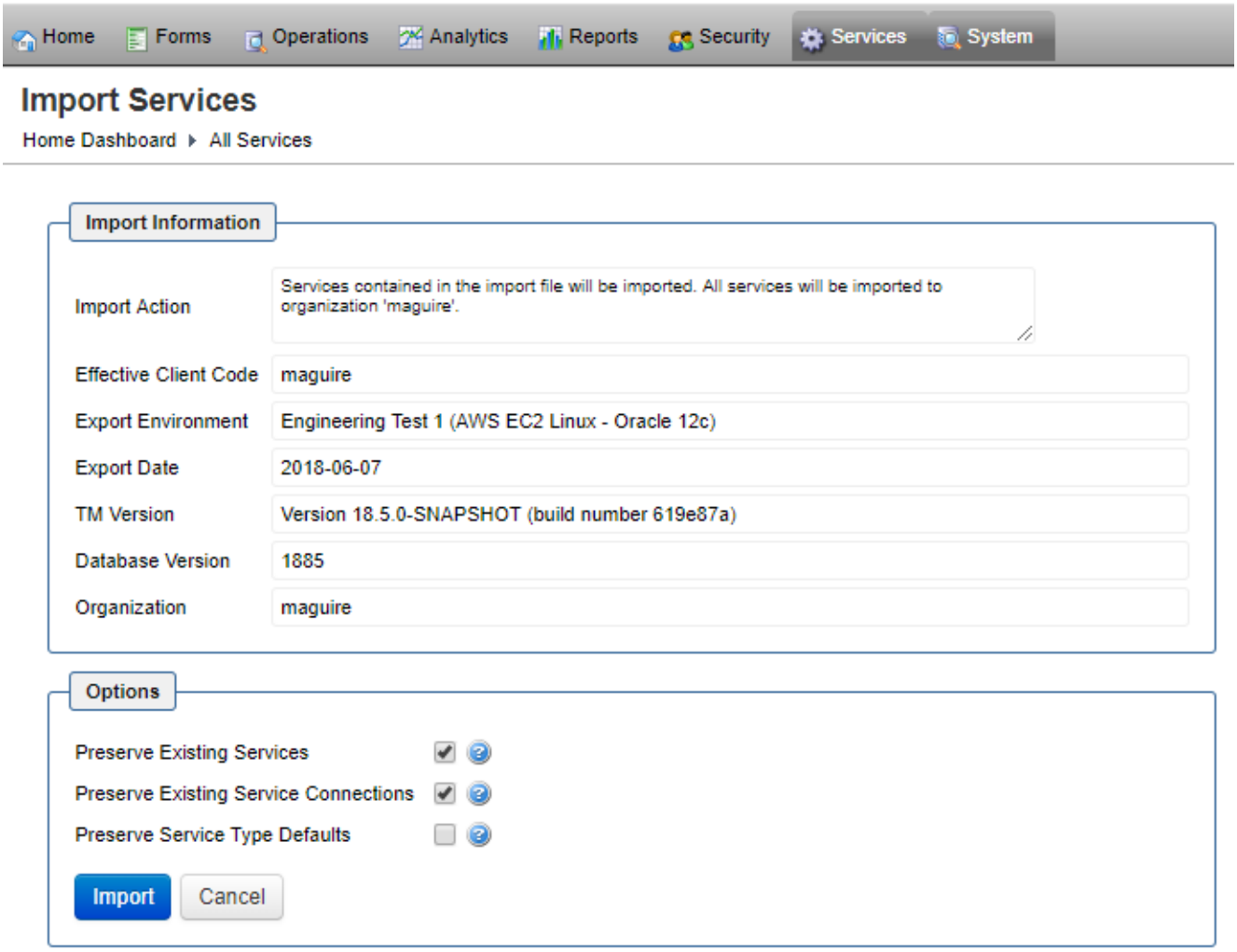
1. Select **Services > Transact Functions** and click **Import**.



2. Click **Choose File** and choose a service archive ZIP file, which has been exported, for example, service-archive-all-2018-06-07.zip.
3. Click **Upload**.



4. Select an organization that you want to add the services to or use the organization already configured in the import archive file.
5. Click **Proceed To Import** to review the Import Options screen, as shown below.



6. Select the **Preserve Existing Services** checkbox to preserver any services that already exist on the target system, so they will not be modified in any way. Only new services will be imported. Note that services are resolved by name. (Default.)
7. Clear the **Preserve Existing Services** option on import if you are importing it as a rollback.
8. Select the **Preserve Existing Service Connections** checkbox to preserve any service connections that already exist on the target system, so they will not be modified in any way. New service connections will still be created. Note that service connections are resolved by name. This option is selected by default.
9. Select the **Preserve Service Type Defaults** checkbox to preserve types of the imported services, so they will be promoted to be the default for their service types, regardless of the default flags on the source system. If not selected, the service type default flag will be imported, and the existing services may be demoted if needed. This isn't selected by default.
10. Click **Import** to complete the import.
11. Click **Close** and check the list of Transact functions to verify the import was successful.

Next, learn how to create a Transact function.

Export a Transact Function

Journey Manager (JM) | Form Builder Platform Developer | 17.10

Manager allows you to export Transact functions. Use this functionality to:

- Backup or archive Transact functions.
- Migrate or promote Transact functions in another environment.

To export one or several Transact functions:

1. Select **Services > Transact Functions**.
2. Click **Export Archive** to export one Transact function or click **Export** to export all Transact functions.

Manager exports them into a service archive ZIP file, which will be downloaded to your PC. The ZIP file contains service-config.xml file with all Transact functions' definitions.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to import Transact functions.

Backup a Transact Function

Journey Manager (JM) | Form Builder Platform Developer | 17.10

It's recommended to back up Transact functions before you update them in Manager. This allows you to:

- Restore a Manager environment after a configuration change was rolled back.
- Re-created Transact functions in another environment.
- Promote or migrate Transact functions into a new environment.

Next, learn how to import Transact functions.

Rollback a Transact Function

Journey Manager (JM) | Form Builder Platform Developer | 17.10

The purpose of a rollback is to restore the target environment to its former state and should only be considered in a situation where the verification tests have determined that the promotion has encountered a significant failure of some kind.

Note that a rollback is not a decision that should be taken lightly, as it can require a lengthy and detailed process to restore components back to their previous state. Often it is preferable to adopt a fix-forward and rectify issues in situ if possible.

If a Transact function has been changed and it does not work as expected anymore, you can try to roll it back to the previous working version. This is only possible if:

- There are backups of previous Transact functions.
- There are records of changes made to the Transact function, known as the **changelog**[1].

To rollback a Transact function:

1. Locate the previous Transact function backup, which is an exported ZIP archive file.
2. Import it into Manager.
3. Verify that it works as expected.
4. Update your changelog.

Next, learn how to backup Transact functions.

[1] is a record of changes to a software project or other technology product. This type of record has its own format for various types of projects, but is generally useful in showing when and why changes were made, and who made them.

Remove a Transact Function

Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to remove a Transact function. You may want to do this if:

- It is incorrectly configured
- It is not required any more

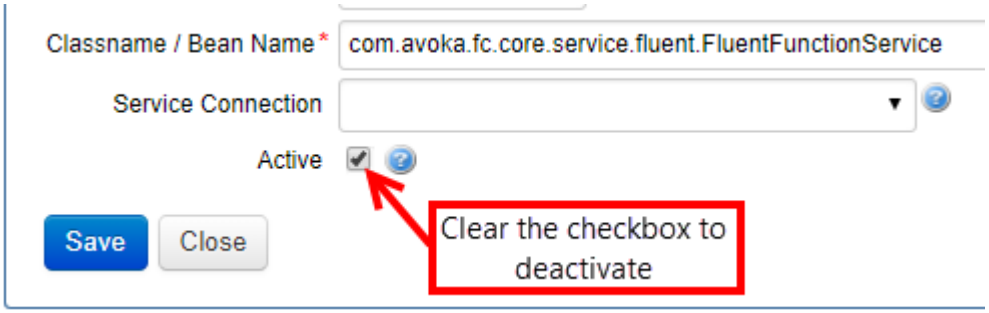
> It's recommended to deactivate a function rather than remove it, so you can use it as a rollback.

To remove a Transact function:

1. Select **Services > Transact Functions**.
2. Locate a function and click **Remove**.
3. Click **OK** to accept it.

To deactivate a Transact function:

1. Select **Services > Transact Functions**.
2. Locate a function and click **Edit**.
3. Clear the **Active** checkbox.



4. Click **Save** to update the changes.

Next, learn how to create a Transact function.

Run a Transact Function Unit Test

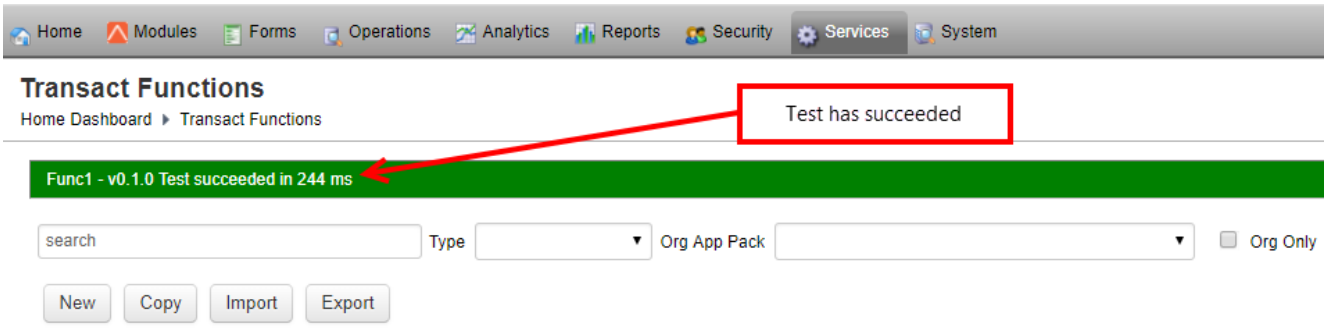
Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager provides the convenient way to test a Transact function by executing a **Unit test**[1] suite, which you must configure for each Transact function.

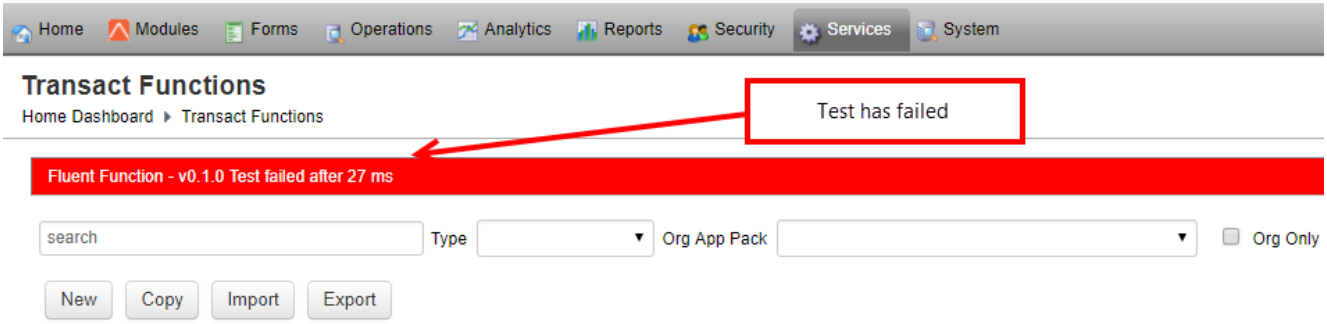
You can run a Unit test from the list of Transact functions or from the Transact function **Unit Test** tab.

To run a Unit test from the list of Transact functions:

1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Run Unit Test**.
3. Check the test result displayed at the top of the page. The successful test is shown below.



4. If the test has an error, as in the example below, re-run it from the **Unit Test** tab, see below.

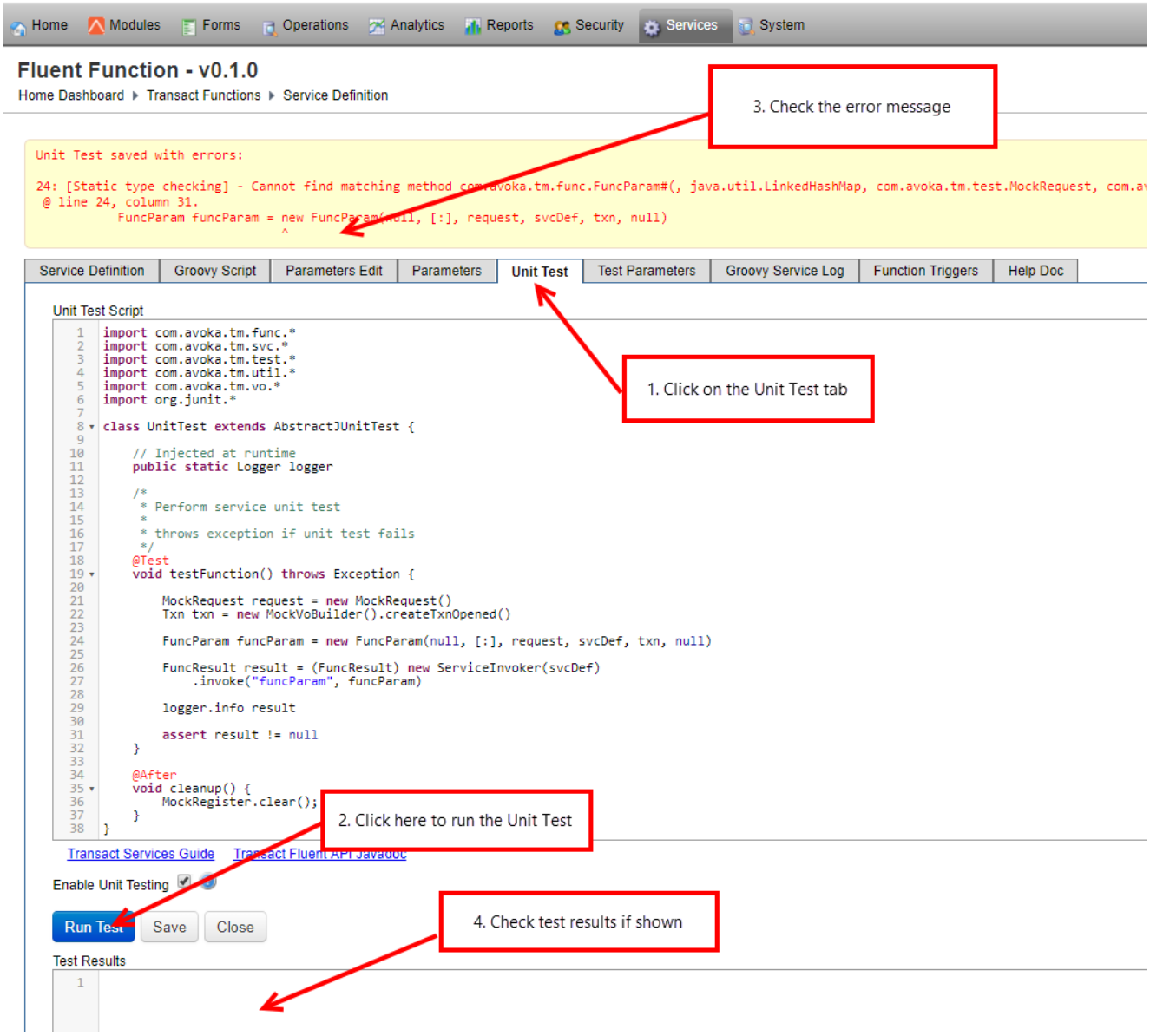


To run a unit test from the **Unit Test** tab:

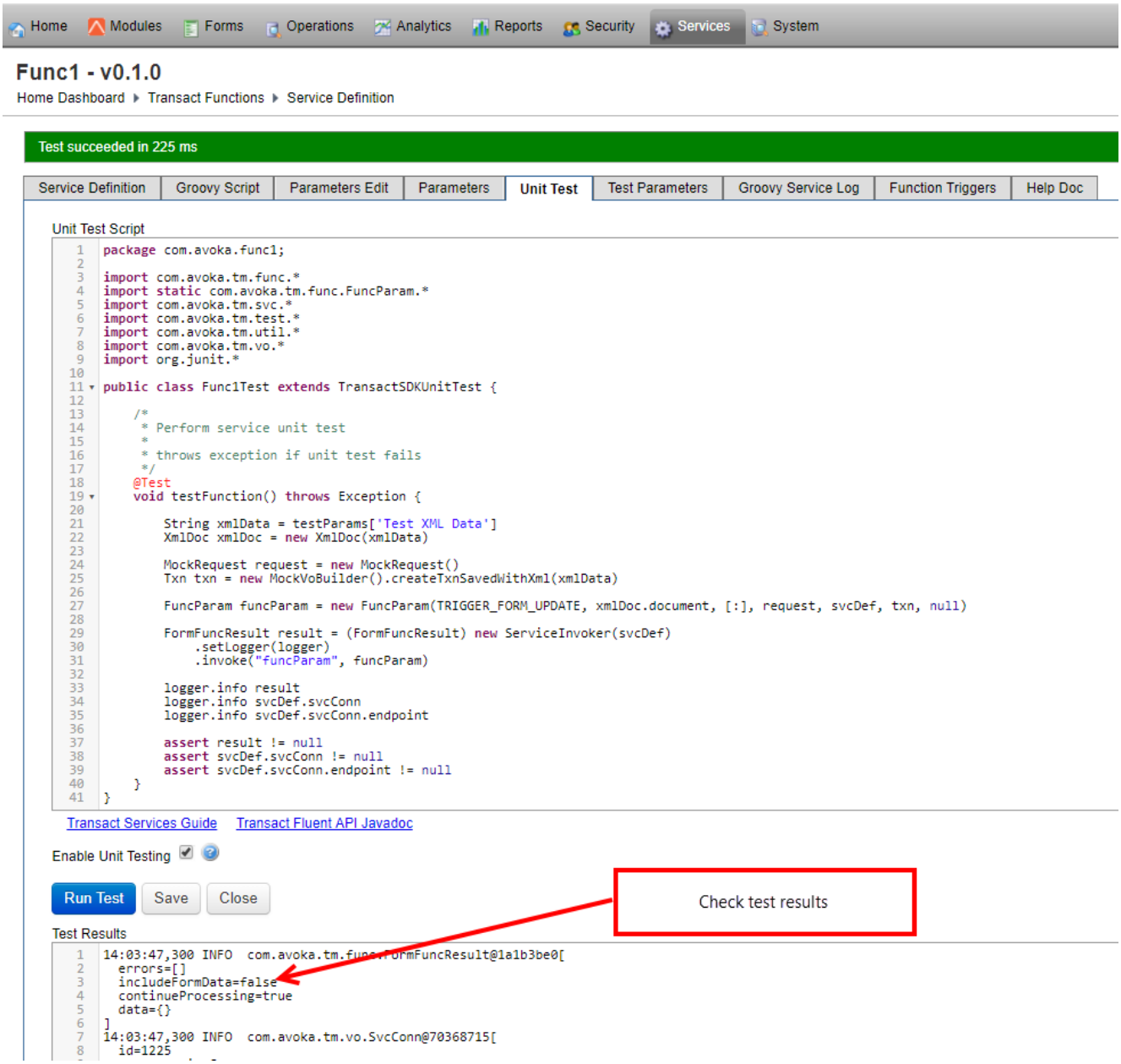
1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Select the **Unit Test** tab and click **Run Test**.

---

[1] is a software development life cycle (SDLC) component in which a comprehensive testing procedure is individually applied to the smallest parts of a software program for fitness or desired operation. Unit Testing is a software testing method by which individual units of source code are tested to determine whether they are fit for use. For more information, see http://docs.groovy-lang.org/next/html/documentation/core-testing-guide.html.



4. Review the test results or error messages. If there are errors, correct the Transact function Groovy script or Unit test Groovy script and re-run the Unit test to see if it works. The successful test message is shown below.



5. Check the event log to see the script's execution output.

Next, learn how to configure Unit test parameters.

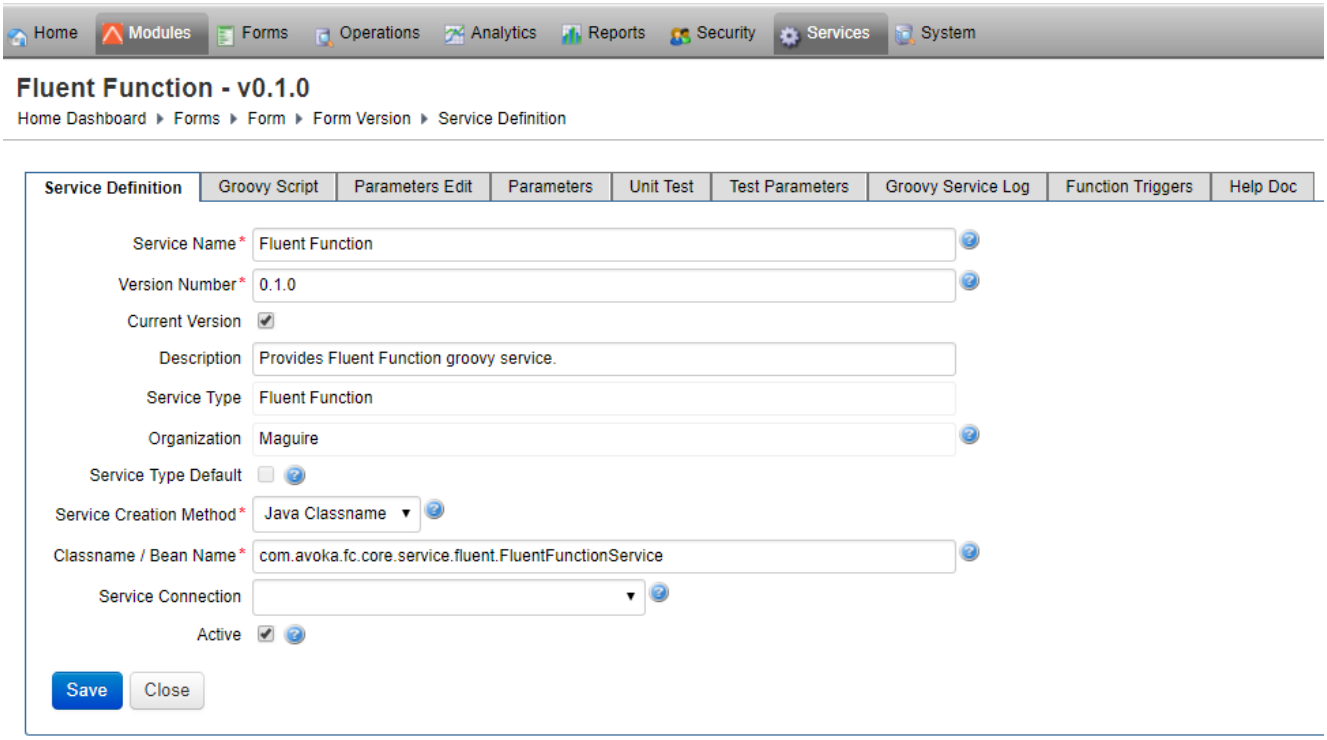
Configure Transact Function Service Definition

Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to configure a Transact function's main parameters in the **Service Definition** tab.

To configure the Transact function service definition:

1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Select the **Service Definition** tab.

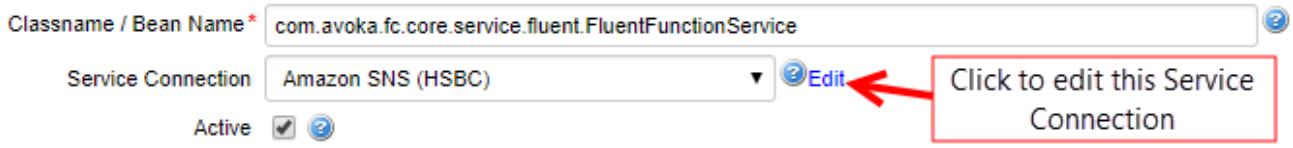


4. Enter a new Transact function name in the **Service Name** filed. This name combined with the **Version Number** must be unique.
5. Enter a Transact function version number in the Version Number filed. Version number must adhere to **semantic versioning**[1] practices.
6. Select the **Current Version** checkbox to allow Manager to use this function in a form.

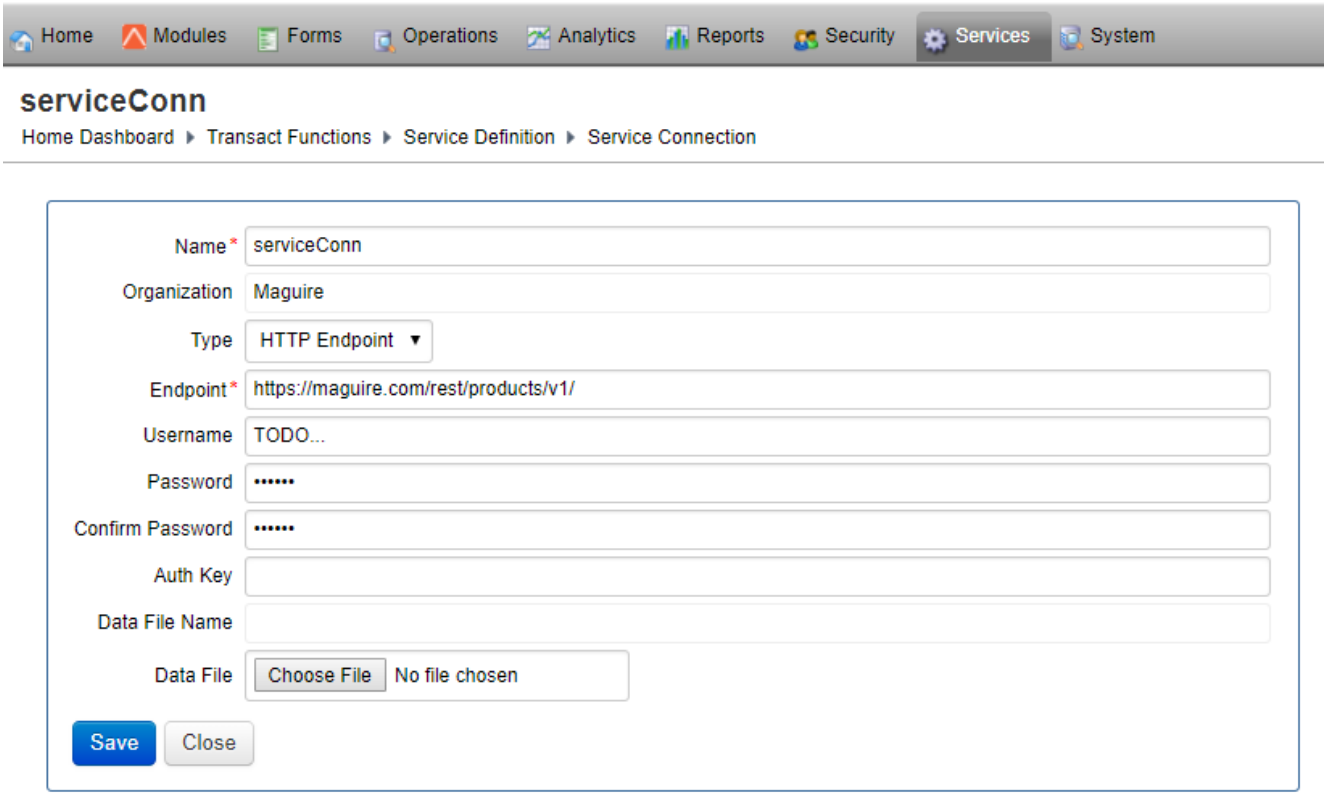
[1]Semantic Versioning is a formal convention for specifying compatibility using a three-part version number: major version; minor version; and patch. The patch number is incremented for minor changes and bug fixes which do not change the software's application programming interface (API). The minor version is incremented for releases which add new, but backward-compatible, API features, and the major version is incremented for API changes which are not backward-compatible. For example, software which relies on version 2.1.5 of an API is compatible with version 2.2.3, but not necessarily with 3.2.4.

You can use this option to rollback this Transact function.

7. Provide a meaningful description of what this Transact function does and why in the **Description** field.
8. Select the Transact function template type from the **Service Type** dropdown list as shown below.
   - Fluent Function.
   - Delivery Function.
9. Specify the organization, which this function will be available to, in the **Organization** field. If the organization is not set, this function is globally available, so it can be used in any Organization.
10. Select the **Service Type Default** checkbox to use this Transact function as the implementation of this Service Type. Only global services and Transact functions can be marked as the default.
11. Select the invocation method to create this Transact function from the **Service Creation Method** dropdown menu:
    - Java Classname to create a new instantiate of this Class it as a new Java Object using Class.forName() method.
    - Spring Bean to look it up from the Sprint configuration files.
12. Enter the Java Class name, for example, com.avoka.fc.core.service.fluent.FluentFunctionService, or the Spring Bean name, which is used to create this function, in the Classname / Bean Name field.
13. Select a service connection that this Transact function should use or leave it blank if it is not required.
14. Click **Edit** to go to this service connection.



15. View or edit the service connection if needed.



16. Select the **Active** checkbox to make this Transact function active so it can be used in Manager. It is selected by default.

> You can use this checkbox option to rollback this Transact function.

17. Specify the minimum Manager version, which is required to run this service, in the **TM Min Version** field.



18. Click **Save** to update the changes.

Next, learn how to edit Transact function Groovy script.

Edit a Transact Function Groovy Script

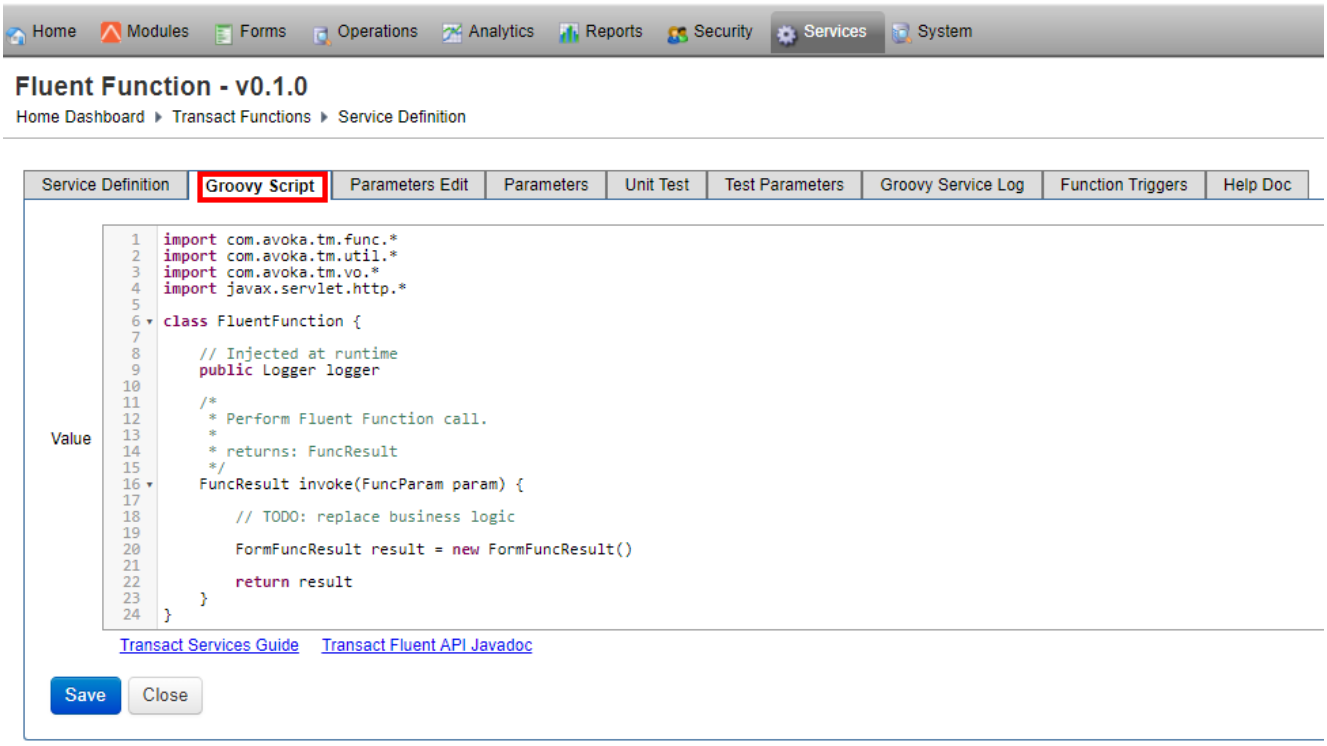
Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to edit a Transact function Groovy script, which implements the desired business logic. When you create a new Transact function, it comes with the default Groovy script, which is merely a template to help you implement new business logic, so you must modify it, in most cases.

> You can test your Groovy script or a part of the script using the Groovy console.

To edit the Groovy script:

1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Click the **Groovy Script** tab.



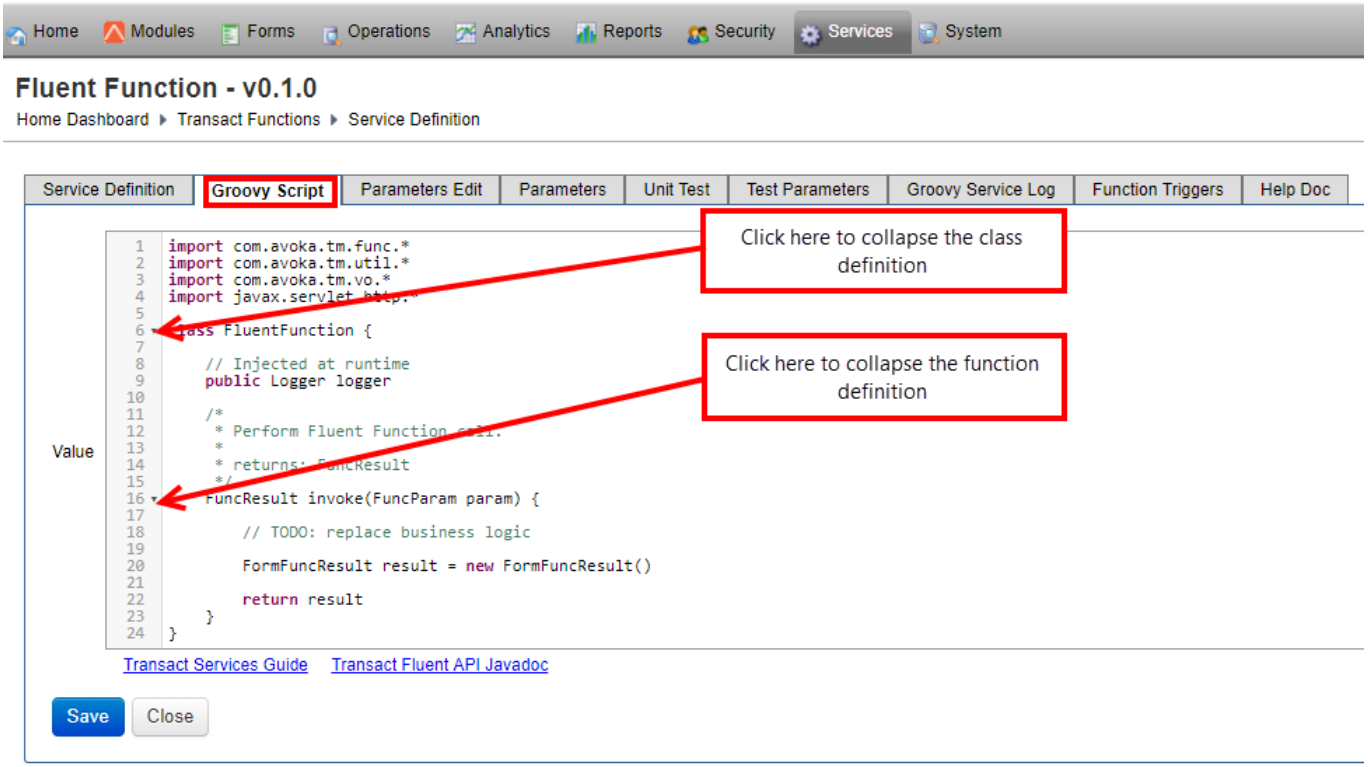
4. Click inside the Groovy script to start editing it. The simple example is shown below.

```
import com.avoka.tm.func.*
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import javax.servlet.http.*

class FluentFunction {
	// Injected at runtime
	public Logger logger
	/*
	* Perform Fluent Function call.
	*
	* returns: FuncResult
	*/
		FuncResult invoke(FuncParam param) {
			def formDoc = new XmlDoc(param.appDoc)
			formDoc.setText('/AvokaSmartForm/Applicant/FirstName', 'John')
			formDoc.setText('/AvokaSmartForm/Applicant/LastName', 'Smith')
			formDoc.setText('/AvokaSmartForm/Applicant/Email', 'jsmith@avoka.com')
			formDoc.setText('/AvokaSmartForm/Applicant/Mobile', '0412 345 678')
			FormFuncResult result = new FormFuncResult()
			result.includeFormData = true
		return result
	}
}
```

5. Click **Save** to update the changes.

The Groovy Script tab provides some basic editing options such as:

- Line numbering.
- Ctrl-Z.
- Ctrl-C and Ctrl-V.
- Collapse and extend class, function and method definitions.




```
import com.avoka.tm.func.*
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import javax.servlet.http.*

class FluentFunction {

    // Injected at runtime
    public Logger logger

    /*
     * Perform Fluent Function call.
     *
     * returns: FuncResult
     */
    FuncResult invoke(FuncParam param) {

        // TODO: replace business logic

        FormFuncResult result = new FormFuncResult()

        return result
    }
}
```

It's recommended to write a Groovy script in your favorite IDEA (Eclipse or IntelliJ) and then copy and paste it into the Groovy Script tab.

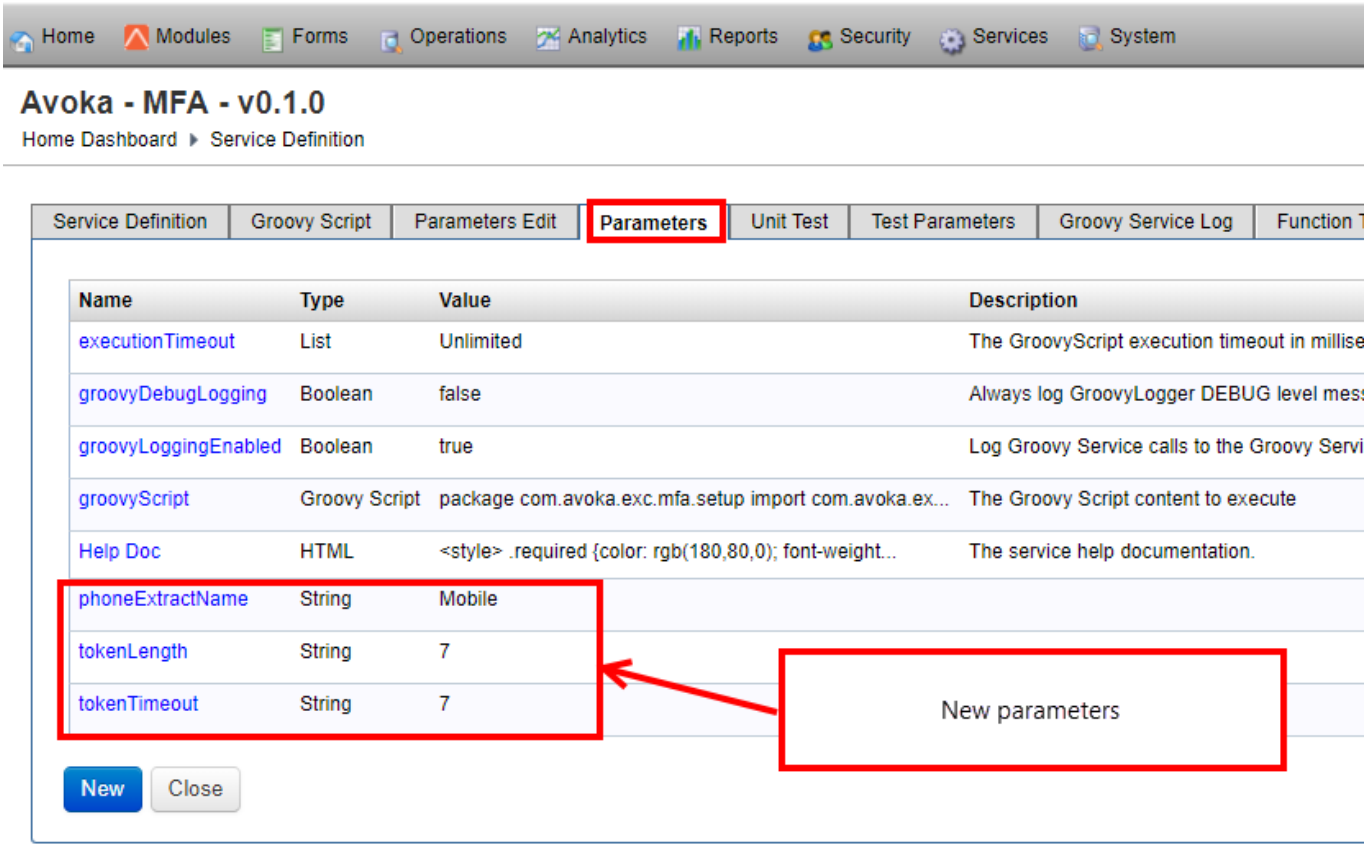
Click the two help links to get help on how to write Groovy scripts and which APIs to use:

- Service guide
- Transact Fluent API **Javadoc**[1]

Create a Groovy unit test script to test the business logic implementation.

If the Groovy script uses some configurable parameters, add them in the Parameters tab. The example below shows the new three parameters: **`tokenTimeout, tokenLength, phoneExtractName`**.

[1] Javadoc is a documentation generator created by Sun Microsystems for the Java language for generating API documentation in HTML format from Java source code.



The Groovy script snippet below shows how to use the new three parameters.

```
int tokenTimeout = inputParams.svcDef.paramsMap["tokenTimeout"]
int codeLength = inputParams.svcDef.paramsMap["tokenLength"]
String ref = (String) inputParams.svcDef.paramsMap["phoneExtractName"]
```

The parameters' values can be changed any time without any Groovy script modifications.

Next, learn how to configure Transact function parameters edit.

Configure Transact Function Parameters

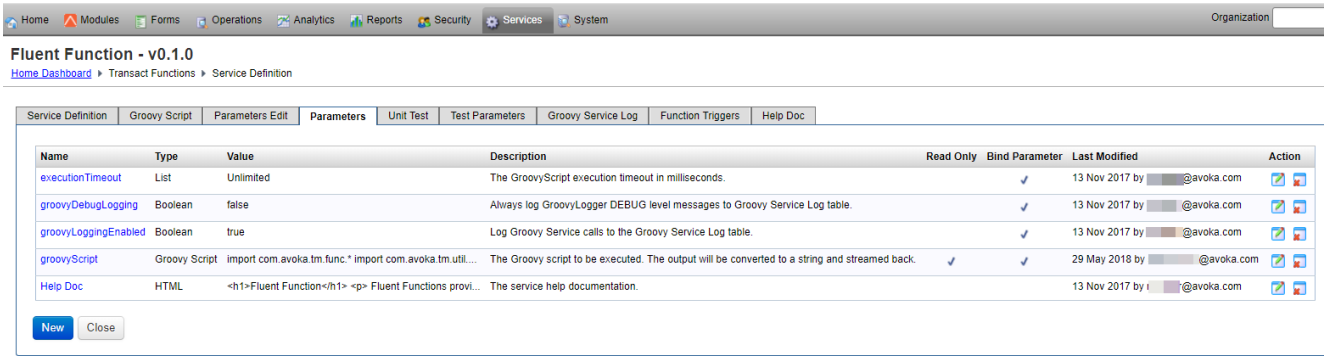
Journey Manager (JM) | Form Builder | 17.10

Manager allows you to view, create, configure, and remove Transact function Groovy script parameters in the **Parameters** tab.

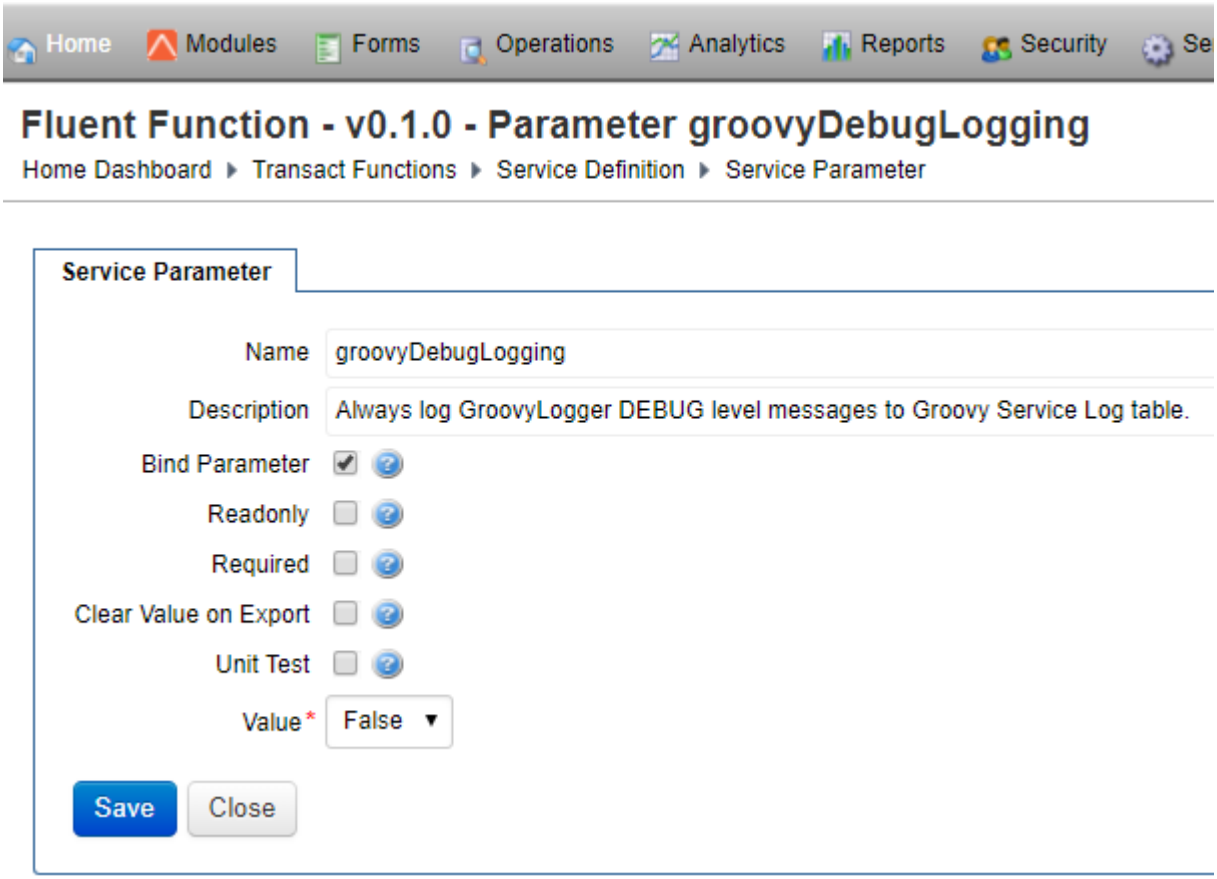
> You can update the same parameters in the Parameters Edit tab.

To configure Transact function parameters:

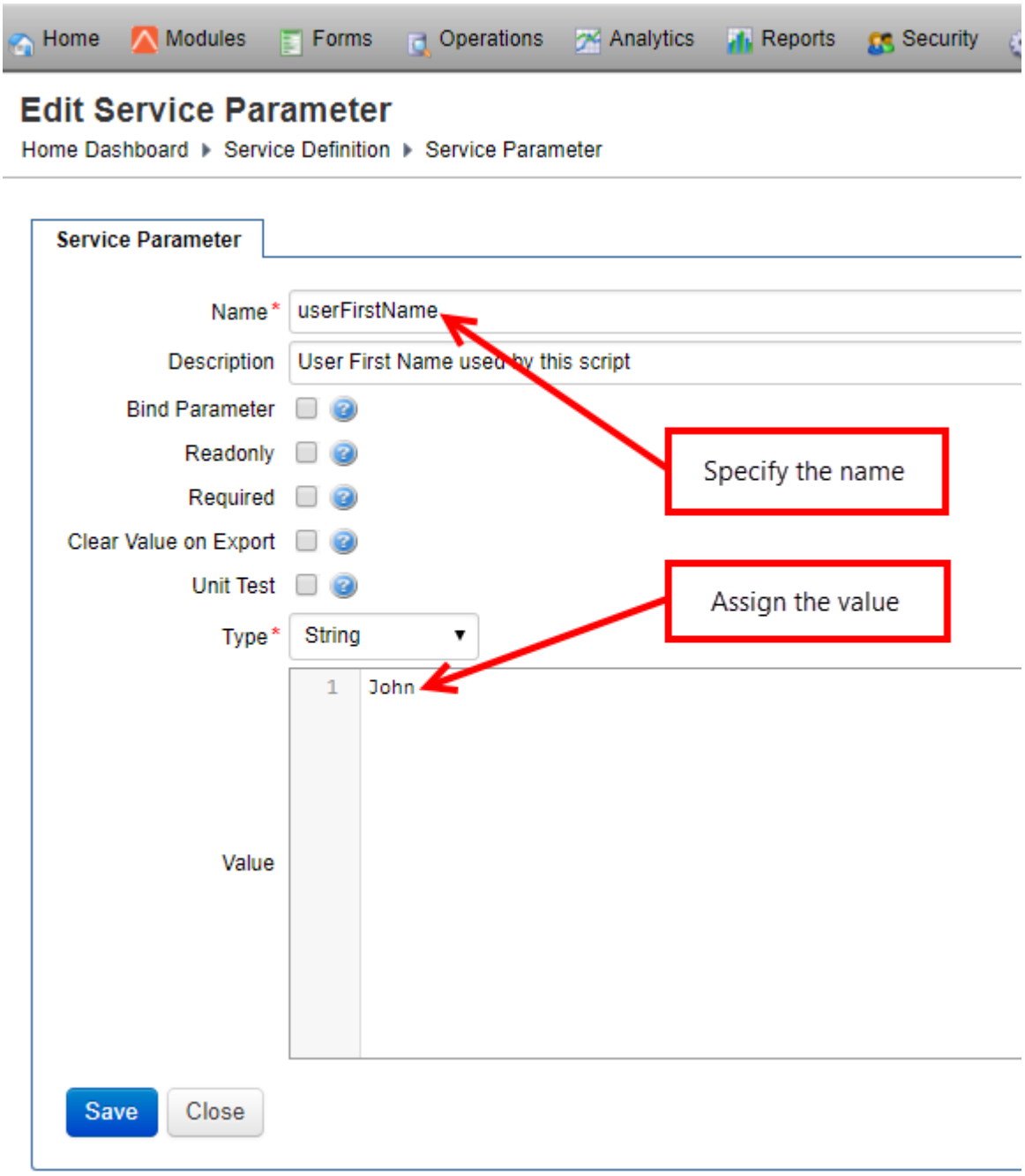
1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Click the **Parameters** tab.



4. Locate a parameter to modify and click **Edit**. For example, click **groovyDebugLogging** to view its configuration as shown below.



5. Click **New** to create a new parameter.
6. Fill in all required fields.



7. Update a Groovy script to make use of this parameter, for example **`String UserFirstName = (String) inputParams.svcDef.paramsMap["userFirstName"]`**.
8. Click **Save** to update the changes.

Next, learn how to configure Transact function Unit test.

Configure Transact Function Parameters Edit

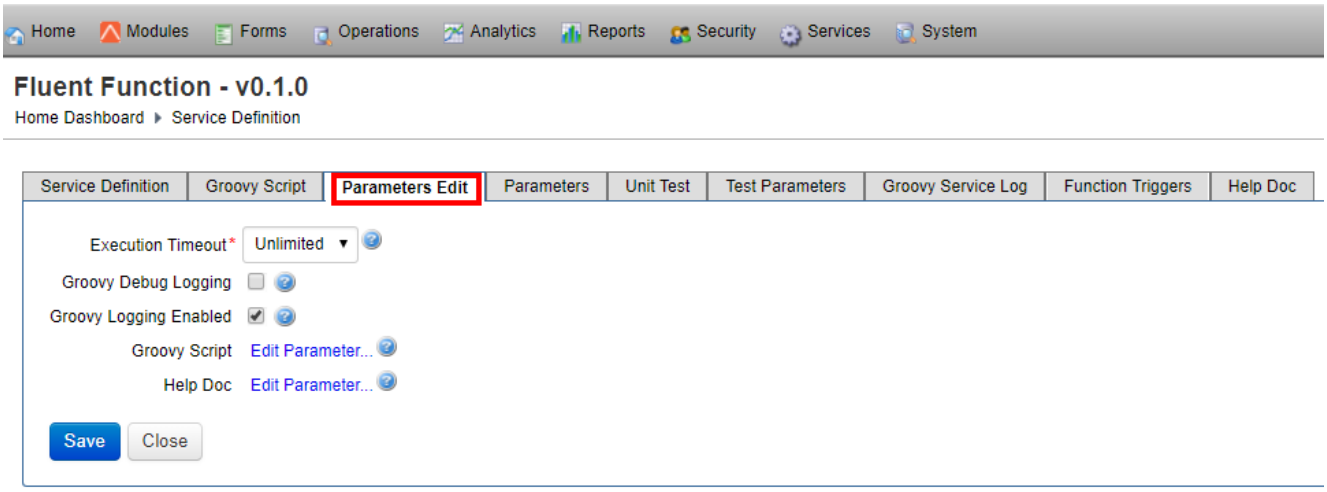
Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to view and configure **Groovy**[1] script parameters, which are used during a service object creation and initialization.

> You can update the same parameters in the Parameters tab.

To configure Groovy script parameters:

1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Click the **Parameters Edit** tab.



> If a Groovy script requires extra parameters, they can be added in the **Parameters** tab.

4. Specify the script execution timeout in milliseconds in the **Execution Timeout** field. The pre-defined values are:

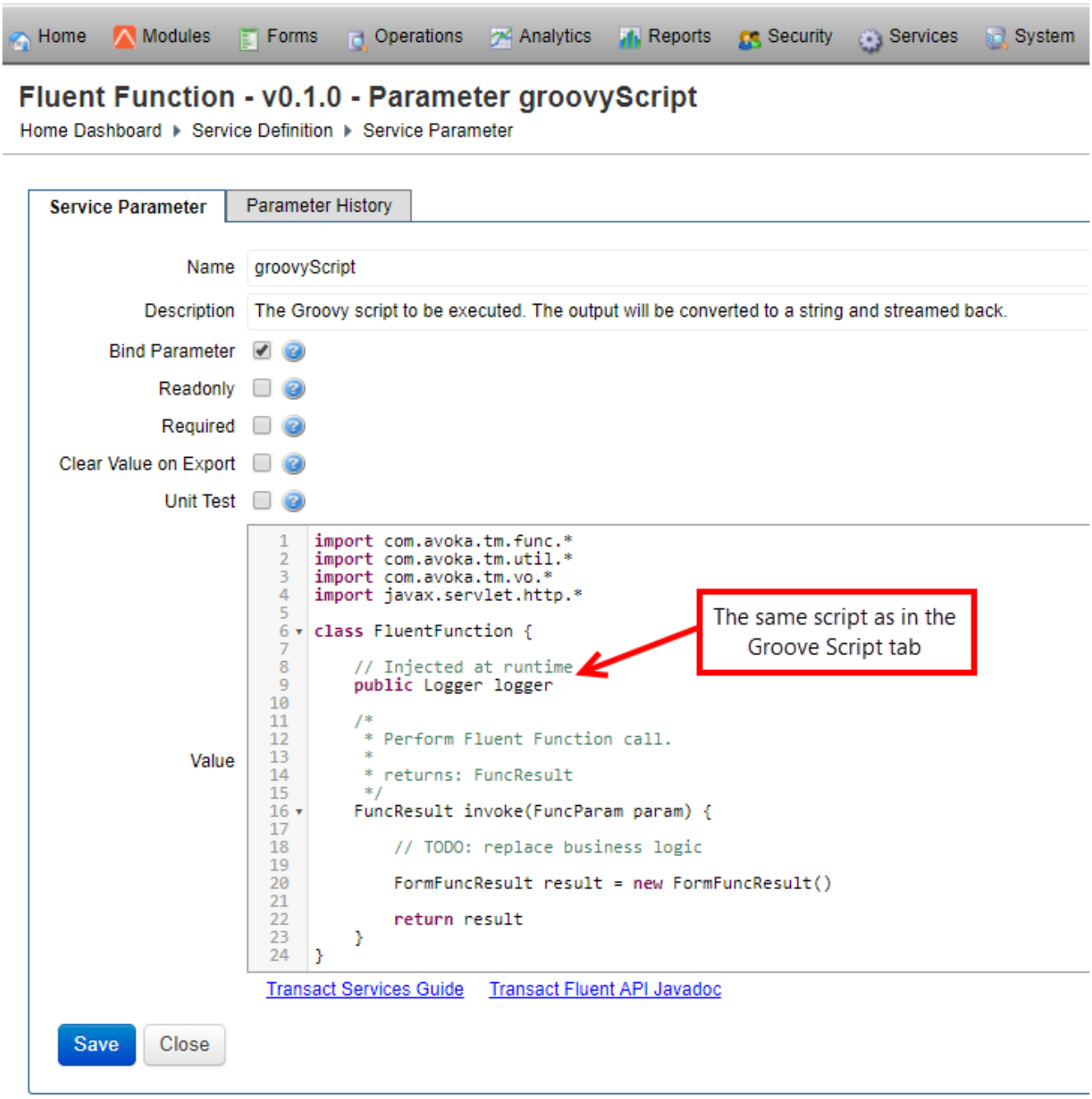
[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

- Unlimited - the service runs in the same thread that never times out and the forking of this thread is disabled. This is the default.
- 10000
- 30000
- 60000

5. Select the **Groovy Debug Logging** checkbox to log **GroovyLogger** messages, with the **DEBUG** level, in to the **GroovyServiceLog** table.

   Use this option to troubleshoot a Groovy script only. Be aware that it generates significant amount of verbose information in the **GroovyLog** table that can deplete a DB from free space and degrade the overall performance. If there are a lot of records in the **GroovyLog** table, the **Groovy Log** tab can open slowly.

6. Select the **Groovy Scripting Enabled** checkbox to log Groovy Service calls to the table. It is selected by default.
7. Click **Groovy Script** to edit the Groovy script and its parameters in the Service Parameter tab.



You can use the Service Parameter tab to edit Groovy script parameters listed below.

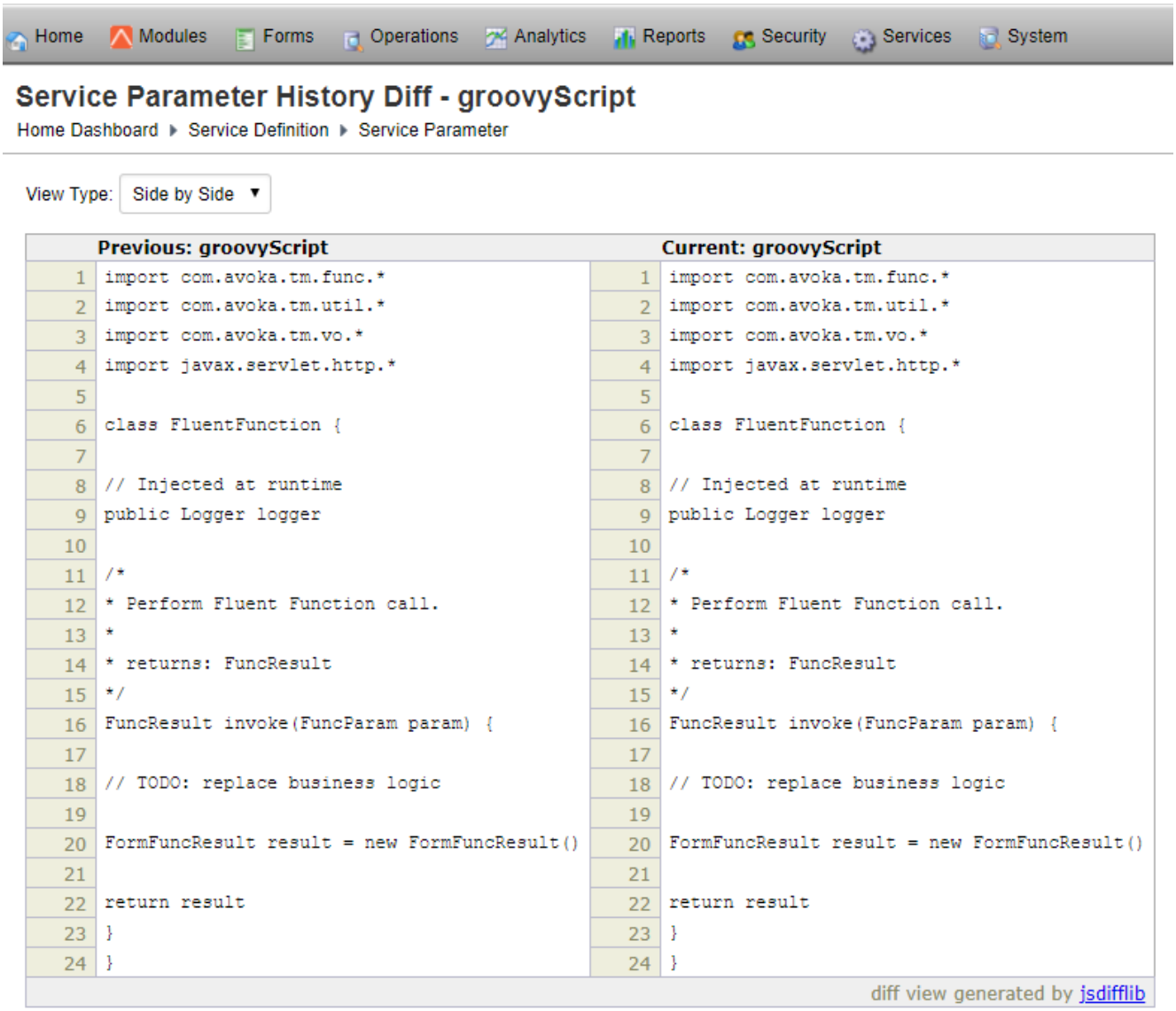
8. Select the **Bind Parameter** checkbox to inject the Groovy script into the created service object. It is selected by default.

The service class of this function must implement a setter method, for example void **`setObject(Object o)`**.

9. Select the **Readonly** checkbox to make the Groovy script read only, so it can't be edited.

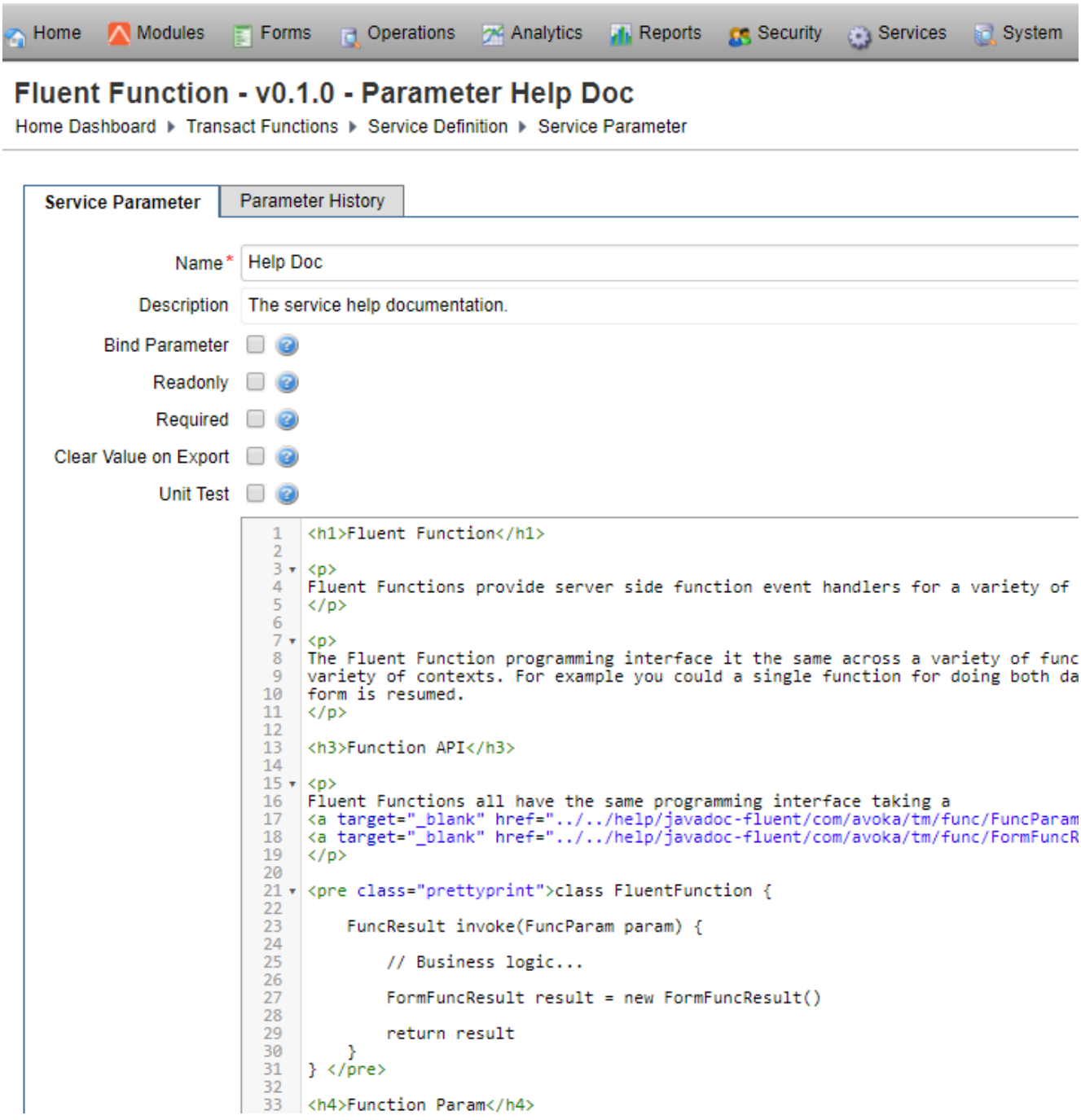
> If you've selected this checkbox, you can't edit any of the following options.

10. Select the **Required** checkbox to make the Groovy script required to create a service object.
11. Select the **Clear Value on Export** checkbox to clear the Groovy script when the service definition is exported. This ensures that server specific configuration can't be accidentally used in other services.
12. Select the **Unit Test** checkbox to use this Groovy script as a Unit Test only. This is useful in mock-up testing.
13. Click the **Parameter History** tab to see Groovy script change history.
14. Click **View Content Difference** to check what was changed in the script. Select the **View Type** to show the difference side by side or inline. This example shows no difference in the contents.



15. Click **Restore History Content** to restore this history content.

16. Click **Remove Service Parameter History** to remove this service history.
17. Click **Help Doc** to view or edit the Groovy script documentation page in the Service Parameter tab.



18. Click the **Parameter History** tab to see Help Doc change history. To configure this tab, see the Groovy script Service Parameter History tab described above.

Next, learn how to configure Transact function triggers.

Configure a Transact Function Unit Test

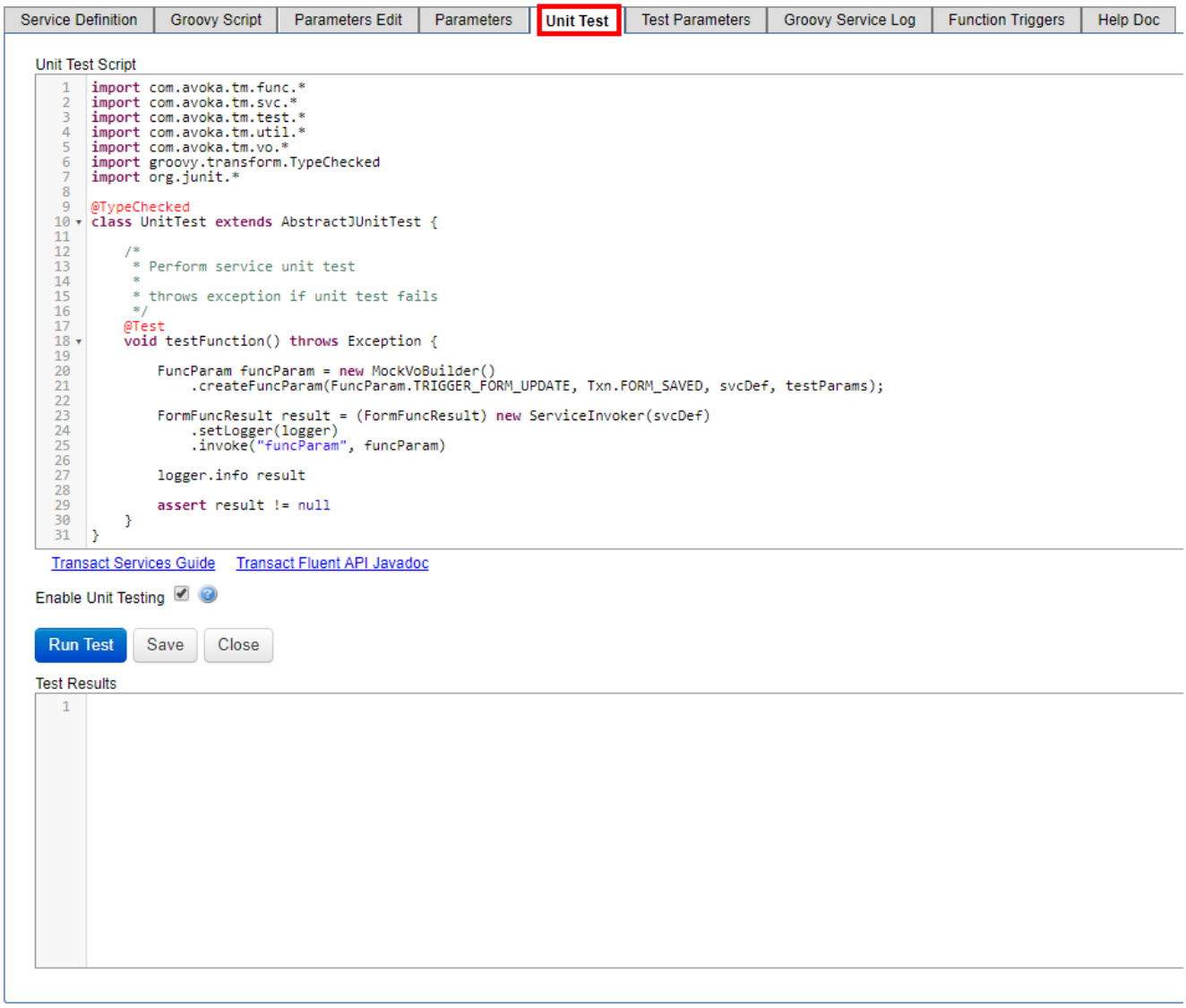
Journey Manager (JM) | Form Builder | Platform Developer | 17.10

It is a good development practice to create a **unit test**[1] for any Transact function Groovy script, so it can be thoroughly tested prior to using it. Manager provides Groovy unit testing capability to all Groovy services.

To configure the unit test:

1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Click the **Unit Test** tab.

[1] is a software development life cycle (SDLC) component in which a comprehensive testing procedure is individually applied to the smallest parts of a software program for fitness or desired operation. Unit Testing is a software testing method by which individual units of source code are tested to determine whether they are fit for use. For more information, see http://docs.groovy-lang.org/next/html/documentation/core-testing-guide.html.



4. Edit a Groovy unit test script as required. When a Transact function is created, it comes with a default Groovy script, which is merely a template to help developers implement a new test logic. You need to change it for each Transact function that you have to test.
5. Click on two help links below to get more help on how to write Groovy scripts and which Transact Fluent API to use:
   - Service guide
   - Transact Fluent API **Javadoc**[1]
6. Select the **Enable Unit Testing** checkbox to enable unit testing for this script.
7. Click **Run Test** to execute the Groovy test script. Test results, success or failure, is displayed in the **Test Result** field.

Next, learn about Transact function help documentation.

---

[1] Javadoc is a documentation generator created by Sun Microsystems for the Java language for generating API documentation in HTML format from Java source code.

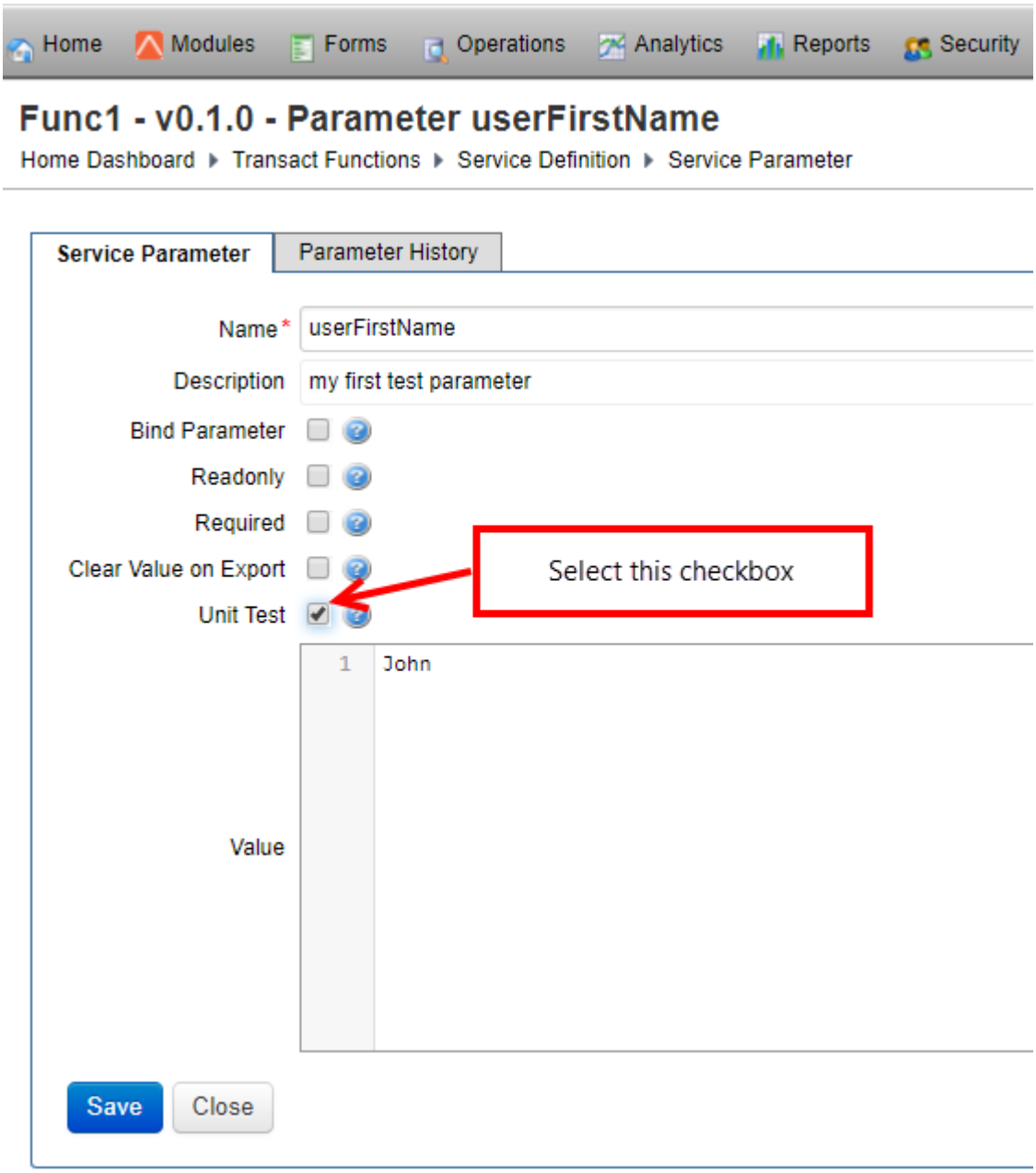
Configure Transact Function Unit Test Parameters

Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to configure Groovy Unit Test script parameters, which you can dynamically update in the **Test Parameters** tab.

To configure the Unit test parameters:

1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Click the **Test Parameters** tab.
4. Click **New** to add and configure a new parameter, for example, **`userFirstName`**.



5. Select the **Unit Test** checkbox to make this new parameter available in the **Unit Test** tab.
6. Click **Save** to update the changes.
7. Verify that the **`userFirstName`** parameter now appears in the **Test Parameters** tab.
8. Update the Groovy unit test script to make use of this parameter.

Next, learn how to run a Transact function Unit test.

Groovy Console

Journey Manager (JM) | Form Builder | Platform Developer | 17.10

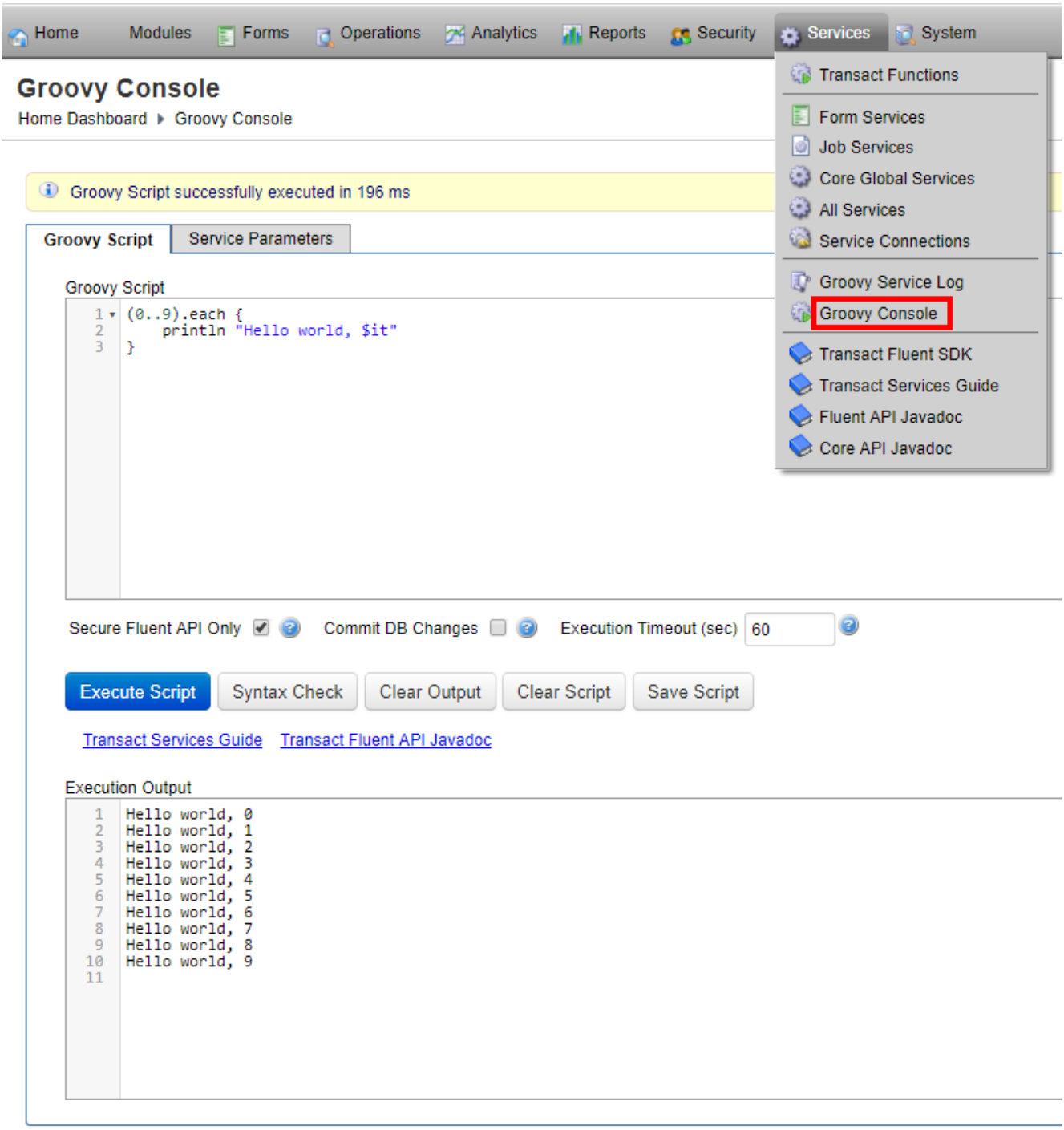
Manager comes with the **Groovy**[1] console enabling developers to rapidly develop and test Groovy scripts. Groovy console has access to all services and libraries, which allows you to simulate the behaviors of any Groovy service.

> You must have the Groovy Console permission set to your user account to run Groovy scripts in the Groovy console.

To test a script in the Groovy console:

[1] Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

1. Select **Services > Groovy Console.**



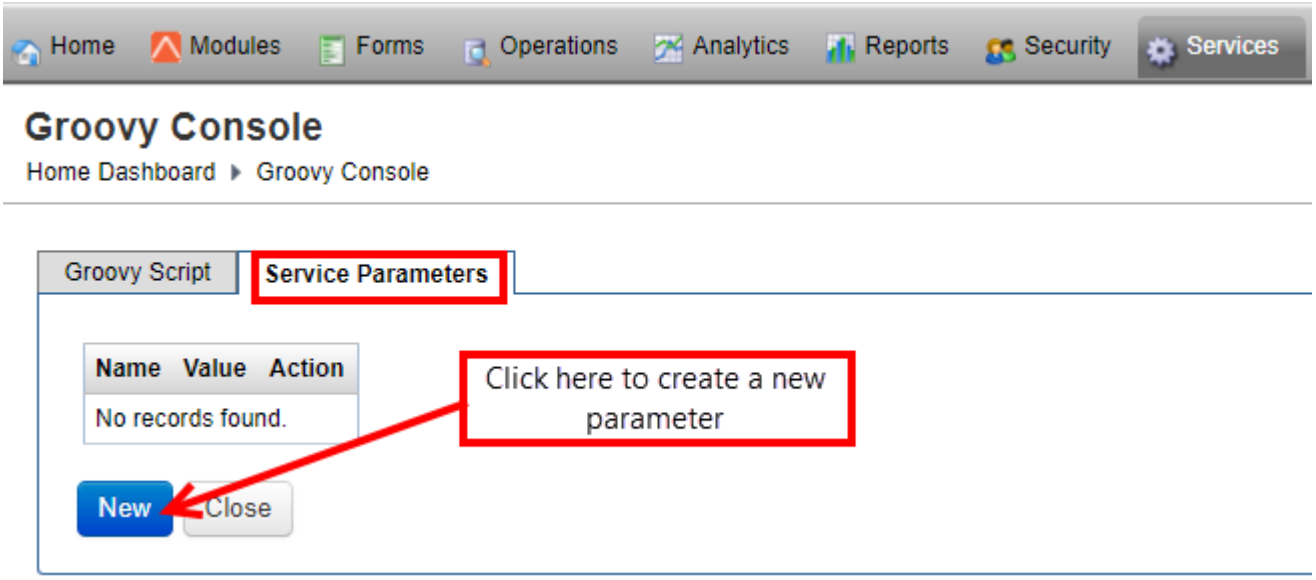
2. Select the **Groovy Script** tab to type a new or copy and paste an existing Groovy script here. You can type only a part of a script as well, which is shown above.
3. Select the **Secure Fluent API Only** checkbox to enforces the use of Transact Fluent API in this Groovy script. It is selected by default. Clear this checkbox if you want to test the Groovy script that uses Transact Core API.

At installation time, the system wide build property **groovy.secure.api** defines whether the **Secure Fluent API Only** service parameter is editable or available.

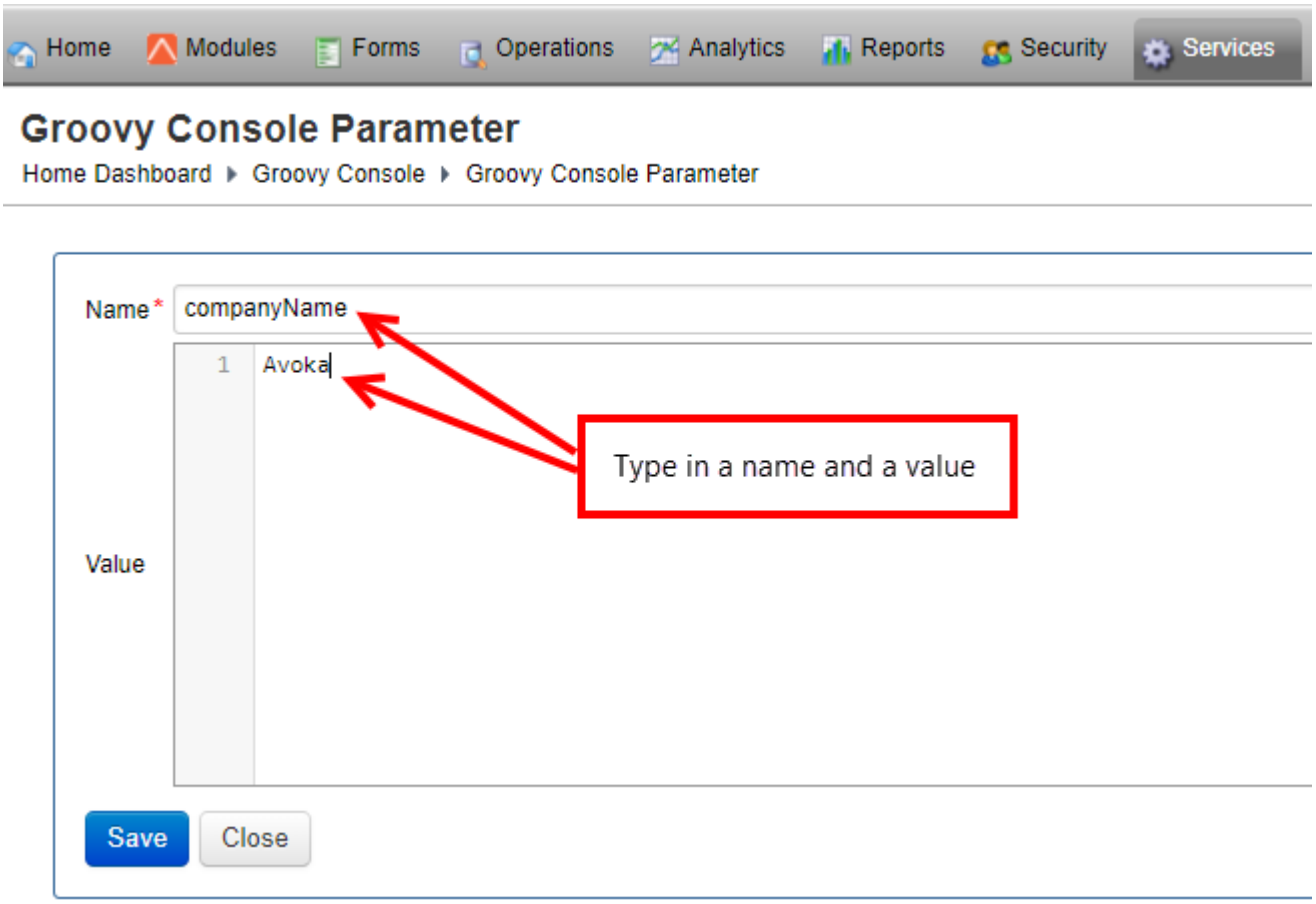
4. Select the **Commit DB Changes** checkbox to define whether any changes made by this script to the database has to be committed or rolled back. This is handy when you want to test a Groovy script that writes to the DB, but you don't want to delete these test records from the DB manually. It is not selected by default.
5. Enter a timeout in the **Execution Timeout (sec)** field to specify the Groovy script execution timeout in seconds, after which a timeout exception will be thrown. The default is 60 sec.
6. Click **Execute Script** to run it. The script output or an error message is shown in the Execution Output window.
7. Click **Syntax Check** to validate the Groovy script, which checks the script spelling, as shown below.

```
[Static type checking] - Cannot find matching method Script1#printl
(groovy.lang.GString). Please check if the declared type is correct and if the
method exists.
       @ line 2, column 1.
       printl "Hello world, $it"
       ^

       1 error
```

8. Click **Clear Output** to clear the **Execution Output** text field.
9. Click **Clear Script** to clear the **Groovy Script** text field.
10. click **Save Script** to save the script.
11. Select the **Service Parameters** tab to create and edit existing Groovy script parameters.
12. Click **New** to create a new parameter.



13. Type in a name and a value.



14. Click **Save** to update the changes. Now you can use **companyName** parameter in the Groovy script, as shown below.

```
def name = serviceParameters.companyName
(0..9).each {
       println "Hello $name, $it"
}
```

You can use more complex Groovy code snippets including imports, as shown below:

```
import com.avoka.tm.query.*
import com.avoka.tm.vo.*

String trackingCode = "LHC5DD"
String txnPropertyName = "Monty"

Txn txn = new TxnQuery()
       .setTrackingCode(trackingCode)
       .firstValue()

// PropertyQuery() to access the Monty property value
String txnPropertyValue = new PropertyQuery()
       .setName(txnPropertyName)
       .setTxn(txn)
       .getValue()

// expected result: txnPropertyValue = Python
println "txnPropertyValue = ${txnPropertyValue}"
```

A script saved in the Groovy console is specific to each user, so when you open the console next time, you will see your recently accessed script in there.

Next, learn about Groovy service log.

Groovy Service Log

Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to view messages generated and logged to files by functions and services written in Groovy, so you can debug and profile them. When a form calls a Transact function, it executes a Groovy script, which can be programmed to write some information into a log file, for example, `logger.debug("Variable: " + varaibleValue)`. In this case, this message is logged and can be viewed in the **Groovy Service Log** tab.

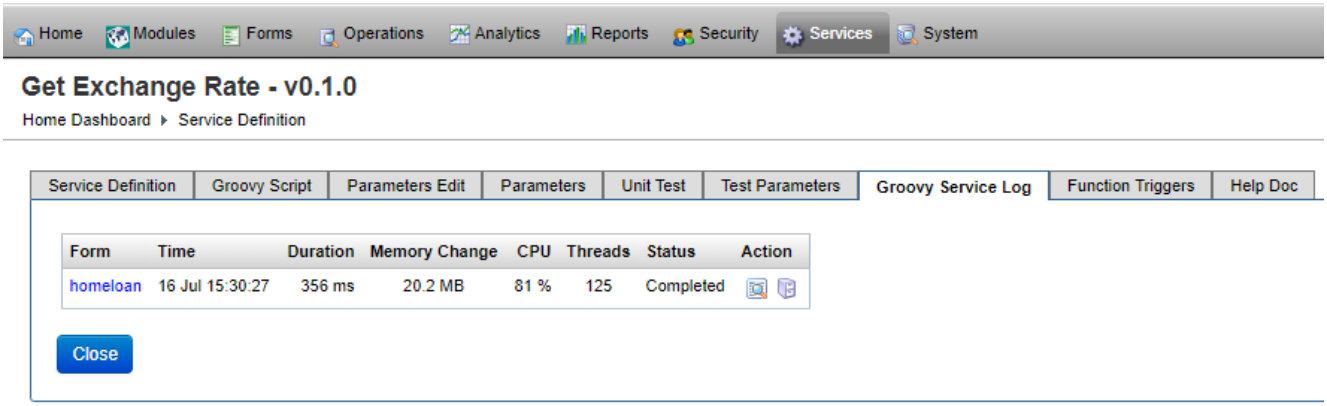
> All log messages associated with a transaction or **PII**[1] data are encrypted so you can't search for a part of encrypted text. If you want to search encrypted messages, you need to implement different logging solutions, for example, using the System Event Publisher.

The log can be empty if there is no debugging configured or if the debugging is disabled.

> We strongly advise against logging PII data to the service log, for example, to debug a service, as there's a risk of exposing customer's details through the log file.

To view a Transact function Groovy service log:

1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Click the **Groovy Service Log** tab to see the list of forms from where functions were invoked with the following details:



[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

- A function invocation time
- Duration of the invocation
- Memory change
- CPU
- Number of threads
- Status

4. Click a form name to see form details in a form dashboard.
5. Click **Groovy Logger** to view the output created during the function invocation.

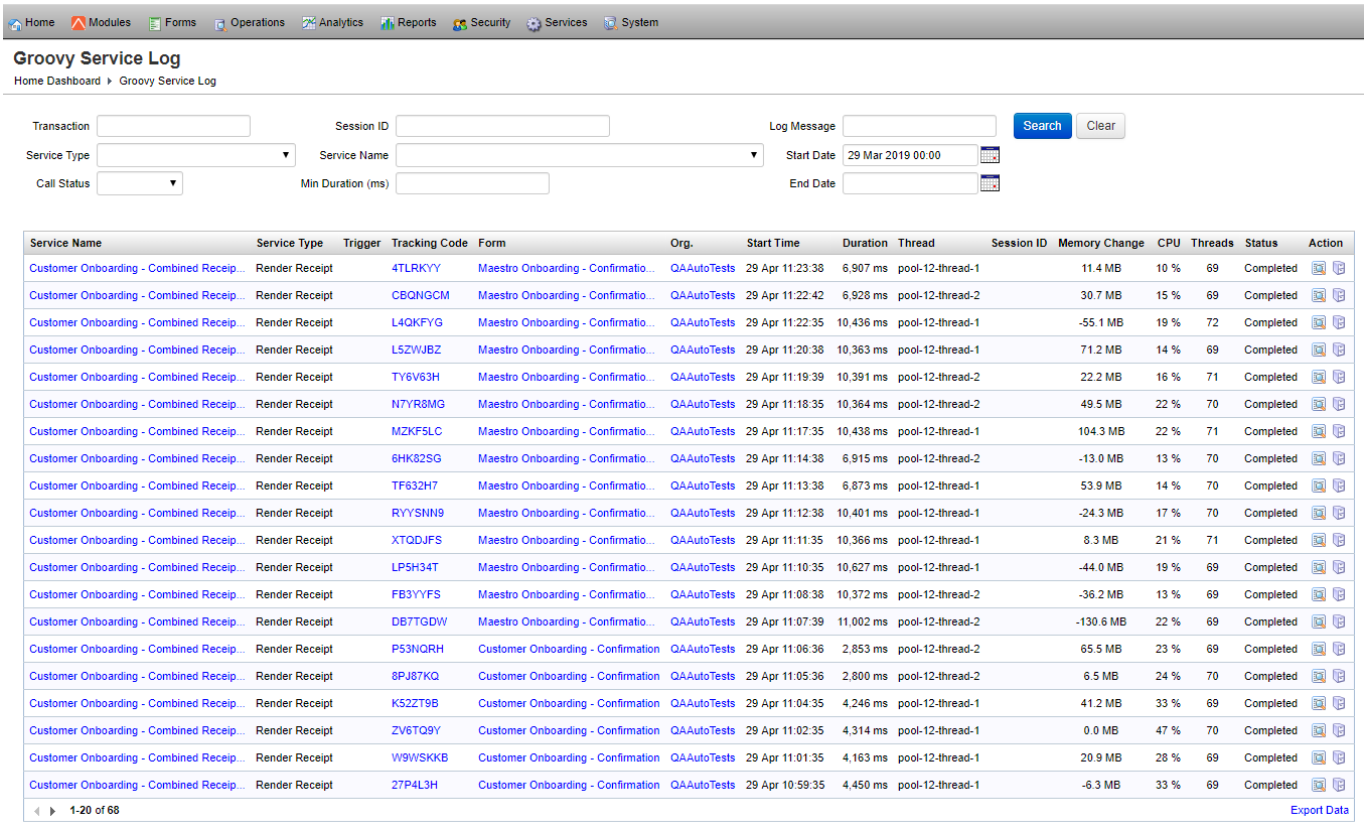
> Select the **Groovy Log** tab to see the same logged messages.

6. Click **Submission View** to see details of a transaction associated with this function call.
7. Click **Close**.

> To enable Groovy logging, select the **Groovy Scripting Enabled** checkbox in the Parameters Edit tab.

To view all Groovy service log messages, select **Services > Groovy Service Log**.

Messages are shown page by page, with the maximum number of records per page configured via the user's preferences.



To filter or search the Groovy Service Log by one or more criteria, specify the following settings and click **Search**:

- Transaction
- Session ID
- Log Message, or part of a message text, This searches through unencrypted messages logged by services that are not transaction-related.

  > You can't use this functionality to look for a part of encrypted text.

- Service Type. Select a type of a service from the dropdown list to see all messages logged by services of this type.
- Service Name. Select a service name from the dropdown list to see all messages logged by services of this name.

  > Only services of a selected service type are shown in the dropdown list, if you used the Service Type filter above.

- Call Status. Select a status of the message from the dropdown list, that can be: **Completed**, **Error**, or both.
- Min Duration (ms). Select a minimum duration of a service call. This is handy to troubleshoot services and overall server performance by filtering down services that took longer to do work.
- Start Date and End Date of the message.

Click **Clear** to reset the search criteria.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how configure Transact function service definition.

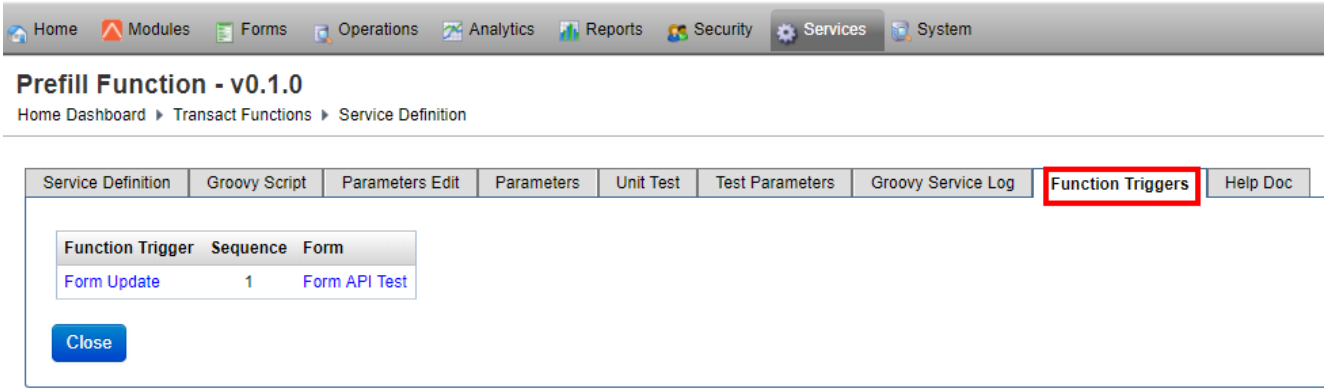
Configure Transact Function Triggers

Journey Manager (JM) | Form Builder ▪ Platform Developer | 17.10

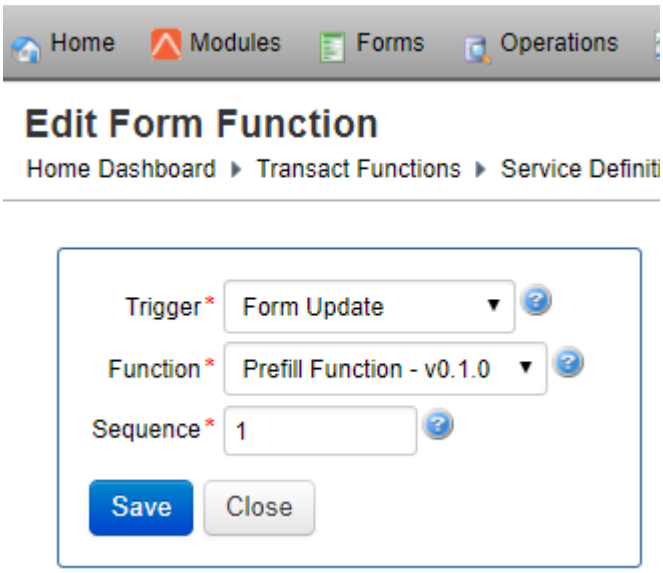
Journey Manager allows you to create and configure a Transact function trigger, which is a method that defines when a Transact function is called. For more information on triggers, see Fluent Functions in the Journey SDK documentation.

To configure Transact function triggers:

1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Click the **Function Triggers** tab to view all configured triggers for this Transact function.



4. Click a trigger name, for example, Form Update, to view and edit its configuration.

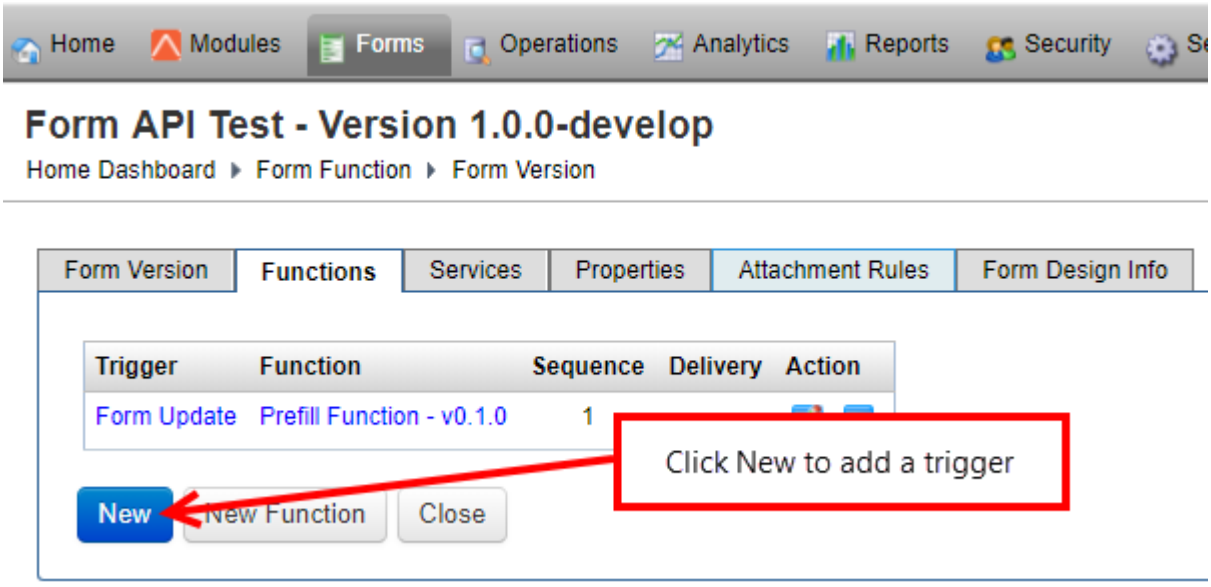


5. Select one of the following triggers from the **Trigger** dropdown list to define when a Transact function is called:

- Form Open is called when new form transaction is created by Manager, immaculately before it is rendered to the browser.
- Form Resume is called by Manager when a saved form transaction is resumed, immediately before it is rendered to the browser.
- Form Save Challenge is called when a save challenge is performed.
- Form Update is called when the form makes an background update operation in a browser.
- Form Ineligible is called when the form determines the user is ineligible to complete the application.
- Form Function is called by Maestro form business rule scripts generally to get dynamic data from the server and perform any transaction updates.
- User Save is called when the user explicitly saves and closes the form in a browser.
- User Submit is called when the user explicitly submits a completed form application in a browser.
- User Cancel is called when the user explicitly cancels and closes a form application in a browser.
- Background Delivery

For more information, see Function Triggers.

6. Select a function, which is invoked on the form trigger event, from the **Function** dropdown list.
7. Specify the sequence order in which Transact functions with the same trigger are called.
8. Click **Save** to update the changes.
9. Edit the form which uses this Transact function trigger in the new window. For more information on form functions, see form functions quick start in the Journey SDK documentation.



10. Click on a form link to view and edit its configuration.

Next, learn how to edit a Transact function Groovy script.

Transact Function Help Doc

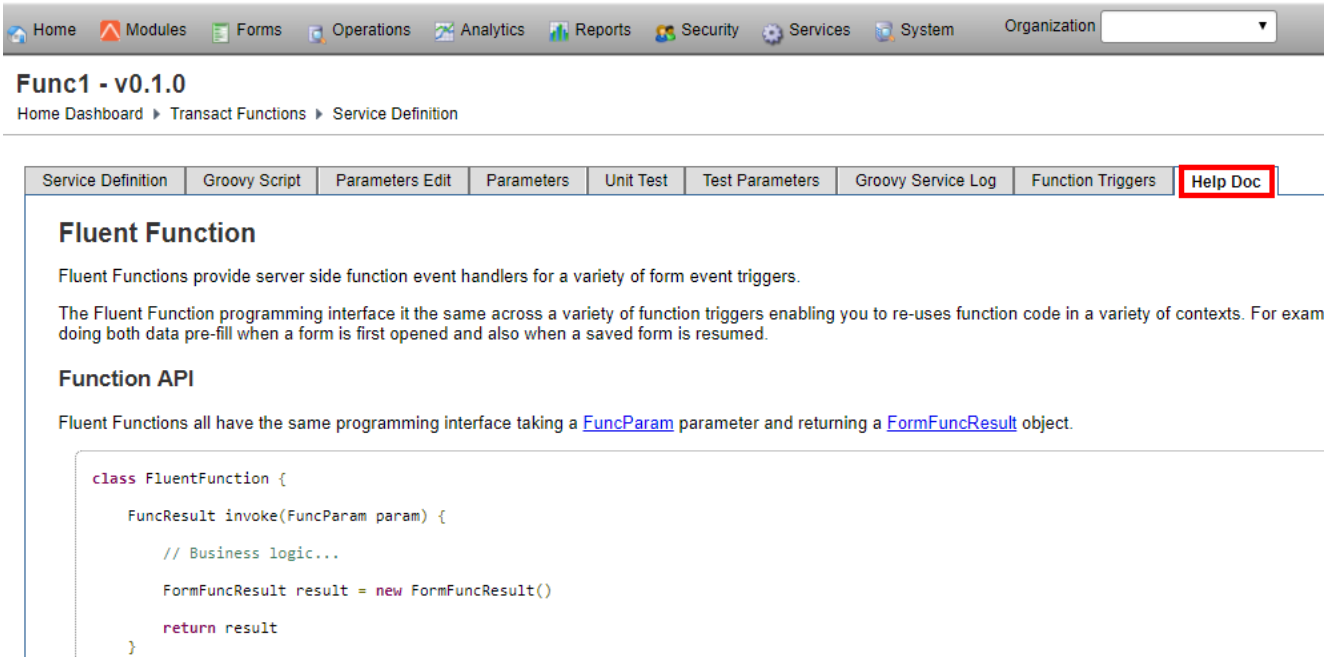
Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager provides the Help Doc tab to view and edit a Transact function documentation, which can include:

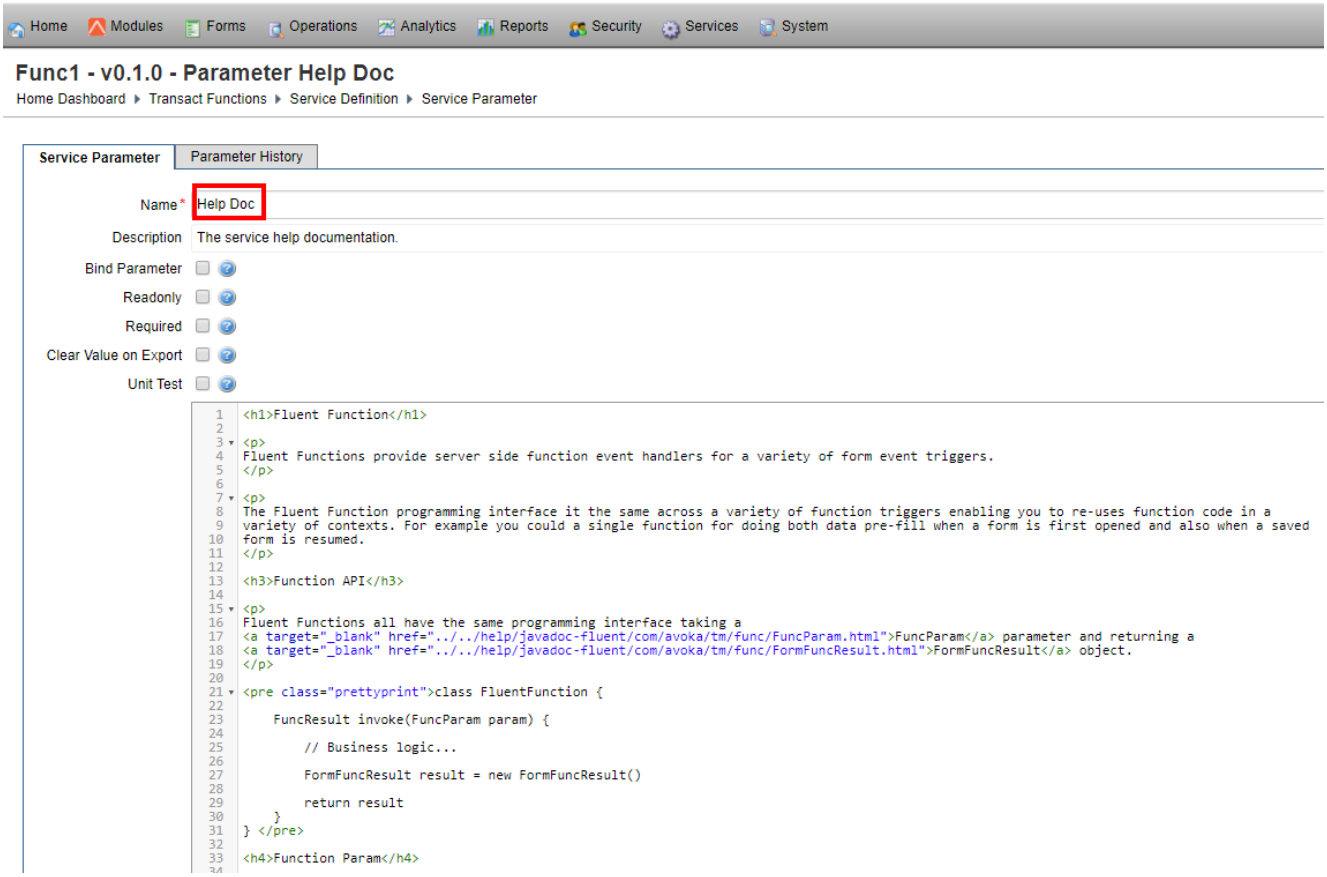
- Description of the function and how it is implemented
- References to Function APIs
- Triggers
- Exceptions
- Class definitions
- Code snippets and examples
- Test cases
- JSON definitions and examples.
- Some other specific information.

To view Transact function documentation:

1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Click the **Help Doc** tab.



4. To edit the help doc information, select the **Parameters** tab and click **Help Doc**.



5. Click **Save** to update the changes.

Next, learn how to configure Transact function parameters.

Server-Side Persistence Pattern

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

The main persistence model for Manager is the XML file. This is stored on the server, sent to the browser when a form is rendered, and sent back to the server when the form is submitted. It is also saved to the server in background save events.

However, there are cases when it may be useful to store information on the server which is not stored in the XML file. Some of these cases include:

- Storing results from a backend server call, such as an actual credit score. We want to record the actual score, but all we want to send to the form is a pass/fail. We don't want it to be in the XML because we don't want the end-customer to see their actual score, or the reasons why they failed.
- We want to temporarily store in-progress data that has been used in back-end service calls. This is to ensure that the end-user doesn't "game" the system, by changing the data in the form at different points in the journey.
- We want to store the results of a back-end service call for auditing or debugging purposes - but it's too much data to inject into the XML.
- There are probably many other cases.
- This data could be as simple as a string, or more complex like some JSON or XML.

There is a simple way to store and retrieve arbitrary string data against a Transaction using Fluent service calls.

Storing data

TxnUpdater has a method setProperty(String name, String value) which will allow you to create (or update) a property value.

Code will look something like this:

```
new TxnUpdater(txn)
    .setProperty("MyProp", "MyValue")
    .update()
```

Retrieving data

You can use TxnQuery to find the relevant Transaction, and then look at the Property Map within the Txn value object to retrieve the particular property.

Code will look something like:

```
Txn txn = new TxnQuery()
    .setSubmitKey(…) // or similar to find the correct transaction
    .withPropertyMap() // to retrieve the properties
    .firstValue() //to find the first matching transaction


// The txn contains an attribute propertyMap of type Map<String,String>
```

Some notes

- Data stored in the Transaction Properties are encrypted and subject to data retention policies, just like XML. (In recent versions of Manager.)
- In some cases, it may be necessary to store data against a collaboration job rather than an individual transaction.
- The PropertyQuery service is used for locating configuration properties for Form Versions, Workspaces, etc. This is for static configuration properties, not for per-transaction or runtime properties.
- It is also possible to use the standard HTTP session to store temporary data. However, this data will not be persisted beyond an individual session. A transaction property is persisted in the database.

Next, learn how to view Transact functions.

Handling Form Function Exceptions in Maestro Applications

Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager comes with the **Groovy**[1] console enabling developers to rapidly develop and test Groovy scripts. Groovy console has an access to all services and libraries, so it can simulate the behaviors of any Groovy service.

Form Functions are easy to use for the happy path scenarios when no errors or exceptions are taken into consideration. The simplest implementation can look like this:

```
Transact.formFunction(ffName, '0.1.0', 'myMilestone',ffData).then(function (response))
```

However, in real life errors do happen, so Form Function call implementations must cater for all sorts of errors and exceptions that may occur and handle both the success and error return messages. There are several ways of doing this; the following pattern is recommended: the Form Function captures all responses and returns different flags, for success or error statuses, and a detailed error message. This article explains how to use this pattern and provides examples of both server and client source code.

Journey Manager - the Server Side

If there is going to be an error, most developers would expect it to happen in a Form Function that runs in Manager. This is where calls to other systems are made, and most data manipulation occurs, so any exceptions should be handled here. However, placing a try-catch block around the code isn't enough. We need a consistent way to return a function call result, a success or a failure, back to the Maestro application's JavaScript, so it can effectively handle it and possibly display to a form end user.

When you develop Transact Functions, try to make them as self-contained as possible with regards to how they send data back to the application. Maestro applications have limited data passed back by a Transact Function when an exception occurs in Manager. Usually, it means that something catastrophic happened on the server side.

To address this limitation, you should create consistently named variable pairs and return them in the **`FormFunctionResult`** object; for example, **`executionStatus`** and **`errorMessage`**.

You can use the **`executionStatus`** variable to indicate whether the Transact Function call has followed a normal execution path or encountered a problem. A good development practice is to create a set of constants to use across your SDK projects. The example below uses constants as String objects. Condition codes are returned to indicate the following outcomes: success, problems with the input data, problems accessing third party APIs, or trapped exceptions.

---

[1] Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

You can use the **errorMessage** variable to hold an additional detailed message to compliment the **executionStatus** value. If the Transact Function doesn't capture an extra error message, this variable will be empty.

When an exception occurs, the Transact Function will catch the exception and populate the response object with one or both of **exceptionStatus** and **errorMessage**, so the Maestro application can better manage the user experience.

Maestro Application - the Client Browser Side

The **Transact.formFunction()** call from an application to a Manager service returns a promise. You can use this callback for successful, also known as happy path, results. However, this callback doesn't allow you to handle all possible conditions gracefully. You will need the second callback **catch()** added to the **then()**. This callback is executed if an error has occurred; for example, a Manager server failure or a network issue between the application and the server. The **catch (function(err))** input parameter is an error object containing a message which you can process and display to the end user.

The **catch()** block handles any rejections of the initial promise, specifically during the first **then()** callback. One use case is when a system-level error is detected in the Form Function on the server, the application can check this in the first callback and throw a **new Error** there. That Error object can then be processed in the **catch()** block.

Remember that **then()** and **catch()** promises are generic JavaScript structures and syntax, not Maestro specific.

Code Examples

The Click Rule JavaScript and Form Function Groovy Script below illustrate the exception handling pattern. They allow a user to select a test scenario, which can be one of the following options: Success, DataError, SystemError, Exception, and TM Error. The console log output shows the path of each scenario through the code.

The Click Rule JavaScript is shown below.

```
var ffName = "Form Function Full Example";
var ffData = {"testType":data.selectTest};

if(data.selectTest === "TM Error") {
  // Changing the name used will cause TM to report an error.
  ffName = "Will Not Work";
}

console.log("");
console.log("Start...");

Transact.formFunction(ffName, '0.1.0', 'myMilestone', ffData)
.then(function (response) {

  console.log("The Form Function was called, and returned something.");

  if(!Util.isBlank(response)) {
    if (!Util.isBlank(response.data) &amp;amp;&amp;amp; !Util.isBlank(response.data.ex-
ecutionStatus)) {
      switch(response.data.executionStatus) {

        case "DATA_ERROR":
          console.log("The return value status code indicates DATA_ERROR, with message: "
+ response.data.errorMessage);
          console.log("An exception is not being thrown based on design.");
          break;

        case "SYSTEM_ERROR":
          console.log("The return value status code indicates SYSTEM_ERROR, with message:
" + response.data.errorMessage);
          console.log("An exception can be thrown here if design dictates.");
          throw "SYSTEM_ERROR - " + response.data.errorMessage;
          break;

        case "SUCCESS":
          console.log("The return value status code indicates SUCCESS, with data: " +
JSON.stringify(response.data));
          break;

      }
    } else {
      console.log("An empty data object indicates an service problem.");
      console.log("An Exception/Error will be thrown manually from JavaScript.");
      throw "Empty Data Object";
```

```
    }
  } else {
    console.log("An empty response object indicates an infrastructure problem.");
    console.log("An Exception/Error will be thrown manually from JavaScript.");
    throw "Empty Response Object";
  }
}, function(err) {
  console.log("TM throwing an exception gets here - Using function(err)");
  console.log("Error Message: " + err.message);
}).catch(function(err){
  console.log("JavaScript throwing an exception gets here - Using promise's catch(func-
tion(err))");
  console.log("Error Message: " + err);
});

console.log("End of call.");
console.log("Calling Form Functions is an asynchronous operation.");
console.log("Execution continues here before the Form Function returns.");
console.log("");
```

The Form Function Groovy Script is shown below.

```
package com.avoka.funcs.fffullexample;

import com.avoka.tm.func.*
import com.avoka.tm.query.*
import com.avoka.tm.svc.*
import com.avoka.tm.util.*
import com.avoka.tm.vo.*

import javax.servlet.http.*

public class FormFunctionFullExample {

    // Injected at runtime
    public Logger logger

    /*
     * Perform Fluent Function call.
     *
     * returns: FuncResult
     */
    FuncResult invoke(FuncParam param) {

        def testType = param.params.get('testType').toString()

        logger.info("Test Type: ${testType}")

        FormFuncResult result = new FormFuncResult()

        def data = [
                rate   : 2.0
        ]

        // Choose execution path
        try {
            switch (testType) {
                case "DataError":
                    // Return data error
                    result.data.put("executionStatus", "DATA_ERROR")
                    String msg = "Invalid Input"
                    result.data.put("errorMessage", msg)
                    result.data.put("exchangeRate", {})
                    break

                case "SystemError":
```

```
                    // Return system error
                    result.data.put("executionStatus", "SYSTEM_ERROR")
                    String msg = "External System Down"
                    result.data.put("errorMessage", msg)
                    result.data.put("exchangeRate", {})
                    break

                case "Exception":
                    // Throw an unexpected exception
                    throw new Exception("FormFunctionFullExampleException")
                    break

                case "Success":
                default:
                    // Return sucessfully
                    result.data.put("executionStatus", "SUCCESS")
                    result.data.put("exchangeRate", data)
                    break

            }
        }
        catch (exception) {
            result.data.put("executionStatus", "SYSTEM_ERROR")
            String msg = "Exception Caught in Service - ${exception.toString()}"
            result.data.put("errorMessage", msg)
            logger.debug msg
            result.data.put("exchangeRate", {})
        }

        return result
    }
}
```

The various return statuses are explained below.

Success

The successful path, or the happy path scenario, is the result when everything in your application and Form Function works as expected. The Form Function validates the input values, any external services return to the caller function, so everything is good. In the example above, this results in the “Success” case being triggered. The response object contains data to be displayed, and the response.data.executionStatus value is “SUCCESS”.

Note the script log output is asynchronous, which is the nature of **`Transact.formFunction()`** call.

DataError

Sometimes development of an application and a Fluent Function get out of sync, or more likely an external service's logic implementation changes. This means that the inputs provided by the Maestro application pass its own validation, but they fail the validation of the Form Function or the third party service. In this case, the Form Function returns "DATA_ERROR" as "**executionStatus**", and some useful error message as **errorMessage**. You should always try to catch and fix these **DataError** situations during your software development and test cycle.

Execution flow through both the Form Function and application JavaScript is often unrestricted, meaning no exceptions are thrown. The problem can be gracefully reported to the end user, so they can report it to a system administrator or developer.

SystemError

SystemError scenarios cover any error conditions more severe than data validations. In the example above, the Form Function calls an external API and the external system returns an error or throws an exception. The Form Function catches this error and returns the appropriate values in result.data.executionStatus and result.data.errorMessage to the application.

The notable difference lies in implementing the application's JavaScript throw clause, which is used when a "SYSTEM_ERROR" value is returned in executionStatus. This moves the execution path into the catch() promise, where the error can be handled differently to a DataError in this pattern.

Exception

This execution path shows how an exception being thrown in the middle of the Form Function code should be captured by a try/catch and the result object populated accordingly. This scenario is otherwise handled as a **SystemError**.

Manager Error

Finally, there is the scenario where Manager has failed catastrophically. The example simulates this by using a Form Function that does not exist. Some infrastructure issues can cause similar errors; for example, network is not available. There is nothing in the Form Function code to handle this kind of error, as it would never be called.

The application JavaScript has a code path that is executed only when this level of error happens, the second promise used for rejections. The console.log() outputs the only information available about the cause of the **error (err.message)**.

Conclusion

This error handling pattern demonstrates two things:

How to code for error conditions using exceptions and the additional promise functionality available;

A way to convey execution status between Manager and the Maestro application run on the end user's device. Using **`executionStatus`** and **`errorMessage`** is a simple, and yet powerful approach. It allows the Form Function to pass error related information back to the client while keeping all error handling on the service.

Next, learn about Groovy service log.

Validate Form Data Against a Schema

Journey Manager (JM) | Form Builder | Platform Developer | 5.1 & Higher

There are some scenarios where you need to validate submitted XML against an XML Schema. This below code snippet is an example of this in Groovy.

```
import javax.xml.XMLConstants
import javax.xml.transform.stream.StreamSource
import javax.xml.validation.SchemaFactory
import org.xml.sax.SAXException

// submissionString is a String variable containing the submitted XML

// the schema content is stored as a parameter on the Service Definition
def xsd = serviceDefinition.getServiceParameter('schema').getValue()


def factory = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI)
def schema = factory.newSchema(new StreamSource(new StringReader(xsd)))
def validator = schema.newValidator()
validator.validate(new StreamSource(new StringReader(submissionString)))
```

Next, learn more about the server-side persistence pattern.

User Builder API Overview

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

The UserBuilder Fluent API is a great little api that can be used to create new TM users, update or change user properties such as their role or spaces assigned, and much more.

Let's take a look at an example of how to create a new user.

Create New User

A simple code example could look like this:

```
import com.avoka.tm.svc.*
import com.avoka.tm.vo.*

Jser thisUser = new UserBuilder(“Maguire”)
                .setEmail(“johndoe@avoka.com”)
                .setFirstName(“John”)
                .setLastName(“Doe”)
                .setRoleNames([“Developer”, "Maestro Developer"] as Set)
                .setOrgNames([“ACME”] as Set)
                .setSpaceNames([“maguire”] as Set)
                .setUserType(User.TYPE_LOCAL)
                .setLoginName(“alanreiley@gmail.com”)
                .setPassword(“testP@sswd12”)
                .create()
// optionally use createOrUpdate() which will only update if the user already exists.
```

The above example will create a new user with a user name of johndoe@avoka.com, an email address of johndoe@avoka.com, has been assigned to the Maguire space, was granted Developer and Maestro Developer roles, Maguire space name, and the ACME organization. We had to add the as Set for the list types (which wasn't mentioned in the API documentation) so that Groovy knows to cast this as a Set. Otherwise, you will get an error. We also can choose between creating a Local account, LDAP-based, or an SSO-based users, as seen illustrated above with setUserType.

Updating an existing user is almost as simple, but comes with a caveat that you can easily overcome. The any property that is a Set, such as Roles, Spaces, and Organization, the update method replaces the existing properties assigned to that user. Let's take a look at updating an existing user's properties.

Update an Existing User

Groovy gives us a very easy way to convert our object to a string representation - JSON.

```
import com.avoka.tm.svc.*
import com.avoka.tm.vo.*

User user = new UserBuilder("Web Plug-in")
  .setLoginName("johndoe@avoka.com")
  .setRoleNames(["Developer","Maestro Developer","Maestro SCM Design","Maestro SCM Org
Library","Maestro SCM Project"] as Set)
  .update()
// Updates the user account specified by the loginName parameter
```

The above example will find the user: johndoe@avoka.com and updates their role assignment to become: Developer, Maestro Developer, Maestro SCM Design, Maestro SCM Org Library, Maestro SCM Project.

Use the Groovy Console

Remember to select "Commit DB Changes" if you're using this script in a Groovy Console or the DB write will be ignored and no user will be created.

Conclusion

The UserBuilder API is a very powerful tool for creating and modifying users and has become a very handy tool for user creation here within the Support Group. Some other helpful functions of the UserBuilder API are available to set the Account Status, remove an Org, and set the User Type to SSO.

Next, learn how to view Transact functions.

Access Form Properties with a Groovy Service

Journey Manager (JM) | Form Builder | Platform Developer | 5.1 & Higher

It can be the case that the operation of a Groovy service (prefill, delivery, dynamic data, etc.) must be tailored slightly for specific forms. One way of storing the data for these customization is at the form level, in form properties. Once the data has been stored in these form properties, it must obviously be accessible from within the service that requires the data.

Accessing properties from within a Groovy script can be achieved by using the PropertyValueDao class. Using this class you can make calls to the 'getPropertyValue' method, providing the name of the property you're looking for and the form object it's associated with. Most Groovy services in TM (Prefill, Dynamic Data, Receipt Number, Form Saved, Submission Complete) have a predefined reference to the form object, in variable called 'form'. For these services properties for that form can be accessed with the following code:

```
import com.avoka.fc.core.dao.PropertyValueDao

PropertyValueDao propertyValueDao = new PropertyValueDao()

def theValue = propertyValueDao.getPropertyValue("The Property Name", form)
```

Delivery Processes on the other hand don't have this existing reference to the form object, but can acquire one easily from the 'submission' object as in the following example:

```
import com.avoka.fc.core.dao.PropertyValueDao

PropertyValueDao propertyValueDao = new PropertyValueDao()

//Notice 'form' has been substituted for 'submission.getForm()'
def theValue = propertyValueDao.getPropertyValue("The Property Name", submission.getForm
())
```

Next, learn how to view Transact functions.

Access Service Parameters of Another Groovy Service

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

The requirement was for a 'Form Prefill' service to be able to access the service parameters configured within an existing 'Delivery Process' service. The use case being that there was some common configuration between the services and that they wished to maintain these settings in a single location. As it turns out this functionality is built into the TM Java libraries and it was fairly straight forward to implement the solution. Start by adding the following imports to the top of your Groovy script:

```
import com.avoka.fc.core.service.IServiceDefinitionAware

import com.avoka.fc.core.service.ServiceLocator

import com.avoka.fc.core.entity.ServiceDefinition
```

Then, use the 'ServiceLocator' class to find the service with the parameters you want to access and retrieve its service definition:

```
IServiceDefinitionAware tmService = ServiceLocator.getServiceForName("The Existing Ser-
vice's Name")

ServiceDefinition serviceDefinition = tmService.getServiceDefinition()
```

Finally, get the service parameter you're looking for and where necessary access the value it contains:

```
def param = serviceDefinition.getServiceParameter("The Parameter's Name");

def paramValue = param.getValue();
```

Next, learn how to view Transact functions.

Create Anonymous Tasks with a Groovy Service

Journey Manager (JM) | Form Builder | Platform Developer | 5.1 & Higher

A task is a form that has been prefilled with data and then assigned to a user. The typical use case for a task is: A user opens and completes a blank form. On submission, a task is created for another user, containing the original user's submitted data. The second user then reviews or modifies the data before submitting for processing.

Tasks can be assigned to Manager users, in which case they are accessed when that user logs into a portal, or they can be assigned anonymously, in which case a saved form is generated, and the URL is sent via email. Anonymously assigned tasks can be created in a groovy script by using the **`SubmissionTaskService`** object.

The SubmissionTaskService API can be accessed at **`<Manager server->/manager/admin/javadoc/com/avoka/fc/core/service/SubmissionTaskService.html`**

The following groovy script can be used to assign anonymous tasks:

```
import com.avoka.fc.core.entity.*
    import com.avoka.fc.core.dao.*
    import com.avoka.fc.core.service.SubmissionTaskService
    import com.avoka.fc.core.service.SubmissionTaskService.SavedFormParam
    import com.avoka.fc.core.util.PortalUtils
    import com.avoka.core.util.StringTemplate
    import com.avoka.fc.core.service.EmailService


    def formUrl = ""

// Setup the saved form parameter DTO
    def param = new SavedFormParam()

// Setup the form parameter
    def form = DaoFactory.getFormDao().getFormByFormCode( serviceParameters.FormCode ?:
"" )
    if(!form){ throw new Exception("Form Code is invalid, form not found.") }
    param.form = form

// Setup the portal parameter
    // uses the portal provided, if none provided, used the one configured as a service
parameter.
    def portalName = serviceParameters.PortalName ?: ""
    def portal = DaoFactory.getPortalDao().getPortalByName( portalName )
    if(!portal){ throw new Exception("Portal Name is invalid, portal not found") }
    param.portal = portal

// Check the email serviceParameters
    if(serviceParameters.SendEmailFlag){
        if(!serviceParameters.ContactEmailAddress){ throw new Exception("Contact Email
Address must be provided"); }
        if(!serviceParameters.EmailSubject){ throw new Exception("Email Subject must be
provided"); }
        if(!serviceParameters.EmailMessage){ throw new Exception("Email Message must be
provided"); }
    }

// taskSubject
    param.taskSubject = ""
    // taskMessage
    param.taskMessage = ""

// saveChallengeAnswer
```

```
    param.saveChallengeAnswer = ""
    // submissionXml
    if(!serviceParameters.FormXml){ throw new Exception("Form Xml must be provided") }
    param.submissionXml = serviceParameters.FormXml

// Create the task
    def submissionTaskService = new SubmissionTaskService()
    def submission = submissionTaskService.createAnonymousSavedForm(param)

// Task created, send notification email if required
    //=====================================================================
    if(serviceParameters.SendEmailFlag){
        def emailService = new EmailService()

    // Get the URL to resume the saved form
        formUrl = PortalUtils.getFormSavedUrlWithReferenceNumber(portal, submission)

    // Create data map for use in Velocity templates
        def dataMap = [:]
        dataMap.put("formUrl", formUrl)

    def bodyStringTemplate = new StringTemplate(serviceParameters.EmailMessage)
        def emailBody = bodyStringTemplate.merge(dataMap)

        emailService.sendMessage(
            serviceParameters.EmailSubject,
            emailBody,
            null,
            serviceParameters.ContactEmailAddress,
            "",
            null)
    }

return formUrl
```

To run the script, you need to configure the following service parameters:

- ContactEmailAddress - <String> the email address that the URL of the created task is sent to
- EmailSubject - <String>
- FormCode - <String> The form code of the form used to generate the task. This form code must exist on the TM server.
- FormXml - <String> The xml data used to prefill the generated task
- PortalName - <String> Then name of the portal that the form belongs to.
- SendEmailFlag - <Boolean> Select 'True' to send the email to the assigned contact
- EmailMessage - <String> A velocity template of the email body HTML. Velocity templates contain HTML with placeholders for dynamic content. In this case, a placeholder must be configured for 'formUrl', which will be populated with the link to the created task.

An example of an email message velocity template is:

```
<html>
        <head>
            <style type="text/css">
                /* custom CSS here */
                body {
                    font-family: Verdana, Arial, Helvetica, sans-serif;
                }
            </style>
        </head>
        <body>
            <!-- custom Content here -->
            <div>
                This is my message.<br>
                Form Link: <a href="$formUrl">$formUrl</a>
            </div>
        </body>
    </html>
```

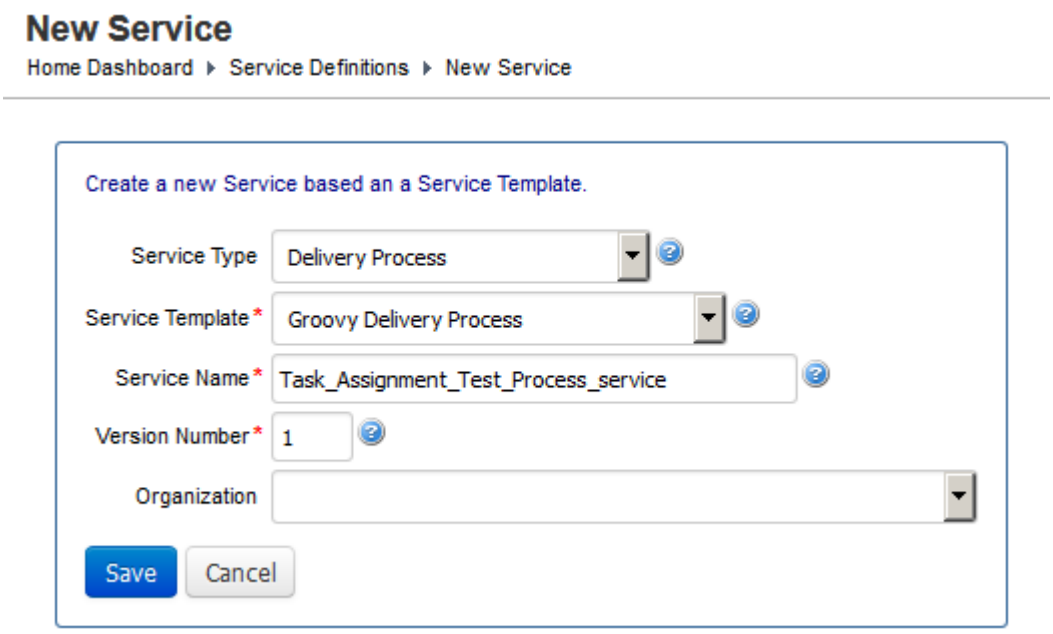
Next, learn how to view Transact functions.

Create Form Tasks with a Groovy Delivery Service

Journey Manager (JM) | Form Builder | Platform Developer | 5.1 & Higher

A form can be configured to create a Form Task on Submission via configuration of a Delivery Process, which is then assigned to the Form.

First, you need to create a delivery process, as shown below.

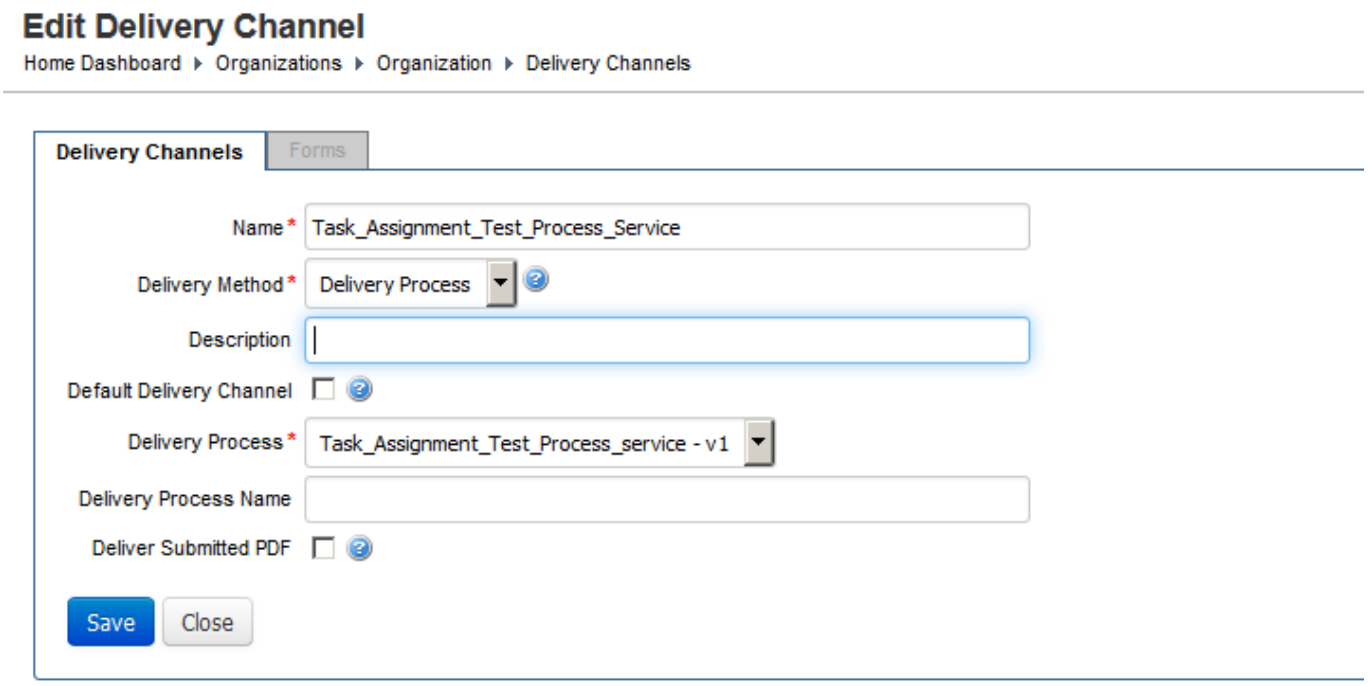


The Service Definition's Groovy Script service parameter is configured with Groovy Script to create a new Form Task using the Task Server createFormTask method.

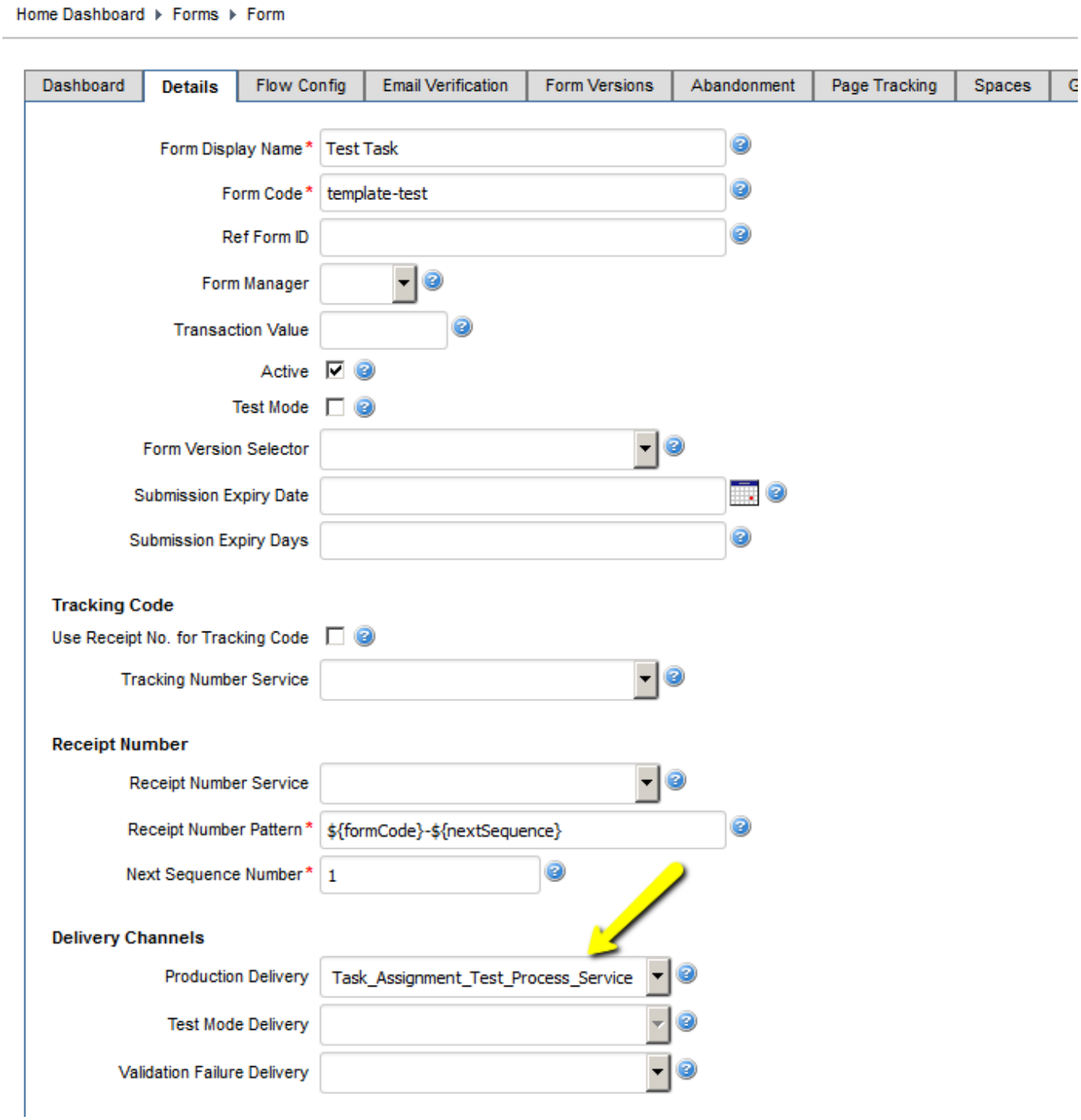
The following script example shows the simplest example possible of the use of the TaskService to assign a task to a user based on the same form and submission data from the initial user submission. The task is assigned to a single user for review. In real world scenarios a different form would be used for the review or the submission Xml would be manipulated to set the form into review mode when rendered from a task.

```
import com.avoka.fc.core.service.ServiceFactory
import com.avoka.fc.core.dao.DaoFactory


def user = DaoFactory.getUserAccountDao().getUserAccountForLogin("reviewUser");

def taskService = ServiceFactory.getTaskService();
def submission = taskService.createFormTask(submission.form,
                                            "Please Review",
                                            "You have a task to review",
                                            null,
                                            user,
                                            null,
                                            submissionXml,
                                            null,
                                            null,
                                            null,
                                            null,
                                            null,
                                            null,
                                            true,
                                            "Please Review",
                                            "You have a task to review",
                                            false);
```

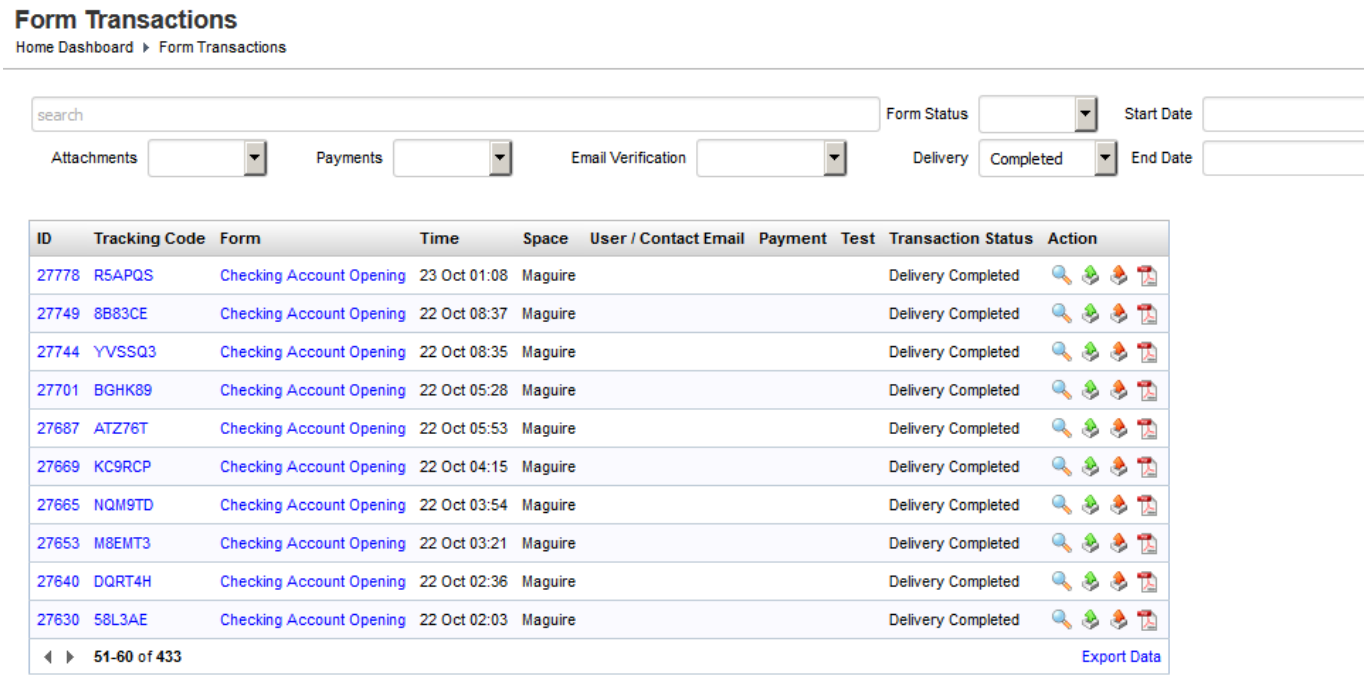
This Delivery Process method is then added to the Form's Organization Form Delivery Detail configuration. This will enable the Delivery Process to be available on the organization's forms configuration.



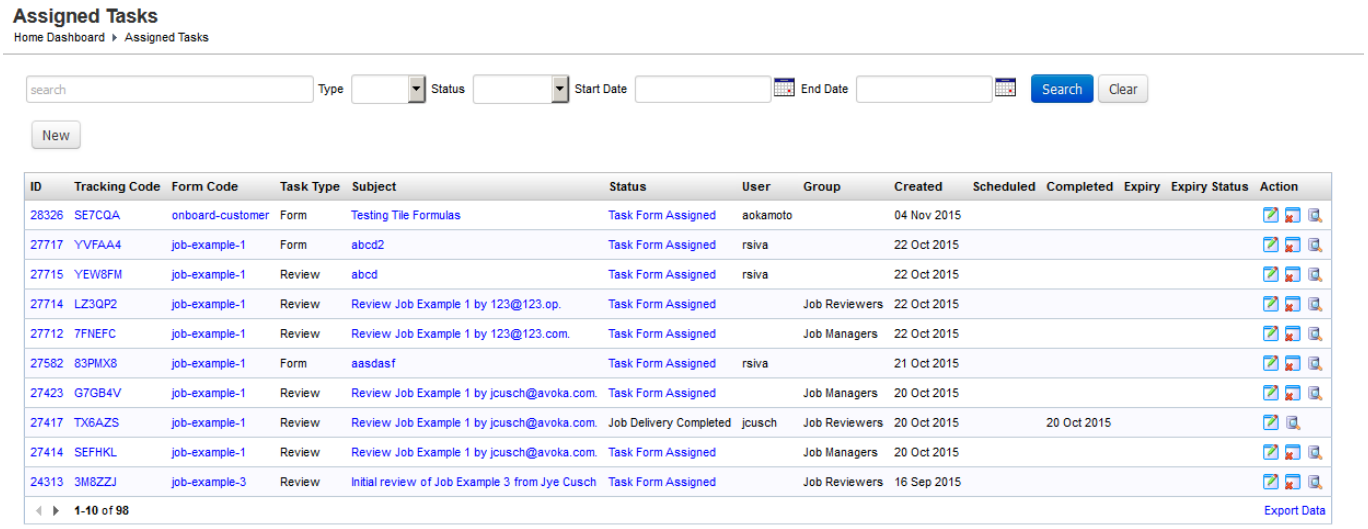
Finally a Form can have its Details section configured with the Organization's Delivery Details for Production and Test Delivery (This is impacted by the Test Mode flag in the Details screen).



Once a Form is submitted, the Transaction can be viewed from the Operations -> Form Transactions screen. This screen will show the Transaction Status which will include the Delivery Status, and also allow Delivery to be retried.



The Operations -> Assigned Tasks screen displays the lists of Assigned Tasks, the assignee and access to the transaction XML.

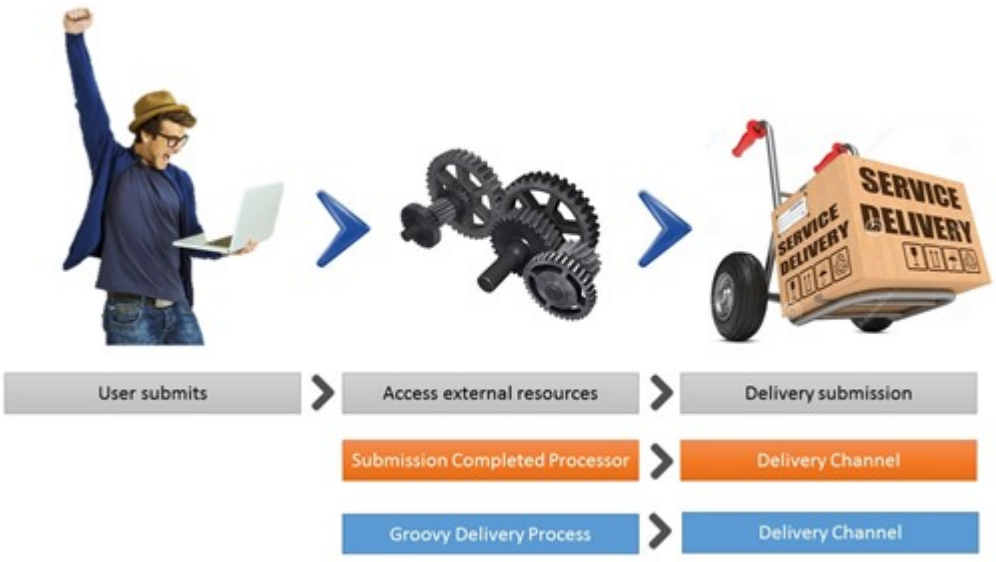


Next, learn how to view Transact functions.

Mix Standard Delivery Functions with Custom Groovy Processing

Journey Manager (JM) | Form Builder | Platform Developer | 5.1 & Higher

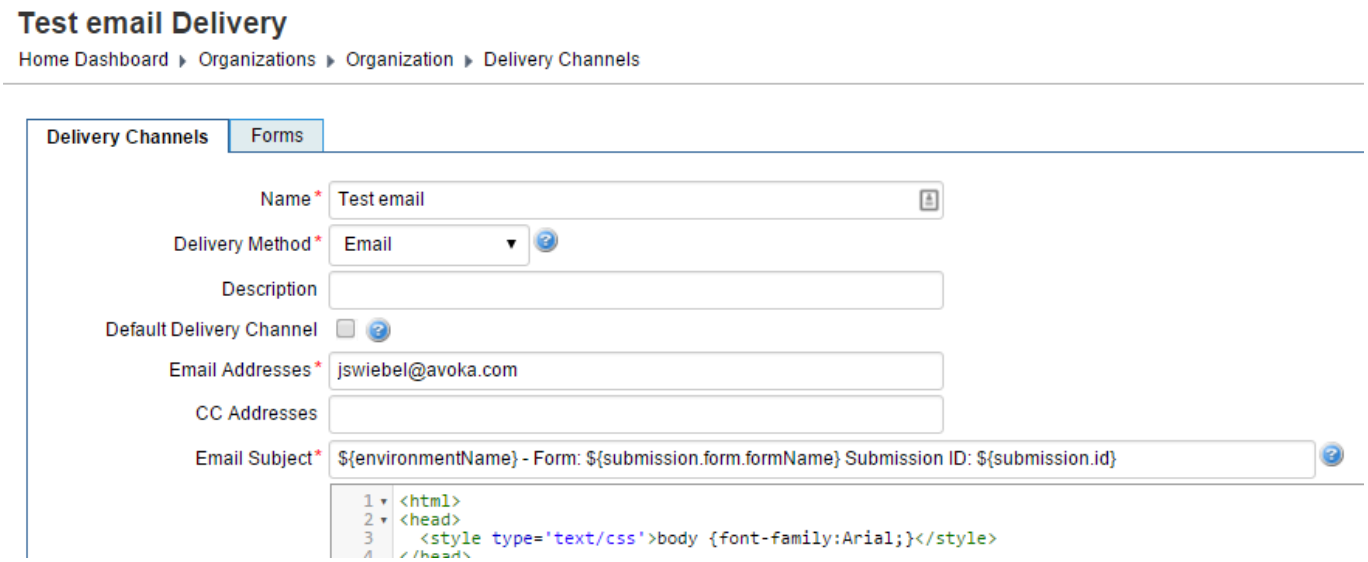
More than once will you be needing to access an external resource (eg web service, file share or perhaps an external mail server) to do something on submission. Typically you would create a Groovy delivery service and do whatever needs to be done in there. Sometimes this approach is a little problematic if your submission requires a delivery method other than Groovy, like Web Service, REST or Email. It's also possible that you already have an existing delivery channel that is used for other forms, but for this one form you need to do something extra before delivery.



The simple solution is to add the logic to the Submission Completed processor, your external resource will be accessed first and then the normal delivery process will kick. But what if the external resource is temporarily not available, the submission completed process will fail and unfortunately cannot be retried.

Let's leek at how the external resources is accessed from a Groovy delivery process, and how the process will change the delivery for that submission to the final (non-Groovy) delivery process.

For this we assume that we already have a non-Groovy delivery channel configured, called "Test Email", using the standard configuration.



Now, create a new Groovy delivery process, and in this you will be able to do whatever it is that needs be done using something external.

```
import com.avoka.fc.core.service.DeliveryResult
import com.avoka.fc.core.service.DeliveryResult.Status
import com.avoka.fc.core.dao.DaoFactory
import com.avoka.fc.core.dao.DeliveryDetailsDao
import com.avoka.fc.core.entity.DeliveryDetails
import com.avoka.fc.core.dao.ClientDao
import com.avoka.fc.core.entity.Form

// here starts the work
// step one, do your external resource stuff like call an external database
if (deliveryCheckpoint.doCheckpoint("Access External Resource", "Write db entry")) {
  try {
     // do the actual method call that does the thing
     storeToDb(.....)
     deliveryCheckpoint.completedCheckpoint("Access External Resource")
  } catch (Throwable e) {
    deliveryCheckpoint.errorCheckpoint("Access External Resource", e)
  }
}
// now make the submission available for Email delivery service
if (deliveryCheckpoint.doCheckpoint("Email Delivery", "Make submission available for
email delivery")) {
  try {
    // First get a new deliveryDetails object for the client (organisation) and the name
of the delivery service you'd like to use
    Form form = submission.getForm()
    DeliveryDetailsDao deliveryDetailsDao = DaoFactory.getDeliveryDetailsDao()
    DeliveryDetails deliveryDetails = deliv-
eryDetailsDao.getDeliveryDetailForClientAndName(
               form.getClient(), "Test email")

    // Set the delivery details on the submission
    submission.setDeliveryDetails(deliveryDetails)

    deliveryCheckpoint.completedCheckpoint("Email Delivery")
    // and return a ready status so that the submission will be ready for delivery via
the newly set delivery process
    return new DeliveryResult(Status.READY)
  } catch (Throwable e) {
    deliveryCheckpoint.errorCheckpoint("Email Delivery", e)
  }
}
```

Then, create a new Delivery Channel for your form's organisation, this channel needs to be of the type "Delivery Process" and make sure you configure it to use your newly created delivery Groovy process.

The last step is to change the delivery channel on your form to use the new Groovy one and not the old Test email service.

> Don't try to change the submission's Delivery details object.

If you try to manipulate the submission's delivery details be prepared for some funky result. Most likely the configuration of the delivery channel itself will be set in an inconsistent state. So don't do this.

If you want to re-trigger delivery, the submission will now go through the submission process twice, so be patient. Re-triggering the submission form TM will now only execute the newly email delivery process, not the original Groovy service.

Next, learn how to view Transact functions.

Invoking Dynamic Data Service from a Groovy Service

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

Use Case

A Dynamic Data service is normally used in a Composer Form to perform a lookup or AJAX calls to get information into the form from various integration endpoints. You may come across some situation where you want to leverage existing dynamic data service in your groovy script. For example, I have come across a situation where I have unformatted address string that I want to validate against Veda address service. Avoka Transact provides two dynamic data services for Veda address lookup integration. "Veda Address Search" service is used for type ahead address lookup to return possible matches and "Veda Address Detail" service is used to get more granular data for selected/provided address such as Unit No, street no, street name, street type, suburb, etc. I wanted to leverage existing "Veda Address Detail" service for two purpose. One to check if address string is a valid address in Veda address database (that is confirmed by if DPID is returned in the response) and second to get granular information from unformatted address string.

Solution

Firstly you will need to obtain a reference to the service with ServiceLocator class and cast it to IDynamicDataService to invoke required service by its name. As Dynamic Data services are intended to work with AJAX requests, it needs HTTPServletRequest object to invoke service. In our case, we cannot use originating request object as we need to pass our own request parameter (i.e. address string). As the request object is mutable and cannot add new request parameters to it, we need to create a mock request using the **`MockHttpServletRequest`** class to fulfill our requirement. The code snippet is shown below.

```
import com.avoka.fc.core.service.ServiceLocator;
import com.avoka.fc.core.service.form.IDynamicDataService;
import org.springframework.mock.web.MockHttpServletRequest;
import groovy.json.JsonSlurper;

def concatedAddress = "300 George St Sydney NSW 2000";

def vedaAddressDetailsService = (IDynamicDataService) ServiceLocator.getServiceForName
("Veda Address Detail");
def mockRequest = new MockHttpServletRequest();
mockRequest.addParameter("addressString" , concatedAddress);
def vedaResponse = vedaAddressDetailsService.getFormData(mockRequest, null, null);
def addressJson = new JsonSlurper().parseText(vedaResponse);
```

Next, learn how to view Transact functions.

Dynamic Data Lookup Service

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

Introduction to Dynamic Data

Dynamic Data Services was introduced as a mechanism to provide back-end querying capability to Composer generated forms. The Dynamic Data service is a capability that is initiated in a Composer generated form, and supplied by a Manager web application. This extends the reach and reuse of data and functionality to users and customers, saving time and money, while improving the end user experience in a secure and efficient way.

The Composer widget maintains its focus on end-user ease-of-use, while providing amazing power and flexibility to the Form filling experience. The widget can appear as a Button or a Link and includes a Test facility so you can mock up the return data in Composer before publishing to Manager.

On the Manager side, a simple to create Service Definition includes all the parameters required to expose a Groovy script which defines the activity of the Dynamic Data Service. Groovy is a language with a syntax very familiar to ActionScript and Javascript programmers alike. You can find out more about Groovy here http://-groovy.codehaus.org/, but this article assumes no prior knowledge of Groovy syntax or structure.

Handling Data

Dynamic Data can cater for single and multiple record return values, and as the Form designer, you need to know when to use the right output structures. Dynamic Data requires only the name of the Dynamic Data Service defined in Manager, but in reality needs an Input Block and an Output Block of widgets to be useful as a look up service. Data passed between the Composer generated SmartForm and Manager (and indeed the test data) is structured using the JSON format, a structure almost universally available on every platform known to mankind. While a complete description is beyond the scope of this article, JSON uses a simple key, value naming arrangement that can support deep nested structures and arrays.

Building the Composer Form

The example form referenced in this article uses an 'ABN Lookup Dynamic Data Service'.

Input Data

The example form takes an ABN number, collected using the ABN widget. The widget is placed inside a Block and configured with whatever other parameters are needed, like mandatory rules and visibility. The Block can be called anything you like and needs no other special configuration. Note that any fields in the Input Block will be automatically bound and transmitted to Manager as input data. This doesn't mean that your Dynamic Data Service has to use them, just be aware that they are there.

Output Data

This example returns one field called CompanyName to a standard block. Note that you don't have to have your output fields in a block of their own, they can be in a block with other fields. We'll see later how the binding of the output data occurs.

The Lookup Widget

In Composer, locate the Dynamic Data Button widget, typically found in theManager section of the widget palette, and drop it on the form where you need it to appear. The following dialog will appear.

Click the Browse buttons next to the Input and Result block entry areas to locate the parent Blocks of the Input and Output fields you defined earlier. Finally change the Button type as desired.

You don't need to change the TM Dynamic Data Service Lookup Name just yet, we'll be using the test mode to prove out our scenario first and then migrating it to Transaction Manager a little later on.

Save the dialog.

Testing the Lookup

Our next step will be to provide some test data so we can be assured of the JSON data format required to prove our scenario. Bring up the property editor for the TM Dynamic Data widget. In the Data section, check the Test Mode flag and enter the JSON formatted return value into the Test Data text area.

The format to use when formulating JSON notation is as follows:

```
{"KeyA":"ValueA","KeyB":"ValueB"}
```

which equates to:

KeyA = ValueA
KeyB = ValueB

This might appear in Composer as:

```
{"FieldNameA":"Field Value","FieldNameB":"Field Value"}
```

There are a few gotchas to be aware of:

Remember to use Field Names as the key, not labels. Don't be fooled into thinking the hierarchy uses field names, you'll need to look up the field names in the property editor for the field (right click the field in the Structure panel on the left, and select "Edit Properties...").
Values need to be formatted as appropriate for the field.

Here is an example of the JSON test data that I formulated for my specific test case. RegistrationNumber, VehicleDescription and RegistrationRenewalD are the three output fields of type text, text and date respectively.

```
{"CompanyName":"Avoka technologies"}
```

When the Dynamic Data Service component receives the results data from either test or Transaction Manager, it binds the associated value to the appropriate field by means of its name. It's this name binding that makes it possible to specify an output block that has other fields that aren't part of the result set.

After saving the properties and previewing the form, you should see your test data appear in output fields. If your data does not appear, or you get errors you'll need to resolve these before migrating the test data to Transaction Manager.

Building the Transaction Manager Dynamic Data Service.

Transaction Manager exposes a single endpoint for Dynamic Data Services, but determines which service to use by a single hidden field included in the input. Manager also secures the use of the Dynamic Data Service by embedding a hash key into the form when published to thwart possible misuse of the service data by unauthorised clients.

Dynamic Data Service Definitions use a combination of Service Definition configuration values and parameters to configure the service.

Copying an existing Dynamic Service Definition

Log in to Transaction Manager instance and select the System > Service Definitions menu item. Select Dynamic Data from the list of Service Types and click the search button. This gives you a list of the existing Dynamic Data Services already developed and you should go through some to get a feel for how they work. For the purpose of this example we're going to go ahead and copy an existing service and then modify it to suit our needs.

On the Service Definitions page, click the Copy button. On the Copy Service Definition page, select Dynamic Data as the Service Type, "Groovy Dynamic Data" as the Existing Service Name and enter a new name for the service.

Modifying the Dynamic Service Definition

Now that the service exists, technically we could call it from a published SmartForm but the output data in the script isn't tailored to our needs, so we'll fix that up first. Edit the Service Definition for the new Dynamic Data service and click the Service Parameters tab. Click on the groovyScript parameter name, this will invoke the text area editor which has a handy code syntax highlighter to help you spot errors.

The script below is used by the 'ABN Lookup Dynamic Data Service' on Transaction Manager Demo. It uses the wslite library to call the Australian Government's ABN lookup service.

```
import wslite.soap.*

import net.sf.json.*

import net.sf.json.xml.*

def abn = request.getParameter('abn')

def client = new SOAPClient('http://abr.business.gov.au/abrxmlsearch/AbrXmlSearch.asmx')
def response = client.send(     """<soapenv:Envelope xmlns:soapen-
v="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
abr="http://abr.business.gov.au/ABRXMLSearch/">

   <soapenv:Header/>

   <soapenv:Body>

      <abr:ABRSearchByABN>

         <!--Optional:-->

         <abr:searchString>""" + abn + """</abr:searchString>

         <!--Optional:-->

         <abr:includeHistoricalDetails>N</abr:includeHistoricalDetails>

         <!--Optional:-->

         <abr:authenticationGuid>avoka-7c4df64c-aa6c-41e5-abf1-0ffe200a8833</-
abr:authenticationGuid>

      </abr:ABRSearchByABN>

   </soapenv:Body>

</soapenv:Envelope>"""

)

String organisationName = response.ABRSearchByABNRe-
sponse.ABRPayloadSearchResults.response.businessEntity.mainName.organisationName;

def jsonResponse = new JSONSerializer().toJSON(["CompanyName": organisationName]);
```

```
return jsonResponse;
```

The important parts of the code to take note of are:

```
def abn = request.getParameter('abn')
```

This line searches the input block in the composer form for a field called 'abn', and saves it's value to a variable in the script called abn.

```
def jsonResponse = new JSONSerializer().toJSON(["CompanyName": organisationName]);
```

This line takes the organization name (saved in the variable 'organisationName', and inserts it into a JSON object, using "CompanyName" as a key. This will save the value returned by this script to a field named 'CompanyName' in the output block in the composer form.

Linking the Dynamic Data service in Transaction Manager to the Composer form.

The Composer form specifies the Dynamic Data service it will use by name, in the 'Dynamic Data Service Name' field. When publishing the form, remember to clear the 'Test Mode' button, or the service will not be called.

A Final Word - Testing Testing Testing

OK, so there's a good chance that when you got it all hooked up it didn't work. For debugging I highly recommend FireFox and Firebug or Chrome and it's Dev Tools for analysing page behavior. In the screenshot you'll notice there's a section at the bottom of the page where you can see the POST request to the Manager servlet that services the Dynamic Data Services requests. The response from that POST includes the data as formatted in my Test Mode data. Yours should too.

Bonus Points - Repeating blocks of results

Thankfully, the Composer developers catered for multiple results sets by allowing data to be returned in a JSON array. This is easily accomplished by wrapping the JSON response data in an array spec and separating records with commas, like so:

```
[{"record1":"value1"},{"record2":"value2"}]
```

On the Composer side, the output block must be Repeatable Content to get the Repeatable-Block field definition. Again, field binding is done by name with each record in the JSON array creating a new repeat section. Do be careful if switching between the two types (Repeating and Standard Blocks) since the Output Block specification differs depending on the type. In my example the Output Block specification for a single result is:

```
../VehicleDetails
```

If I were to change the block to Repeatable Content in response to some change in development, I must ensure my Output Block specification is updated and appears as follows:

```
../VehicleDetails[*]
```

Conclusion

Dynamic Data Services is a very important and strategic feature for Transact providing a mechanism to further enrich the user experience by including interactivity between Composer and Transaction Manager. The range of possible applications for this capability are untold with the flexibility and power in the current design.

Next, learn how to view Transact functions.

Check a Form Request IP Address

Journey Manager (JM) | Form Builder | Platform Developer | 5.1 & Higher

It may be necessary in certain instances to determine the IP address of the user who has requested a form, for example, so that internal users can view a different version of a form than external users, or so that users in different countries or areas see different form content.

The form must be pre-populated with either the IP address of the requesting user, or an indicator of what type of user they are, which has been determined using the IP address in a pre-populated Groovy script.

You can do this in Manager:

1. Create the Groovy prefill service:

```
// Script parametes include: form (Form), formRequestParams (Map<String, Object>),
request (HttpServletRequest), schemaSeed (Document), userAccountId (String), ser-
viceDefinition (ServiceDefinition), serviceParameters (Map<String, String>)
// Script return : Form XML data string value or return empty value and update the
schemaSeed Document parameter
def returnXml = '''
<prefillData>
    <value1>''' + request.getRemoteAddr() + '''</value1>
    <value2>''' + request.getHeader("X-Forwarded-For") + '''</value2>
</prefillData>
''';
return returnXml;
```

2. Create a Service Parameter containing the groovyScript. The groovyScript includes a request parameter. The request parameter is a HttpServletRequest Java object, which contains information about the request. The IP address that the request came from can be accessed using the following code:

```
request.getRemoteAddr()
```

Depending how the TransactionManager server is set up, this may not always give you the user's address. For example, tm.demo.avoka.com sits behind a load balancer, which catches the user's HTTP request and uses it to create a new one. Using the above code will return the IP address of the load balancer, rather than the user. In this instance, the user's IP address must be accessed through the HttpServletRequest object's header, using the code:

```
request.getHeader("X-Forwarded-For")
```

3. Create the sample input XML. In the form version data config, go to the input XML Prefill Mapping tab click Edit Input XML

4. Create the XML Mapping. On the same tab click the Edit XML Mapping button
5. Update the form version to include the Form Prefill Data Service.

Next, learn how to view Transact functions.

Groovy Service Tips and Tricks

Journey Manager (JM) | Form Builder | Platform Developer | 5.1 & Higher

Many services and functions in Manager are written in **Groovy**[1]. This is because it gives you extra flexibility in implementing a wide range of customer requirements that often can be rather complex. To make it even easier for you, we provide access to most Manager functionality via Fluent APIs, which are available in the SDK for each product release,

When working with Groovy using our API, you may face some typical problems, so we have captured some useful Groovy tips and tricks below to help you solve them.

Get the version of Groovy

```
print GroovySystem.getVersion()
```

Get Date widget into a W3C dateTime

The W3C has a specification for date time formats which has resulted in the sublime XMLGregorianCalendar Java class. While Groovy provides some convenience mechanisms to work with Java types such as Date, an it's relatively easy to use format specifiers to produce compatible dateTime formats, I prefer the brute force code approach. Below is an example that takes a date from a Composer widget and converts it into W3C compatible dateTime via Calendar objects

```
javax.xml.datatype.DatatypeFactory.newInstance().newXMLGregorianCalendar(Date.parse
("yyyyMMdd","19710517").toCalendar())
```

Here's a similar method that takes a date object and produces the compatible dateTime format

```
def StringBuffer timestamp = new StringBuffer(
       new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZ").format(new Date())
)
```

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

Manager included libraries

An easy way to check this is go to JBoss module folder for groovy, such as **C:\avoka\transact\manager\server\modules\org\codehaus\groovy\main**.
The file module.xml specifies like the following:

Manager 3.5

```
<module xmlns="urn:jboss:module:1.0" name="org.codehaus.groovy">
        <resources>
            <resource-root path="groovy-all-2.1.0.patch.jar"/>
            <resource-root path="groovy-wslite-0.7.1.jar"/>
            <resource-root path="http-builder-0.5.2.jar"/>
        </resources>

    <dependencies> <module name="com.avoka.core" />
            <module name="com.microsoft.sqlserver" />
            <module name="com.mysql" /> <module name="com.nuodb" />
            <module name="com.oracle" /> <module name="javax.api"/>
            <module name="javax.servlet.api"/>
            <module name="net.sourceforge.json" />
            <module name="net.sourceforge.saxon" />
            <module name="net.sourceforge.nekohtml"/>
            <module name="org.apache.camel.core" />
            <module name="org.apache.camel.spring" />
            <module name="org.apache.commons.codec" />
            <module name="org.apache.commons.collections" />
            <module name="org.apache.commons.lang" />
            <module name="org.apache.commons.logging" />
            <module name="org.apache.commons.io" />
            <module name="org.apache.httpcomponents"/>
            <module name="org.apache.log4j"/>
            <module name="org.apache.xml.xml-resolver"/>
            <module name="org.junit"/> <module name="org.opensaml" />
            <module name="org.slf4j" /> <module name="sun.jdk" />
            <module name="system" />
        </dependencies>
    </module>
```

The dependency section and resources section are libs that your groovy scripts can use.

Groovy script body building

You can use the following code template to develop a SOAP message in the Groovy Script Console in Manager

```
import wslite.soap.*
    def body = new SOAPMessageBuilder().build {
        version SOAPVersion.V1_2
        encoding('UTF-8')
        envelopeAttributes(
            'xmlns:_04':'http://cai.dataaction.com.au/Cai/Member/New/2011_04_15/',
            'xmlns:dat':'http://schemas.datacontract.org/2004/07/DataAction.Cai.Wcf',
            'xmlns:new':'http://cai.dataaction.com.au/Cai/Member/New/'
        )
        header{
            Action('soap-env:mustUnderstand':1,
xmlns:'http://schemas.microsoft.com/ws/2005/05/addressing/none','http://cai.dataaction.co-
m.au/Cai/Member/New/2011_04_15/MemberContro...)
            To('soap-env:mustUnderstand':1,
xmlns:'http://schemas.microsoft.com/ws/2005/05/addressing/none','https://data-dfv-
cai3.cai.dataaction.com.au:8443/cai/member/new/2011...)
        }
        body{
            '_04:CreateNewPersonalMember'{
                '_04:newMember'{
                    'dat:Organisation' ('Avoka')
                    'dat:RequestId'(UUID.randomUUID())
                }
            }
        }
    }
    print body
```

Capturing Groovy Script Output

Capturing Groovy Script Output with SystemOuptutInterceptor

```
import groovy.ui.SystemOutputInterceptor
    ...
    def outputString = ''
    // a simple closure to add the iterator to the string; returning false diverts the
output from system.out
    def interceptor = new SystemOutputInterceptor({ outputString += it; false})


    interceptor.start()
    // ...do some Groovy code that generates output
    interceptor.stop()


    print outputString
```

Use a Builder and a Map to Conveniently Return Dynamic Data

Dynamic Data Services in Avoka Transact require data ina specific format that can be a little tricky to format. It uses an array around Json objects to parse and fill repeating groups of output fields in the form. That format looks like this:

```
[{"Name":"Value"},{"Name":"Value"}]
```

But that can be a real pain to get right if you're trying to do it using string concatenation and placeholders, error prone at best. This example produces a JSON object using a JsonBuilder with a Closure. Wraps it nicely inside an array and then returns as a string (to the Dynamic Data Service)

```
JsonBuilder + Map = Dynamic Data
    def json = new JsonBuilder({
            Name(Value)
        })
    def arr = [json.toString()]
    return arr.toString()
```

Mocking up Web Service SOAP Return Data

Developing services on Transaction Manager is not the friendliest or easiest of experiences simply because it's intent is to provide a lean administration system, not a development enviornment. Yet there are times when the only way to get at the values you need are on Transaction manager, especially when connected to remote, thrid party systems. This is a simple example top mock up a web service response which you can then use to develop the response action like extracting data.

Mocking up a SOAP Response

```
def soapMessageText = '''
    <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" xmlns:a-
a="http://www.w3.org/2005/08/addressing">
        <s:Header>
            <a:Actio s:mustUnder-
stand="1">http://cai.dataaction.com.au/Cai/Member/New/2011_04_15/MemberContro...
                CreateNewPersonalMemberResponse</a:Action>
        </s:Header>
        <s:Body>
            <CreateNewPersonalMemberResponse xmlns-
s="http://cai.dataaction.com.au/Cai/Member/New/2011_04_15/">
                <CreateNewPersonalMemberResult xmlns:b-
b="http://schemas.datacontract.org/2004/07/DataAction.Cai.Wcf"
                    xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
                    <b:ReceiptNumber>8071296</b:ReceiptNumber>
                    <b:RequestId>d9da9951-1a6f-4f49-bfc0-9d00fc44f491</b:RequestId>
                    <b:Body xmlns:c="http://cai.dataaction.com.au/Cai/Member/New/">
                        <c:MemberNumber>22562764</c:MemberNumber>
                    </b:Body>
                </CreateNewPersonalMemberResult>
            </CreateNewPersonalMemberResponse>
        </s:Body>
    </s:Envelope>

'''
    import wslite.soap.*
    import wslite.http.*
    def httpRequest = new HTTPRequest()
    def httpResponse = new HTTPResponse()
    def soapEnvelope = new XmlSlurper().parseText(soapMessageText)

    response = new SOAPResponse(httpRequest:httpRequest,
                                httpResponse:httpResponse,
                                envelope:soapEnvelope,
                                text:soapMessageText)

    print response.en-
velope.Body.CreateNewPer-
sonalMemberResponse.CreateNewPersonalMemberResult.Body.MemberNumber
```

Groovy is a very powerful and dynamic language, and there are several different ways to work with and manipulate objects of different types.

You also need to be aware that Transact Services implement static type checking, in order to improve error checking and security. This creates some restrictions in the coding styles you can use.

Typed Objects

The best way to represent objects is as classes. (Some will no doubt have alternate opinions or a different use case, and this is an arguable point.)

Some simple code looks like this:

```
class Customer {
    String first
    String last
}

Customer customer = new Customer()
customer.first = "Mary"
customer.last = "Smith"

assert "Mary" == customer.first
```

Using classes is a little more work, because you have to define the class. And you have to modify it if there are any changes to the data structure. (This is compared to collections, which are inherently dynamic.)

But the advantages are:

A very simple syntax: "customer.first"

If you are using an IDE (you should be) then the IDE will give you code completion when you are typing

Safety - if you accidentally type: "customer.furst", you will get an error. In a dynamic Groovy collection, you have just created a new entry called "first".

JSON

Groovy gives us a very easy way to convert our object to a string representation - JSON.

```
import groovy.json.JsonOutput

String str =  JsonOutput.toJson(customer)
println str
println JsonOutput.prettyPrint(str)
```

This will produce: **`{"first":"Mary", "last", "Smith"}`**

The pretty version will produce a more easily human readable form including line breaks and indentation.

Collections and JSON

You can alternately use collections - maps and lists - as an alternative to statically defined classes. Let's try this by converting our JSON string into a collection.

```
import groovy.json.JsonSlurper

def slurper = new groovy.json.JsonSlurper()
def customerMap = slurper.parseText(str)
```

This will convert the JSON sting into a collection. In this case, it's just a simple map, but if the JSON was more complicated, the map could contain other maps and lists, instead of just strings.

In regular Groovy, we can refer to the elements within the collection like this:

```
println customerMap.first
```

However, because Manager implements static type checking, you can't use this syntax directly. Because Groovy doesn't know what type "customerMap" is, it assumes it's just a regular object, which doesn't have a "first" attribute.

To fix this, you can either using casting (which gets a bit tedious, especially as your object graph gets deeper), or use the following alternate syntax:

```
println customerMap["first"]
```

Create an Object Based on a Collection

Groovy provides a very useful default constructor for any object based on a collection. This means that if we choose to go with typed classes, we can easily convert from our collection to a typed object.

The code below shows how we can get from a JSON string to a collection, and from a collection to a typed object. You could of course do this in a single line of code, but we've expanded for clarity.

```
import groovy.json.JsonSlurper

def slurper = new groovy.json.JsonSlurper()
def customerMap = slurper.parseText(str)
Customer customer2 = new Customer (customerMap)
```

Next, learn how to view Transact Functions.

HTTP Session Object Pattern

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

You can store and retrieve data against the http session object in the usual way. However, the session object is (by definition) temporary/volatile, and may "go away" under several circumstances, such as save-and-resume. Data stored in the session object cannot be easily viewed in Manager, and therefore makes debugging difficult. Don't mess with the session object itself, because it is used internally by Manager. For example, the following code will cause all sorts of problems:

```
request.getSession(false)?.invalidate() // DON'T do this!!!!
```

As a general recommendation, you should strongly consider storing and retrieving data using the Manager built-in services.

Next, learn more about the server-side persistence pattern.

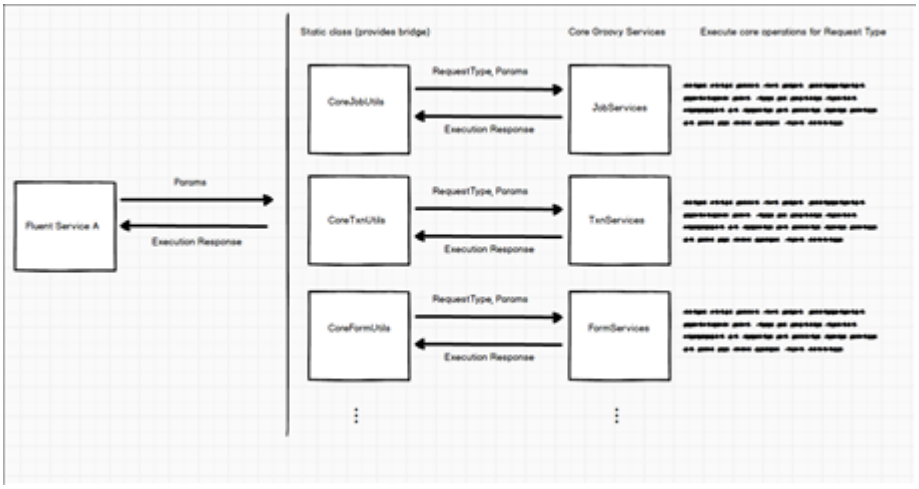
Invoke Core Services Pattern

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

Fluent SDK provides great ability to execute Fluent APIs efficiently. Most of your project requirements should be served through Fluent APIs. However, you may find a situation where some of the operations that you were able to do earlier (prior Transact 17.10) using Core APIs are no longer supported by Fluent API. You should reach out to product team to discuss these requirements and create a request to add the support in Fluent API in future release. This may take some time due to product release cycle and your project cannot wait till then. This article explains the design pattern you should use to invoke Core APIs in your project and eventually replace when product team delivers that functionality with new release.

Let's take a use case for explaining this design pattern. In this example your project has a fluent service A needs to execute certain operations such as updating "Job Status" that is only available through Core API (at the time of writing this article). Fluent service does not allow you to import core classes. So you have to create a core groovy service by invoking "svc-scaffold-legacy" ant target in your sdk.

As illustrated in the diagram below, Fluent service may pass parameters that is required by core service to execute core operations. I have split core operations by categories such as Job, Transaction, Form etc as you may need certain operations under each category which are not available in fluent API yet.



Step 1

In this example below, I have created three static classes (which will be added dependencies in Fluent service definition). You can add more static classes as you need for other categories of Transact platform.

- CoreJobUtils - Provides all helper methods for collaboration job object (code is in Step 5)
- CoreTxnUtils - Provides all helper methods for Transaction object
- CoreFormUtils - Provides all helper methods for Form object.

Create these core services under projects "utils > core" package for better isolation. e.g. com.avoka.project.utils.core

Step 2

For consistency we will need to create a generic POJO that can be used as a wrapper for the response from core service. Let's called it "ExecutionResponse" groovy class and put that under "vos" package.

```
package com.avoka.cnb.cnb1.vos;

public class ExecutionResponse {

    public static final String STATUS_SUCCESS = "Success";
    public static final String STATUS_ERROR = "Error";

    private String status;
    private String errorMessage;
    private String serviceResponse;

    public ExecutionResponse() {}

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public String getErrorMessage() {
        return errorMessage;
    }

    public void setErrorMessage(String errorMessage) {
        this.errorMessage = errorMessage;
    }

    String getServiceResponse() {
        return serviceResponse
    }

    void setServiceResponse(String serviceResponse) {
        this.serviceResponse = serviceResponse
    }
}
```

Step 3

Fluent service needs to include required static class and ExecutionResponse in service definition so that it's available during execution.

```
{
    "name": "groovyScript",
    "filePath": "CNBProductSelectorPreprocessorAng.groovy",
    "fileIncludes": [
        "com/avoka/cnb/cnbl/utils/core/CoreJobUtils.groovy", "com/avoka/cnb/cnbl/vos/ExecutionResponse.groovy"
    ]
}
```

Step 4:

Create a core service by executing "svc-scaffold-legacy" target. Let's call it "CNB Job Services". This single groovy service will execute all core API operations by looking at "RequestType" request parameter which will be passed by Utils class in Step 4. This way we dont need to create multiple core groovy services for each missing operation. This service is responsible for creating "ExecutionResponse" object with relevant attribute values such as Status, error message and service's raw response. This will be then forwarded back to Utils class and eventually back to caller which is Fluent service.

```
package com.avoka.cnb.cnb1.services.core.job

import com.avoka.cnb.cnb1.vos.ExecutionResponse;

/* Generic Groovy script to perform all Job related operations that currently not sup-
ported by Fluent

   Script parameters include:
       parameters : Map<String, String>
       serviceDefinition : com.avoka.fc.core.entity.ServiceDefinition
       serviceParameters : Map<String, String>
       args : List<String>
       job : com.avoka.fc.core.entity.Job
       jobAction : com.avoka.fc.core.entity.JobAction

   Script return:
       script result
*/
import com.avoka.core.groovy.GroovyLogger as logger
import com.avoka.fc.core.dao.DaoFactory
import com.avoka.fc.core.service.job.impl.JobControllerService
import com.avoka.tm.util.Contract

ExecutionResponse executionResponse = new ExecutionResponse();

String status = ""
String errorMessage = ""
String requestType = parameters["RequestType"];
Contract.notNull(requestType, "Request Type");

Map<String, Object> executionParams = parameters["ExecutionParams"];
String jobReferenceNumber = executionParams["JobReferenceNumber"];

def requestedJob = DaoFactory.getJobDao().getJobForReferenceNumber(jobReferenceNumber);
if(requestedJob) {
    if (requestType == "STATUS_UPDATE") {
        String newStatus = executionParams["JobStatus"];
        requestedJob.setStatus(newStatus);
        new JobControllerService().commitChanges();
        status = ExecutionResponse.STATUS_SUCCESS;
    }
} else {
    status = ExecutionResponse.STATUS_ERROR;
    errorMessage = "Unable to find a job with reference number ${jobReferenceNumber}";
```

```
}

executionResponse.setStatus(status);
executionResponse.setErrorMessage(errorMessage);

return executionResponse
```

Step 5

Add a static method in "CoreJobUtils.groovy" class to invoke core service to update Job status. This service is responsible for updating job status and return back "ExecutionResponse" object with execution status.

```
packagecom.avoka.cnb.cnb1.utils.core;
importcom.avoka.cnb.cnb1.vos.ExecutionResponse;
importcom.avoka.tm.svc.GroovyServiceInvoker;
importcom.avoka.tm.vo.SvcDef;
publicclassCoreJobUtils {
    /**
     * Updates the job status
     * @param svcDef Service that requests this activity to be added to the package life-
cycle
     * @param jobReferenceNumber Collaboration Job Reference Number
     * @param jobStatus Job Status
     * @return ExecutionResponse object contains status and error message
     */
    publicstaticExecutionResponse updateJobStatus (SvcDef svcDef, String jobRe-
ferenceNumber, String jobStatus) {
        Map<String, Object> reqParams = newHashMap<>();
        reqParams.put("RequestType", "STATUS_UPDATE");
        Map<String, Object> executionParams = newHashMap<>();
        executionParams.put("JobReferenceNumber", jobReferenceNumber);
        executionParams.put("JobStatus", jobStatus);
        reqParams.put("ExecutionParams", executionParams);
        Object activityResponseData = newGroovyServiceInvoker()
                .setServiceName("CNB Job Services")
                .setClientCode(svcDef.clientCode)
                .invoke(reqParams);
        return(ExecutionResponse) activityResponseData;
    }
}
```

Step 6

Now, let's invoke the utility method in the Fluent service itself that allows us to update job status by invoking actual core groovy service.

```
ExecutionResponse executionResponse = CoreJobUtils.updateJobStatus(svcDef, job.ref-
erenceNumber, Job.STATUS_CANCELLED);
if(executionResponse.status == ExecutionResponse.STATUS_SUCCESS) {
    logger.info "Cancelling job "+ job.referenceNumber + " due to edit package event"
} else{
    logger.error "There is a problem with cancelling job ${job.referenceNumber}. Error:
${executionResponse.errorMessage}";
}
```

Benefits

- Provides a controlled approach to isolate core and fluent services usage in your project.
- Utility classes lists all missing methods that eventually you need a support in Fluent API. This can also help you to find a usage of those methods in your project to understand the impact of a change. If you directly invoking core groovy service from fluent service, you have to rely on text search for service name in your project which can be messy in a big complex project.
- As we get fluent API support with an upgrade, you can find all usages and replace the code with Fluent API easily. This can also help you understand which area you have to focus on testing in order to make sure things work as expected. You can then delete that method from Util class. Ideally we don't want any methods in that class as support should be through Fluent.
- Fluent SDK combines all file includes in service definition into a single file during application deploy to Manager. Splitting into multiple Utils class provides less footprint in actual service output. Not something you need to worry about but still better to have less duplicate and lengthy code across file which provides better readability if someone viewing from Manager admin console.

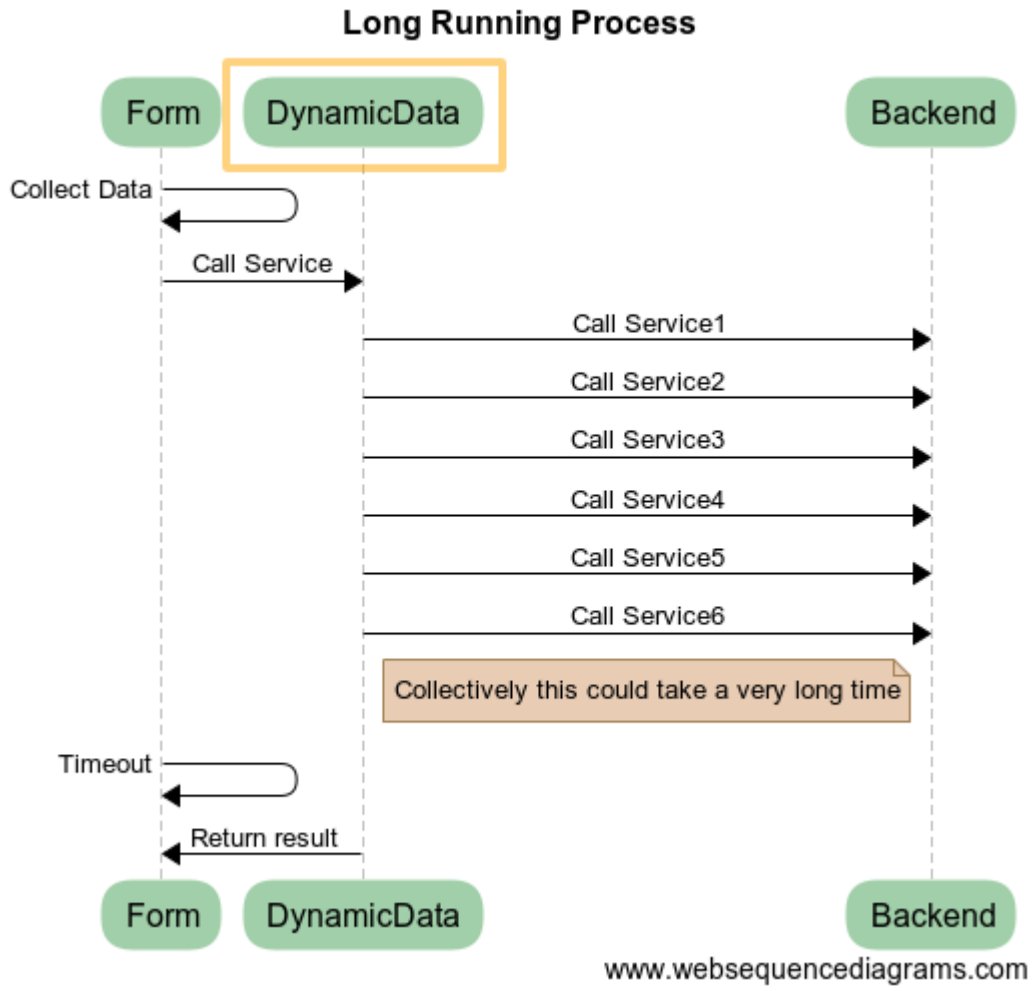
Next, learn more about the server-side persistence pattern.

Long Running Service Pattern

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

Sometimes you may need to invoke a Dynamic Data Service from a form, and you know that it may take some time. Perhaps there is a back-end service that is slow, or perhaps you are calling several different services in order to get your answer.

It might look something like this:



There is two problems with this:

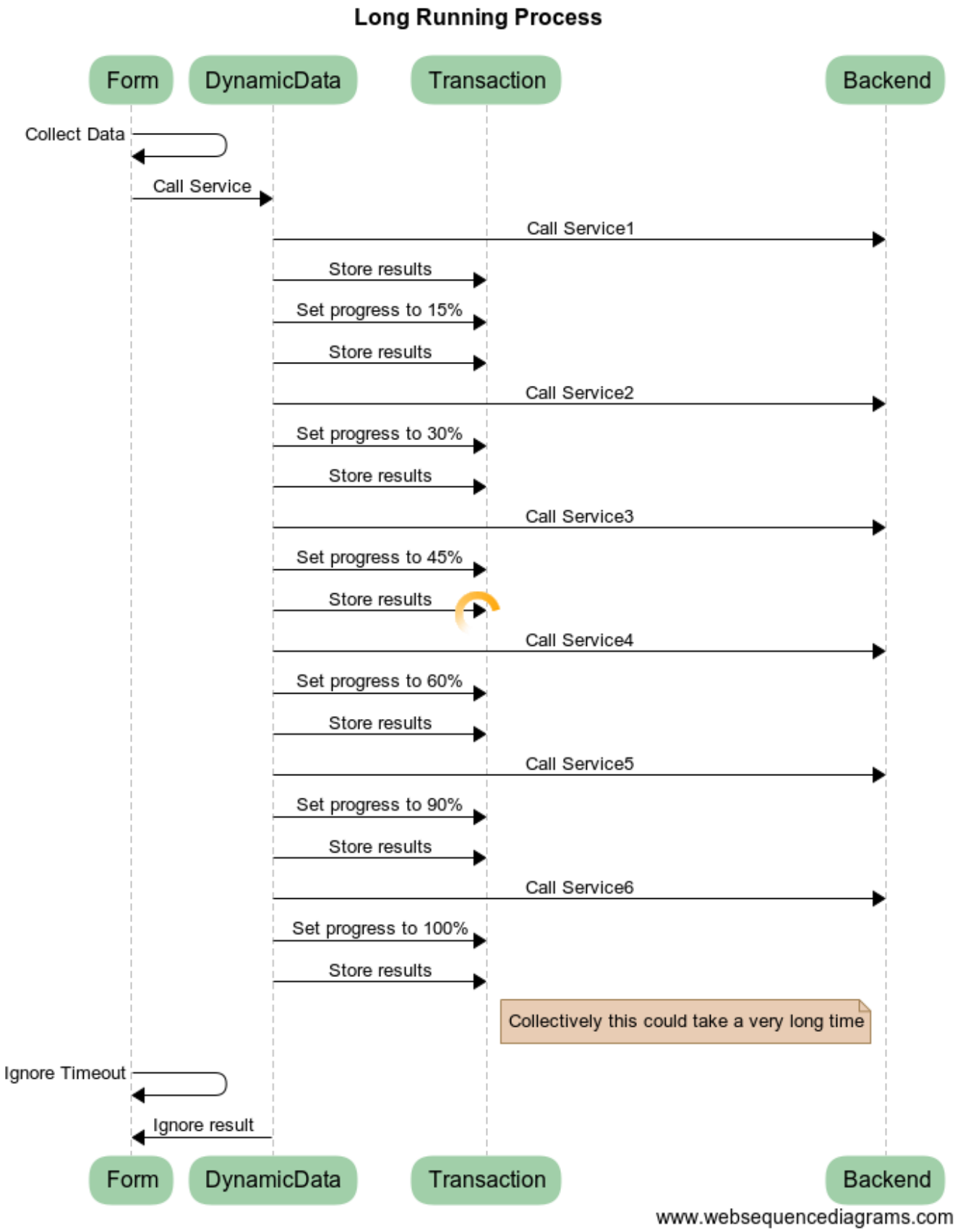
- Both Maestro and the Transact Manager web server (usually Apache) have a built-in service timeouts. If you exceed this timeout, you will receive an error in your application. This means that you will never actually receive a result.
- While the service is running, the user has no feedback on what is going on.

One solution would be to increase the timeouts. However, the timeouts are there for a reason - they are there to protect the application server from having a large number of long-running HTTP connections open. Too many open connections, and the server will eventually run out of resources.

Here are two patterns that may be helpful to work around this problem.

Progress Polling

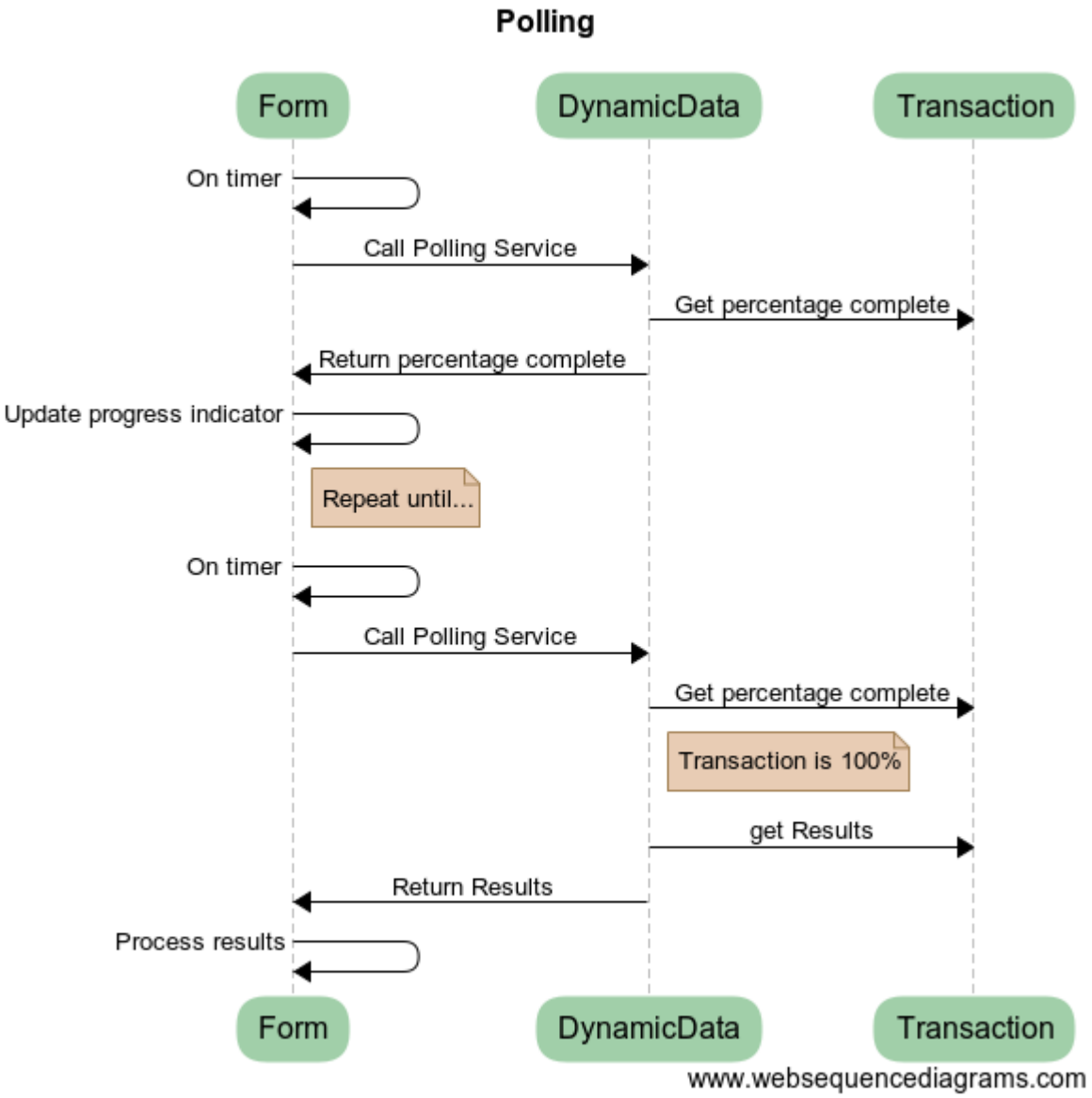
One of the useful things about the web (and Manager) is that services are all asynchronous. So we can modify the above sequence to look something like this:



We've basically made three changes:

1. Update a transaction property to represent the percentage progress of the long running process.
2. Store the intermediate results of the service calls in a transaction property (or multiple properties).
3. Ignore both the timeout and the result of the initial Dynamic Data Service call.

Then, in parallel, we will implement another Dynamic Data Service call. This will be on a timer, and will call the Manager server periodically.



The form will call the Polling Service periodically.

The polling service will check the percentage complete, and either;

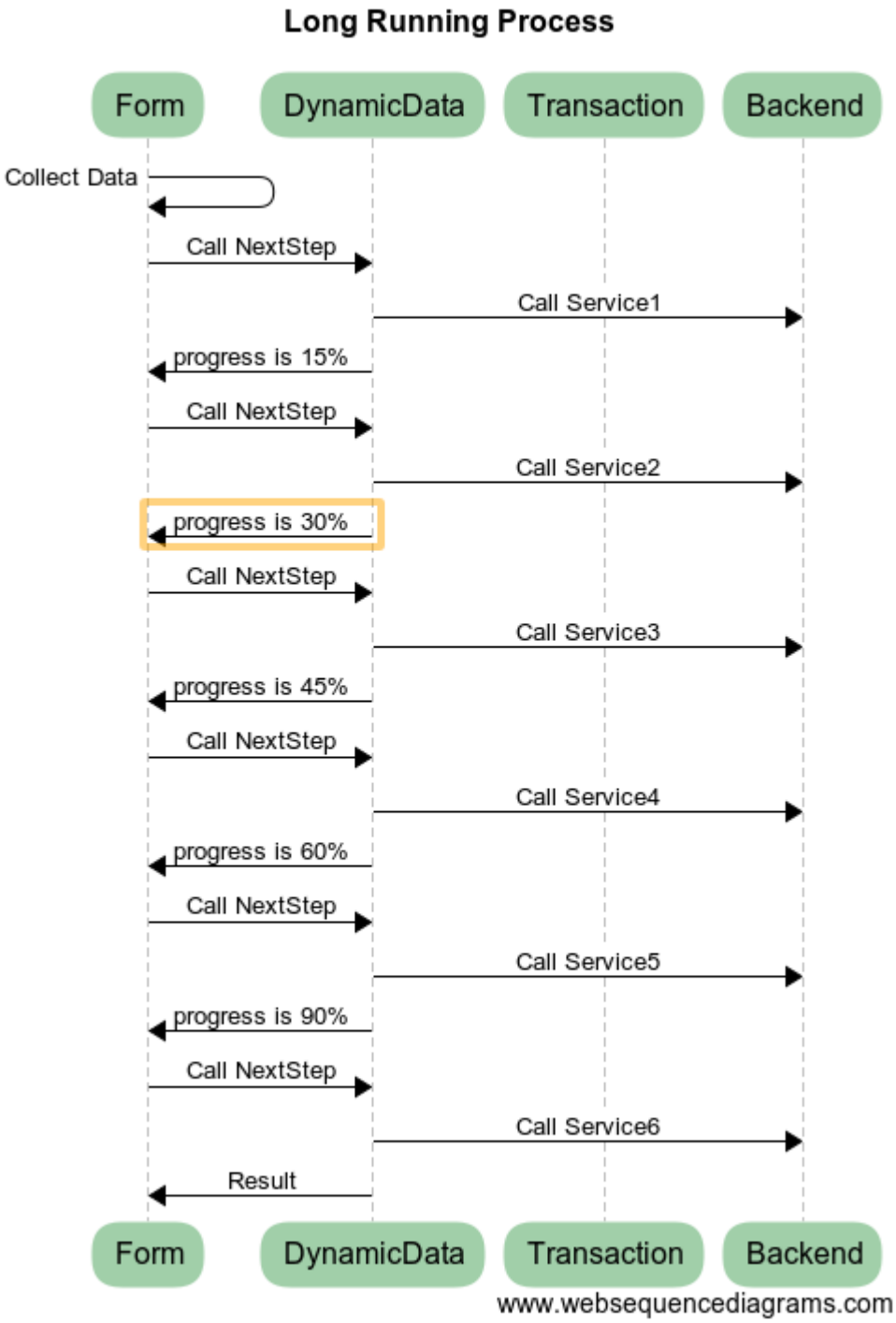
- Return the percentage complete, so that the form can update the progress meter (or similar) (This is shown in the top half of the diagram above.)
- If the process is complete, return the results, so the form can use them. (This is shown in the bottom half of the diagram above.)

This solves both problems.

Blind State Machine

Another way to solve this is to use a blind state machine.

This breaks up the sequence of execution, but does so in a way that the Form is unaware of the details or logic that is occurring on the server. This helps to make sure that the form cannot be compromised by a malicious user.



In this pattern:

- We break up the calls to ensure that none of them exceeds the timeout value.
- In the same way as the above solution, we are giving feedback to the form that allows it to update a progress meter.
- We are careful not to expose the internals of the state machine, or how its logic works. All we are doing is calling "NextStep" until we get back a result rather than a progress indicator.

Enterprise Service Bus

A third approach is to use an Enterprise Service Bus or similar architecture. In this case, we would use message queues or similar patterns to "kick off" a long running process. We would then using the polling pattern to discover when the process is complete.

Conclusion

Both the patterns described should be fairly easy to implement, and solve the required problem. The progress polling pattern is arguably superior, because it completely hides the server implementation from the Form.

Next, learn how to view Transact functions.

Multi-Step Groovy Delivery Process

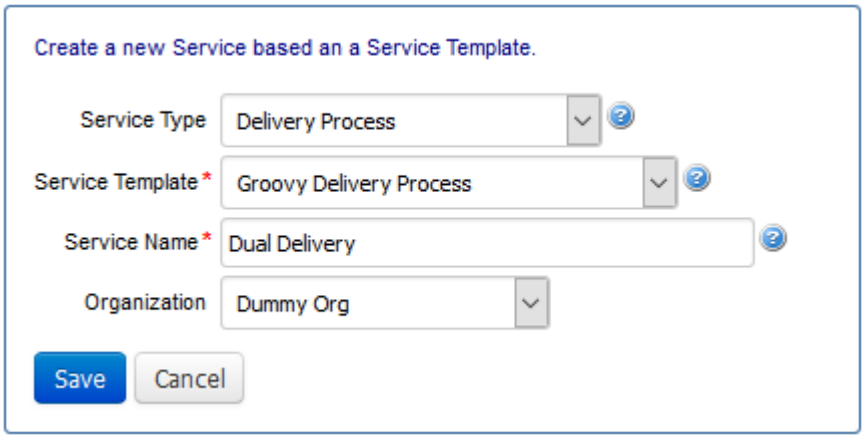
Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

This guide provides an introduction and how-to for creating multi-step groovy delivery processes.

Transact Manager already provides the capacity for multi-step delivery processes through the use of delivery check points in the Groovy Delivery Process. However for larger or more complicated delivery methods it may be easier to separate delivery steps into individual delivery processes. This has the added advantage of allowing forms to call only the delivery method they need. For forms that require these multi stage delivery methods it is possible to configure a groovy delivery process to call other delivery processes. This allows form developers to create a type of landing point for form submissions that can branch off to any number of different delivery methods, providing a wide range of functionality for accepting form submissions.

Basic Groovy Delivery Process

First lets make a basic Groovy Delivery Process. In TM go to the service definitions page and click new and fill out the new service information.



The basic groovy delivery process will have the following code provided for you:

```
if (deliveryCheckpoint.doCheckpoint("Checkpoint A", "Description...")) {
	try {
		// TODO: provide your custom delivery code
		deliveryCheckpoint.compietedCheckpoint("Checkpoint A")
	} catch (Throwable e) {
		deliveryCheckpoint.errorCheckpoint("Checkpoint A", e)
	}
}
if (deliveryCheckpoint.doCheckpoint("Checkpoint B", "Description...")) {
	try {
		// TODO: provide your custom delivery code
		deliveryCheckpoint.compietedCheckpoint("Checkpoint B")
	} catch (Throwable e) {
		deliveryCheckpoint.errorCheckpoint("Checkpoint B”, e)
	}
}
```

When this process is called, after a form submission is made, it will first run the code in the try/catch block labled Checkpoint A and then the code in the next try/catch block labled Checkpoint B. If the code in the try catch block should fail it will retry this code again in 15 mins.

The delivery retry delay and the number of retry attempts can be adjusted in the service parameters of the delivery process, the fields are retryDelayMins and maxDeliveryAttempts.

In this code even if Checkpoint A should fail Checkpoint B will still make an attempt to be delivered, this is known as a parrallel delivery method. You can also setup the delivery process to be serial, this is useful if you have delivery processes that rely on each other. The groovy code for a serial delivery looks like this:

```
if (deliveryCheckpoint.doCheckpoint("Checkpoint A", "Description...")) {
       try {
               // TODO: provide your custom delivery code
               deliveryCheckpoint.compietedCheckpoint("Checkpoint A")
       } catch (Throwable e) {
               deliveryCheckpoint.errorCheckpoint("Checkpoint A", e)
       }
}
if (deliveryCheckpoint.isCompleted("Checkpoint A") &&
    deliveryCheckpoint.doCheckpoint("Checkpoint B", "Description...")) {
       try {
               // TODO: provide your custom delivery code
               deliveryCheckpoint.compietedCheckpoint("Checkpoint B")
       } catch (Throwable e) {
               deliveryCheckpoint.errorCheckpoint("Checkpoint B”, e)
       }
}
```

This code won't call the second checkpoint until the first has been successfully completed. For more information, see the services guide.

Dual Delivery

It is possible,using groovy script, to call another process from within your delivery process, as shown below:

```
IDeliveryProcessService deliveryService = (IDeliveryProcessService) Ser-
viceLocator.getServiceForName('[SERVICE_NAME]'))
deliveryService.deliverSubmission(submission, new DeliveryDetails())
```

These two lines of code first locate the service by its name, then calls the service by passing the submission object and delivery details to the service. Using this code in a groovy delivery service it is possible to create laning point for submissions that can be delivered in a variety of methods based on any number of factors, such as submission data or other successful delivery methods.

An example case:

A form is submitted that calls a basic delivery process 'logger', which logs the form submission data for review by a backend system. The form is updated to include an email delivery function only if a certain email is input into the form. To get this functionality to work a delivery process is created to make the initial 'logger' delivery and also check for the email and complete that delivery process if it is found. The snippet of the code is shown below.

Dual Delivery

```
import com.avoka.fc.core.entity.DeliveryDetails
import com.avoka.fc.core.entity.Submission
import com.avoka.fc.core.dao.SubmissionDataDao
import com.avoka.fc.core.service.IDeliveryProcessService
import com.avoka.fc.core.service.ServiceLocator
import com.avoka.fc.core.service.submission.DeliveryCheckpointService
import com.avoka.fc.forms.api.DeliveryException

import java.util.List

if (deliveryCheckpoint.doCheckpoint("Checkpoint A", "Description...")) {

  try {
        // Call first delivery process
        IDeliveryProcessService deliveryService = (IDeliveryProcessService) Ser-
viceLocator.getServiceForName('Logger Delivery')
        deliveryService.deliverSubmission(submission, new DeliveryDetails())
        deliveryCheckpoint.completedCheckpoint("Checkpoint A")

   } catch (Throwable e) {
         deliveryCheckpoint.errorCheckpoint("Checkpoint A", e)
   }

}

if (deliveryCheckpoint.doCheckpoint("Checkpoint B", "Description...")) {

  try {

    def list = submission.getSubmissionExtractData();
    //Iterate through Submission Data
    for(def temp in list){
      //Check for Email Field with specifc Email value
      if(temp.getName() == "email" && temp.getValue() == "BackEnd@domain.com"){
              //Call Second Delivery process
              IDeliveryProcessService deliveryService = (IDeliveryProcessService) Ser-
viceLocator.getServiceForName('Email Delivery')
              deliveryService.deliverSubmission(submission, new DeliveryDetails())
        }
   }
    deliveryCheckpoint.completedCheckpoint("Checkpoint B")

   } catch (Throwable e) {
```

```
        deliveryCheckpoint.errorCheckpoint("Checkpoint B", e)

    }

}
```

Next, learn how to view Transact functions.

Restore Undeliverable Transactions

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

Sometimes things don't go as planned, as illustrated with the following scenario. Earlier this week you deployed the form change that was requested, it all tested ok and you were quite happy that it all went smoothly. Until two days later, when you find out that your form change caused a downstream system to fail and all submissions for these two days can't be processed anymore. Quickly restoring the previous form version will at least stop the bleeding, but what to do with the submissions that are "stuck"? You know how to fix them up, a change to the delivery process would be simple. Unfortunately, this change would also impact the current submissions, and a change would require regression testing and all kinds of approvals.

To restore transactions that can't be delivered:

1. The easiest solution at this point is to create a new delivery channel within the same organisation, let's call this one "fixer upper delivery channel".
2. For this new delivery channel you can write the Groovy code that's necessary to "massage" the data for the stuck transactions so that the downstream process can handle it.
3. We assume that you have a list of tracking codes of the stuck transactions.
4. From the Groovy console you can run the following code that will change the delivery channel on these transactions and make them ready for delivery:

```
import com.avoka.fc.core.dao.SubmissionDao
import com.avoka.fc.core.dao.DeliveryDetailsDao
import com.avoka.fc.core.dao.ClientDao
import com.avoka.fc.core.service.DeliveryResult.Status
import com.avoka.fc.core.service.ServiceFactory
import com.avoka.fc.core.entity.Submission

// create your collection of tracking codes
String[] trackingCodes = ["TC-1", "TC-2", ...]

// Find the client and then create a new DeliveryDetails object for the delivery
channel for that client
client = new ClientDao().getClientByCode("Your Organisation code");
deliveryDetails = new DeliveryDetailsDao().getDeliveryDetailForClientAndName(client,
"fixer upper delivery channel")

// we need to get the submissions using the SubmissionDa
sDao = new SubmissionDao()

for (String trackCode in trackingCodes) {
    submission =  sDao.getSubmissionByTrackingNumber(trackCode)
    if (submission) {
        println submission.formName + " found";
        submission.setDeliveryDetails(deliveryDetails)
        submission.setDeliveryStatus(Submission.STATUS_Ready)

            println submission.trackingNumber  + " changed to new delivery process"
    }
}
ServiceFactory.getSubmissionService().commitChanges()
```

Next, learn how to view Transact functions.

Reload Form Submission Data Extracts from the Groovy Console

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

If a client requests new submission data extracts after a number of initial submissions have already taken place, you can add these new data extracts to the form data configuration and then reload the data extracts for every past submission of the form using the script below in the groovy script console. However, this only works on existing XPath locations, which are existing fields in the form.

```
import com.avoka.fc.core.dao.SchemaSeedDao
import com.avoka.fc.core.dao.FormDao
import com.avoka.fc.core.service.SubmissionService
import com.avoka.fc.core.service.ServiceFactory
import java.util.List;
import com.avoka.fc.core.dao.SubmissionDataDao
import com.avoka.core.util.XmlUtils
import org.apache.cayenne.BaseContext;
/*
 Reloads data extracts for said form
*/
//Set the form code to the form you want to update the extracts.
def formCode = "PUT-FORM-CODE-HERE"

def submissionDataDao = new SubmissionDataDao();

def formDao = new FormDao();

def schemaDao = new SchemaSeedDao()

        def submissionService = new SubmissionService();

def form = formDao.getFormByFormCode(formCode)

def curVersion = schemaDao.getSchemaForVersion(form.getCurrentVersion())

def submissions = form.getSubmissions()

/*
 Loop submissions and update extracts
*/

if(submissions) {
          submissions.each(){
            def submissionXml = submissionDataDao.getSubmissionXmlString(it)
            // only if we have a submission XML string value
            if(submissionXml?.trim()) {
              def document = XmlUtils.parseDocumentFromString(submissionXml)
              submissionService.populateSubmissionExtractData(document, it, curVersion)
            } else {
              println "Skipping blank submission XML for submission track-
ingCode:${it.getTrackingCode()}"

    }
```

```
            }

}

BaseContext.getThreadObjectContext().commitChanges();
```

Next, learn more about the server-side persistence pattern.

Server-Side Persistence Pattern

Journey Manager (JM) | Form Builder Platform Developer | 5.1 & Higher

The main persistence model for Manager is the XML file. This is stored on the server, sent to the browser when a form is rendered, and sent back to the server when the form is submitted. It is also saved to the server in background save events.

However, there are cases when it may be useful to store information on the server which is not stored in the XML file. Some of these cases include:

- Storing results from a backend server call, such as an actual credit score. We want to record the actual score, but all we want to send to the form is a pass/fail. We don't want it to be in the XML because we don't want the end-customer to see their actual score, or the reasons why they failed.
- We want to temporarily store in-progress data that has been used in back-end service calls. This is to ensure that the end-user doesn't "game" the system, by changing the data in the form at different points in the journey.
- We want to store the results of a back-end service call for auditing or debugging purposes - but it's too much data to inject into the XML.
- There are probably many other cases.
- This data could be as simple as a string, or more complex like some JSON or XML.

There is a simple way to store and retrieve arbitrary string data against a Transaction using Fluent service calls.

Storing data

TxnUpdater has a method setProperty(String name, String value) which will allow you to create (or update) a property value.

Code will look something like this:

```
new TxnUpdater(txn)
    .setProperty("MyProp", "MyValue")
    .update()
```

Retrieving data

You can use TxnQuery to find the relevant Transaction, and then look at the Property Map within the Txn value object to retrieve the particular property.

Code will look something like:

```
Txn txn = new TxnQuery()
    .setSubmitKey(…) // or similar to find the correct transaction
    .withPropertyMap() // to retrieve the properties
    .firstValue() //to find the first matching transaction


// The txn contains an attribute propertyMap of type Map<String,String>
```

Some notes

- Data stored in the Transaction Properties are encrypted and subject to data retention policies, just like XML. (In recent versions of Manager.)
- In some cases, it may be necessary to store data against a collaboration job rather than an individual transaction.
- The PropertyQuery service is used for locating configuration properties for Form Versions, Workspaces, etc. This is for static configuration properties, not for per-transaction or runtime properties.
- It is also possible to use the standard HTTP session to store temporary data. However, this data will not be persisted beyond an individual session. A transaction property is persisted in the database.

Next, learn how to view Transact functions.

Tasks Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager allows you to create tasks based on forms and assign them to users or groups. The task can have prefill data associated with it, and completes when one of the assignees completes the task form.

In Manager's terms, a **task** is a combination of:

- A form or form-code
- Some XML data, which is used to pre-populate the form
- An ID of a user or group to whom the task is allocated

Internal tasks are also created when a user has not completed a transaction, for example, when they have saved but not submitted a form, submitted but not paid or added attachments, and so on.

A task consists of a form that needs to be filled in by a user. It may contain some prefill data for form customization, such as reference numbers, a task subject, and a task message that is used to identify the task to the user or other users, for example, a reviewer. When a task is created, Manager generates a submission object that is marked as a task and contains all task's related data. As the user opens and completes the task, the submission is updated as per its normal life cycle.

An assigned task type can be one of the following:

- Form - a basic task that can be created using either form XML or prefill XML data.
- Review - a task associated with a previous submission, which is often prefilled with the submitted data for review purposes. It doesn't accept prefill XML data.

Tasks can be assigned to either a single user or a group of users. The task status indicates whether the task is assigned, completed or has expired. Task expiration is controlled via an optional expiry date. When a task expires, it can invoke a **Task Expiry Process** service to notify an external system about the task expiry.

You can abandon a task if it's not required any longer or can't be performed.

Next, learn how to view all tasks.

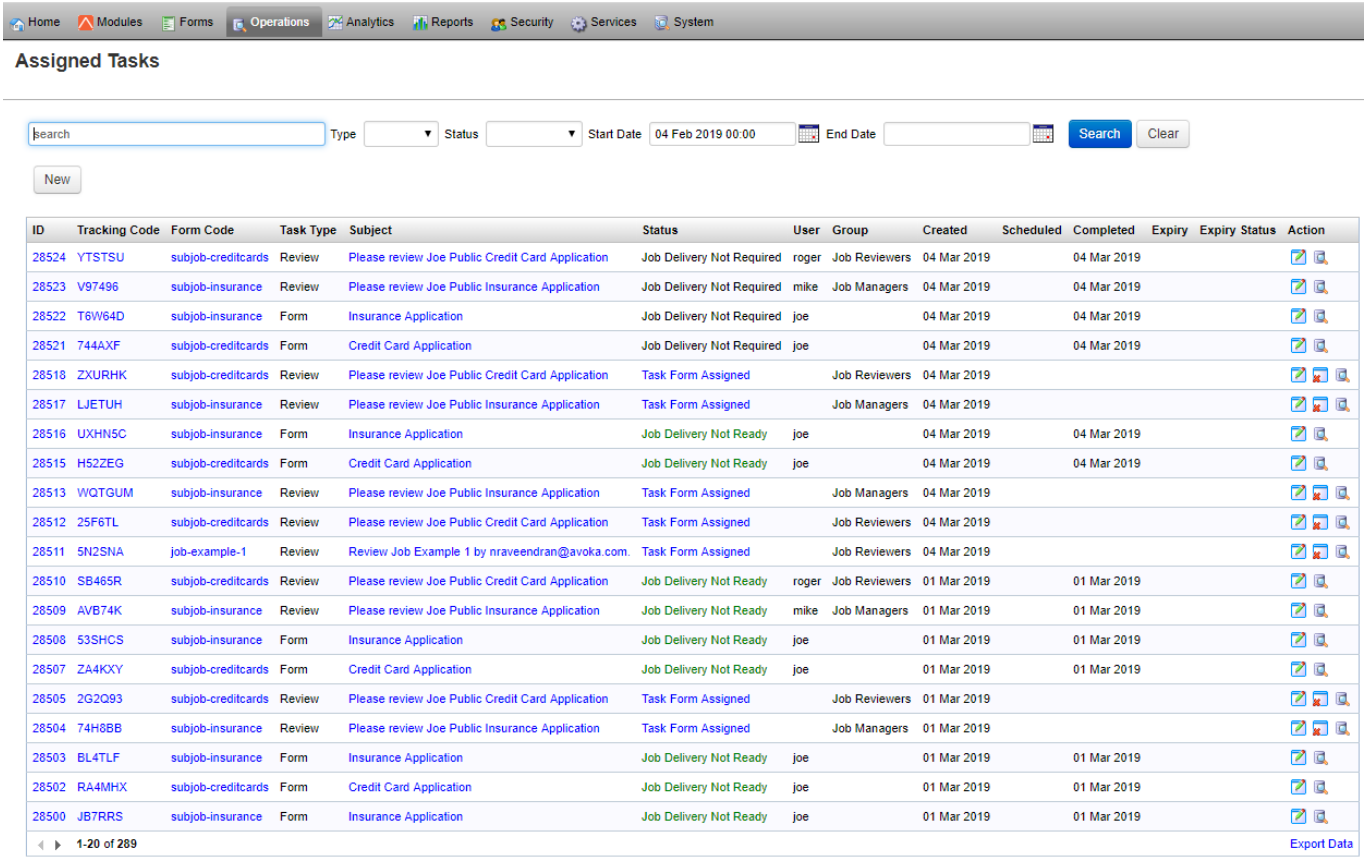
View Tasks

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view and create tasks, so you can easily manage them and search for transactions associated with each task.

To view tasks, select **Operations > Assigned Tasks.**

Tasks are shown page by page, with the maximum number of records per page configured via the user's preferences.



To filter or search the tasks by one or more criteria, specify the following settings and click **Search**:

- A transaction ID, a tracking code, a task subject name or part of its name.
- Task type

  An assigned task type can be one of the following:

- Form - a basic task that can be created using either form XML or prefill XML data.
- Review - a task associated with a previous submission, which is often prefilled with the submitted data for review purposes. It doesn't accept prefill XML data.
- Status of a task.

  An assigned task status can be one of the following:

- Abandoned
- Assigned
- Completed
- Expired
- Opened
- Saved
- Submitted
- A time interval from Start Date to End Date when a task was created.

Click **Clear** to reset the search criteria.

You can check the following task details:

- A transaction ID of the submission generated by a task
- A transaction tracking code
- A form code of a form this task was assigned for
- A task type
- A task subject that describes it to a user who will execute it
- A transaction status
- A user name whom this task is assigned to. Leave it blank to assign it to a group
- A group this task is assigned to
- Task's creation date
- Task's completion date
- Task's scheduled date
- Task's expiry date
- Task's expiry status

You can also perform the following actions:

- Create a task
- View and configure a task
- Abandon a task
- View transaction details of a task

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

> You can also find the task's transaction information using form transactions or transaction support search functionality.

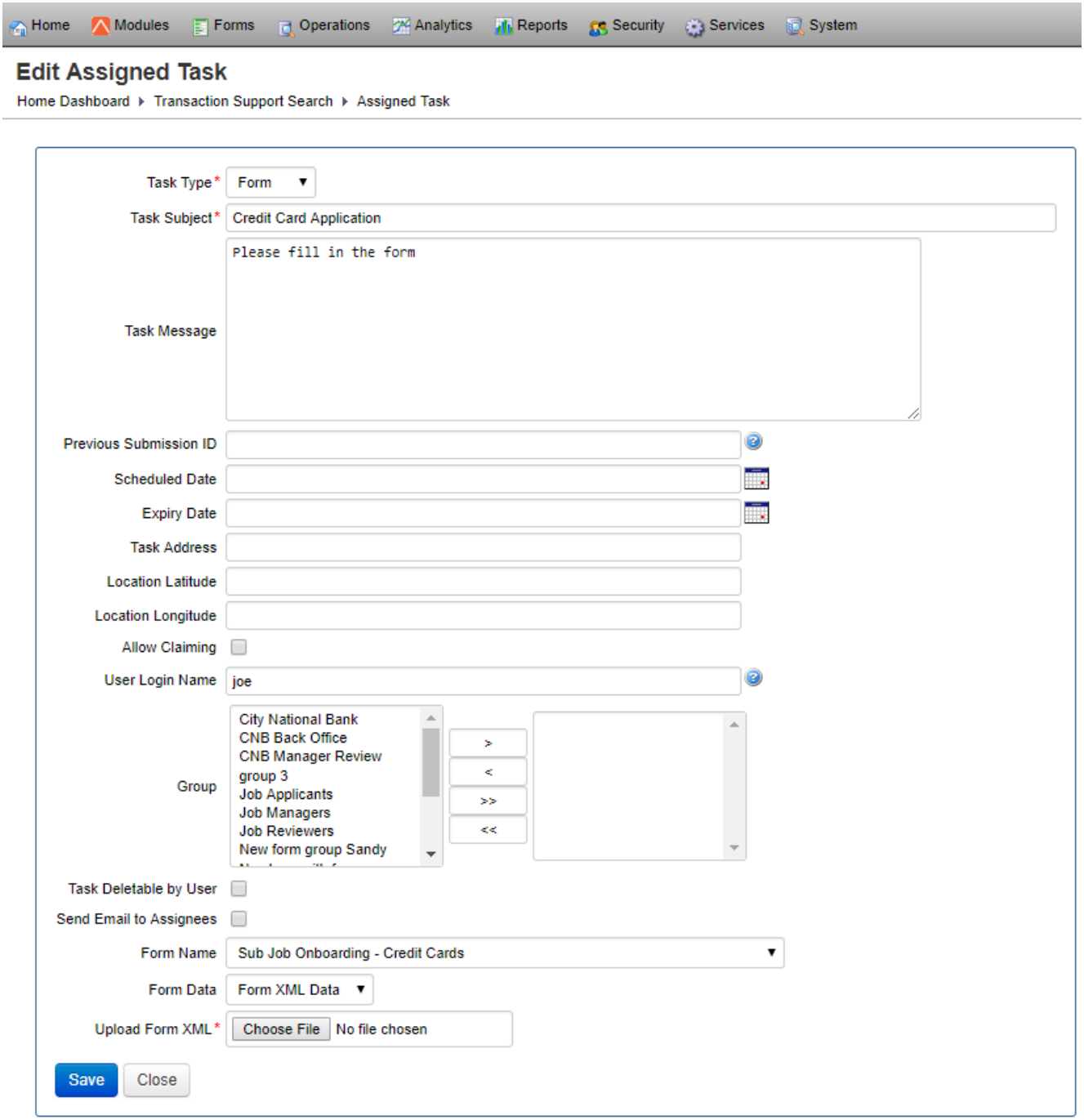
Create a Task

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create new or configure existing tasks. You can assign a task to an authenticated user, as shown below, or anonymous user.

To create a new task:

1. Select **Operations > Assigned Tasks**.
2. Click **New**.



3. Select a task type from the **Task Type** dropdown list. A task type is very useful to determine the type of the transaction and to distinguish between the submissions and review tasks.

   An assigned task type can be one of the following:

   - Form - a basic task that can be created using either form XML or prefill XML data.
   - Review - a task associated with a previous submission, which is often prefilled with the submitted data for review purposes. It doesn't accept prefill XML data.

4. Provide a brief description of a task in the **Task Subject** field.
5. Enter a task message that describes the task to a user who will perform it, for example, reference numbers. This information is shown to assignees of the workspace. You can use variables that will be dynamically substituted with configurable strings in runtime, for example, **`Please review the Job Example 1 by Joe Public. $func.startSubmission().id()`**
6. Enter the submission ID that the task is based on in the **Previous Submission ID** field. This for review tasks only.
7. If the assignee needs to be informed that the task should be done on a specific date, complete the **Scheduled Date** field.
8. Enter an expiry date in the **Expiry Date** field if you want the task to expire at a specific date and time. If a task is not completed before the expiry date, the transaction processing job will run in the background and change the task submission's form status to Expired. At this point, the default workspace will not allow assignees to complete the task. Expired task submissions can't be progressed any further.
9. Provide the physical address of a location in the **Task Address** field.
10. Enter latitude and longitude information in the **Location Latitude** and **Location Longitude** fields, if available. The latitude value needs to be in a range of -90 to 90 and the longitude value between -180 and 180. This information will be displayed to the assignee. The default spaces include Google Map support.
11. Select the **Allow Claiming** checkbox to put the task into a special claiming mode, where users have to claim the task before being able to work on it. In addition to being claimed, such tasks can be reassigned to other users by the claiming user or other space users with special privileges.
12. Provide a user's login name in the **User Login Name** field to assign the task to a specific user. Leave it blank if the task is assigned to a group, see below.
13. Select a user group name from the **Group** dropdown list so any member of the group is able to view and complete the task.

> Users joining a group will be able to view and work on all tasks existing for the group.

14. Select the **Task Deletable by User** checkbox to allow assignees to delete the task.

> Assignees will not be able to delete completed tasks to preserve the audit trail.

15. Select the **Send Email to Assignees** checkbox to send an email to the assignees to notify them of the new task. In this case, you can also fill in the **Email Subject** and **Email Message** fields. Otherwise, the default values are used.

> An email is sent to the current assignees only once. If new users join the group later, they will not receive a notification email.

16. Select a form, which needs to be filled to complete the task, from the **Form Name** dropdown list. The form can have payment and attachments associated with it as usual. For review tasks, you can leave the dropdown blank, so the form on which the original submission was based is used.
17. To prefill the form that will be presented to the assignees, you have the following options:
    - For Form tasks, if XML prefill is set up for the form, select **Input XML Prefill** from the **Form Data** dropdown list to use standard XML prefill (see section 5.16.6) and upload the prefill XML file using the **Upload Input XML Prefill** field.
    - For Form tasks, if you have a populated seed file structurally matching the form seed XML data, select **Form XML** from the **Form Data** dropdown list and upload the seed file using the **Upload Form XML** field.
    - For Review tasks, the XML data of the previous submission is automatically used, if nothing else is specified.
18. Click **Save** to update the changes. A task submission will be created with the form status as **Assigned**, and users will be able to fill in the form and complete the task using the workspace associated with the form.

Next, learn how to view all tasks.

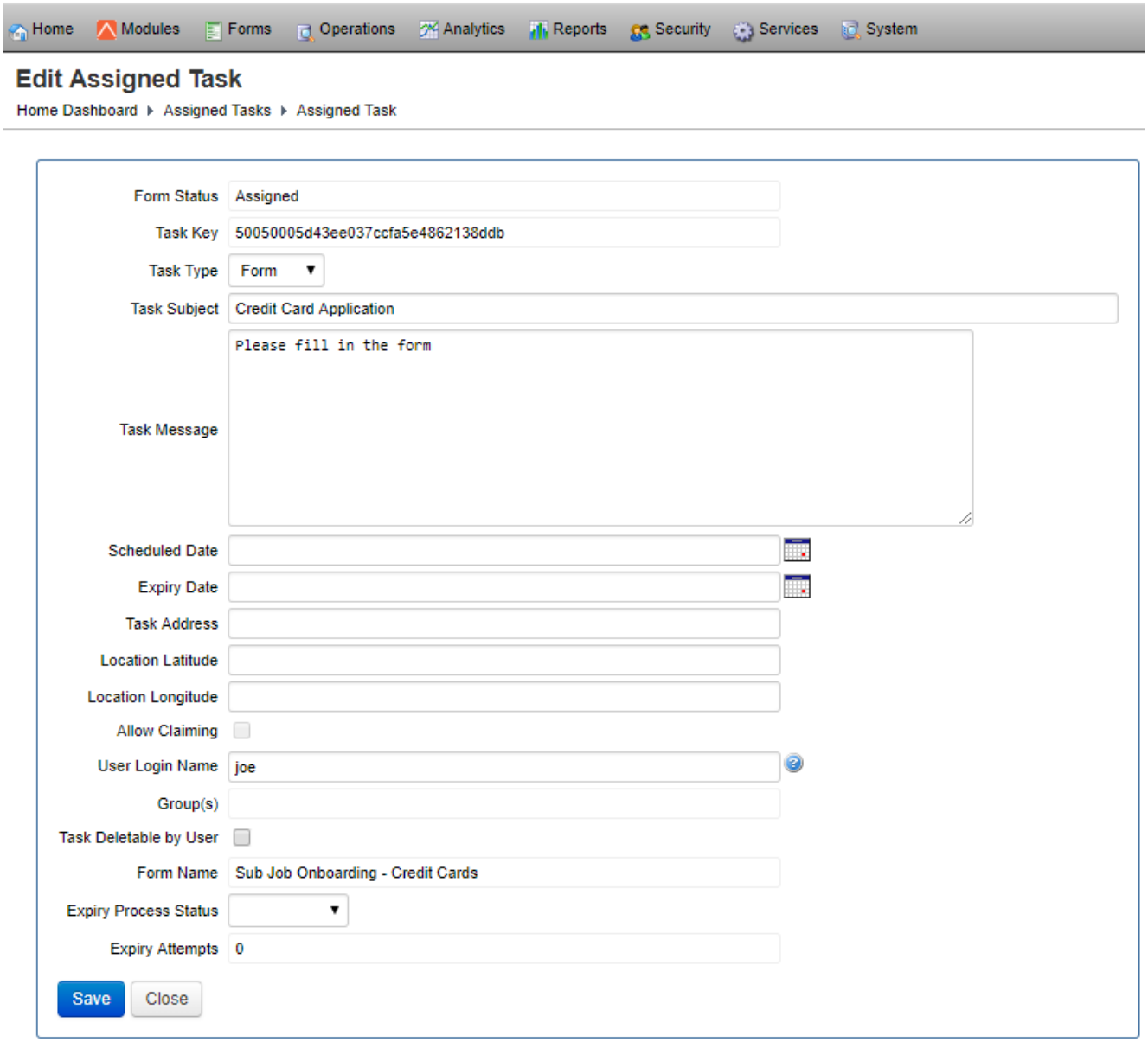
Configure a Task

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create new and configure existing tasks.

To configure a task:

1. Select **Operations > Assigned Tasks**.
2. Locate a task and click **Edit**.
3. Click **Edit** to view the existing configuration. For information on task configuration, see create a task.



4. Check the unique **Task Key** generated by Manager.
5. Select a status from the **Expiry Process Status** dropdown list that can be one of the following:

- Required
- Error
- Completed

> This should not be necessary during normal operations because this status is updated by the expiry process service.

6. **Expiry Attempts** show the number of attempts taken at this task.
7. Click **Save** to update the changes.

Next, learn how to view the task's transaction details.

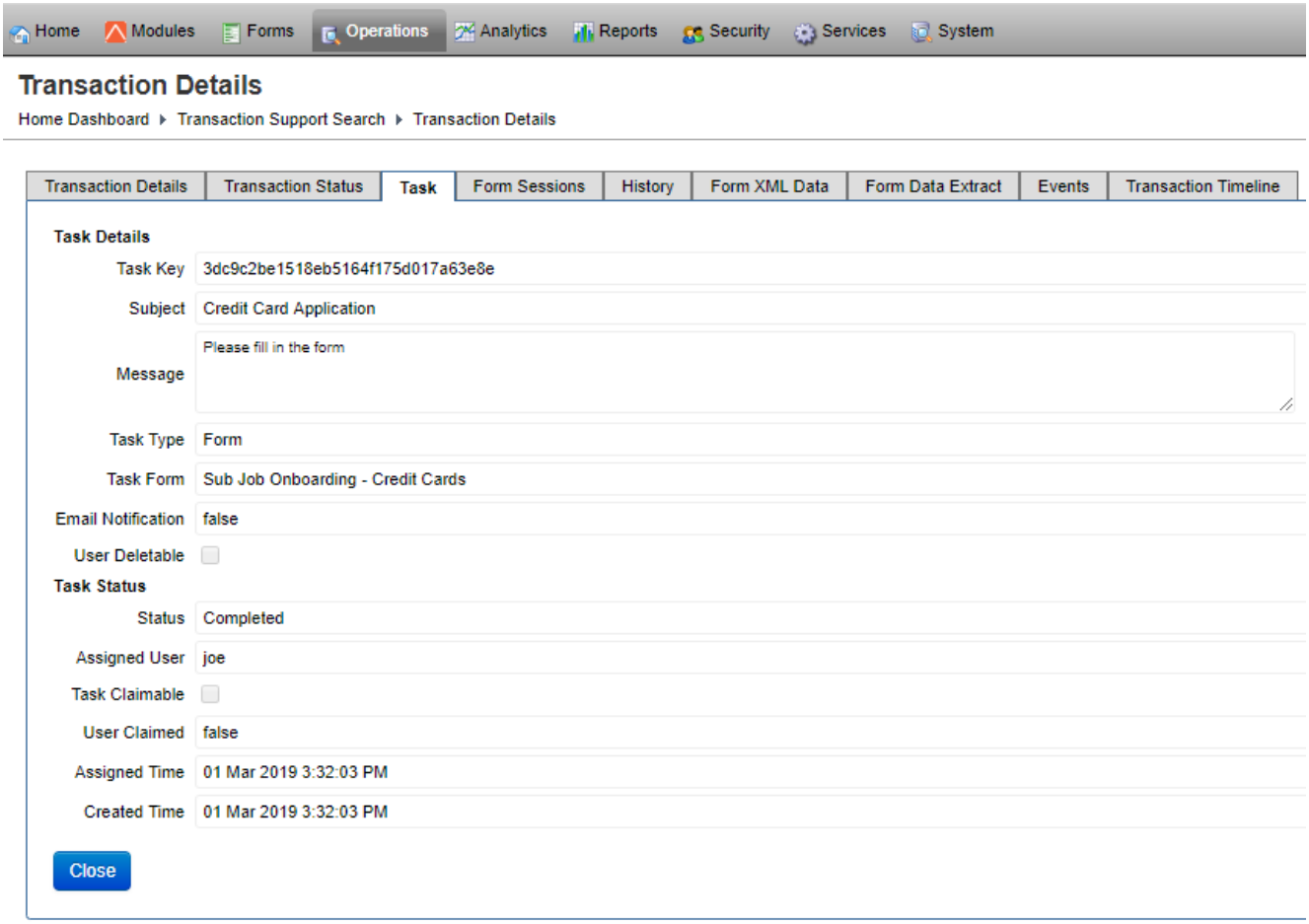
View Task Transaction Details

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view transaction details of tasks. You can use this functionality to troubleshoot tasks and form submissions.

To view the task's transaction details:

1. Select **Operations > Assigned Tasks**.
2. Locate a task.
3. Click **View Transaction Details** and select the **Task** tab.



4. Click **Close**.

> You can also find the task's transaction details using the transaction support search functionality.

Next, learn how to abandon a task.

Abandon a Task

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to abandon a task when it's no longer required. Abandoned tasks will be purged.

To abandon a task:

1. Select **Operations > Assigned Tasks**.
2. Locate a task.
3. Click **Abandon Task** and confirm your decision.

Next, learn how to create a task.

System Event Publisher Overview

Journey Manager (JM) | System Manager / DevOps | 18.05

Journey Manager allows you to send server events and other information to external systems in near real-time using System Event Publisher (SEP). You can use this functionality to monitor Manager from your external systems, such as Splunk, via a third-party message queue using standard protocols.

System Event Publisher hooks into Manager's transaction processing engine, records events in memory and sends the applicable events to a queue. It all happens in the background to maintain performance of the user flow. Manager attempts to retry delivery of event messages in case of an error, but it can't be guaranteed, meaning, event messages are published on a best effort basis.

A Manager administrator can configure a System Event Publisher to publish Manager's events using Amazon AWS SQS, and, if required, authenticate a connection using the **Amazon AWS Security Token Service**[1]. You can further customize it to publish only events that are essential to be monitored within your external systems.

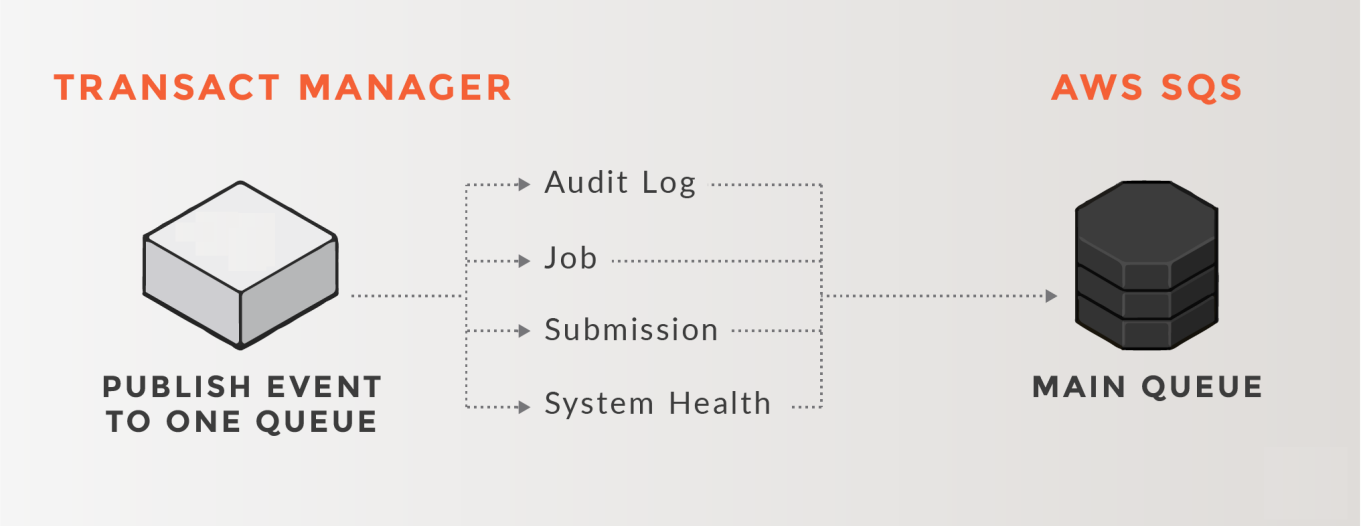
You can configure System Event Publisher to publish the following events to external systems:

- Audit log
- Collaboration Jobs: steps, actions and action properties, and collaboration job update messages
- Submissions including data extracts, properties, milestones, and Groovy logs
- Submission update messages
- Submission form data mapping
- Error logs
- Event logs
- Submission Groovy service logs
- Submission Groovy service log data
- Submission milestones
- Submission properties
- Transaction statuses
- System health
- User authentication events. | 18.11

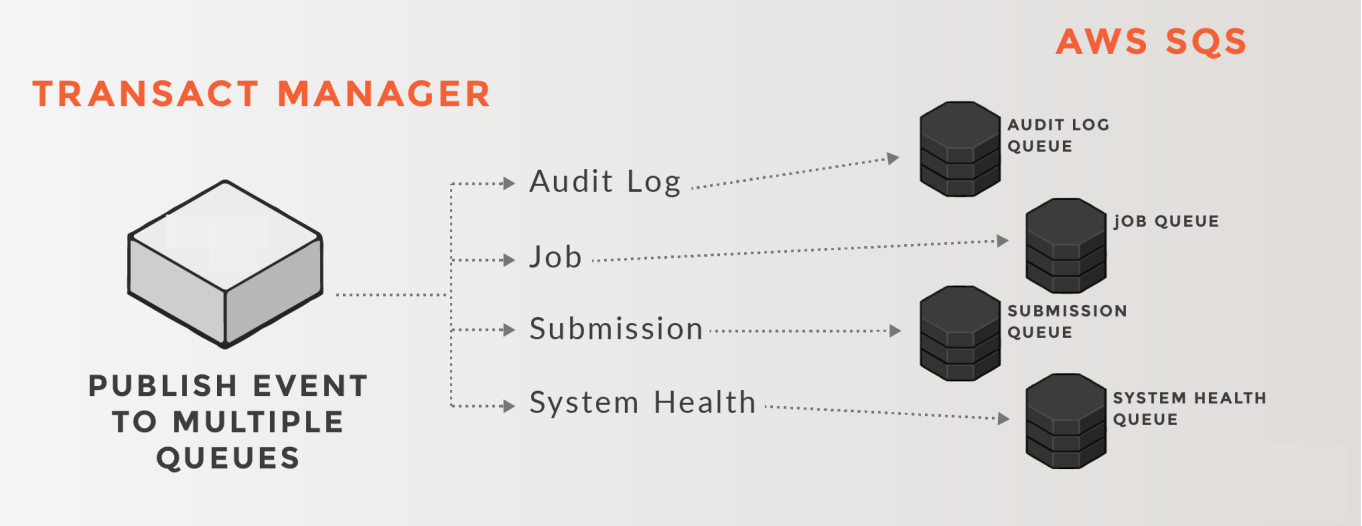
System Event Publisher can publish events to the one default queue and up to three separate queues. For example, job, submission and system health events can be sent to three separate queues respectively. To achieve this, System Event Publisher should be configured differently. There are two distinct configurations:

[1]The AWS Security Token Service (STS) is a web service that enables you to request temporary, limited-privilege credentials for AWS Identity and Access Management (IAM) users or for users that you authenticate (federated users).

A Single Queue



Multiple Queues



Next, learn how to configure system event publisher.

Configure System Event Publisher

Journey Manager (JM) | System Manager / DevOps | 20.05

Manager version 18.05 and higher comes with the System Event Publisher, which you can use to publish events into various queues. Initially, Manager supported only AWS queues, but starting from the release 20.05, it also supports Azure queues.

> A Manager administrator can configure the System Event Publisher.

To start using the System Event Publisher:

1. Configure a service connection to connect to a queue. Use an AWS SQS service connection or AWS STS service connection for AWS cloud, or an Azure service connection for Azure cloud.
2. Configure a system event publisher service: an AWS SQS System Event Publisher service for AWS cloud or Azure System Event Publisher Service for Azure cloud.
3. Configure the external to Manager processes and systems to transfer, store and report on the raw data according to your requirements.

Once the configuration is complete:

- Each time a Manager event takes place, the System Event Publisher formats the event relevant details as a JSON[1] message and places it on the internal queue.
- If opted for the system health message to be published, the System Event Publisher formats it as a JSON message with the system health message details and places it on the internal queue each hour.
- The internal queue is published immediately to the queueing system, such as AWS SQS or Azure Queue.

Next, learn about different types of System Event Publisher events.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

System Event Publisher Events

Journey Manager (JM) | System Manager / DevOps | 18.05

The System Event Publisher publishes the following Manager events:

- Audit Log event
- Job event
- Submission event | 18.11
- System Health event
- User authentication event | 18.11

The System Event Publisher uses the following generic JSON[1] message format for publishing events:

```
{
        "environmentName": "Engineering Test 1",
        "systemId": "c41d354134397c0362846d5efd7f34f4",
        "serverNode": "ip-192-168-120-149.us-west-2.compute.internal - TM Server",
        "threadName": "TM_Worker-1",
        "auditLog": {
                ...
        }
}
```

```
{
        "environmentName": "Engineering Test 1",
        "systemId": "c41d354134397c0362846d5efd7f34f4",
        "serverNode": "ip-192-168-120-149.us-west-2.compute.internal - TM Server",
        "threadName": "TM_Worker-1",
        "job": {
                ...
        }
}
```

The event message contains only one child object component, such as **auditLog**, **job**, **submission** or **systemHealth**.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

Next, learn how to configure System Event Publisher.

System Event Publisher Audit Log Events

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager allows you to configure a [%TransactVariables.SystemEventPublisher%] to publish [%TransactVariables.AuditLog%] events.

> Select the **Audit Log Publish** checkbox in the AWS SQS System Event Publisher service configuration to publish this event.

System Event Publisher sends out Audit Log events in the following JSON[1] format.

```
{
	"environmentName": "server1.com",
	"systemId": "c41d354134397c0362846d5efd7f34f4",
	"serverNode": "ip-192-168-120-149.us-west-2.compute.internal - TM Server",
	"threadName": "TM_Worker-4",
	"auditLog": {
		"id": 517,
		"eventTime": "2018-05-13T00:08:47+1000",
		"user": "administrator",
		"eventType": "View",
		"entityName": "Submission",
		"entityId": 93,
		"message": "Submission XML viewed by administrator"
	}
}
```

Next, learn about other events.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

System Event Publisher Job Events

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager allows you to configure a [%TransactVariables.SystemEventPublisher%] to publish job events.

> Select the **Job Publish** checkbox in the AWS SQS System Event Publisher service configuration to be able to publish this event.

System Event Publisher sends out job events in the following **JSON**[1] format.

```
{
        "environmentName": "server1.com",
        "systemId": "c41d354134397c0362846d5efd7f34f4",
        "serverNode": "ip-192-168-120-149.us-west-2.compute.internal - TM Server",
        "threadName": "TM_Worker-4",
        "job": {
                "id": 3,
                "eventTime": "2018-05-10T21:37+1000",
                "jobKey": "19677169dac21105ce99be73d13bf3eb",
                "jobName": "Customer Onboarding Job",
                "jobRefNumber": "5VFL8Q7",
                "clientCode": "maguire",
                "status": "Completed",
                "stepName": "Application Completed",
                "stepStatus": "Completed"
        }
}
```

The **status** is a job status.

A job status can be one of the following:

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

- In Progress - the collaboration job is active and in progress.
- Completed - the collaboration job has been completed. This happens when a endpoint step has completed.
- Expired - this happens when an endpoint expiry collaboration job has been automatically expired by Journey Manager.
- Cancelled - the collaboration job has been manually canceled by an administrator.
- Error - the collaboration job has a critical or unrecoverable error that requires an administrator intervention before it can be run again.
- Paused - the collaboration job has been paused temporarily.

The **`stepStatus`** is a job step status.

A job step status can be one of the following:

- In Progress- the step is in progress now.
- Completed - the step is completed.
- Expired - the step is expired.
- Cancelled - the step has been manually cancelled by an administrator.

Next, learn about other events.

System Event Publisher Submission Events

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager allows you to configure a System Event Publisher to publish submission events.

> Select the submission checkboxes in the AWS SQS System Event Publisher service configuration to publish these events.

System Event Publisher sends out submission events in the following JSON[1] format.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

```
{
	"environmentName": "server1.com",
	"systemId": "c41d354134397c0362846d5efd7f34f4",
	"serverNode": "ip-192-168-120-149.us-west-2.compute.internal - TM Server",
	"environmentName": "Engineering Test 1",
	"threadName": "default task-87",
	"submission": {
		"id": 17253,
		"eventTime": "2018-10-05T13:15:30Z",
		"submitKey": "ade3e4ebfe7016c8c10a3b5c0ee107e7",
		"requestKey": "28673f7e0c92d9d7ebc8bd80af8318af",
		"trackingCode": "Y85C5H",
		"clientCode": "maguire",
		"formCode": "CCA-RB",
		"formStatus": "Saved",
		"deliveryStatus": "Not Ready",
		"user": "john.smith@maguire.com",
		"jobRefNumber": "RJ6GJR9",
		"formDataMap": {
			"First Name": "John",
			"Last Name": "Smith",
			"Email": "john.smith@amcore.com",
			"Mobile": "0417 298 273"
		},
		"errorLog": [
		{
			"id": 1850,
			"errorTime": "2018-05-10T20:36+1000",
			"errorType": "Email",
			"message": "SmtpEmailService: SMTP Host=null"
		}
		],
		"eventLog": [
		{
			"id": 69378,
			"eventTime": "2018-10-05T13:15:30Z",
			"eventType": "INFO",
			"message": "Email sent to: test_user1@company.com"
		}
		],
		"groovyServiceLog": [
		{
			"id": 74308,
			"invokeTime": "2018-10-05T13:15:30Z",
```

```
                        "serviceName": "IDV Service",
                        "versionNumber": "1.0.2",
                        "durationMs": 26,
                        "status": "Completed",
                        "groovyLog": [
                                "13:15:30,056 INFO  Applicant CIN: 121483-38390\r",
                                "13:15:30,094 INFO  IDV Status: Completed"
                        ]
                }
        ],
                "milestone": [
                        "IDV" : "Completed"
                ],
                "property": {
                        "CIN": "121483-38390"
                },
                "requestLog": {
                        "id": 32813,
                        "requestTimestamp": "2018-05-10T20:34+1000",
                        "requestKey": "6e29b53f7b8cc4e5bf39148be987ef69",
                        "sessionId": "oC20STVJ4tvEOPl4jP256zhucOAX8YH5AIv7gzFj",
                        "ipAddress": "114.141.100.202",
                        "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101
fox/60.0"
                }
        }
}
```


For more information on the form status and delivery status values, see data retention status.

If the **formStatus** is Abandoned, an additional **formAbandonmentType** indicating the reason for abandonment is included.

A form abandonment types can be one of the following:

- Bounced
- Started
- Cancelled
- Ineligible
- Saved
- Submitted

The following is an example of an abandoned transaction with an additional **formAbandonmentType**:

```
...
        "formStatus": "Abandoned",
        "deliveryStatus": "Not Ready",
        "formAbandonmentType": "Cancelled",
        "formDataMap": {
                ...
        }
...
```

Manager includes submission PII data purging information in a JSON message. | 18.11

This event is always published by the AWS SQS System Event Publisher service.

You can use this functionality to:

- Monitor submission PII data purging in Manager.
- Build external systems to monitor the progress of submission PII data purging.

The JSON format is the same as for submissions events with the following extra fields:

- scheduledSubmissionPurge
- scheduledPiiDataPurge
- piiDataDeletedTime

Before purging

```
{
        ...
        "deliveryStatus": "Completed",
        "scheduledPiiDataPurgeTime": "2018-10-23T15:12:22+1100",
        "scheduledSubmissionPurgeTime": "2019-01-14T15:12:22+1100",
        "user": "test_user1@company.com",
        ...
}
```

After purging

```
{
        ...
        "deliveryStatus": "Completed",
        "scheduledPiiDataPurgeTime": "2018-10-16T15:16:50+1100",
        "piiDataPurgedTime": "2018-10-16T15:16:59+1100",
        "scheduledSubmissionPurgeTime": "2019-01-14T15:12:23+1100",
        "user": "test_user1@company.com"
        ...
}
```

Next, learn about other events.

System Event Publisher System Health Events

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager allows you to configure a [%TransactVariables.SystemEventPublisher%] to publish [%TransactVariables.SystemHealth%] events.

> Select the **System Health Publish** checkbox in the AWS SQS System Event Publisher service configuration to publish this event.

System Event Publisher sends out System Health events in the following **JSON**[1] format.

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

```
{
        "environmentName": "server1.com",
        "systemId": "c41d354134397c0362846d5efd7f34f4",
        "serverNode": "ip-192-168-120-149.us-west-2.compute.internal - TM Server",
        "threadName": "TM_Worker-1",
        "systemHealth": {
                "id": 392781,
                "eventTime": "2018-10-05T13:00:00Z",
                "abandonCount": 0,
                "deliveryCount": 0,
                "deliveryErrorStateCount": 0,
                "deliveryReceiptRenderCount": 0,
                "errorCollabJobCount": 0,
                "errorCount": 0,
                "errorCriticalCount": 0,
                "errorDeliveryCount": 0,
                "errorDynamicDataCount": 0,
                "errorEmailCount": 0,
                "errorFwSyncCount": 0,
                "errorGroovyCount": 0,
                "errorLivecycleCount": 0,
                "errorManagementCount": 0,
                "error_paymentGatewayCount": 0,
                "errorReceiptRenderCount": 0,
                "errorRenderCount": 0,
                "errorRestSoapCount": 0,
                "errorScheduledJobCount": 0,
                "errorSecurityManagerCount": 0,
                "errorSubmissionCount": 0,
                "errorUnclassifiedCount": 0,
                "eventCount": 0,
                "formRenderCount": 0,
                "fwSyncCount": 0,
                "groovyServiceCount": 0,
                "jobCreatedCount": 0,
                "saveCount": 0,
                "scheduledJobCount": 0,
                "submissionCount": 0,
                "userReceiptRenderCount": 0
        }
}
```


Next, learn about other events.

System Event Publisher User Authentication Events

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager allows you to configure a System Event Publisher to publish user authentication events. You can use this functionality to monitor Manager for fraudulent and suspicious activity.

> Select the **User Auth Publish** checkbox in the AWS SQS System Event Publisher service configuration to publish this event.

System Event Publisher sends out user authentication events in the following JSON[1] format.

```
{
	"environmentName": "server1.com",
	"systemId": "f121e2b022fb852999c3be5f2ef7067f",
	"serverNode": "127.0.0.1 - TM Server",
	"threadName": "default task-47",
	"userAuth": {
		"id": 1234,
		"user": "test_user1@company.com",
		"portal": "Maguire",
		"eventTime": "2018-10-15T11:27:33+1100",
		"eventIpAddress": "127.0.0.1",
		"eventSessionId": "nILJPPBTZB-NuNU9tukO_l5UAgeH4fRpqLR2KzeJ",
		"eventUserAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/69.0.3497.100 Safari/537.36",
		"eventType": "Login Success"
	}
}
```

A user login event type can be one of the following:

- Account Locked Temp.
- Login Failure.
- Log Out.
- Login Success.
- Password Reset.
- Session Expiry.

---

[1] JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

Next, learn about other events.

Security Managers Overview

Journey Manager (JM) | System Manager / DevOps | 21.11

Journey Manager is designed with a robust security architecture in mind that ensures a safe and secure foundation for users to access forms and applications hosted on the server. The security architecture is implemented using Security Managers, which are software components within Manager that handle users request access to different form spaces. Users can be external or internal, and they can require access to form spaces or a combination of form spaces and modules in different environments.

Security Managers are responsible for:

- **User authentication**[1]
- **User authorization**[2]
- User self-registration
- **Session management**[3]

Security Manager use the latest technologies, such as TLS, SSL, SHA and AES, to allow them fulfill their responsibilities.

Additionally, Manager has the following built-in features to harden security of applications it runs:

- TLS encryption for data in transit
- Form XML data at rest is stored as Encrypted BLOB values using AES 256-Bit encryption
- Passwords are stores as SHA 512 hash values
- SMTP Relay over TLS for integration with bank's email systems
- TLS, SSL mutual authentication and IP Whitelisting for internal integrations
- Role based permissions can be integrated with authentication systems, such as ADFS, WS-Trust, OAuth2, SAML

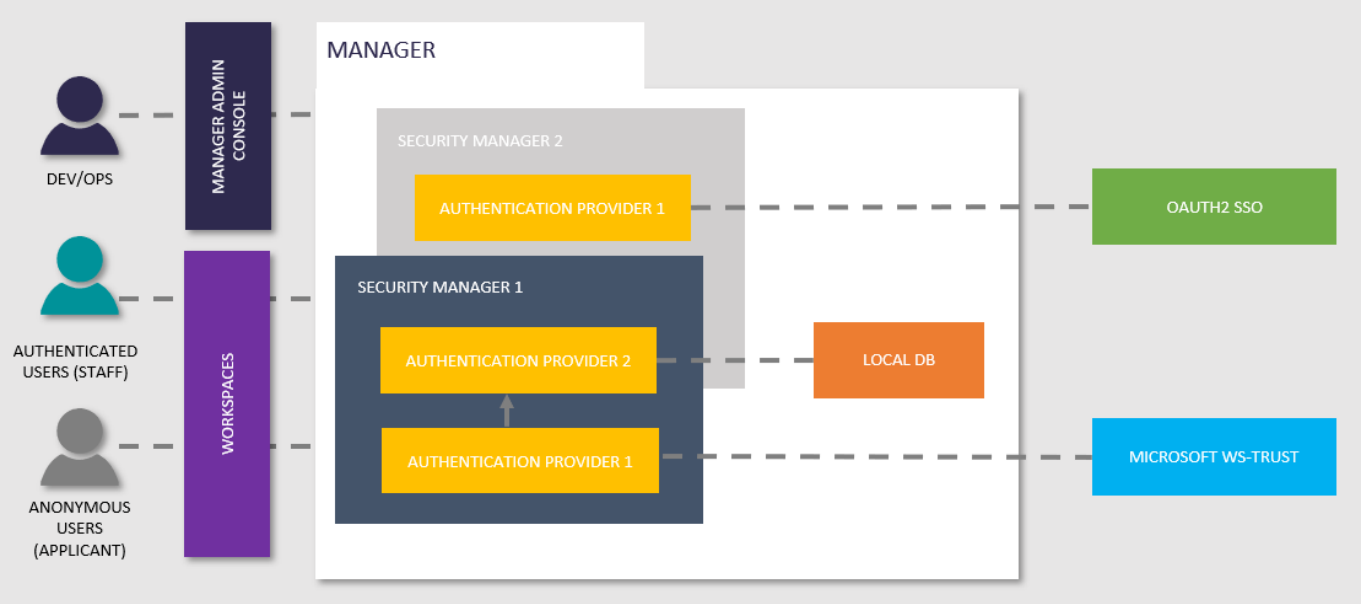
The security architecture enables developers and system administrators to implement best security practices, such as:

---

[1]User authentication is the verification of an active human-to-machine transfer of credentials required for confirmation of a user's authenticity.
[2]User authorization is interested in who the user is, and is used to determine what functions, actions, data, or other parts of an application the user has access to.
[3]Session management is the rule set that governs interactions between a web-based application and users. Browsers and websites use HTTP to communicate, and a web session is a series of HTTP requests and response transactions created by the same user. Since HTTP is a stateless protocol, where each request and response pair is independent of other web interactions, each command runs independently without knowing previous commands. In order to introduce the concept of a session, it is necessary to implement session management capabilities that link both the authentication and access control (or authorization) modules commonly available in web applications.

- Never log **PII**[1] information to the Groovy service log
- Never turn on debug level of logging in production environments
- Never store passwords in the clear formats
- Never log passwords to the log files
- Never pass internal security data to the client, for example, by adding them in the XML files

A security manager contains one or more authentication providers that can be chained. That is, if one authentication provider can't authenticate a user due to some reason, the next authentication provider is called and so on until the user is authenticated or no authentication providers are left. Providers can be local or delegated, as shown below.



An authentication provider's main job is to authenticate a user that will result in allowing or denying user access to a resource, which is often a form. Security Managers can be exported and imported across different Manager instances to make it easier to implement security requirements.

Manager comes with the following ready-to-use security managers:

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

- Local Security Manager
- LDAP Security Manager
- Microsoft ADFS Security Manager
- Microsoft WS-Trust Security Manager
- SSO Security Manager
- OAuth2 SSO Security Manager
- Fluent Microsoft ADFS Security Manager | 19.11
- Fluent SSO Security Manager | 19.11
- Fluent OAuth2 SSO Security Manager | 21.11

SSO security managers, such as Microsoft ADFS, have specific Groovy scripts, which you can configure to acquire a SSO token or call a chain of authentication providers. It allows you to create various SSO and login flows.

SSO Flow

1. SSO Get Auth Token. The result is passed into the authentication providers. If there's no result or a local authentication provider exists, it can go to the login flow
2. Authentication provider list
3. SSO authentication OK response
4. SSO re-validation

Login Flow

1. Login page
2. Authentication Providers, for example, WS-Trust and LDAP or Local DB

Manager controls access to the Manager Dashboard using permissions and organization based filters, whereas access to forms and content on the form space is controlled using groups, permissions, and user account based filtering.

Download the Security Managers PDF

We've replaced the Security Managers online documentation with a PDF version. Here's how you can get a copy of the PDF.

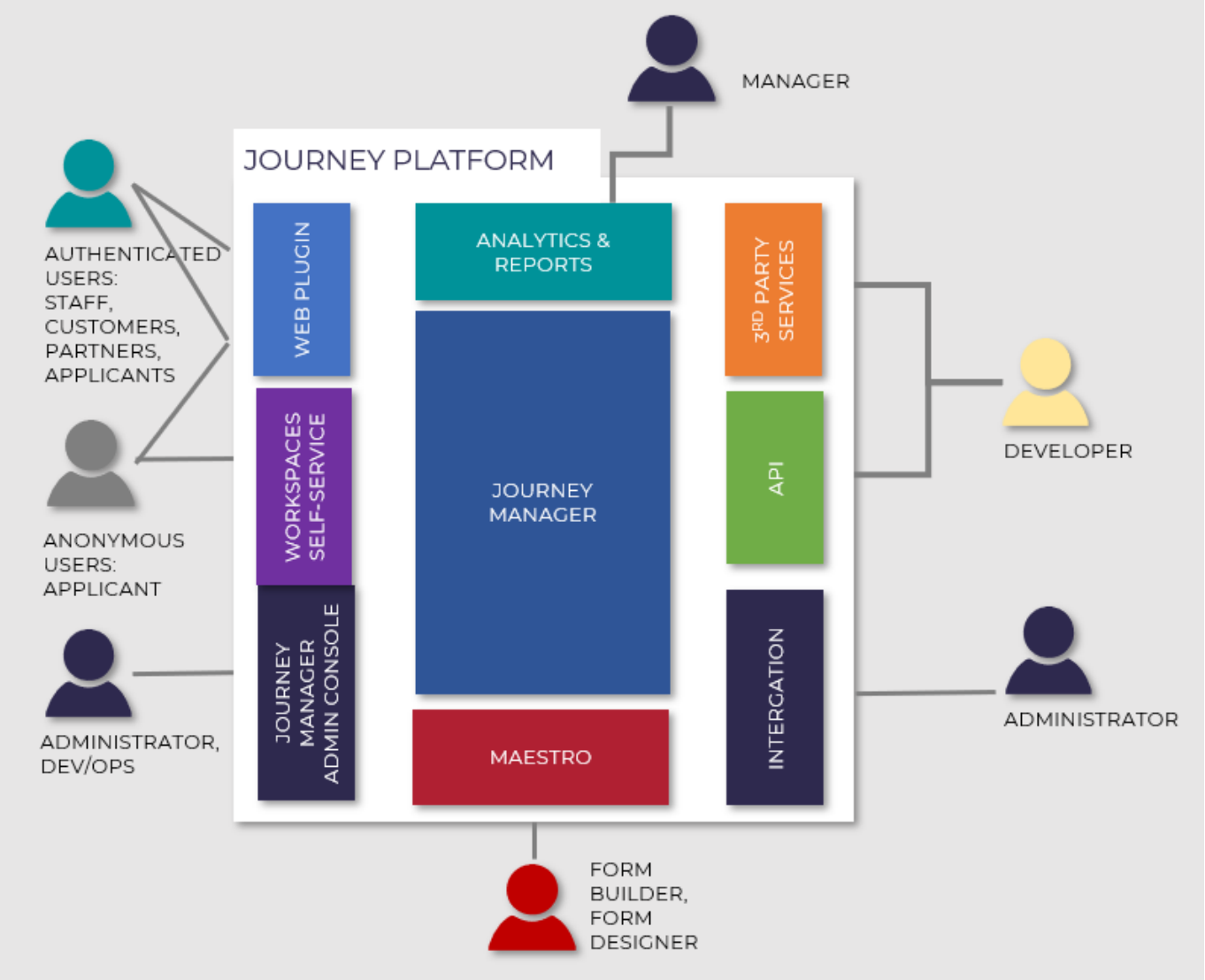
- If you have access to the Temenos instance of Microsoft Teams, you can download the Security Managers PDF from Teams. Follow these steps to download the version you need:
  1. Login to Microsoft Teams.
  2. Select the **Temenos Journey Manager (TJM)** team.
  3. Select the **Temenos Document Share** channel.
  4. Select **Files** and browse to the **SecurityManager (INTERNAL USE ONLY)** folder.
  5. Download the Security Managers PDF for your version of Manager (**Security-Managers.pdf**).
- Otherwise, contact your customer support manager (CSM) to request a copy of the Security Managers PDF.

Next, learn about organizations.

User Accounts Overview

Journey Manager (JM) | System Manager / DevOps | 19.05

A user account is part of the Journey Manager security management system, which allows for fine-grained control based on user's functions, roles, groups, and access permissions. There are two types of the users - authenticated and anonymous (unauthenticated or public) - both user types can access forms. The diagram below illustrates various user types and what they can access in the Journey Platform.



Manager provides the easy-to-use mechanism to create, modify, and deactivate user accounts, as well as manage access permissions, user roles and groups. It also comes with the bulk import functionality to create multiple users, including user's details, roles, groups and access configurations, which significantly simplifies user provisioning task.

Authenticated Users

Before users can log in to a form space, which requires authentication, they must have valid user accounts. Manager allows an administrator to create a user and grants access to one or more form spaces. The user receives an email with login credentials, such as a username and password, to access forms hosted on the form space.

Manager allows authenticated users to login to a form pace using credentials by delegating authentication and authorization to the 3rd party system, such as Active Directory, **LDAP**[1] or **SSO**[2]. Manager can also maintain local users, such as contractors that are not in the organization's LDAP system.

The authenticated users can be:

- Business users internal to a company,
- Government departments,
- Staff and contractors that work in the field.

Authenticated users have access to the advance features of the secured portal such as Form, Task and History pages.

Anonymous Users

Anonymous (unauthenticated or public) users can access forms if its form space is configured to allow anonymous access. An example of such a form space is the Web Plug-in form space. Members of the public click on a link from a company website which opens an unsecured form page in the portal. The available list of forms is manually maintained by the company on their website.

Using anonymous users has the following advantages:

- Removes the overhead of maintaining users and passwords.
- Minimizes privacy issues related with storing public users' data, such as **PII**[3].

Anonymous users don't have access to the advance features of the secured portal such as Form, Task and History pages.

---

[1]Lightweight Directory Access Protocol (LDAP) s an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol network.
[2]Single sign-on (SSO) is a property of access control of multiple related, yet independent, software systems. With this property, a user logs in with a single ID and password to gain access to any of several related systems.
[3]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

An anonymous user can still use the forms, but with the following limitations:

- If they save a form, an email will be sent to them with a link that will open the saved form.
- Tasks are emailed to a user with a link to open the task.
- Job statuses can be emailed out the user as the job progresses.

Anonymous users must provide their email address. Optionally, a security question can be set to open saved forms and tasks.

You can grant standard roles to users to control their access to and define capabilities at their form spaces. Each form space has a set of roles that can be granted to a user. You can create a new role as needed.

A typical form user must only be granted access to a form space where forms are hosted. This allows self-registered users immediate access to forms. However, most users must be granted access to:

- Form spaces
- Modules
- Organizations
- Roles
- Groups

When working with user accounts, you should consider the environment that users are accessing, which can be production, development, or testing environment. Manager is designed to be customizable to accommodate enterprise's security configurations and policies. The following users require access to a production environment:

- Form users to access, complete and submit forms.
- Operational staff to monitor and manage transactions and collaboration jobs.
- Administrators to manage the environment and keep it secure.
- System managers to view and analyze transaction reports.
- Reviewer or manager to approve or reject customer form applications.

The following users require access to a development and test environments:

- Administrators to manage the environment and keep it secure.
- Form builders to create and configure forms.
- Testers to test forms.

Users can only log into a form space if their user accounts are assigned to that form space. It is not possible to assign a form space to a user account when the form space is only accessed anonymously.

The following users should have access to form spaces:

1. Form users to log into a form space.
2. Form Builders to log into one or more form spaces:

- Maestro
- Manager
- Journey Analytics
- Business Reports
- Salesforce
- Workspaces

3. Administrators to log into Manager.
4. Testers to log into various form spaces.
5. Operators to log into Manager.
6. Managers to log into one or more form spaces:
   - Journey Analytics
   - Business Reports
   - Salesforce
   - Workspaces

A standard user account contains the following information:

- Credentials - username and password.
- User type - security managers used in your form space, which can be Local, LDAP or SSO.
- User profiles - user contact details. A user can have more than one user profile containing different contact details.
- Form spaces - assigned to the user to authorize access to specific form spaces.
- Organizations - assigned to the user to authorize access to specific organizations. Applies to modules, but not relevant for form spaces.
- Roles - grants individual roles which specify fine-grained permissions to access Journey platform and form spaces.
- Groups - belongs to an individual group for authorizing access to restricted forms, receive alerts and notifications pertaining to a group.

Manager enforces some extra security rules to ensure that users cannot access restricted content, such as organizations, groups, users, forms, transactions and collaboration jobs. | 19.05

Next, learn how to view user accounts.

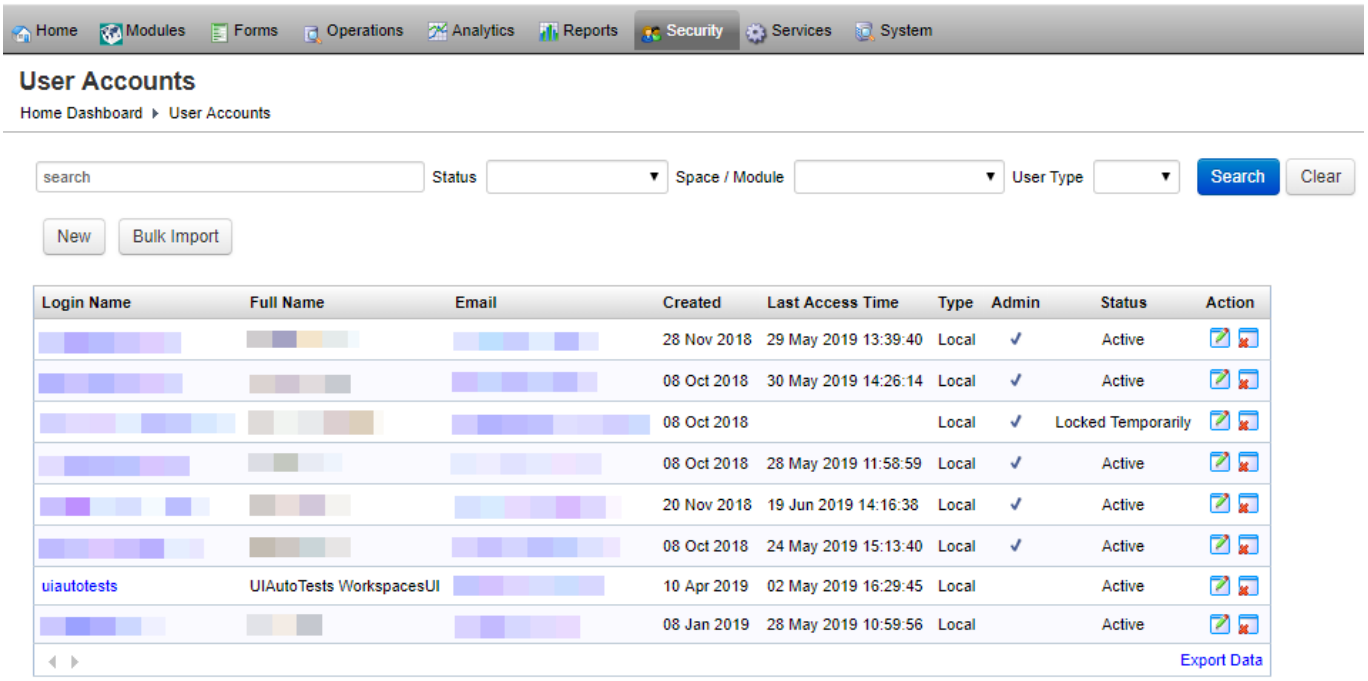
View User Accounts

Journey Manager (JM) | System Manager / DevOps | 20.05

Manager allows you to view the list of available user accounts.

To view a list of user accounts, select **Security > User Accounts**.

User accounts are shown page by page, with the maximum number of records per page configured via the user's preferences.



To filter or search the user accounts by one or more criteria, specify the following settings and click **Search**:

- User account's name or part of a name.
- Status

  A user account status can be one of the following:

- Active
- Inactive
- Locked
- Locked Temporarily
- Pending
- Rejected
- Space
- User Type

  A user account types can be one of the following:

- Local
- LDAP
- SSO
- Organization

Click **Clear** to reset the search criteria.

> A check mark against a user in the **Admin** column means the user has access to the **Transact Manager** form space.

You can also perform the following actions:

- Create a user account.
- Configure a user account.
- Remove a user account.
- Bulk Import user accounts.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to bulk import user accounts.

Create a User Account

Journey Manager (JM) | System Manager / DevOps | 20.05

Manager allows you to create user accounts, so you can set up a new or migrate an existing user from another environment. Creating a user is the first step in setting up a user account, so you also need to assign roles, groups, organizations, and form spaces to the user account before the user can perform actions in Manager. Then, the user can create and configure a user profile to adjust it to the user's needs.

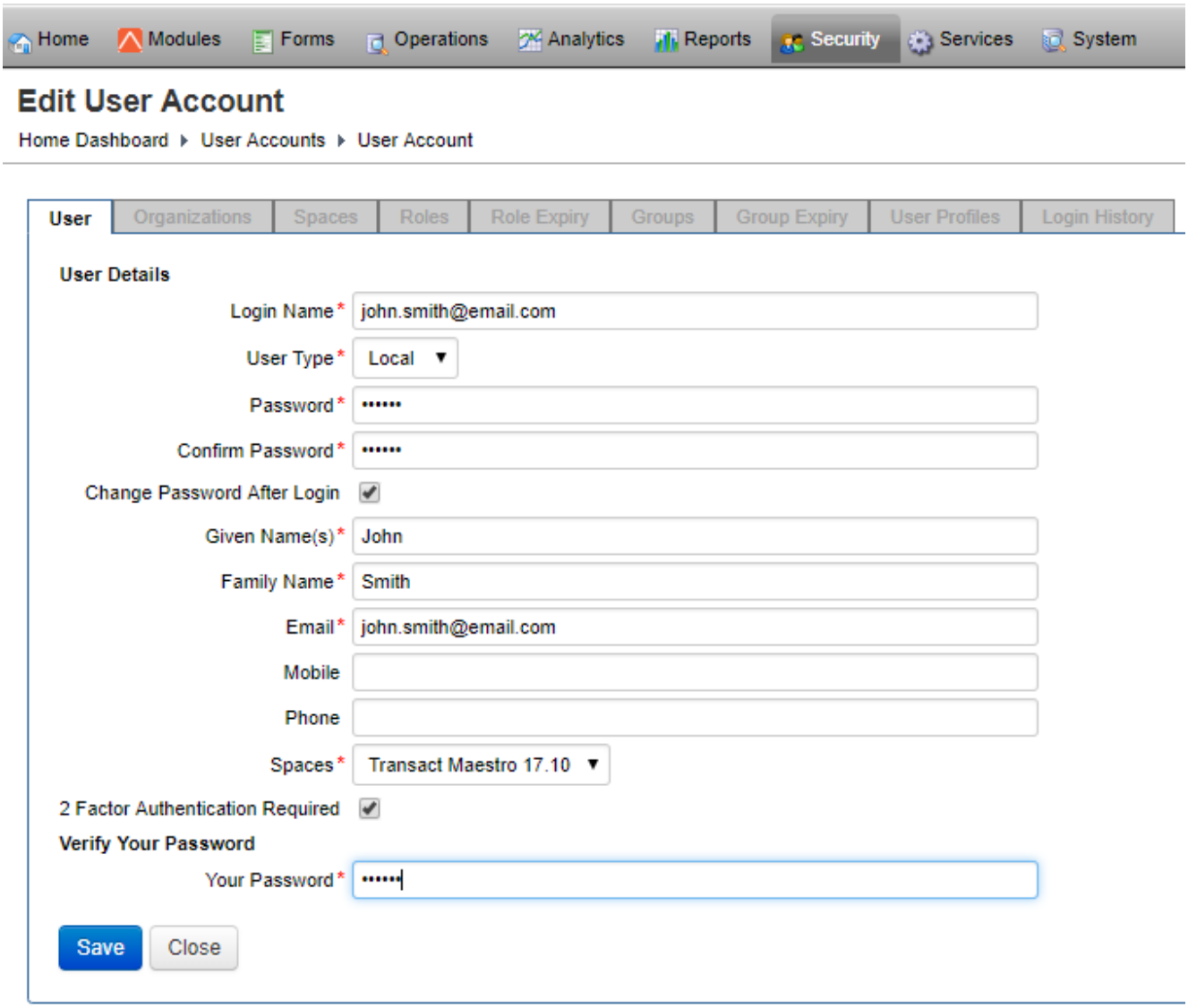
Use bulk import to create multiple users.

You must be an administrator to create a new user account.

Only a Maestro Organization User Manager can create a new user for their assigned organizations.

To create a user account:

1. Select **Security > User Accounts**.
2. Click **New**.



3. Enter a login name for a new user in **Login Name**. The user can use this login name to log into Manager, Journey Maestro, Composer and form spaces, but it must be the same as the Login Name used for the Maestro cloud instance. Otherwise, users will not be able to update Manager form versions built in Maestro.

> It is recommended to use an email address as a login name.

4. Select a user type from the **User Type** dropdown list. Configuration is different for each user type, so use the specific details as shown below.

- Local - user accounts managed by Manager. This is the default user type.
- **LDAP**[1] - user accounts managed externally by a Manager LDAP security manager, which must be correctly configured.
- **SSO**[2] - user accounts managed externally by a Manager SSO security manager, which must be correctly configured.

5. Enter a user account password in the **Password** field.

   The standard Journey Manager password requirements are:

   - Must be at least 12 characters in length.
   - Must not exceed 20 characters | 5.1 & Higher or 40 characters | 20.05 .
   - Must contain at least one or two letters and one digit.
   - Must contain at least one non-alphanumeric character such as %, &.
   - Must contain mixed case, one upper case, and one lower case.
   - Must not be a password existing in the Illegal Password Values list.

> An administrator can customize all above password requirements, except for the not exceeding 20 characters, for the Local Security Manager.

6. Enter the exact same password as was just entered in Password in Confirm Password.
7. Select **Change Password After Login** to force the new user to change their password when first logging in. They are unable to access restricted content until the initial password has been changed.
8. Enter a user's first name and middle names (if any) in **Given Name(s)**.
9. Enter a user's last name in **Family Name**.
10. Enter a valid user's email address in **Email**.
11. Enter a user's mobile number in Mobile. Only accepts numeric and must be at least 10 digits.
12. Enter a user's phone number in Phone. Only accepts numeric and must be at least 10 digits.
13. Select a form space, which you want the user to log onto, from the **Spaces** dropdown list. For example, select the **Transact Maestro** to provide the user with the access to the Maestro shared environment only. You will also need to assign the user to the other Transact Maestro form spaces for compatibility mode. For Maestro version 5.0, you need to add the **Transact Maestro 5.0** form space. This is the same for all other versions of Maestro as they have their own form spaces.

---

[1]Lightweight Directory Access Protocol (LDAP) s an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol network.
[2]Single sign-on (SSO) is a property of access control of multiple related, yet independent, software systems. With this property, a user logs in with a single ID and password to gain access to any of several related systems.

You can assign more spaces from the **Spaces** tab.

14. Select **2 Factor Authentication Required** to enable two-factor authentication for better protection of user logging to Manager. If the space has a security manager that requires Two-factor Authentication configured, then the **2 Factor Authentication Required** is selected by default. | 18.11 .
15. Enter your password in the **Your Password** field.
16. Click **Save** to update the changes.

Next, learn how to configure a user account.

Configure a User Account

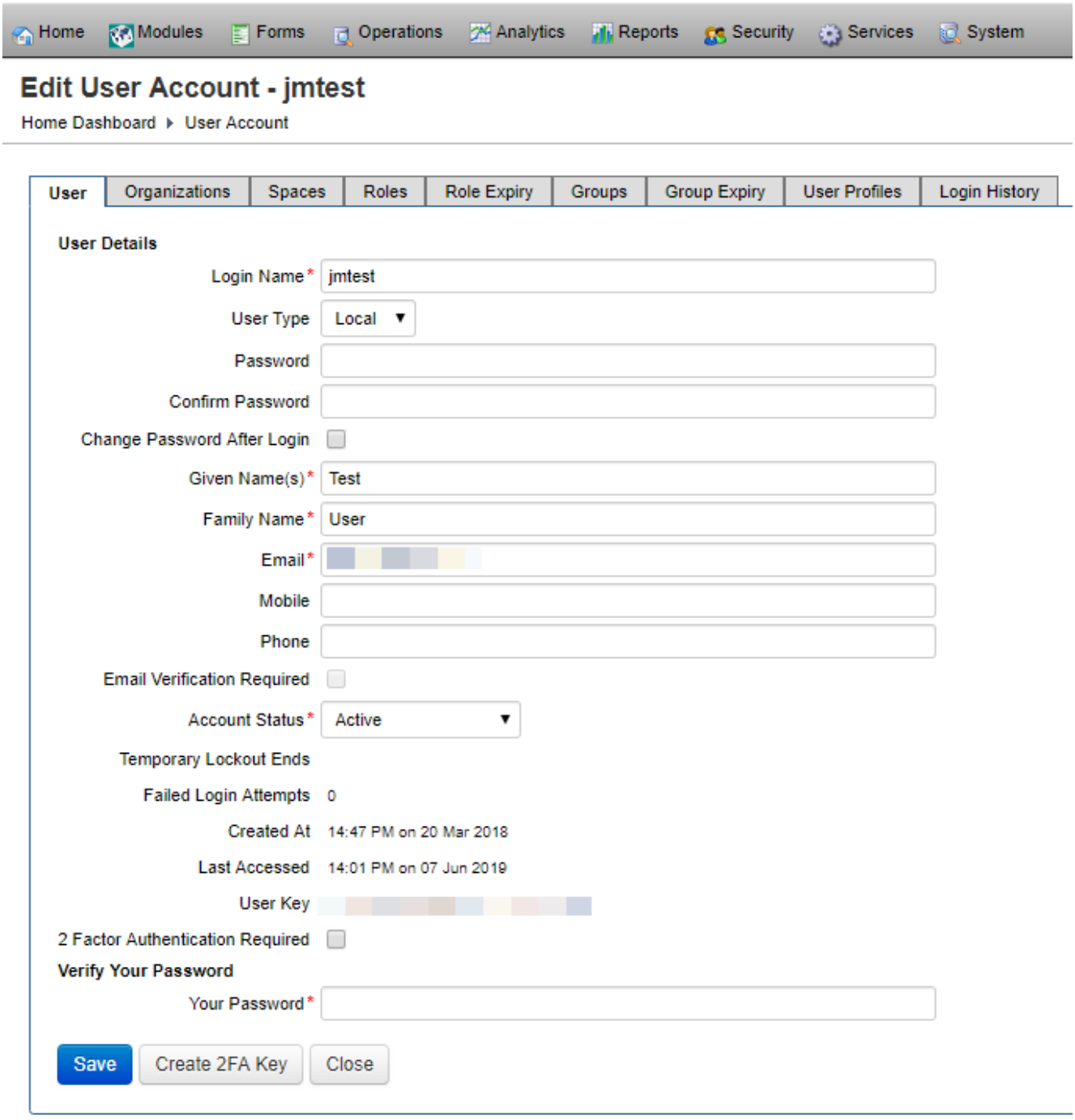
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

After you have created a user, Manager allows you to configure the user account to make sure the user can operate Temenos Journey platform products, such as Manager, Maestro, Exchange, and Analytics. You can also use this functionality to:

- Reset a password,
- Change a user account status,
- Change user's contact details,
- Deactivate a user account.

To configure a user account:

1. Select **Security > User Accounts**.
2. Locate a user account and click **Edit**.
3. Select the **User** tab to update user's details. For more information, see create a user account.



4. Select a user account status from the **Account Status** dropdown list, if user's status changed, for example, the account was locked.

   A user account status can be one of the following:

- Active
- Inactive
- Locked
- Locked Temporarily
- Pending
- Rejected

5. Click **Create 2FA Key** to generate a **2FA**[1] key and send it as an email to the user to complete the account configuration. If you have already configured 2FA, the **Reset 2FA Key** button is shown. | 18.11 Use this functionality to reset your verification code, for example, when you want to set up the 2 step verification in Work Space.

> You must select the 2 Factor Authentication Required checkbox for this process to work.

6. Click **Save** to update the changes.
7. Assign roles to a user account.
8. Assign organizations to a user account.
9. Assign form spaces to a user account.
10. Assign groups to user account.
11. Configure user role expiry.
12. Configure user group expiry.
13. Click **Save** to update the changes.

Next, learn how to create a user profile.

---

[1]Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.

Assign Roles to a User Account

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to assign roles to a user. You can use this functionality to:

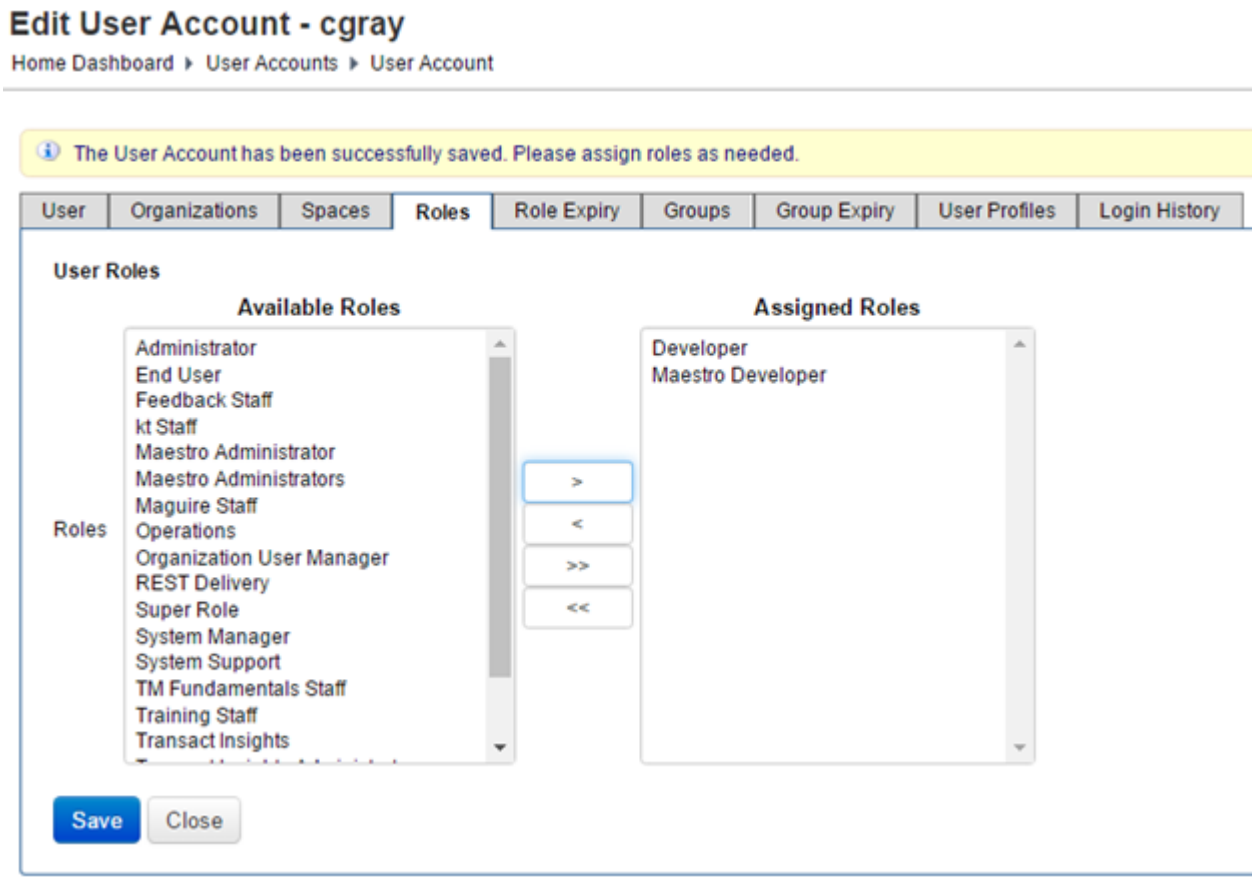
- Assign roles to a new user.
- Grant or revoke roles to an existing user.

Each role has a set of permissions associated with it, so by assigning a role, you enable certain product functions to this user. Consider the following before assigning roles to the user:

- As a minimum requirement, for most form users, such as your customers and applicants, logging into the form space grants them access, allowing them to select and complete forms. No roles are needed to be assigned to these user accounts.
- The types of activities the user is required to perform in each form space. Roles are specific to each form space.
- What each of the individual permissions associated with a role is allowing the user to do. Each role has a specific set of permissions controlling what is permissible within each space.
- The role of the administrator is a specialized role which gives the user permission to manage the security in Manager.

To assign roles to a user:

1. Select **Security > User Accounts**.
2. Locate a user account and click **Edit**.
3. Select the **Roles** tab to see the currently assigned roles.



4. Select one or more roles from **Available Roles** that you want to assign to the user.

> Use the `Shift` and `Ctrl` keys to select multiple spaces on a Windows PC.

5. Click **>** to move the selected roles from **Available Roles** to **Assigned Roles**.
6. Click **<** to revoke the selected roles from **Assigned Roles.**
7. Click **>>** to assign all roles.
8. Click **<<** to revoke all roles.
9. Enter your password in **Your Password** to make changes to the roles.
10. Click **Save** to update the changes.

> A user must log out and then log back into the form space for the changes to take effect.

> A user account's assigned roles can be changed at any stage, as required.

If you suspect that a user account may have been compromised, remove all their roles and spaces and make the user account inactive.

From the **Roles** tab, a global administrator will see and can assign all roles to a user account (provided their own roles contain the permissions around role assignment). An organization administrator, however, can only see and assign roles that are organization assignable roles. These are roles where the **Organization Assignable** checkbox is ticked. Any roles where the **Organization Assignable** checkbox is not ticked are hidden from and are unavailable to an organization administrator. Both global and organization administrators can change the role assignments for their own user account. An organization administrator assigning roles to their own account, the same rule applies where they can only see and assign organization assignable roles.

Next, learn how to assign organizations to a user.

Assign Organizations to a User Account

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to assign organizations to a user so you can control user access to forms belonging to a certain organization. Most administrators are assigned to specific organizations as organization administrators to provide system administration these organizations.

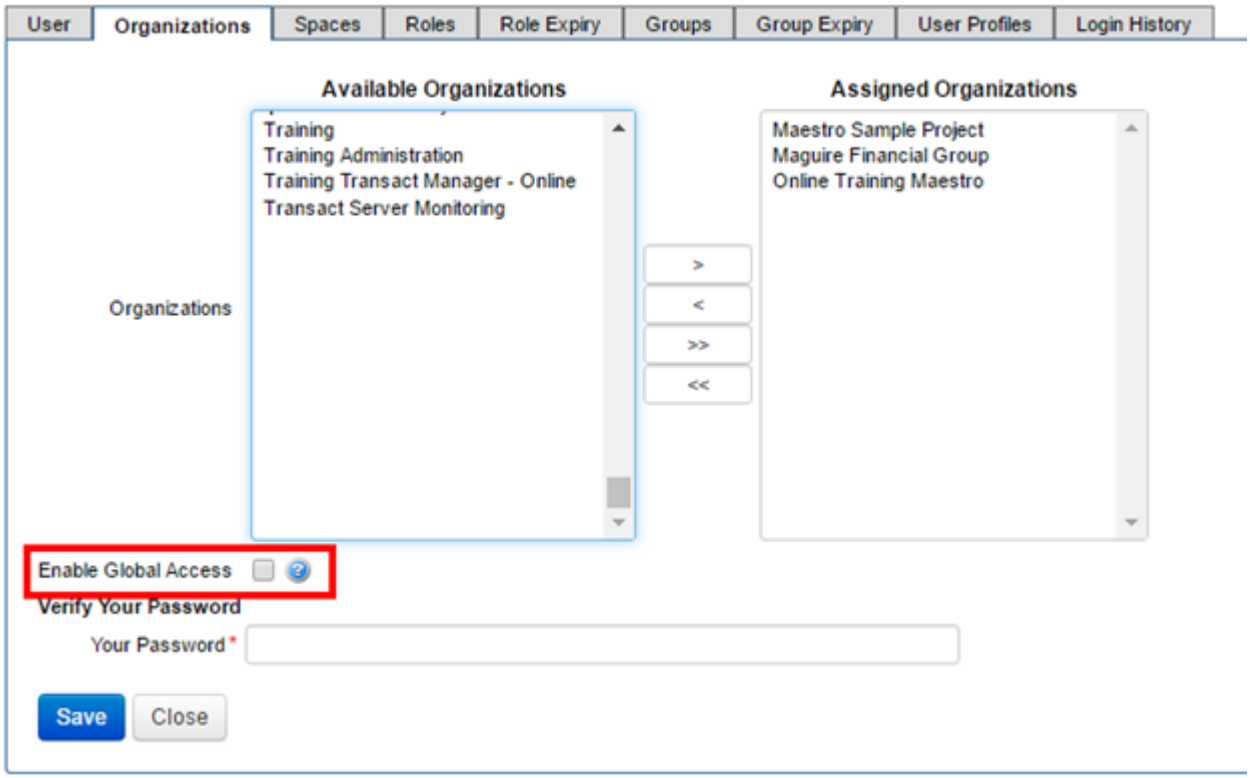
Consider the following before assigning or re-assigning organizations to a user account:

- Form builders must be granted access to organizations containing the forms they need to create or update.
- Administrators can be either an organization administrator (for most) or a global administrator. An organization administrator is only allowed to administer on behalf of specific organizations.
- Organization administrators have restricted access privileges, so they can only see the subset relevant to the specific organizations assigned to their user account.
- Organization administrators can only assign roles, which are organization specific, to other administrators. This is to prevent them from promoting themselves or others to become a global administrators.
- A global administrator is not constrained in any way and therefore can perform operation on behalf of all organizations.
- Operators are either granted access to specific organizations or to all organizations. Like administrators, operators monitor submissions and collaboration jobs on behalf of the specifically assigned organizations or all organizations, when globally enabled.
- Form space users don't need to assign any organizations to their user accounts as form spaces do not support restrictions based on organizations. When a user is granted access to a form space, they normally view and access all forms pertaining to any organization registered to host forms within that space, unless other restrictions are placed such as form groups.
- Within development and testing environments, it is often necessary to allow users a range of functions. This may be managed according to your enterprise's security policies.

Assign selected individual organizations to a user account to provide a fine-grained control of user access to the system.

To assign organizations to a user account:

1. Select **Security > User Accounts**.
2. Locate a user account and click **Edit**.
3. Select the **Organizations** tab to see the currently assigned organizations. For example, this is what an organization administrator will see.



4. Select one or more organizations from **Available Organizations** that you want to assign to the user account. This list depends on whether you are logged in as a global administrator or as an organization administrator. Global administrators have all organizations listed in available organizations section, but only those organizations granted to an organization administrator are listed.

> Use the `Shift` and `Ctrl` keys to select multiple spaces on a Windows PC.

5. Click **>** to move the selected organizations from **Available Organizations** to **Assigned Organizations**.
6. Click **<** to revoke the selected organizations from **Assigned Organizations**.
7. Click **>>** to assign all organizations.
8. Click **<<** to revoke all organizations.
9. Enter your password in the **Your Password** field to make changes to the organizations.
10. Select the **Enable Global Access** checkbox to allow only a global administrator to grant the global administrator privilege to another user. In this example, the organizations in the Assigned Organizations section will revert to the Available Organizations section and it is no longer possible to assign organizations individually to this user account. That is, either this user account is for a global administrator or for an organization user. For a global administrator user account, it is assumed that all organizations are assigned to it.

> This option is only available to a global administrator.

If you disable global access for your own user account, it can only be reinstated by another global administrator.

11. Click **Save** to update the changes. Now, it is possible to reassign individual organizations to this user account.

Next, learn how to view all user accounts.

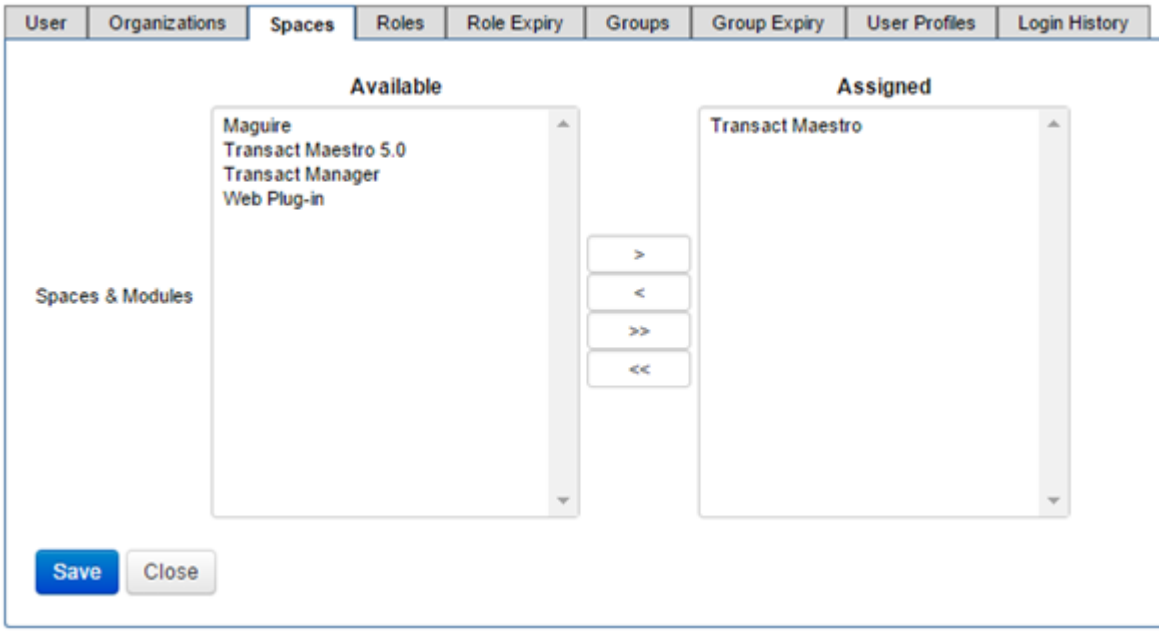
Assign Form Spaces to a User Account

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to assign form spaces to a user to grant or remove user access to form spaces

To assign form spaces to a user:

1. Select **Security > User Accounts**.
2. Locate a user account and click **Edit**.
3. Select the **Spaces** tab.



4. Select one or more spaces from **Available Spaces** that you want to assign to the user account. This list depends on whether you are logged in as a global administrator or as an organization administrator. Global administrators have all spaces listed in the Available Spaces & Modules section, but only those spaces granted to an organization administrator are listed.

> Use the `Shift` and `Ctrl` keys to select multiple spaces on a Windows PC.

5. Click **>** to move the selected form spaces from **Available Spaces** to **Assigned Spaces**.
6. Click **<** to revoke the assigned form spaces from **Assigned Spaces**.
7. Click **>>** to assign all form spaces.
8. Click **<<** to revoke all form spaces.
9. Enter your password in the **Your Password** field to make changes to the form spaces.
10. Click **Save** to update the changes.

Next, learn how to assign organizations to a user.

Provision a User Account

Journey Manager (JM) | System Manager / DevOps | 19.05

User Account Provisioning is a process that ensures user accounts are created, given proper permissions, changed, deactivated, and removed.

Provisioning users in Journey Manager is a frequent task, which an administrator performs to create new or migrate existing users from different environments. This process is straightforward, but you have to understand what roles, groups and permissions a user should have to utilize the product efficiently and securely. You also need to figure out what organizations and form spaces the user will need to access to be able to do the work. After that, setting up a user is as simple as following these steps:

1. Create a new or configure an existing user account.
2. Assign roles to the user account.
3. Assign organizations to the user account.
4. Assign form spaces to the user account.
5. Assign groups to the user account.
6. Configure user role expiry to harden the Manager security.
7. Configure user group expiry to harden the Manager security.

You can use the bulk import functionality to create multiple users including user's details, roles, groups and access configurations. This significantly simplifies user provisioning task. Finally, if a user is no longer required, you can deactivate it.

There are the following differences in provisioning different users, which are described below.

Provision a User Account for Accessing a Maestro Shared Environment Only

You may want to create users that can access and use only one product. Maestro is a good example as it is a product that can be run by itself and is used primarily by form developers, who don't need to use other products, such as Analytics.

Only a Maestro Organization User Manager can create a new user for their assigned organizations.

To configure a user:

1. Configure a user as described above.
2. Assign the **Maestro Developer** and **Maestro Administrator** roles to the user.

3. Assign the **Transact Maestro** form space to a user account. You will also need to assign the user to the other Transact Maestro form spaces, such as Transact Maestro 5.0, Transact Maestro 5.1, Transact Maestro 17.05, Transact Maestro 18.11, for compatibility mode. For Maestro version 5.0, you need to add the **Transact Maestro 5.0** form space because each Maestro has its own form space.

> There are multiple roles for Maestro users. Manager access is not required, so do not assign Manager roles.

Provision a User Account with the Organization User Manager Role

Manager has a set of roles that can be assigned to a user. One of these roles is the Organization User Manager role, which allows a user with this role to create new users, edit user details, and reset passwords.

To configure a user as the Organization User Manager:

1. Configure a user as described above.
2. Assign the **Organization User Manager** role to the user.
3. Assign the following form spaces to the user:
   - Maestro
   - Manager
   - Maestro 5.0
   - Maestro 17.10
   - Maestro 18.05

> If you use Maestro prior to version 18.05, you only need to assign the form spaces up to the version of Maestro you are using. For example, if you use Maestro 17.10, you need to assign both the Transact Maestro 5.0 and the Transact Maestro 17.10 form spaces to the user.

4. Assign the required organizations to the user.

> Do not assign global access to the user account.

Troubleshooting

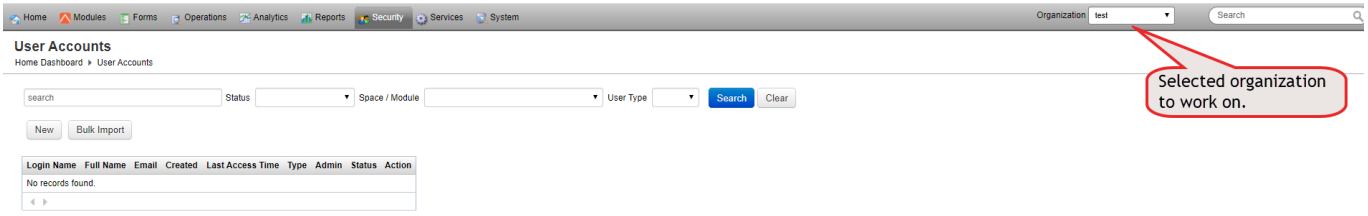
Sometime, after you have created a user account in Manager, it's still not visible while searching for it. There are a few primary causes, which are described below.

> An Organization User Manager (Org User Manager) is responsible for administering Manager users for a

company. They will be contacted first before raising a support request.

General Cause 1

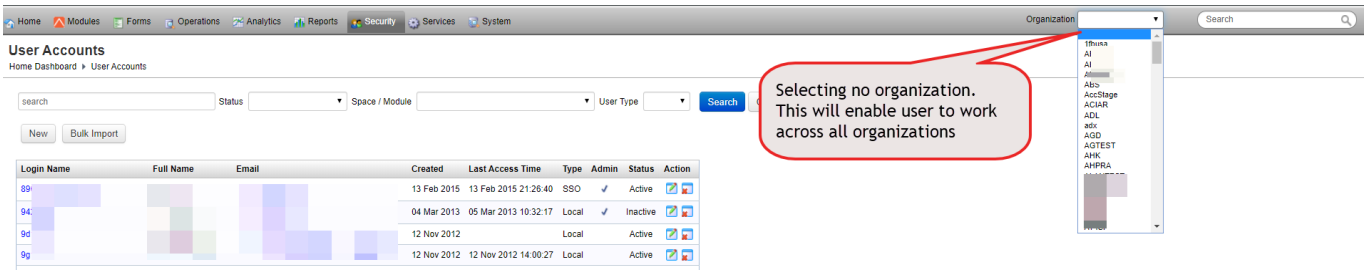
Incorrect organization is selected in the "Organization" field of JM menu bar (shown below).



Solution : When you do not see a drop down field like above, and usually see a fixed "Organization" name, it implies you have access to only one organization. Hence, this scenario is not applicable, please move on to next general cause.

If you see a drop down field like the above image, it implies you have access to multiple organizations. Please select an appropriate organization which is also assigned to the new user. After selecting the appropriate organization if you do a user account search the user shall be visible.

You can also leave the "Organization" field blank. If no organization is selected in the "Organization" field and it is left blank (see below), you will see the users for all organizations that you have access to in the search results. Hence, if unsure of user's assigned organizations please keep this field blank as shown below:



General Cause 2

The user account was created by someone and assigned to an organization which you don't have access to.

Solution : After checking the general cause 1, if you still cannot see the user in search results, please ask your Org User Manager to assist you in this regard. The Org User Manager should be able to assign the correct organization to the user. Please ensure that you have access to the same organization as the new user to make them visible in your search results.

General Cause 3

You have just created the user account but you cannot see it in search results . Usually, the user account was not assigned any organization.

Solution : In this scenario, the newly created user account does not belong to any organization hence, it is not visible to users who have access to particular organizations.

However, users with global access (generally, Org User Manager) can see the user and should be able to assign an appropriate organization to the new user account so that it is visible to you.

Provide your Org User Manager with the username of the newly created user so the account can be searched. Org User Manager should be able to edit the newly created user account and assign the organization as per business requirements.

Since you have created the user it is very likely you have administration rights for the organization, please ensure that the new user was assigned to your organization to make them visible in your search results.

> If the user is still not visible after above troubleshooting, raise a request with Temenos Support team using our support portal.

Next, learn how to view user accounts.

User Profiles Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager uses user profiles to represent various user contact details in a user account. These contact details are predominately used for identification purposes and prefilling forms. User contact details may be maintained by an administrator from within Manager or by a user with an account that supports user profile management. Users and Manager administrators can create and manage one or more user profiles for a user account.

When logged into a form space (and if the standard **Account** tab is available), users can:

- Update their contact details.
- Create more than one user profile.
- Elect another user profile to be the default, when more than one user profile exists.
- Delete a user profile.

An administrator is also able to view and manage user profiles.

When you create a user account, it automatically creates a default user profile named **My Profile**, which contains a copy of the following user contact details:

- Given Name
- Family Name
- Email
- Mobile

Next, learn how configure a user profile.

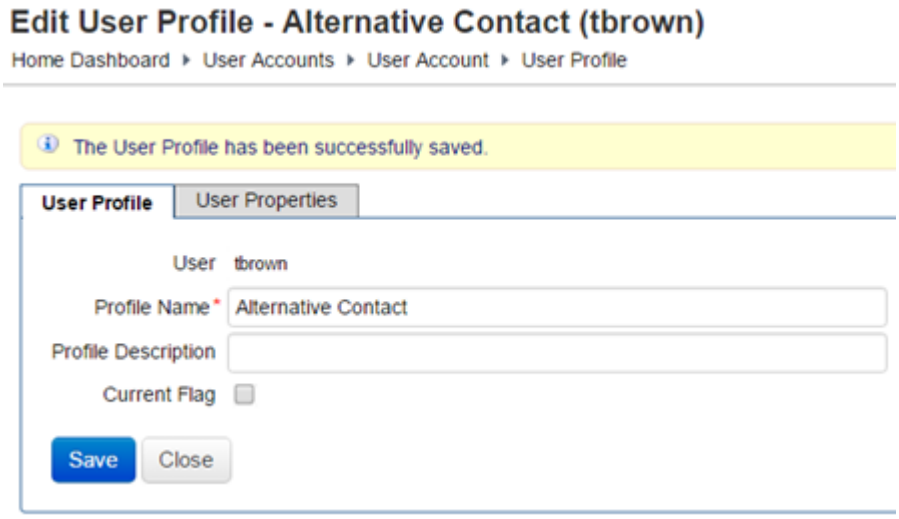
Create a User Profile

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

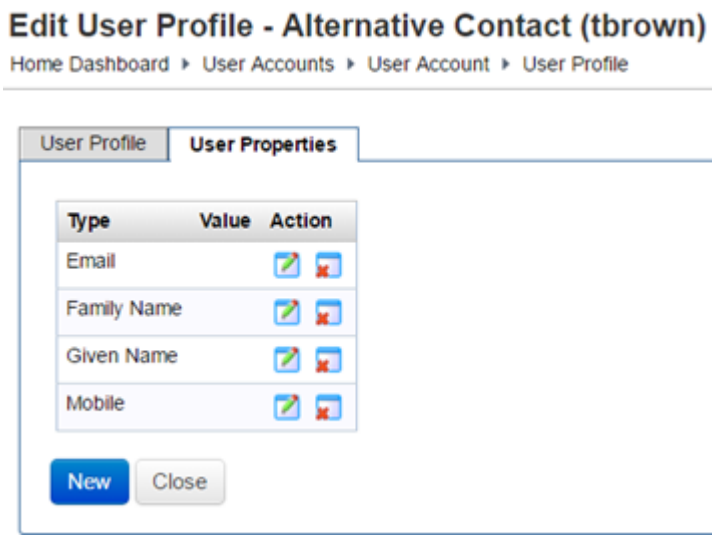
Manager allows you to create several user profiles for the existing user account.

To create a new user profile:

1. Select **Security > User Accounts**.
2. Locate a user account and click **Edit**.
3. Click the **User Profiles** tab to see all user profiles for this user account. The current user profile has a check mark against it it it is used as the default user profile for form prefill.
4. Click **New**.



5. Enter a unique profile name in the **Profile Name** field.
6. Enter a description in the **Profile Description** field.
7. Select **Current Flag** to make this the current user profile for this user account.
8. Click **Save** to update the changes.
9. Select the **User Properties** tab to list all user properties for this user profile, with blank values in the **Value** column.



10. Click **Edit** to set values for each of the user properties.
11. Select a property type and enter a value for the user property in **Value**, for example an email address.
12. Click **Save** to update the changes.

You can remove a user property, if it is not required. However, it's not recommended to delete a user property as this can cause the user profile to be inconsistent with other user profiles. If you want to remove the user property, leave its value field blank.

For prefilling a form with the user's contact details sourced from a default user profile, you need to map each data item, such as Given Name and Family Name, to target locations using form data configuration.

Next, learn about user property types.

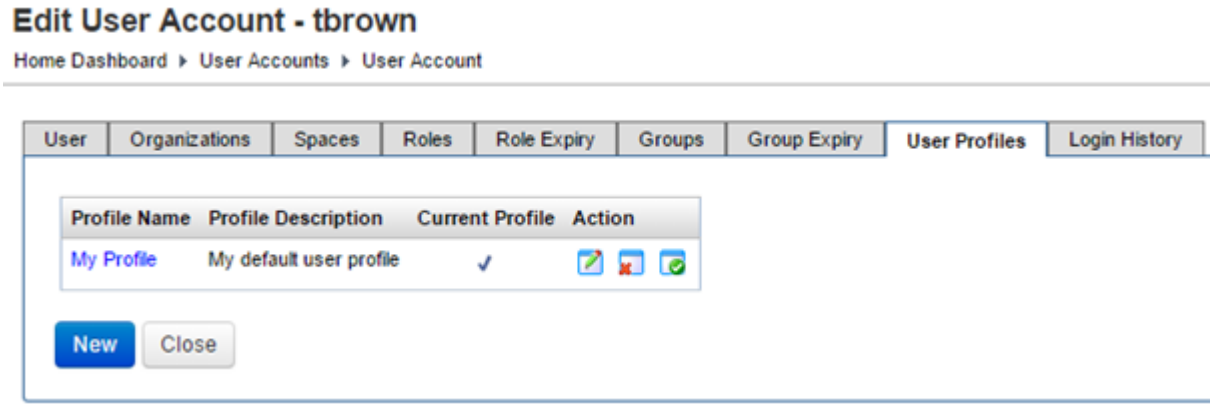
Configure a User Profile

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to configure and update the user profile of a user account.

To configure a user profile:

1. Select **Security > User Accounts**.
2. Locate a user account and click **Edit**.
3. Click the **User Profiles** tab to see all user profiles for this user account. The current user profile is selected in the **Current Profile** column, so it is used as the default user profile for form prefill.



4. Click **Edit** to update the profile.
5. Click **Remove** to delete the profile.
6. Click **Set as default profile** to set the profile as default, when there are more than one user profile for this user account.

Next, learn how to create a user profile.

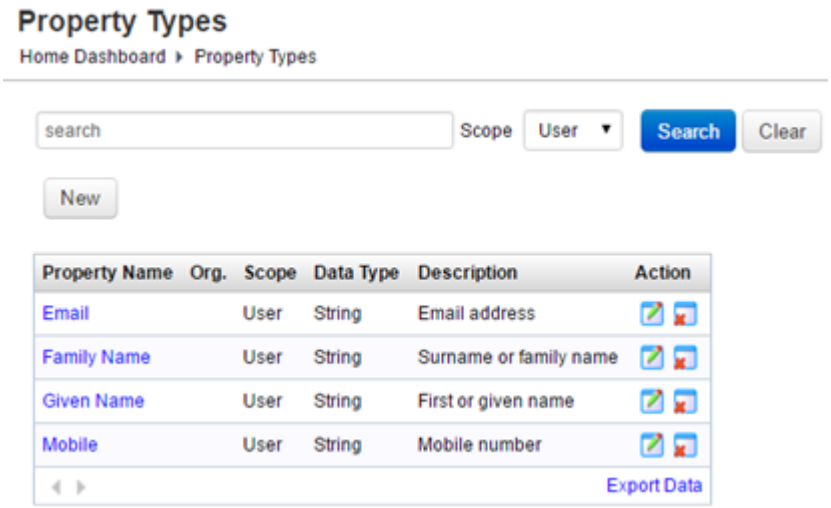
User Property Types

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A user profile is comprised of a set of fields that are defined and maintained as user property types in Manager. Property types define user property scope.

Out of the box, Manager comes with a standard set of user profile fields where each corresponds to a predefined property type of the user property scope. Each user property type is available to all user profiles for maintaining a user's contact details both in Manager (by an administrator) and the assigned form spaces (by a user).

Manager uses the following standard property types:



- Email
- Family Name
- Given Name
- Mobile

An administrator can customize a user profile by adding some new details that you want users to provide. This is done by creating a new property type with a property scope of User.

> Only update the contact details of a user via the user profile of a user account.

Administrators can update the Given Name, Family Name, Email, and Mobile details that are registered directly against a user account. However, these changes won't be automatically reflected in the corresponding fields against the default user profile because that user contact information is always sourced from the user's default user profile and not from the contact details.

Next, learn how to create a user profile.

Remove a User Account

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

If a user account is not required any more or you want to stop the user from logging into any spaces or modules, Manager provides you with two options:

- Deactivate the user account.
- Remove the user account.

> We recommend deactivating a user account rather than removing it.

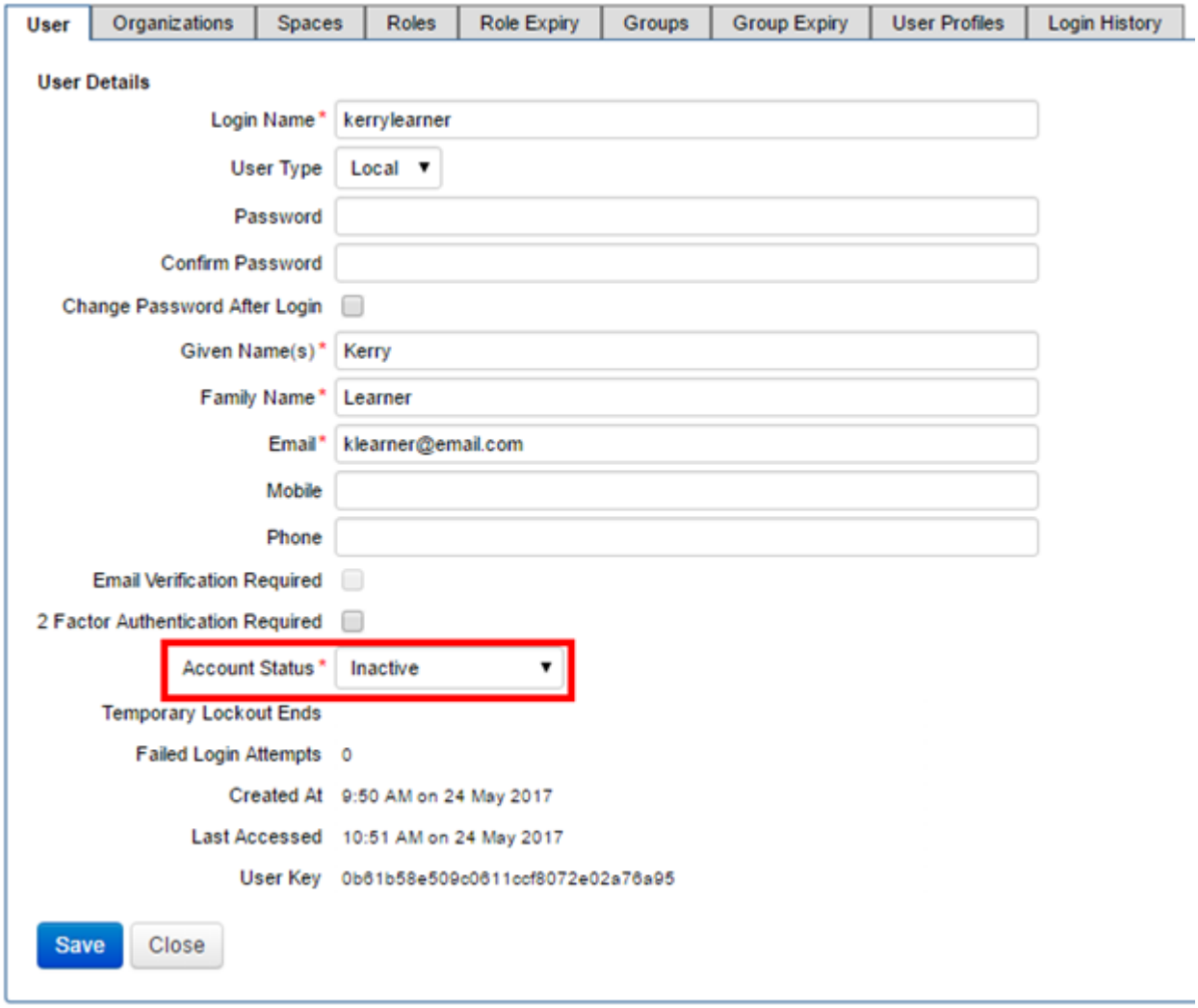
You can also configure your security manager in such a way that it automatically makes a user account inactive when the user hasn't logged onto the server for a specified duration of time.

Deactivate a User Account

You can deactivate a user account to prevent a user from logging in successfully using their credentials. Their details, however, will be preserved in Manager. Submissions made by a user are preserved even after their user account is deleted, but the submissions will no longer associated with a user account. Deactivation of a user account can be reversed by reactivating the user account.

To deactivate a user account:

1. Select **Security > User Accounts**.
2. Locate the user account and click **Edit**.
3. Click the **User** tab and select **Inactive** from the **Account Status** dropdown list.



4. Click **Save** to update the changes.

To reactivate the user account, select **Active** from the **Account Status** dropdown list.

Remove a User Account

When you remove a user account, Manager irrevocably removes all data associated with the user account, such as transactions and collaboration jobs. This action can't be rolled back.

To remove a user account:

1. Select **Security > User Accounts**.
2. Locate the user account and click **Remove**.
3. Click **OK** to confirm.

You can't remove the user account you are currently logged in as.

Even after a user account has been deleted, the changes that the user made to the system are still visible in the audit log.

Next, learn how to create a user account.

Bulk Import of User Accounts

Journey Manager (JM) | System Manager / DevOps | 20.05

Manager allows you to import multiple user accounts specified in an Excel spreadsheet to speed up user creation process. A bulk upload spreadsheet can include the following user's details:

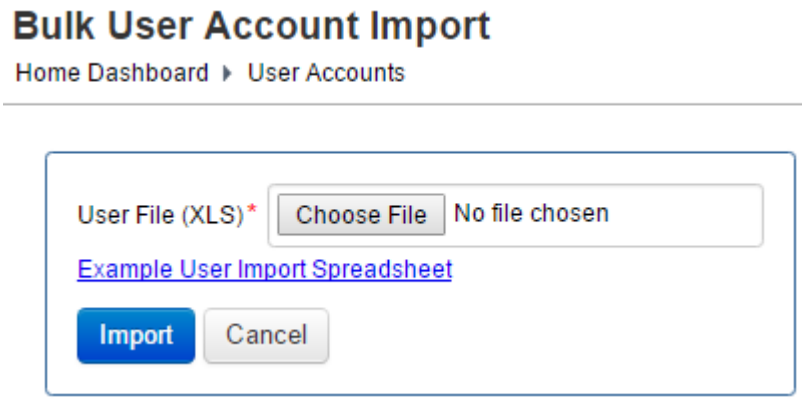
- Login Name
- Given Name
- Family Name
- Email
- Password
- Change Password
- Portals
- Send User Enrollment Approval Email
- Groups. This includes access check that an imported user can access a group. | 20.05
- Global Access. This includes access check that an imported user is global. | 20.05
- Org codes | 20.05
- Roles | 20.05
- Mobile

You should consider the following before running a bulk import:

- Existing users are ignored and won't be updated.
- Only local users will be created.
- Duplicates user accounts, identified by their **Login Names**, won't be imported.
- Users can only be associated with form spaces used to host forms, but not other spaces such as the Manager console. You can manually assign the missing form spaces after the importing.
- If a user who does the bulk import is not a global administrator, then it can't grant global access to users being imported.

To bulk import user accounts from an Excel spreadsheet:

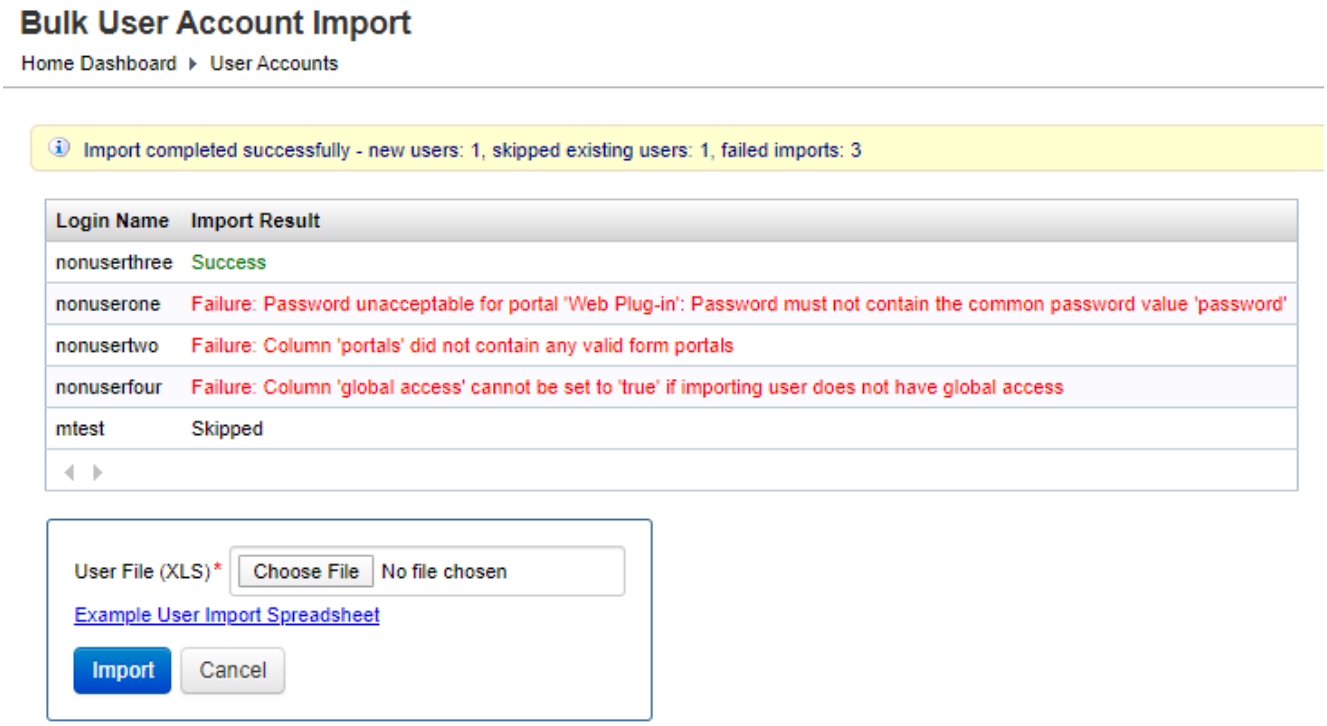
1. Select **Security > User Accounts.**
2. Click **Bulk Import**.



3. Click **Example User Import Spreadsheet** to download a sample **XLS**[1] file to assist you in entering user account details.

> If the XLS file doesn't follow the expected format, the bulk import will fail. Specifically, the header column names need to be correct and in a certain order.

4. Click **Choose File** and locate an XLS file you wish to import.
5. Click **Import**.



[1] is a file extension for a spreadsheet file format created by Microsoft for use with Microsoft Excel. XLS stands for eXceL Spreadsheet. Microsoft Excel files use a proprietary format for storing Microsoft Excel documents.

6. Check the import results. The successfully imported user accounts are displayed at the top. They are followed by users for which an error occurred, such as a mandatory column not being set. User accounts that were skipped because they already exist are displayed at the bottom.
7. For each error, you may choose to revise and fix your XLS import file and re-import it. Alternatively, you can create each user account manually.

Next, learn how to configure a user account.

Migrate User Accounts

Journey Manager (JM) | System Manager / DevOps | 21.05

Manager provides a mechanism to migrate user accounts from one environment to another, which significantly saves time to set up new servers. You can also configure and test user accounts in different environments before migrating them over to production.

To migrate user accounts:

1. Export local user accounts from a source environment
2. Check an exported XLS file and adjust it as required, for example, remove users you don't need in a new environment
3. Bulk import user accounts into a target environment using the exported XLS file
4. Adjust user configuration if needed

Next, learn how to create a user account.

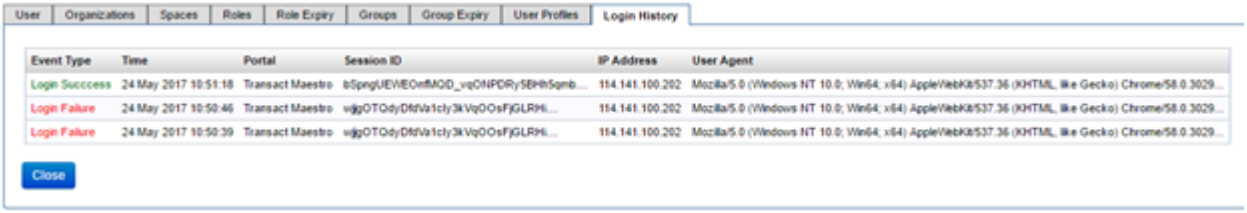
View User Login History

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view a list of all login attempts by each user or all users.

To view a user's login history:

1. Select **Security > User Accounts**.
2. Locate a user account and click **Edit**.
3. Select the **Login History** tab to see each login attempt and its details:



- Event Type is the status of the login attempt.

  A user login event type can be one of the following:

  - Account Locked Temp.
  - Login Failure.
  - Log Out.
  - Login Success.
  - Password Reset.
  - Session Expiry.
- Time is the date and time of the login attempt.
- Portal is the space or portal a user accessed or attempted to access.
- Session ID is a user's **session ID**[1] of the login attempt.
- IP Address is a user agent's **IP address**[2] used for this login attempt.
- User Agent is a **user agent**[3] used for this login attempt.

4. Click **Close**.

---

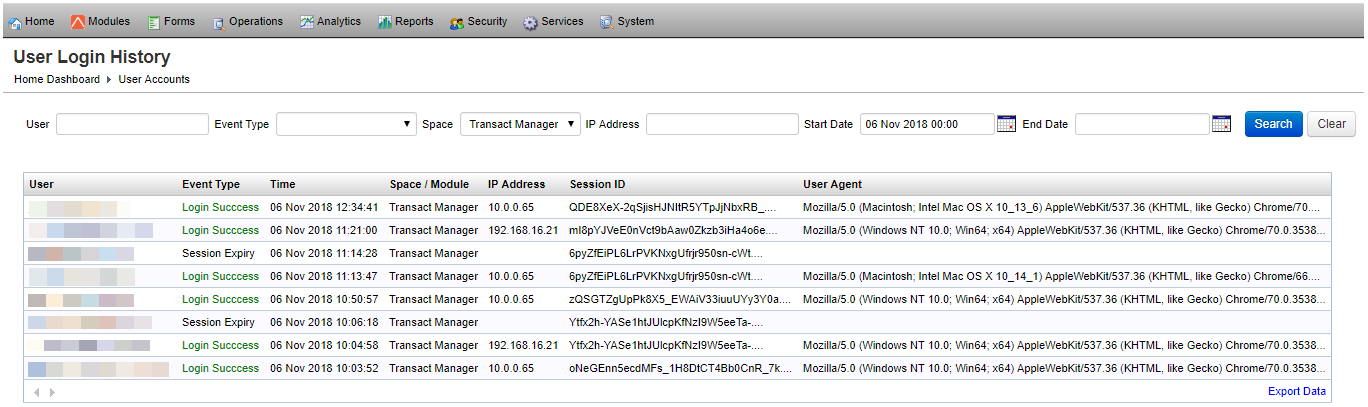
[1]Session ID is a unique number that a Web site's server assigns a specific user for the duration of that user's visit (session). The session ID can be stored as a cookie, form field, or URL (Uniform Resource Locator). Some Web servers generate session IDs by simply incrementing static numbers.

[2]IP address is a unique string of numbers separated by full stops that identifies each computer using the Internet Protocol to communicate over a network.

[3]User agent is software (a software agent) that is acting on behalf of a user. One common use of the term refers to a web browser that retrieves, renders and facilitates end user interaction with Web content.

To view all users' login history, select **Security > User Login History**.

User login history records are shown page by page, with the maximum number of records per page configured in the users preferences.



To filter or search the user login history by one or more criteria, specify the following settings and click **Search**:

- User account's name or part of a name.
- Event Type.

  A user login event type can be one of the following:

  - Account Locked Temp.
  - Login Failure.
  - Log Out.
  - Login Success.
  - Password Reset.
  - Session Expiry.
- Space.
- IP Address is a user agent's **IP address**[1] used for this login attempt.
- Start Date.
- End Date.

Click **Clear** to reset the search criteria.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to view user accounts.

---

[1]IP address is a unique string of numbers separated by full stops that identifies each computer using the Internet Protocol to communicate over a network.

User Visibility Rules for Restricted Content

Journey Manager (JM) | System Manager / DevOps | 19.05

Manager implements rules to ensure that users of the Workspaces portal can't access restricted content, such as organizations, groups, users, forms, and form categories configuration. The rules can be summarized below:

1. A global user, with the global access enabled, will see all organizations, groups, users, forms, and form categories information, if they are assigned to the Workspaces portal.
2. A user belonging to an organization without any form categories won't see any categories.
3. A user belonging to many organizations will see all the categories associated with these organizations.
4. A users without an organization won't see any organizations, groups, users, forms, and form categories information.
5. A user can see organizations they are assigned to and should be part of the Workspaces portal.
6. A user can see forms available to the organizations where the user has access to and is part of the Workspaces portal.
7. A user can see users available to the organizations where the user has access to and is part of the Workspaces portal.

Manager also maintains a separate set of rules for restricting:

- User access to collaboration jobs
- User access to transactions

Workspaces APIs uses these rules to filter the category, form, group, org, user, and txn query responses.

Next, learn more about user transaction security.

User Visibility Rules for Collaboration Jobs

Journey Manager (JM) | System Manager / DevOps | 19.05

Manager allows a user to view collaboration jobs in the Workspaces portal and retrieve them using the APIs, if the following rules apply:

1. A user must have the Collaboration Job View permission for the form space to see collaboration jobs.
   - Only collaboration jobs with the **In Progress** status are further selected.
   - Only the collaboration jobs that are in the same form space as the user are selected. If this is not defined, this rule is not applied.
2. Manager further refines rules to select a subset of collaboration jobs from the collection above:
   - The collaboration jobs have no groups assigned or if there are groups assigned, the user is in one of those groups and
   - The user does not have global access then the user must be in a organization that the collaboration job is in.

To explain how these rules work, we will list several users, collaboration jobs, and their combinations showing whether a user can view these collaboration jobs or not.

The first table shows users with the access to four organizations, some of which have global access, and two groups.

	Org1	Org2	Org3	Org4	Global Access	Group1	Group2	Permission
User1								Collaboration Job View
User2								Collaboration Job View
User3								Collaboration Job View
User4								Collaboration Job View
User5								Collaboration Job View
User6								Collaboration Job View

User7	Collaboration Job View
User8	Collaboration Job View
User9	

The second table shows whether a user can see transactions or not. collaboration jobs belonging to one of the two organizations and two groups.

	Org1	Org2	Group1	Group2
Job1				
Job2				
Job3				
Job4				
Job5				

A collaboration job is limited to one organization but can have any number of groups.

This table shows whether a user can see collaboration jobs or not.

	Job1	Job2	Job3	Job4	Job5
User1					
User2					
User3					
User4					
User5					
User6					
User7					
User8					
User9					

This table helps you understand why users might not see some or any collaboration jobs in their work spaces, so you can review the user role, group, and permission configurations to resolve the access issues. To illustrate this, we have provided a few examples:

- The User 1 starts working on the Job 1 collaboration job, but can't continue with it. The User 3 has right permissions to work on the Job 1, so the User 3 can complete with the work the User 1 started.
- The User 1 can't see the Job 3 collaboration job, event though it is in the same Org1 organization, because it is assigned to the Group 3.
- The User 9 can't see any collaboration jobs, because this user doesn't have the Collaboration Job View permission.

Next, learn how to view roles.

User Visibility Rules for Transactions

Journey Manager (JM) | System Manager / DevOps | 19.05

Manager doesn't maintain relationships between users and organizations to determine which transactions a user can see when using the Workspaces portal or the APIs. Instead, Manager defines a set of rules to filter out transactions that a user can see:

1. Manager applies the Help Desk View permission to select an initial collection of transactions. A user can see the whole or part of this collection depending on the following conditions:
   - A user with the Help Desk View permission can see transactions with a group, user or both.
   - A user without the Help Desk View permission by default can only see transactions if they are allocated to it or if they are in the same group.
   - A user with the Help Desk Authenticated Edit permission can see other users' authenticated transactions.
2. Manager applies more rules to select a subset of transactions from the collection above:
   - A global user, with the global access enabled, can see all above transactions.
   - Otherwise
   - A user with a limited number of organizations can see transactions from above for those organizations only.
   - A user which is not global and has no organization can see only own transactions from above.

To explain how these rules work, we will list several users, transactions and their combinations showing whether a user can view these transactions or not.

The first table shows users with the access to organizations, some of which have global access, groups and the Workspaces permissions assigned to them.

	Org-1	Org-2	Org-3	Org-4	Org-5	Org-6	Global Access	Group1	Group2	Group3	Group4	Permission
User-1												
User-2												
User-3												

User-4	Help Desk Authenticated Edit
User-5	Help Desk View
User-6	
User-7	

The second table shows transactions belonging to an organization, a group and having a user assigned.

	Org1	Org2	Org6	Group1	Group2	Group3	Group4	User1	User2	User6	User7
Anon Txn											
User-1 Txn											
User-2 Txn											
Group1 Txn											
Group2 Txn											
Group3 Txn											
Gro-											

up4 Txn

This table shows whether a user can see transactions or not.

	Anon Txn	User1 Txn	User2 Txn	Group1 Txn	Group2 Txn	Group3 Txn	Group4 Txn
User1							
User2							
User3							
User4							
User5							
User6							
User7							

This table helps you understand why some users don't see some or any transactions in their space, so you can review the user role, group and permission configurations to resolve the access issues. To illustrate this, we have provided a few examples:

- The User 1 creates an anonymous transaction by filling a form, but can't continue with it, so the User 1 calls the help desk. The User 5 is a help desk user with the Help Desk View permission, so the User 5 can see the transaction and therefore can help the User 1 to complete it.
- The User 7 can see the Group4 Txn, because it's assigned to this user.
- The User 3 can't see any transactions, because there are no transactions belonging to Org3 organization, and the user doesn't have a global access and doesn't belong to any group.

Next, learn how to view roles.

Roles Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A role in Manager contains a set of permissions that allow a user to perform specific functions within a form space or module. Manager has the following set of standard roles automatically created and configured during its first start-up. You can use the standard roles, customize the standard roles, or create new roles.

Root Administrator

This is the most powerful role in Manager. Root administrators are assigned all permissions related to the Manager console and can access all restricted content. Critical operations, such as removing organizations and editing deployment properties, should be accessible to only these users. Root administrators can create new roles and customize any existing ones.

Developer / System Manager

These two roles have identical permissions. Developer/System Manager users have a wide range of permissions available to them to manage Manager configurations. Users in these roles can edit organizations, forms, and all related entities. Developers and System Managers are also able to view requests, view submissions as well as view a wide range of system configuration settings. These roles are extremely powerful but cannot access submission data and user accounts and are not able to modify roles.

Operations

Organization operation administrators can view submissions (included submitted XML data) and reports as well as various logs. Organization operation administrators are given permission to handle support requests submitted by space users.

However, they cannot access forms and associated configurations, nor can they access the details for user accounts stored in Manager.

Organization User Manager

Organization user administrators can create and configure users for their organization. This is a role that might be used in addition to other roles to give existing administrators access to user accounts, though it can also be used as a stand-alone role.

System Support

The system support role can be used by the support staff for debugging purposes. By default, this role has extensive read-only permissions to aid in debugging. However, XML data submitted by users cannot be viewed.

Transaction Data Access

This role provides access to transaction data entered by users and allows administrators to edit this data and make submission status changes. As this role provides access to sensitive information, this role should be assigned carefully.

Next, learn how to view roles.

Standard Roles

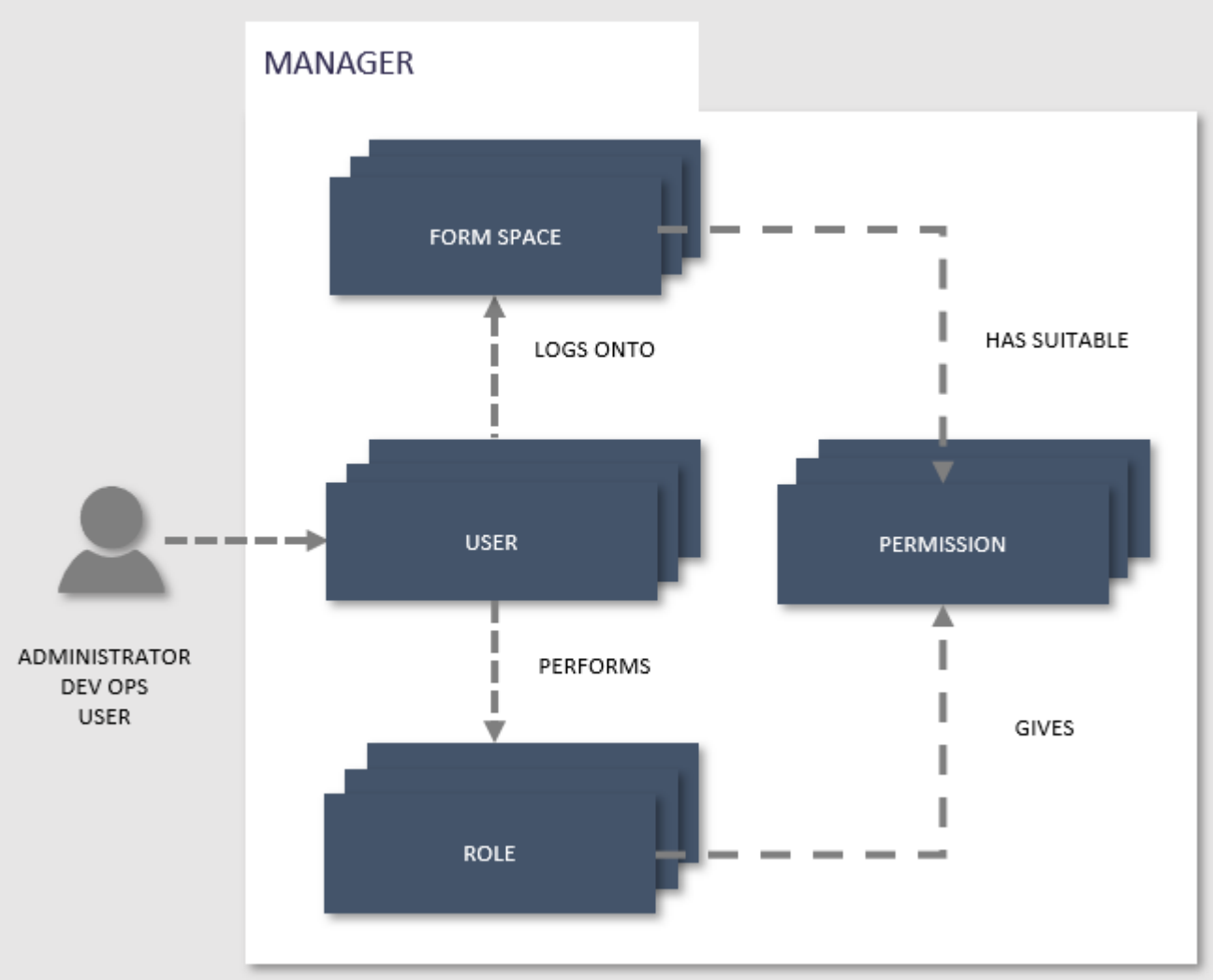
Journey Manager (JM) | System Manager / DevOps | 21.11

Manager provides roles and permissions to control user access to form spaces and modules. You can use this functionality to grant users specific access that matches the functions they are permitted to perform within each space or module. Manager checks what privileges a user has when logged into a form space or module. The roles granted to a user determine these privileges and controls what they can see and do within that space or module.

Before either adding extra roles or granting roles to your users, it is important to understand the following concepts:

- A user, via their user account, can be granted one or more roles.
- A role sanctions a group of permissions related to a function that is to be performed within a space or module. That is, a role is a group of permissions that is relevant only to the intended space or module.
- Each space or module will have a list of valid permissions available to assign to a role. That is, the role may only grant the types of access relevant to that space or module.
- Each role is mostly for one space or module but can be for more than one.
- Each permission is normally for one space or module but can be for more than one.

The diagram below depicts the conceptual data model for users logging into spaces or modules and their required roles and permissions.



Out of the box, Manager comes with the standard set of roles. Each role is already configured with its permissions for each standard form space and module. It is possible to alter the standard roles as well as add new ones. However, it is recommended to preserve the standard set and, only when essential, add new roles. It is also important to understand what privileges each of the roles provide before assigning them to your users.

Manager has the following standard roles:

Role	Description
Administrator	This is the most privileged role. There should be minimal need to use this role once the system has been installed, especially not in Production. The exception would be for the duration needed to perform a deployment where organizational or user roles need to be altered or imported. Regular form upgrades can be done with the System Manager role.
Avoka Exchange	Allows a user to review and install Avoka Exchange Components.

Developer	This is a role for users who will develop and manage applications. This role is identical to the System Manager role but has been retained for historical reasons. The Developer role should not be used on Production. System configuration and installation in Production can be achieved using the System Manager role.
Maestro Administrator	Allows a user to use the developer functions, plus create and delete projects, import and export files and libraries.
Maestro Developer	Allows a user to use most of the functions. This is the default access.
Maestro SCM Design	Allows a user to use Maestro SCM on Designs.
Maestro SCM Org Library	Allows a user to use Maestro SCM on Organization Libraries.
Maestro SCM Project	Allows a user to use Maestro SCM on Projects.
Maestro Site Administrator	Allows a user to administer the Maestro Instance at a global level.
Maguire Staff	Allows a user to use Works Space form space.
Manager	This is a role for users who will perform operations monitoring and management of submission transactions and collaboration jobs. This role also has limited access to customer submission data, such as receipts. Allows a user to operate a product as a Manager.
Operations	Allows a user to monitor and manage form transactions
Organization Administrator	Allows a user to test Users and Permissions.
Organization User Manager	This Allows a user to manage user accounts for their organization.
REST Delivery	Allows a user to perform submission delivery by invoking REST Delivery Service.
Security Manager Role	Allows a user to test Security Manager Permission.

Service Test Administrator	Allows a user to test Users and Permission service Definition.
ServiceUser	Allows scheduled Groovy services run as ServiceUser, which defaults to a admin level role. This role can be limited to enhance the security around these services, if required. | 21.11
System Manager	This is a role for users who are almost as privileged as an administrator, except they cannot view submission data or create users and roles. This role should only be used where extensive system access is required.
System Support	This is a role for users to be able to do basic troubleshooting and triage of issues. This is the least privileged role in Manager.
Test Role	Allows a user to run test cases.
Transact Insights	Allows a user to use Journey Analytics.
Transact Insights Administrator	Allows a user to change global settings for Journey Analytics.
Transaction Data Access	This is a role for users to be able to view the form submission XML and receipt data. This is sensitive personal data that needs to be protected, as it allows a user to access private customer form transaction data.
Work Space Staff	For business Work Space users
WorkSpaces Staff	For business Workspaces users.
WorkSpaces Api Test Role	Allows a user to test Workspaces API Test.

The roles depend on the functions that each team member performs. The following table is an example of how your team can be set up with specific roles:

Teams	Development	UAT	Staging	Production
Implementation				

Form Builder	System Manager. Transaction Data Access	System Manager. Transaction Data Access	No Access	No Access
Manager Developer Tech Lead	Administrator	Administrator	Administrator Transaction Data Access	No Access
Manager Tester Operations	Operations	Operations	Operations	Operations
System Administrator	Administrator	Administrator	Administrator	System Manager. Organization User Manager
Form User Support Operations	Operations. System Manager	Operations. System Manager	Operations. System Manager	Operations. System Manager
Temenos Teams Support Engineers. Tech Lead	Administrator	Administrator	System Manager	System Operator
Development Teams	Developer	Developer	No Access	No Access

Next, learn about permissions.

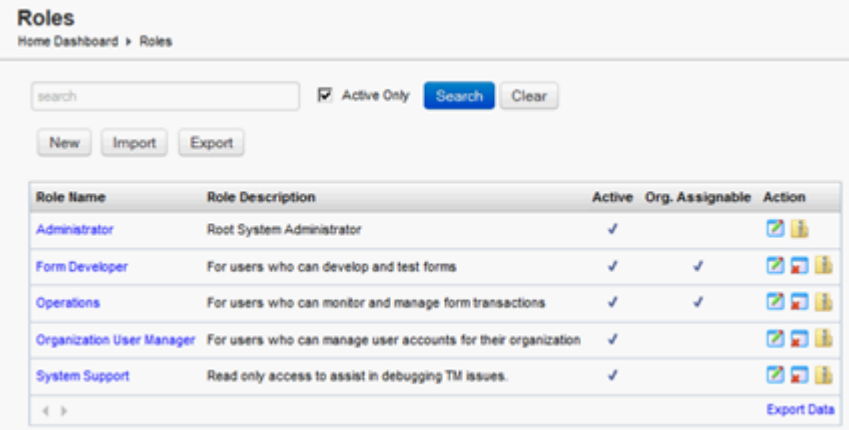
View Roles

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows administrators to view all available roles, so they can grant and revoke a user's roles and privileges.

To view all configured roles, select **Security > Roles**.

Roles are shown page by page, with the maximum number of records per page configured via the user's preferences.



To filter or search the roles by one or more criteria, specify the following settings and click **Search**:

- Role's name or part of a name
- Space
- Active Only

Click **Clear** to reset the search criteria.

You can also perform the following actions:

- Create and configure a role
- Remove a role
- Export roles
- Import roles

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Only global administrators can view all existing roles.

Organization administrators can view, edit, import and export only organization assignable roles, which are any organizations having **Org. Assignable** column ticked.

Next, learn how to assign permissions to a role.

Create a Role

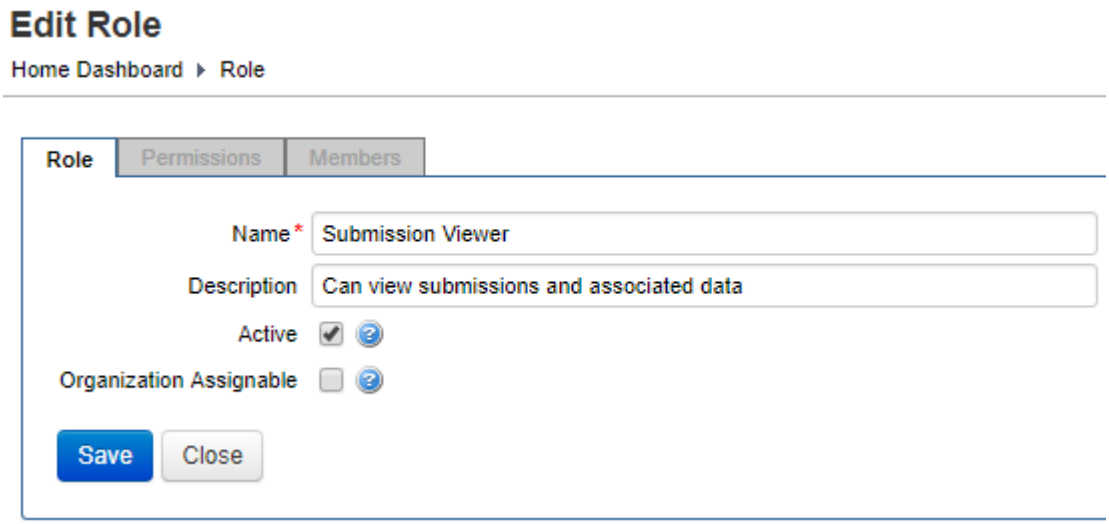
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create new roles. You can use this functionality to:

- Accommodate new customer requirements.
- Adjust existing roles to the customer organization hierarchy.

To create a new role:

1. Select **Security > Roles**.
2. Click **New**.



3. Enter a unique name for the new role in the **Name** field.
4. Enter a suitable description in the **Description** field.
5. The **Active** checkbox is selected by default, but you can clear it to disable this role for the time being.

Inactive roles can't be assigned to user accounts until they have been reactivated.

6. Select the **Organization Assignable** checkbox to permit organization administrators to assign this role to user accounts.

If you want only global administrators to be able to assign this role to user accounts, don't select **Organization Assignable**, so an organization administrator won't see it in their list of roles. If you don't select it, even organization administrators with the permission to assign roles will be unable to see or choose the role. For example, a global administrator role with all privileges will most likely not be organization assignable.

7. Click **Save** to update the changes.
8. Assign permissions to the new role.

> The **Submission Viewer** role can only be assigned to a user account by a global administrator.

Next, you will need to assign permissions to a role.

Manage Role Members

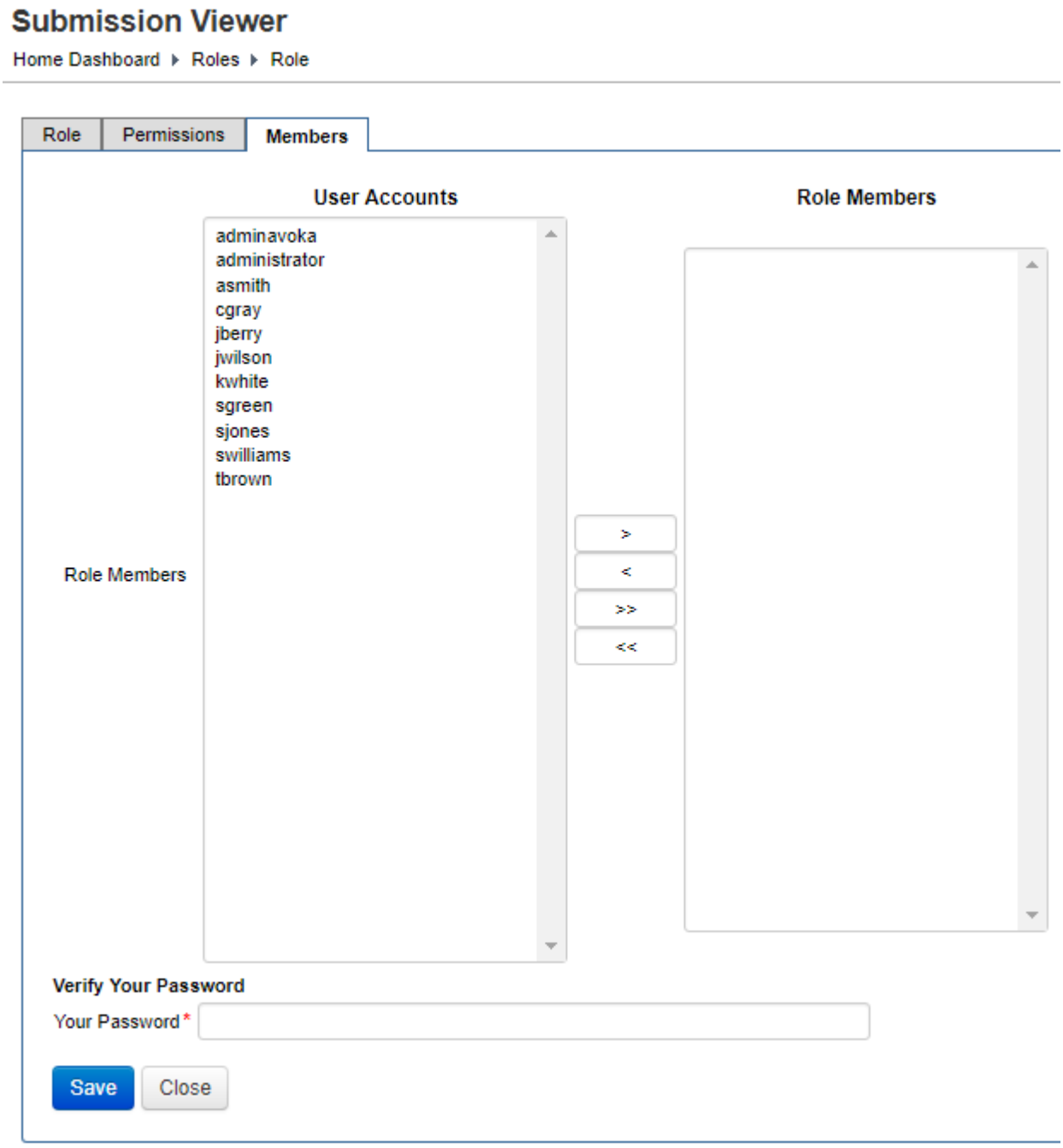
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows an administrator to view and modify the list of user accounts that are currently assigned to a role known as role members. You can also view and assign role members from the individual user accounts.

After you have created a role, you can assign role members to it. As an ongoing maintenance, you can assign new members or remove existing members for an existing role.

To assign role members:

1. Select **Security > Roles**.
2. Locate a role.
3. Click the **Members** tab to view role members. The **Role Members** column displays users that are currently assigned the selected role and the **User Accounts** column displays users you can assign to the selected role.



4. Select one or more users from User Accounts to add them to the role or remove them from it.

> You can use the **Shift** and **Ctrl** keys to select multiple permissions on Windows PC.

5. Click **>** to move the selected user accounts from **User Accounts** to **Role Members**.
6. To revoke any assigned user accounts, select the user accounts from **Role Members** and click **<**.
7. To assign all user accounts, click **>>**.
8. To revoke all user accounts, click **<<**.

9. Provide your password in the **Your Password** field, for security reasons.
10. Click **Save** to update the changes.

> All user accounts assigned to a role are loaded when the user logs in. Any changes will be visible to users as soon as they log out and then log back into the space.

Next, learn how configure role expiry.

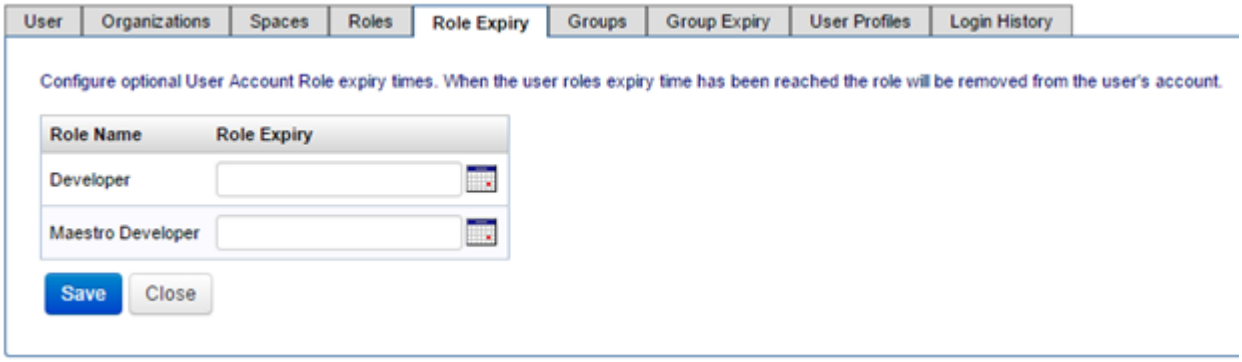
Configure Role Expiry

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to enforce an expiry time to a user's role. You can use this functionality to temporarily assign a role to a user account to allow a support team to assist in debugging a problem.

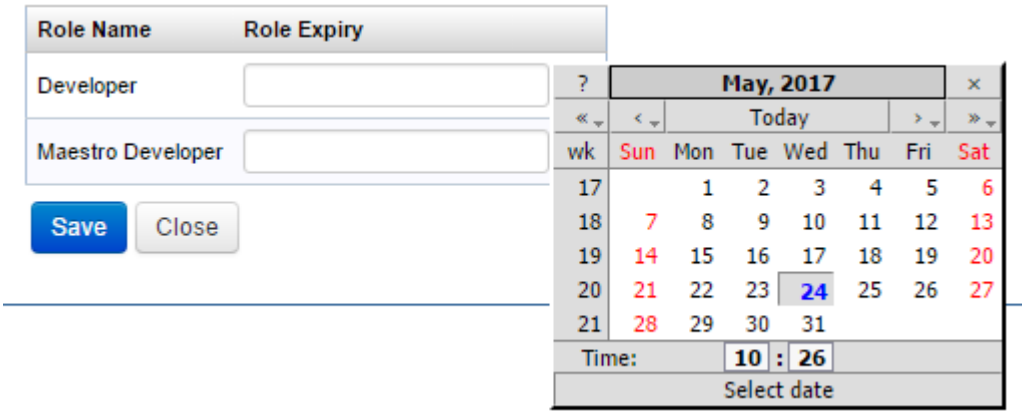
To enforce a time limit on a user's role:

1. Select **Security > User Accounts**.
2. Locate a user account and click the **Role Expiry** tab.



All roles assigned to this user account are listed under **Role Name** column. Any temporary role assignments show an expiry date in the **Role Expiry** column.

3. Click Calendar in the **Role Expiry** column.
4. Select a date and time for when this user's role membership must expire in a Calendar pop up. You can select the time first, as the pop up automatically closes on selection of a date.



To make a user's role membership permanent again, remove the expiry date and time from the **Role Expiry** column.

5. Click **Save** to update the changes. When the expiry time arrives, the Security Policy Manager background job removes this user's role membership from the user account.

Next, learn how to create a user account.

Remove a Role

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

If a role is not required any more, Manager provides you with two options:

- Deactivate a role.
- Remove a role.

> In general, we recommend deactivating a role rather than removing it.

Deactivate a Role

When you deactivate a role, Manager does the following:

- The role details are preserved, so the role can be reactivated if needed. On reactivation, a role can once again be assigned to a user account.
- User accounts remain assigned to the role, but its permissions are revoked.

To deactivate a role:

1. Select **Security > Roles**.
2. Locate the role and click **Edit**.
3. Clear the **Active** checkbox.
4. Click **Save** to update the changes.

> To reactivate the role, select its **Active** checkbox.

Remove a Role

When you remove a role, Manager irrevocably removes all data associated with the role, so it can not be rolled back.

> You can't remove the system-generated Administrator role.

To remove a role:

1. Select **Security > Roles**.
2. Locate the role and click **Remove**.
3. Click **OK** to confirm.

Next, learn how to create roles.

Export Roles

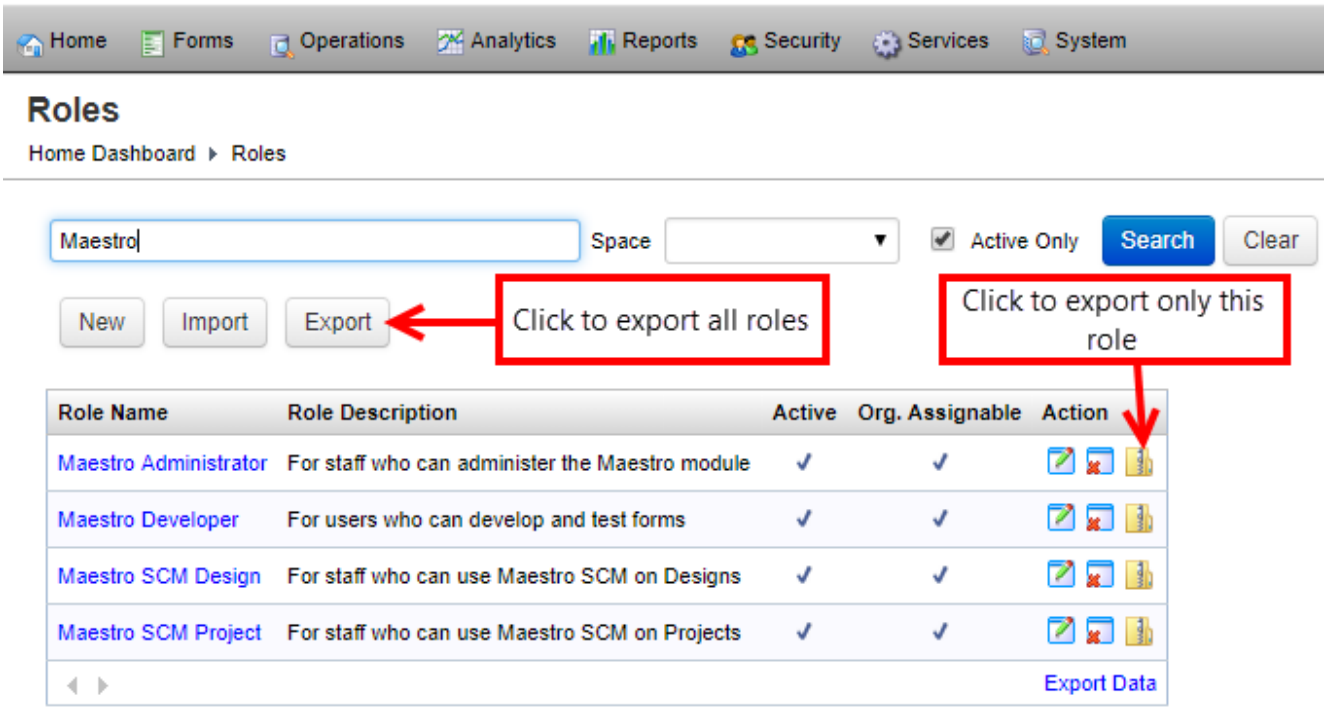
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows administrators to export roles from one environment and import them into another. You can use the export functionality to:

- Back up or archive roles.
- Re-create or migrate roles in another Manager.
- Promote one or several roles to other environments.

To export one or several roles:

1. Select **Security > Roles**.
2. Click **Export Archive** to export one role or click **Export** to export all roles as shown below.



Manager exports the role into a group archive ZIP file, which will be downloaded to your PC. The ZIP file contains role-config.xml file with all roles' definitions and permissions.

> To avoid any lockout issues, the Administrator role is not included in the export.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to import roles.

Import Roles

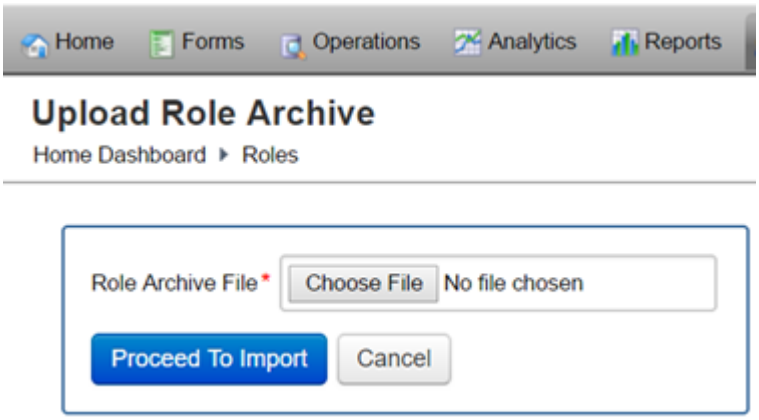
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to import roles as a ZIP file. You can use the import functionality to:

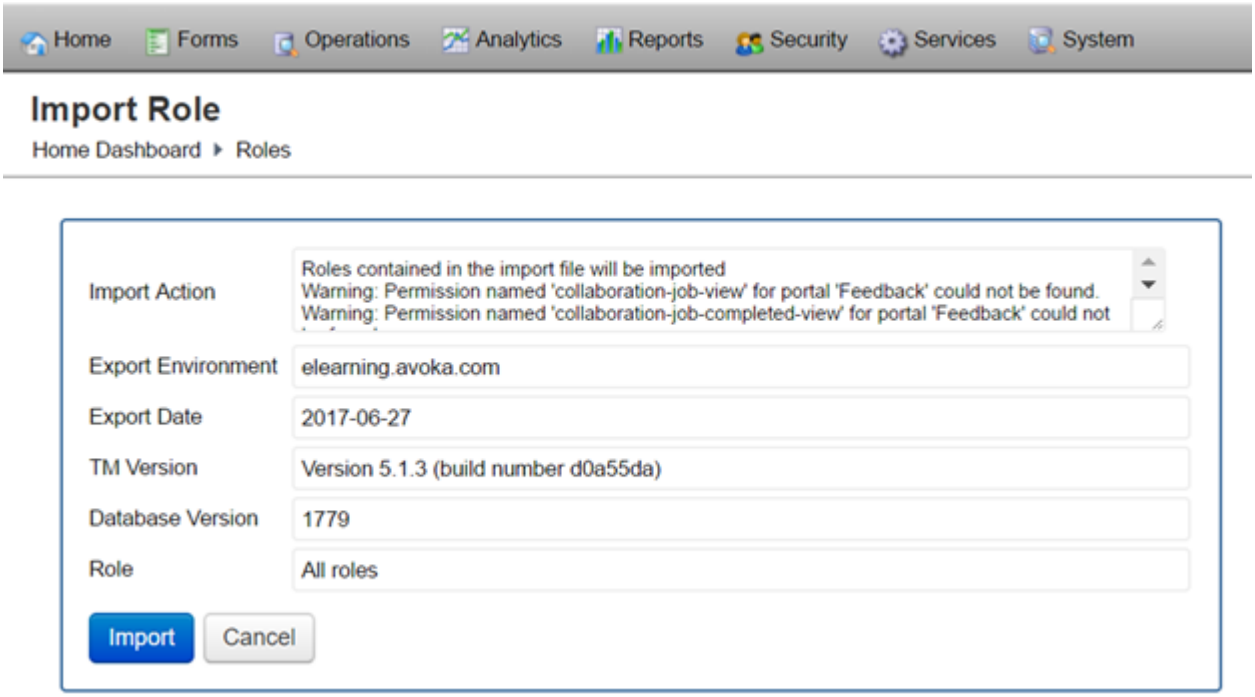
- Roll back one or several roles.
- Re-create one or several roles in another Manager.
- Promote one or several roles to other environments.

To import one or several roles:

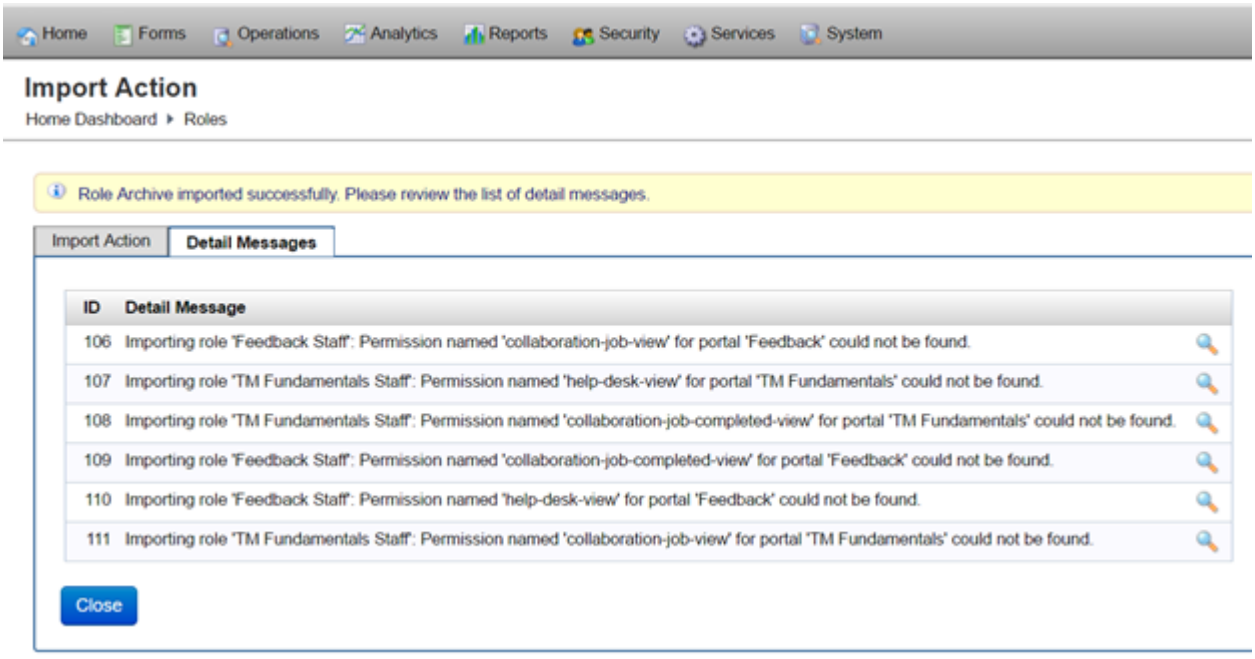
1. Select **Security > Roles** and click **Import**.



2. Click **Choose File** and locate a role archive ZIP file, which has been previously exported.
3. Click **Proceed To import**.
4. Review the **Import Options**.



5. Click **Import** to complete the process. The expected result page is shown below.



6. Press **Close** and check the list of roles to verify the imported roles have been added.

Next, learn how to migrate roles.

Migrate Roles

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides a mechanism to migrate roles from one environment to another. This allows you to develop and test roles in different environments before migrating them over to the production environment. It also saves time to set up the roles in new Manager environments.

> Role migration doesn't include user accounts and user data associated with the role.

> It is recommended that only global administrators migrate roles, otherwise only organization assignable roles are included.

To migrate one or several roles:

1. Export roles.
2. Import roles to another Manager environment.

> You can use the same process to perform a role rollback.

Next, learn how to assign permissions to a role.

Groups Overview

Journey Manager (JM) | System Manager / DevOps | 17.10

Manager allows you to create and use groups to implement a fine-grained control of user access to your environment. You can use this functionality to:

- Control user access to forms and their resources.
- Add users to groups to enforce required user functions and responsibilities.
- Extend the privileges defined through roles and their permissions.
- Create working groups to allow users to collaborate to review, approve or reject application forms. For example, a reviewer who approves or rejects an application form must belong to a reviewer form group.

Manager uses the following pre-defined group types to create new groups:

- Alert groups - to notify and alert users about certain events.
- Form groups - to restrict user access to forms or tasks that are associated with this group.
- Job groups - to restrict access to job coordinators.

Manager can use dynamically loaded groups when users are managed externally using **LDAP**[1] or **SSO**[2] authentication. Manager loads the user's group associations at login time. For LDAP users, groups are usually defined as a distinguished name, such as **"CN=mygroup,OU=Distribution Groups,OU=MyBusiness,DC=mycompany"**. Manager parses these group name, so **"CN=mygroup"becomes "mygroup"**. For SSO users, the group names can be passed in while creating a SSO token.

Next, learn how to view groups.

[1]Lightweight Directory Access Protocol (LDAP) s an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol network.
[2]Single sign-on (SSO) is a property of access control of multiple related, yet independent, software systems. With this property, a user logs in with a single ID and password to gain access to any of several related systems.

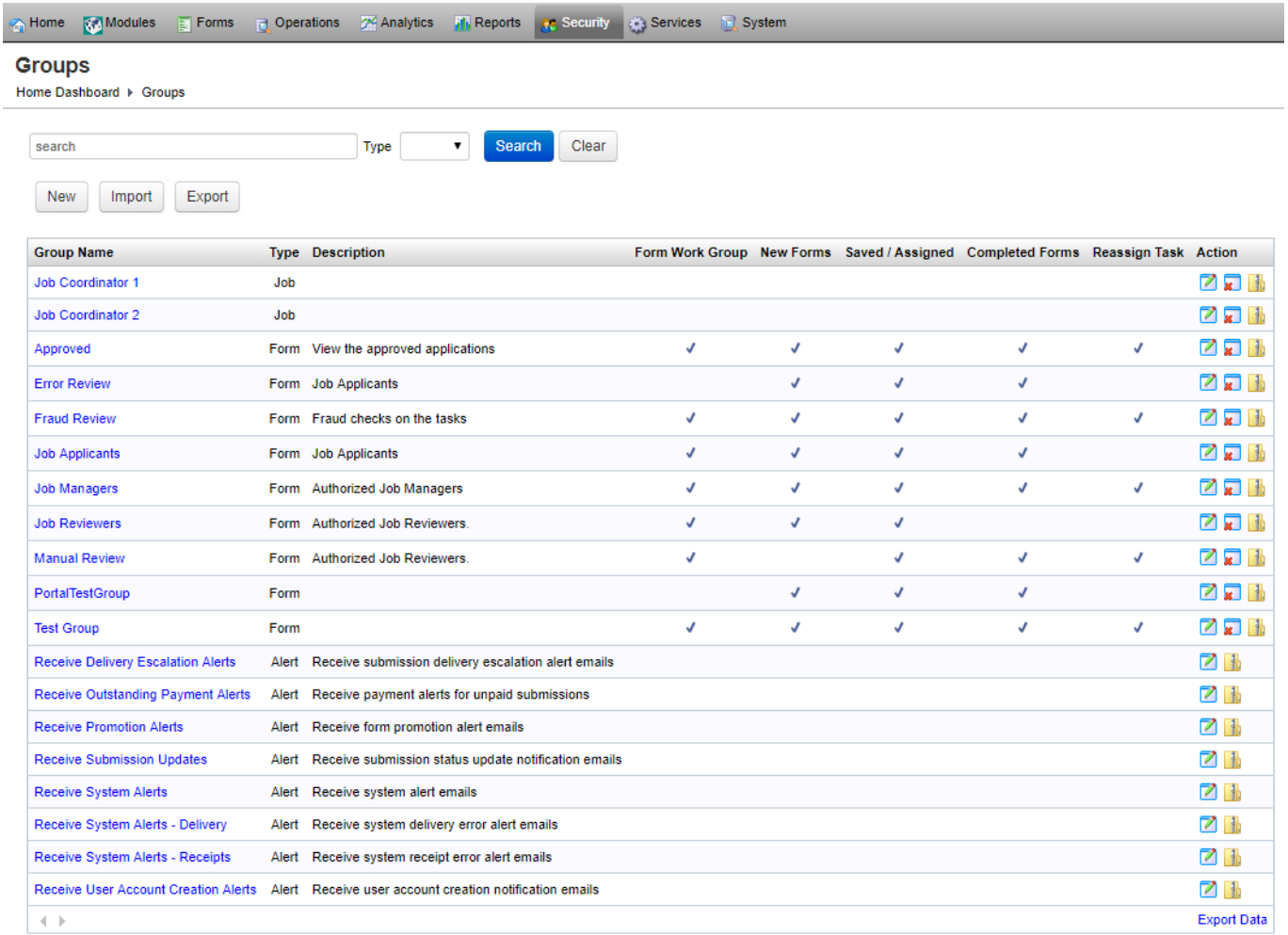
View Groups

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows user administrators to view the list of available groups.

To view all configured groups, select **Security > Groups**.

Groups are shown page by page, with the maximum number of records per page configured in the users preferences.



To filter or search the groups by one or more criteria, specify the following settings and click **Search**:

- Group's name or part of a name.
- Type of a group.

  A group type can be one of the following:

- Alert - notifies and alert users about certain events. A user must be assigned to a group of this group type.
- Form - restricts user access to forms or tasks that are associated with this group.
- Job - restrict access to job coordinators when overseeing jobs that they are responsible for.

Click **Clear** to reset the search criteria.

You can also perform the following actions:

- Create and edit a group
- Remove a group
- Export groups
- Import groups

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to assign groups to a user.

Alert Groups

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager uses the following predefined alert groups to notify relevant users, such as IT operations, of various important events and errors, which require their attention. Users are notified by emails, so they receive an email message for each event from each subscribed alert group. This might generate a significant number of emails if users are assigned to many alert groups. In this case, the users should review the assigned groups and revoke some of them.

Receive Delivery Escalation Alerts

The submissions with secure email delivery have not been acknowledged after a certain period of time. Scheduled Job = Delivery Escalation, Context = Transaction Organization Users and Global Users. Users who receive alerts when a transaction delivered by secure email has not been acknowledged yet.

Receive Outstanding Payment Alerts

There is a submission with an outstanding payment for their organization. This group is relevant only if administrators want to monitor submissions that are abandoned at the payment stage and contact an applicant to provide support. Event = Scheduled Job = Payment Reminder, Context = Users in group and their scope including global users. Event = Fraud and Duplicate Payment Detection, Context = Transaction. Users who receive email alerts reporting transactions that appear to have been abandoned at the payment stage.

Receive Promotion Alerts

Ends alert emails when a form is promoted. Event = Form promoted, Context = Form Organization Users and Global Users. User to notify when a form belonging to their organization is promoted.

Receive System Alerts

The system monitoring jobs find what important system activities are impacted. Event = Internal Exception Raised, Context = Global. Users who will be notified when the System Monitoring job finds that important system jobs are impacted.

Receive Submission Updates

A submission processing status has been updated. This group is relevant only to form users, but not administrators. Event = WebService Call = FormsServerWebService::UpdateProcessingStatus, Context = Transaction User. Users who will be notified when the processing status for a submission they have made has been updated. This group is relevant only to space users, not administrators.

Receive System Alerts - Delivery

Submission delivery fails. Event = Exception during delivery, Context = Transaction Organization Users but NOT Global User who are not part of the Organization. Users who will be notified when submission delivery fails.

Receive System Alerts - Receipts

Submission receipt generation fails. Event = Exception during receipt generation, Context = Transaction Organization Users but NOT Global User who are not part of the Organization. Users who will be notified when submission receipt generation fails.

Receive User Account Creation Alerts

A user has self-registered, creating a user account on a form space. Event = Account Creation, Context = All Users in Group but NOT Organization Specific. Users who will be notified when a user has applied for an account on a user space.

Custom Alerts

If you don't find a predefined alert that satisfies your needs, you can always create a custom one.

Alternatively, you can use system event publisher to send external notification events by publishing them to Azure or AWS SQS queues.For example, you can configure system event publisher to publish a notification each time there is a problem with the address component in a form and the problem is being recorded in the error log. In this case, you need to make sure that the error you want to log is one that is configurable in system event publisher.

> You can't change or delete the predefined alerts groups.

Next, learn about form groups.

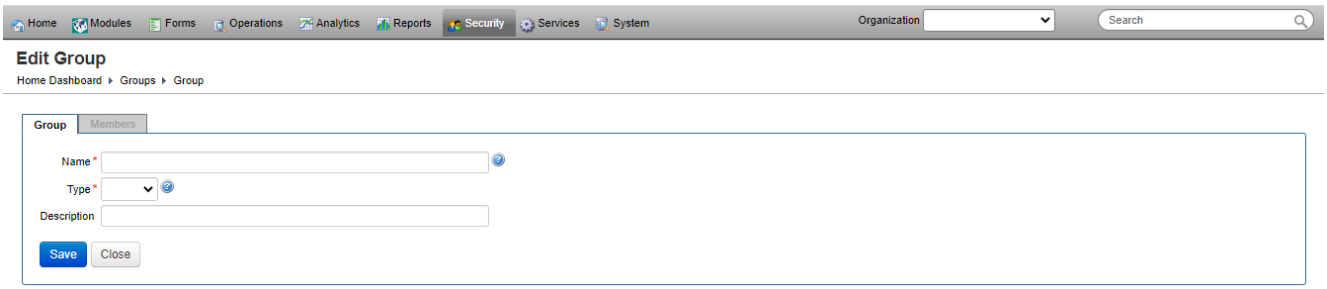
Create a Group

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

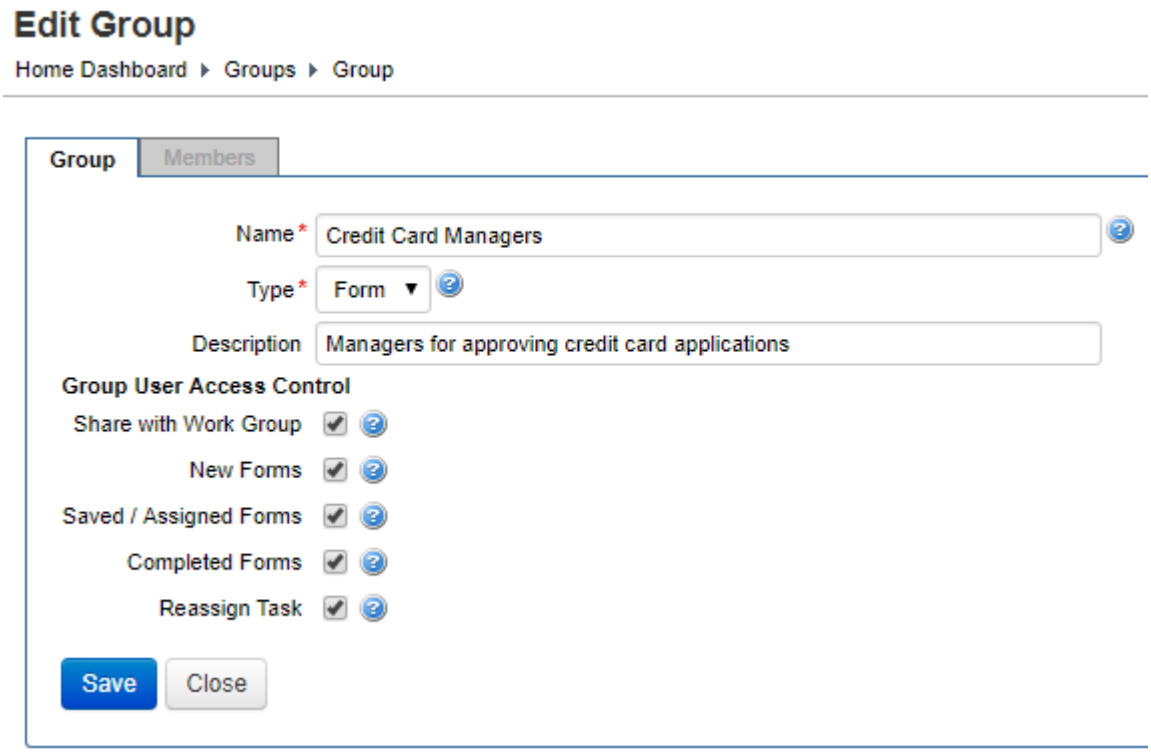
Manager allows you to create a group to a set of users with certain privileges, such as form or job access.

To create a group:

1. Select **Security > Groups**.
2. Click **New**.



3. Enter a new group name in the **Name** field.
4. Select a group type from the **Type** dropdown list, which can be:
   - Alert
   - Form
   - Job - Job groups are automatically created by the Job Controller definition of a collaboration job., so you don't need to add them manually. However, you need to wait until each Job group is automatically generated before assigning members to the group.
5. Enter a group description in the **Description** field.
6. Complete the additional **Group User Access Control** section, which is shown for the **Form** group type only.



7. Select **Share with Work Group** to allow users of this work group to share a form.
8. Select **New Forms** to allow users to actively start filling in forms belonging to the group. Otherwise, they can't access the forms at all, except when tasks are assigned to the group and the group is configured to share tasks and incomplete submissions.
9. Select **Saved / Assigned Forms** to allow sharing of tasks assigned to the group as well as incomplete submissions for these forms.
10. Select **Completed Forms** to allow sharing of the list of completed submissions and access to receipts.
11. Select **Reassign Task** to allow members to reassign task submissions to other users within the form work group.
12. Click **Save** to update the changes.
13. Assign user accounts to this group.
14. Configure form access in the **Group Access** tab for each restricted form.
15. If the new group is to limit user access to a new form, make sure only the **Authenticated Access** checkbox is selected in a form space where the form is hosted. For more information, see Assign Form Spaces to a Form.

In the case of a Form group, it is now possible to restrict individual forms to be accessed only by members of this Form group. By default, a form is publicly available to all users logged in to a Form Space. Once a form is restricted to at least one Form group, then it is only listed for the members belonging to the Form groups in the **Forms** tab. For more information, see Assign Form Spaces to a Form.

Any forms restricted to this Form group are now listed on the **Forms** tab from the **Edit Group** page.

Next, learn how to view groups.

Assign Users to a Group

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

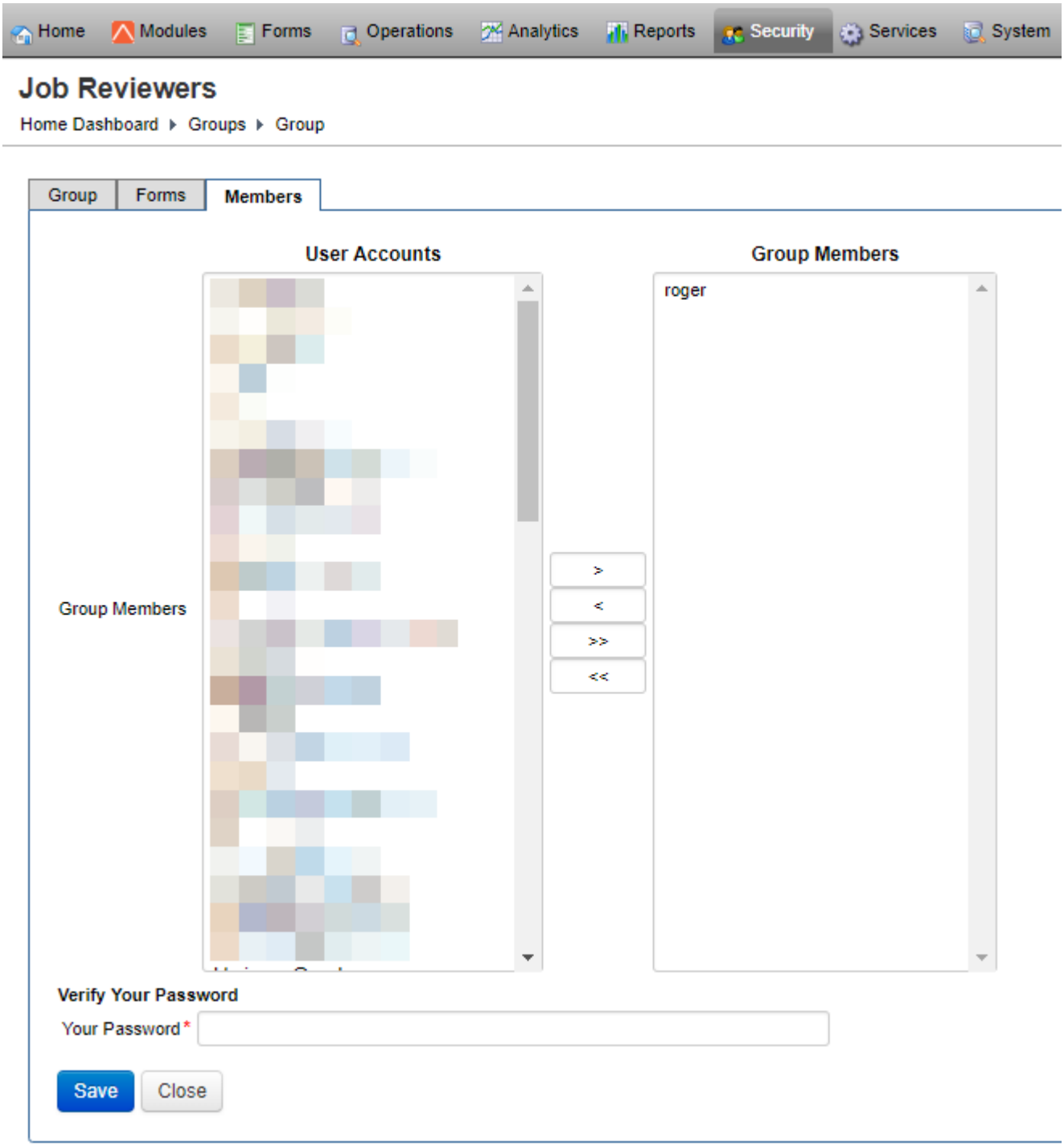
Manager allows you to assign users to a group and manager this association. You can use this functionality to:

- Assign several user accounts to a newly created group.
- Change an existing group's membership after you added or removed some user accounts.

To assign user accounts to a group:

1. Select **Security > Groups**.
2. Locate a group and click **Edit**.
3. Select the **Members** tab to see a list of the assigned user accounts in **Group Members**.

> This list is empty for a newly created group.



4. Select one or more user accounts from **User Accounts** and click **>** to assign them to this group.

> You can use the `Shift` and `Ctrl` keys to select multiple user accounts on Windows PC.

5. To revoke assigned user accounts, select the user accounts from **Group Members** and then click **<**.
6. Click **>>** to assign all user accounts.
7. Click **<<** to revoke all user accounts.

8. Provide your password in the **Your Password** field, for security reasons.
9. Click **Save** to update the changes.

> You can also manager a user group membership from the user account.

> If this is a form group, check which forms this group has been given privileged access to.

> You can also check any form's group access.

Next, learn how to assign groups to a user.

Assign Groups to a User

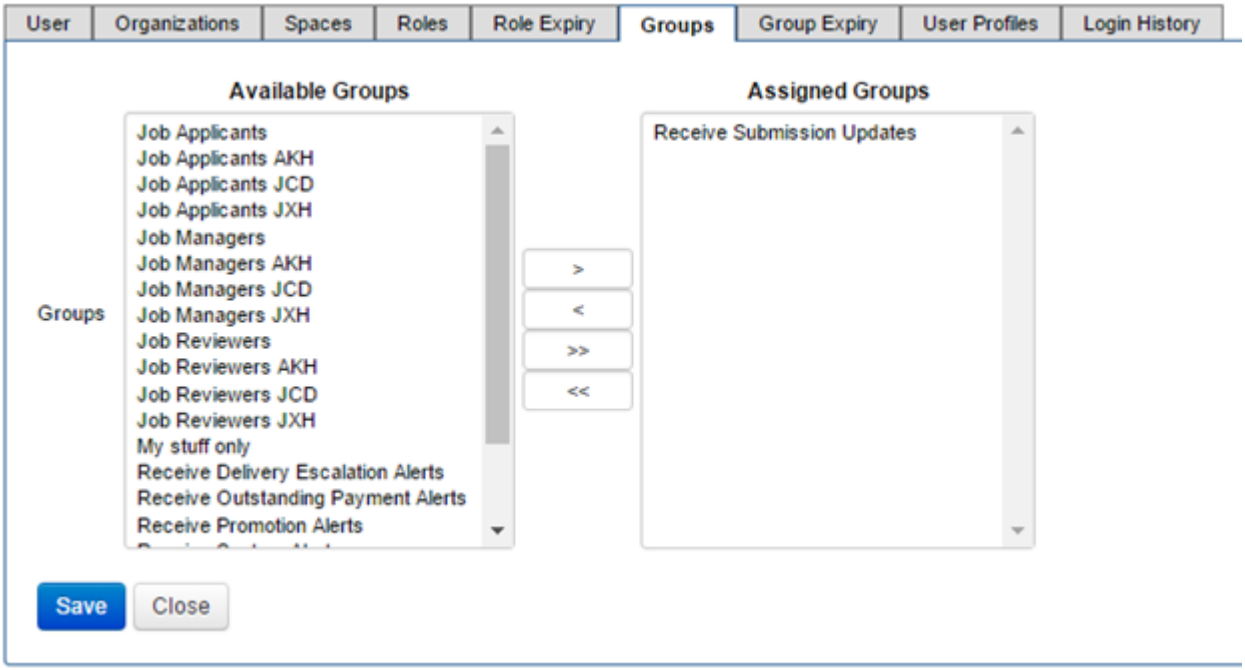
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to assign groups to users. You can use this functionality to:

- Assign several user accounts to a newly created group.
- Change an existing group's membership after you added or removed some user accounts.

To assign groups to a user account:

1. Select **Security > User Accounts**.
2. Locate a user account and click **Edit**.
3. Select the **Groups** tab to see the assigned groups for this user account in **Assigned Groups**.



4. Select one or more groups from **Available Groups** and click **>** to move them to **Assigned Groups**.

> You can use the `Shift` and `Ctrl` keys to select multiple groups on Windows PC.

5. To revoke any assigned groups, select the groups from **Assigned** Groups and click **<**.
6. To assign all groups, click **>>**.
7. To revoke all groups, click **<<**.
8. Provide your password in the **Your Password** field, for security reasons.
9. Click **Save** to update the changes.

You can also assign users to a group.

Next, learn how to configure group expiry.

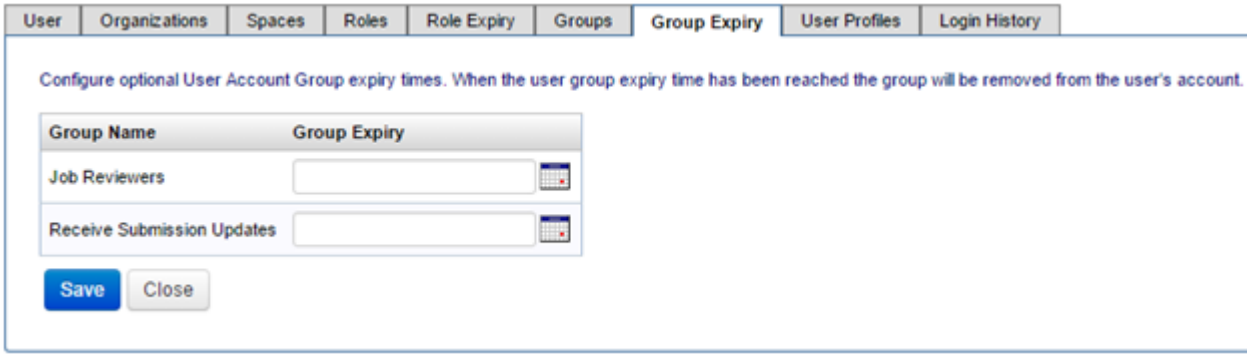
Configure Group Expiry for a User Account

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

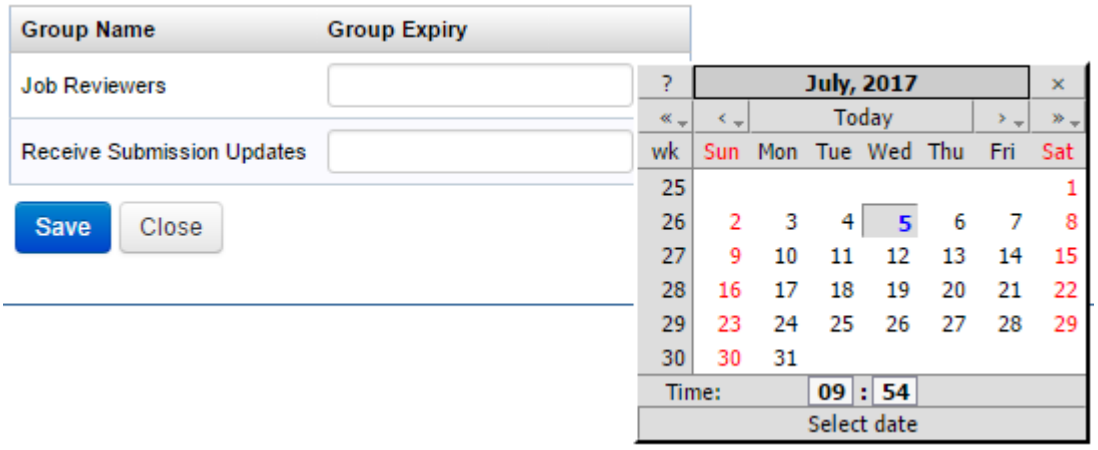
Manager allows you to enforce an expiry time to a user's group. You can use this functionality to temporarily assign a group to a user account to allow a support team to assist you in debugging a problem.

To enforce a time limit on a user's group:

1. Select **Security > User Accounts**.
2. Locate a user account and click **Edit**.
3. Select the **Group Expiry** tab to see groups that the user is assigned to in the **Group Name** column. Any temporary group assignment show an expiry date in the **Group Expiry** column.



4. Click **Calendar** in the **Group Expiry** column to make a user's group membership temporary.



5. Select the time and date when the user's group membership should expire. Select the time first, as the pop up automatically closes on the selection of a date.

> To make a user's group membership permanent again, remove the expiry date and time from the **Group Expiry** column.

6. Click **Save** to update the changes. When the expiry time comes, the Security Policy Manager background job removes the user's group membership from their user account.

Next, learn how to view groups.

View Forms Assigned to a Group

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

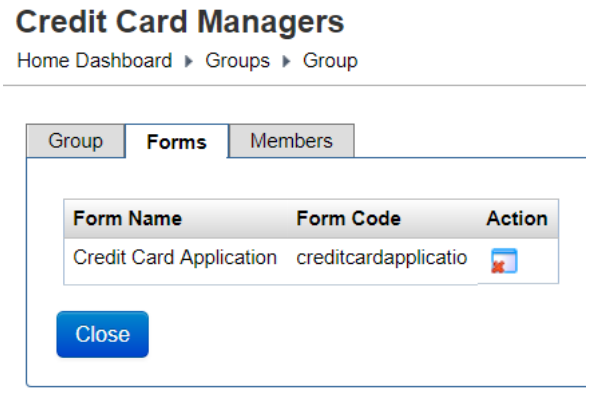
A newly created group doesn't have any forms assigned to it, so you have to do it manually. This restricts access to a form and ensures that only users assigned to the group will be able to access the form.

To assign forms to a group:

1. Create a new group.
2. Assign user accounts to the group.
3. Assign this group to a form. When a group is granted access to at least one form, it is listed in the **Forms** tab from the Edit Group page.

To view forms assigned to a group:

1. Select **Security > Groups**.
2. Locate a group and click **Edit**.
3. Select the **Forms** tab.



4. Click **Remove** to remove group access to this form.

Next, learn how to configure a group expiry.

Migrate Groups

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides a mechanism to migrate groups from one environment to another. This mechanism allows you to develop and test groups in different environments before migrating them over to the production environment. It also saves time to set up the groups in new Manager environments.

When a group is migrated, the following group properties are included:

- Group name.
- Group description.
- Group's user control.

> Group migration doesn't include user accounts and forms associated with the group.

To migrate one or several groups:

1. Export groups.
2. Import groups to another Manager environment.

> You can use the same process to perform a role rollback.

Next, learn how to assign groups to a user.

Export Groups

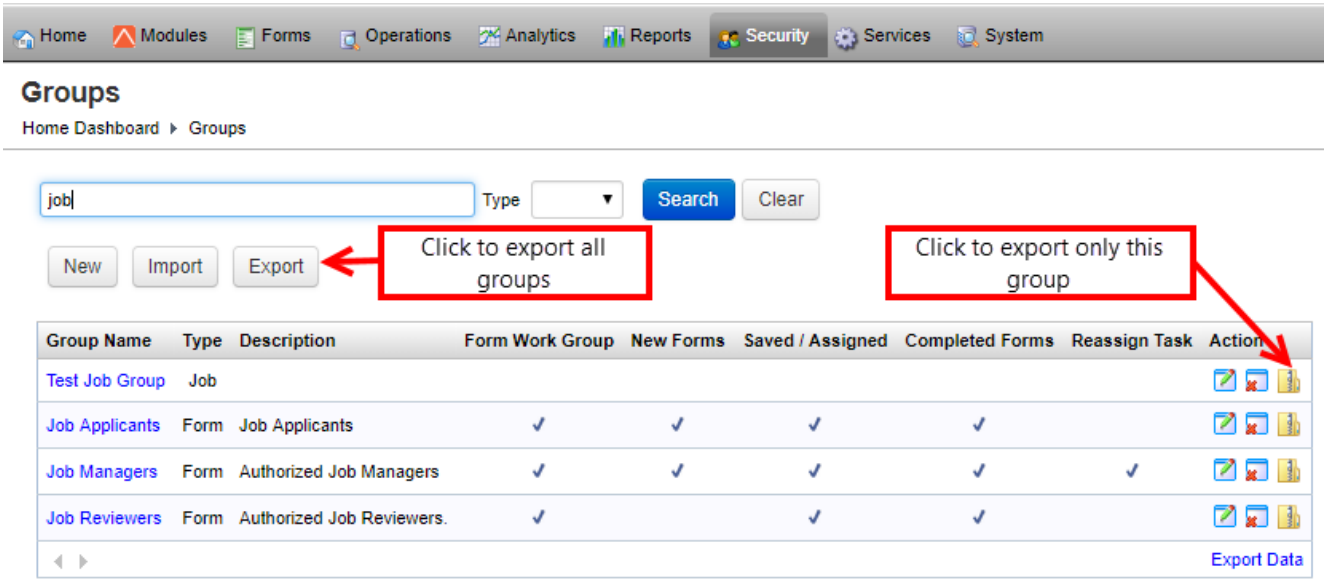
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows administrators to export groups from one environment and import them into another. You can use the export functionality to:

- Backup or archive groups.
- Re-create or migrate groups in another Manager.
- Promote one or several groups to other environments.

To export one or several groups:

1. Select **Security > Groups**.
2. Click **Export Archive** icon to export one group or click **Export** to export all groups as shown below.



Manager exports groups into a group archive ZIP file, which will be downloaded to your PC.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to import groups.

Import Groups

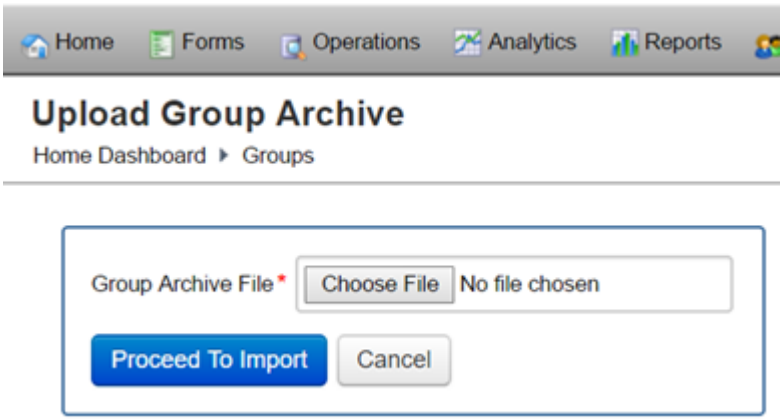
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to import groups as a ZIP file. You can use the import functionality to:

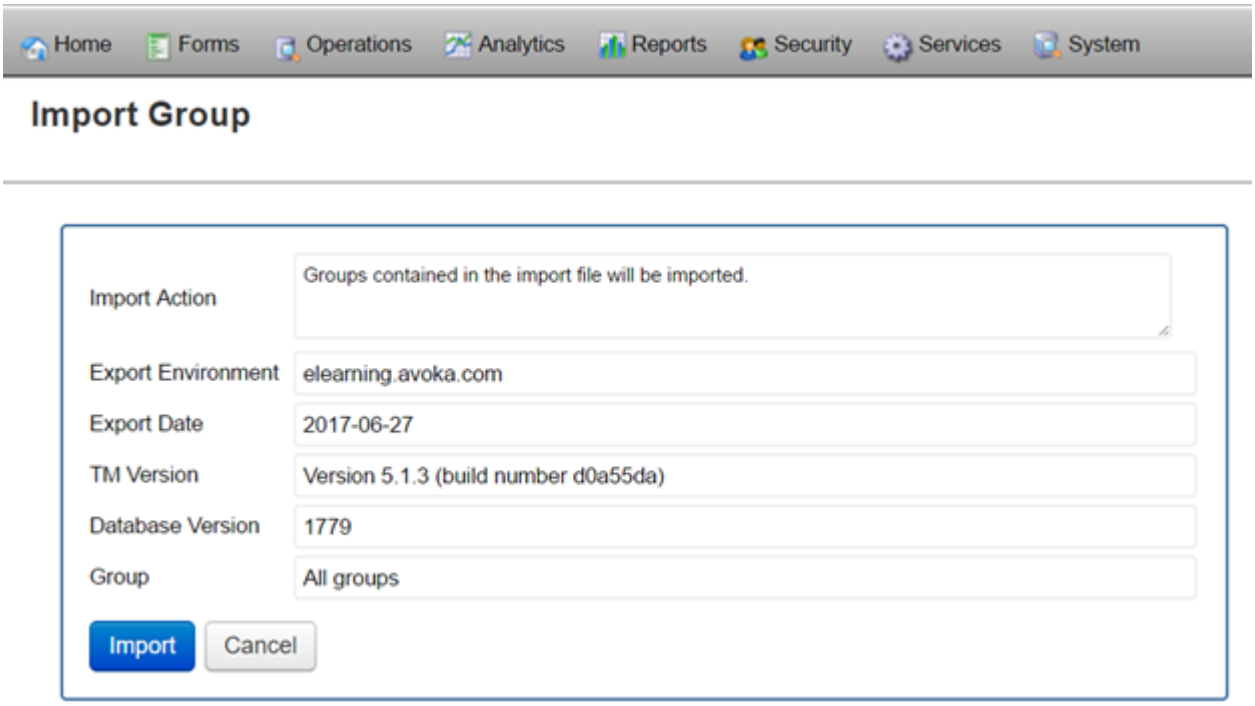
- Roll-back one or several groups.
- Re-create one or several groups in another Manager.
- Promote one or several groups to other environments.

To import one or several groups:

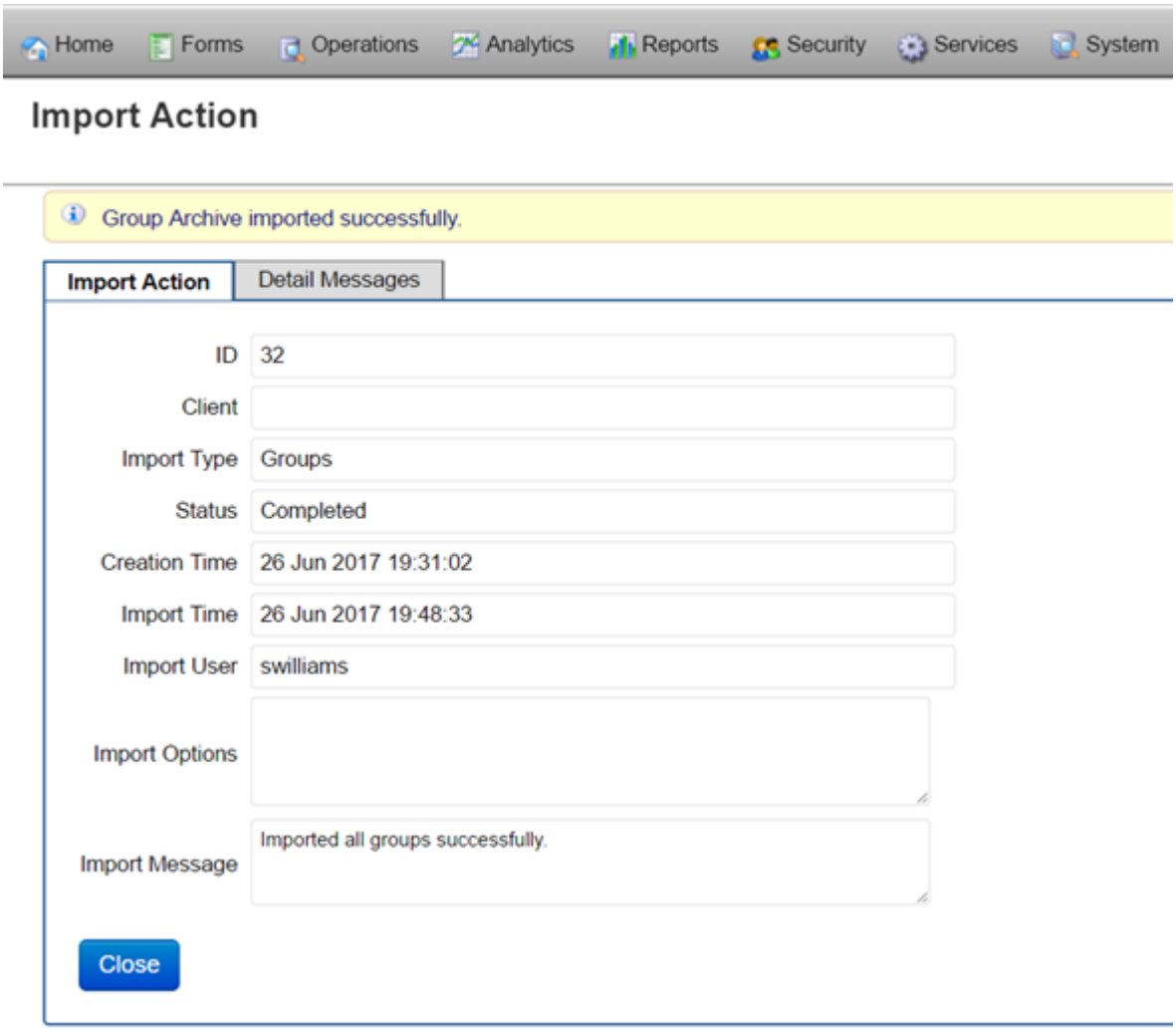
1. Select **Security > Groups** and click **Import**.



2. Click **Choose File** and locate a group archive ZIP file, which has been previously exported.
3. Click **Proceed To import**.
4. Review the **Import Options**.



5. Click **Import** to complete the process. The expected result page is shown below.



6. Click **Close** and check the list of groups to verify the imported groups have been added.

Next, learn how to migrate groups.

Permissions Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager comes with a broad range of permissions to control user access to restricted content, such as form spaces and modules. Permissions are specific to a form space. There is the standard set of permissions defined for any form space, but you can define your own permissions and implement security rules for each form space.

The Manager console has the following permissions that govern user access:

- View - gives users read-only access to the content.
- Edit - allows users to modify the content.
- Remove - allows users to delete the content.

By granting the edit or remove access, it doesn't automatically grant the view access. For each of edit and remove permission, you need to include the view access to the role. For example, to allow users to create new or modify existing organizations in Manager, their role must require the Organization View and Organization Edit permissions.

Manager has different sets of permissions for the following modules:

- Manager
- Maestro
- Insights
- Exchange
- Workspaces

Next, learn about standard roles.

Core Permissions

Journey Manager (JM) | System Manager / DevOps | 21.05

Manager provides the following permissions to control user access to restricted content. The permissions are mapped to the standard roles below, so you know what roles to assign to users to enable them to use certain product functionality.

Permission	Description	Role
Admin Directory	This is permission is required to access the Manager console. Users accessing the Manager console must be granted at least one role with this permission, otherwise they will not be able to use it.	System Manager Organizarion User Manager Transaction Data Access Operations System Support Administrator Developer
Audit View	Provides access to the system audit log and user login history.	System Manager System Support Administrator Developer
Collaboration Job View	Provides read-only access to collaboration jobs, including viewing details about a collaboration job and its steps, actions and events. This permission is also found in each of the form spaces. However, it would grant different types of privileges for a form space.	System Manager Operations System Support Administrator

Collaboration Job Edit	Allows users to manage collaboration jobs by pausing, triggering job processing, editing job properties and action details.	Developer System Manager Operations Administrator
Collaboration Job Remove	Allows users to delete collaboration jobs. Only canceled collaboration jobs can be deleted.	Developer System Manager Operations Administrator
Core Service Edit	Allows users to edit advanced service settings and some collaboration job related services. This permission is not assigned to any of the standard Manager roles, so use it with care and only when absolutely necessary. You should seek advice from the Manager support beforehand.	Developer System Manager Administrator
Data Retention View	Allows users to view global data retention settings.	Developer System Manager Operations System Support Administrator
Data Retention Edit	Allows users to modify and apply global data retention settings.	Developer System Manager Administrator
Db Version Update View	Allows users to view the database version history.	System Manager Administrator

Delivery Details View	Allows users to view the list of delivery channels and trigger delivery actions for submissions manually.	System Manager Operations System Support Administrator Developer
Delivery Details Edit	Allows users to create, edit and delete delivery channels.	System Manager Administrator Developer
Delivery Details Remove	It is not currently in use, so apply the Delivery Details Edit permission instead.	System Manager Administrator Developer
Deployment Property View	Allows users to view the list of system deployment properties and their values.	System Manager System Support Administrator Developer
Deployment Property Edit	Allows users to edit deployment property values, access and modify email settings.	System Manager Administrator Developer
Document Type View	Allows users to view document types and their details.	System Manager Operations

		System Support Administrator Developer System Manager
Document Type Edit	Allows users to create and edit document types.	Administrator Developer System Manager
Document Type Remove	Allows users to remove document types.	Administrator Developer System Manager
Email Delivery Acknowledge	It is related to delivery of type Email Secure, where administrators receive an email notifying them of new submissions, asking them to acknowledge delivery by accessing a page on Manager and confirming. This permission controls access to this page. Must be assigned to only administrators involved in processing secure email delivery.	Operations Administrator Developer System Manager
Email Queue View	Allows users to access the system email queue, but does not allow them to view each email details. You need to assign the Email Queue Edit permission for that.	Operations Administrator Developer System Manager
Email Queue Edit	Allows users to create, edit and access email queue details.	Operations Administrator Developer

Error Log View	Provides access to the error log and error details. It also enables links to error log entries from various Manager tabs.	System Manager Operations Administrator Developer
Event Log View	Provides access to the event log and event details. It also enables links to event log entries from various Manager tabs.	System Manager Operations Administrator Developer
Export Application Package	Allows users to export application packages, including form bundles and services, for organizations they have access to.	System Manager Operations Administrator Developer
Export Form	Allows users to export a form and all its versions.	
Export Form Version	Allows users to export a single form version.	System Manager Operations Administrator Developer
Export Groups	Allows users to export one or more groups.	System Manager Operations Administrator Developer

Export Metadata	Allows users to export the set of global metadata tags.	System Manager Operations Administrator Developer
Export Organization	Allows users to export an organization.	
Export Portal	Allows users to export a form space.	System Manager Operations Administrator Developer
Export Roles	Allows users to export one or more roles	System Manager Operations Administrator Developer
Export Services	Allows users to export one or more service definitions.	
Export Security Managers	Allows users to export one or more **security managers**[1].	System Manager Operations Administrator Developer

[1]Security managers are the Journey Manager's components that are responsible for User Authentication, User Authorization, User Self-Registration, and Session Management. Security mangers are designed to handle various sets of users requiring access to different form spaces and can be associated with one or more form spaces. Users can be external or internal, and they can require access to form spaces or a combination of form spaces and modules in different environments. You can find security managers under Security > Security Managers.

Form View	Allows access to the list of forms and a read-only view of most of form configuration. It also enables links to forms from various Manager tabs. If you need to view form version or form data configuration details, you need to add the Form Edit permission.	System Manager Operations Administrator Developer
Form Edit	Allows users to view all details about a form as well as create and edit forms. It includes editing form versions and form data configuration.	System Manager Operations Administrator Developer
Form Service Edit	Allows administrators to manage service definitions related to form development. This permission is a less powerful than the Service Edit permission.	System Manager Operations Administrator Developer
Form Test	This permission exposes links to access forms on a form space for testing. It also allows users to access the PDF receipt test harness, such as a link in the Form URLs section of the form edit page.	System Manager Operations Administrator Developer
Form Data Retention	Allows users to edit form specific data retention settings.	System Manager Operations Administrator Developer
Form Remove	Allows users to delete forms, form versions, form promotion entries and form data extract mappings.	System Manager Operations

		Administrator Developer System Manager
Groovy Console	Allows access to the Groovy console.	Operations Administrator Developer System Manager
Groovy Log View	Allows access to the Groovy service log.	Operations Administrator Developer System Manager
Group View	Allows users to view the list of groups, but it does not allow access to group detail information, so you should add the Group Edit permission for that.	Operations Administrator Developer System Manager
Group Edit	Allows users to create, edit and view group details.	Operations Administrator Developer System Manager
Group Remove	Allows users to remove groups.	Operations Administrator Developer

Group Assign	Allows users to assign users to a group. It is also necessary to access the Forms panel on the group edit page.	System Manager Operations Administrator Developer
Import Action View	Allows access to the import log. It does not include read-only access to import action details.	System Manager Operations Administrator Developer
Import Action Edit	Allows users to view and edit import log details . It is also necessary to import form and form space archives exported from another Manager server, along with the import permission corresponding to the entity to be imported.	System Manager Operations Administrator Developer
Import Action Remove	Allows users to delete entries from the import log. Only completed import actions can be deleted.	System Manager Operations Administrator Developer
Import Application Packagev	Allows users to import application packages, such as form bundles and services, for organizations they have access to.	System Manager Operations Administrator Developer
Import Composer Form	Allows users to import forms published by Composer and view the import details.	System Manager Operations

Import Form	Allows users to import form version archives created by exporting a form version from another Manager server. to import form version archives successfully, users also currently need the Import Action Edit permission.	Administrator Developer System Manager Operations
Import Groups	Allows users to import group archives created by exporting one or more groups from another Manager server.	Administrator Developer System Manager Operations
Import Metadata	Allows users to import metadata tag archives created by exporting one or more metadata tags from another Manager server. In addition, this permission is needed to upload files with values for metadata tags of types List and List Hierarchy.	Administrator Developer System Manager Operations
Import Organization	Allows users to import an organization archive, which contains organization configuration but no forms, exported from another Manager server. Only administrators with access to more than one organization can see the import button.	Administrator Developer System Manager Operations
Import Portal	Allows users to import a space archive, which contains form space configuration but not the backing WAR file, exported from another Manager server as well as space WAR files. To import space archives successfully, users also currently need the Import Action Edit permission.	Administrator Developer System Manager Operations

Import Roles	Allows users to import role archives, which contains roles and associated permissions, exported from another Manager server.	System Manager Operations Administrator Developer
Import Security Manager	Allows users to import security manager archives exported from another Manager server.	System Manager Operations Administrator Developer
Import Services	Allows users to import service archives, which contains service definitions and their service connections, exported from another Manager server.	System Manager Operations Administrator Developer
Import TPac	Allows users to import TPacs directly into Manager.	System Manager Operations Administrator Developer
Licensing Edit	Allows users to edit system licensing information and configure how licensing report data will be published to Avoka.	System Manager Operations Administrator Developer
Licensing Report View	Allows users to access the Transact Licensing and Job Licensing reports, otherwise these reports don't appear in the Reports menu.	System Manager Operations

Metadata Tag View	Allows users to view metadata tags and their details.	Administrator Developer System Manager Operations
Metadata Tag Edit	Allows users to create and edit metadata tags.	Administrator Developer System Manager Operations
Metadata Tag Removev	Allows users to delete metadata tags.	Administrator Developer System Manager Operations
Metadata Value View	Allows users to view values configured for metadata tags on the organization or form version levels.	Administrator Developer System Manager Operations
Metadata Value Edit	Allows users to create and edit values configured for metadata tags on the organization or form version levels.	Administrator Developer

Metadata Value Remove	Allows users to delete values configured for metadata tags on the organization or form version levels.	System Manager Operations Administrator Developer
Notification View	Allows users to view the list of notifications configured for form spaces.	System Manager Operations Administrator Developer
Notification Edit	Allows users to create and edit notifications configured for form spaces.	System Manager Operations Administrator Developer
Notification Remove	Allows users to delete notifications configured for form spaces.	System Manager Operations Administrator Developer
Offline Sync Log View	Allows users to view the T.Field related offline sync and client logs.	System Manager Operations Administrator Developer
Operation Chart View	Allows users to view a number of Manager analytics charts.	System Manager Operations

		Administrator Developer System Manager
Operation Export Report View	Not used.	Operations Administrator Developer System Manager
Operation Form Events View	Not used.	Operations Administrator Developer System Manager
Organization View	Allows users to view organization information and part of its configuration. It also enables links to organizations.	Operations Administrator Developer System Manager
Organization Edit	Allows users to create and edit organizations and associated configuration, such as categories and application packages.	Operations Administrator Developer System Manager
Organization Remove	Allows users to remove organizations. Because deleting organizations will also delete all associated forms and submissions, this is a permission that should be assigned sparingly.	Operations Administrator Developer

Organization Copy	Allows users to create a copy of an existing organization.	System Manager Operations Administrator Developer
Organization Portal Assign	Allows users to assign spaces to an organization for form hosting purposes.	System Manager Operations Administrator Developer
Organization Payment Edit	Allows users to view and modify an organization's payment gateway.	System Manager Operations Administrator Developer
Organization Property View	Allows users to view the set of organization property values.	System Manager Operations Administrator Developer
Organization Property Edit	Allows users to edit and delete organization property values.	System Manager Operations Administrator Developer
Organization Property Remove	Not used. The edit permission confers deletion privileges.	System Manager Operations

		Administrator Developer System Manager
Organization Storage Edit	Allows users to edit organization data storage and retention settings.	Operations Administrator Developer System Manager
Payment Log View	Allows users to view payment transactions.	Operations Administrator Developer System Manager
Portal View	Allows users to view details of form spaces and modules.	Operations Administrator Developer System Manager
Portal Edit	Allows users to edit form spaces, modules and all associated configurations.	Operations Administrator Developer System Manager
Portal Content Edit	Allows users to edit form space content, such as pages and resources.	Operations Administrator Developer

Portal Style Edit	Allows users to edit some form space content related to styling, such as the index page, childportal.css and all resources of JPG type.	System Manager Operations Administrator Developer
Portal Remove	Allows users to remove form spaces and modules.	System Manager Operations Administrator Developer
Promotion View	Allows users to view the form version promotions and a tab on the form edit page.	System Manager Operations Administrator Developer
Promotion 1 Development	Allows users to promote form versions to be in the Development status.	System Manager Operations Administrator Developer
Promotion 2 Ready For Test	Allows users to promote form versions to be in the Ready For Test status.	System Manager Operations Administrator Developer
Promotion 3 Test Failed	Allows users to promote form versions to be in the Test Failed status.	System Manager Operations

		Administrator Developer System Manager
Promotion 4 Test Passed	Allows users to promote form versions to be in the Test Passed status.	Operations Administrator Developer System Manager
Promotion 5 Ready For Production	Allows users to promote form versions to be in the Ready For Production status.	Operations Administrator Developer System Manager
Promotion 6 Not Ready For Production	Allows users to promote form versions to be in the Not Ready For Production status.	Operations Administrator Developer System Manager
Promotion 7 Production	Allows users to promote form versions to be in the Production status.	Operations Administrator Developer System Manager
Property Type View	Allows users to view property types.	Operations Administrator Developer

Property Type Edit	Allows users to create and edit property types	System Manager Operations Administrator Developer
Property Type Remove	Allows users to remove property types.	System Manager Operations Administrator Developer
Report View	Allows users to view reports, but it does not give read-only access to report configuration details, so you should add the Report Edit permission for that.	System Manager Operations Administrator Developer
Report Edit	Allows users to create and edit reports.	System Manager Operations Administrator Developer
Report Remove	Allows users to remove reports.	System Manager Operations Administrator Developer
Report Render	Allows users to render reports on screen and access the organization report schedules.	System Manager Operations

		Administrator Developer System Manager
Report Schedule View	Allows users to view report schedules on the organization level.	Operations Administrator Developer System Manager
Report Schedule Edit	Allows users to create and edit report schedules on the organization level.	Operations Administrator Developer System Manager
Report Schedule Remove	Allows users to remove report schedules on the organization level.	Operations Administrator Developer System Manager
Report Schedule Trigger	Allows users to trigger report schedules on the organization level that send an email for the most recent reporting period.	Operations Administrator Developer System Manager
REST Delivery Service API	Allows users to invoke the REST Delivery API provided in Manager.	Operations Administrator Developer

REST Fluent API	Allows users to invoke Fluent API via the REST service provided in Manager. | 21.05	Administrator System Manager
REST Service Definitions API	Allows users to invoke the REST Service Definitions API provided in Manager.	Operations Administrator Developer
REST Application Package API	Allows users to access the Application Package REST service endpoint for listing application packages, and importing and exporting application packages in Manager.	System Manager Operations Administrator Developer
REST Form Groups API	Allows users to access the Form Groups REST service endpoint for querying and creating Form Groups in Manager.	System Manager Operations Administrator Developer
REST Groovy Service Invoke	Allows users to access the Groovy Service REST service endpoint to invoke a Groovy service. This service endpoint can be used to perform operations such as uploading reference data for use in dynamic data or form prefill services. Enable the service parameter REST Invoke Enabled service parameter.	System Manager Operations Administrator Developer
REST Tasks API	Allows users to access the Tasks API REST service endpoint for creating and updating tasks in Manager in a secure and efficient manner.	System Manager Operations Administrator Developer

REST Test Center API	Allows users to access the TestCenter API REST service endpoint for performing test operations in Manager.	System Manager Operations Administrator Developer
REST Transaction History API	Allows users to access the Transaction History API REST service endpoint for querying the OLAP Transaction History table in a secure and efficient manner.	System Manager Operations Administrator Developer
REST TPac API	Allows users to access the TPac API REST service endpoint for deploying TPacs, and downloading or deleting TPac archive files in Manager.	System Manager Operations Administrator Developer
REST Transactions API	Allows users to invoke the REST Transactions API provided in Manager.	System Manager Operations Administrator Developer
Role View	Allows users to view roles, but it does not give read-only access to role configuration details, so you should add the Role Edit permission for that.	System Manager Operations Administrator Developer
Role Edit	Allows users to create, edit and view role details.	System Manager Operations

Role Remove	Allows users to remove roles.	Administrator Developer System Manager Operations
Role Assign	Allows users to assign user accounts to a role. The set of roles that a user can assign is further restricted based on whether the user is a global administrator.	Administrator Developer System Manager Operations
Security Manager View	Allows users to view the set of **security managers**[1].	Administrator Developer System Manager Operations
Security Manager Edit	Allows users to create, edit and view all details of security managers, including managing authentication providers. The Service Edit permission currently controls access to some content, such as creating security managers and authentication providers and parameters.	Administrator Developer System Manager Operations
Security Manager Remove	Allows users to remove security managers.	Administrator Developer System Manager

[1]Security managers are the Journey Manager's components that are responsible for User Authentication, User Authorization, User Self-Registration, and Session Management. Security mangers are designed to handle various sets of users requiring access to different form spaces and can be associated with one or more form spaces. Users can be external or internal, and they can require access to form spaces or a combination of form spaces and modules in different environments. You can find security managers under Security > Security Managers.

		Operations Administrator Developer System Manager
Service View	Allows users to view the full set of service definitions, service connections, collaboration job definitions, server nodes, and scheduled jobs. It doesn't give read-only access to the configuration details, so you should add the Service Edit permission for that.	Operations Administrator Developer System Manager
Service Edit	Allows users to create, edit and view configuration details of service definitions, service connections, collaboration job definitions, server nodes, scheduled jobs, and system data retention settings. For a form developer to modify their own form related services, without giving them access to all service types, you should use the Form Service Edit permission instead.	Operations Administrator Developer System Manager
Service Remove	Allows users to remove service definitions, service connections, collaboration job definitions, server nodes, and scheduled jobs.	Operations Administrator Developer System Manager
Submission View	Allows users to view form transactions, transactions history, transaction delivery, saved transactions, abandoned transactions. It also enables links to submissions on other pages. Access to the actual content submitted by users is not granted by this permission, so you should add the Submission Data View and Submission XML Data View permissions.	Operations Administrator Developer System Manager
Submission Edit	Allows users to modify submission statuses and retry delivery.	Operations Administrator

Submission Data View	Allows users to view non-XML submission content, such as receipt and attachment content, but not the submitted XML, submitted by the user. It also provides access to the set of submission data extract values, including the form submission data. Allows access to user entered data, so you must handle it carefully.	Developer System Manager Operations Administrator
Submission XML Data View	Allows users to view XML submission content, that is, the data entered into the form by the user. Allows access to user entered data, so you must handle it carefully.	Developer System Manager Operations Administrator
Submission XML Data Edit	Allows users to edit XML submission content and delete submission data extract values.	Developer System Manager Operations Administrator
Submission Save Edit	Allows administrators to open and fill in saved transactions.	Developer System Manager Operations Administrator
System Health View	Allows users to view system health.	Developer System Manager Operations Administrator

System Info View	Allows users to access some pages that provide information on system info, server performance, and LiveCycle performance.	System Manager Operations Administrator Developer
System Report Schedule View	Allows users to view report schedules for global reports.	System Manager Operations Administrator Developer
System Report Schedule Edit	Allows users to create and edit report schedules for global reports.	System Manager Operations Administrator Developer
System Report Schedule Remove	Allows users to remove report schedules for global reports, but you should also add the Report Schedule Edit permission for this to work.	System Manager Operations Administrator Developer
System Report Schedule Trigger	Allows users to trigger report schedules for global reports that sends an email for the most recent reporting period.	System Manager Operations Administrator Developer
Task View	Allows users to view the list of task submissions.	System Manager Operations

Task Edit	Allows users to create task submissions and edit related task.	Administrator Developer System Manager Operations
TPac Remove	Allows users to remove imported TPacs.	Administrator Developer System Manager Operations
TPac View	Allows users to view the imported TPacs.	Administrator Developer System Manager Operations
User View	Allows administrators to view user accounts, but it does not include a read-only view of user details, so you should add the User Edit permission for that.	Administrator Developer System Manager Operations
User Edit	Allows administrators to create, edit and view User Edit user properties.	Administrator Developer System Manager Operations

User Remove	Allows administrators to remove user accounts.	System Manager Operations Administrator Developer
User Bulk Import	Allows administrators to bulk import user accounts defined in an Excel spreadsheet.	System Manager Operations Administrator Developer
User Profile Edit	Allows administrators to create and edit user profiles, as well as mark a profile as the default one for a user.	System Manager Operations Administrator Developer
User Profile Remove	Allows administrators to remove user profiles.	System Manager Operations Administrator Developer
User Property View	Allows administrators to view user properties.	System Manager Operations Administrator Developer
User Property Edit	Allows administrators to create, edit and view user properties.	System Manager Operations

		Administrator Developer System Manager
User Property Remove	Allows administrators to remove user properties.	Operations Administrator Developer System Manager
Web Service Invoke	Allows users to invoke Manager's web services.	Operations Administrator Developer

Next, learn how to view roles.

Maestro Permissions

Journey Manager (JM) | System Manager / DevOps | 20.11

Maestro uses the following permissions to control user access to restricted content. The permissions are mapped to the standard roles below, so you know what roles to assign to users to enable them to use certain product functionality.

Permission	Description	Role
Form Edit	Allows users to edit a form.	Maestro Administrator Maestro Developer
Form Publish	Allows users to build a form.	Maestro Administrator Maestro Developer
Form Remove	Allows users to delete a form.	Maestro Administrator Maestro Developer
Form View	Allows users to view a form.	Maestro Administrator Maestro Developer
Library Edit	Allows users to edit a library.	Maestro Administrator Maestro Developer
Library Remove	Allows users to delete a library.	Maestro Administrator Maestro Developer
Library Share	Allows users to share libraries. | 20.11	Maestro Administrator Maestro Developer
Library Share Global	Allows users to share libraries globally. | 20.11	Maestro Administrator Maestro Developer
Library View	Allows users to view a library.	Maestro Administrator Maestro Developer
Organization Edit	Allows users to edit an organization.	Maestro Administrator Maestro Developer

Project Edit	Allows users to edit a project.	Maestro Administrator Maestro Developer
Project Export	Allows users to export a project.	Maestro Administrator Maestro Developer
Project Import	Allows users to import a project.	Maestro Administrator Maestro Developer
Project Remove	Allows users to delete a project.	Maestro Administrator Maestro Developer
Project View	Allows users to view a project.	Maestro Administrator Maestro Developer
Release Edit	Allows a Maestro Administrator to edit releases.	Maestro Administrator
Release Remove	Allows a Maestro Administrator to delete releases.	Maestro Administrator
Release Version View	Allows a Maestro administrator to view release versions.	Maestro Administrator
SCM Design Delete	Allows staff who can use Maestro SCM to delete designs.	Maestro SCM Design
SCM Design Edit	Allows staff who can use Maestro SCM to edit designs.	Maestro SCM Design
SCM Design View	Allows staff who can use Maestro SCM to view designs.	Maestro SCM Design
SCM Org Library Add	Allows staff who can use Maestro SCM to add organization libraries	Maestro SCM Org Library
SCM Org Library Edit	Allows staff who can use Maestro SCM to update organization libraries	Maestro SCM Org Library
SCM Org Library View	Allows staff who can use Maestro SCM to view organization libraries	Maestro SCM Org Library

SCM Project Delete	Allows staff who can use Maestro SCM to delete projects	Maestro SCM Project
SCM Project Edit	Allows staff who can use Maestro SCM to update projects	Maestro SCM Project
SCM Project View	Allows staff who can use Maestro SCM to view projects	Maestro SCM Project
SCM Project Library Add	Allows staff who can use Maestro SCM to delete project libraries	Maestro SCM Project
SCM Project Library Edit	Allows staff who can use Maestro SCM to update project libraries	Maestro SCM Project
SCM Project Library View	Allows staff who can use Maestro SCM to view project libraries	Maestro SCM Project
Shared Domain Model Edit	Allows staff to edit shared domain model.	Maestro Administrator
Versions Admin	Allows staff to manage versions.	Maestro Administrator

Next, learn how to view roles.

Exchange Permissions

Journey Manager (JM) | System Manager / DevOps | 18.05

Exchange uses the following permissions to control user access to restricted content. The permissions are mapped to the standard roles below, so you know what roles to assign to users to enable them to use certain product functionality.

Permission	Description	Role
TPac Install	Allows users to install and upgrade Exchange packages.	Avoka Exchange
TPac Review	Allows users to add a review for the Exchange packages.	Avoka Exchange
TPac Uninstall	Allows users to uninstall Exchange packages.	Avoka Exchange

Next, learn how to view roles.

Journey Analytics Permissions

Journey Manager (JM) | System Manager / DevOps | 18.05

Journey Analytics uses the following permissions to control user access to restricted content. The permissions are mapped to the standard roles below, so you know what roles to assign to users to enable them to use certain product functionality.

Permission	Description	Role
Insights Admin	Allows administrators to access the Journey Analytics module, so they are able to change its global settings.	Transact Insights Administrator
Insights View	Allows users to access the Journey Analytics module, so they are able to view its analytics reports.	Transact Insights

Next, learn how to view roles.

Workspaces Permissions

Journey Manager (JM) | System Manager / DevOps | 22.04

Journey Workspaces uses the following permissions to control user access to restricted content. The permissions are mapped to the standard roles below, so you know what roles to assign to users to enable them to use certain product functionality.

Permission	Description	Role
Collaboration Job Completed View	Allows the reviewers and managers to monitor and manage completed tasks in the Workspaces form space. Compare it with the Manager's Collaboration Job View permission.	Workspaces Staff
Collaboration Job View	Allows the reviewers and managers to monitor and manage tasks in the Workspaces form space. Allows the help desk staff members to monitor submissions from authenticated customers in the Help Desk tab, which, by default, excludes the submissions from authenticated users to be viewed by the Workspaces staff and Maguire staff roles. Assign this permission to standard roles to include submissions from authenticated users. This permission is required to call the following API: • **`txn/cancel`** to cancel a **Saved** or **Opened** form transaction, which is not a task submission. You don't need this permission to call **`txn/cancel`** API to cancel an **Opened** form transaction, which is a task submission and is assigned to a user. If you call **`txn/cancel`** API for Workspaces Operations Portal, you must have the **Help Desk View** permission configured instead.	Workspaces Staff
Help Desk Authenticated Edit		Workspaces Staff

Help Desk View	Allows the reviewers and managers to monitor and manage tasks in the Workspaces form space. It also enables you to see and use the **Help Desk** tab. The Workspaces Operations Portal requires this permission to call the following API: • **txn/status** | 22.04 • **txn/task** | 22.04 • **txn/purge** | 22.04 • **txn/cancel** | 22.04 You don't need this permission to call **txn/cancel** API for other portals. In this case, you must have the **Help Desk Authenticated Edit** permission configured.	Workspaces Staff
Operations All Transactions	Allows the reviewers and managers to view all transactions from all form spaces. The Workspaces Operations Portal requires this permission to call the following API: • **txn/query**, with the query scope as **ALL** in a request, to list all transactions for a form space. | 22.04 You don't need this permission to call **txn/query** API with other query scope options.	Workspaces Staff Workspaces API

Next, learn how to view roles.

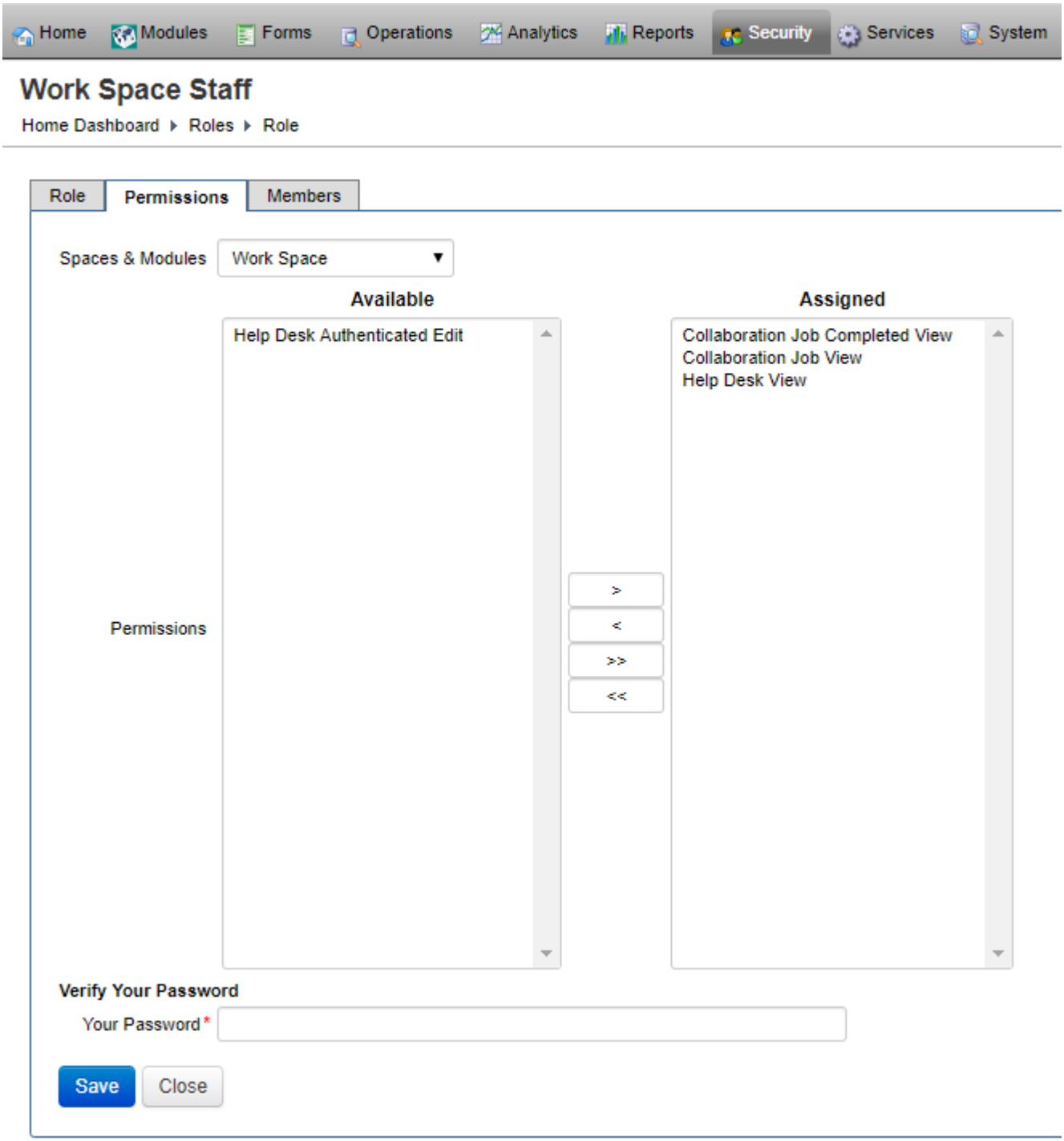
Assign Permissions to a Role

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows administrators to assign permissions to a role.

To assign permissions to a role:

1. Select **Security > Roles**.
2. Locate a role and click **Edit**.
3. Select the **Permissions** tab to add or remove permissions.



4. Select a form space of this role from the **Spaces & Modules** dropdown list. The set of permissions that are applicable to a form space are listed in the **Available** section, and if assigned to this role, the assigned permissions are listed in the **Assigned** section.

Not all spaces and modules are included in this list. Only the spaces where access is managed by roles and their associated permissions are applicable and those missing are not applicable to have permissions

assigned to roles.

5. Select one or more permissions from **Available** permissions and click **>** to move the selected permissions to **Assigned** permissions.

You can use the `Shift` and `Ctrl` keys to select multiple groups on Windows PC.

6. To revoke any assigned permissions, select the permissions from **Assigned** permissions and click **<**.
7. To assign all permissions, click **>>**.
8. To revoke all permissions, click **<<**.
9. Provide your password in the **Your Password** field, for security reasons.
10. Click **Save** to update the changes.
11. Check this form space is assigned to the user.

All permissions assigned to a role are loaded when a user logs into Manager, so the user has to log out and then log back in to see the changes to take effect.

Next, learn how manage role members.

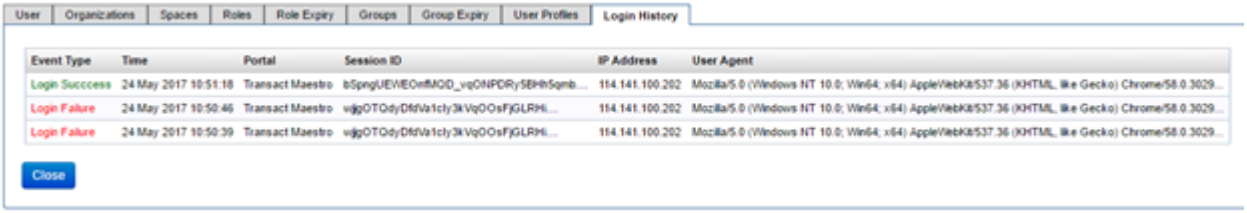
View User Login History

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view a list of all login attempts by each user or all users.

To view a user's login history:

1. Select **Security > User Accounts**.
2. Locate a user account and click **Edit**.
3. Select the **Login History** tab to see each login attempt and its details:



- Event Type is the status of the login attempt.

  A user login event type can be one of the following:

  - Account Locked Temp.
  - Login Failure.
  - Log Out.
  - Login Success.
  - Password Reset.
  - Session Expiry.
- Time is the date and time of the login attempt.
- Portal is the space or portal a user accessed or attempted to access.
- Session ID is a user's **session ID**[1] of the login attempt.
- IP Address is a user agent's **IP address**[2] used for this login attempt.
- User Agent is a **user agent**[3] used for this login attempt.

4. Click **Close**.

---

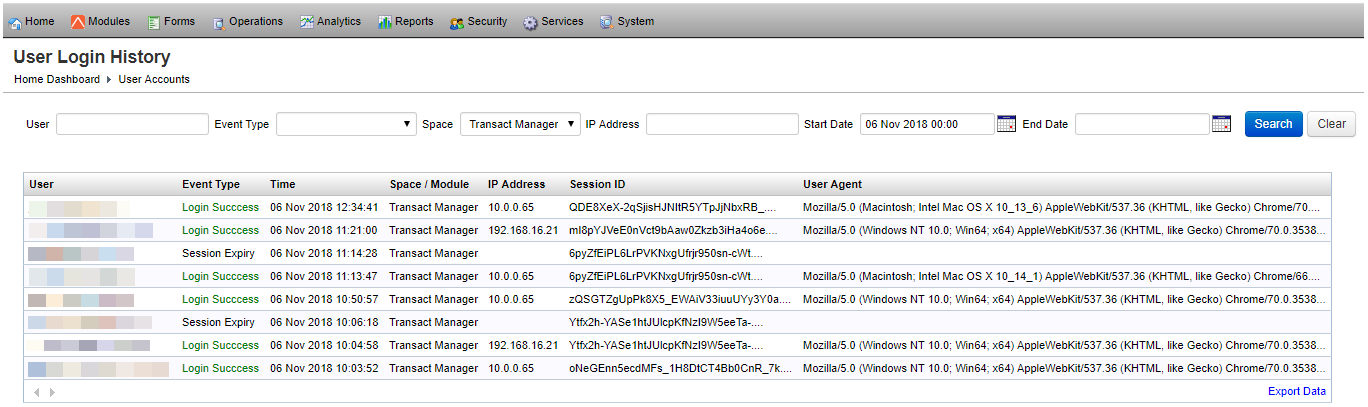
[1]Session ID is a unique number that a Web site's server assigns a specific user for the duration of that user's visit (session). The session ID can be stored as a cookie, form field, or URL (Uniform Resource Locator). Some Web servers generate session IDs by simply incrementing static numbers.

[2]IP address is a unique string of numbers separated by full stops that identifies each computer using the Internet Protocol to communicate over a network.

[3]User agent is software (a software agent) that is acting on behalf of a user. One common use of the term refers to a web browser that retrieves, renders and facilitates end user interaction with Web content.

To view all users' login history, select **Security > User Login History**.

User login history records are shown page by page, with the maximum number of records per page configured in the users preferences.



To filter or search the user login history by one or more criteria, specify the following settings and click **Search**:

- User account's name or part of a name.
- Event Type.

  A user login event type can be one of the following:

  - Account Locked Temp.
  - Login Failure.
  - Log Out.
  - Login Success.
  - Password Reset.
  - Session Expiry.
- Space.
- IP Address is a user agent's **IP address**[1] used for this login attempt.
- Start Date.
- End Date.

Click **Clear** to reset the search criteria.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to view user accounts.

---

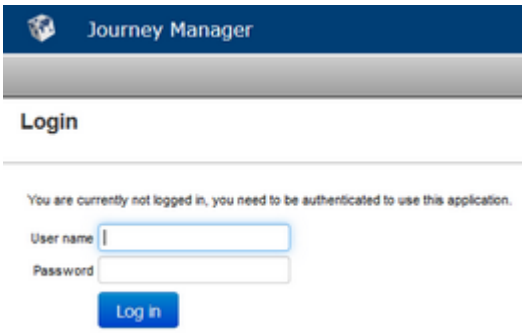
[1]IP address is a unique string of numbers separated by full stops that identifies each computer using the Internet Protocol to communicate over a network.

Log in to Manager

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager is a highly secure application that requires all users to sign in using their valid credentials, such as a user name and a password. You can also enable **Two-factor Authentication**[1] to harden the security of your applications, which is especially important in production environments.

When users want to log onto Manager, they must provide valid credentials on the login page.



When you create new users, you can also force them to change their system generated passwords at first login by selecting the **Change Password After Login** checkbox.

A user can be forced to change password due to various reasons, such as, overall system security hardening. Check your security manager configuration as it defines how the login and password change work.

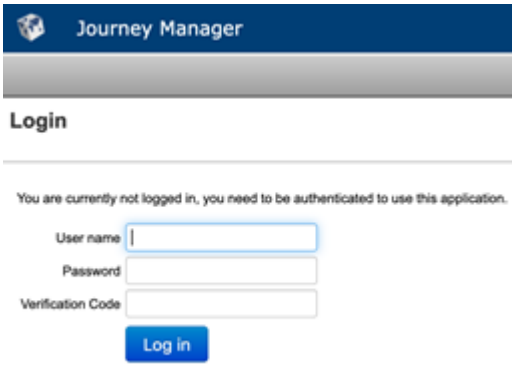
Two-factor Authentication

When required by business to improve application security, you can enable **2FA**[2] to force a user to provide an additional verification code during the login process, as shown below.

---

[1]Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.

[2]Two-factor authentication (2FA), sometimes referred to as two-step verification or dual factor authentication, is a security process in which the user provides two different authentication factors to verify themselves to better protect both the user's credentials and the resources the user can access. Two-factor authentication provides a higher level of assurance than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically a password or passcode. Two-factor authentication methods rely on users providing a password as well as a second factor, usually either a security token or a biometric factor like a fingerprint or facial scan.



To do so, you need to configure your local security manager to use Google Authenticator, which requires users to install Google authenticator app on their mobile devices for 2FA to work.

If required you can also specify a grace period - the date when 2FA will be enforced. Users can continue to sign in using their username and password during the grace period, but must configure 2FA in order to sign in after the grace period expires.

> You must have Administrator access and sufficient 2FA knowledge to configure a security manager. We recommend contacting our Customer Service team to configure 2FA for your particular needs.

Next, learn how to log out of Manager.

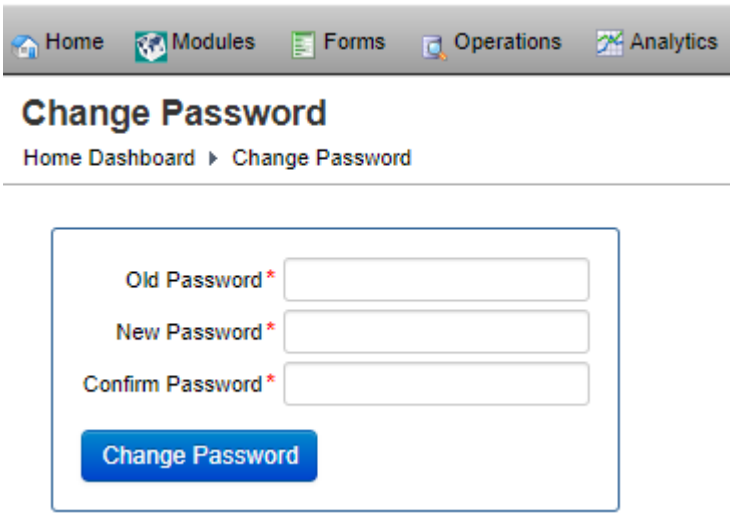
Change a Password

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Every now and then, you need to change your password to log onto Journey Manager, This is the case when you are given access to Manager for the first time, your password expired or is compromised.

To change a password:

1. Select **Security > Change Password**.



2. Enter your old password in the **Old Password** field.
3. Enter a new password in the **New Password** field.

   The standard Journey Manager password requirements are:

   - Must be at least 12 characters in length.
   - Must not exceed 20 characters | 5.1 & Higher or 40 characters | 20.05 .
   - Must contain at least one or two letters and one digit.
   - Must contain at least one non-alphanumeric character such as %, &.
   - Must contain mixed case, one upper case, and one lower case.
   - Must not be a password existing in the Illegal Password Values list.

   Password requirements strongly depend on a security manager configured in your environment.

4. Confirm the new password in the **Confirm Password** field.
5. Click **Change Password**.

Next, learn how to log out of Manager from the client.

Log out of Manager

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Explicitly logging out of Journey Manager is important in a few use cases. One example is using **Single Sign-On**[1] with a 3rd-party application. If the 3rd-party application user can logout, then login as a different user, the 3rd-party application user is now different than the Manager user.

Here is the actual example that drove the creation of this page: User A logs into Salesforce, gets an SSO account in Manager and passively logs in to Manager. User A then exits the Manager form and logs out of Salesforce. User B then logs in to Salesforce in the same browser. Because the User A, a login session is still running, User B's form is prefilled with User A's data.

The solution is to explicitly log out of Manager when the user exits the form. Here is a non-exhaustive list of possible places where this should be done:

- Cancel or Exit
- Form Submission
- The before unload or unload browser events.

To logout of Manager, make a **`GET`** request to the **`/logout`** path from the appropriate form space, which does the following:

- Logs the user out of Manager.
- Clears the Manager-related cookies.

A sample JavaScript code is shown below:

```
/* Logout of Transaction Manager. */
function logoutTMInstance() {
    console.debug("Start Avoka logout");
    var href = window.location.href;
    var arr = href.split("/");
    var url = arr[0] + '\/\/' + arr[2] + '\/' + arr[3] + '\/logout.htm';
    var xmlHttp = new XMLHttpRequest();
    /* Call service synchronously. Change 3rd parameter to "true" for an asynchronous
call. */
    xmlHttp.open( "GET", url, false);
    xmlHttp.send(null);
    console.debug("Avoka logout complete");
};
```

---

[1] Single sign-on (SSO) is a property of access control of multiple related, yet independent, software systems. With this property, a user logs in with a single ID and password to gain access to any of several related systems.

Next, learn how to log in to Manager.

Operations Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager allows you to manage and keep track of all transactions that have resulted in user form submissions. The support can use this functionality to track user submissions, search transactions, and thus troubleshoot production issues. They can also perform the following maintenance activities:

- Change a transaction status
- Abandon a transaction
- Configure form abandonment
- View form submission data
- View collaboration jobs

You can quickly check a form transaction and then drill down into its details, such as:

- Transaction status and update it as needed
- Form sessions of the transaction
- Transaction history
- Form XML data of the transaction
- Form Data Extract of a submitted form
- Attachments that are part of the transaction
- Groovy log messages related to the transaction
- Events related to the transaction
- Properties related to the transaction
- Transaction timeline

Manager also provides variable reporting capabilities so you can see how the system runs and performs.

Next, learn how to use transaction support search.

Transactions Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A transaction is an integral part of Journey Manager, which represents a user interaction with a form

Manager distinguishes between the following transaction types:

- Internal transactions associated with business users.
- External transactions associated with customers and applicants.

A transaction created as a result of a user submitting a form is referred to as a submission.

Transactions follow their life cycles. Submission life cycle is the most important one. It covers user navigation through a form and therefore very important for designing a form with ideal customer journey.

Next, learn how to view transaction licensing.

Submission Life Cycle

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager tracks user interaction with forms using submission objects, which are specialized transactions with a well-defined life cycle. A submission life cycle is the series of actions that a user makes while navigating through a form.

Manager creates a submission object when a user opens a form or a task submission is assigned to a user. As the user progresses through the form and performs form-specific actions, such as save, submit, or payment, Manager knows what the next action is in the submission life cycle, allowing you to build complex form workflows. Manager updates the submission object with user actions and performs additional tasks, such as virus scanning or delivery, and eventually purges it from the system.

Submission life cycle consists of two flows:

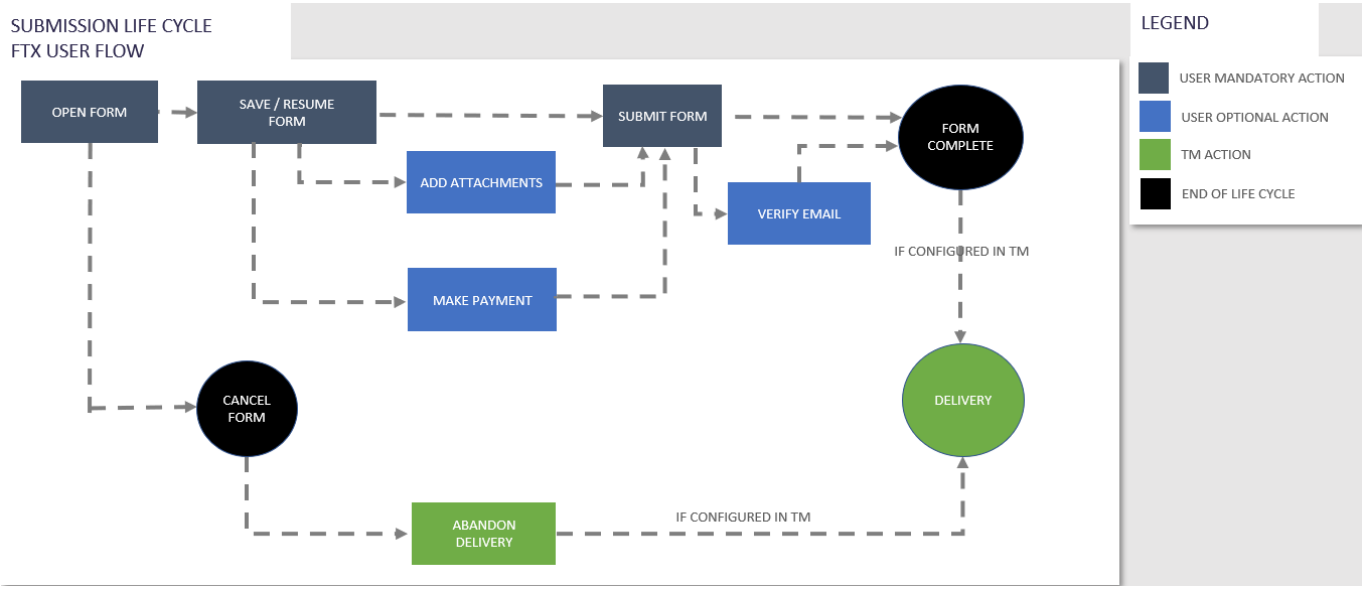
- A user flow, which defines actions a user performs to complete a form, so it can be delivered by Manager.
- A Manager back end flow, which defines actions Manager performs to deliver a form to a user.

Manager supports the following user flows:

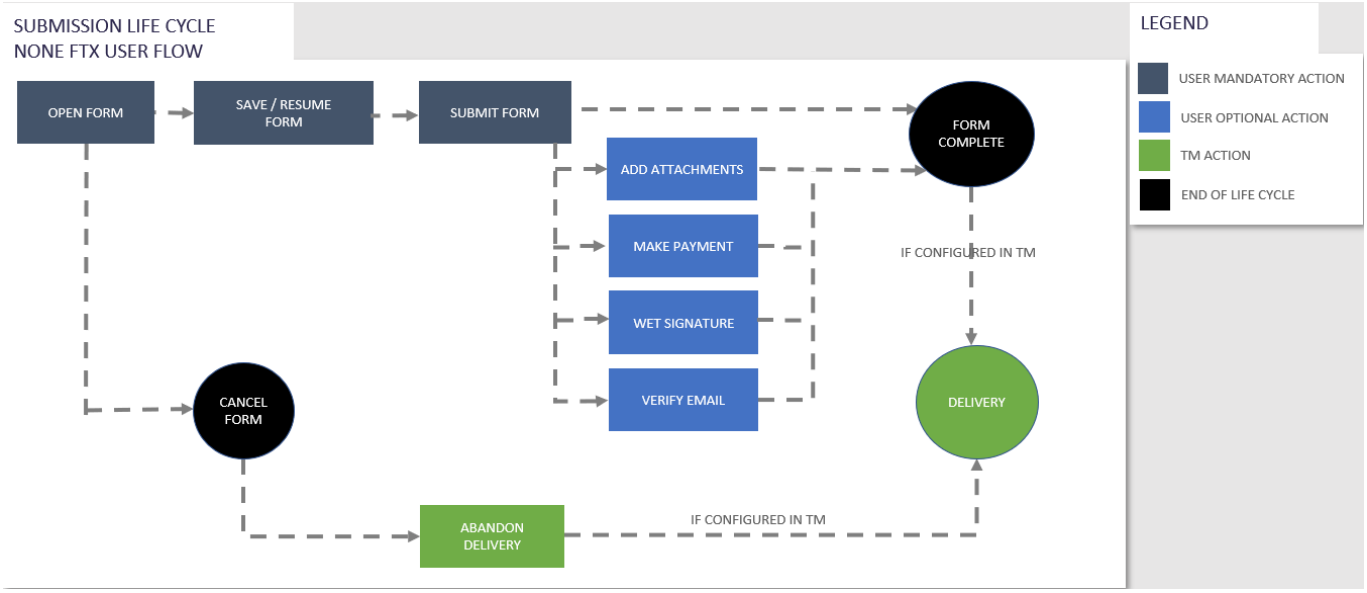
- Form Transaction Experiences (FTX) user flow when a transaction goes straight to Form Complete on form submission as all the attachments and payments are taken before the form is actually submitted. This Includes email verification.
- Traditional (none FTX) user flow when attachments, signatures and payment are captured after the user submits the form.

These flows are illustrated in the diagrams below.

User flow - Form Transaction Experiences (FTX)



User flow -Traditional (None FTX)



Delivery flow



The detailed description of submission life cycle actions are listed in the table below.

User Actions

Action	Description	Next Actions
Add attachments	A user adds one or several attachments as a part of the application form. Manager marks the submission form status as **Submitted** or **Completed**, and the attachment status as **Completed**.	Make payment Sign a form
Authenticate a form	An unauthenticated user submits a form that requires authentication. This is often the case after a form has been saved offline. Manager marks the submission form status as **Submitted** and marks the authentication status as **Completed** after the user has acknowledged the submission.	Add attachments Make payment Sign a form

		Verify email address
Cancel a form	A user cancels the form. Manager marks the submission form status as **Abandoned**.	Abandonment delivery
Make payment	A user completes the payment as a part of the application form. Manager marks the submission form status as **Submitted** or **Completed**, and the payment status as **Completed**.	Sign a form
Open a form	A user opens the form in a browser. Manager creates a submission object, adds relevant entries in the request and submission log, marks the submission form status as **Opened**, and renders the form to the user.	Save a form Save a form offline Submit a form
Reject electronically signed submission	A user rejects to sign the form. Manager marks the submission form status as **Submitted**, the signature status as **Cancelled**, and the signature type as **eSignature**. The submitted data may be delivered as part of abandonment delivery, if it's configured in Manager.	End of life cycle
Reject submission	A user rejects the form. Manager marks the submission form status as **Submitted** and the authentication status as **Cancelled**.	End of life cycle
Save a form offline	A user saves the form offline (locally on the computer). The submission record is not updated in Manager. It's relevant for PDF forms.	Submit a form
Save a form	A user saves the form. Manager marks the submission form status as **Saved**.	Save a form offline Submit a form
Sign a form	A user signs the form and adds it as an attachment. Manager marks the submission form status as **Submitted** or **Completed**, the signature status as **Completed**, and the signature type as **Wet**.	Delivery Receipt render Virus scan
Submit a form	A user submits the form. Manager marks the submission form status as **Submitted**, if the user need to do any further tasks, otherwise, it's marked as **Completed**.	Add attachments

		Authenticate a form Make payment Sign a form Verify email address
Verify email address	Manager sends an email to users asking them to click on a link to verify their email address. This action happens in parallel with other actions, such as adding attachments and payment. Manager marks the submission form status as **Submitted** and the email verification status as **Completed** after the user has verified the email address. Manager must have submission email verification enabled.	Add attachments Make payment Sign a form

These actions might not occur in every workflow, or they may occur in a different order. For example, attachments and payments can be completed in either order. This is subject to the form workflow implementation.

Manager Actions

Action	Description	Next Actions
Delivery	This is the final step. It occurs after the user has completed the submission and when all other Transaction Manager tasks have been completed successfully.	End of life cycle
Receipt render	This is done when a form is submitted, either when the user requests a receipt or as part of a routine job. If a form requires eSignatures, the generated receipt will contain the electronic signature of the organization hosting the form.	Virus scan
Virus scan	If a virus scanning service is configured, all submission attachments will be scanned for viruses and submissions will not be completed until the scans have finished.	Delivery

At any stage of the submission life cycle, an error can be caused by a user or Manager. For more information about an error, check the submission events for a particular transaction.

Next, learn how to view form transactions.

Submission Categories

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager categorizes transactions depending on which part of transaction life cycle they are currently in. There are three distinct submission categories:

1. Completed submissions.
2. Abandoned submissions.
3. Submissions as part of collaboration jobs.

You need to understand the differences between these categories to configure Manager's data retention policies, which are used by the Transaction Processing scheduled job to perform the following operations:

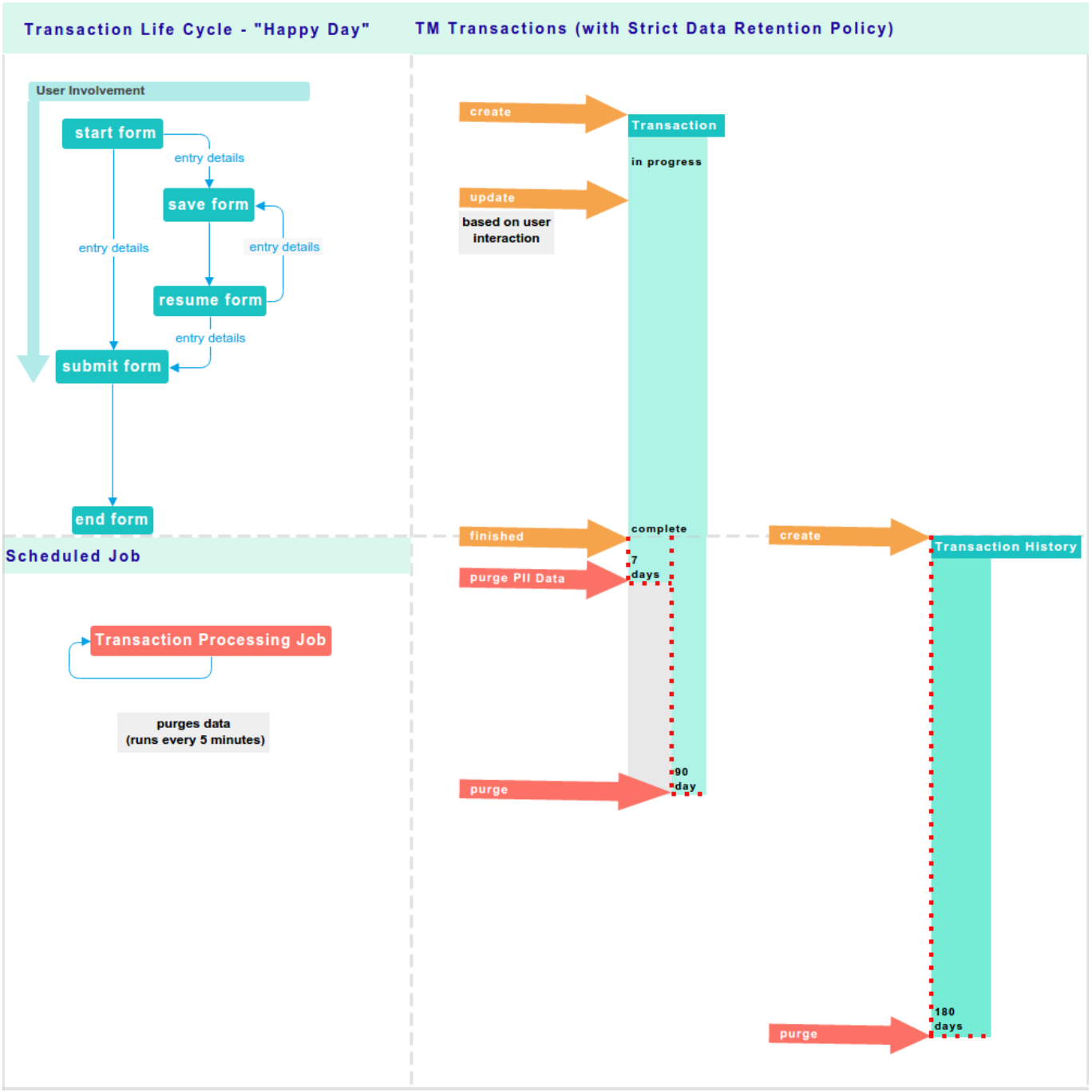
- Abandons any transactions that are ready to be abandoned.
- Purges any data and transactions that are ready to be purged.

Next, learn about submission life cycle.

Completed Submissions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager tracks all form submissions including their statuses. Each transaction goes through a life cycle. It's important to understand the transaction cycle so you know where each transaction is at and how to troubleshoot them. One of the most importance life cycles is the completed transaction cycle for a form submission. It is shown below.



This is a Happy Day scenario where the user completes and submits a form within the allowed time frame that is established by the saved transaction data retention setting, which is 30 days for strict policy.

The following steps describe this process:

1. The form is started - opened - by the user and a transaction record is created. At this point the user may:
   - Complete and submit the form in one sitting or;
   - Save the form and return to it, subsequently completing and submitting it within the allowable time-frame. Before submitting the form, the user can save and resume the form as many times as needed. Each time the user resumes and saves the form, the timeframe allowed, configured by the saved transaction data retention setting, begins from this save. That is, the user can return and submit the form prior to the 30 days (for strict policy) having elapsed from when it was last saved.
2. On submission, the form is processed and delivered to the backend system.
3. Once delivered, the transaction is considered a finished transaction and a transaction history record is created.
4. From this point onwards, the transaction record and the transaction history record will be retained in Manager according to the data retention settings.
5. The Finished Transaction PII Data setting is used to determine when to remove PII data from the finished transaction. PII data makes up most of a submission's data footprint. Removing the PII data for a brief time (7 days for strict policy) after a transaction has been completed, has the following benefits:
6. The Manager database is leaner. The purge will remove submitted XML and attachment data.
7. All sensitive data is only available for a brief time.
8. When PII data has been deleted, the submission is still listed in form transactions, but the XML data can't be viewed, and attachments can't be downloaded anymore.
9. The Finished Transactions setting is used to determine when to purge the completed transaction from the transaction record. For the strict policy, the transaction will be deleted 90 days after the transaction has finished.
10. The Transaction History setting (in the System Logs section) is used to determine when to purge the related transaction history data. For the strict policy, the transaction history record will be deleted 180 days after the transaction has finished.

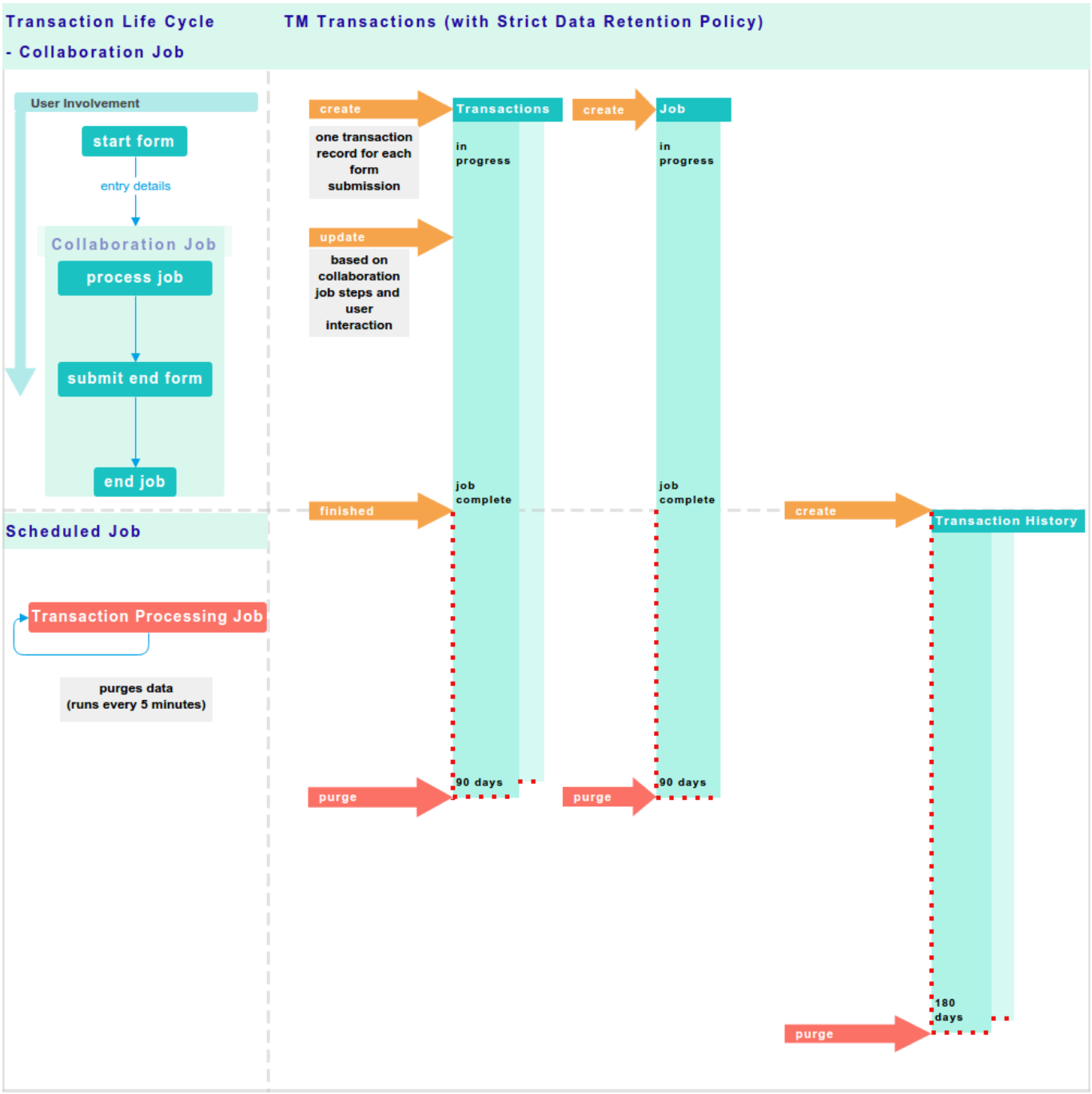
Next, learn about submission categories.

Collaboration Job Submission

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

If a submission is part of a collaboration job, it is not purged as part of standard submission purging. Instead, Manager uses the Finished Collaboration Jobs policy to purge the collaboration job and all its submissions, once the entire job has finished. It is important to emphasize here that, in contrast with a single form submit, a form submission that is part of a collaboration job is never automatically abandoned by Manager.

The image below illustrates the transaction life cycle for a collaboration job form submission that has been successfully completed.



The list below describes the process illustrated in the above diagram.

1. The form is started (opened) by the user and a transaction record is created. As this form involves a collaboration job, once the user submits the start form, it triggers the collaboration job processing which first creates a job record. This job record contains the job audit information and keeps track of all submissions related to the collaboration job.
2. One or more users who are involved interact as requested by the collaboration job. Typically, each user interaction is a separate submission reflected by creating a transaction record.
3. The last step is reached and the user responsible submits the end form.
4. On submission of the end form, the forms in the collaboration job are processed and delivered to the backend system.
5. Once delivered, it is considered a finished collaboration job reflected by a finished job record and a related set of one or more finished transactions. A transaction history record is now created for each of the finished transactions within the collaboration job.
6. From this point onwards, each transaction record and the corresponding transaction history record are retained in Manager according to the data retention settings.
7. The Finished Collaboration Jobs setting is used to determine when to purge the job record and all its related completed transactions. For the strict policy, these are deleted 90 days after the collaboration job has finished.
8. The Transaction History setting (in the System Logs section) is used to determine when to purge the related transaction history data. For the strict policy, the transaction history records are deleted 180 days after the collaboration job has finished.

However, a collaboration job may also be cancelled by a Manageroperations user. In this case, the incomplete job is marked as finished, and the same data retention procedure applies like a successfully completed job.

Next, learn about submission categories.

Abandoned Submissions

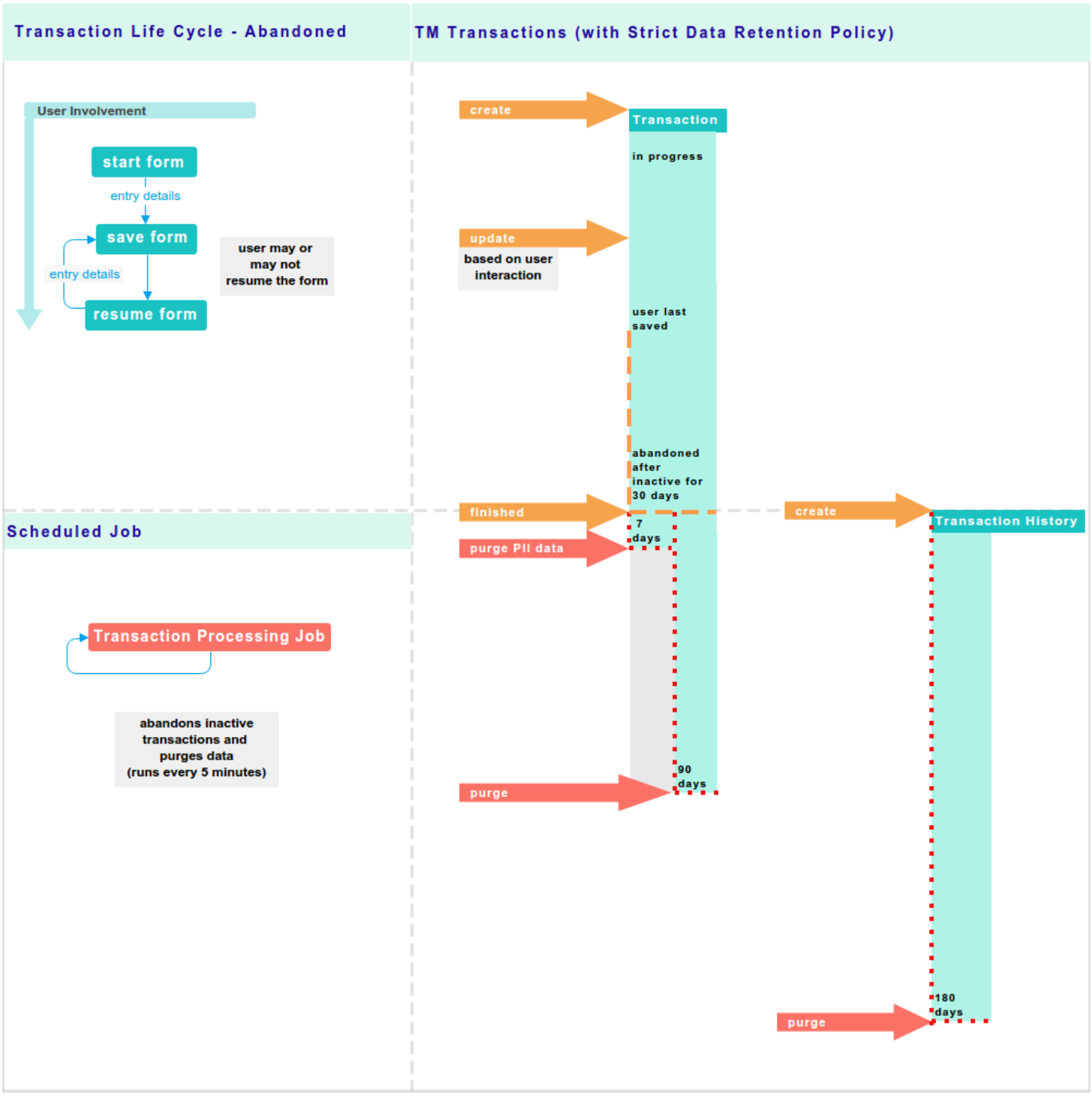
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A transaction can be abandoned in several ways:

- Explicitly by the user clicking the Cancel button or closing the browser while still on the first page. Please note that some forms may automatically save the transaction as the user navigates between different pages. In this case, the form will temporarily be marked as Saved. If the user does not return to the form and save or submit, it will be marked as abandoned.
- Explicitly by an administrator canceling from the Managerconsole (only when necessary).
- Automatically by a Manager job, where a user:
- Saved the form but did not resume it in a timely fashion.
- Submitted the form but it remained incomplete, waiting for say missing attachments.
- Automatically by a form session timing out.
- Unintentionally by say some break in internet communication.

It is a balancing act to optimize conversion rates whilst keeping your database as lean as possible. As with the case of a saved form, it is hoped that the user will soon resume, complete, and submit it. We advise that sufficient time is allowed for all users to resume their forms before automatically abandoning them. According to the strict policy, the Saved Transactions setting gives the user 30 days to return to a previously saved form.

The image below illustrates an example where a transaction, that was previously saved, is abandoned automatically after a period of inactivity (based on the Saved Transactions setting).



The following steps describe this process:

1. The form is started (opened) by the user and a transaction record is created.
2. The user saves the form with the intention of completing it later.
3. However, the user does not resume the form within the allowed time frame (30 days as per the saved transactions setting of the strict policy).
4. After the 30 days, Manager automatically abandons the transaction.

5. An abandoned transaction is considered a Finished Transaction. At this stage, a transaction history record is created, and the same data retention procedure applies like a successfully submitted and delivered (Completed Transaction) submission.

In most cases, abandoned transactions are deleted according to the Finished Transactions setting. The only exception is where a user opens the form and leaves it open without having saved it (this includes any background saving). In this case, the transaction is automatically abandoned one day after first opened and then it's deleted 2 days after being abandoned.

Data for abandoned submissions is delivered if the abandoned delivery channel is configured in TM.

Next, learn about submission categories.

Authenticated vs Unauthenticated Transactions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager supports both authenticated or unauthenticated transactions.

When a transaction is authenticated, the user completing the transaction has logged into and been identified by Manager, usually by supplying a username and password.

When a transaction is unauthenticated, the user completing the transaction has not logged and doesn't have an identity with Manager.

Manager provides similar features to both authenticated and unauthenticated users, but the features are implemented differently.

Form Pre-fill

Authenticated	Unauthenticated
When a user is authenticated, their identity, including the data stored about them in back-end systems, can be used to partially complete forms with information that is already known about them.	When a user is unauthenticated, data can be pre-filled by using request parameter prefill mapping, which allows pre-fill data to be passed to a transaction through **HTTP**[1] request parameters, session attributes, or cookies. After that, any personal details from the form can be used to call a dynamic data service to recover all missing anonymous user data from a back-end system so the form is pre-filled.

Form Save & Resume

Authenticated	Unauthenticated
When a user is authenticated, they can save a partially completed transaction and associate it with their identity. Later, if they log back into their account, the incomplete transaction will be available to them.	When a user is unauthenticated, the save and resume functionality relies upon a **reference code**[2], similar to an airline booking. Optionally, a security question can be defined to provide a basic layer of security.

Transaction History

Authenticated	Unauthenticated

[1] HTTP (HyperText Transfer Protocol) is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

[2] A reference number is a unique identifier assigned to each transaction.

When a user is authenticated, they can log into Manager to see a history of their transactions. Generally, they only see a summary of the transactions, as data retention policies will often result in the details being purged.	When a user is unauthenticated, one method of providing a transaction history is sending a confirmation email to the user once the transaction is complete.

Tasks

Authenticated	Unauthenticated
When a user is authenticated, tasks can be assigned to them in Manager. These tasks will then appear in their task list.	When a user is unauthenticated, tasks can be assigned to them by directing them to a dynamically generated URL. When they visit this URL, they'll be prompted to perform the task.

Best Practices

Based on the content covered in this topic, you might be wondering if it's better to use authenticated or unauthenticated transactions. As with most decisions of this nature, it depends on exactly what your organization is trying to achieve.

For context, consider the following details:

1. Many organizations already have their own methods of authentication integrated with their back-end systems and would rather not introduce a new mechanism of logging in via a username and password.
2. Users are often reluctant to sign up with a username and password purely for the sake of saving and resuming a transaction. If they're forced to register, many users will abandon the transaction.
3. Identity — and specifically, the identity of customers outside your organization's firewall — can be difficult to implement. Manager does support industry-standard technologies, such as **LDAP**[1] and **Single Sign-On**[2] (using SAML), but these do add complexity to the implementation process.

For these reasons, we generally recommend the following:

1. **External customers should be treated as unauthenticated users**. This may have an impact on the ability to pre-fill data in their forms, but this approach simplifies the integration requirements, the security implications, and the time to market.
2. **Internal users, such as help desk and call center staff, should be treated as authenticated users**. This is usually simpler to implement as internal users will be already be authenticated on the network, and implementing LDAP or Single Sign-On is relatively straight-forward.

---

[1]Lightweight Directory Access Protocol (LDAP) s an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol network.
[2]Single sign-on (SSO) is a property of access control of multiple related, yet independent, software systems. With this property, a user logs in with a single ID and password to gain access to any of several related systems.

In some cases, organizations will require external customers to be authenticated, and as we've covered in this topic, this is most certainly possible. it will, however, affect both the time and cost of implementation.

Next, learn how to change the access type for a form space.

Configure Form Abandonment

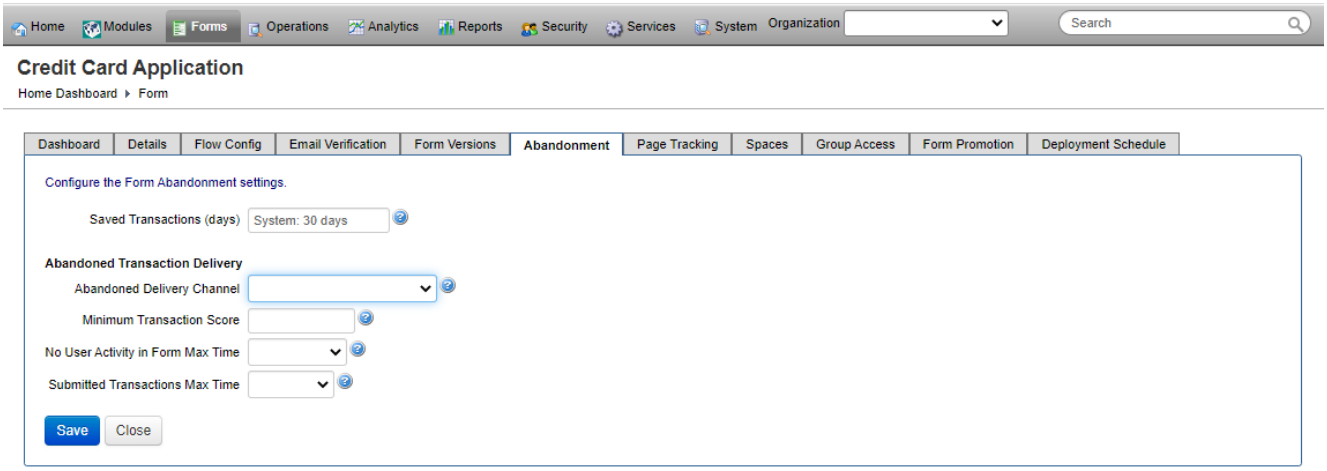
Journey Manager (JM) | Platform Developer | 5.1 & Higher

Transactions can become abandoned based on the user actions, such as canceling a transaction, or based on the form's abandonment settings. You can use abandonment configuration to define when a form is considered as **abandoned**[1] after it has been saved, and what happens after a form transaction is abandoned. Manager allows you to configure abandonment rules of form's transactions at a global, organization, or form level. These configurations depend on:

- Customer requirements
- Server performance
- Specific data retention policies

To configure form abandonment:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Click the **Abandonment** tab.



4. Specify the maximum age of user saved transactions after which they will be automatically abandoned (marked as Abandoned) in the **Saved Transaction (days)** field. The default is 30 days, which is either the organization data retention setting, if exists, or the system data retention setting. When a saved transaction reaches this maximum age, it is marked as Abandoned by the system. The maximum age is based on the number of days a form has been untouched since it was last saved.
5. Select the organization delivery channel for delivering abandoned submissions from the **Abandoned Delivery Channel** dropdown list. The next three fields are enabled when a delivery channel is selected.

[1]Any transaction that has been deemed as Abandoned in Manager. When a user leaves the application without explicitly canceling it, the transaction lives in Started status for a period defined in Manager, after which it is considered as an Abandoned transaction.

Manager version 17.10 and higher has the following fields available for Composer forms only. However they are still displayed for Maestro forms.

6. Edit the **Minimum Transaction Score** to specify the transaction score that a form transaction must reach for it to be delivered to the Abandoned Delivery Channel. A transaction score is a numerical value calculated by the form, generally based on how much of the form was filled in. When left blank, all abandoned form transactions are delivered. Otherwise, only those that have been "adequately" filled in, meaning they have reached the Minimum Transaction Score, are delivered. Minimum Transaction Score specifies a minimum value representing how much of the form must be filled in for it to be delivered to the abandoned delivery channel. That is, a transaction score is a numerical value calculated by the form to indicate how much of the form was filled in. If you would like only submissions that exceed a certain transaction score to be delivered, enter the minimum transaction score into this field.
7. Select a time from the **No User Activity in Form Max Time** dropdown list to specify a time frame for allowing any form transaction by an anonymous user to remain untouched before marking it as abandoned. This is the period of time after which anonymous transactions are marked as abandoned. These transactions have been saved in the background and untouched by a user.

A form transaction must also have been background saved for this setting to apply.

8. Select a time from the **Submitted Transactions Max Time** dropdown list to specify a time frame for allowing any form transaction that has been submitted by a user to remain incomplete before marking it as abandoned. This is the maximum age before a submitted but incomplete, for example missing attachments, transaction are marked as abandoned.

A form transaction may be incomplete because it's waiting for a post-submission requirement such as an attachment to be provided by the user.

9. Click **Save** to update the changes.

You can view abandoned transactions to troubleshoot user form interactions.

Next, learn hot to configure form details.

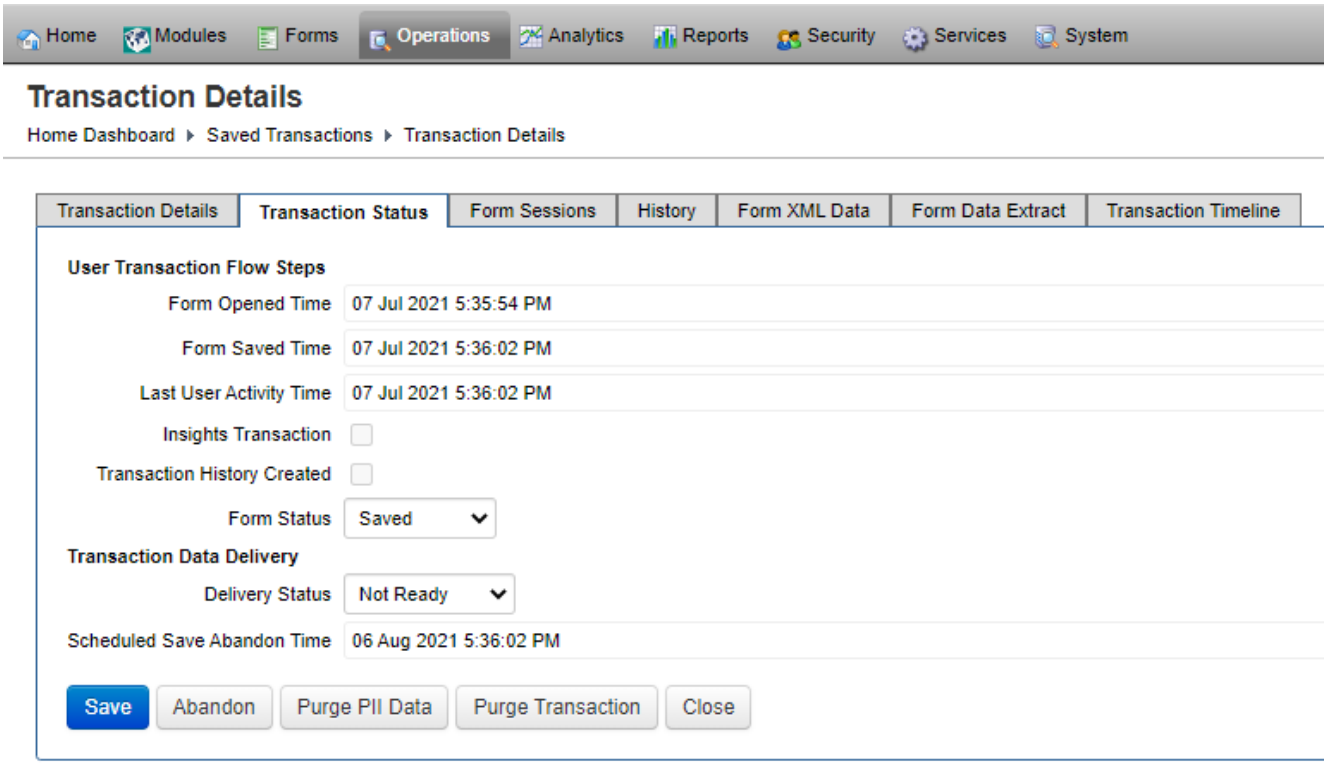
Abandon a Transaction

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows an administrator to abandon a transaction manually. Only transactions with a Form Status as Opened, Saved, or Submitted can be abandoned manually. You can use this functionality to prevent a user from accessing certain transactions.

To abandon a transaction:

1. Select **Operations > Form Transactions** or **Operations > Saved Transactions** or **Forms > Form**, locate a form and click **ID** in the **Latest Transactions** section.
2. Locate a transaction and click **View**.
3. Select the **Transaction Status** tab to update a form status.



4. Select **Abandoned** from the **Form Status** dropdown list.
5. Click **Save** to update the changes. The transaction is moved to the abandon transactions.

Next, learn how to view saved transactions.

Transactions and Tasks Abandonment

Journey Manager (JM) | System Manager / DevOps | 18.05

When a user submits a form, Manager creates a transaction, associated with this form. As a result of transaction processing and user inputs, it can become completed or abandoned. Manager considers abandoned transactions as completed transactions, so they can be purged. Abandoned transactions can also be configured to deliver data depending on a threshold value, which determines how incomplete form data was before the transaction became abandoned. Manager stores abandoned transactions along with all other transactions that are part of the transaction life cycle.

Transactions can be abandoned automatically or manually.

Automatic Abandonment

Automatic abandonment can happen under varying conditions depending on your Manager configuration. Manager applies the following rules to govern when draft submissions are automatically abandoned. The rules use a set of data retention parameters that you may need to configure for your particular requirements.

Transaction Abandonment

Data retention policy, which determines how long a saved transaction is maintained in Manager before it is automatically abandoned. Forms with abandonment delivery can have a more aggressive abandonment policy than the standard 1 day (24 hours) policy.

Form Status	Standard	Abandoned Delivery
Open / Started	Abandon after 1 day	Not applicable
Background Saved	Abandon after 1 day [1]	Abandon as per the form level **No User Activity in Form Max Time** setting. Transaction will then be delivered if the transaction score threshold has been exceeded. Transactions which are rendered but never actually captured any data or transactions which are never saved by a user, even though they are background saved by Maestro Editor or using the **`Form.backgroundSave()`** Maestro API.
User Saved	Abandon after data retention setting for **Saved Transactions (days)** [2]	Abandon after the data retention setting for **Saved Transactions (days)** [2]
Submitted	Do not abandon automatically [3]	Abandon as per the form level **Submitted Transactions Max Time** setting. Transaction will then be delivered if the transaction score threshold has been exceeded.

Expired	Abandon 7 Days after expiry	Abandon 7 Days after expiry

[1] - You can't change this 1 day abandonment policy time interval.

[2] - Saved Transactions (days) can be set at the global, organization and form levels.

[3] - Transactions in this state are mainly legacy. Normally, with a FTX flow, transactions will be completes before the transaction submits. When a transaction is left in the submitted state, it will be waiting for attachments or payments and the expectation is that help desk staff or administrators will be reviewing these transactions and either helping the users complete the form or manually abandoning them.

Task Abandonment

Tasks with configured expiration time are abandoned after this time has elapsed. If no expiration time is set, then the save abandonment time is used.

Form Status	Standard	Abandoned Delivery
Assigned / Open / Started	Abandon after data retention setting for **Saved Transactions (days)** [1]	Not applicable
Background Saved	Abandon after data retention setting for **Saved Transactions (days)** [1]	Abandon as per the form level **No User Activity in Form Max Time** configuration. Transaction will then be delivered if the transaction score threshold has been exceeded.
User Saved	Abandon after data retention setting for **Saved Transactions (days)** [1]	Abandon after data retention setting for **Saved Transactions (days)** [1]
Submitted	Do not abandon automatically [2]	Abandon as per the form level **Submitted Transactions Max Time** setting. Transaction will then be delivered if the transaction score threshold has been exceeded.
Expired	Expiry time overrides the above for **Assigned**, **Open**, and **Started** statuses and the tasks will abandon as per the expiry setting.	Expiry time overrides the above for **Assigned**, **Open**, and **Started** and the tasks will abandon as per the expiry setting.

[1] - Saved Transactions (days) can be set at the global, organization and form levels.

[2] - Transactions in this state are mainly legacy. Normally, with a FTX flow, transactions will be completes before the transaction submits. When a transaction is left in the submitted state, it will be waiting for attachments or payments and the expectation is that help desk staff or administrators will be reviewing these transactions and either helping the users complete the form or manually abandoning them.

Transaction and Task Purging

Form Status	Date	Record
Open / Started	Not applicable	Transactions which were abandoned from this state will be purged in a fixed 2 days
Abandoned / Completed	It'll be purged as per the **Finished Transaction PII Data (days)**	It'll be purged after delivery (if required) as per the **Finished Transactions (days)**

For Collaboration Jobs transaction data purging is a per standard policy but transaction record purging is different and the record will be purged when the collaboration job is purged. Collaboration Job purging is controlled by the 'Finished Collaboration Jobs (days)' data retention setting.

Manual Abandonment

There are situations when manual abandonment is required. It can come from:

- A user via Workspaces
- A system administrator via the Manager dashboard
- A Fluent API call using the `new TxnUpdater(txn).setFormStatus(Txn.FORM_ABANDONED).update()` method. For more information, see Fluent API documentation for your specific Manager version.

Next, learn how to view submissions history.

Transaction Details

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you view detailed information of a selected transaction associated with a form. This is handy when you troubleshoot form submissions.

To view transaction details:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View** to see transaction details.


Home
Modules
Forms
Operations
Analytics
Reports
Security
Transaction Details
Home Dashboard ▸ Job Details ▸ Collaboration Job Action ▸ Transaction Details
Transaction Details
Transaction Status
Form Sessions
History
Form XML Data
Transaction Details
Submission ID 28010
Tracking Code F2LG4U
Receipt Number F2LG4U
Collaboration Job F2LG4U
Submission Key bc1608f58710c81b28ce597fdf57ddf3
Storage Service Database Submission Data Storage
Revision Number 1
Form Status Completed
Form Started
User Saved
Completed Time 28 Jan 2019 10:50:51 AM
Time Taken to Submit 4 seconds
Form Details
Form Name Job Maestro 1 18.05 STD
Form Version 18.5-Branch
Form Type Maestro Form
Release Version 18.05.3
Organization QA Auto Tests - Do not change these forms
Space Work Space
User Information
Created By joe
Completed By joe
Contact Email Address
Sent Completed Email To
User Operating System Windows 10
User Browser Chrome 71
Close

Some transaction details are different depending on the content and status of the selected transaction. For example, if the transaction encountered any errors, the **Errors** tab is displayed.

3. Check the Transaction Details section, including:
   - Submission ID
   - Tracking code
   - Receipt number
   - The collaboration job number associated with this transaction
   - Storage service used to persist this transaction
   - Revision number
   - Form status
   - Form started
   - Used saved - shows whether a user saved a form or it was background saved.
   - Saved time - shows the time stamp when the form was saved.
4. Check the Form Details section, including:
   - Form name
   - Form version
   - Form type
   - Release version
   - The organization and space associated with the form.
5. Check the User Information section, which documents the following details of the user who completed the transaction:
   - Created by and completed by.
   - Contact Email Address of the user that created the transaction.
   - User Operating System that was used to create the transaction.
   - User browser that was used to create the transaction.

Next, learn how to change a transaction status.

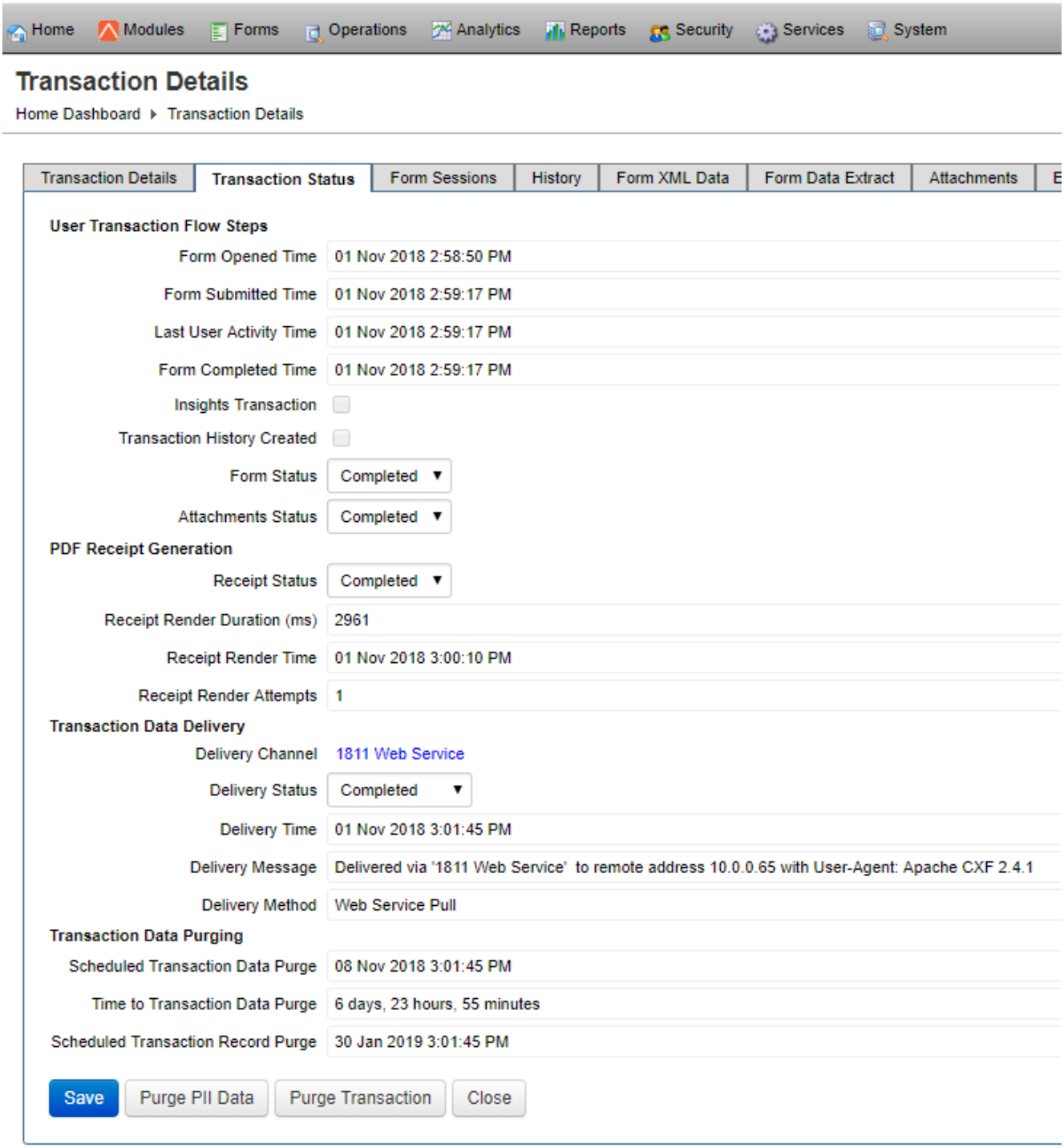
Transaction Status

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager allows you to check a status of a transaction and also update it, if you need to correct a transaction flow. When Manager creates a transaction, it assigns it a status that depends on the current progress of the transaction. You can also review relevant form's, receipt's, and delivery's properties pertaining to the current **Transaction Status**, which you can see in the Form Transactions list. This means that a different set of properties is shown for the same transactions as its status changes. Related properties are grouped into sections for easy understanding.

To check a transaction status:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View**.
3. Select the **Transaction Status** tab.



4. Check the date and time when a form was opened in the **Form Opened Time** field.
5. Check the date and time when the form was submitted or initially saved by the user in the **Form Submitted Time** field.
6. Check the date and time of the form last activity, such as viewing or rendering, in the **Last User Activity Time** field.
7. Check the date and time when the form was completed in the **Form Completed Time** field.

8. Check the Save Challenge Data is selected for this form so users can resume form applications using the Save Challenge.
9. Check whether this is an Insights transaction in the **Insights Transaction** field.
10. Check whether the transaction history was created for this transaction in the **Transaction History Created** checkbox.
11. Check the transaction status in the **Form Status** field.
12. Check the PDF receipt status under the **PDF Receipt Generator** section.

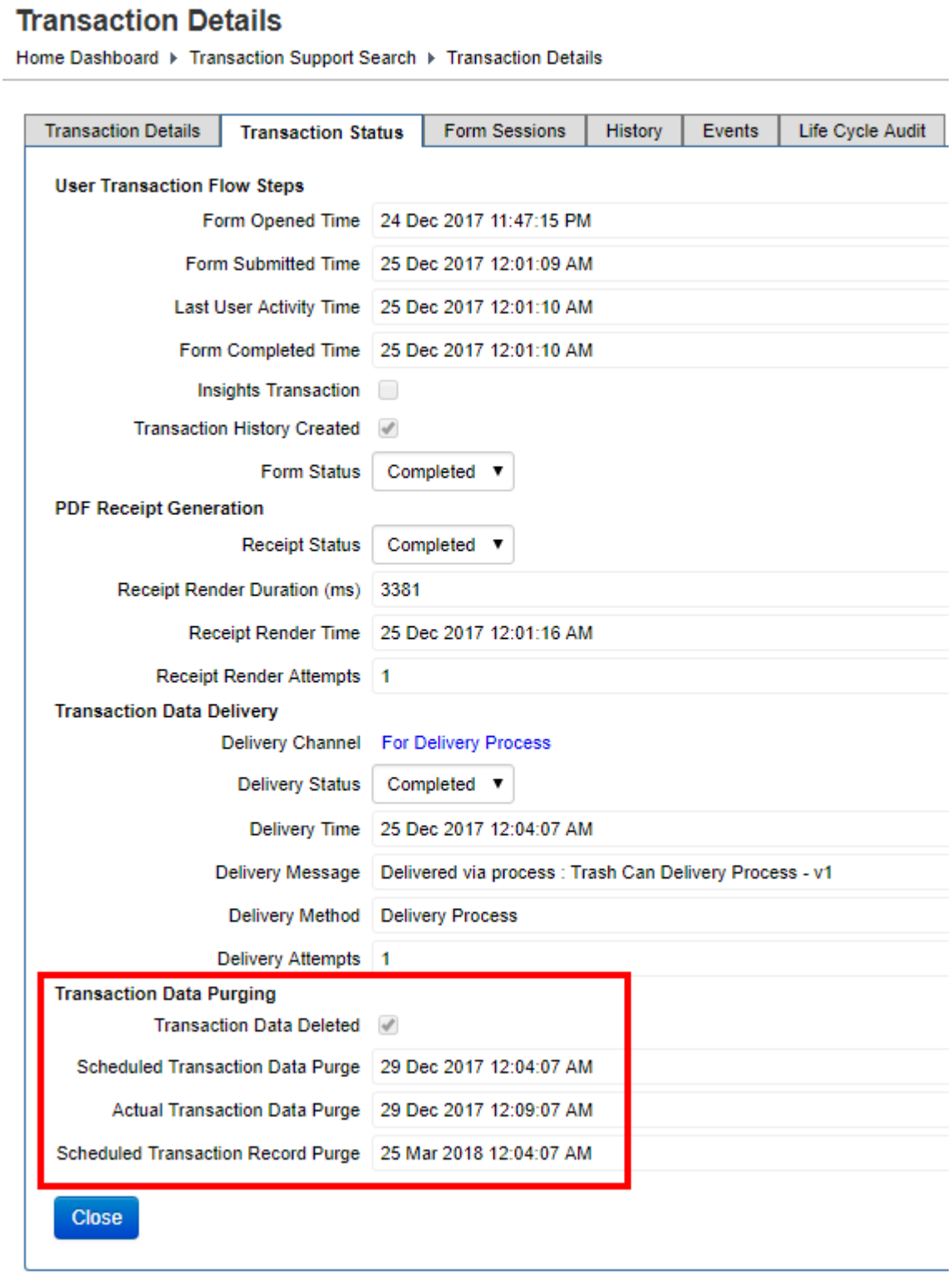
    A receipt status can be one of the following:

    - Ready - a receipt is waiting to be allocated to a receipt generator.
    - In Progress - a receipt is in a receipt generator's queue.
    - Completed - a receipt has been generated.
    - Error - there is an error during receipt generation. A retry may fix it.
    - Error - No Data - there is no XML data available to generate a receipt.
13. Click a link in the **Delivery Channel** field to view the configuration of a delivery channel used by this transaction.
14. Check the **Delivery Status** of the transaction under the **Transaction Data Delivery** section. You can manually update the delivery status of the transaction depending on the transaction type and its current status.

A delivery status can be one of the following:

- Completed - a transaction has been successfully delivered, based on the configured delivery channel.
- Undeliverable - a delivery is not possible. Check the error logs for further details of the error.
- Not Ready - a transaction is not ready for delivery yet. This is the start state before a transaction has been completed by the user. Generally, the receipt render service will kick the transaction from this state to the Ready state when the receipt is rendered.
- Not Required - a delivery is not required. It may be that this form is part of a collaboration job and the delivery is to be handled separately say as a consolidation of the transaction details from all the associated forms.
- Ready - a transaction is ready to be delivered. A transaction will be marked as Ready until the delivery channel is processed, or it may remain marked as Ready if a delivery channel is not configured.
- In Progress - a delivery has been started but is waiting to be completed.
- Paused - a transaction delivery is paused. This transaction's status can be changed by some other means, for example by a Groovy Scheduled Task.
- Error - a transaction delivery has failed and is waiting to be retried.
- Sent Email - a secure email has been sent and now a transaction is waiting for the user to log in and process it.
- Pending

15. Check the form's delivery time in the **Delivery Time** field.
16. Check the **Delivery Message** field to see an additional information on how the form was delivered.
17. Check what delivery method was used for this form in the **Delivery Method** field, for example, Delivery Process.
18. Check how many delivery attempts it has taken to deliver a form in the **Delivery Attempts** field.
19. Click a collaboration job number in the **Collaboration Job** field to view the collaboration job details. This is only applicable if this form is configured to be used with collaboration jobs.
20. Check the collaboration job status in the **Job Status** field.
21. Check the **Transaction Data Purging** section to see when this transaction is to be purged or it has been already purged. If the **Transaction Data Deleted** checkbox is selected, then the PII data was purged. Check the **Actual Transaction Data Purge** and **Scheduled Transaction Data Purge** times to understand when it was purged. Check the **Scheduled Transaction Record Purge** time to see when the transaction data record is due to be deleted. For example, from the transaction details below we can tell that the PII data was purged (the **Transaction Data Deleted** is selected) a few minutes after it had been scheduled to be purged (the **Actual Transaction Data Purge** and **Scheduled Transaction Data Purge**). Also, the transaction data record is due to be deleted on the 25 March 2018 (the **Scheduled Transaction Record Purge**). Note that after the 25 March, this transaction along with its submission details will no longer exist. Only a transaction history record remains, which contains some details about the transaction.


Transaction Details
Home Dashboard ▸ Transaction Support Search ▸ Transaction Details
Transaction Details
Transaction Status
Form Sessions
History
Events
Life Cycle Audit
User Transaction Flow Steps
Form Opened Time
24 Dec 2017 11:47:15 PM
Form Submitted Time
25 Dec 2017 12:01:09 AM
Last User Activity Time
25 Dec 2017 12:01:10 AM
Form Completed Time
25 Dec 2017 12:01:10 AM
Insights Transaction
Transaction History Created
Form Status
Completed
PDF Receipt Generation
Receipt Status
Completed
Receipt Render Duration (ms)
3381
Receipt Render Time
25 Dec 2017 12:01:16 AM
Receipt Render Attempts
1
Transaction Data Delivery
Delivery Channel
For Delivery Process
Delivery Status
Completed
Delivery Time
25 Dec 2017 12:04:07 AM
Delivery Message
Delivered via process : Trash Can Delivery Process - v1
Delivery Method
Delivery Process
Delivery Attempts
1
Transaction Data Purging
Transaction Data Deleted
Scheduled Transaction Data Purge
29 Dec 2017 12:04:07 AM
Actual Transaction Data Purge
29 Dec 2017 12:09:07 AM
Scheduled Transaction Record Purge
25 Mar 2018 12:04:07 AM
Close

22. Click **Purge PII Data** to purge only **PII**[1] data of this transactions. Active transactions will be abandoned, if they are not already abandoned. | 18.11
23. Click **Purge Transaction** to purge this transaction immediately. Active transactions will be abandoned, if they are not already abandoned. | 18.11

> You can purge PII data and transactions in bulk.

24. Click **Save** to update the changes.

You can use Fluent API to retrieve transaction status details, for example, a transaction's Last User Activity Time.

```
import com.avoka.tm.query.*

def txn = new TxnQuery()
    .setTrackingCode("1234567") // change for your transaction
    .firstValue()

txn.timeUserLastModified
```

Next, learn how to track submissions.

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

Change Transaction Status

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to change a status of a form transaction manually, for example, change its status from an abandoned to a saved status.

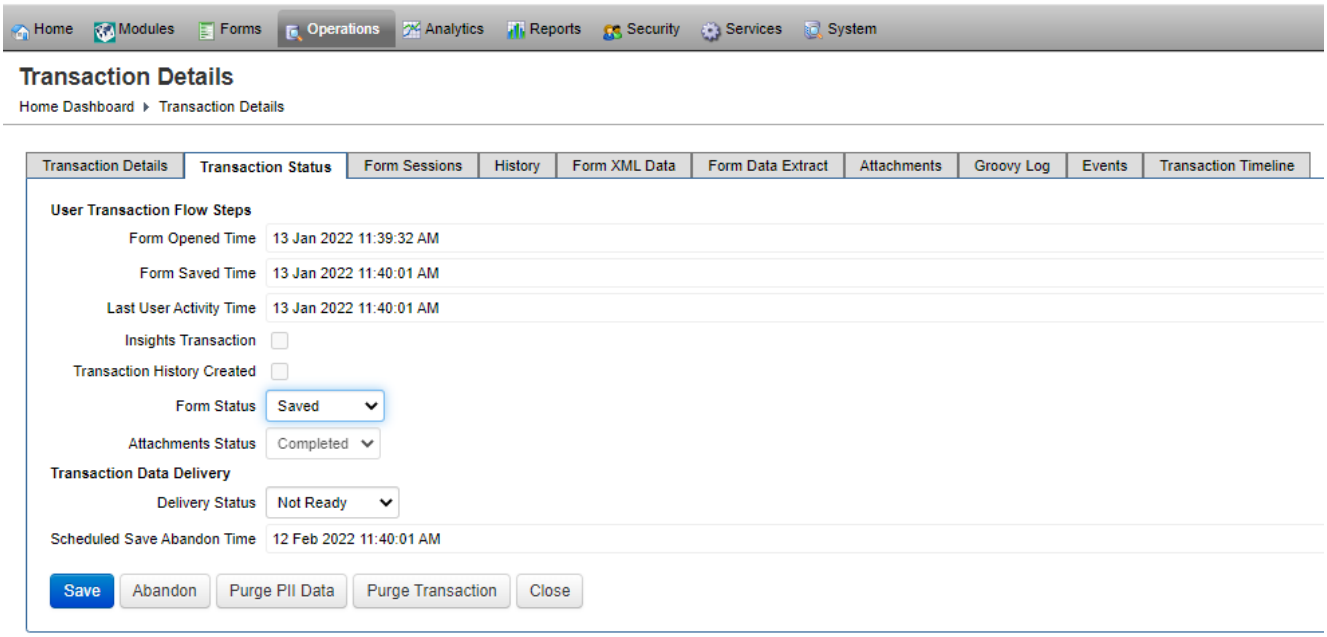
> Only a Manager administrator can change an abandoned transaction status to **Saved**.

Manager provides you with the following methods to change a transaction status:

Method 1

To change a transaction status :

1. Select **Operations > Form Transactions**, or select **Operations > Saved Transactions**, or select **Forms > Form**.
2. Locate a form and click **ID** in the **Latest Transactions** section.
3. Find a transaction you want to update and click **View**.
4. Select the **Transaction Status** tab.

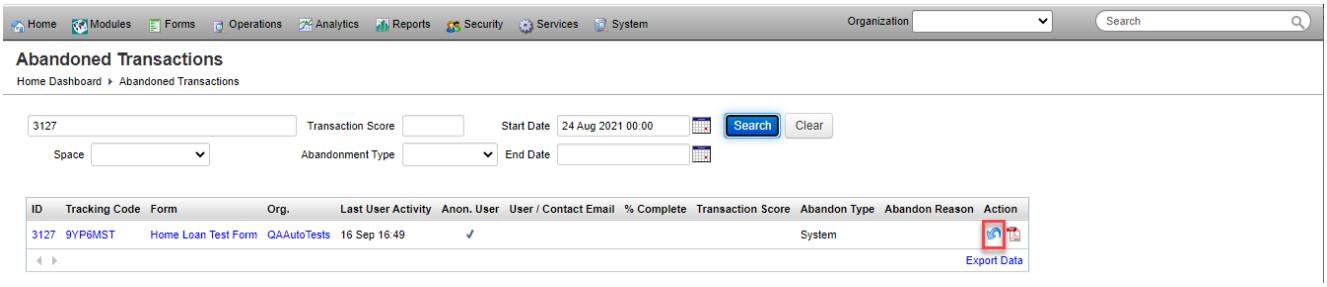


5. Select a desired form status from the **Form Status** dropdown list, which can be **Saved** or **Abandoned**,.
6. Click **Save** to update the changes.
7. Check the updated transaction status in either abandoned transactions or saved transactions.

Method 2

To change a transaction status:

1. Select **Operations > Abandoned Transactions**.
2. Locate a transaction and click **Make Abandoned Transaction Saved**.



3. Click **OK** to confirm. The transaction is moved from the abandoned transactions to the saved transactions.

Next, learn how to configure form abandonment.

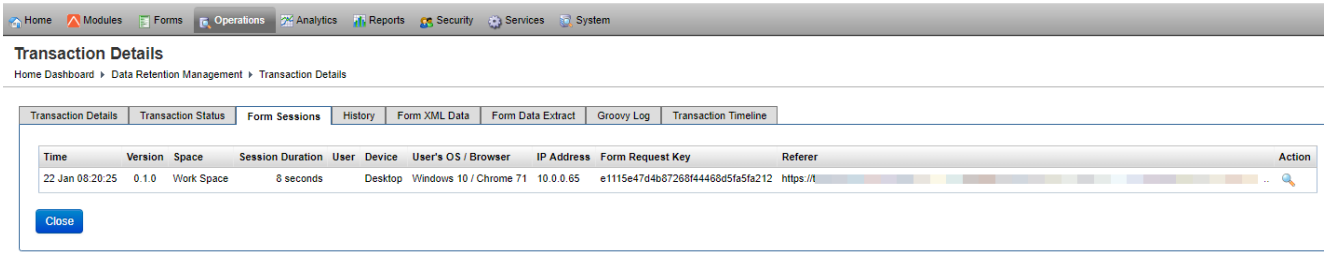
Form Sessions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager creates a form session when a user opens a form and ends it as soon as the user leaves the form. Manager allows you to view and analyze each of the form sessions that were created for a transaction.

To view form session information:

1. Locate a transaction.
2. Select the **Form Sessions** tab.



3. Check the following information for each session:
   - Date and time of the session.
   - The version of the session.
   - The form space associated with the selected transaction.
   - The session duration, which is the amount of time (in minutes and seconds) that the user spent on the transaction in this session.
   - User.
   - The device used to access a form from.
   - User OS/ Browser used to create a session.
   - The IP Address of the user that created a session.
   - Form Request Key, which is a unique identifier of a session.
   - The referrer URL that the user used to access the form that was used to create a session.
4. Click **Action** to check Form Request Details.

This data can be used to analyze the efficiency of a form.

Next, learn how to view transaction history.

Transaction History

Journey Manager (JM) | System Manager / DevOps | 22.04

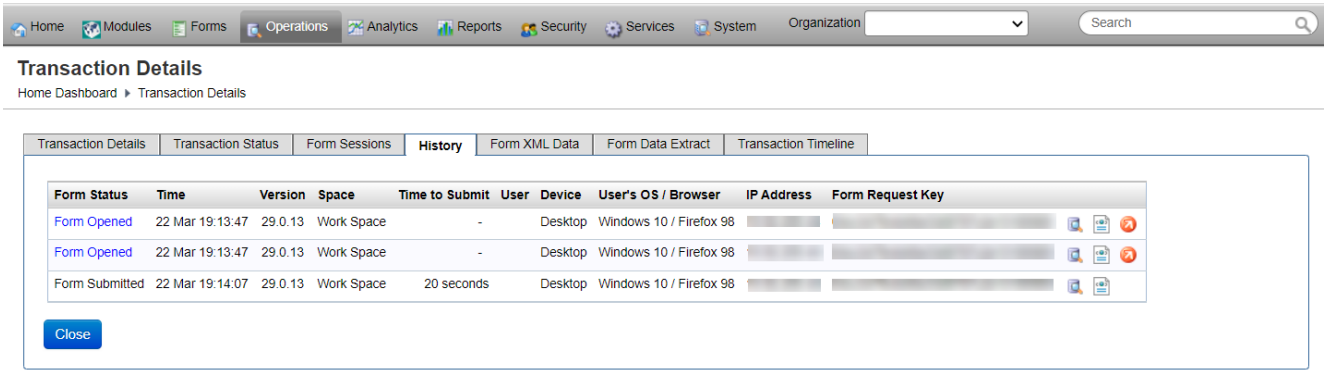
Manager allows you to view transaction history of user's interaction with a form. You can use this functionality for transaction's troubleshooting, analysis, and audit.

> We recommend using our Journey Analytics product to check transaction's trends and analyze user's behavior.

When a transaction is finished, Manager writes all relevant submission information, such as final status, form name, time to submit, and browser information into a transaction history table. This table does not contain any **PII**[1] data, aside from an email address, which is used for security auditing purposes.

To view a transaction's history:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View**.
3. Select the **History** tab to see all transaction versions.



4. Check the following information for each transaction:

[1]Personally identifiable information (PII) is any data that could potentially identify a specific individual. Any information that can be used to distinguish one person from another and can be used for de-anonymizing anonymous data can be considered PII.

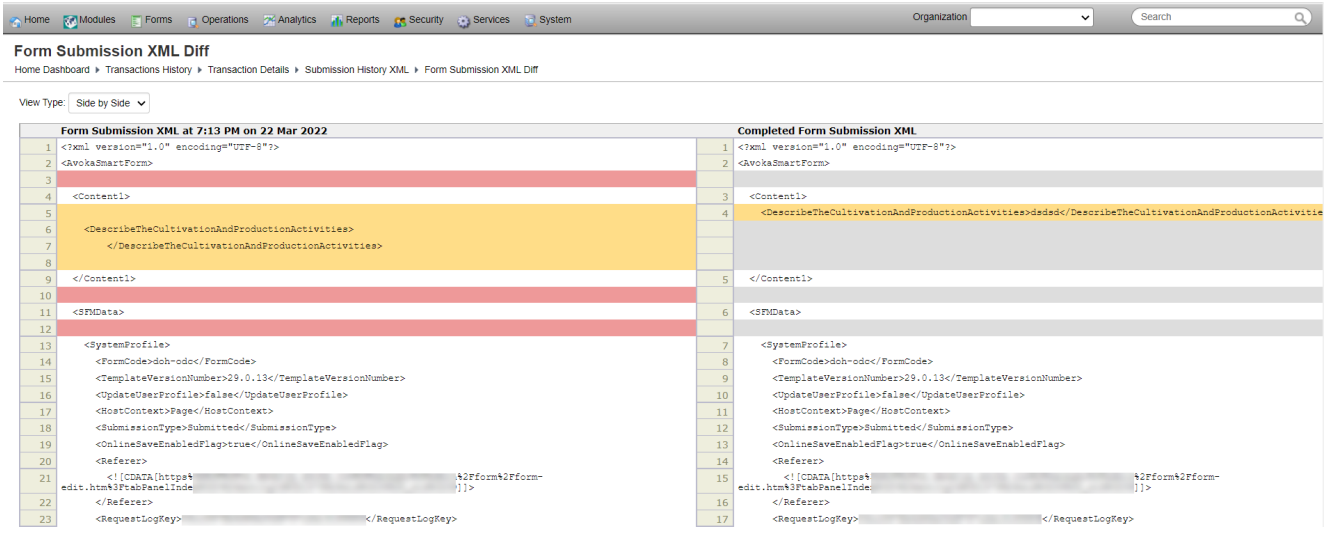
- Form status
- Date and time of a transaction
- Version of a transaction. A new version of a transaction is created each time a user opens and saves a form until the form is submitted.
- A form space associated with the selected transaction
- The time to submit, which is a time it took for a user to fill in a form and submit it
- User. The field is blank for an anonymous use
- The device used to access a form from
- The user's OS/ browser of the user that created a transaction. This is used for auditing purposes.
- The IP address of the user that created a transaction. This is used for auditing purposes.
- Form Request Key, which is a unique identifier of a transaction

5. Click **Form Session** to view details about the transaction in Form Request Details.
6. Click **View XML History** to download a copy of the submitted XML data. This option is commonly available to Composer Forms.

   > The View XML History option is available only if submission XML data is logged. Select the **Log Submission History XML** checkbox in the form details configuration to start logging submission XML data.

7. Click **View XML Diff** to view the difference between the submitted XML data and the XML data of the blank form with no input from the user. This option is commonly available to Composer Forms.

   > The View XML Diff option is available only if submission XML data is logged. Select the **Log Submission History XML** checkbox in the form details configuration to start logging submission XML data.



8. Click **Close**.

Next, learn how to track submissions.

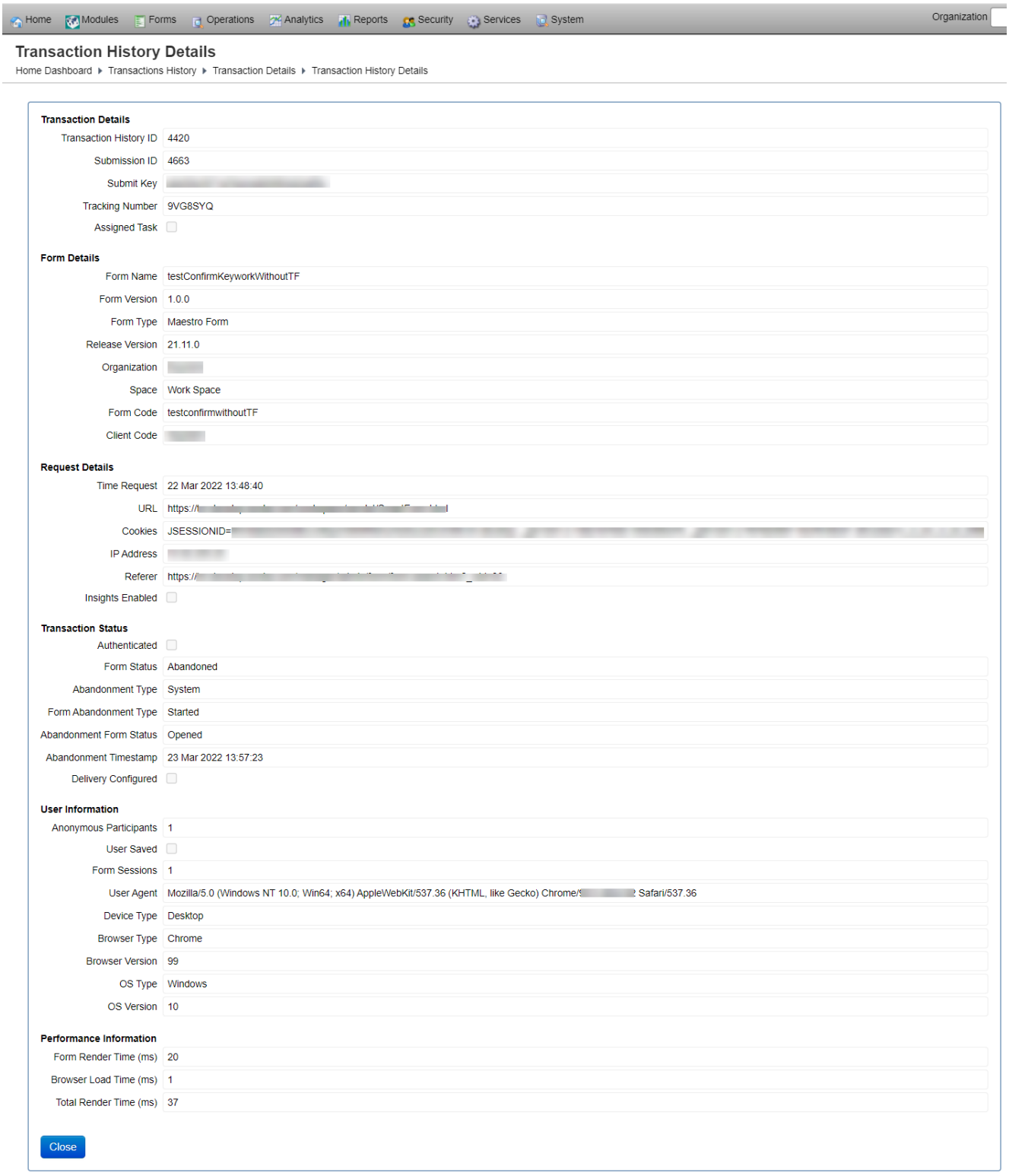
Transaction History Details

Journey Manager (JM) | System Manager / DevOps | 21.11

Manager allows you view detailed information of a selected transaction associated with a form. This is handy when you troubleshoot form submissions.

To view transaction history details:

1. Select **Operations > Transactions History**.
2. Locate a transaction and click **View** to see the transaction details.



For more information, see transactions details.

3. Click **Close**.

Next, learn how to change a transaction status.

Form XML Data

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view the **XML**[1] data of a selected transaction submitted by the user. An administrator can access and edit the submitted data to re-attempt a delivery and fix the following issues:

- A user completed a form, but form data has not yet been purged, and an error is preventing receipt generation or delivery.
- A user entered characters that has caused the back-end processing issues. Even if all attempts in a form's design have been made to capture invalid data before a form is submitted, there may be rare occasions where some invalid data entered into a form may slip through and cause the receipt generation or delivery to be blocked.
- Clean the data before it reaches the user and the back-end processing.

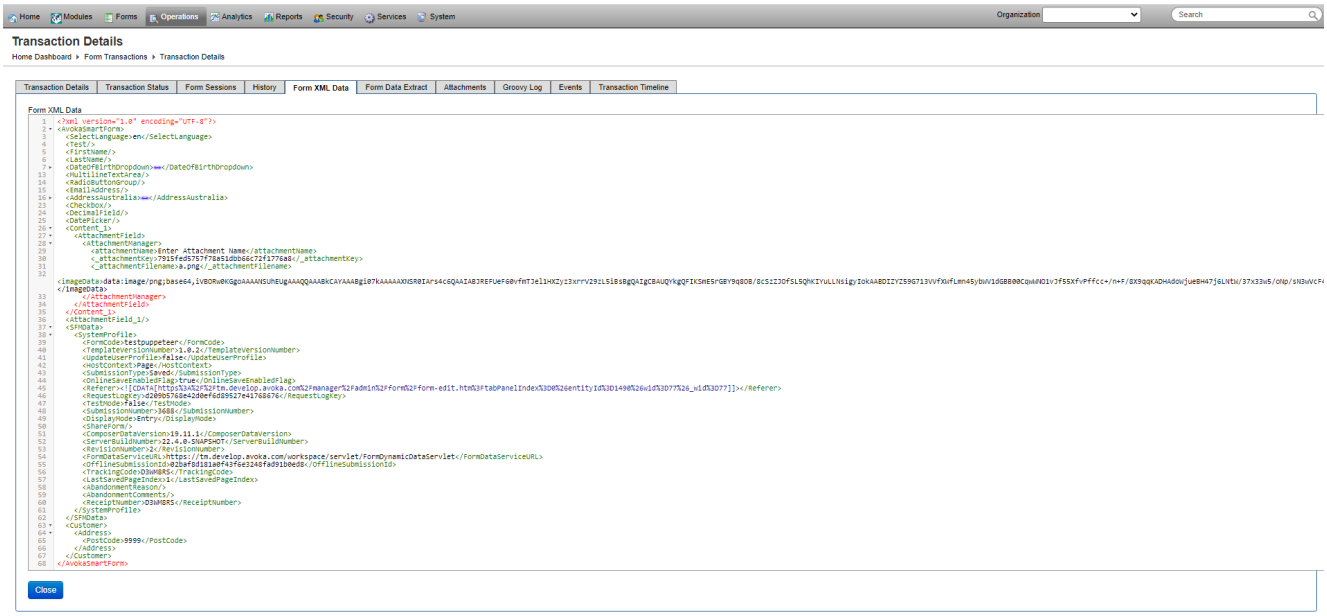
Whenever an administrator views or changes user XML data, Manager logs the operation to the audit log. When an administrator changes the data and saves the changes, the resulting XML is verified and stored as a new submission history entry, preserving the data previously submitted by the user.

To view form XML data of a transaction:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View**.
3. Select the **Form XML Data** tab to view the transaction XML data. For more information on the structure of the XML file, see Form XML Data.

   This tab is not shown if transaction data is purged.

[1]XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.



4. Click **Close**.

You can change some data formatting, such as the submit date string.

Next, learn how to track submissions.

Form Data Extract

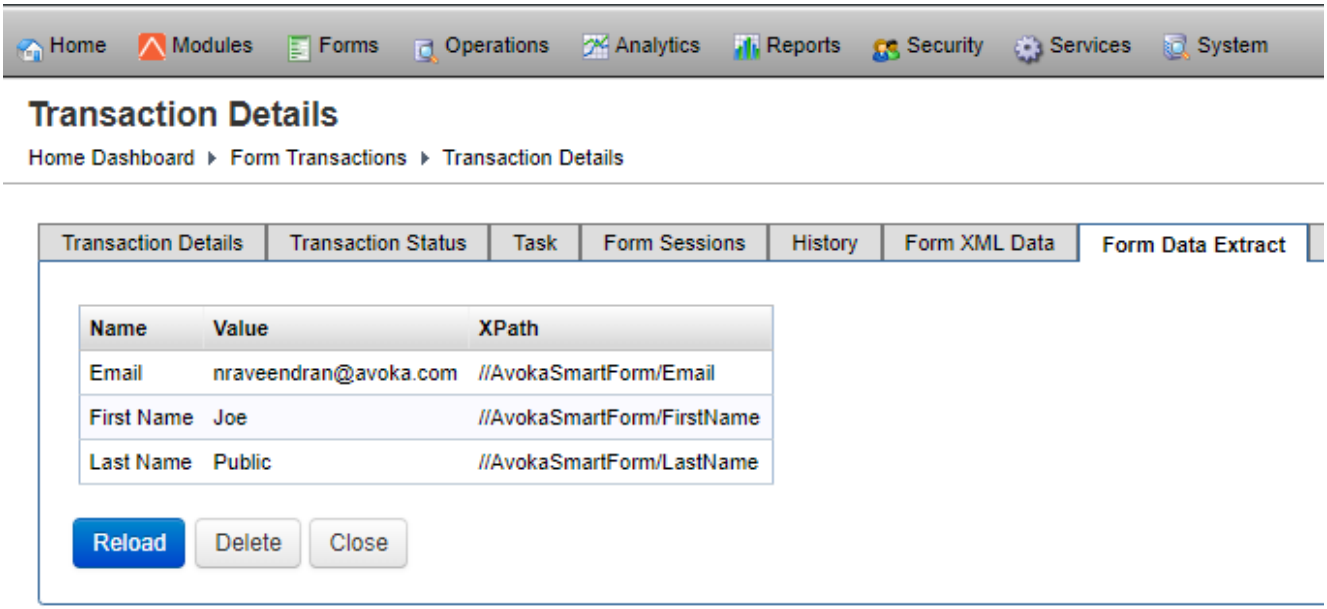
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view all items and their values that were extracted from the submitted XML data of a form.

> The **Form Data Extract** tab is only available if form data extracts are configured.

To view a form data extract:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View**.
3. Select the **Form Data Extract** tab.



   Manager extracts these values and places them in the **Value** column for each of the configured data extracts.

4. Click **Reload** to refresh form XML data extract values.
5. Click **Delete** to remove the data extracts.
6. Click **Close**.

For more information, see view form submission data extract.

Next, learn how to view form submission data.

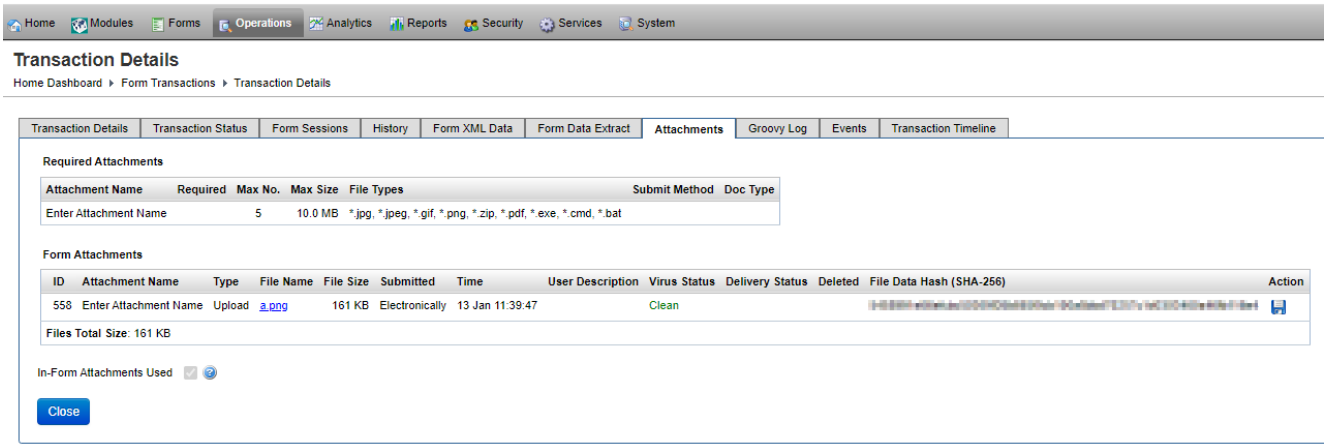
Attachments

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view form's attachments, if any, after a form user submits the form. You can use this information to check and ensure that attachments are appropriate for the form.

To view submitted attachments associated with a transaction:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View**.
3. Select the **Attachments** tab.



4. Check the **Required Attachments** section that shows the following information of all attachments required to successfully submit the form:
   - Attachment name
   - Whether it is required or not
   - Max number of attachments
   - Max size of attachments
   - File types of attachments
   - Submit method used
   - Doc Type
5. Check the **Form Attachments** section that shows the following information of all required and additional attachments that have been submitted with the form:

- ID of the attachment
- Attachment name
- Type of the attachment action
- File name of the attachment
- File size of the attachment
- How the attachment is submitted, for example Electronically.
- Time the document is attached to the form
- User browser description
- Virus check of the attachment status, such as Clean.
- Delivery status
- Whether it is delivered or not
- File data hash (SHA-256) uniquely identifying the attachment

6. Click **Download File** or a file name link to download the attachment on your local PC.

   You must be a Manager administrator to be able to download the form attachments.

7. Click **Close**.

Next, learn how to view saved transactions.

Transaction Groovy Log

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

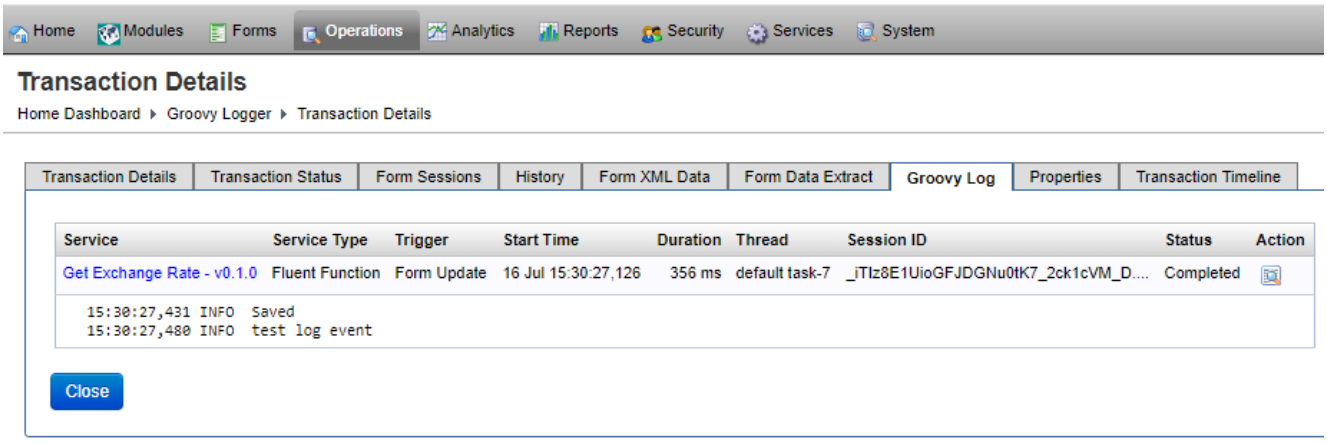
Manager allows you to view Groovy service output messages for a selected transaction, which you can use for troubleshooting purposes.

It's not recommended to log **PII**[1] data to the service log, because it is not secure development practice.

To view transaction's Groovy log:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View**.
3. Select the **Groovy Log** tab to see the log details.

This tab is only available if there are log details pertaining to the selected transaction.

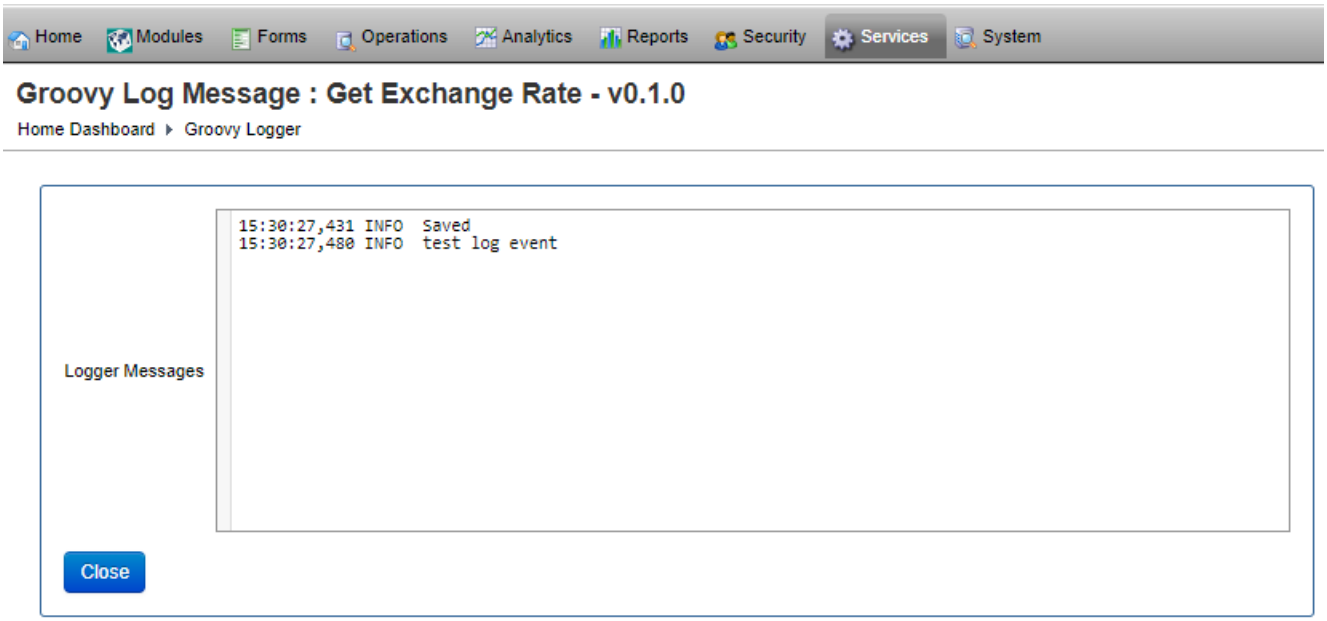


4. Click a service name to view the service configuration.

Select the service's **Groovy Service Log** tab to view the same logged messages.

5. Click **Groovy Logger** to see the detailed messages generated during the service invocation.

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.



6. Click **Close**.

Next, learn about the transaction details.

Transaction Events

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to see all events that were logged against a selected transaction.

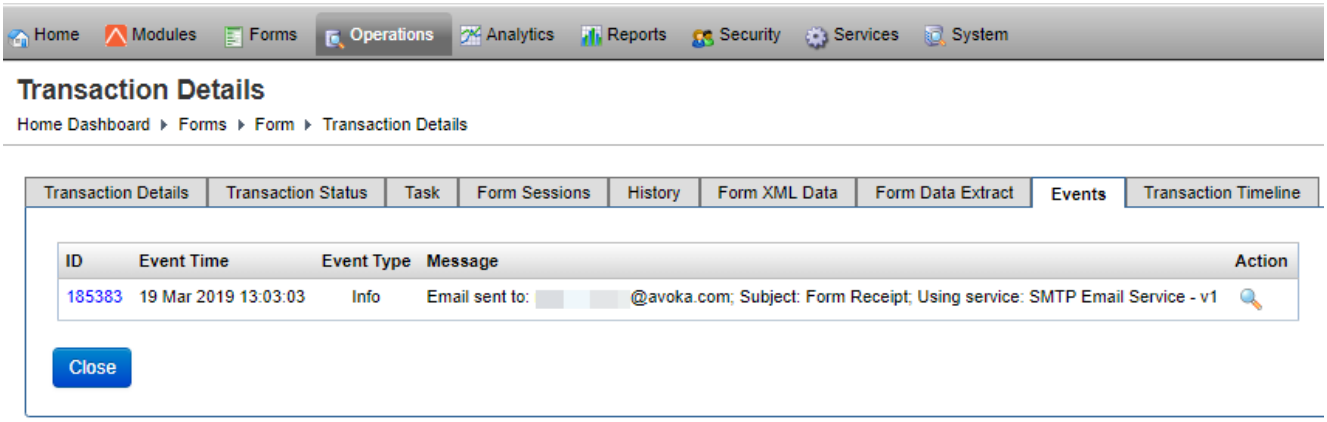
Events include:

- Error transactions
- Saved transactions

To view all events pertaining to a transaction:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View**.
3. Select the **Events** tab to see all events pertaining to the selected transaction.

> This tab is only available if events have occurred within the selected transaction.



4. Click **View** to see the event's details.
5. Click **Close**.

Next, learn how to track transactions.

Transaction Properties

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view and create transaction properties, which represent arbitrary data as string objects. The main persistence model in Manager is an **XML**[1] file, which is stored on the server and sent to a browser when a form is rendered. It's then sent back to Manager after the form is submitted. It is also saved to the server in background save events.

There are situations when it is useful to store some information, which is not part of the XML file, on the server only. For example, you may want to record a user's actual credit score, but don't show it to the user. In this case, you can store results from the backend server calls and send only a pass or fail result to the form, so the form users won't see their actual score, or the reasons why they failed. You can also temporarily store in-progress data that has been used in the backend service calls to ensure that the form users don't "game" the system by changing the data in the form at different points of their journey.

Manager comes with the Unified Application Data Document Model that allows you to store integration data separately from a form data. This means that data, which should not be visible or accessible in a browser, is not stored in the **AvokaSmartform** tag in the XML. Instead, data is stored in the **server xml** node which remains accessible only on server-side.To achieve this, you need to enable the **Unified App Data** feature for a form version, which you can easily do while configuring the form version.

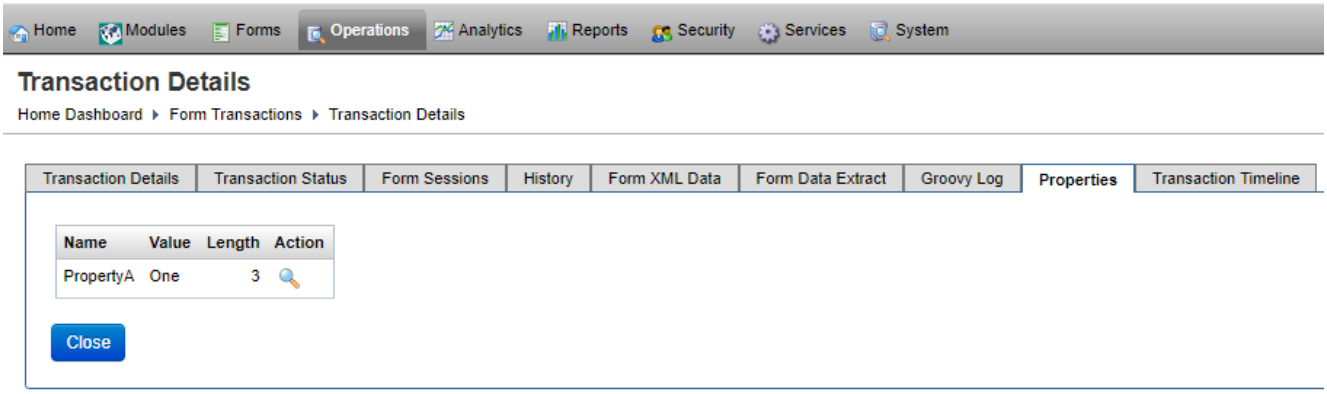
> It's strongly recommended to store integration data using the Unified Application Data Document Model.

To view transaction's properties:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View**.
3. Select the **Properties** tab to see all properties pertaining to the selected transaction.

> This tab is only available if there are properties associated with the selected transaction.

[1] XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.



4. Select a property and click **View** to see the property's value in a new browser tab.
5. Click **Close**.

Sometimes, you may want to add a new property to a transaction so it will be stored together. To add and store a transaction property, you should use the following Fluent Groovy API:

```
package exchange;

import com.avoka.tm.func.*
import com.avoka.tm.svc.*
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import javax.servlet.http.*
import com.avoka.tm.query.*

public class GetExchangeRate {

	// Injected at runtime
	public Logger logger

	/*
	* Perform Fluent Function call.
	*
	* returns: FuncResult
	*/
	FuncResult invoke(FuncParam param) {

		logger.info(param.txn.formStatus)

		new TxnUpdater(param.txn)
			.setProperty("PropertyA", "One")
			.update()

		FormFuncResult result = new FormFuncResult()

		logger.info("test log event")
		logger.debug("Debug log event")

		return result
	}
}
```

To retrieve transaction properties, you can use the Fluent Groovy code:

```
Txn txn = new TxnQuery()
	.setSubmitKey(…) // or similar to find the correct transaction
	.withPropertyMap() // to retrieve the properties
	.firstValue() //to find the first matching transaction
// The txn contains an attribute propertyMap of type Map<String,String>
```

When working with transaction properties, you should take into account the following:

- Data stored in the Transaction Properties are encrypted and subject to data retention policies, just like XML.
- In some cases, it may be necessary to store data against a collaboration job rather than an individual transaction.
- The PropertyQuery service is used for locating configuration properties for form versions, Workspaces, and so on. This is for static configuration properties, not for per-transaction or runtime properties.
- It is also possible to use the standard HTTP session to store temporary data. However, this data will not be persisted beyond an individual session. A transaction property is persisted in the database.

Next, learn about the transaction timeline.

Transaction Timeline

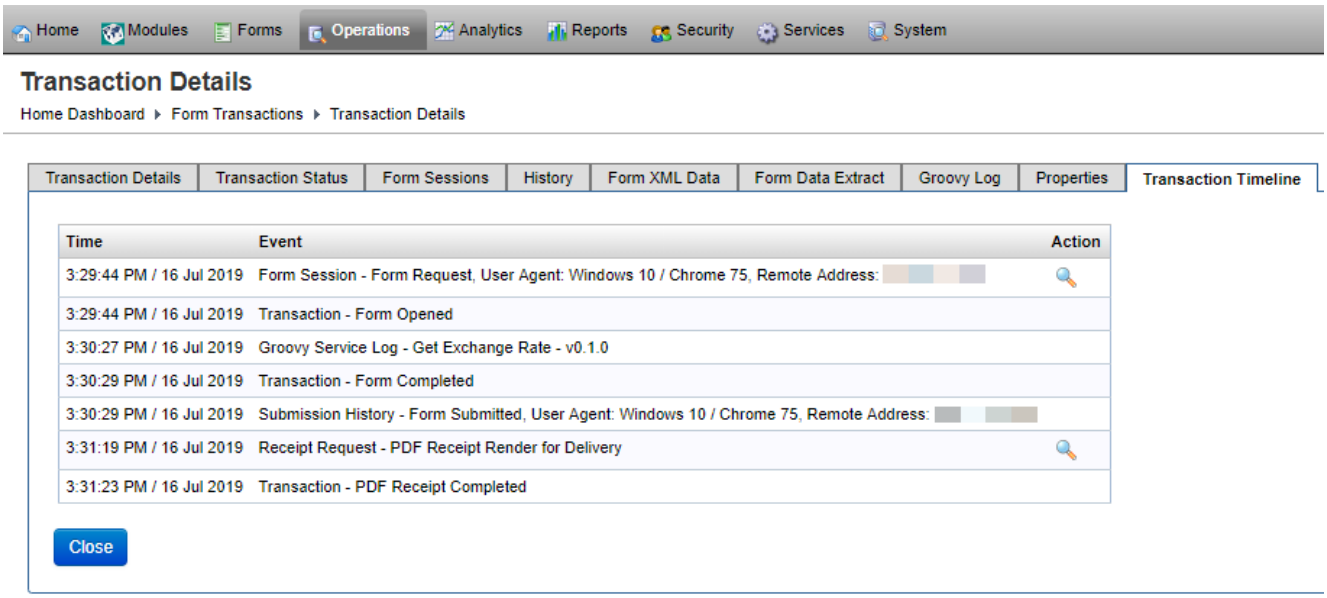
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view transaction timeline listing the important events that happened to a selected transaction. A transaction timeline is not a transaction audit trail, meaning that some of the entries derived from timestamps of the transaction, and, over time, some of the values may change to represent the latest state of the transaction.

To view transaction's timeline:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View**.
3. Select the **Transaction Timeline** tab to see timeline of the selected transaction.

> This tab is only available if there are events pertaining to the selected transaction.



4. Click **View** to see the transaction's details.
5. Click **Close**.

Next, learn about the transaction details.

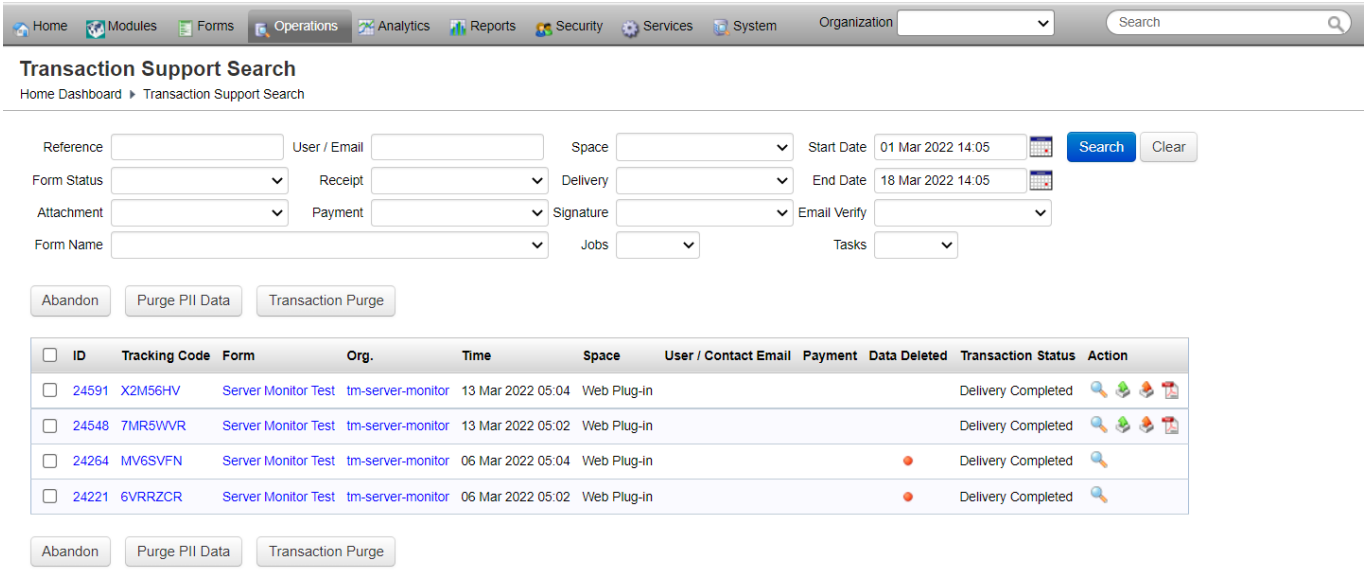
Transaction Support Search

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager allows you to search transactions based on wide range of criteria. This is useful when troubleshooting transactions.

To navigate to transaction support search, select **Operations > Txn Support Search**.

Transactions are shown page by page, with the maximum number of records per page configured via the user's preferences.



The transaction list displays the following details for each transaction:

- Transaction ID
- Tracking Code
- Name of the form
- Organization associated with the form
- Time and date of a transaction, which can be Form Opened Time, Form Saved Time, Form Submitted Time, Last User Activity Time, Form Completed Time. This time is stored in **`submission.time_request`**.
- Space
- User/ Contact Email
- Payment
- Data Deleted - indicated with a red dot if the transaction data, including **PII**[1], was removed from Manager
- Transaction Status

To filter or search the transactions by one or more criteria, specify the following settings and click **Search**:

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

- Reference: a transaction ID or tracking code.
- Form status.

  A form status can be one of the following:

  - Abandoned - transactions that have been abandoned by the user, or based on the data retention policy.
  - Assigned - a task which is waiting to be completed by a user.
  - Completed - transactions that have been successfully submitted and contain all required data.
  - Opened - transactions that have been started by a user. When a transaction is created by a user (by starting a new form), the initial form status is set to opened. This status will be changed once the user saves, cancels, or submits the transaction.
  - Saved - transactions that have been saved by the user or automatically by the system.
  - Submitted - transactions that have been submitted but may require additional information (attachments, payment, etc.). Once the required data is collected, the transaction status will change to Completed. This status is not common for Maestro forms as all information, such as attachments and payments, are collected prior to the user submitting the form. Most Maestro forms would skip the Submitted status and be marked as Completed after the form is submitted by the user. This is a legacy feature.
  - Expired - a form used by this transaction has expired and is no longer available.
- Attachments.

  An attachment type can be one of the following:

  - Completed.
  - Optional.
  - Required.
- Form Name - select a form name from the dropdown list.
- User / Email - provide a user name or an email address.
- Receipt.

  A receipt status can be one of the following:

  - Ready - a receipt is waiting to be allocated to a receipt generator.
  - In Progress - a receipt is in a receipt generator's queue.
  - Completed - a receipt has been generated.
  - Error - there is an error during receipt generation. A retry may fix it.
  - Error - No Data - there is no XML data available to generate a receipt.
- Payment.

  A payment type can be one of the following:

- Required.
- Completed.
- Error.
- Pending.

- Space.
- Delivery.

  A delivery status can be one of the following:

  - Completed - a transaction has been successfully delivered, based on the configured delivery channel.
  - Undeliverable - a delivery is not possible. Check the error logs for further details of the error.
  - Not Ready - a transaction is not ready for delivery yet. This is the start state before a transaction has been completed by the user. Generally, the receipt render service will kick the transaction from this state to the Ready state when the receipt is rendered.
  - Not Required - a delivery is not required. It may be that this form is part of a collaboration job and the delivery is to be handled separately say as a consolidation of the transaction details from all the associated forms.
  - Ready - a transaction is ready to be delivered. A transaction will be marked as Ready until the delivery channel is processed, or it may remain marked as Ready if a delivery channel is not configured.
  - In Progress - a delivery has been started but is waiting to be completed.
  - Paused - a transaction delivery is paused. This transaction's status can be changed by some other means, for example by a Groovy Scheduled Task.
  - Error - a transaction delivery has failed and is waiting to be retried.
  - Sent Email - a secure email has been sent and now a transaction is waiting for the user to log in and process it.
  - Pending

- Signature.

  A signature type can be one of the following:

  - Required.
  - Completed.
  - Completed - Deliver Manually.
  - Completed - Upload Scanned Copy.

- Jobs.

  A job type can be one of the following:

  - Include - transactions associated with a job are included.
  - Exclude - transactions associated with a job are excluded.

- Start Date - the date and time you want to search from. Manager uses timestamps of several distinct transaction's life cycle states to search for transactions, for example, Form Opened Time.
- End Date - the date and time you want to search until. Manager uses timestamps of several distinct transaction's life cycle states to search for transactions, for example, Form Opened Time.

  The time interval is inclusive.

- Email Verify.

  An email verification type can be one of the following:

  - Required.
  - Pending.
  - Completed.
  - Abandoned.
  - Error.
  - Error - No Data.
- Tasks - to filter transactions related to tasks.

  A task type can be one of the following:

  - Include - transactions associated with a task are included.
  - Exclude - transactions associated with a task are excluded.

Click **Clear** to reset the search criteria.

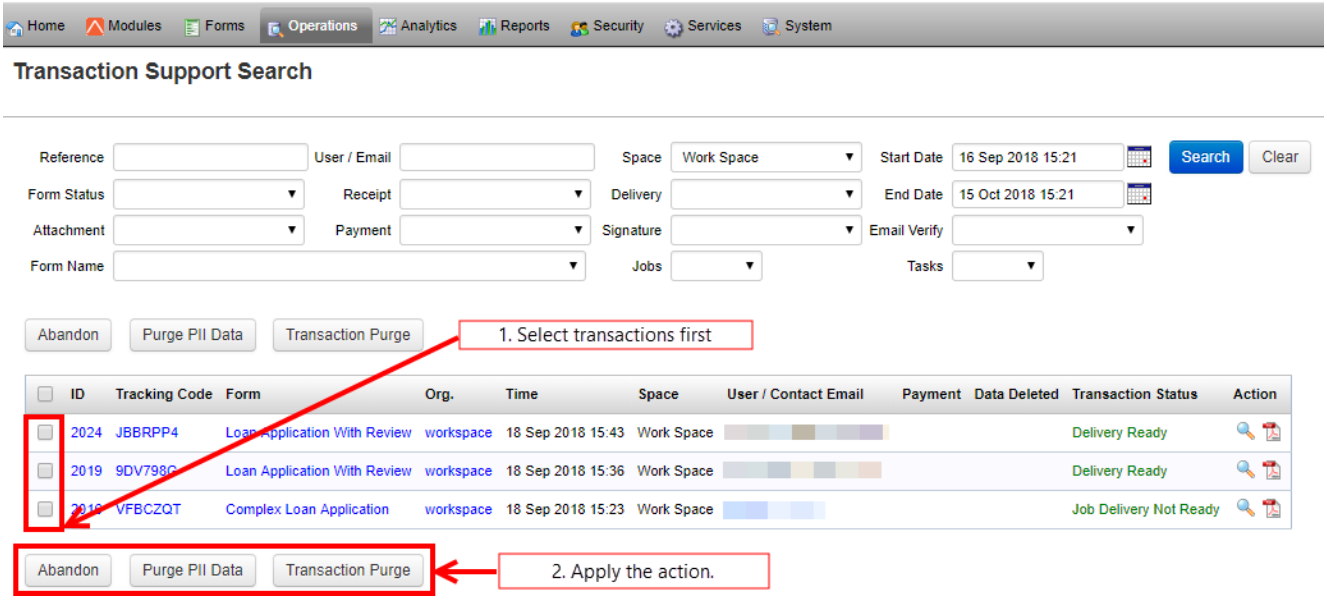
You can perform one of the following actions for each selected transaction:

- Click **View** to view transaction details.
- Click **View receipt** to view the transaction PDF receipt.
- Click **Open Saved Form** to open the saved form to allow an administrator with the **Submission Save Edit** permission to complete the form on behalf of the user.

You can perform several bulk operations on multiple selected transactions or a group of transactions. | 18.11

To perform a bulk operation:

1. Select or filter transactions you need, as described above.



2. Select all these transaction ID checkboxes or select the header checkbox to bulk select all filtered transactions.
3. Apply one of the following actions:
   - Click **Abandon** to abandon the selected transactions.
   - Click **Purge PII Data** to purge only **PII**[1] data of the selected transactions. Active transactions will be abandoned, if they are not already abandoned.
   - Click **Transaction Purge** to purge the selected transactions immediately. Active transactions will be abandoned, if they are not already abandoned.
4. Click **OK** to accept the warning message that the transactions and all the data will be purged.

> Transactions relating to collaboration jobs are rejected and this is recorded in the event log entry. The user is warned this has happened.

5. Check the event log entry to see if the purging was successful. An example of a successful message is: **`Submission 3090 was marked for immediate PII data purging by user john.-smith@email.com`**.

> You must be an administrator with Submission Data Purge or Submission Edit privileges to perform these bulk operations.

---

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

You can purge each transaction separately from the Transaction Status tab. | 18.11

The System Event Publisher publishes submission PII purging events. | 18.11

Next, learn how to view form transactions.

Track Transactions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

One of the key features of Manager is to keep track of all transactions made by form users. Manager captures this information in form transactions, so operations staff can search for a particular transaction and view its details.

You can also use the following Manager functionality to track user transactions:

- View abandoned transactions
- View form submission data
- View saved transactions
- View submitted transactions

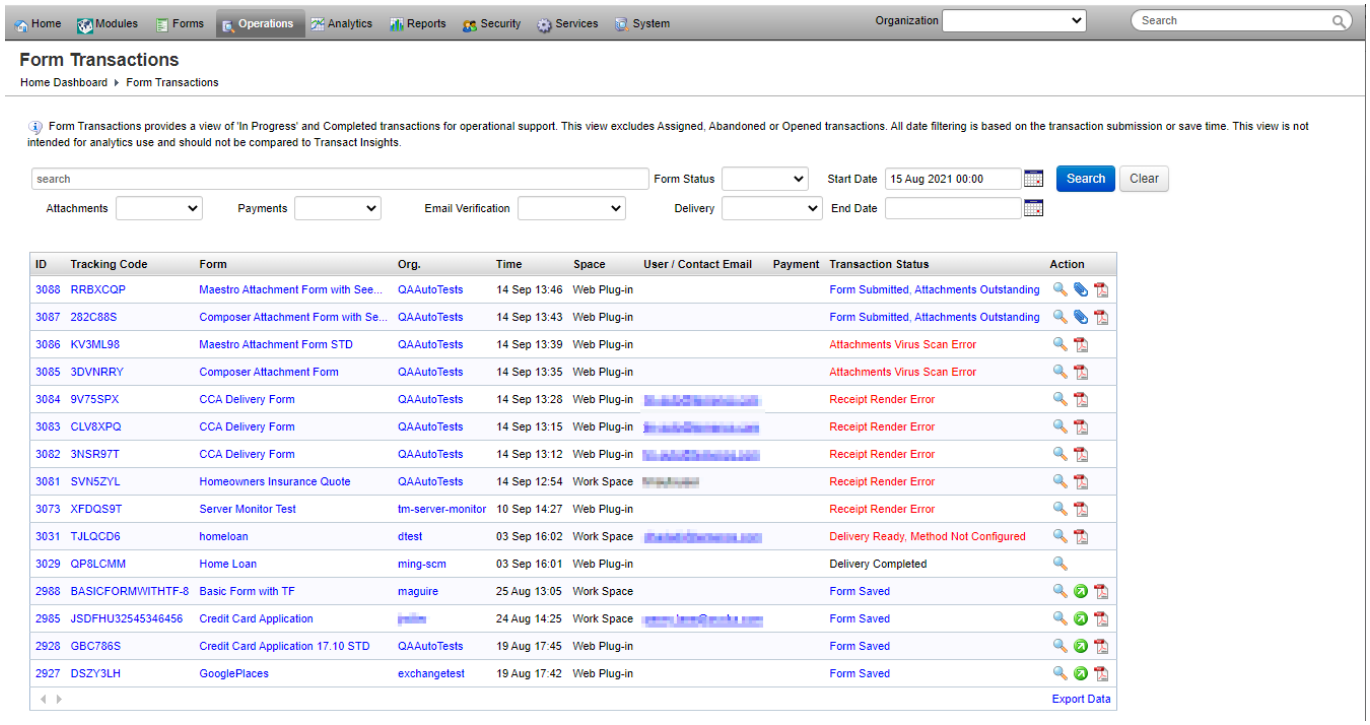
View Form Transactions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager keeps track of all transactions, representing form submissions made by application form users.

To view form transactions, select **Operations > Form Transactions**.

Transactions are shown page by page, with the maximum number of records per page configured via the user's preferences.



To filter or search the transactions by one or more criteria, specify the following settings and click **Search**:

- Transaction's name or part of a name
- Form status:
  - Saved
  - Submitted - If transactions have post submission processing or elaborated delivery methods, they can remain in a submitted state. You can use this status to locate submissions, which are potentially having problems in delivery and so on, so they have not moved to completed state.
  - Completed
- Attachments
- Payments
- Email Verification
- Delivery
- Start Date
- End Date

Click **Clear** to reset the search criteria.

The key column is the Transaction Status, which indicates the status of the selected transaction and various actions associated with it.

You can perform the following actions:

- Click **View** to view transaction details.
- Click **View receipt** to view the PDF receipt of the selected transaction.
- Click **Retry delivery** to retry delivery of the transaction. This is only available if delivery of a transaction has not been completed, but submission data is still available.
- Click **Retry delivery from scratch** to attempts to deliver the submission using the current delivery details set up in the form after deleting all existing delivery checkpoints for the submission. If no delivery checkpoints exist, this is equivalent to the retry delivery option above.This is only available if delivery of a transaction has not been completed, but submission data is still available.
- Click **Open Saved Form** to open the saved form to allow an administrator to complete the form on behalf of the user. As this is a sensitive operation, this action is only available to administrators whose roles include the permission, Submission Save Edit.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to view form requests.

Form Data Extract

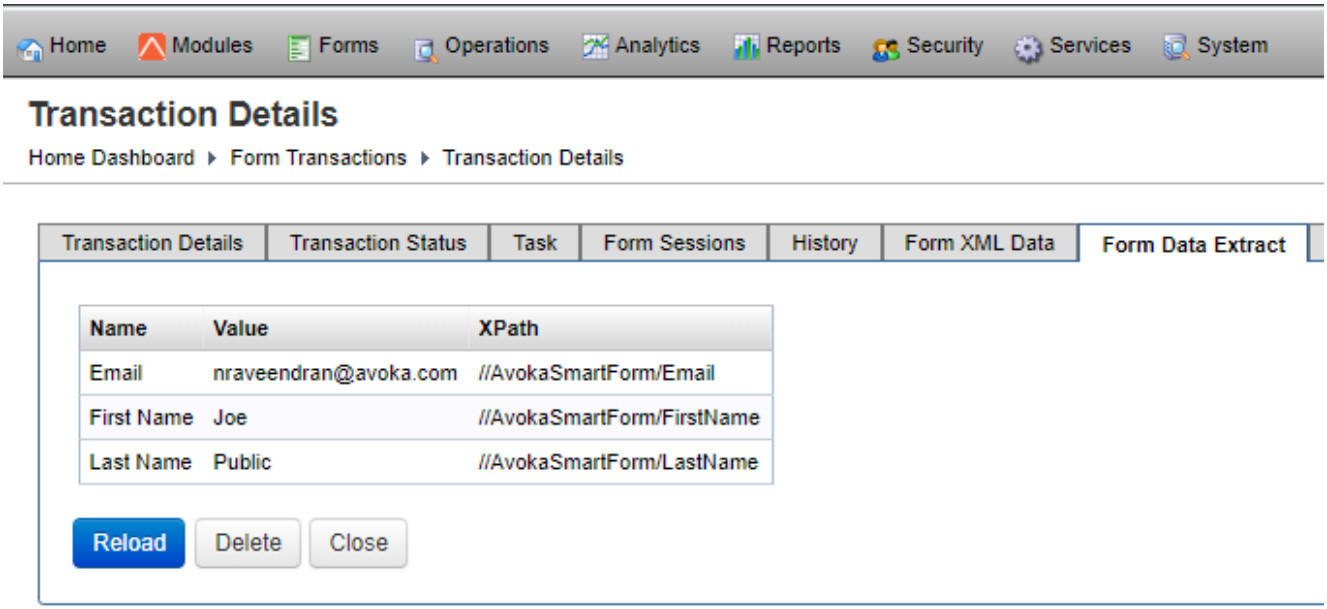
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view all items and their values that were extracted from the submitted XML data of a form.

> The **Form Data Extract** tab is only available if form data extracts are configured.

To view a form data extract:

1. Select **Operations > Form Transactions**.
2. Locate a transaction and click **View**.
3. Select the **Form Data Extract** tab.



   Manager extracts these values and places them in the **Value** column for each of the configured data extracts.

4. Click **Reload** to refresh form XML data extract values.
5. Click **Delete** to remove the data extracts.
6. Click **Close**.

For more information, see view form submission data extract.

Next, learn how to view form submission data.

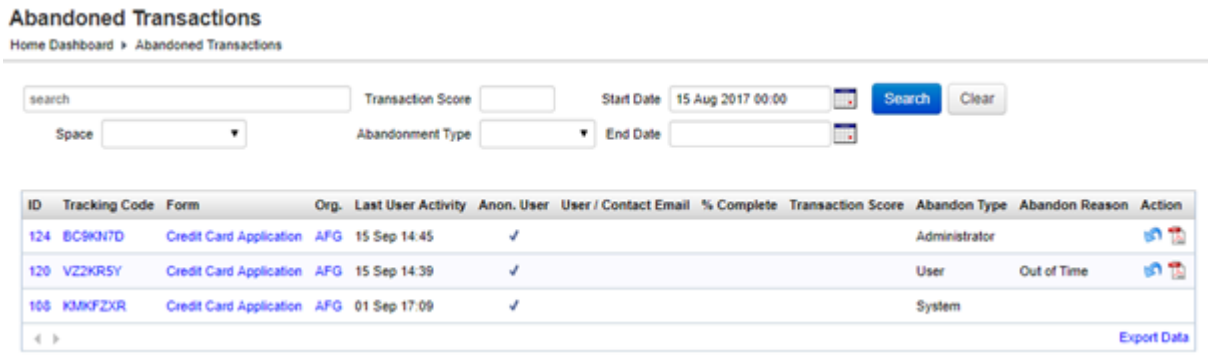
View Abandoned Transactions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view abandoned transactions so you can inspect them and make some corrective actions, for example, change their status.

> Only a Manager administrator can change an abandoned transaction status.

To view the abandoned transactions and submissions, select **Operations > Abandoned Transactions**.



The list displays the following details for each transaction:

- The Transaction ID
- The Tracking Code
- The Name of the form
- The Organization associated with the form
- The time and date of the user's last activity with the transaction
- The name of the user and contact email.
- The identification if the form user is anonymous represented by a tick ( )
- The Abandonment Type
- The Abandonment reason (if known)

To search the abandoned transactions list by one or more criteria, specify the following settings and click **Search**:

- Spaces
- Transaction Score
- Abandonment Type: User, System, Administrator, or Ineligible.
- Transaction text, or part of a transaction text.
- Name
- Start Date and End Date.

Click **Clear** to reset the search criteria.

You can perform the following actions on a selected transaction:

- Click **Make Abandoned Transaction Saved** to revert the submission into a Saved status if submission data is available. Use this if an original submitter would like to complete the submission now.
- Click **View Receipt** to view the receipt as a PDF file.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

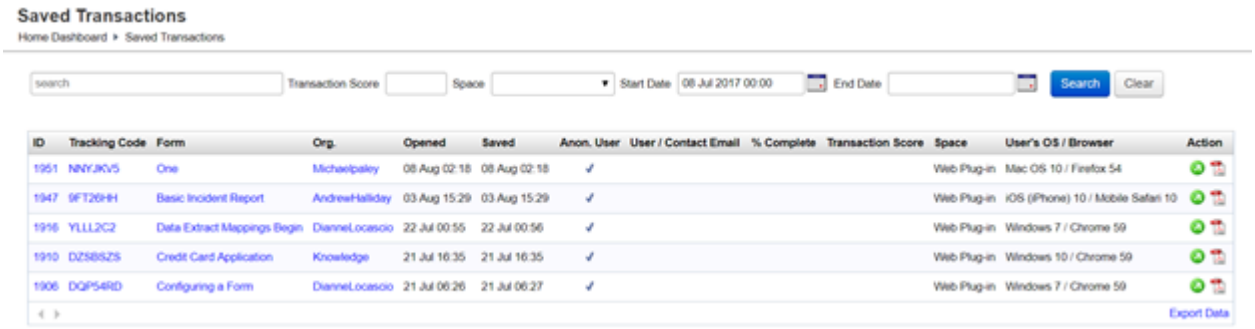
Next, learn how to configure form abandonment.

View Saved Transactions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows an administrator to view and update saved transactions.

To view the saved transactions, select **Operations > Saved Transactions**.



The transaction list displays the following details for each transaction:

- Transaction ID
- Tracking Code
- Name of the form
- Organization associated with the form
- Time and date the form was opened
- Time and date the form was saved
- Name of the user and contact email
- % Complete
- Transaction Score
- Space
- User's OS / Browser

To search the saved transactions by one or more criteria, specify the following settings and click **Search**:

- Transaction text, or part of a transaction text.
- Transaction Score.
- Space
- Start Date
- End Date

Click **Clear** to reset the search criteria.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

You can perform the following actions on a selected transaction:

- Click **Open Saved Form** to allow an Administrator, with Submission Save Edit permission, to complete the form on behalf of a user.
- Click **View Receipt** to view the receipt as a PDF file.

Next, learn how to view submitted transactions.

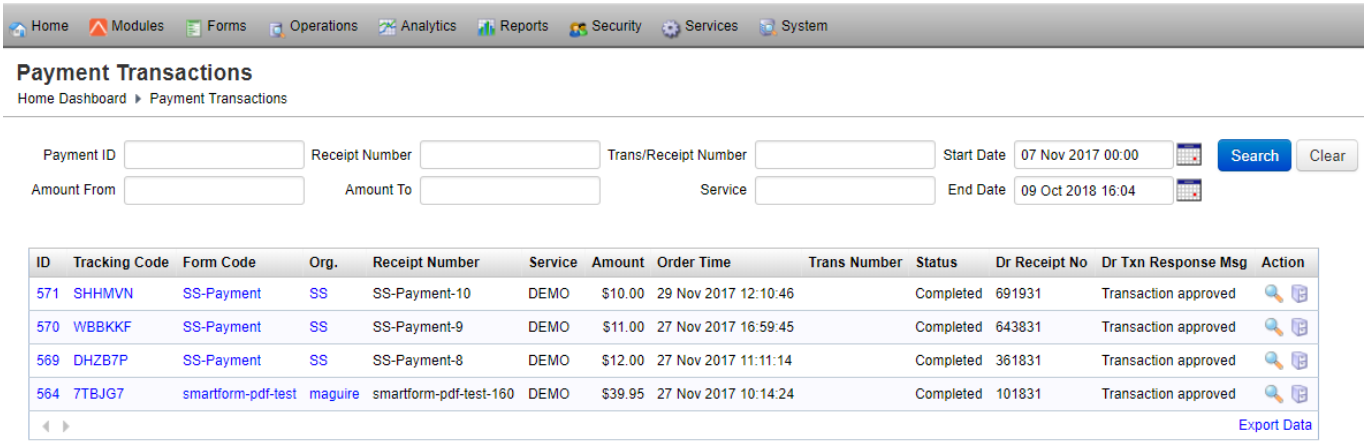
View Payment Transactions

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view payment transactions.

To view payment transactions, select **Operations > Payment Transactions**.

Transactions are shown page by page, with the maximum number of records per page configured via the user's preferences.



To filter or search the transactions by one or more criteria, specify the following settings and click **Search**:

- Payment ID
- Amount From
- Receipt Number
- Amount To
- Trans/Receipt Number
- Service
- Start Date
- End Date

Click **Clear** to reset the search criteria.

You can also perform the following actions:

- Click **View** to view transaction details.
- Click **View Submission** to view the submitted transaction.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to view form requests.

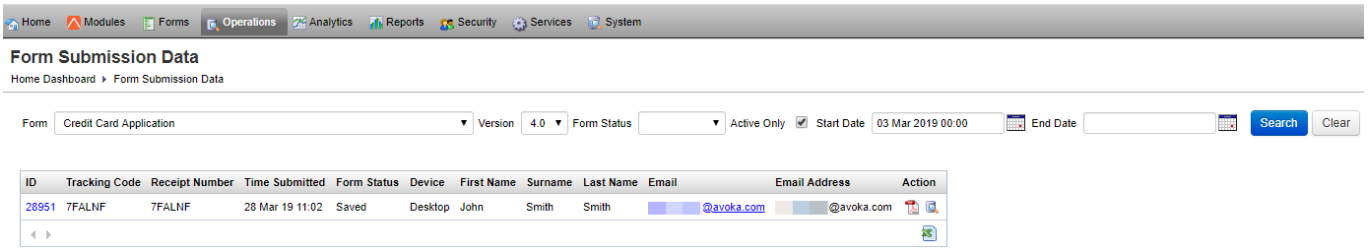
View Form Submission Data

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view form submission data extracts, which you can use to analyze and troubleshoot form submissions.

> Data extracts are purged frequently and may not be available by the time you perform a search.

To view form submission data, select **Operations > Form Submission Data**.



To search the form submission data by one or more criteria, specify the following settings and click **Search**:

- Form
- Version of a form
- Form Status: Saved, Submitted, Completed, and Abandoned
- Active Only
- Start Date and End Date

Click **Clear** to reset the search criteria.

Click **ID** or **View Submission Details** to see the transaction details with data extract information.

Click **View PDF Receipt** to view and download a selected form submission data as a PDF file.

> You can't export multiple forms as a PDF file.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

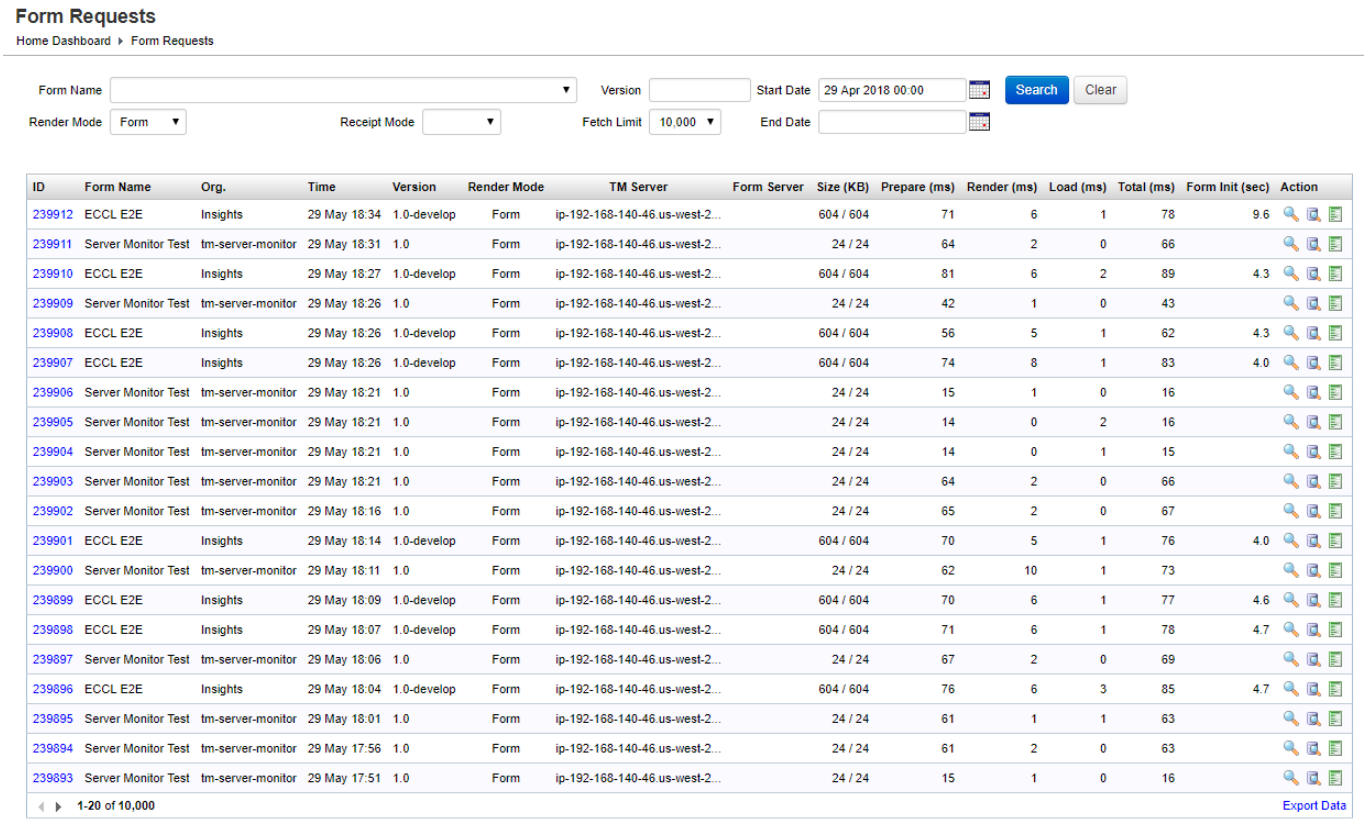
Next, learn how to view form transactions.

View Form Requests

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager allows you to view all form requests initiated by users. When a user selects a form, this action is logged as a form request, which can even include a generated receipt. You can use this information to troubleshoot transaction issues and analyze form request processing performance.

To view form requests, select **Operations > Form Requests**.



The form requests list displays the following details for each transaction:

- ID - the transaction request identifier.
- Form Name - the name of the form.
- Org - the organization associated with the form.
- Time - the date and time that the request was made.
- Version - a form version.
- Render Mode - refers to the task requested by the user, which either a form request or receipt request.
- TM Server - the host name of a Manager server.
- Size (KB) - the size of the request (in kilobytes).
- Prepare (ms) - the amount of time (in milliseconds) it took for the server to prepare the request.
- Render (ms) - the amount of time (in milliseconds) it took for the server to render the request.
- Load (ms) - the amount of time (in milliseconds) it took for the server to load the request.
- Total (ms) - the total amount of time (in milliseconds) it took from the request being made to the request being rendered to the user.
- Form Init (sec) - the amount of time (in seconds) it took to initialize the form. This is only relevant for forms using form functions.

To search the form requests by one or more criteria, specify the following settings and click **Search**:

- A form name.
- A form version.
- Render Mode – Form or Receipt.
- Receipt Mode

  A receipt modes can be one of the following:

- Web – the receipt was requested by a user using a web browser.
- Delivery – the receipt was generated during the submission delivery process.
- Email – a form user had requested an email copy of the receipt.
- Fetch limit to constrain the number of records to select and display. | 18.05

> This page is automatically limited to display 10,000 log entries to avoid performance issues. If you wish for more or fewer log entries to be displayed, select a number from the **Fetch Limit** dropdown list.

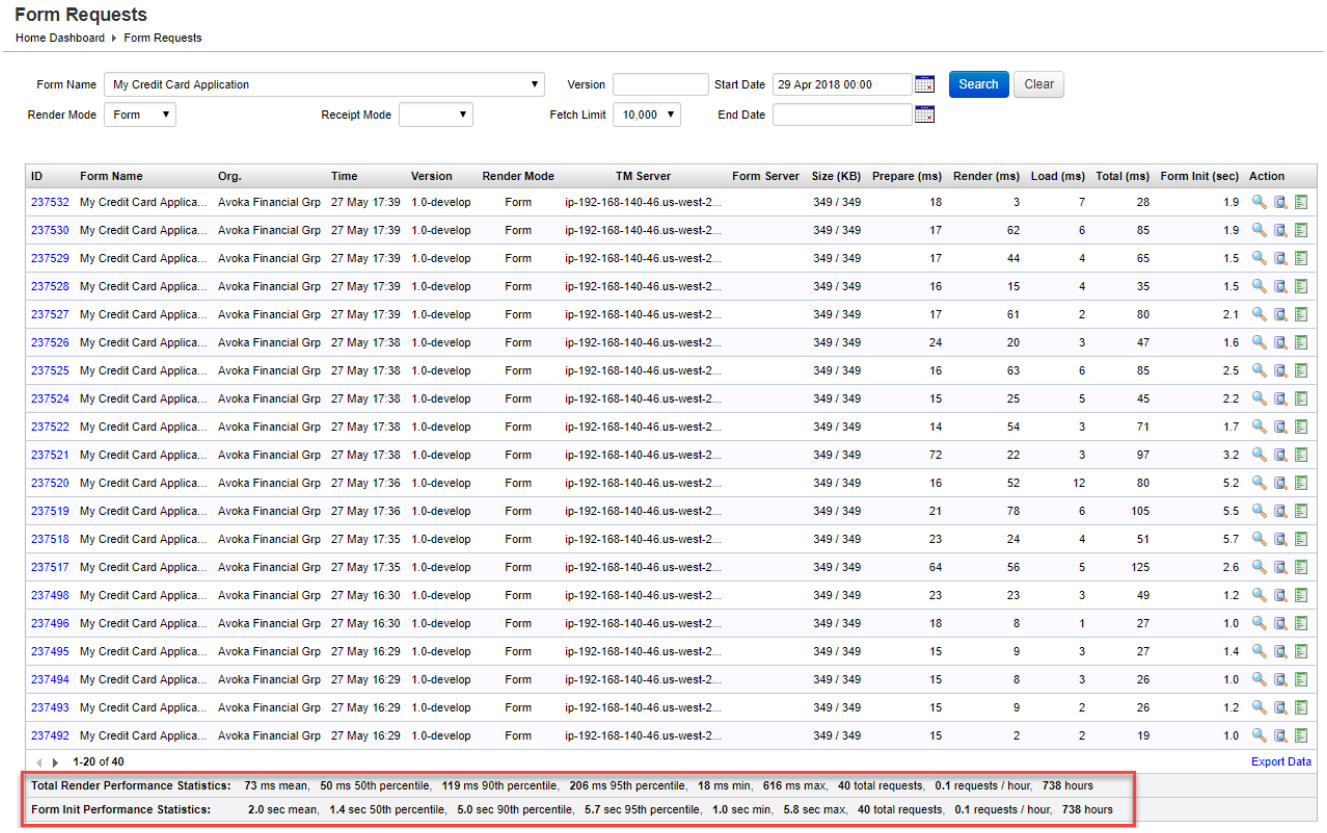
- Start Date and End Date – By default, this is set to a date range starting one month before, up to and including today.

To view a form performance statistics summary for each form:

1. Select the **Show Performance Stats** checkbox in system preferences
2. Select the form from the **Form Name** dropdown list.

3. Click **Search** to generate the following statistics. | 18.05

- mean
- 50th percentile
- 90th percentile
- 95th percentile
- minimum time
- maximum time
- total requests
- number of requests per hour



You can also perform the following actions:

- Click **View** to see the additional details of a selected form request.
- Click **View Submission Details** to see the transaction details related to a form submission.
- Click **View Form Details** to go to the dashboard of a form related to this form request.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

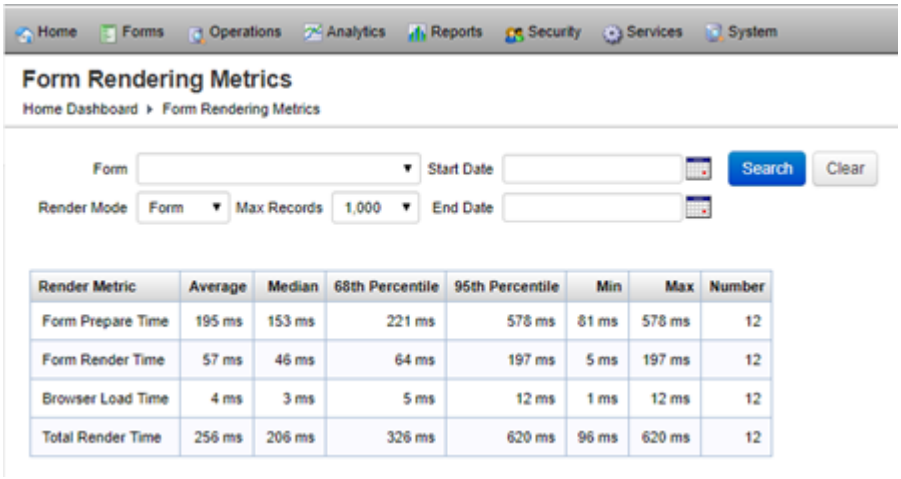
Next, learn how to view form sessions.

View Form Rendering Metrics

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to see how fast forms are rendered to users, so you can analyze their rendering performance. The metrics include the total render time (in milliseconds) and its components for all form requests.

To view the form rendering metrics, select **Operations > Form Rendering Metrics**:



Render Metric	Average	Median	68th Percentile	95th Percentile	Min	Max	Number
Form Prepare Time	195 ms	153 ms	221 ms	578 ms	81 ms	578 ms	12
Form Render Time	57 ms	46 ms	64 ms	197 ms	5 ms	197 ms	12
Browser Load Time	4 ms	3 ms	5 ms	12 ms	1 ms	12 ms	12
Total Render Time	256 ms	206 ms	326 ms	620 ms	96 ms	620 ms	12

The form render metrics display the following information.

- **Form Prepare Time** – the amount of time it took to prepare the render request.
- **Form Render Time** – the amount of time it took to render the form request on to the server.
- **Browser Load Time** – the time it took to stream the rendered form to the user's browser.
- **Total Render Time** - the total time it took from the moment the request was made to when the form was displayed in the browser. This time is the sum of the form prepare time, form render time, and browser load time.

Each of the render metrics are broken down into the following groups:

- Average
- Median
- 68th Percentile
- 95th Percentile
- Min
- Max
- Number

To filter or search the form rendering metrics by one or more criteria, specify the following settings and click **Search**:

- Form
- Render Mode: form, receipt
- Max Records
- Start Date and End Date

Click **Clear** to reset the search criteria.

> If you select a large number of records in the **Max Records** field, the metrics calculation may take longer.

Next, learn how to view form requests.

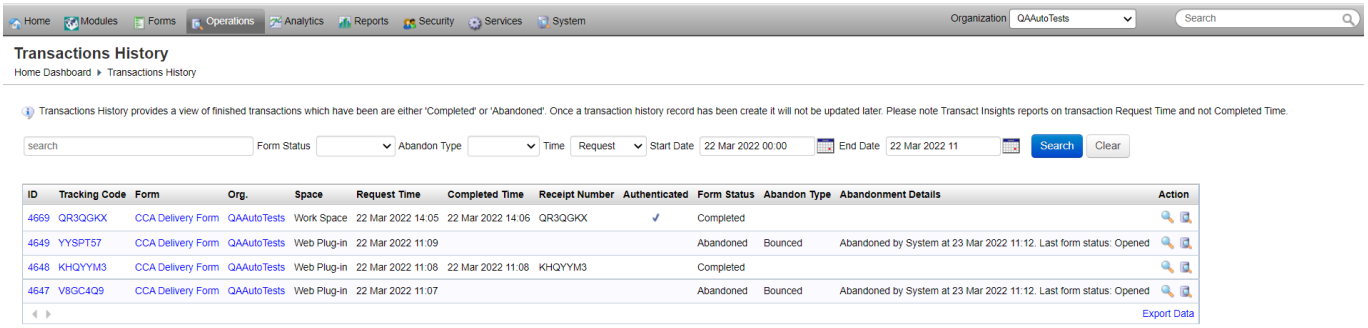
View Transactions History

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view transactions history of all submissions made by form users. It provides a view of finished transactions which have been are either Completed or Abandoned. Once a transaction history record has been created, it will not be updated later. Journey Analytics reports on transaction Request Time and not Completed Time.

To view transactions history, select **Operations > Transactions History**.

Transactions are shown page by page, with the maximum number of records per page configured via the user's preferences.



To filter or search the transactions by one or more criteria, specify the following settings and click **Search**:

- Transaction's name or part of a name
- Form status can be one of the following:
  - Completed
  - Abandoned
- Abandon Type can be one of the following:
  - Bounced - a transaction is bounced
  - Started - a transaction is started
  - Saved - a transaction is saved
  - Cancelled - a transaction is canceled
  - Ineligible - a transaction is ineligible
  - Submitted - a transaction is submitted
- Time can be one of the following:
  - Request
  - Completed
- Start Date is the date and time of a transaction
- End Date is the date and time of a transaction

Click **Clear** to reset the search criteria.

You can perform the following actions:

- Click **View** to the transaction history or transaction history details.
- Click **View Submission Details** to view transaction's details.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to view form requests.

Debugging Maestro Forms with Chrome DevTools

Maestro | Form Builder | 5.1 & Higher

If a Maestro form isn't behaving as expected, it might be related to one of the business rules you've created using the Maestro script editor. Tracking down problems in business rules can be tricky; this topic explains some useful techniques for finding and fixing problems using Google's Chrome DevTools.

> This topic only scratches the surface of what Chrome DevTools can do. To learn more, read Google's official documentation.

Prerequisites

This topic assumes you have experience with HTML, CSS, and JavaScript.

Maestro runs on the AngularJS JavaScript framework, so AngularJS experience will be useful if you'll be debugging on a regular basis, but is not required for this guide. To learn about AngularJS, read the official tutorial.

We recommend Google Chrome for debugging Maestro forms. It has a wide range of debugging tools and it's what we'll be using in this topic.

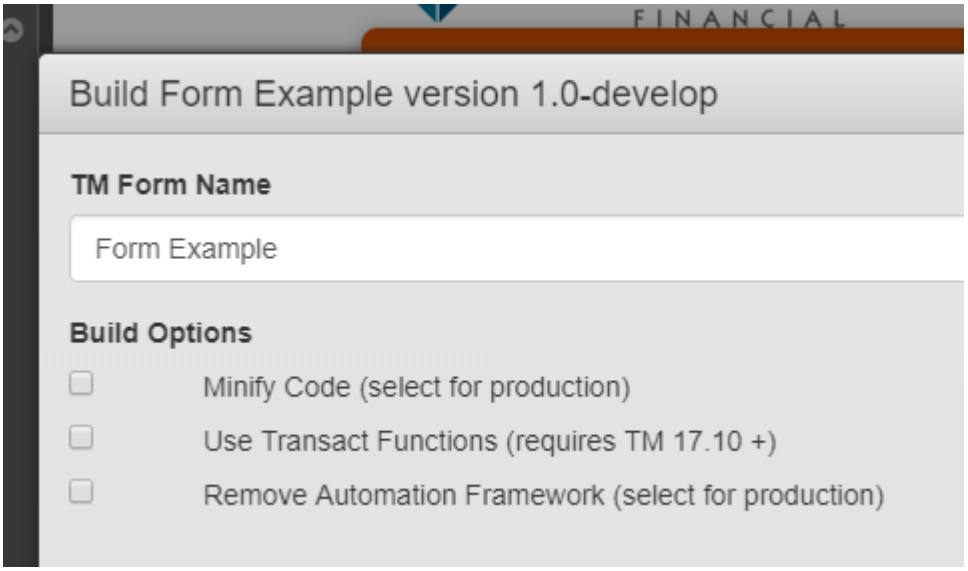
Building a Form

Before a Maestro form can be debugged, it needs to be built with some specific build options and then rendered.

> You cannot debug a Maestro form via the **Design**, **Code**, or **Preview** tabs.

To prepare a Maestro form for debugging:

1. In the Maestro Editor, click **Build > Build with Options** to display the **Build** modal.
2. Ensure the **Minify Code** checkbox is cleared. Selecting this option minifies the JavaScript in the rendered form, which compresses the code by removing unnecessary characters such as white space, new lines, and comments. Minification is great for performance but interferes with debugging.
3. Ensure the **Remove Automation Framework** checkbox is cleared. Selecting this option removes the Maestro Automation Framework (MAF), which is used to test Maestro forms with tools like Selenium. However, the MAF also has some features useful for debugging, so it's best if we don't remove it until the form is ready to go live.



4. Click **Build**.
5. Click **Render Form**.

The form will open in a new browser tab. If a new browser tab doesn't open, check if your browser is blocking popups and allow them from the Maestro URL.

Whenever you make a change to the form, you'll need to rebuild it. If you're doing this a lot, you can streamline the process by using some keyboard shortcuts: `Control+P` (`Command+P` on Mac) to open the **Build** modal and `Control+Enter` (`Command+Enter`) to build the form.

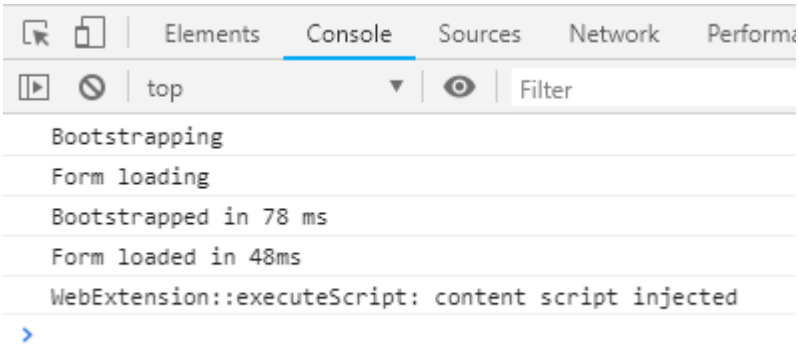
Opening Chrome DevTools

Chrome DevTools is a set of web developer tools built into the Google Chrome browser. These tools provide access to a range of features for diving into the look, functionality, and performance of web applications.

To open Chrome DevTools:

1. Select the browser tab that contains the rendered Maestro form.
2. Press `F12` or `Control+Shift+I`.

The DevTools pane will appear on the right or bottom edge of the browser window.



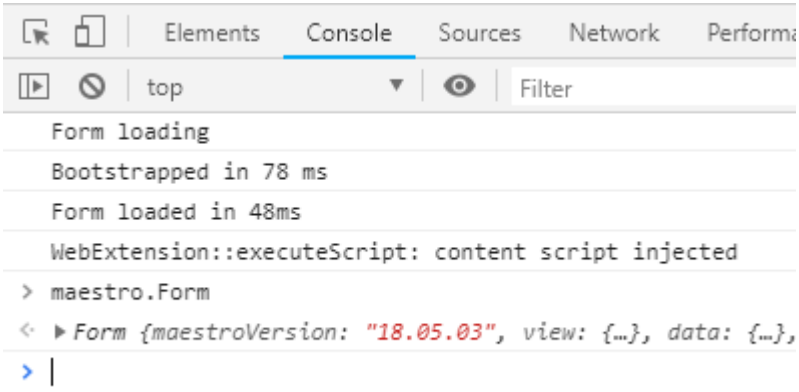
Inspecting the Form Object

When you load a Maestro form in your browser, a Form object is initialized that contains the data required to render the form. It's worth understanding the Form object so you know where to look when something goes wrong.

You can inspect the Form object in Chrome DevTools. To do this, open the **Console** tab and enter:

```
maestro.Form
```

This statement is available to us through the Maestro Automation Framework that we chose not to remove when building the form. When we execute this statement, the Form object is returned:



You can also access this data via jQuery:

```
$("Form").scope().Form
```

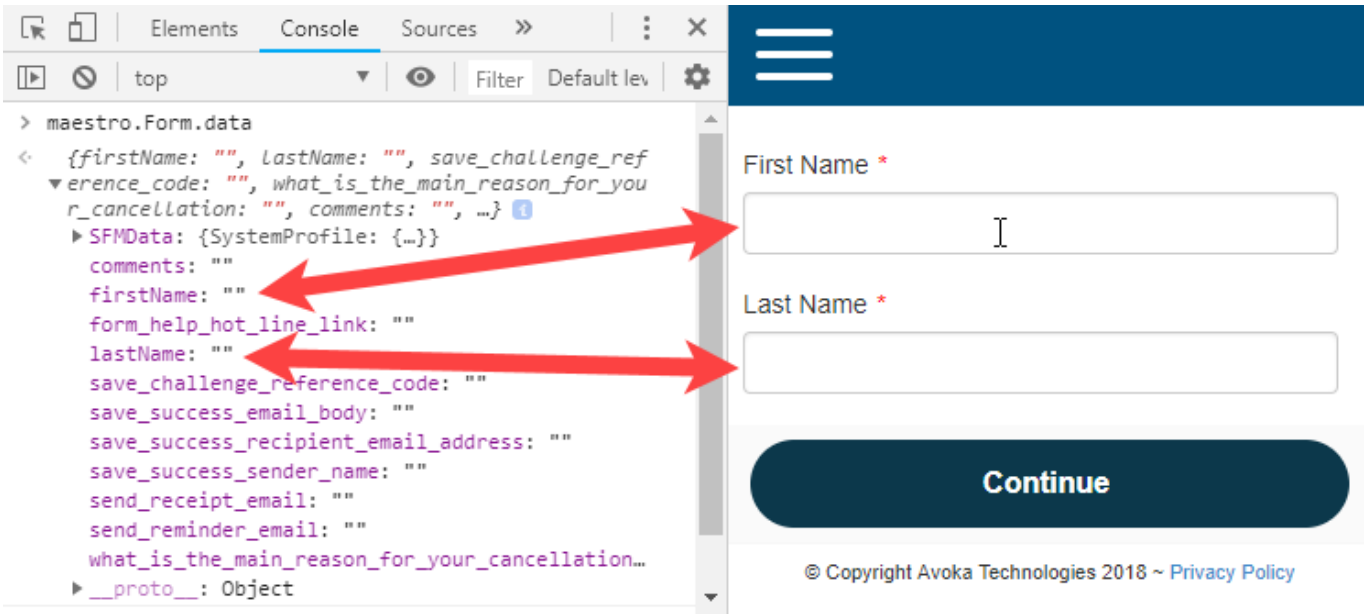
If you expand the Form object, you'll see that it's quite large. You don't, however, need to understand every part of this object. It's just useful to have a broad understanding of how this object describes the structure of a Maestro form.

There are three objects within the Form object that are worth exploring: **data**, **items**, and **view**.

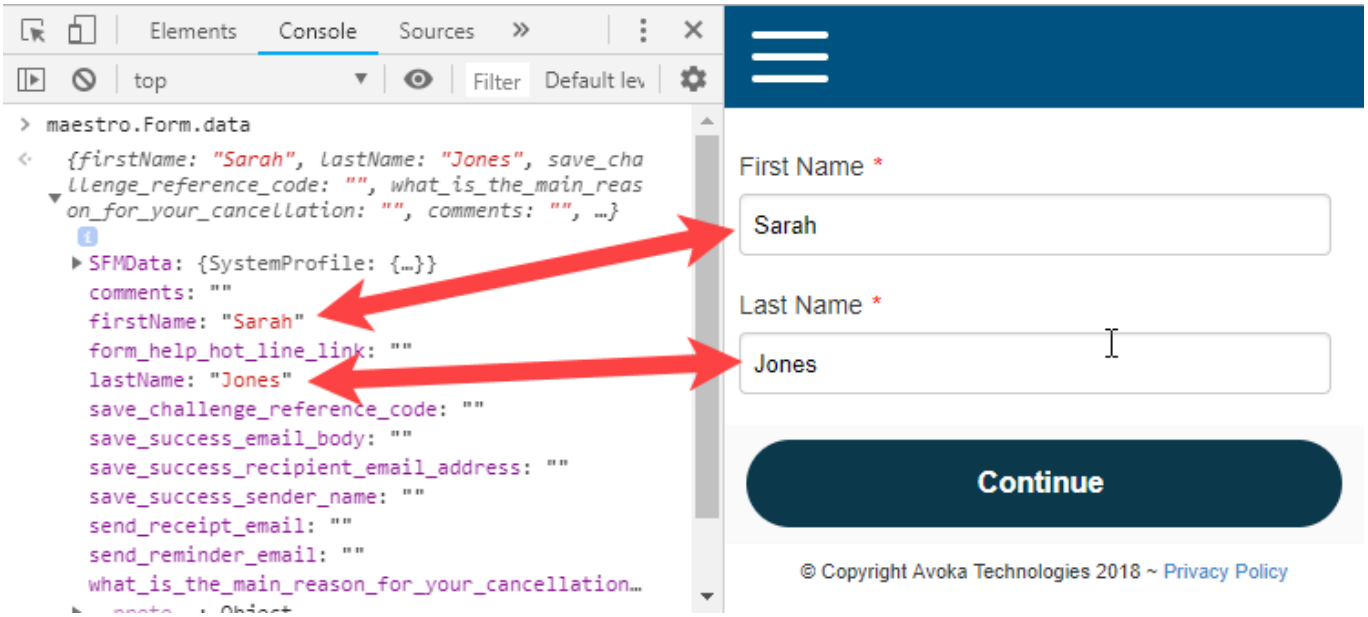
data

The **data** object describes the current state of the form's data. When values are entered in to the fields of a form, the **data** object is updated to reflect the new values.

For example, if you have an empty form, there won't be any values in the data object's properties:



But if you enter some values into the form, maestro.Form.data will contain those values:



items

The **items** object describes the components (such as fields and buttons) and containers (such as blocks and sections) within a form. Each component and container is considered an item.

For example, this is what the **item** object for the "First Name" text field looks like:

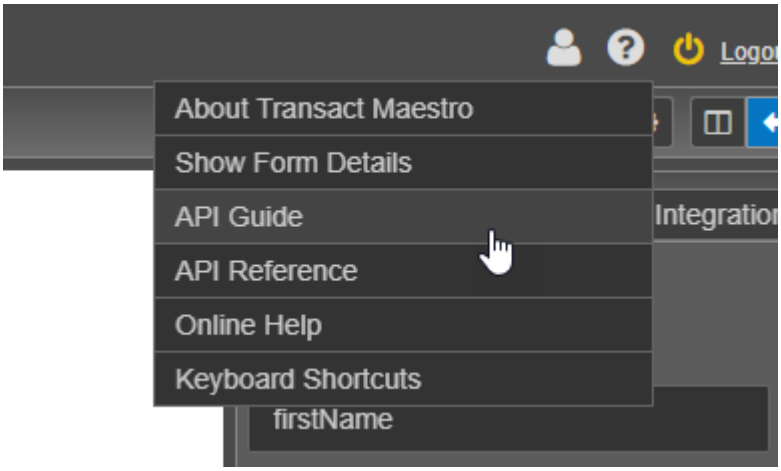
```
firstName: {
       $$parent: {span: {…}, offset: {…}, label: "About You", rows: Array(2), styles: Array(1),
…}
       $fieldSequence: 19
       category: "Input Fields"
       exclude: false
       getData: ƒ ()
       hasCustomId: true
       htmlLabel: "First Name"
       icon: "services/formresources/160/widgets/text-input/text-input.png"
       id: "firstName"
       isDisabled: ƒ ()
       label: "First Name"
       mandatory: true
       mandatoryMsg: "First Name is required"
       offset: {xs: 0, sm: 0, md: 0, lg: 0}
       path: "FirstName"
       properties: {inputType: "text", maxLength: "", preventCopyPaste: false, placeholder: "",
helpText: ""}
       setData: ƒ (value)
       span: {md: 12, lg: 12, xs: 12, sm: 12}
       styles: ["av-form-field"]
       type: "text-input"
}
```

Each item in the **items** object is identified by a unique ID. In the above example, "firstName" is the unique ID. Because of this, the items object doesn't need to nest items within a hierarchy. This makes it easier to access specific items.

view

The **view** object describes the design of the form. It includes a range of data, including the form's responsive breakpoints, the available brands, the current brand, the selected template, and the structure of rows and columns that define the layout of the form.

You can find more information about these objects in the Maestro API Guide. To view the API Guide, Open Maestro, click the **Help** button and select **API Guide**.



Debugging JavaScript Rules

All business rules in Maestro, from validation to calculation, are implemented in JavaScript. As a result, a straight-forward way to debug these rules is by using **`console.log`** statements in the script editor:

```
console.log("Hello world...");
```

To distinguish between different types of output, you can also use the warn and error methods:

```
console.warn("This message will have a yellow background");
console.error("This message will have a red background");
```

Below are some useful snippets for outputting relevant data inside a business rule:

```
// Inside a business rule, "value" refers to the data attached to the current field:
console.log(value);

// The "data" object, explained above, can be output from inside a business rule:
console.log(data)

// You can access any field value via the data object by referring to its ID:
console.log(data.lastName)

// The "item" object, explained above, can be output from inside a business rule:
console.log(item)

// Some components, such as the Page Controller component, output extra information via
the "info" object:
console.log(info)
```

While **`console.log`** statements are easy to use, you'll need to rebuild and reload the form every time you make a change. This makes it a fairly slow method of debugging.

For more productive debugging, breakpoints are a better option.

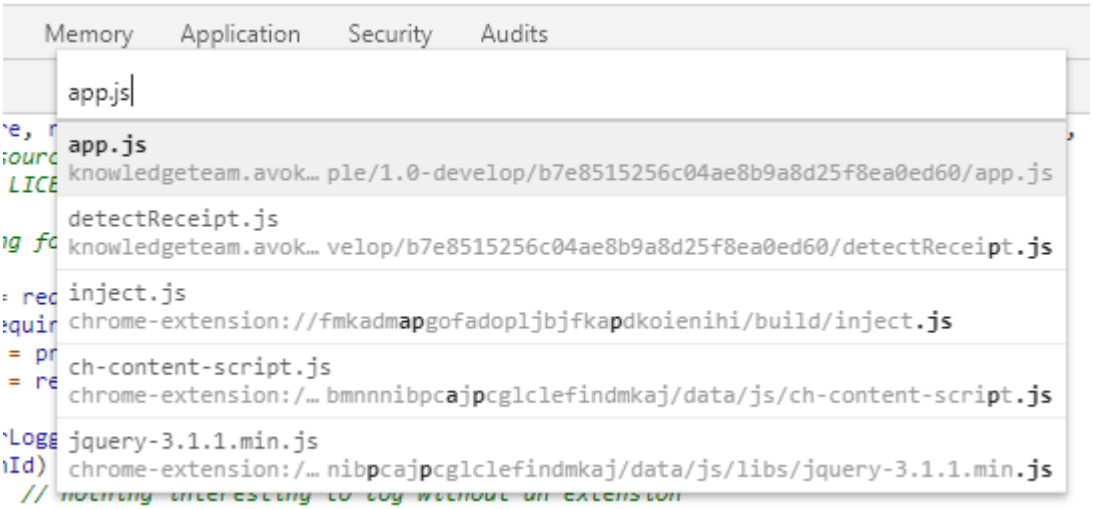
Using Breakpoints

Breakpoints allow you to define points in your JavaScript should pause its execution. This allows you to figure out exactly where your code is faulty without having to write a bunch of **`console.log`** statements or continually rebuild and refresh the form.

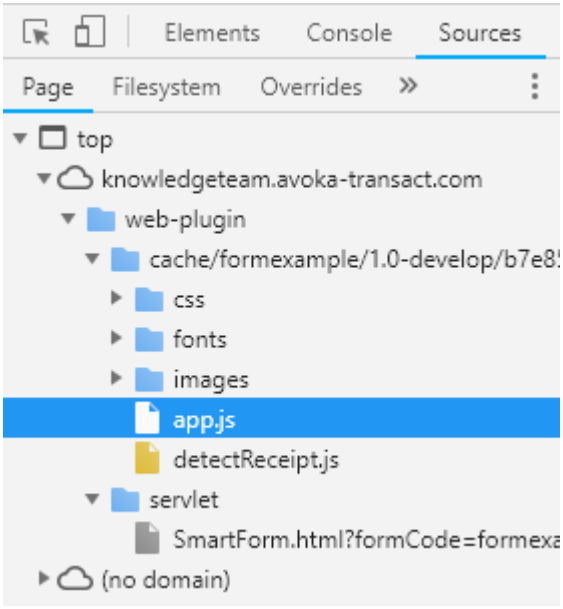
To use breakpoints, we need to locate the app.js file of our rendered Maestro form. This file contains the JavaScript code for our form's business rules.

To find the app.js file:

1. Open a rendered Maestro form.
2. In the **DevTools** pane, switch to the **Sources** tab.
3. Press `Control+O` (`Command+O` on a Mac), type "app.js", and click the first result:



Alternatively, you can navigate through the **Page** hierarchy on the left and double-click the app.js file:

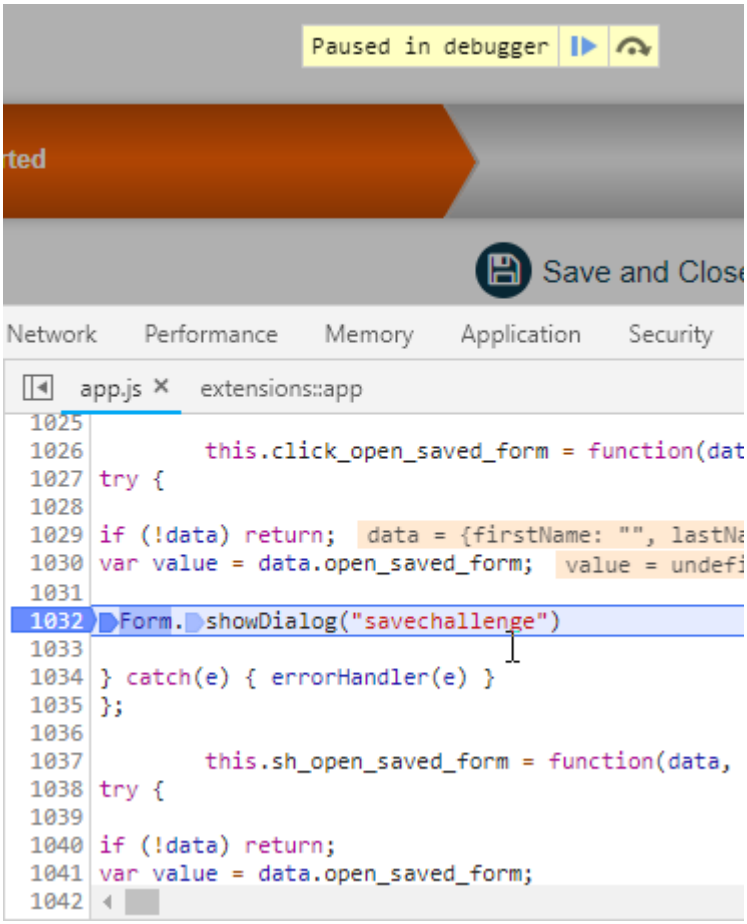


Then, to set a breakpoint, use the find feature in the DevTools pane (`Control+F` on Windows, `Command+F` on Mac) to find the following line of code:

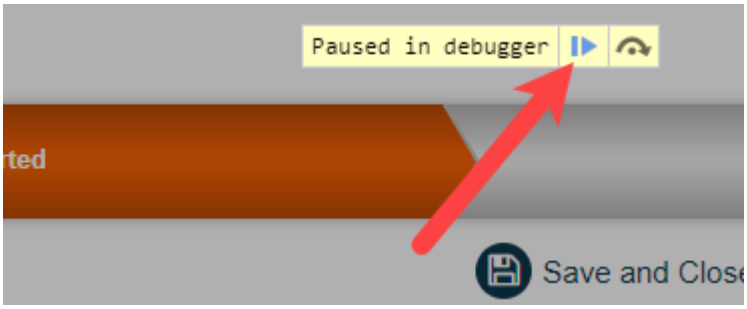
```
Form.showDialog("savechallenge")
```

This code is responsible for opening the modal that allows users to re-open a previously saved form.

If you click **Open Saved Form**, you'll notice that the modal doesn't appear. Instead, the line of code where the breakpoint is set will be highlighted, indicating that the execution of the JavaScript has paused on that line. Also, a "Paused in Debugger" message is displayed.



To continue executing the script, thereby allowing the modal to open, click the **Play** button:

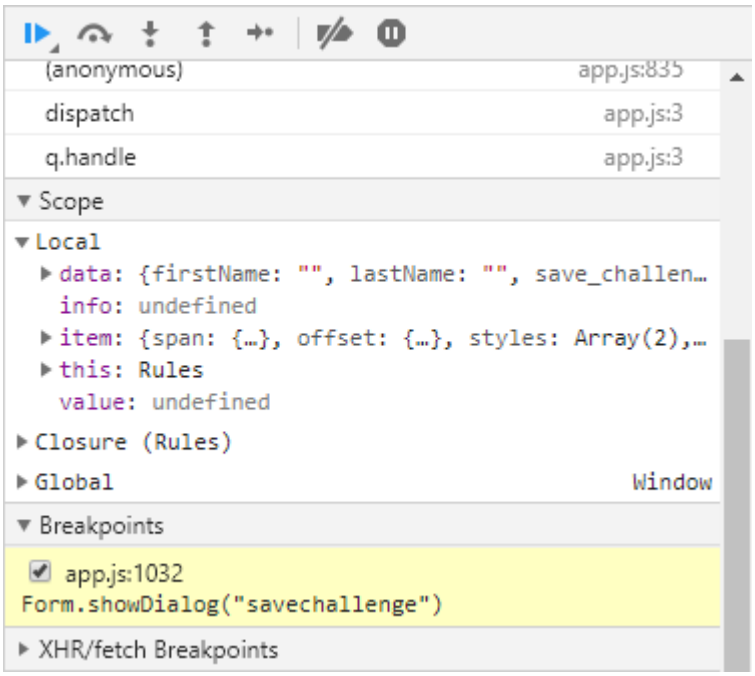


If you're trying to debug more complicated logic, you can set multiple breakpoints:



When you continue after each breakpoint, execution of the script will stop at the next breakpoint, allowing you to move step-by-step through a script.

While moving between breakpoints, pay attention to the **Scope** section. This will show the current state of the form's data, which may provide insight into why something isn't behaving as expected.



Rule Types and JavaScript Function Naming

As demonstrated above, it's not difficult to find the app.js file. In an actual form though, the file may contain hundreds of rules, which can make it difficult to find and debug the code for a specific rule.

One solution is to use the `Control+F` shortcut to search for a statement from one of the rules. For example, if you know there is a **`console.log`** statement in the rule, you can search for "console.log". But if you have a lot of rules, those rules may contain similar statements, making this approach less convenient.

Fortunately, every rule in Maestro has a corresponding JavaScript function that follows a strict naming convention. The functions are named using a rule type code and the ID of the field, with an underscore between them:

```
<rule-type-code>_<field-id>
```

Below are some example function names:

Function Name	Rule
click_verifyIdentity	A click rule for the verifyIdentity button.
eq_totalIncome	A calculation rule on the totalIncome currency field.
load_AvokaSmartForm	A load rule on the form.
ok_emailAddress	A validation rule on the emailAddress field.
sh_previousAddress	A visibility rule on the previousAddress block.

These function names are guaranteed to be unique, so they're a handy way to find specific rules in the app.js file.

Rule Type Codes

Here are the valid rule type codes:

Rule Type Code	Description
chok	Short for "change OK". This is the code for a validate after change rule that augments the standard validation rule to provide support for dynamic data validations. The result of this rule can contain the error message (like the standard validation rule) or a promise.
dc	An acronym for "dynamic class". This rule can be used to dynamically apply a CSS class to a field.
eq	Short for "equals". This is the code for a calculation rule. The result of this rule is the calculated value of the field. Note that calculation rules are not executed if the field they belong to is not visible.

md	Short for "mandatory". This is the code for a mandatory if rule that controls the required state of an item. The result of this rule is a boolean value where **`true`** = mandatory, **`false`** = optional.
ok	This is the code for a validation rule that controls the error state of the field. The result of this rule is a blank string for valid values or an error message string for invalid values.
sh	Short for "show". This is the code for a visibility rule that controls whether the field is presented on screen. This rule returns a boolean value where **`true`** = visible and **`false`** = read-only.
us	Short for "usable". This is the code for an editability rule that controls the read-only status of a field. This rule returns a boolean value where **`true`** = editable and **`false`** = read-only.

Action Rules

Action rules are triggered at various points in the lifecycle of a form.

Rule Type Code	Description
blur	Executed when the field loses focus.
change	Executed when the value of a field changes.
click	Executed when a clickable UI item is clicked.
focus	Executed when a field acquires focus.
load	Executed when the form is loaded.
presubmit	Executed prior to the form submit event.
postsubmit	Executed after the form submit event.

Dynamic Data Rules

Dynamic Data rules are specific to the Dynamic Data Button component and are triggered upon the successful or unsuccessful completion of a dynamic data call.

Rule Type Code	Description
onFailure	Executed when a dynamic data call fails.
onSuccess	Executed when a dynamic data call succeeds.

Next, learn about advanced debugging of Maestro forms.

Change Data Format in Form XML Data

Journey Manager (JM) | System Manager / DevOps | 18.11

Manager allows you to change data format in the **XML**[1] data of a selected transaction submitted by the user.

Starting from Manager version 18.11.0 the product now uses Open JDK 11, therefore the XML-based locale data is the Unicode Common Locale Data Repository (CLDR), older version of JDK used JRE(COMPAT), the new XML based locale introduced a stop character after the month Abbreviation for some locales like en_AU.

Below is an extract of the Form XML Data, where the Submit Date String before Transact Manager 18.11 is displayed:

```
<OfflineSubmissionId/>
<ReceiptNumber>XXXXXX</ReceiptNumber>
<SubmitDateString>17 Sep 2019 23:35:55 pm</SubmitDateString>
<UpdateUserProfile>false</UpdateUserProfile>
<ShareForm/>
<ServerBuildNumber>18.11.5</ServerBuildNumber>
<RevisionNumber>1</RevisionNumber>
<TrackingCode>XXXXXX</TrackingCode>
```

[1]XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

Below is an example of the Submit Date String from Manager 18.11 onwards:

```
<OfflineSubmissionId/>
<ReceiptNumber>XXXXXX</ReceiptNumber>
<SubmitDateString>17 Sep. 2019 23:35:55 pm</SubmitDateString>
<UpdateUserProfile>false</UpdateUserProfile>
<ShareForm/>
<ServerBuildNumber>18.11.5</ServerBuildNumber>
<RevisionNumber>1</RevisionNumber>
<TrackingCode>XXXXXX</TrackingCode>
```

To be able to get the date stamp without the extra full stop after the month, you will need to update your JBoss virtual machine variables.

You have to update the **JAVA_OPTS** variable to append the property string **-Djava.locale.providers=COMPAT**.

This action needs to be performed at **manager/server/bin/standalone.conf** on the Manager instance of the server.

> You will need to restart the Manager server to pull the new change.

Below is how the change in the configuration file should looks like in the different operating systems:

Linux

File: **standalone.conf**

Config: **JAVA_OPTS="$JAVA_OPTS -Djava.locale.providers=COMPAT"**

Windows

File: **standalone.conf.bat**

Config: **set "JAVA_OPTS=%JAVA_OPTS% -Djava.locale.providers=COMPAT"**

Next, learn how to track submissions.

System Logs

Journey Manager (JM) | System Manager / DevOps | 21.11

Manager captures a wide range of system and service run-time information and stores it in different log files. It provides a convenient interface for viewing this information, making it easier to troubleshoot diverse production problems.

Manager creates and maintains the following logs, which reside both on the file system and in the database:

- Event Log
- Error Log
- Error Log Trend
- Database Version Log
- T.Field Device Log | Removed in 21.11.
- T.Field Sync Log | Removed in 21.11.
- Import Log
- Audit Log
- Groovy Service Log
- Security Manager Log
- Apache Log

You can use System Event Publisher Overview to publish some logging events into various queues that are used by 3rd party tools, for example, Splunk.

When you log various events into a log file, there are a few recommended practices, as shown below:

Dos:

- Do choose the correct log file for the scenario
- Do make your log entries meaningful
- Do add a correlation key, if multiple log entries will need to be tracked (a submission key is often sufficient)
- Do use try and catch blocks for known issues

Don'ts:

- Don't log any Personally Identifying (PI) data - if you really must in order to write a peice of code then make a note or create a story to remove it later
- Don't log any Personally Identifying (PI) data - did I mention that already? Oh, well it is really important!
- Don't be overly chatty

Next, learn more about the Event Log.

Error Log

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

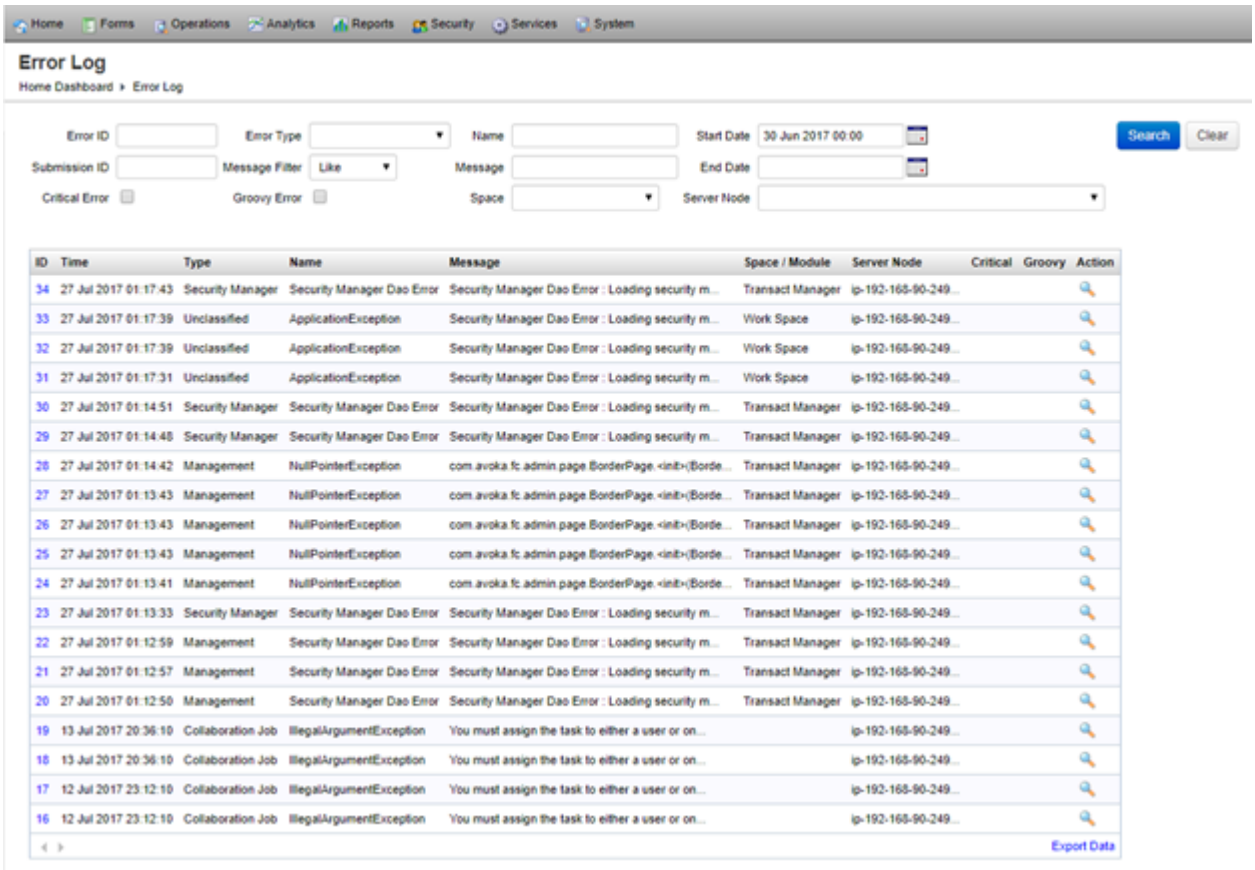
Manager records different types of errors that occur across all spaces and environments, such as:

- Lost Database connections
- Wrong form data configurations
- Delivery errors

Manager allows you to view , filter, and search for specific errors.

To view the error log, select **System > Error Log**.

Errors are shown page by page, with the maximum number of records per page configured via the user's preferences.

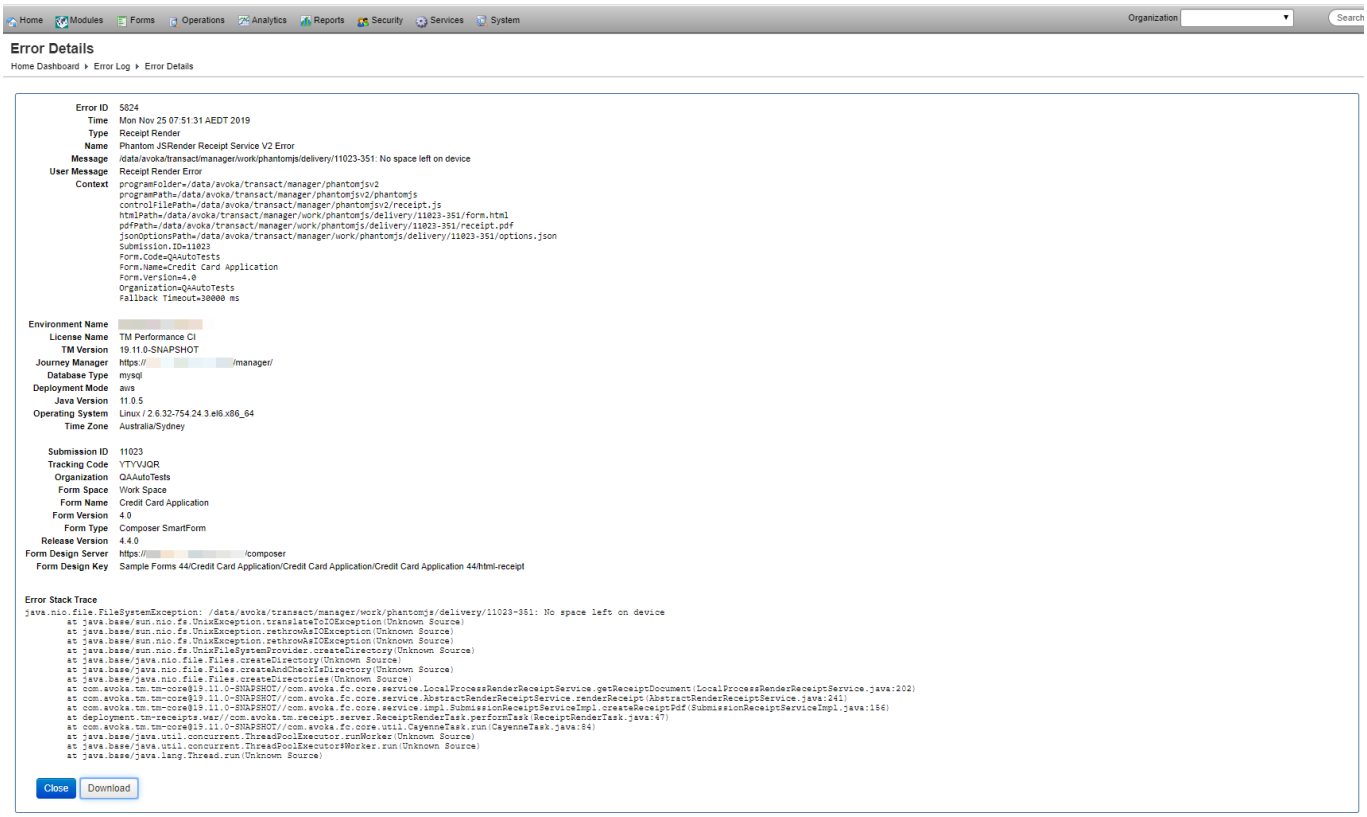


To filter or search the Error Log by one or more criteria, specify the following settings and click **Search**:

- Error ID
- Submission ID
- Critical Error
- Groovy Error
- Error Type is a type of the error, which Manager uses to group errors.
  - A error type can be one of the following:
    - Collaboration Job - errors encountered during collaboration job execution.
    - Email
    - Form Delivery - delivery of form data errors.
    - Form Dynamic Data
    - Form Render
    - Form Submissions
    - Insights
    - LiveCycle
    - Management – console errors, for example, errors cased by importing, modifying or deleting entries such as forms, spaces, or organizations.
    - Payment Gateway
    - Receipt Render
    - REST SOAP API
    - Scheduled Job
    - Security Manager
    - T.Field Sync
    - Unclassified – any other errors.
- Message text, or part of a message text. The Message Filter provides control on how the message text is used in the search, so select one of these: Like, Not Like, Equal, Not Equal.
- Name
- Start Date and End Date of the event.
- Space
- Server Node

Click **Clear** to reset the search criteria.

To view additional information, click **Details**.



This is a great source of information for error troubleshooting and it typically includes:

- Error details, such as Error ID, Time, Type, Name, Message, User Message, Context. Context often has a transaction ID or tracking code, so you can find this transaction in Transaction Support Search, where you are able to see more details as well as transaction properties. They may contain some crucial information to help you diagnose this error.
- Environment details, such as Environment Name, License Name, TM Version, Journey Manager, Database Type, Deployment Mode, Java Version, Operating System, Time Zone.
- Submission details, if present, such as Submission ID, Tracking Code, Organization, Form Space, Form Name, Form Version, Form Type, Release Version, Form Design Server, Form Design Key.
- Error stack trace.

Click **Close** to return to the previous page or click **Download** to save this error as an HTML file on your computer.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

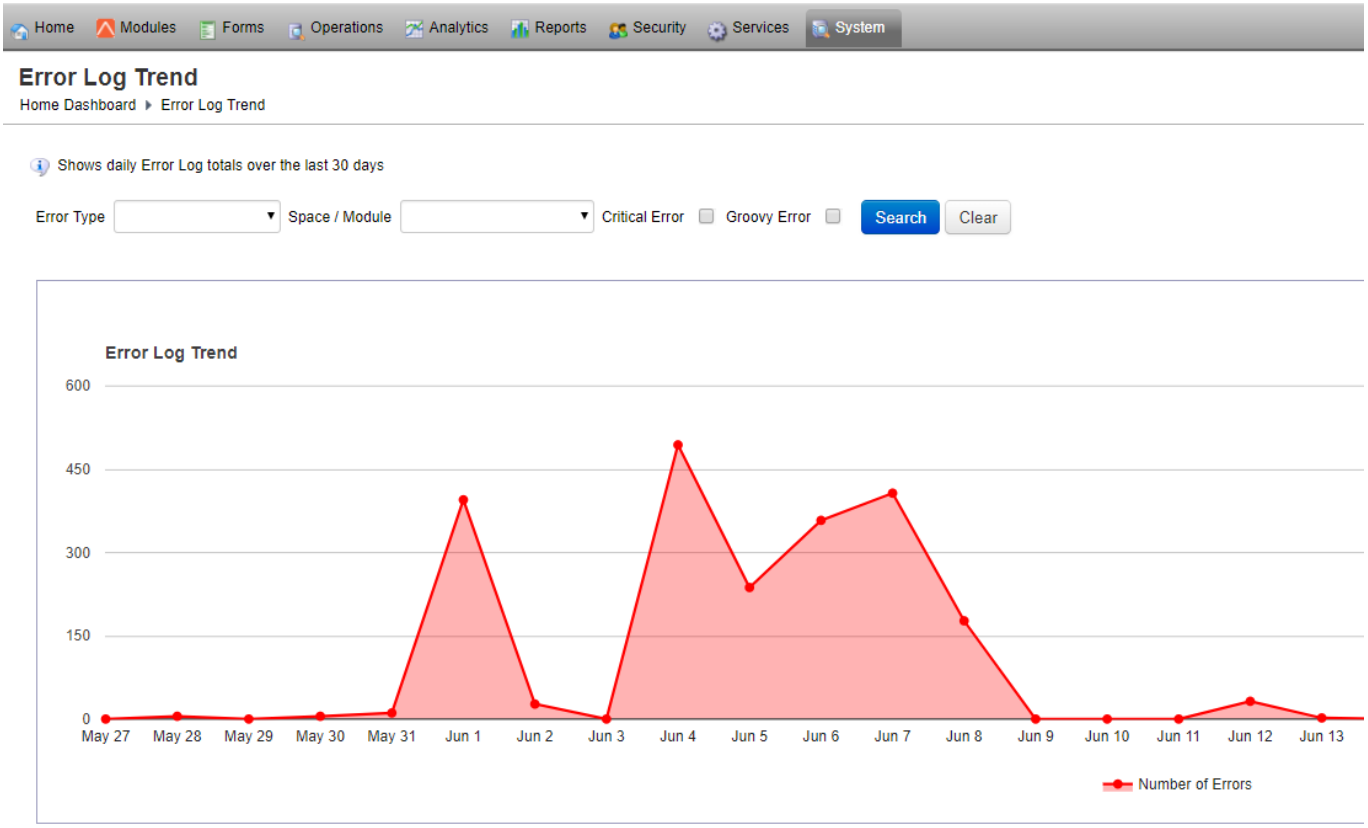
Next, learn about the import log.

Error Log Trend

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

The Error Log Trend chart shows the number of error entries logged by Manager for the last month. It provides a quick way of checking the health of the server.

To view Error Log Trend, select **System > Error Log Trend**.



To filter or search the errors by one or more criteria, specify the following settings and click **Search**:

- Error Type. For more information, see error log
- Space.
- Critical Error.
- Groovy Error.

Next, learn about other Manager logs.

Event Log

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager records different events in the Event Log, such as:

- A user notification for a submission delivery
- A notification of a submission is delivered to the organization
- A Manager restart or shutdown

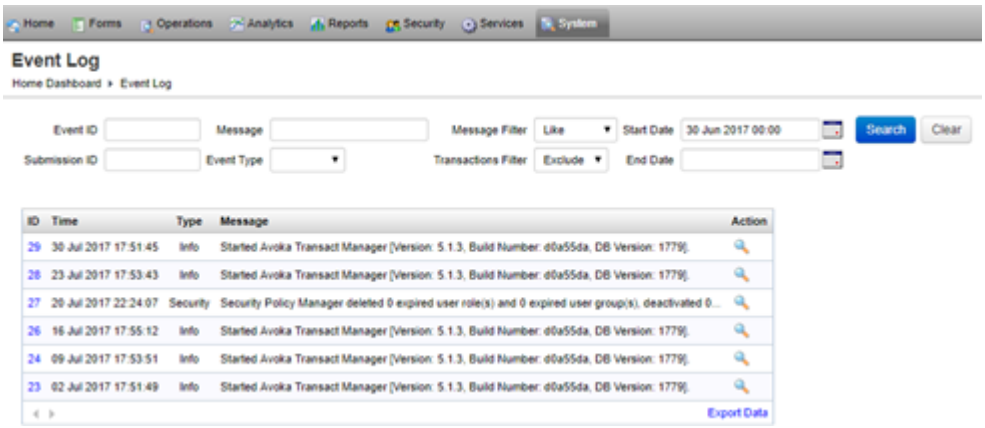
There are the following Event Types in the Event Log:

- **Info** - this event describes a Manager management console information
- **Warning** - this event identifies a warning message a user
- **Error** - this event identifies a Manager's error
- **Security** - this event indicates a security error, for example user accounts and log in authentication errors

Manager allows you to view, filter, and search for specific system events.

To access the event log, select **System > Error Log**.

Events are shown page by page, with the maximum number of records per page configured in the users preferences.

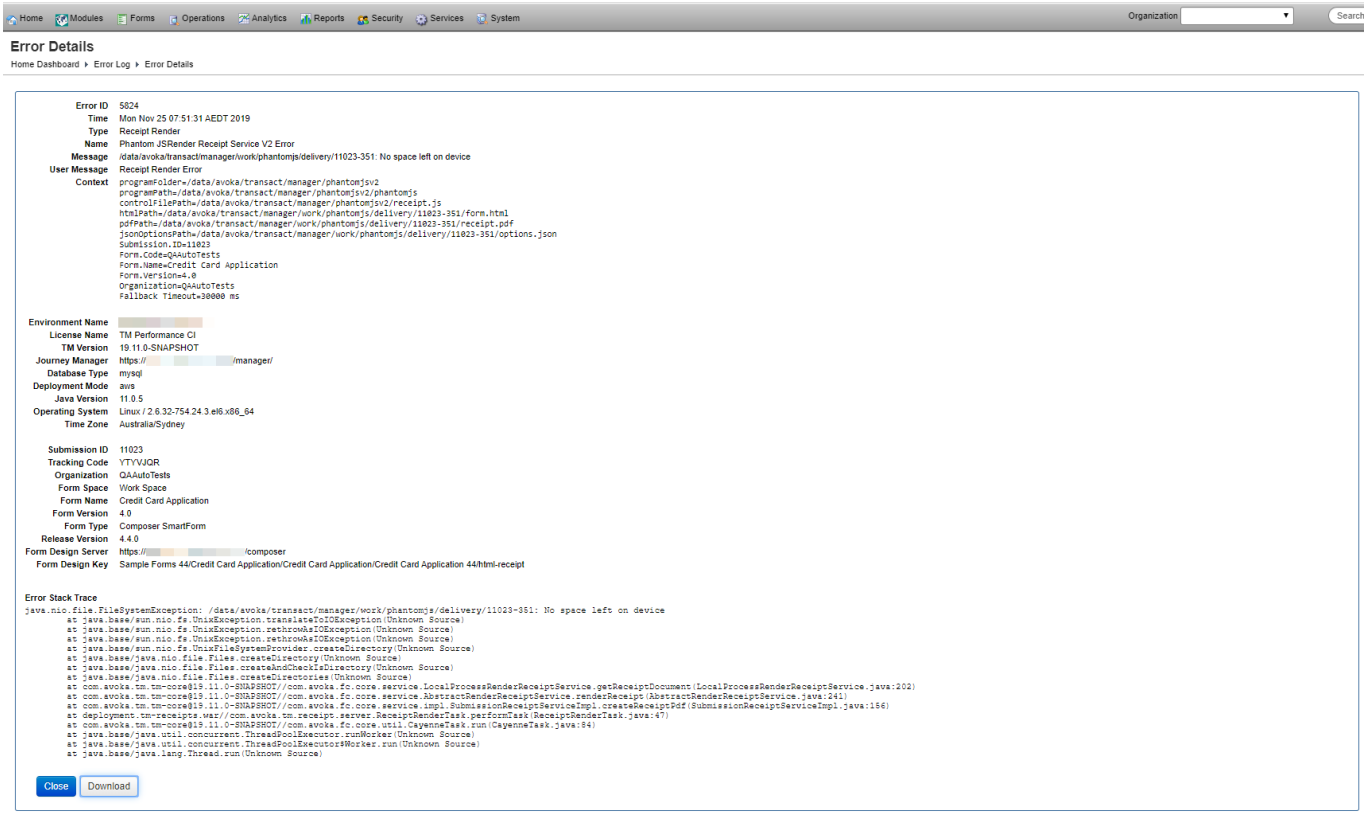


To filter or search the event log by one or more criteria, specify the following settings and click **Search**:

- Event ID
- Submission ID
- Event Type
- Message text, or part of a message text. The Message Filter provides control on how the message text is used in the search, so select one of these: Like, Not Like, Equal, Not Equal.
- Start Date and End Date of the event
- Transaction Filter with the Includeand Exclude options

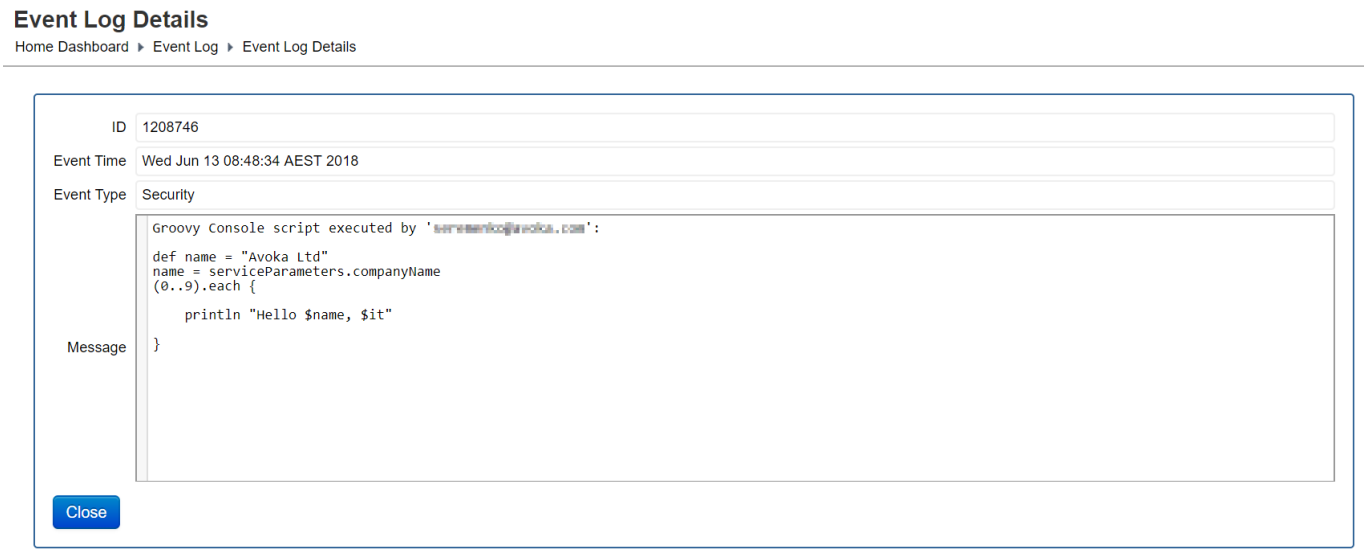
Click **Clear** to reset the search criteria.

To view additional information, click **Details**. The example below shows the security event message saying that a user has been locked out.



Click **Close** to return to the previous page.

> The Groovy script execution information is logged in the Event Log.



Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

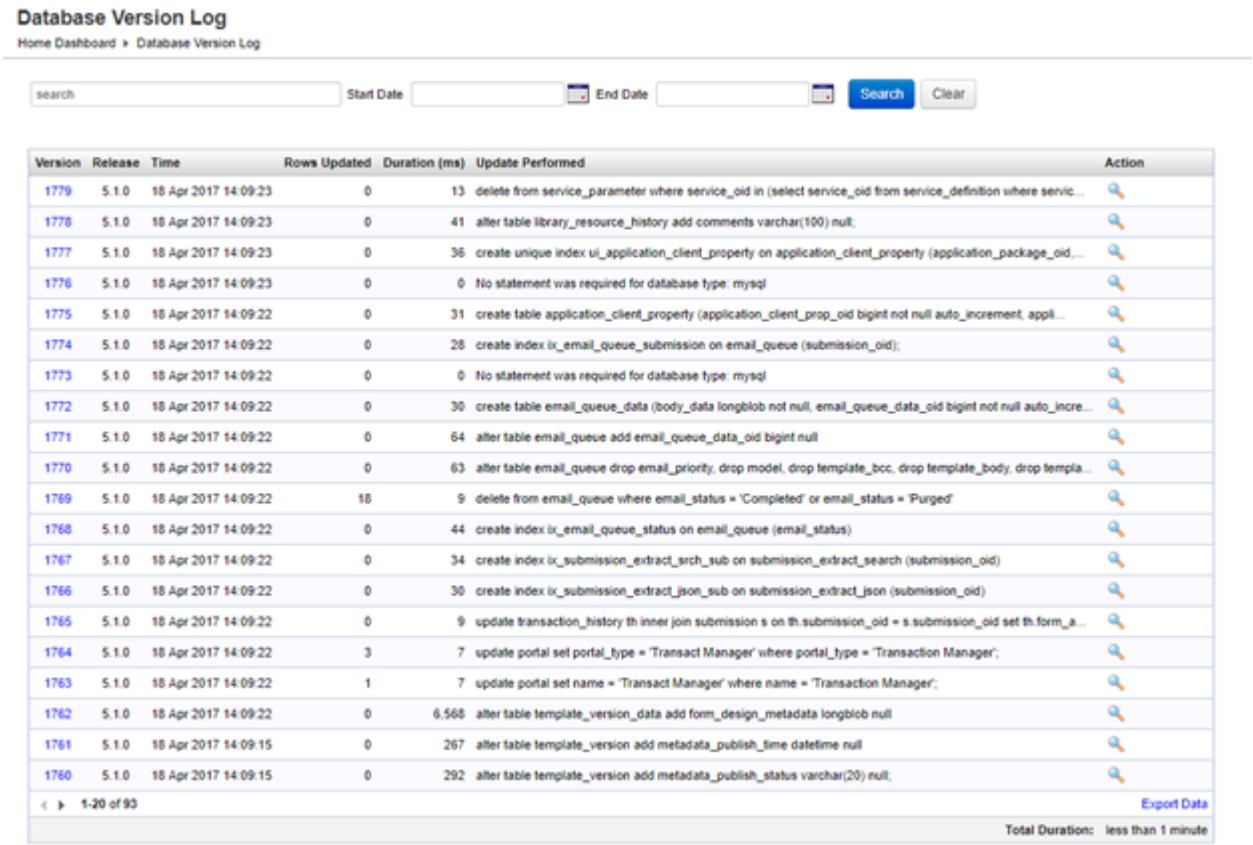
Next, learn about the error log.

Database Version Log

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Every time it starts, Manager checks the database version and schema name, runs update statements, and stores this information in the database version log. The database version log lists all updates applied to the database since Manager was installed. Each entry contains a version number and the description of an update.

To view the database version log, select **System > Database Version Log**.



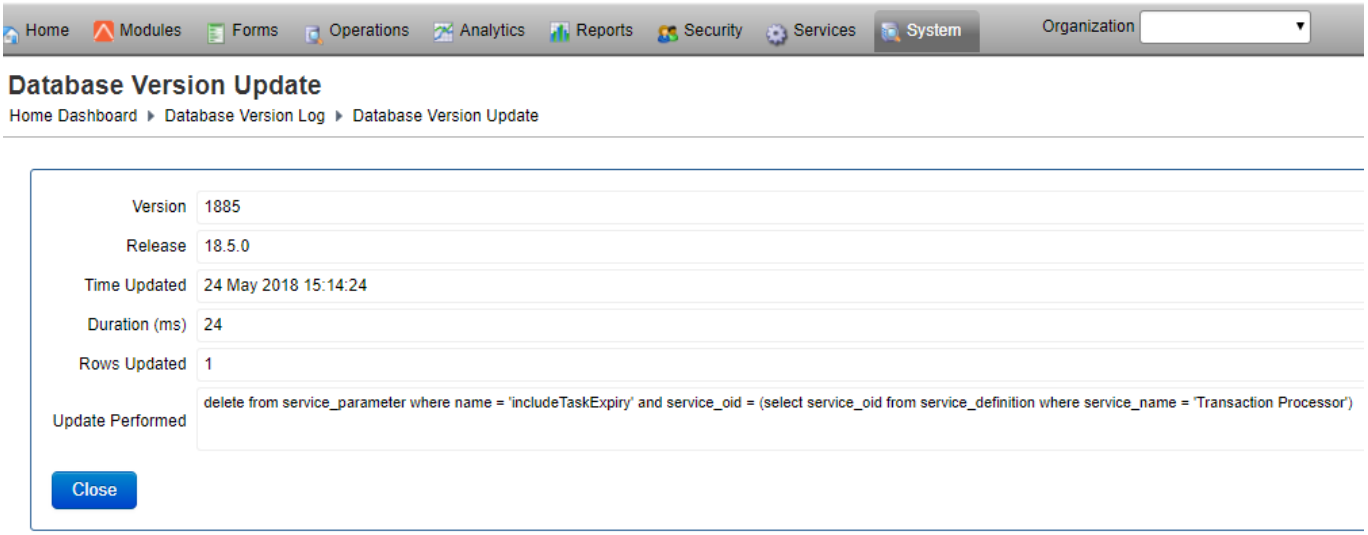
Log records are shown page by page, with the maximum number of records per page configured via the user's preferences.

To filter or search the database version log by one or more criteria, specify the following settings and click **Search**:

- Search text, which can be a release number, for example, 18.5.0, or a description of the update, as shown in the **Update Performed** column.
- Start Date and End Date of the event.

Click **Clear** to reset the search criteria.

To view additional information, click **Details**.



Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn about the error log.

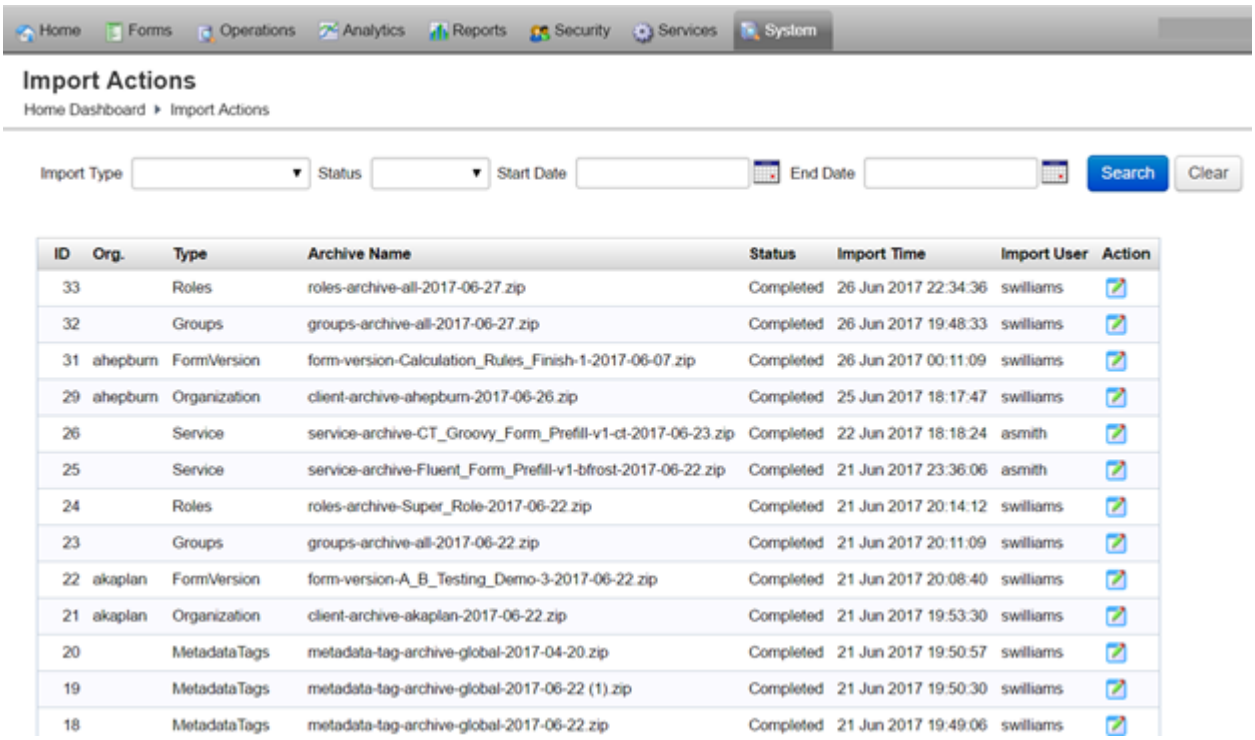
Import Log

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager keeps a log of all import operations and actions that have been performed for tracking and auditing purposes. This log contains records of each import outcome and any warnings or issues during the import process.

To access the import log, select **System > Import Log**.

Import records are shown page by page, with the maximum number of records per page configured via the user's preferences.



ID	Org.	Type	Archive Name	Status	Import Time	Import User	Action
33		Roles	roles-archive-all-2017-06-27.zip	Completed	26 Jun 2017 22:34:36	swilliams	
32		Groups	groups-archive-all-2017-06-27.zip	Completed	26 Jun 2017 19:48:33	swilliams	
31	ahepburn	FormVersion	form-version-Calculation_Rules_Finish-1-2017-06-07.zip	Completed	26 Jun 2017 00:11:09	swilliams	
29	ahepburn	Organization	client-archive-ahepburn-2017-06-26.zip	Completed	25 Jun 2017 18:17:47	swilliams	
26		Service	service-archive-CT_Groovy_Form_Prefill-v1-ct-2017-06-23.zip	Completed	22 Jun 2017 18:18:24	asmith	
25		Service	service-archive-Fluent_Form_Prefill-v1-bfrost-2017-06-22.zip	Completed	21 Jun 2017 23:36:06	asmith	
24		Roles	roles-archive-Super_Role-2017-06-22.zip	Completed	21 Jun 2017 20:14:12	swilliams	
23		Groups	groups-archive-all-2017-06-22.zip	Completed	21 Jun 2017 20:11:09	swilliams	
22	akaplan	FormVersion	form-version-A_B_Testing_Demo-3-2017-06-22.zip	Completed	21 Jun 2017 20:08:40	swilliams	
21	akaplan	Organization	client-archive-akaplan-2017-06-22.zip	Completed	21 Jun 2017 19:53:30	swilliams	
20		MetadataTags	metadata-tag-archive-global-2017-04-20.zip	Completed	21 Jun 2017 19:50:57	swilliams	
19		MetadataTags	metadata-tag-archive-global-2017-06-22 (1).zip	Completed	21 Jun 2017 19:50:30	swilliams	
18		MetadataTags	metadata-tag-archive-global-2017-06-22.zip	Completed	21 Jun 2017 19:49:06	swilliams	

To filter or search the import log by one or more criteria, specify the following settings and click **Search**:

- Import Type, for example organization
- Status of the import.
- Start Date and End Date.

Click **Clear** to reset the search criteria.

Click **Edit** to view the additional import information.

Click **Close** to return the main page of the imported component type, that is the TPac in this example.

To remove an import record, click **Remove** and confirm it.

In general, we don't recommend removing import records.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn about database version log.

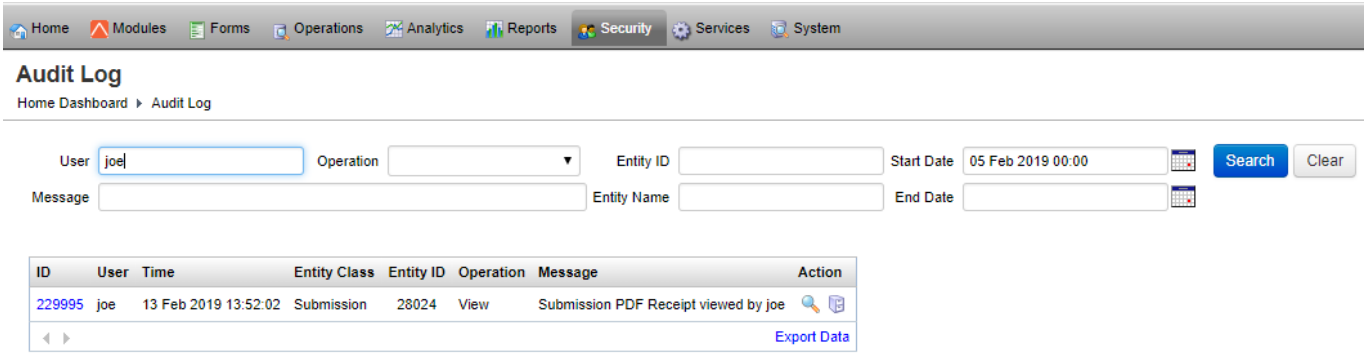
Audit Log

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager keeps track of various operations performed by users in the audit log. You can use this log to see what changes have been made and by whom.

To view the audit log, select **Security > Audit Log**.

Audit events are shown page by page, with the maximum number of records per page configured via the user's preferences.

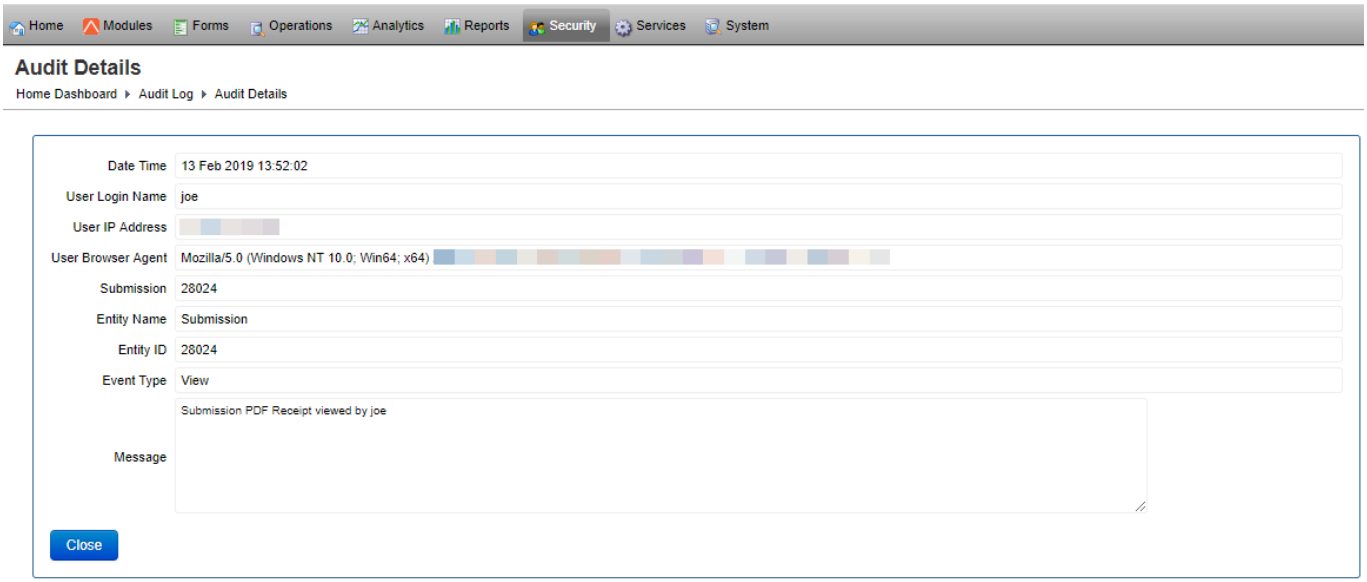


To filter or search the audit log by one or more criteria, specify the following settings and click **Search**:

- User name or email
- Operation: Create, Delete, Export, Import, Update, View
- Message text
- Entity ID
- Entity Name, for example Submission
- Start Date
- End Date

Click **Clear** to reset the search criteria.

Click **View** to see additional audit information as shown below.



Click **Close** to return the audit log.

Click **Submission View** to see transaction details of a selected submission.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn about the event log.

Groovy Service Log

Journey Manager (JM) | Form Builder | Platform Developer | 17.10

Manager allows you to view messages generated and logged to files by functions and services written in Groovy, so you can debug and profile them. When a form calls a Transact function, it executes a Groovy script, which can be programmed to write some information into a log file, for example, `logger.debug("Variable: " + varaibleValue)`. In this case, this message is logged and can be viewed in the **Groovy Service Log** tab.

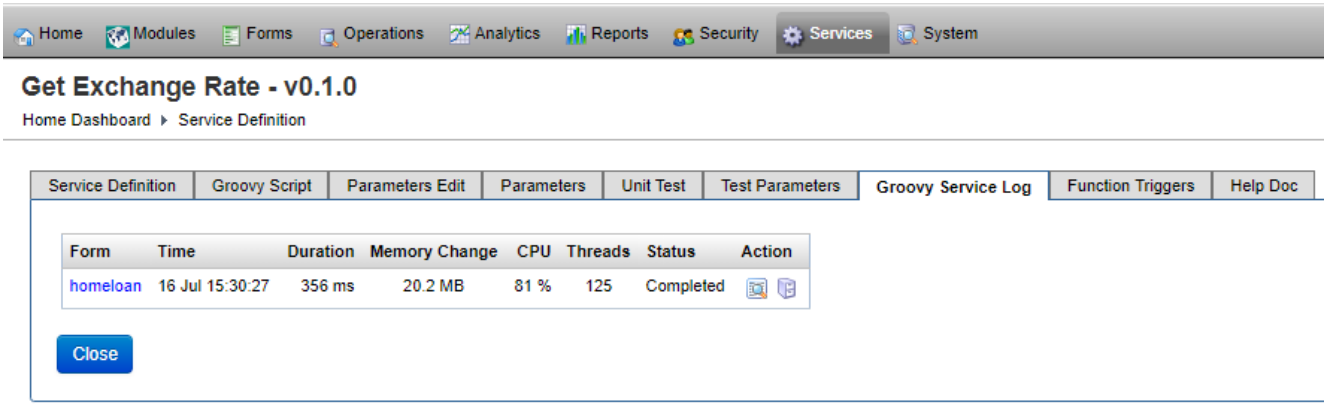
> All log messages associated with a transaction or **PII**[1] data are encrypted so you can't search for a part of encrypted text. If you want to search encrypted messages, you need to implement different logging solutions, for example, using the System Event Publisher.

The log can be empty if there is no debugging configured or if the debugging is disabled.

> We strongly advise against logging PII data to the service log, for example, to debug a service, as there's a risk of exposing customer's details through the log file.

To view a Transact function Groovy service log:

1. Select **Services > Transact Functions**.
2. Locate a Transact function and click **Edit**.
3. Click the **Groovy Service Log** tab to see the list of forms from where functions were invoked with the following details:



[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

- A function invocation time
- Duration of the invocation
- Memory change
- CPU
- Number of threads
- Status

4. Click a form name to see form details in a form dashboard.
5. Click **Groovy Logger** to view the output created during the function invocation.

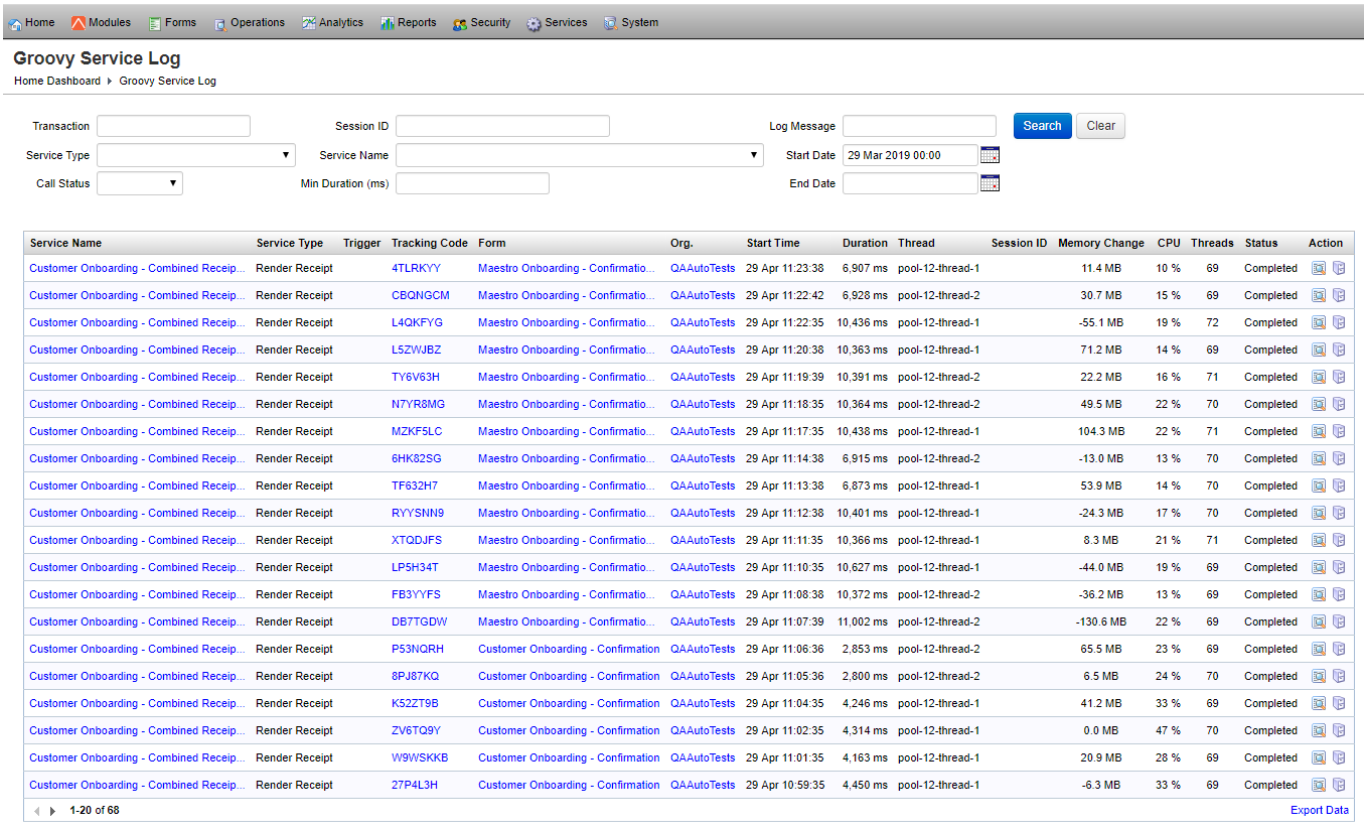
> Select the **Groovy Log** tab to see the same logged messages.

6. Click **Submission View** to see details of a transaction associated with this function call.
7. Click **Close**.

> To enable Groovy logging, select the **Groovy Scripting Enabled** checkbox in the Parameters Edit tab.

To view all Groovy service log messages, select **Services > Groovy Service Log**.

Messages are shown page by page, with the maximum number of records per page configured via the user's preferences.



To filter or search the Groovy Service Log by one or more criteria, specify the following settings and click **Search**:

- Transaction
- Session ID
- Log Message, or part of a message text, This searches through unencrypted messages logged by services that are not transaction-related.

  > You can't use this functionality to look for a part of encrypted text.

- Service Type. Select a type of a service from the dropdown list to see all messages logged by services of this type.
- Service Name. Select a service name from the dropdown list to see all messages logged by services of this name.

  > Only services of a selected service type are shown in the dropdown list, if you used the Service Type filter above.

- Call Status. Select a status of the message from the dropdown list, that can be: **Completed**, **Error**, or both.
- Min Duration (ms). Select a minimum duration of a service call. This is handy to troubleshoot services and overall server performance by filtering down services that took longer to do work.
- Start Date and End Date of the message.

Click **Clear** to reset the search criteria.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how configure Transact function service definition.

Groovy Log to Splunk Integration

Journey Manager (JM) | System Manager / DevOps | 21.05

Splunk is popular software for searching, monitoring, and analyzing machine-generated data, such as log files, which you can use it to have insights into errors and perform proactive monitoring and alerts for your applications.

Although Manager doesn't provide a direct integration with Splunk, you can configure a System Event Publisher with an AWS SQS adapter to push log records to an AWS queue. Then, you integrate Splunk with the AWS SQS, so it pulls Manager log records from there.

The Splunk integration is summarized below:

1. Enable and configure a system event publisher in Manager
2. Configure an AWS SQS adapter and configure a service connection to use the AWS SQS service
3. Configure Splunk AWS SQS adapter to consume (pull) the AWS SQS messages, transform and store them in the local Splunk data storage
4. Start using Splunk to analyze and monitor your applications

It is important to understand that you will deal with messages as events, not raw exports of log data, so you need to distinguish what an event and log record mean in this context.

Downside of such integration is that an event may contain redundant data, such as link to a transaction, within a set of messages (events) for the same transaction, which can dramatically increase amount of raw data to be stored on a client's side. The logic importing this data to Splunk should consider that.

It means that, for instance, a submission status change event will have event logs and error logs linked to the submission. If the submission status changes multiple times (and it often does), all event messages to the AWS SQS will contain all submission's event and error log records up to that milestone. In short, you will be getting duplicates of the early recorded log records.

> This solution creates significant amount of data in Splunk, as compared to Manager logs.

If you want to analyze the Groovy service log records in Splunk, you need to know that the Manager's **`GroovyLogger`** class internally has a SLF/Log4J logger which writes messages to a **`service.log`** file. The default logging level is **`ERROR`**, which means **`DEBUG`**, **`INFO`** and **`WARN`** messages are not logged to disk. The reason for this default setting is to prevent **PII**[1] data being logged to disk.

---

[1] Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

If this is configured to **INFO** or **DEBUG** level, then **GroovyLogger** statements at those levels will be logged to the **server.log** file as well. From here they can be feed into Splunk or other log management systems, regardless of the Groovy Service Logging configurations for the database.

The System Event Publisher can feed Groovy Service Log Data message to the AWS SQS endpoint to support integration with other log management systems without having to write PII data to disk. The Groovy Service Log Data messages will include the same content as would be included in the Groovy Service Log Data records, so it cannot be used as a complete substitute to the Groovy Service Log records.

Next, learn about the system logs.

Apache Log

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Being a web application that runs on an Apache server, Manager writes some important system information, such as event and error messages, into Apache log files. You generally don't need to check these log files unless you are troubleshooting some server errors, for example, **FormDynamicDataServlet - Servlet not authorized**.

To view the content of Apache log files, contact your Support team or Cloud Services team and ask them to send you a log file or a snippet of a file as per your requirements, for example, a date and time interval when an error took place.

> You can't access Apache log files from the Manager console.

Next, learn about the error log.

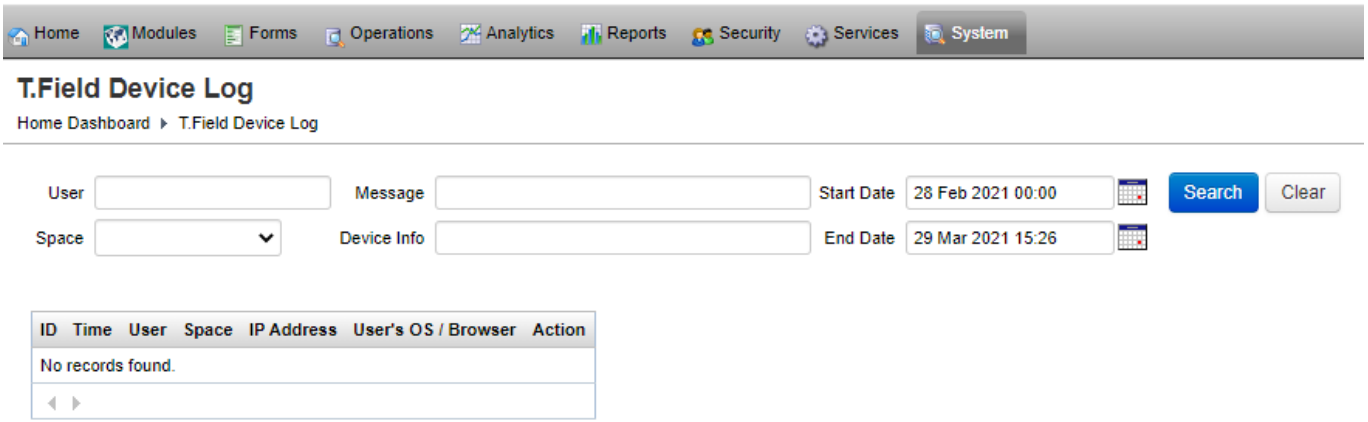
T.Field Device Log

Journey Manager (JM) | System Manager / DevOps | Removed in 21.11.

Manager comes with many types of log files and one of them is T.Field Device Log that allows you to view log files of a device, which runs the TransactField App. A TransactField App user must elect to upload their device log to a Manager server first. This can be very helpful in debugging issues occurring on the device running the TransactField App.

To view T.Field Device Log, select **System > T.Field Device Log**.

Log entries are shown page by page, with the maximum number of records per page configured in the users preferences.



To filter or search log entries by one or more criteria, specify the following settings and click **Search**:

- User
- Message
- Space - select a form space.
- Device Info
- Start Date and End Date of the event.

Click **Clear** to reset the search criteria.

To view additional information, click **Details**. You can also view the corresponding synchronization log entry.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn about T.Field sync log.

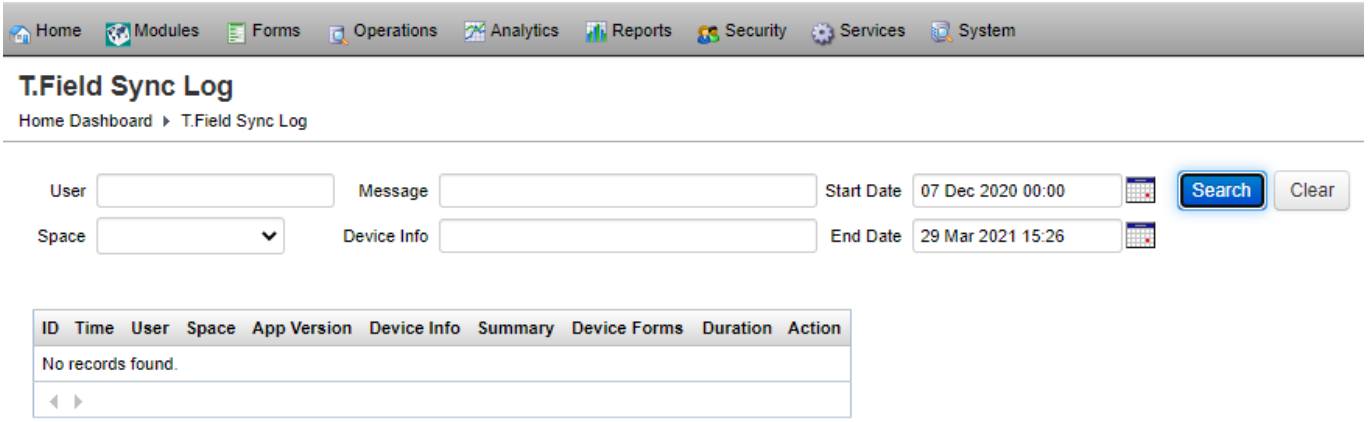
T.Field Sync Log

Journey Manager (JM) | System Manager / DevOps | Removed in 21.11.

Manager comes with many types of log files and one of them is T.Field Sync Log that allows you to view operations invoked by devices running the TransactField App. This is useful in debugging the TransactField App issues.

To view the T.Field Sync Log, select **System > T.Field Sync Log**.

Log entries are shown page by page, with the maximum number of records per page configured in the users preferences.



To filter or search log entries by one or more criteria, specify the following settings and click **Search**:

- User
- Message
- Space - select a form space.
- Device Info
- Start Date and End Date of the event.

Click **Clear** to reset the search criteria.

To view additional information, click **Details**.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

> TransactField App users can elect to upload their device log to the Manager server, which you can viewe using T.Field Device Log

Next, learn about T.Field device log.

Journey Manager Analytics

Journey Manager (JM) | Analytics User | 5.1 & Higher

Journey Manager collects a broad range of data related to user's interactions with forms, such as form submissions, and visualizes it with via the following charts.

- Transactions trends
- Forms activity
- Referer submissions
- Form submissions
- Device types submissions
- Reader Versions
- Submission Trend - | Removed in 18.05.

Most charts are based on an additional set of database tables that hold a subset of submission and request data. These tables are regularly updated are more lightweight than the submission and request tables themselves. The data contained in the tables can also be purged automatically and independently from the submission and request tables.

Manager also provides A/B testing functionality, so you can perform A/B testing and analyze the results to decide which form version to use.

For more extensive transaction analysis, you should use Journey Analytics.

Analyze Transaction Trend

Journey Manager (JM) | Analytics User | 18.05

Manager provides an easy to use tool to visualize and analyze transaction trends using the following charts:

- Transaction volume
- Completion rate
- Conversion rate
- Form abandonment rate
- Bounce rate
- Median form completion time taken by users to complete a submission.

Transactions are often long running and can take many days or even weeks to complete, so Manager makes the following assumptions while calculating the trends:

- Analytics for completion, conversion, form abandonment, and bounce rates exclude in-progress transactions, such as opened, saved, and assigned.
- Completion, conversion, form abandonment, and bounce rates are trailing indicators which become more accurate over time.
- Daily bounce rates and form started abandonment rates become available 48 hours after forms were first opened (request time).
- The start date and end date filters are based on the transaction request time (start time).
- Analytics exclude collaboration job, task, and Manager server monitoring transactions.

All transaction trend charts display a daily value on the x-axis. If the selected date range is too wide, Manager plots the trend graph for every two or more days to fit this date range in the given screen size, so we advise you to restrict date filtering to shorter ranges. However, you can see an exact daily figure for volume, percentage, rate, and time, by hovering over specific points plotted on any of the graphs.

> You can also use Journey AnalyticsJourney Analytics to analyze transaction trends as it provides more options to drill down into user transactions and better visualization. You will, however, need the Journey Analytics module installed in your Manager environment.

> Manager and Journey Analytics calculate transaction trends differently depending on multiple factors, such as analytics implementation, and are beyond the scope of this documentation.

To view the transaction trend:

1. Select **Analytics > Transaction Trend**.



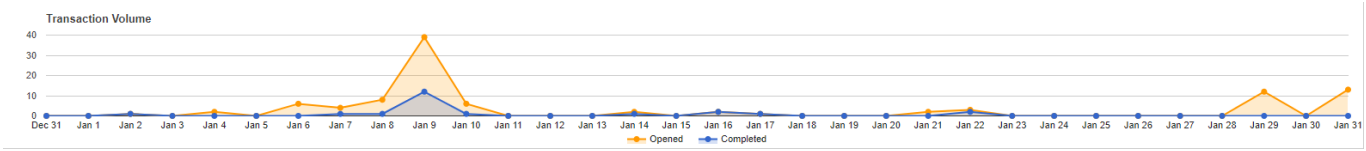
2. Select an organization from the **Organization** dropdown list to retrieve statistics relevant to this organization only. If not selected, statistics for all organizations is shown.
3. Select a form from the **Form** dropdown list to include only the transactions related to that form. If left blank, the transactions related to all forms are included in the chart.
4. Select a form space from the **Space** dropdown list to include only the transactions of that form space.
5. Select a device from the **Device** dropdown list to include transactions accessed by users from the following devices:
   - Desktop
   - Phone
   - Tablet
6. Select the **Jobs & Tasks** checkbox to include collaboration job and task transactions. Otherwise, they are excluded. | 18.05 .
7. Specify the date range of transactions to be included in the chart from the **Start Date** and the **End Date**. By default, this is set to a date range starting one year before, up to and including today.

> The maximum date range is 6 months only.

8. Click **Filter** to generate the transaction trends based on the selection criteria and review them as shown below.

Transaction Volume

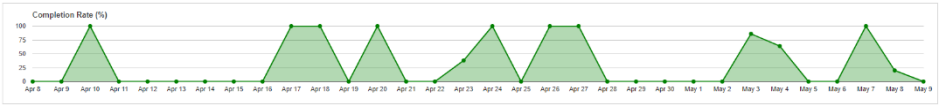
This chart displays the number of Opened (both started and bounced) and Completed transactions per day. Transactions completed on a later date are counted against the day they were started. Transactions not completed on the same day as they were started, appear against their start date and not their completion date. Collaboration job and tasks are not included.



The opened transactions are plotted in yellow and the completed transactions are plotted in blue.

Completion Rate (%)

This chart provides the percentage completed transactions divided by the number of started form transactions excluding bounced transactions as they are typically form visits where the user did not start filling in the form.



Conversion Rate (%)

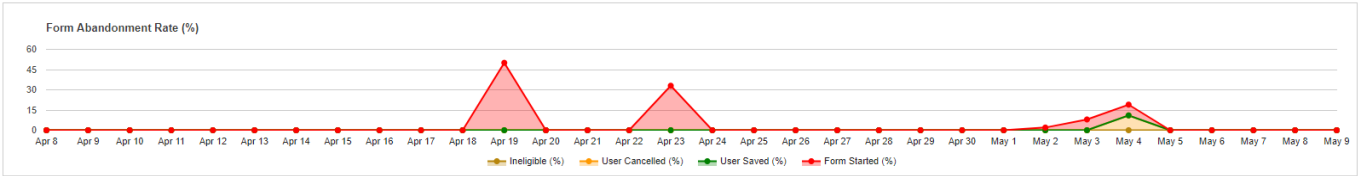
This chart provides the percentage completed transactions divided by the number of opened transactions or visits (includes bounces). Form bounces or visits are included in the conversion rate. | 17.10



Form Abandonment Rate (%)

This chart displays the percentage breakdown of the different types of user abandonment (never completed or submitted). These types of form abandonment include:

- Ineligible - the form determined that the user is not eligible to complete the transaction, and the transaction was terminated.
- User Cancelled - the user explicitly clicked on the Cancel form button. Note not all form designs include this button. This information becomes available immediately.
- User Saved - the user has explicitly saved the form and then closed it but has never returned to resume the form. This type of abandonment is determined over time, and can take up to 30 days depending upon the form's configuration.
- Form Started - the user has started interacting with the form (entering data, clicking on buttons or navigating around) but they never perform an explicit save or submit. Form Start abandonment is determined 24 hours after the transaction has been opened.

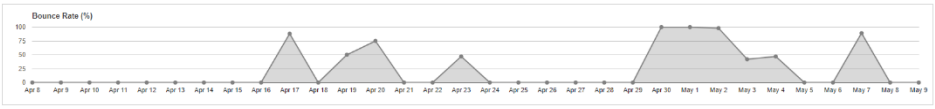


Bounce Rate (%)

This chart displays the percentage of **bounced**[1] transactions divided by opened (rendered) transactions per day.

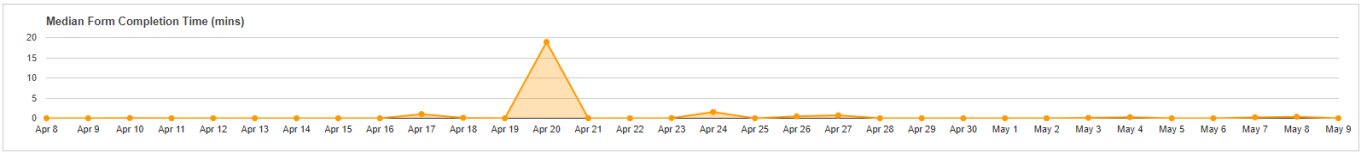
---

[1]Bounce is when a user views a form but never starts it and simply closes the window immediately or navigates away.



Median Form Completion Time (mins)

This chart displays the median form completion time in minutes or the cumulative amount of time it took users to complete the transaction, potentially across multiple form sessions.



The averages of the above stats are displayed on the right hand side of the screen.

Click **Download CSV** to export the displayed data as a CSV file.

Manager prior to v17.10, uses **Export Excel Data** link instead of the **Download CSV** button.

Next, learn about all Manager analytics.

Form A/B Testing Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

A/B testing, also known as split testing, is a process of comparing two versions of a form to find out which one performs better. Two variants of the same form, A and B, are shown to similar visitors and the one that gives a better conversion rate will only be used.

Improving conversion rates is often imperative to any form modification. To easily measure the effectiveness of any proposed changes made to a form prior to adopting them, the form A/B testing feature allows for a trial period. After the trial period, it is hoped that a declared clear winner can be implemented as the newly adopted version of the form.

Manager provides the form A/B testing functionality, which you can use to compare versions of a form, usually the new proposed one versus the previous, to test the effectiveness of certain modifications. During this trial period, the modified form is tested in the following way:

- Whenever a user requests this form, Manager randomly chooses a form version from the candidates to display.
- All user activities, especially submissions by form version, are captured.
- Form version metrics, for example completed transactions, are monitored and analyzed.

You can view form A/B testing results as a set of graphs to determine the effectiveness of form changes between the versions.

> For more extensive form A/B testing, you should use Journey Analytics.

Next, learn how to configure form A/B testing.

Configure Form A/B Testing

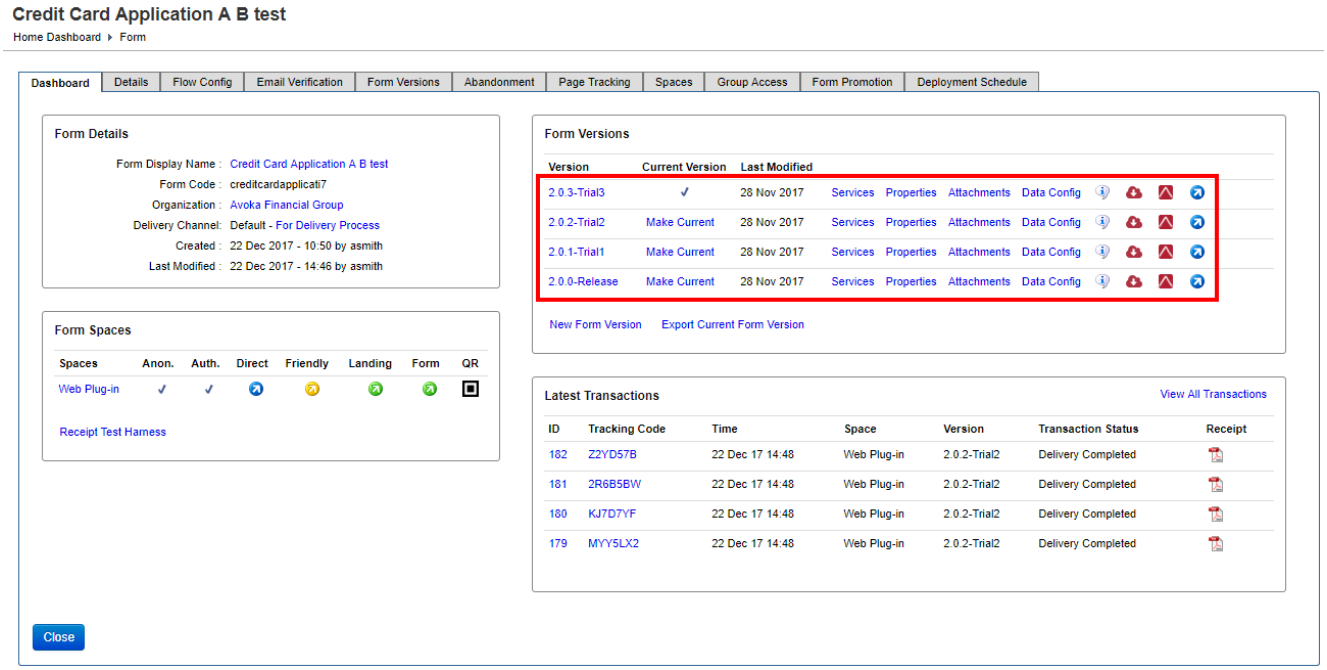
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to perform A/B testing of a form, if it uses the A/B Testing Form Version Selector service which captures and visualizes all user's requests to this form, so you can analyze this information and decide which form is most suitable and should be used.

After you have finished with the A/B testing, you need to change the form configuration back to the original one.

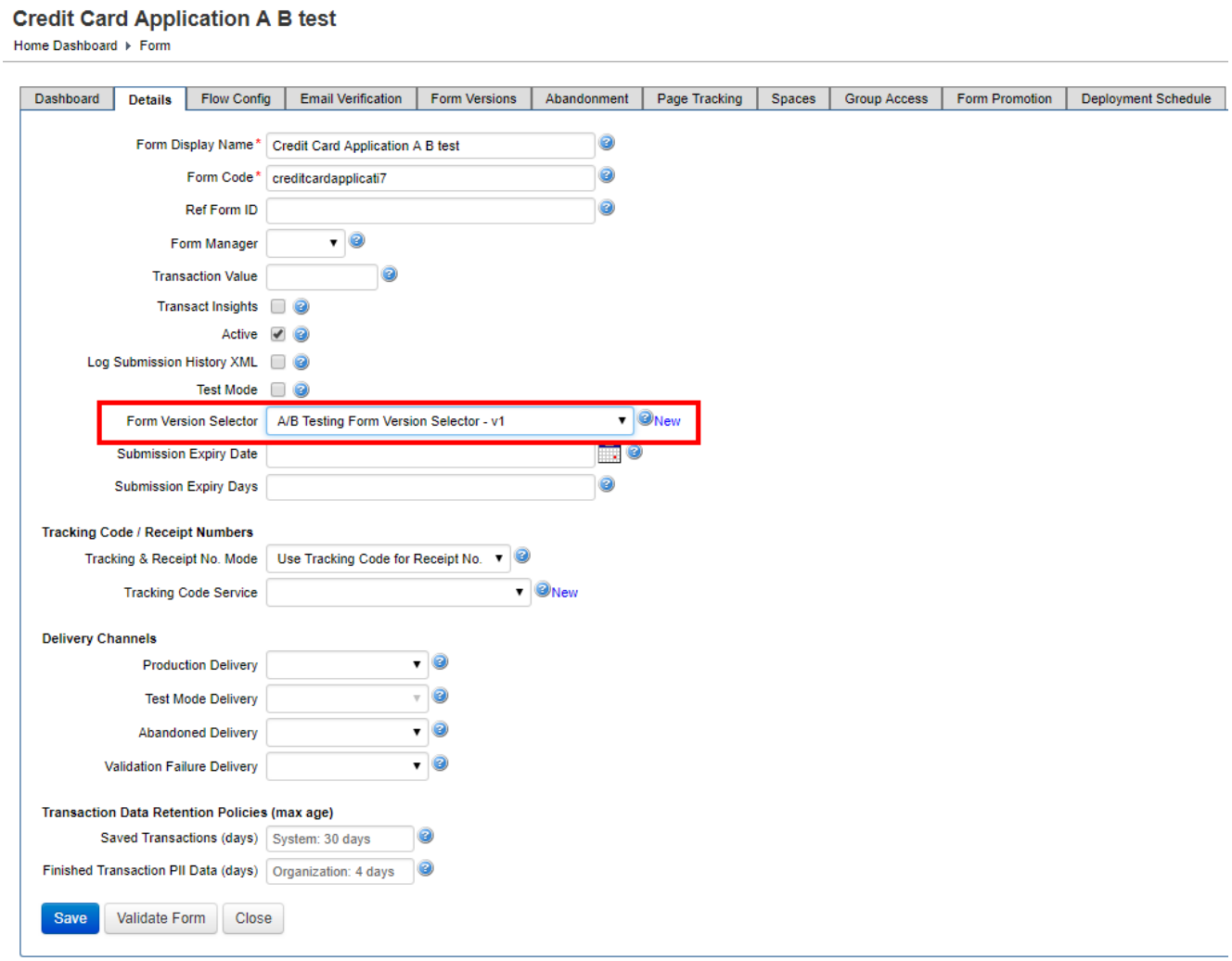
To start the form A/B testing:

1. Select **Forms > Forms.**
2. Locate a form and click **Edit**.



In this example, the form has not yet been configured for A/B testing, so only the current version of the form is offered to users.

3. Click the **Details** tab.
4. Select the **A/B Testing Version Selector - v1** service from the **Form Version Selector** dropdown list.



5. Click **Save** to update the changes.

Based on this change, the A/B Testing Form Version Selector will randomly select and display one of the form's versions to form users. To have more control over which versions are tested, learn how to elect versions a form for A/B testing.

To stop the form A/B testing:

1. Select **Forms > Forms**.
2. Locate your form and click **Edit**.
3. Select the **Details** tab.
4. Select the blank option from the **Form Version Selector** dropdown list.
5. Click **Save** to update the changes. Users will now be presented with the current form version.

Next, learn how to view form A/B testing results.

Elect Form A/B Testing

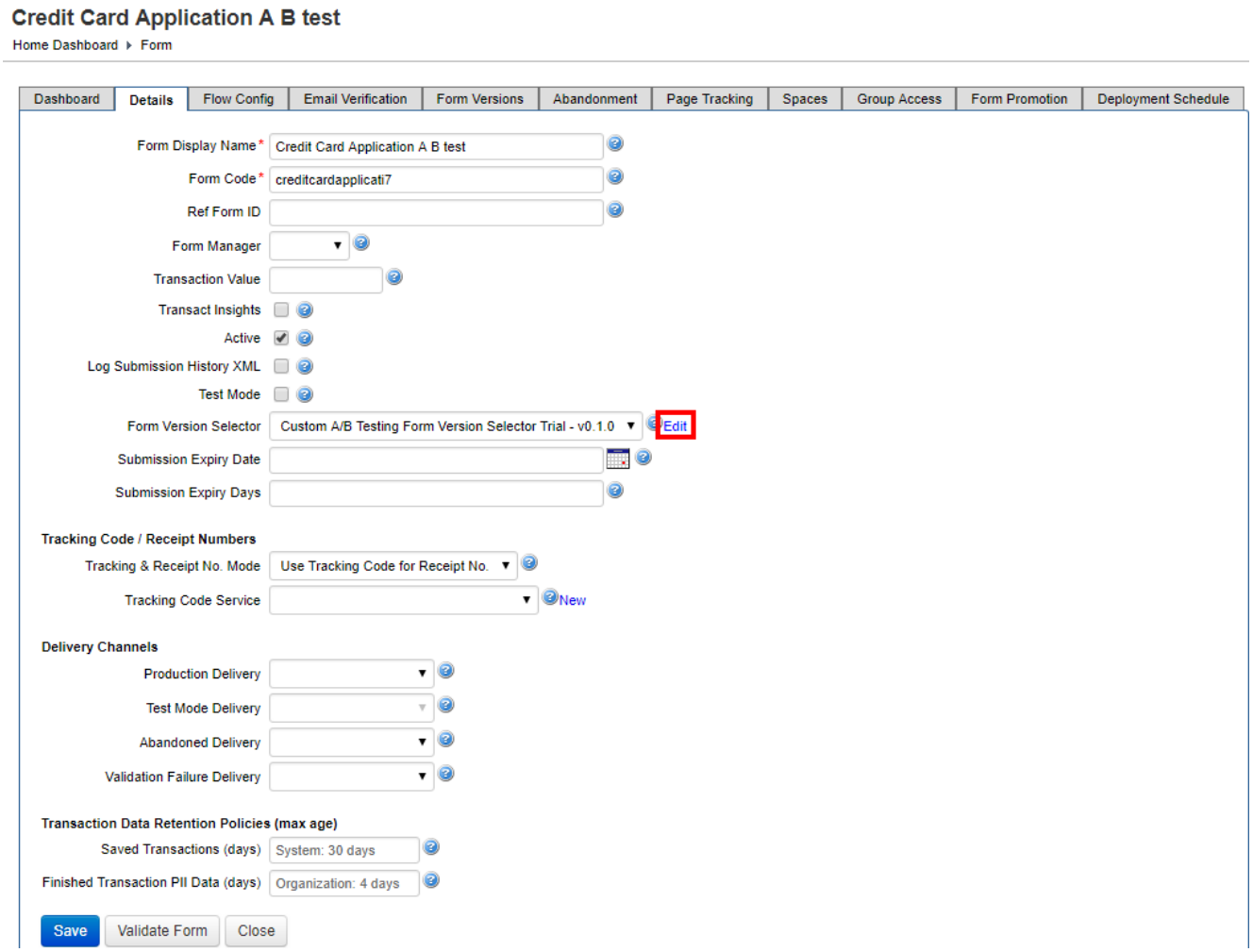
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to perform A/B testing of a form. In most cases, the form A/B testing is for a new version of a form versus its previous version. To accomplish this, you must *elect* the latest two versions of the form to include in the form A/B testing.

> Form A/B testing is handled by the A/B Testing Form Version Selector service, which is a core global service that must never be modified. You should copy this service before continuing.

To nominate specific form versions to be included in the form A/B testing:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select the **Details** tab.
4. Select the copied service from **Form Version Selector** dropdown list. In this example, the copied service is named **Custom A/B Testing Form Version Selector Trial**.



5. Click **Save** to update the changes.
6. Click **Edit** to elect only specific versions of your form.
7. Enter the nominated form versions, separated by commas (and without spaces), in the **Included Form Versions** field.

> If **Included Form Versions** is left blank, all form versions are included in the form A/B testing. This is the default option.



8. Click **Save** to update the changes. From now on, whenever a user selects this form, the Custom A/B Testing Form Version Selector service will randomly choose either 2.0.2-Trial2 or 2.0.1-Trial1 form version to be rendered to the users.

Next, learn how to view form A/B testing results.

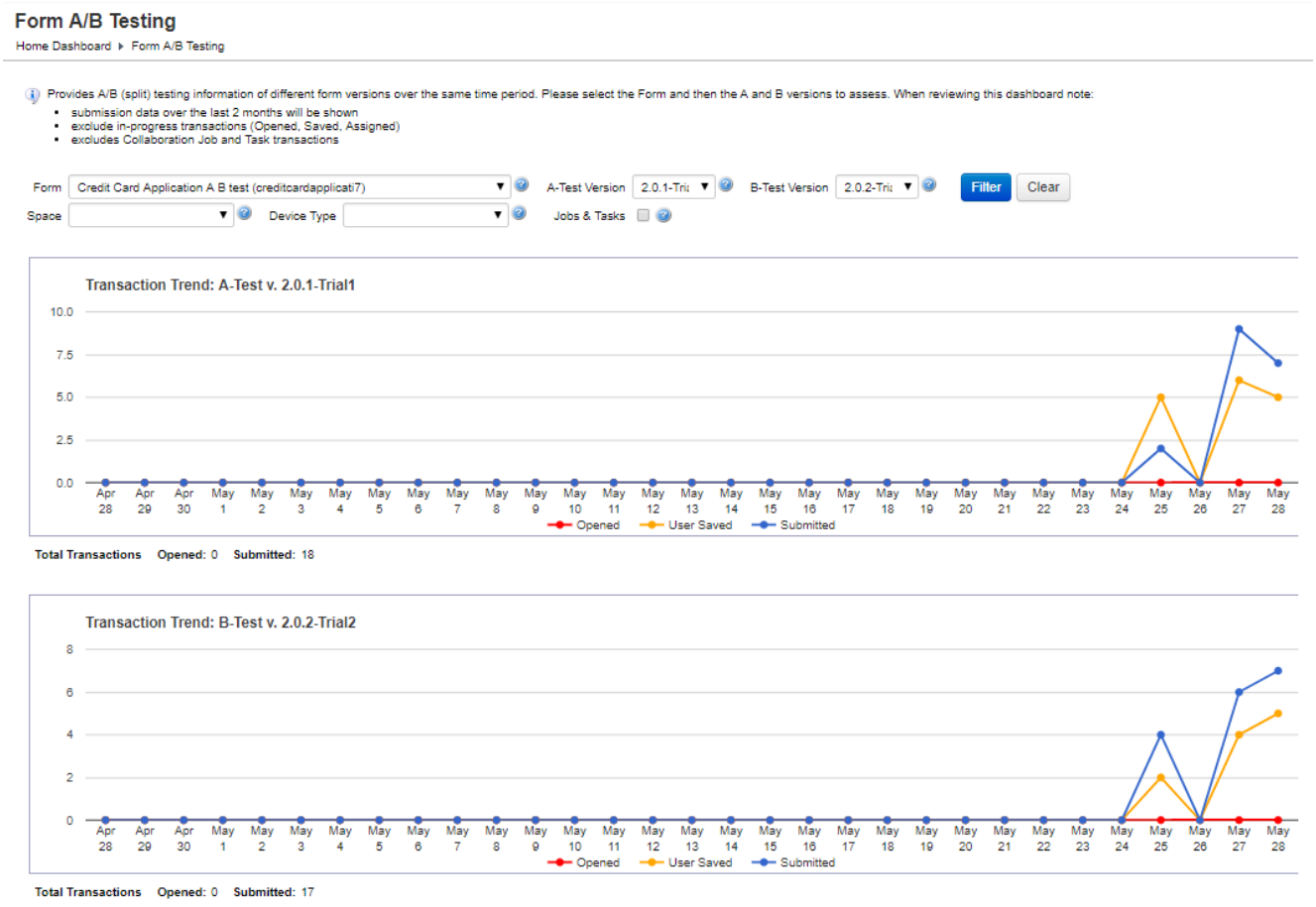
Analyze Form A/B Testing

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager provides out of the box analytics for A/B testing, which allow you to compare any two form versions, from the elected group of form versions for a duration of a form A/B testing trial. You can configure the form A/B testing at any time during the trial period.

To view the form A/B testing chart:

1. Select **Analytics > Form A/B Testing**.



2. Select an organization from the **Organization** dropdown list to retrieve statistics relevant to this organization only. If not selected, statistics for all organizations is shown.
3. Select a form from the **Form** dropdown.
4. Select a form version to represent the A-test results from the **A-Test Version** dropdown list.
5. Select a form version to represent B-test results from the **B-Test Version** dropdown list.
6. Select a form space from **Space** dropdown to include form transactions of this form space only.
7. Select a device from the **Device Type** dropdown to include only the transactions by users who accessed their form using this device.

A device type can be one of the following:

- Desktop
- Phone
- Tablet

8. Select the **Jobs & Tasks** checkbox to include collaboration job and task transactions, otherwise they are excluded.
6. Click **Filter** to generate the report. The comparison for each form A/B test is plotted as Opened, User Saved, and Submitted.

Next, learn about form A/B Testing analytics and reporting in Journey Analytics.

Analyze Forms Activity

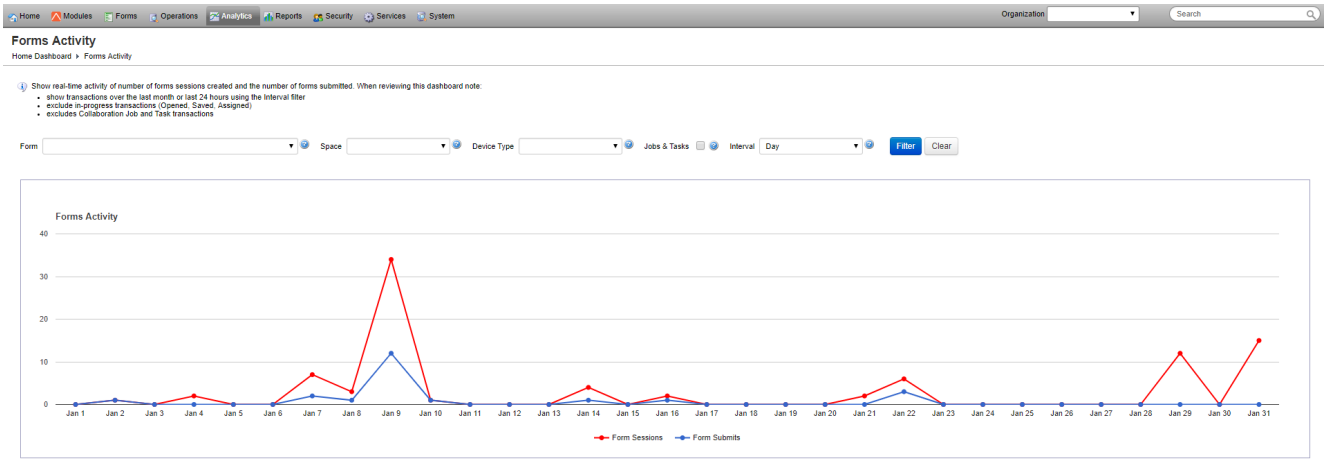
Journey Manager (JM) | System Manager / DevOps | 18.05

Manager generates this chart to visualize the real-time activity of the number of form sessions created and the number of forms submitted. The following assumptions are applied:

- The chart shows transactions over the last month or last 24 hours depending on the filter selection.
- The chart excludes in-progress transactions, such as opened, saved, assigned, collaboration job, and task transactions.

To view the forms activity chart:

1. Select **Analytics > Forms Activity**.



2. Select an organization from the **Organization** dropdown list to retrieve statistics relevant to this organization only. If not selected, statistics for all organizations is shown.
3. Select a form from the **Form** dropdown list to include the transactions related to that form only. If left blank, the transactions related to all forms are included in the chart.
4. Select a form space from the **Space** dropdown list to include the transactions of that form space only.
5. Select a device from the **Device** dropdown list to include transactions accessed by users from the following devices:
   - Desktop
   - Phone
   - Tablet
6. Select the **Jobs & Tasks** checkbox to include collaboration job and task transactions. Otherwise, they are excluded. | 18.05 .
7. Select an activity interval from the **Interval** dropdown list, which can be one of the following:
   - Day - to see the daily form activity for the last month. This is the default.
   - Hour - to see the hourly form activity for the last 24 hours.
8. Click **Filter** to generate the chart based on the selection criteria.

Next, learn about referrer submissions.

Analyze Device Type Submissions

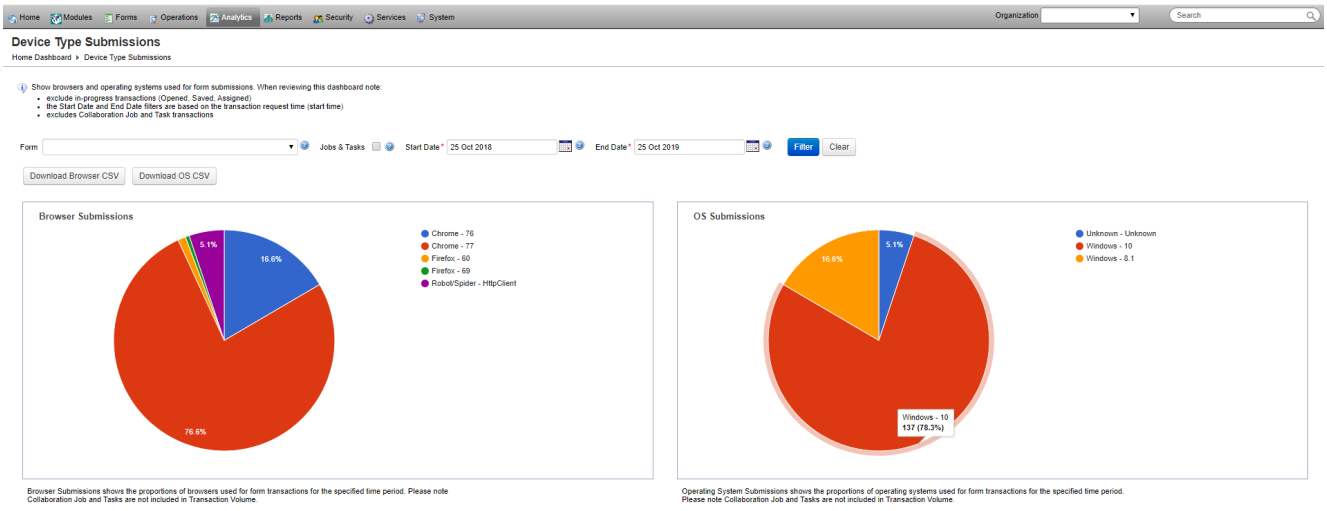
Journey Manager (JM) | Analytics User | 18.05

Manager generates charts to visualize the proportion of browsers and operating systems used for form transactions for a specified time period. The following assumptions are applied:

- Charts exclude in-progress transactions, such as opened, saved, assigned, collaboration job and task transactions.
- The Start Date and End Date filters are based on the transaction request time (start time).

To view the device types submissions charts:

1. Select **Analytics > Device Types Submissions**.



2. Select an organization from the **Organization** dropdown list to retrieve statistics relevant to this organization only. If not selected, statistics for all organizations is shown.
3. Select a form from the **Form** dropdown list to include the submissions of that form only. If left blank, submissions of all forms are included in the charts.
4. Select the **Jobs & Tasks** checkbox to include collaboration job and task transactions only. Otherwise, they are excluded. | 18.05 .
5. Specify the date range of submissions to be included in the chart from the **Start Date** and the **End Date**. By default, this is set to a date range starting one year before up to and including today.
6. Click **Filter** to generate the charts based on the selection criteria.
7. Click **Download Browser CSV** to export the browser submissions data as a CSV file.
8. Click **Download OS CSV** to export the operating system submissions data as a CSV file.

Manager prior to v18.05 has two separate charts: browser submissions and OS submissions.

Next, learn about reader versions.

Analyze Form Submissions

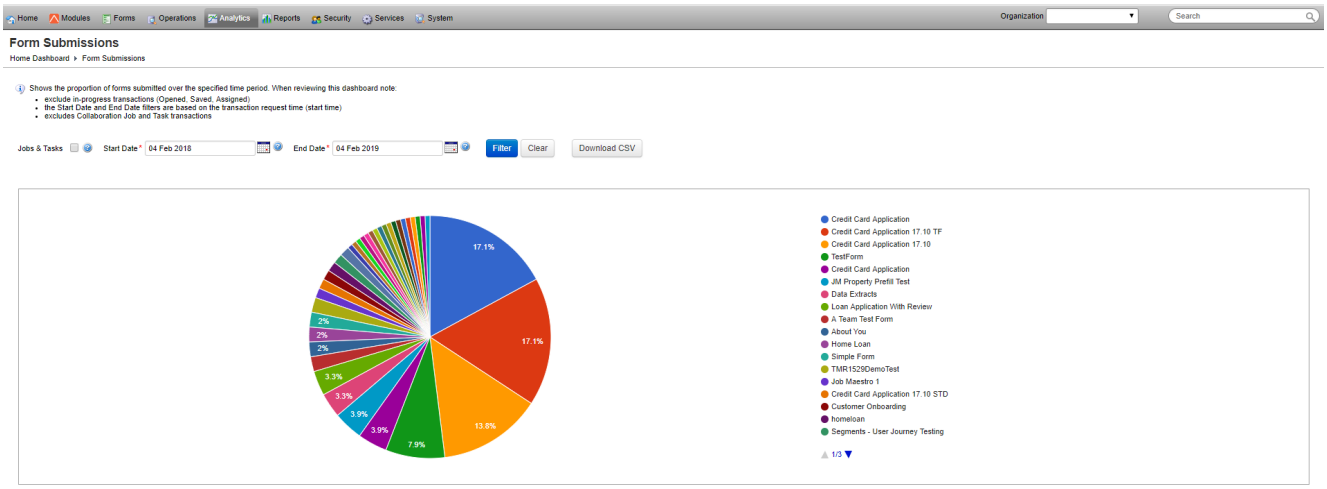
Journey Manager (JM) | System Manager / DevOps | 18.05

Manager generates this chart to visualize the most active forms and the proportion of forms submitted over the specified time period. The following assumptions are applied:

- Charts exclude in-progress transactions, such as opened, saved, assigned, collaboration job and task transactions.
- The Start Date and End Date filters are based on the transaction request time (start time).

To view the form submissions chart:

1. Select **Analytics > Form Submissions**.



2. Select an organization from the **Organization** dropdown list to retrieve statistics relevant to this organization only. If not selected, statistics for all organizations is shown.
3. Select the **Jobs & Tasks** checkbox to include collaboration job and task transactions only. Otherwise, they are excluded. | 18.05 .
4. Specify the date range of submissions to be included in the chart from the **Start Date** and the **End Date**. By default, this is set to a date range starting one year before up to and including today.
5. Click **Filter** to generate the chart based on the selection criteria.

You can see the form name and the total number of submissions made for each form by hovering the mouse over the chart slice.

The chart legend is shown on the right-hand side of the chart with forms ordered according to highest percentage of submissions.

6. Click **Download CSV** to export the displayed data as a CSV file.

> Manager prior to v17.10, uses **Export Excel Data** link instead of the **Download CSV** button.

Next, learn about device type submissions.

Analyze Submission Trend

Journey Manager (JM) | System Manager / DevOps | Removed in 18.05.

This functionality has been removed in Manager v18.05. Use transaction trend instead.

Manager generates charts to visualize the number of form requests compared to the number of completed submissions for each month during the selected date range.

To view the submission trend chart:

1. Select **Analytics > Submission Trend**.



2. Select a form from the **Form Name** dropdown list to see submission trends for a specific form only.
3. Specify the date range of submissions to be included in the chart from the **Start Date** and the **End Date**. By default, this is set to a date range starting one year before up to and including today.
4. Click **Refresh** to generate the charts based on the selection criteria.
5. Click to export the displayed data as an XLS file.

Next, learn about transaction trends.

Analyze Referer Submissions

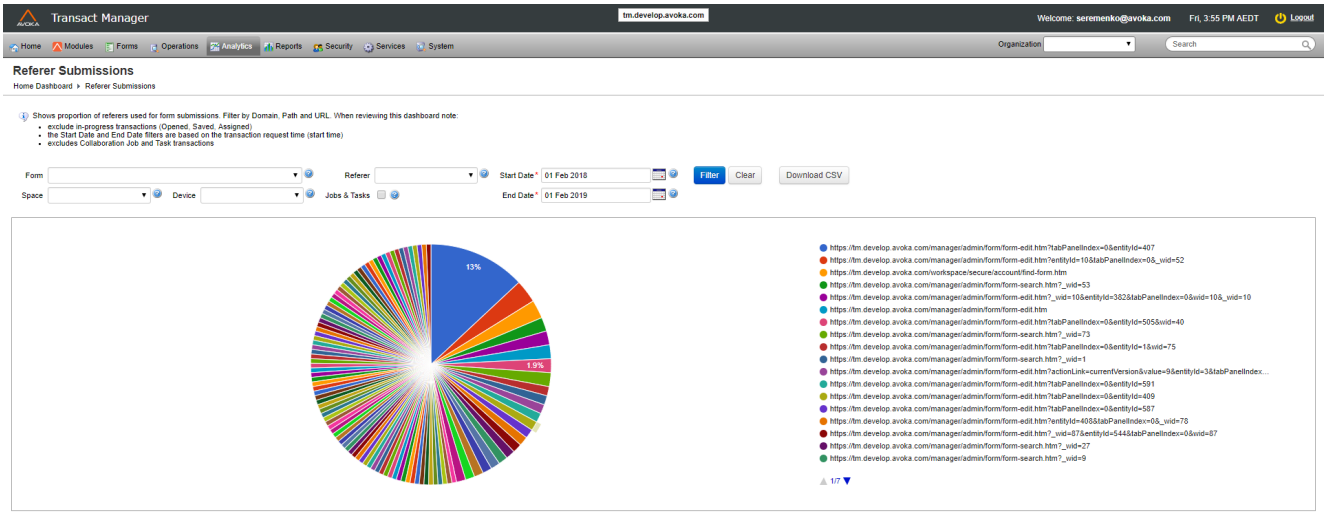
Journey Manager (JM) | System Manager / DevOps | 18.05

Manager generates this chart to visualize the top **referrer URLs**[1] used for form submissions. The following assumptions are applied:

- Charts exclude in-progress transactions, such as opened, saved, assigned, collaboration job and task transactions.
- The Start Date and End Date filters are based on the transaction request time (start time).

To view the referrer submissions chart:

1. Select **Analytics > Referrer Submissions**.



2. Select an organization from the **Organization** dropdown list to retrieve statistics relevant to this organization only. If not selected, statistics for all organizations is shown.
3. Select a form from the **Form** dropdown list to see **referers**[2] used for this form only. If left blank, referers of all forms are included in the charts.
4. Select a form space from the **Space** dropdown list to include transactions of that form space only.
5. Select a device from the **Device** dropdown list to include transactions accessed by users from the following devices:

---

[1]Referer URL is the URL that was used to navigate to the form.

[2]The HTTP referer (HTTP referrer) is an optional HTTP header field that identifies the address of the webpage that linked to the resource being requested. By checking the referrer, the new webpage can see where the request originated.

A device type can be one of the following:

- Desktop
- Phone
- Tablet

6. Select the **Jobs & Tasks** checkbox to include collaboration job and task transactions only. Otherwise, they are excluded. | 18.05 .
7. Specify the date range of submissions to be included in the chart from the **Start Date** and the **End Date**. By default, this is set to a date range starting one year before up to and including today.
8. Click **Filter** to generate the chart based on the selection criteria.
9. Click **Download CSV** to export the displayed data as a CSV file.

Manager prior to v17.10, uses **Export Excel Data** link instead of the **Download CSV** button.

Next, learn about form submissions.

Analyze Browser Submissions

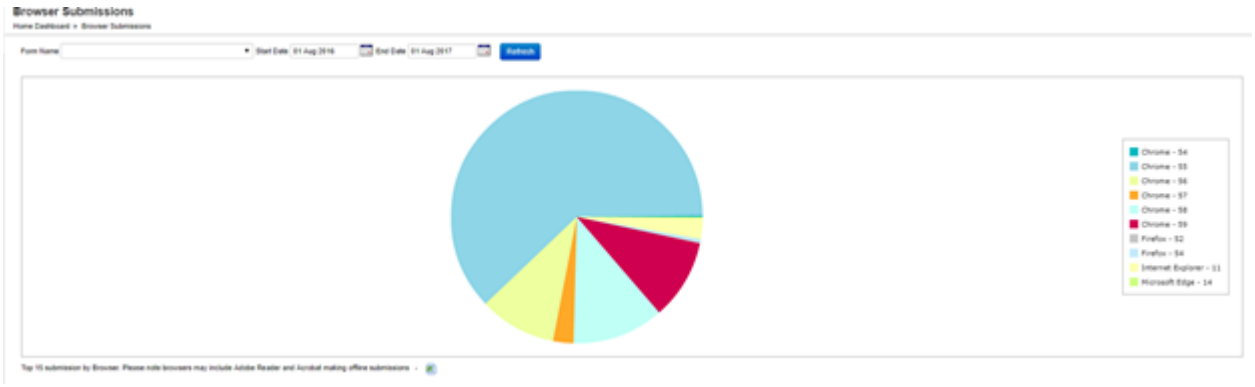
Journey Manager (JM) | System Manager / DevOps | Removed in 18.05.

This functionality was removed in Manager v18.05. Use transaction trend instead.

Manager generates browser submissions charts to visualize the number of the internet browsers most used by form users.

To view Browser Submissions chart:

1. Select **Analytics > Browser Submissions**.



2. Select a form from the **Form Name** dropdown list to see submission trends for a specific form only.
3. Specify the date range of submissions to be included in the chart from the **Start Date** and the **End Date**. By default, this is set to a date range starting one year before up to and including today.
4. Click **Refresh** to generate the charts based on the selection criteria.
5. Click to export the displayed data as an XLS file.

Next, learn about transaction trends.

Analyze OS Submissions

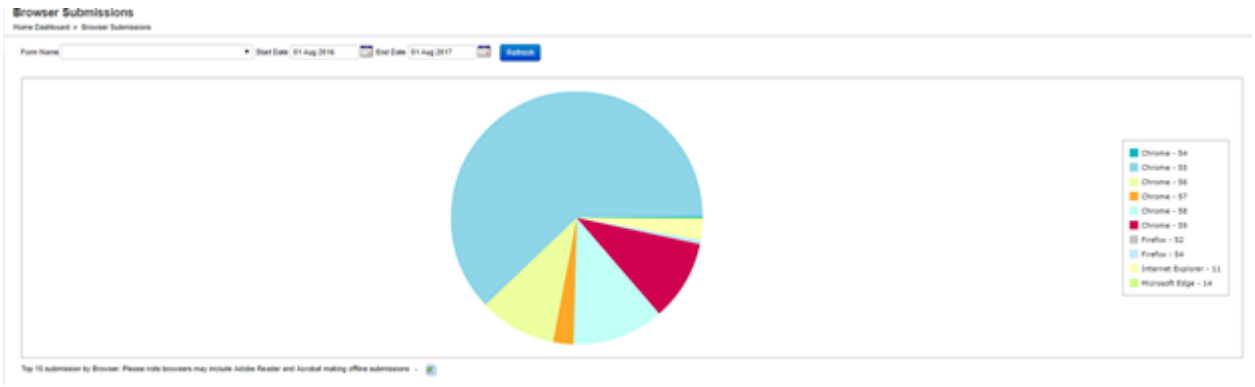
Journey Manager (JM) | System Manager / DevOps | Removed in 18.05.

This functionality has been removed in Manager v18.05. Use device type submissions instead.

Manager generates charts to visualize the operating systems most used by form users.

To view the operating system submissions chart:

1. Select **Analytics > Operating Systems Submissions**.



2. Select a form from the **Form Name** dropdown list to see submission trends for a specific form only.
3. Specify the date range of submissions to be included in the chart from the **Start Date** and the **End Date**. By default, this is set to a date range starting one year before up to and including today.
4. Click **Refresh** to generate the charts based on the selection criteria.
5. Click to export the displayed data as an XLS file.

Next, learn about device type submissions.

Analyze Reader Versions

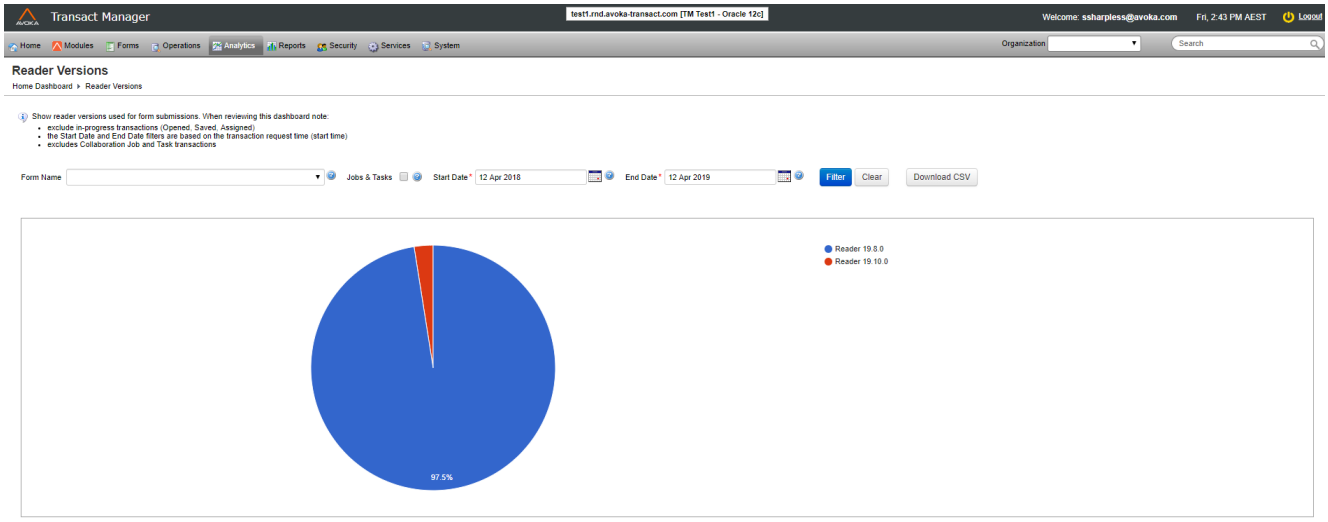
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager generates this chart to visualize a breakdown of the most common Adobe Reader versions used to submit forms. The following assumptions are applied:

- Charts exclude in-progress transactions, such as opened, saved, assigned, collaboration job and task transactions.
- The Start Date and End Date filters are based on the transaction request time (start time).

To view the form submissions chart:

1. Select **Analytics > Reader Versions**.



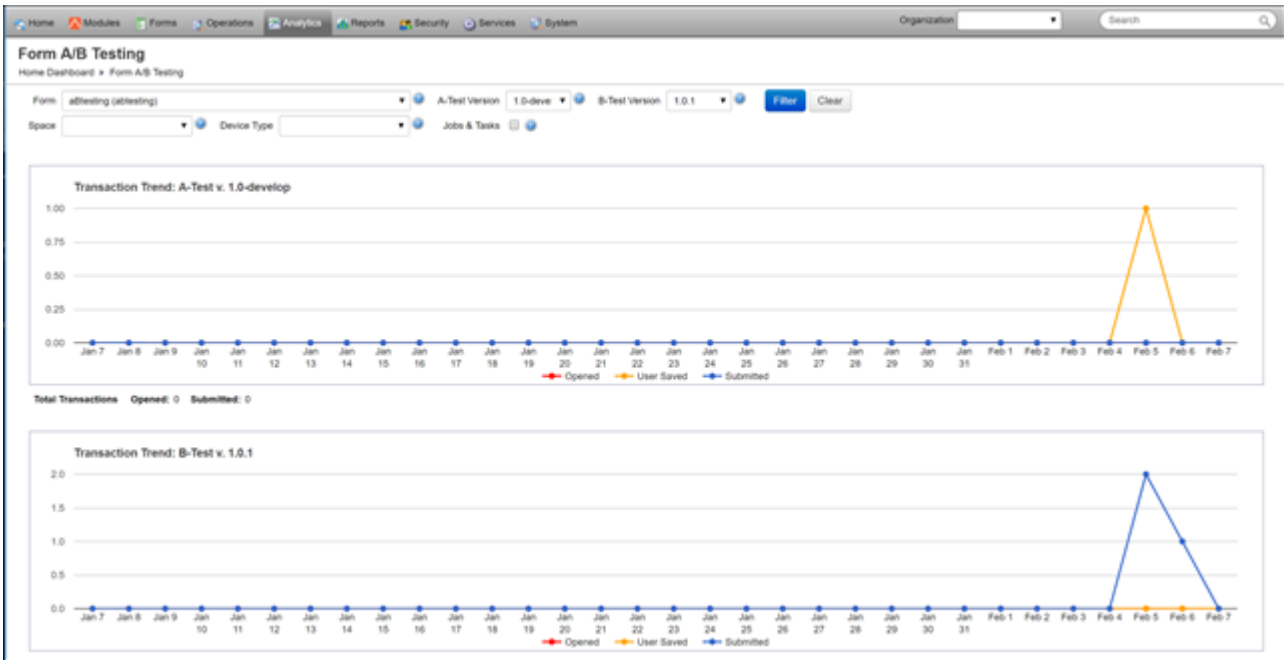
2. Select an organization from the **Organization** dropdown list to retrieve statistics relevant to this organization only. If not selected, statistics for all organizations is shown.
3. Select a form from the **Form** dropdown list to include the submissions of that form only. If left blank, submissions of all forms are included in the charts.
4. Select the **Jobs & Tasks** checkbox to include collaboration job and task transactions only. Otherwise, they are excluded. | 18.05 .
5. Specify the date range of submissions to be included in the chart from the **Start Date** and the **End Date**. By default, this is set to a date range starting one year before up to and including today.
6. Click **Filter** to generate the charts based on the selection criteria.
7. Click **Download CSV** to export the displayed data as a CSV file.

Next, learn about transaction trends.

Maestro and Composer Forms A/B Testing

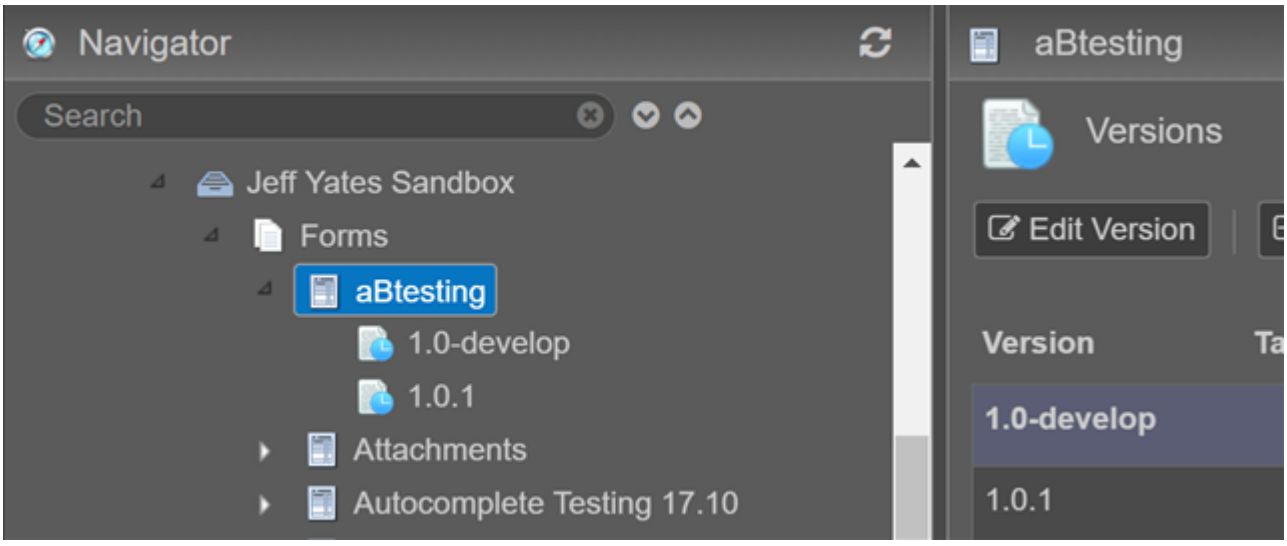
Journey Manager (JM) | System Manager / DevOps | 18.05

First thing to do is to determine a need to do A/B Testing at all. Avoka Insights does an excellent job of highlighting problems in the User Experience that a change in Form design may address. Or, more proactively, investigating a new Form feature can involve A/B Testing, to determine that the new additions won't stop end users from submitting.



Maestro

To test multiple versions of a Form, the versions must of course first be created. As an example, a Form named aBtesting was made. It is a default Maguire Template form, with two Text Fields added.



In this example, the Text Fields are arranged horizontally in the first version, and vertically in the second. This is similar to A/B Testings intended use case how does a possibly small change to a form impact the user experience, as seen in completion rates.

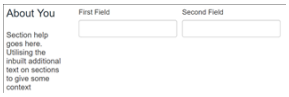


Figure 1 - the "A" design

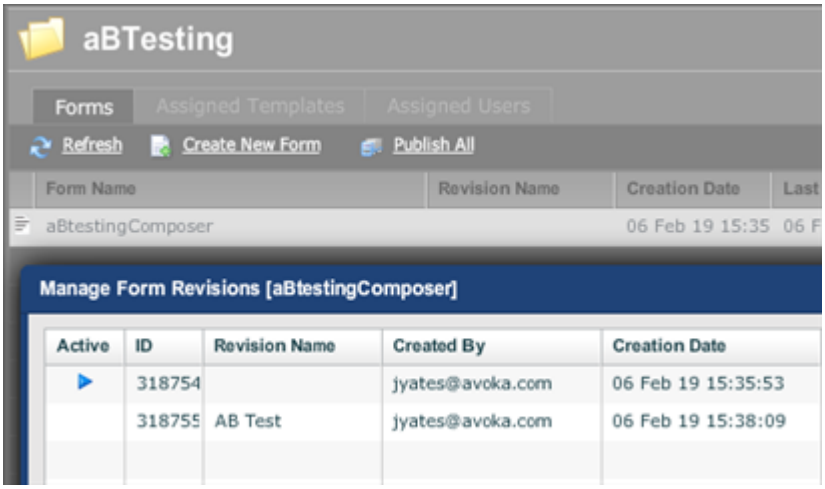


Figure 2- the "B" design

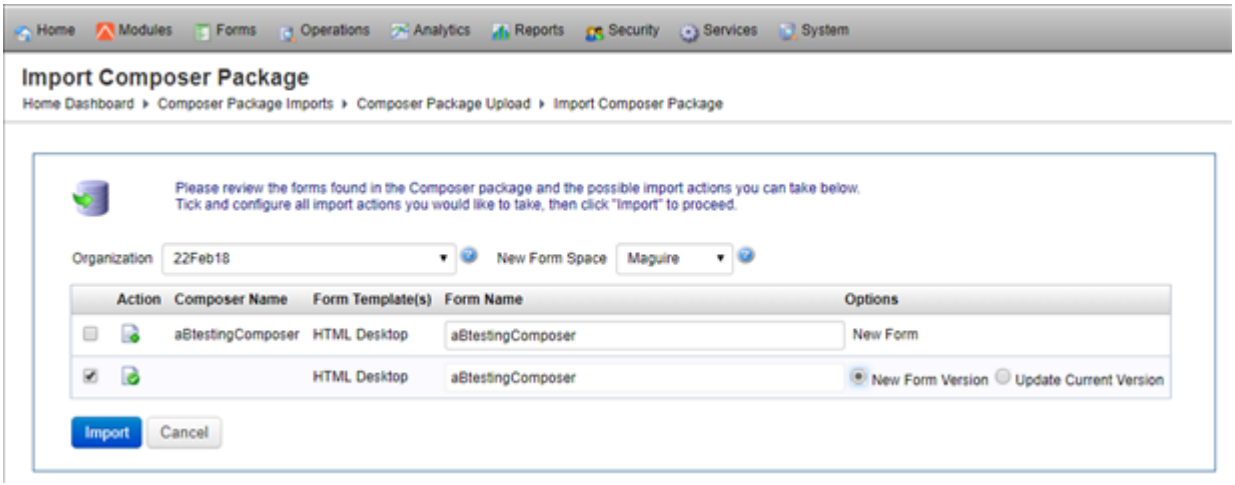
Build and deploy both versions to Transact as you normally would.

Composer

The same concepts apply when using Composer as the form designer. Build the first design, the A version, and publish it to a ZIP file. Then use Composers Form Revisions feature to create the B version. Publish the B version to a ZIP file as well.

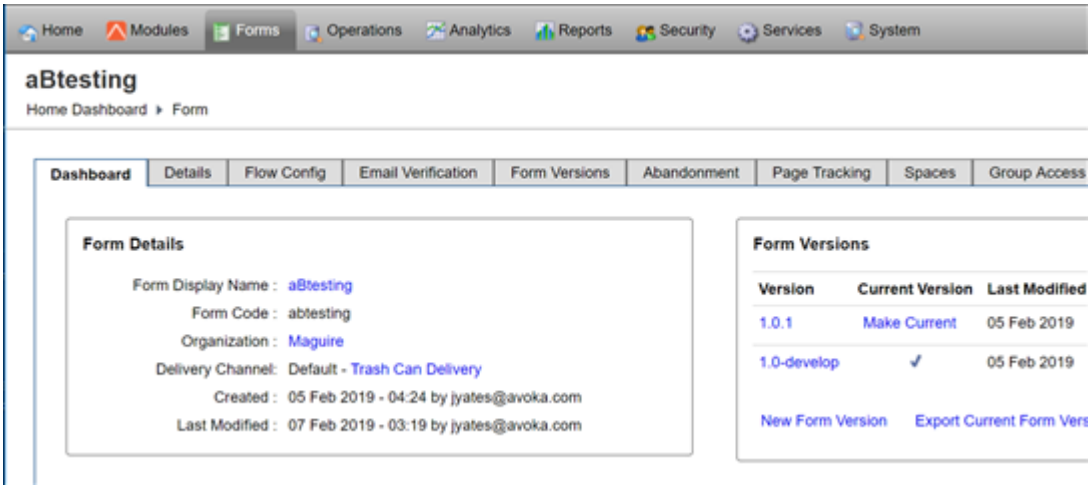


Import both ZIP files to Transact using the Import Composer Package feature. You will create a new form for the first ZIP file imported, and a new version of that form when you import the second ZIP file.



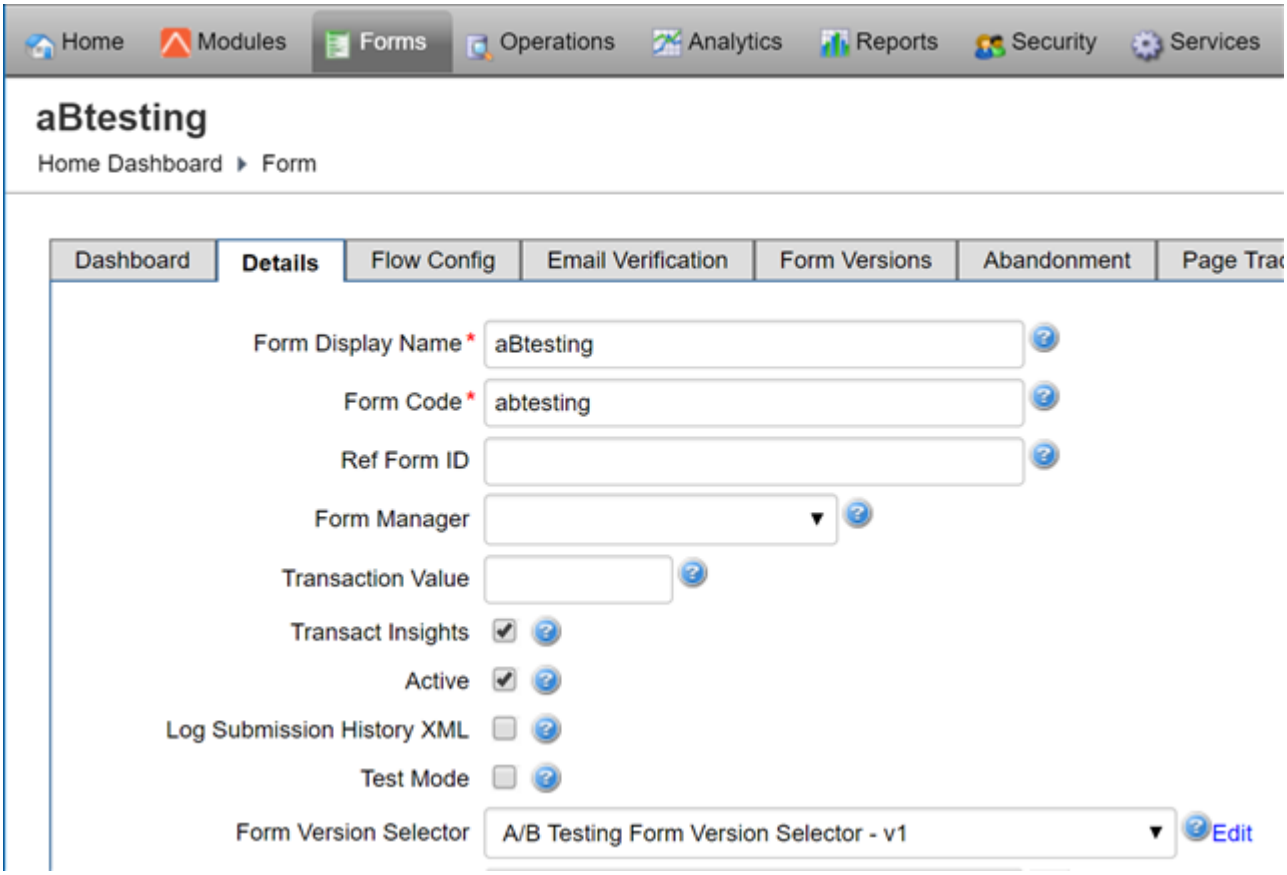
Manager

Once deployed to Transact, the Form dashboard would look like shown below .

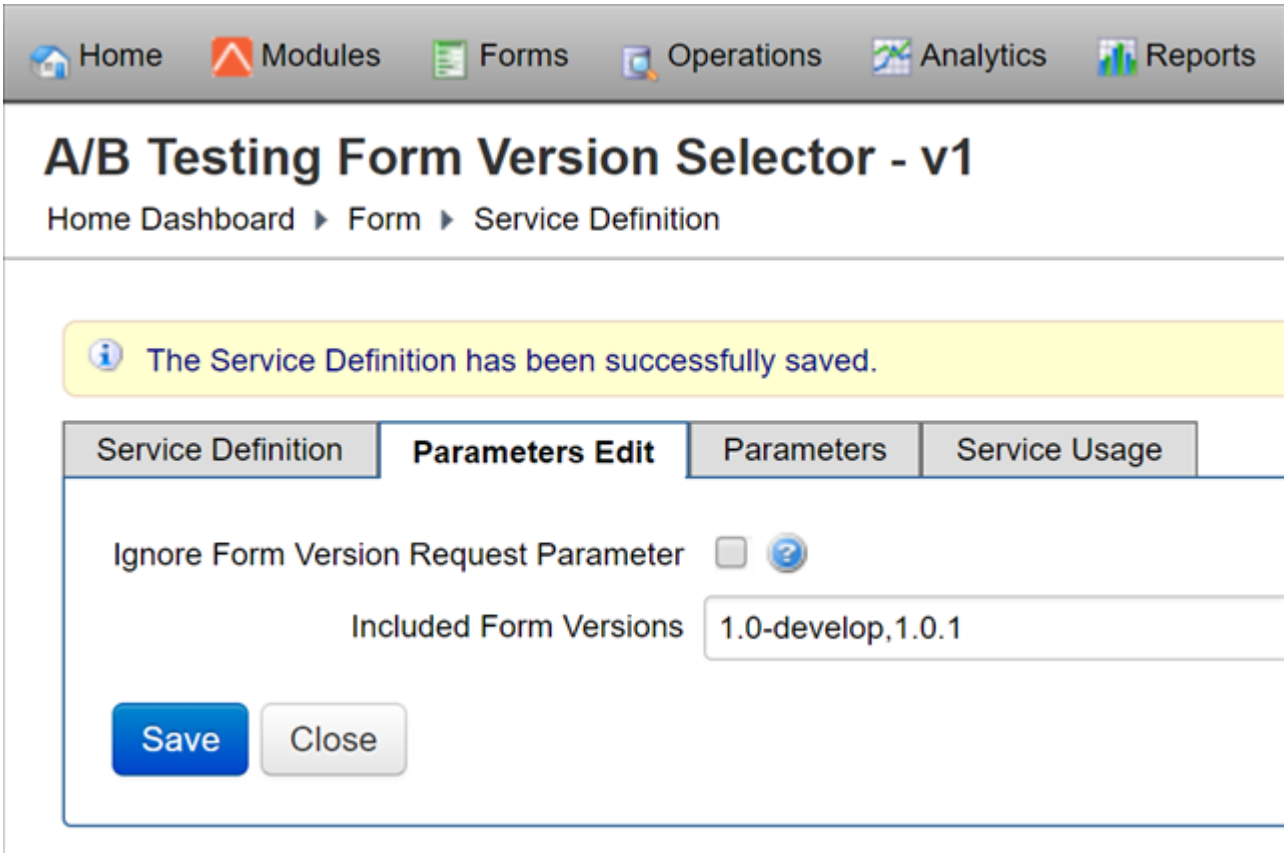


This shows that both Form versions are deployed. For A/B testing, it does not matter which version is marked as Current Version. In fact, neither needs to be current. The Form Version Selector Service will chose which version to render.

The next step is to then add a Form Version Selector Service to the Form. Switch to the Details tab and add one.



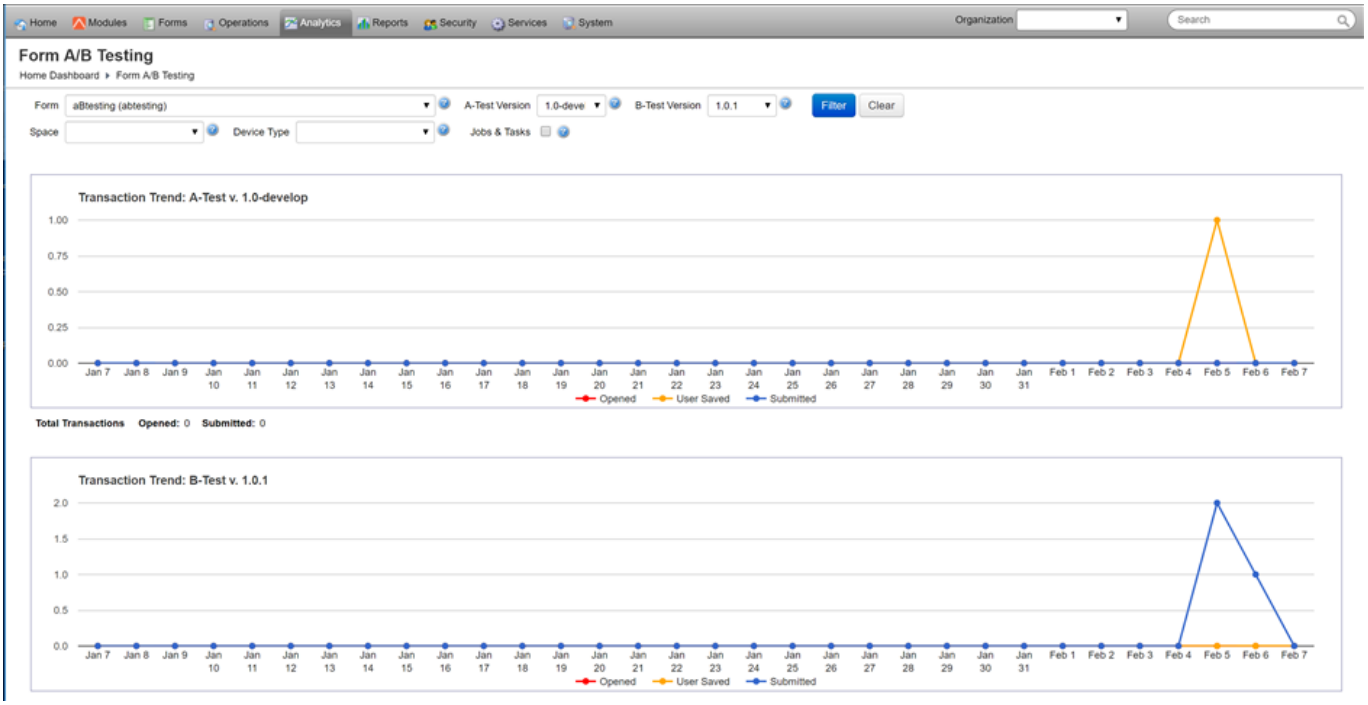
The Form Version Selector Service shown above is a default service. In practice, a copy of this Service should be made and that copy used for this Form's testing. This needs to be done to allow the Service to be configured with the Form Version numbers specific to this Form. Click the Edit link to setup the Form Version Selector Service.



On this screen you add the Form version numbers, separated by commas, that need to be tested. Note that A/B Testing can actually be used against three or more different Form Versions!
The Form Version Selector Service will randomly choose from the supplied list of versions, evenly distributing the selections. So in our example, version 1.0-develop will be selected 50% of the time, and version 1.0 will be selected 50% of the time.
If one of the versions needs to be presented more often than the other, include it in the list more than once. Using 1.0-develop,1.0.1,1.0.1 will result in version 1.0.1 being selected 2/3 of the times.
That's all that's needed. The A/B Testing is now live and functioning. Accessing the Form's base url will trigger the Form Version Selector Service, and it will return a randomly selected version. Normally the base url (like https://training.avoka.com/web-plugin/servlet/SmartForm.html?formCode=abtesting) would return the Form Version marked Current.

Performing the Tests, Reviewing Results

Now that the A/B Test is live and running, let it collect data. There is nothing additional to do to perform the test other than let the system run. After a suitable amount of time (to be determined by each project team for each form Forms under heavy use may only need a day's data, others may need a week), review the results and turn off the test.



The Analytics menu item on Transact's main menu contains the Form A/B Testing link. This page will show a graph of the relative performance of the entire Form, one graph for each version tested (as shown above). Of course, Insights can also be used to review the results of A/B Testing. And with Insights, you can see differences between the two Form Version's performance on a much more granular level.
After reviewing the results, the A/B Test should be shut off. Go back to the Form Dashboard tab and mark the proper Form Version as Current. Then on the Details tab, clear the Form Version Selector setting.

What is implied above is that this is a production server test. Live Forms being used by end users. So all versions tested this way must go through all of the normal development processes full testing on lower environments, etc. After all, the desired goal is to find a new production Form design, that removes friction from the user experience.

Next, learn about form A/B Testing analytics and reporting in Journey Analytics.

Reports Overview

Journey Manager (JM) | System Manager / DevOps | 21.05

Manager automatically records licensing data, which includes organizations, forms, transactions, and services, and generates a license usage report for each organization.

Next, learn how to view job licensing report.

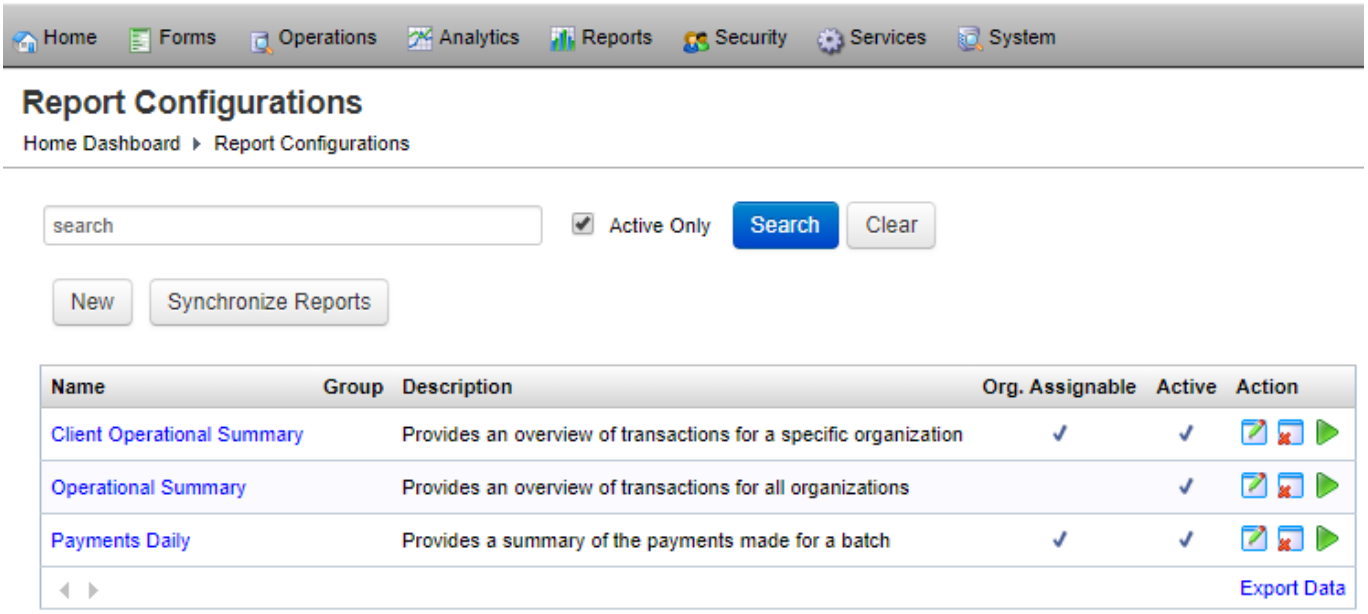
Report Configurations

Journey Manager (JM) | System Manager / DevOps | Removed in 19.05.

Manager, prior to version 19.05, allows you to view, create and edit reports.

> You can try using the standard Job Licensing and Transaction Licensing reports before creating your own.

To view report configurations, select **Reports > Report Configurations**.



To filter reports by one or more criteria, specify the following settings and click **Filter**:

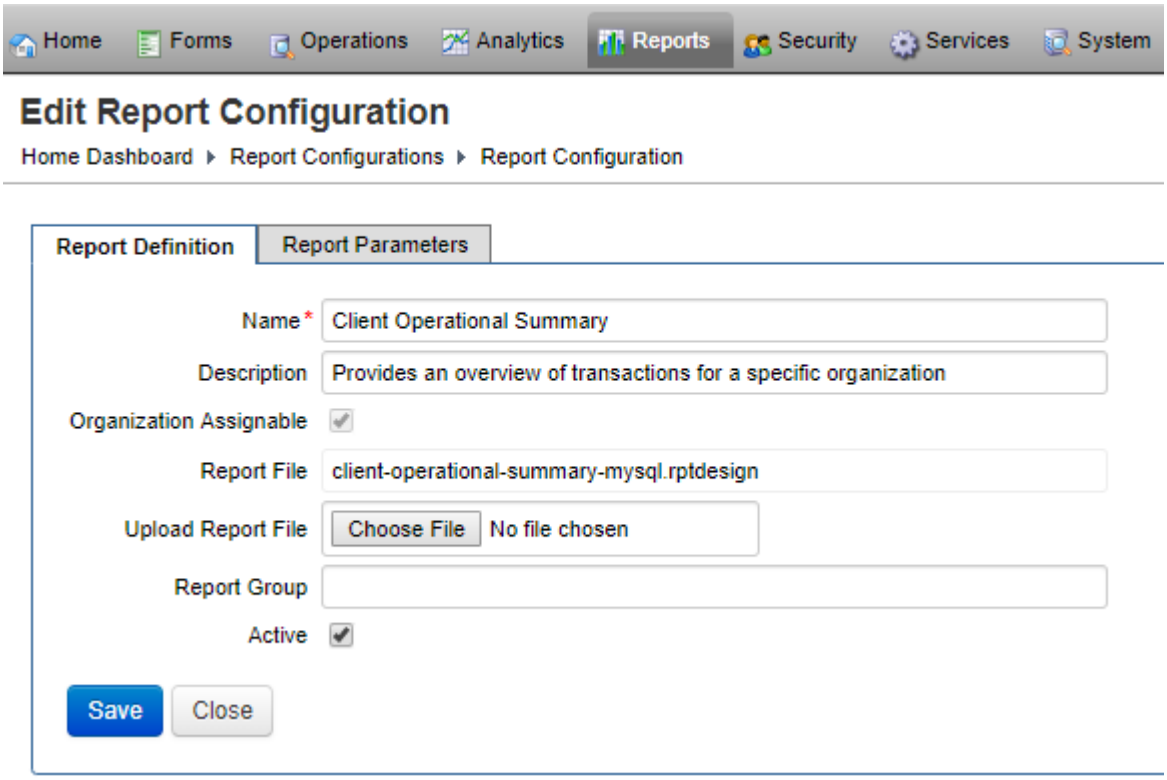
- A report name or part of the name
- Active Only

Click **Clear** to reset the search criteria.

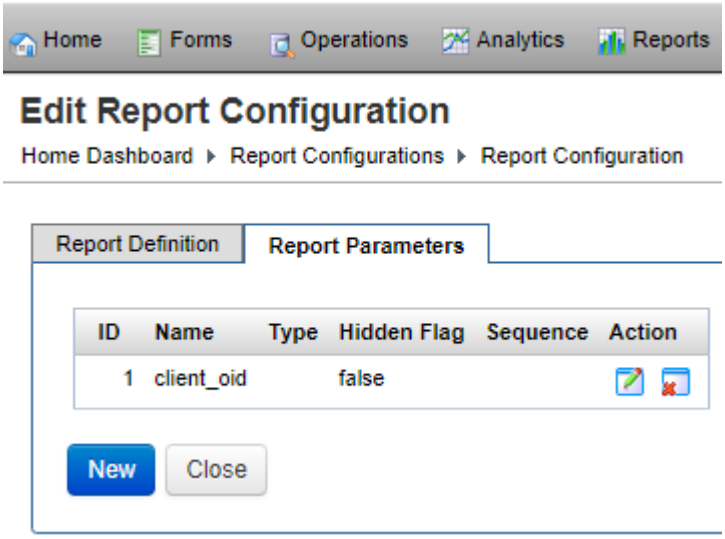
Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

To configure a report:

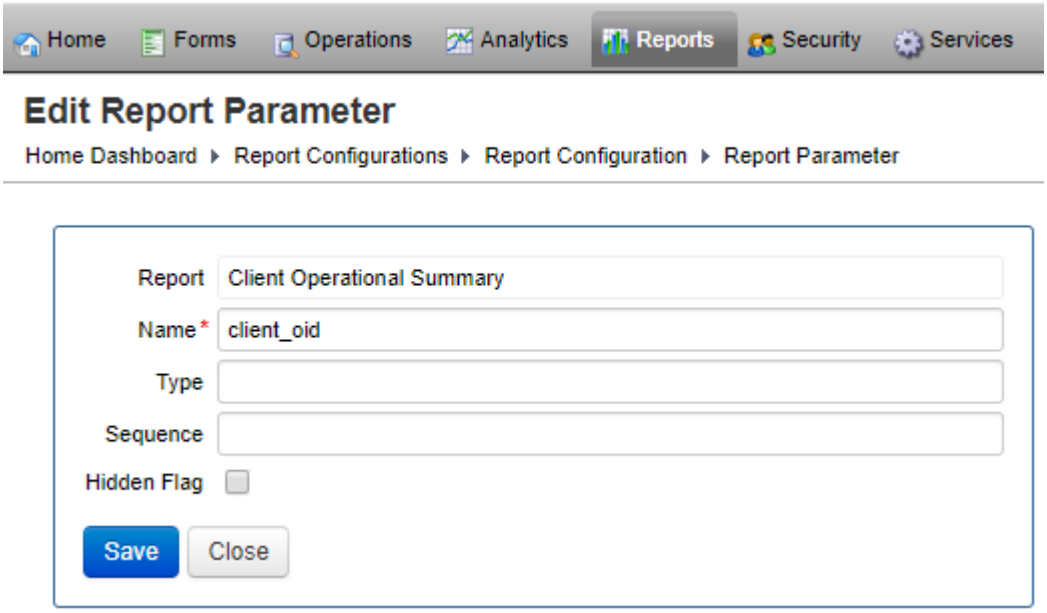
1. Select **Reports > Report Configurations**.
2. Locate a report and click **Edit** or click **New** to create a new report.



3. Edit the report name in the **Name** field.
4. Provide a report description in the **Description** field.
5. Select the **Organization Assignable** checkbox to make the report available to individual organizations.
6. Click **Choose File** to upload a report file.
7. Specify the **Report Group**.
8. Select the **Active** checkbox to make it the active report.
9. Click **Save** to update the changes.
10. Select the **Report Parameters** tab to create or configure report parameters.



11. Locate a parameter and click **Edit** or click **New** to create a new parameter.



12. Edit the parameter name in the **Name** field.
13. Provide the parameter type in the **Type** field.
14. Update the sequence number in the **Sequence** field.
15. Select the **Hidden Flag** checkbox to make it hidden.
16. Click **Save** to update the changes.

To remove a report:

1. Select **Reports > Report Configurations**.
2. Locate a report and click **Remove Report** to delete a report.

Click **Synchronize Report** to synchronize reports.

Click **View the Report Using the Report Viewer** to see the rendered report.

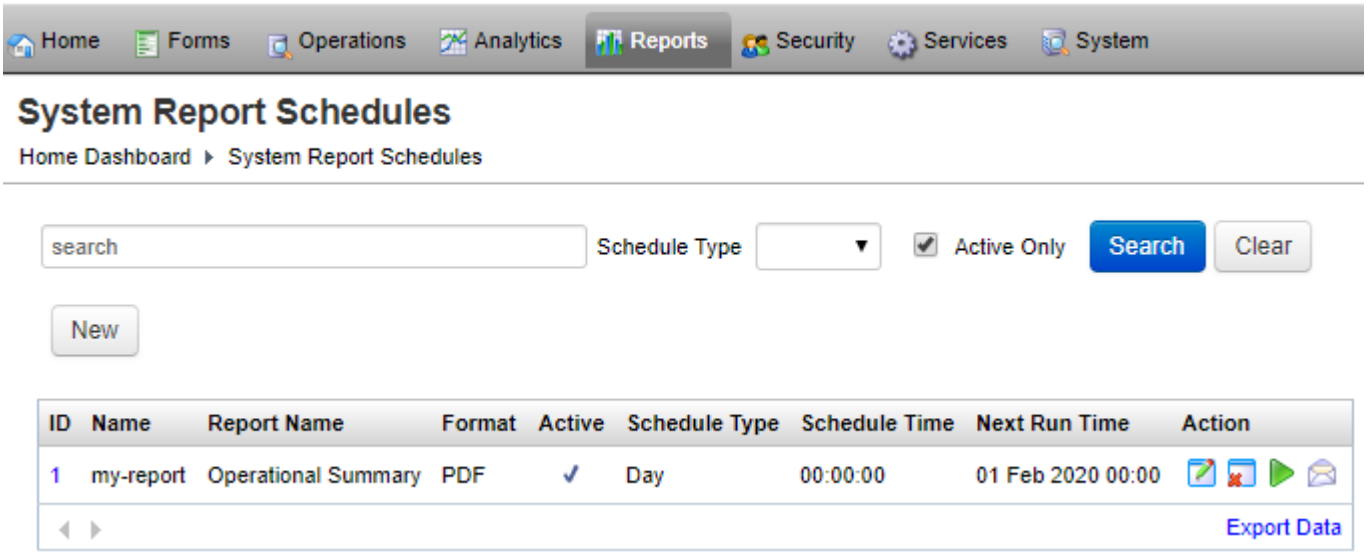
Next, learn about system report schedules.

System Report Schedules

Journey Manager (JM) | System Manager / DevOps | Removed in 19.05.

Manager allows you to view, create new and edit existing system report schedules.

To view system report schedules, select **Reports > System Report Schedules**.



To filter reports by one or more criteria, specify the following settings and click **Filter**:

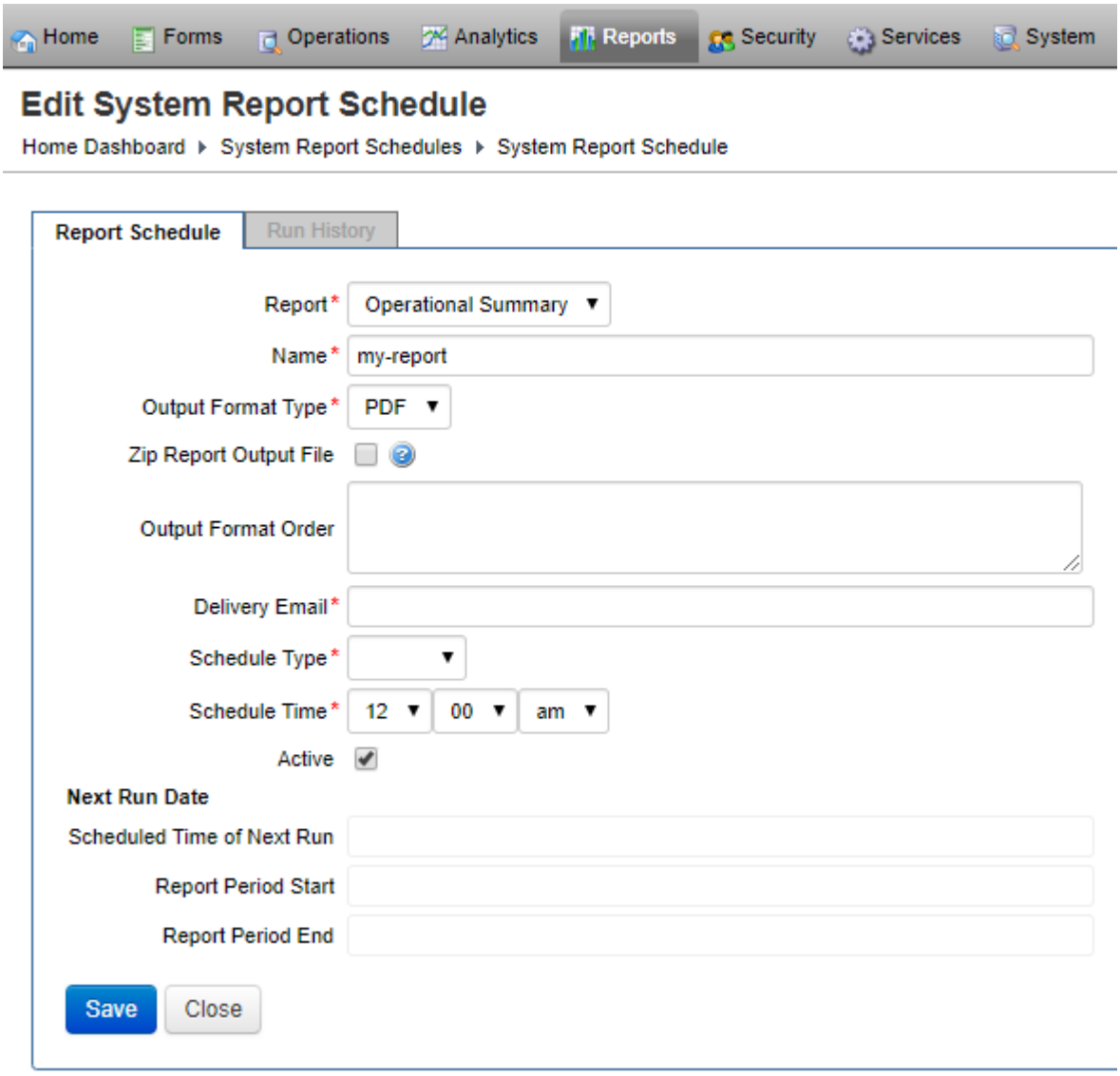
- A scheduler type
- Active Only

Click **Clear** to reset the search criteria.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

To configure a system report scheduler:

1. Select **Reports > System Report Schedules**.
2. Locate a report and click **Edit** or click **New** to create a new report.



3. Select a report from the **Report** dropdown list.
4. Edit the report name in the **Name** field.
5. Select an output format type from the **Output Format Type** dropdown list.
6. Select the **Zip Report Output File** checkbox to ZIP the generated report.
7. Specify the output format order in the **Output Format Order** field.
8. Provide a valid email address in the **Delivery Email** field.
9. Select a scheduler type from the **Schedule Type** dropdown list.
10. Specify scheduler time from the **Schedule Time** dropdown lists.
11. Select the **Active** checkbox to make this scheduler active.
12. Click **Save** to update the changes.

To remove a system report scheduler:

1. Select **Reports > System Report Schedules**.
2. Locate a report and click **Remove Report Scheduler** to delete a scheduler.

Click **View the Report On-Screen** to see the rendered report.

Click **Trigger the report scheduler** to run the scheduler

Next, learn how to view job licensing report.

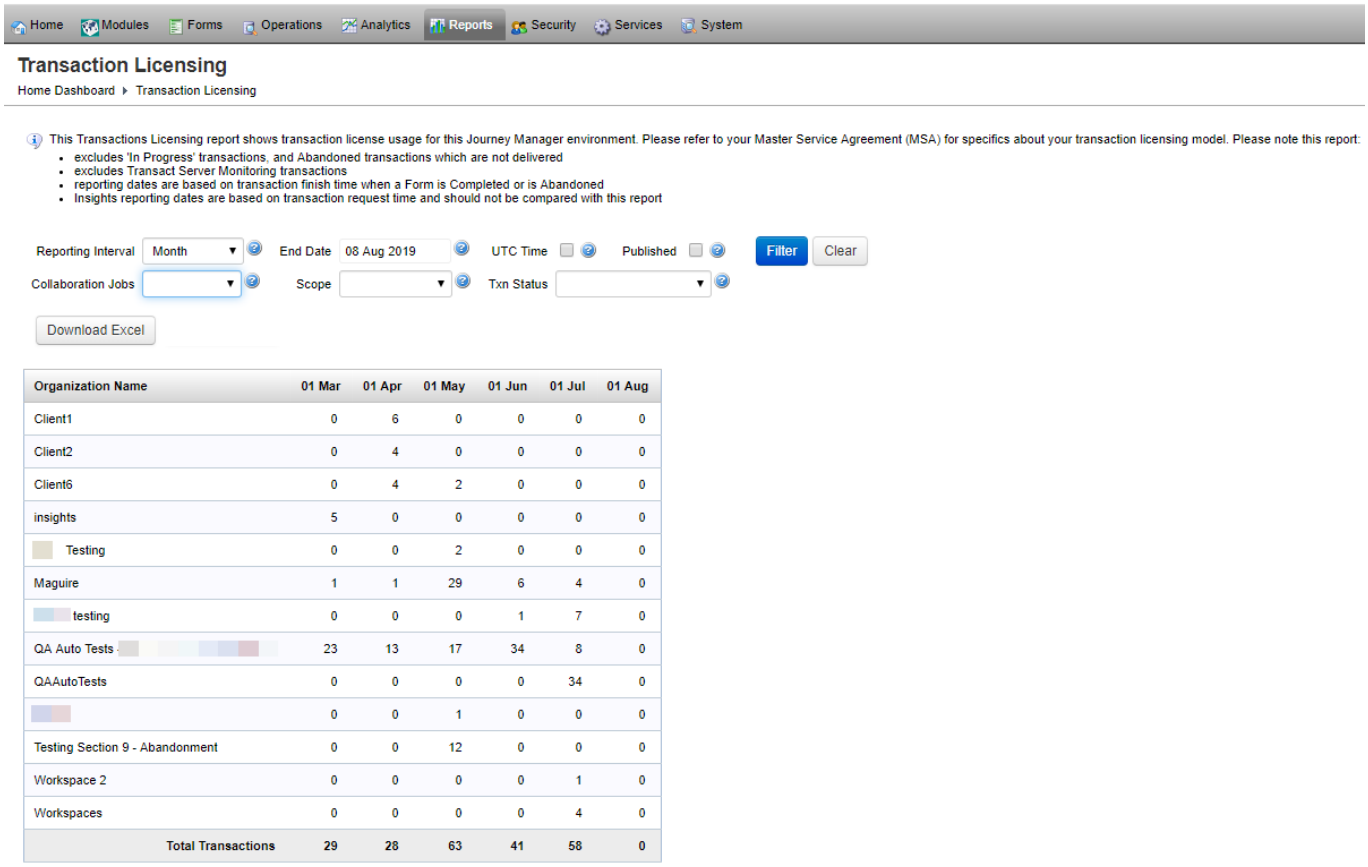
View Transaction Licensing Report

Journey Manager (JM) | System Manager / DevOps | 21.05

Manager automatically records licensing data, which includes organizations, forms, transactions, and services, and generates a license usage report for each organization.

> By default, the transaction licensing report is run nightly.

To view the transaction licensing report, select **Reports > Transaction Licensing**.



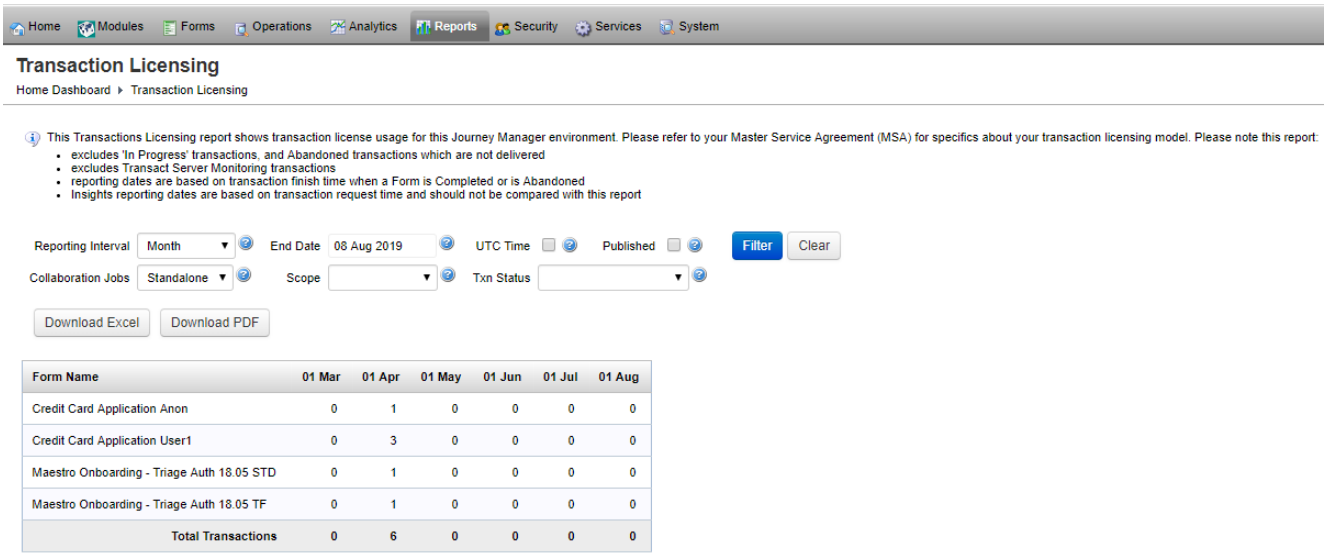
Organization Name	01 Mar	01 Apr	01 May	01 Jun	01 Jul	01 Aug
Client1	0	6	0	0	0	0
Client2	0	4	0	0	0	0
Client6	0	4	2	0	0	0
insights	5	0	0	0	0	0
Testing	0	0	2	0	0	0
Maguire	1	1	29	6	4	0
testing	0	0	0	1	7	0
QA Auto Tests	23	13	17	34	8	0
QAAutoTests	0	0	0	0	34	0
	0	0	1	0	0	0
Testing Section 9 - Abandonment	0	0	12	0	0	0
Workspace 2	0	0	0	0	1	0
Workspaces	0	0	0	0	4	0
Total Transactions	29	28	63	41	58	0

To filter the transaction licensing report by one or more criteria, specify the following settings and click **Filter**:

- Select the reporting intervals as Day, Week, or Month from the **Reporting Interval** dropdown list.
- Select a collaboration job type from the **Collaboration Jobs** dropdown list:
  - **Jobs Txns** - to show transactions associated with a collaboration job.
  - **Standalone** - to show standalone transactions which are not part of a collaboration job.
  - Blank - to show both types of transactions. This is the default.
- Edit the **End Date** of the reporting period.
- Select a transaction type from the **Scope** dropdown list:
  - **Internal Txns** - to show transactions made by internal users.
  - **External Txns** - to show transactions made by external users, who are customers and applicants.
  - Blank - to show both types of transactions. This is the default.

You can change what type of licensing transactions a form space supports.

- Select the **UTC Time** checkbox to use the **Coordinated Universal Time**[1] (UTC) in the report.
- Select the **Published** checkbox to display transactions that have been published to Temenos. Otherwise, all transactions are included in the report.
- Select whether the report displays completed transactions or delivered abandoned transactions in the **Txn Status** dropdown list. Otherwise, all transactions are included in the report.
- Select an organization from the **Organization** dropdown list to show the transaction breakdown for each form belonging to this organization, as shown below.



Click **Clear** to reset the search criteria.

Click **Download Excel** to download a copy of this report as a Microsoft spreadsheet XLS file.

---

[1] Coordinated Universal Time (UTC) is the primary time standard by which the world regulates clocks and time.

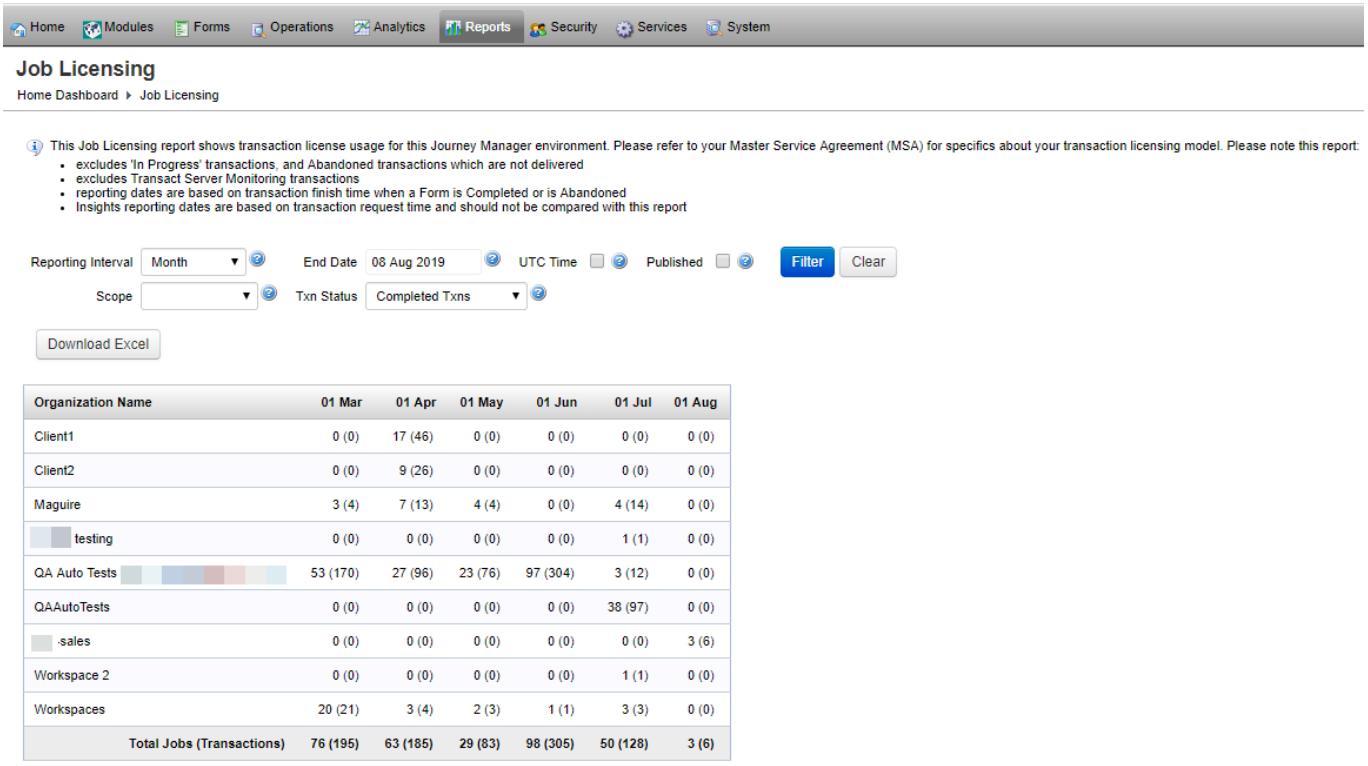
Next, learn how to view job licensing report.

View Job Licensing Report

Journey Manager (JM) | System Manager / DevOps | 21.05

Journey Manager allows you to view job licensing data for transactions that were completed using a collaboration job. You can view this information in the job licensing report.

To view the job licensing report, select **Reports > Job Licensing**.



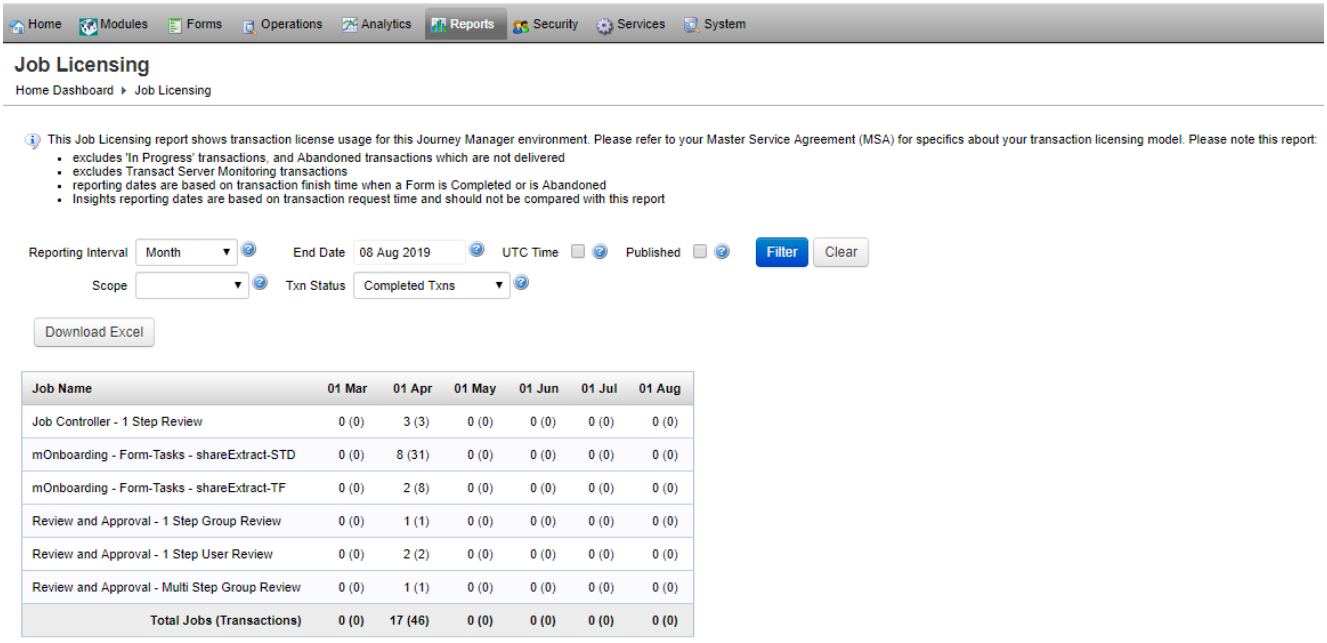
Organization Name	01 Mar	01 Apr	01 May	01 Jun	01 Jul	01 Aug
Client1	0 (0)	17 (46)	0 (0)	0 (0)	0 (0)	0 (0)
Client2	0 (0)	9 (26)	0 (0)	0 (0)	0 (0)	0 (0)
Maguire	3 (4)	7 (13)	4 (4)	0 (0)	4 (14)	0 (0)
testing	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	0 (0)
QA Auto Tests	53 (170)	27 (96)	23 (76)	97 (304)	3 (12)	0 (0)
QAAutoTests	0 (0)	0 (0)	0 (0)	0 (0)	38 (97)	0 (0)
sales	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	3 (6)
Workspace 2	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	0 (0)
Workspaces	20 (21)	3 (4)	2 (3)	1 (1)	3 (3)	0 (0)
Total Jobs (Transactions)	**76 (195)**	**63 (185)**	**29 (83)**	**98 (305)**	**50 (128)**	**3 (6)**

To filter the job licensing report by one or more criteria, specify the following settings and click **Filter**:

- Select the reporting intervals as Day, Week, or Month from the **Reporting Interval** dropdown list.
- Edit the **End Date** of the reporting period.
- Select a transaction type from the **Scope** dropdown list:
  - **Internal Txns** - to show transactions made by internal users.
  - **External Txns** - to show transactions made by external users, who are customers and applicants.
  - Blank - to show both types of transactions. This is the default.

You can change what type of licensing transactions a form space supports.

- Select the **UTC Time** checkbox to use the **Coordinated Universal Time**[1] (UTC) in the report.
- Select the **Published** checkbox to display transactions that have been published to Temenos. Otherwise, all transactions are included in the report.
- Select whether the report displays completed transactions or delivered abandoned transactions in the **Txn Status** dropdown list. Otherwise, all transactions are included in the report.
- Select an organization from the **Organization** dropdown list to show the transaction breakdown for each form belonging to this organization, as shown below.



Click **Clear** to reset the search criteria.

Click **Download Excel** to download a copy of this report as a Microsoft spreadsheet XLS file.

Next, learn how to view transaction licensing report.

---

[1] Coordinated Universal Time (UTC) is the primary time standard by which the world regulates clocks and time.

Create a Custom Report

Journey Manager (JM) | System Manager / DevOps | Removed in 19.05.

Manager allows you to create and present a custom report.

A quite common requirement at the end stages of a project is to create "on-demand" reports on the data within TM. Reports help management and operational users clearly see data that could even be PII data. One benefit of creating reports that are presented within a formspace is that we are not responsible for the delivery of PII data. The results are displayed on screen or within a CSV file and it's up to the end user to safely and securely download and store the file. Reports can be heavily customized to show data from submission objects, data extracts, or form xml. Remember that the form xml will be purged based on data retention settings. Reports can also be quite processor heavy so special care should come with load testing.

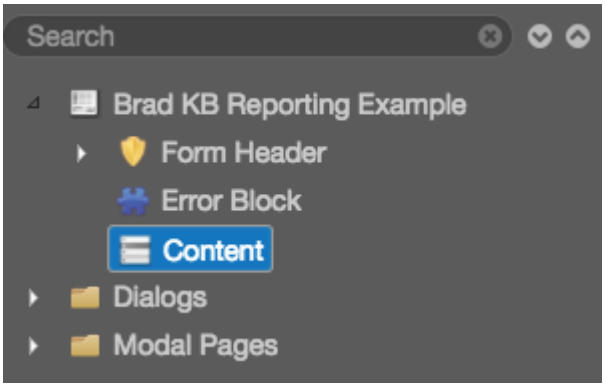
Let's walk though a very simple report that uses the Core API to query submissions. Typically I use the Core API because the Fluent API doesn't have a start and end date as part of the querying criteria. Start and end dates are a standard for most reporting criteria. Most reports are based on submissions and all DAOs query off of submission date except for the TransactionHistoryDAO which queries off of form start/opened date.

The following example only shows how to display 1 report but the architecture is setup in a way to easily add more reports.

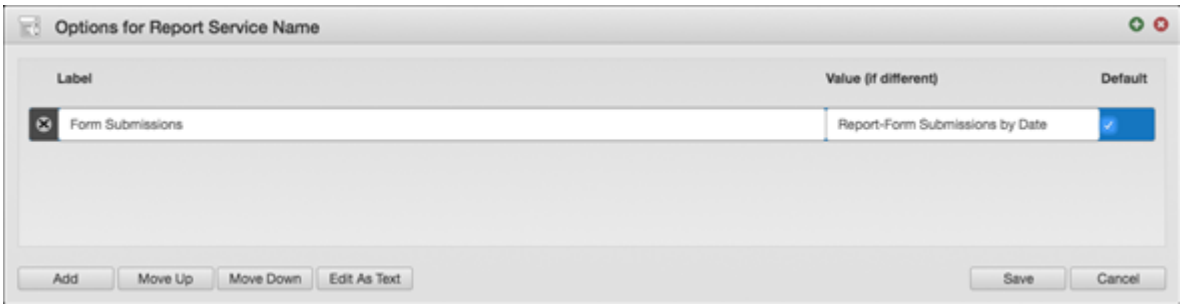
Form

After creating a new blank form with your customer's branding:

1. Delete all non-essential components, especially the submit and save buttons. We don't want any transactions getting submitted to Manager or the customer will incur a fee.
   - On the Maguire form I deleted: Wizard Nav Bar, Form Footer, Function Bar, Navigation Second Level, Navigation Chevrons, Job Details Block, and everything within the "Content" Page Controller.



2. Add a Page named"Reporting" to the Content Page Controller.
3. Add a Section named"Report Criteria" to your "Reporting" Page.
4. Add a dropdownnamed "Report Service Name" to your "Report Criteria" Section. We will only have one report but this is where you can add multiple reports if required.

- Add one option to the dropdown. Make the label "Form Submissions" and the value has to be the name of the Groovy Service that creates the report data. We'll call our service "Report-Form Submissions by Date".
- 

5. Add a block called "Criteria1" to the "Reporting Criteria" Section. The reason for the block is to add visibility rules if you have different reports with different criteria.
6. Add a 2 date fields called "Start Date" and "End Date".
   - Prefill the "Start Date" and "End Date" with today's date.
   - Add Validation that they must be Past Dates and Include Today.
   - Make both fields Mandatory.
     - Add the following validation rule to "End Date" (optional):

```
if (Calc.daysBetween(data.startDate, data.endDate) > 7)
return "Date range cannot exceed 7 days.";
```

7. Add a Text Display field beside the dates that states the valid date range. i.e. "Date range cannot exceed 7 days."
8. Add a dropdown called "Output Format" with 2 options: CSV and HTML to the "Reporting" Section.
9. Add a "Run Report" button.
10. Add a Block called "OutputBlk" to house all the hidden data and output fields.
11. Add a Text Display field with "Output records have been returned." to the "OutputBlk".
12. Add a "recordsReturned", "rawData", and "outputReturned" Data Fieldsto the "OutputBlk".
13. Add a "Download CSV"Button to the "OutputBlk" with the following Visibility rule: **`data.outputreturned === "CSV"`**
14. Add a "Show HTML" Button to the "OutputBlk" with the following Visibility rule: **`data.outputreturned === "HTML"`**
15. At this point, all the necessary fields have been added. Now it's time to insert the logic to the fields.

16. Add the following "Click" script to the "Run Report" button:

```
var runReport = function()
{
data.rawdata = null;
data.recordsreturned = 0;
data.outputreturned = data.outputFormat;
Form.showProgress("Your report is being generated...");
var inputData = {
startDate: data.startDate,
endDate: data.endDate,
reportServiceName: data.reportServiceName,
outputType: data.outputFormat
}
DynamicData.call("GenerateDynamicDataReport", inputData).then(
function(response) {
Form.showProgress("");
console.log(response);
data.rawdata = response.data;
data.recordsreturned = response.recordsReturned;
}).catch(
function(err) {
Form.showProgress("");
console.log(err);
}
);
}
Form.validate("data.reportCriteria").then(
function(response) {
if (response.valid == true) {
runReport();
} else {
return;
}
}
).catch(
function(err) {
console.log(err);
}
);
```

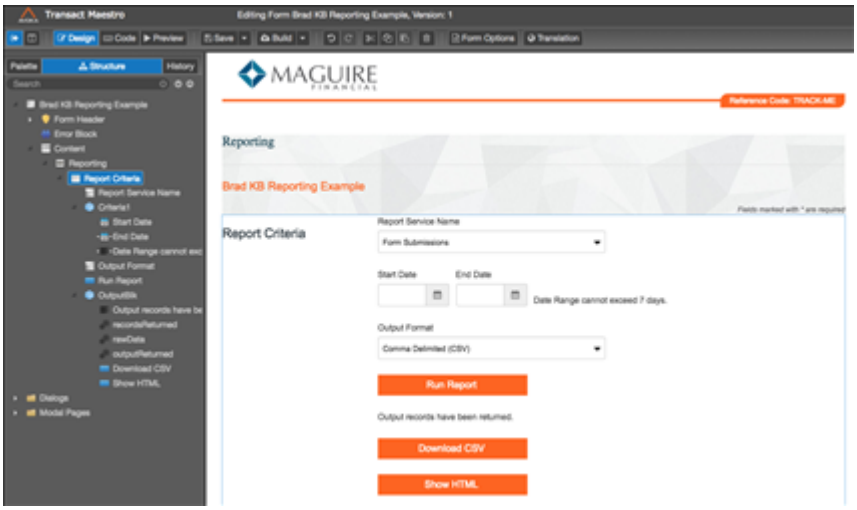
17. Add the following Click script to the "Download CSV" Button:

```
function requiredFeaturesSupported()
{
return ( BlobConstructor() && msSaveOrOpenBlobSupported() );
}
function BlobConstructor()
{
return (window.Blob)
}
function msSaveOrOpenBlobSupported()
{
return (window.navigator.msSaveOrOpenBlob)
}
function b64toBlob(b64Data, contentType, sliceSize)
{
contentType = contentType || '';
sliceSize = sliceSize || 512;
var byteCharacters = atob(b64Data);
var byteArrays = [];
for (var offset = 0; offset < byteCharacters.length; offset += sliceSize) {
var slice = byteCharacters.slice(offset, offset + sliceSize);
var byteNumbers = new Array(slice.length);
for (var i = 0; i < slice.length; i++) {
byteNumbers[i] = slice.charCodeAt(i);
}
var byteArray = new Uint8Array(byteNumbers);
byteArrays.push(byteArray);
}
var blob = new Blob(byteArrays, {type: contentType});
return blob;
}
if (requiredFeaturesSupported()) {
blobObject = b64toBlob((data.rawdata), "text/csv");
window.navigator.msSaveOrOpenBlob(blobObject, 'report.csv');
} else {
var a = document.body.appendChild(document.createElement("a"));
a.download = "report.csv";
a.href = "data:text/csv;charset=utf-8; base64," + data.rawdata;
a.innerHTML = "download report csv";
a.hidden = true;
a.click();
}
```

18. Add the following code to the "Show HTML" Button:

```
if (data.rawdata) {
var win = window.open("", "Report", "toolbar=no, location=no, directories=no, status-
s=no, menubar=no, scrollbars=yes, resizable=yes, width=780, height=200, top="+
(screen.height-400)+", left="+(screen.width-840));
win.document.body.innerHTML = data.rawdata;
}
```

All done... now it's time to write the services! Here's what it should look like:



Groovy Script

The form is designed to call one generic DDS with the name of the report service as a parameter. There are 3 services which sort of represent a typical 3 layer java framework of "Service Layer", "Manager Layer", and "DAO Layer". Only the "Manager Layer" has to change in most situations since the Submission object that gets returned gives you access to everything you'll need.

1. GenericDynamicDataReport - This is the main DDS service that the form calls. It's just passes on the parameters to the right Groovy Service. In a perfect situation you would also handle any bubbled exceptions here too.

```
import com.avoka.core.groovy.GroovyLogger as logger
import com.avoka.fc.core.service.ServiceLocator
import com.avoka.fc.core.dao.*
import com.avoka.fc.core.entity.*
import groovy.json.*
String reportServiceName = request.getParameter('reportServiceName')
String startDate = request.getParameter('startDate')
String endDate = request.getParameter('endDate')
String outputType = request.getParameter('outputType')
def reportService = ServiceLocator.getServiceForName(reportServiceName)
Map<String,Object> result = reportService.invokeService([
startDate:startDate,
endDate:endDate,
outputType:outputType
])
return JsonOutput.toJson(result)
```

2. Report-Form Submissions by Date - This is the name of the service that you configured in your Maestro form. It has to match perfectly with the value of the dropdown in "Report Service Name" and the parameters have to match as well. This service assembles the data returned from the next DAO call. This is where you can get really creative and return back submission properties, data extracts, form XML, etc... But this is where you also have to be careful with performance. The more introspecting and filtering you do the slower your report will be. The "filterData" function.

```
import com.avoka.core.groovy.GroovyLogger as logger
import com.avoka.fc.core.dao.*;
import com.avoka.fc.core.entity.*;
import com.avoka.fc.core.service.ServiceLocator
import groovy.json.*
DATEFORMAT = 'yyyy-MM-dd'
String formCode = serviceParameters.formCode // You'll have to add a service para-
meter or hardcode your formcode here.
String startDate = parameters.startDate
String endDate = parameters.endDate
String outputType = parameters.outputType
String encodeCsv = parameters.encodeCsv ?: 'false'
//logger.info "-------------------------------------------------"
//logger.info serviceDefinition.getServiceName()
//parameters.each { logger.info "$it.key: $it.value" }
//logger.info "-------------------------------------------------"
def reportHeaders = createReportHeaders()
def reportData = retrieveData(formCode, startDate, endDate)
reportData = filterData(reportData)
def output = null
if (outputType.toUpperCase() == "CSV") {
String csvString = createCsvReport(reportHeaders, reportData)
if (encodeCsv == 'true')
output = Base64.getEncoder().encodeToString(csvString.getBytes())
else
output = csvString
} else if (outputType.toUpperCase() == "HTML") {
StringBuilder reportHtmlStr = new StringBuilder();
reportHtmlStr.append startHtmlBody()
reportHtmlStr.append createHtmlReport(formCode, reportHeaders, reportData,
startDate, endDate)
reportHtmlStr.append endHtmlBody()
output = reportHtmlStr.toString()
}
return ["data":output, "recordsReturned": reportData.size]
/*******************
* HELPER FUNCTIONS
*******************/
def createReportHeaders()
{
def reportHeaders = ["ApplicationId" // 1
,"Submit Date" // 2
```

```
,"Submit Time" // 3
,"Current Status" // 4
,"Last Updated" // 5
]
return reportHeaders
}
def retrieveData(formCode, startDate, endDate)
{
def getSubmissionDeliveryListService = ServiceLocator.getServiceForName("GetSub-
missionDeliveryList")
def results = getSubmissionDeliveryListService.invokeService(["formCode":formCode,
"startDate":startDate, "endDate":endDate])
return results
}
def filterData(results)
{
StringBuilder dataStr = new StringBuilder();
def dataTable = [][]
def filteredResults = results.sort { it.timeSubmission }.reverse()
filteredResults.eachWithIndex { it, i ->
dataTable[i] = [
it.trackingNumber, // 1
it.timeSubmission.format(DATEFORMAT), // 2
it.timeSubmission.format(TIMEFORMAT), // 3
it.job?.currentStep?.name, // 4
it.job?.currentStep?.timeCreated.format(DATETIMEFORMAT) // 5
]
}
return dataTable
}
def createHtmlReport(formCode, reportHeaders, dataTable, startDate, endDate)
{
StringBuilder dataStr = new StringBuilder();
dataStr.append "<p>Applications submitted for ${formCode} from ${startDate} to
${endDate}</p>"
dataStr.append "<table>"
dataStr.append "<tr>"
reportHeaders.each {
dataStr.append "<th>${it}</th>"
}
dataStr.append "</tr>"
dataTable.each{ it1 ->
```

```
dataStr.append "<tr>"
it1.each {
dataStr.append "<td>${it ?: ""}</td>"
}
dataStr.append "</tr>"
}
dataStr.append "</table>"
return dataStr.toString()
}
def startHtmlBody()
{
StringBuilder dataStr = new StringBuilder();
dataStr.append "<!DOCTYPE html><html>"
dataStr.append '''<head>
<style type="text/css">
body {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 10px;
}
table {
border-collapse: collapse;
}
table, th, td {
font-size: 10px;
padding: 3;
border: 1px solid black;
}
td {
color: #898989;
font-size: 10px;
padding: 3;
text-align: right;
}
</style>
</head>'''
dataStr.append "<body>"
return dataStr.toString()
}
def endHtmlBody()
{
return "</table> </body> </html>"
}
```

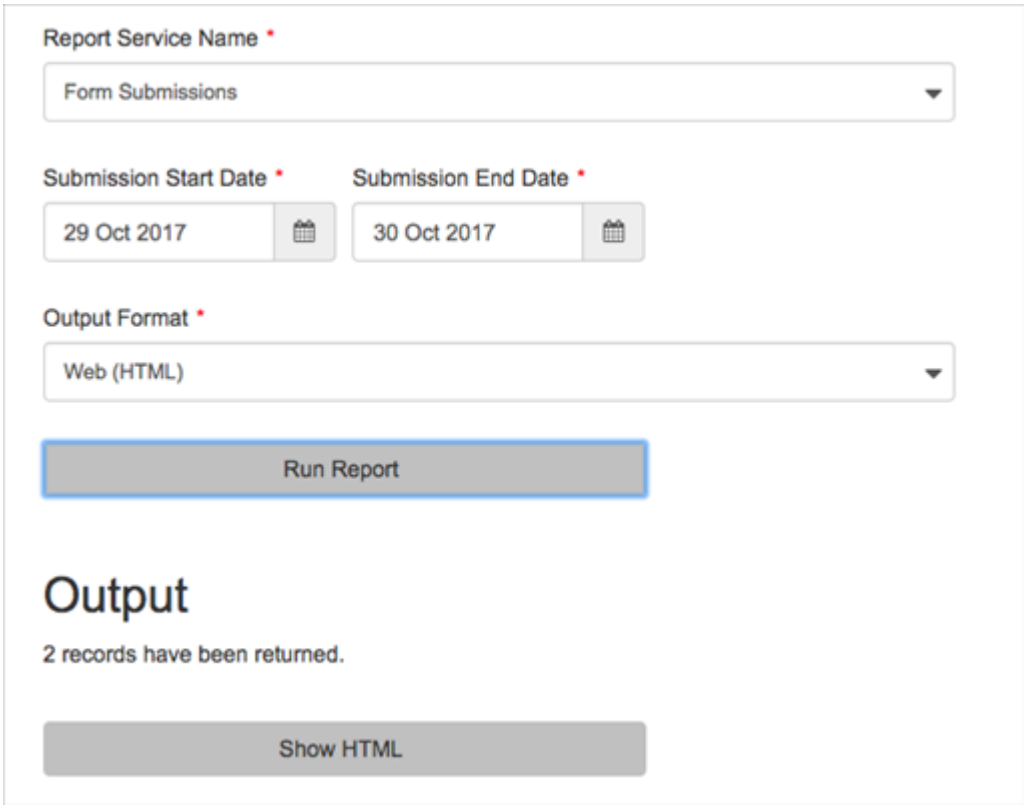
```
def createCsvReport(reportHeaders, dataTable)
{
StringBuilder dataStr = new StringBuilder();
reportHeaders.each {
dataStr.append "\"${it}\","
}
dataStr.append "\n"
dataTable.each{ it1 ->
it1.each {
dataStr.append "\"${it ?: ""}\","
}
dataStr.append "\n"
}
return dataStr.toString()
}
```

3. GetSubmissionDeliveryList - This "DAO Layer" extracts submission objects for a given date range for a specific formcode. I've played with many DAOs and maybe one suits your purpose more than this but what I've found is that many of the DAOs cut off after a certain limit regardless of what you specify. The requirements I've had in many reports goes beyond these limitations so I've found this script/DAO to be the best and surprisingly performs very well. You can also pass a parameter to include Saved transactions.

```
import com.avoka.core.groovy.GroovyLogger as logger
import com.avoka.fc.core.service.ServiceLocator
import com.avoka.fc.core.dao.*
import com.avoka.fc.core.entity.*
import groovy.json.*
String formCode = parameters.formCode
Date startDate = Date.parse('yyyy-MM-dd', parameters.startDate)
startDate.set(hourOfDay : 0)
startDate.set(minute : 0)
startDate.set(second : 0)
Date endDate = Date.parse('yyyy-MM-dd', parameters.endDate)
endDate.set(hourOfDay : 23)
endDate.set(minute : 59)
endDate.set(second : 59)
String includeSaved = parameters.includeSaved ?: 'false'
//logger.info "-------------------------------------------------"
//logger.info serviceDefinition.getServiceName()
//parameters.each { logger.info "$it.key: $it.value" }
//logger.info "-------------------------------------------------"
List deliveryStates = ["Not Ready", "Ready", "Sent Email", "In Progress", "Pending",
"Completed", "Error", "Undeliverable", "Not Required"]
String clientId = DaoFactory.getClientDao().getClientByCode("digital").getId();
String deliveryDetailsId = null
String deliveryMessageLike = null
int fetchLimit = 10000
SubmissionDao submissionDao = DaoFactory.getSubmissionDao();
List allResults = []
deliveryStates.each {
allResults.addAll(submissionDao.getSubmissionDeliveryList(it,
clientId,
deliveryDetailsId,
formCode,
deliveryMessageLike,
startDate,
endDate,
fetchLimit))
}
if (includeSaved == 'true') {
def savedList = submissionDao.getSavedSubmissionList(null, clientId, null, null,
startDate, endDate)
def filteredResults = savedList.findAll{ it.formCode == formCode }
allResults.addAll(filteredResults)
}
```

```
//logger.info "Total records retrieved - getSub-
missionDeliveryList.size:${allResults.size}"
return allResults
```

The interface for the custom report is shown below:



You can also use the **TxnHistoryQuery** class to create a custom report using a Groovy script.

Next, learn how to view job licensing report.

Outgoing Email Service Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager uses a **SMTP**[1] server to send emails to various types of users, such as application form users, system administrators and DevOps, to notify them about the following events:

- A link to a user to download a PDF receipt of their completed transaction
- A PDF receipt as an attachment to a user directly
- A form is saved, shared, submitted, and so on
- Delivery process
- Form promotions
- Scheduled reports
- Alerts and system notifications

Emails are either sent directly as part of application form processes or can be scheduled using the email queue. The later provides a mechanism of sending emails, keep track of sent emails including their statuses, and re-send emails.

Manager doesn't process incoming emails and it doesn't monitor the SMTP server for bounced emails. Bounced emails can take place when form users enter incorrect email addresses while filling in forms. In this case, Manager still sends emails to these addresses, which are bounced on the SMTP level. If you want to monitor bounced emails, you need to ask your cloud hosting team or your SMTP server administrator to configure the bounces to go to a mailbox where you can monitor them and action accordingly.

Manager doesn't directly support a full secure email mechanism through SMTP, so you can not send encrypted emails from a Manager server to a customer email server. However, Manager allows you to enable TLS connections for the Manager's mail server. If Manager is configured to use **Amazon Simple Email Service (SES)**[2] to send emails, it uses TLS up to the point when SES send out the emails to the recipient SMTP.

You can change an email notification format to adjust it to your requirements.

---

[1] SMTP (Simple Mail Transfer Protocol) is a TCP/IP protocol used in sending and receiving e-mail. However, since it is limited in its ability to queue messages at the receiving end, it is usually used with one of two other protocols, POP3 or IMAP, that let the user save messages in a server mailbox and download them periodically from the server. In other words, users typically use a program that uses SMTP for sending e-mail and either POP3 or IMAP for receiving e-mail. On Unix-based systems, sendmail is the most widely-used SMTP server for e-mail. A commercial package, Sendmail, includes a POP3 server. Microsoft Exchange includes an SMTP server and can also be set up to include POP3 support.

[2] Amazon Simple Email Service (SES) is a pay-per-use service that allows you to build in email functionality into an application that you are running on AWS.

Next, learn how to configure the SMTP server.

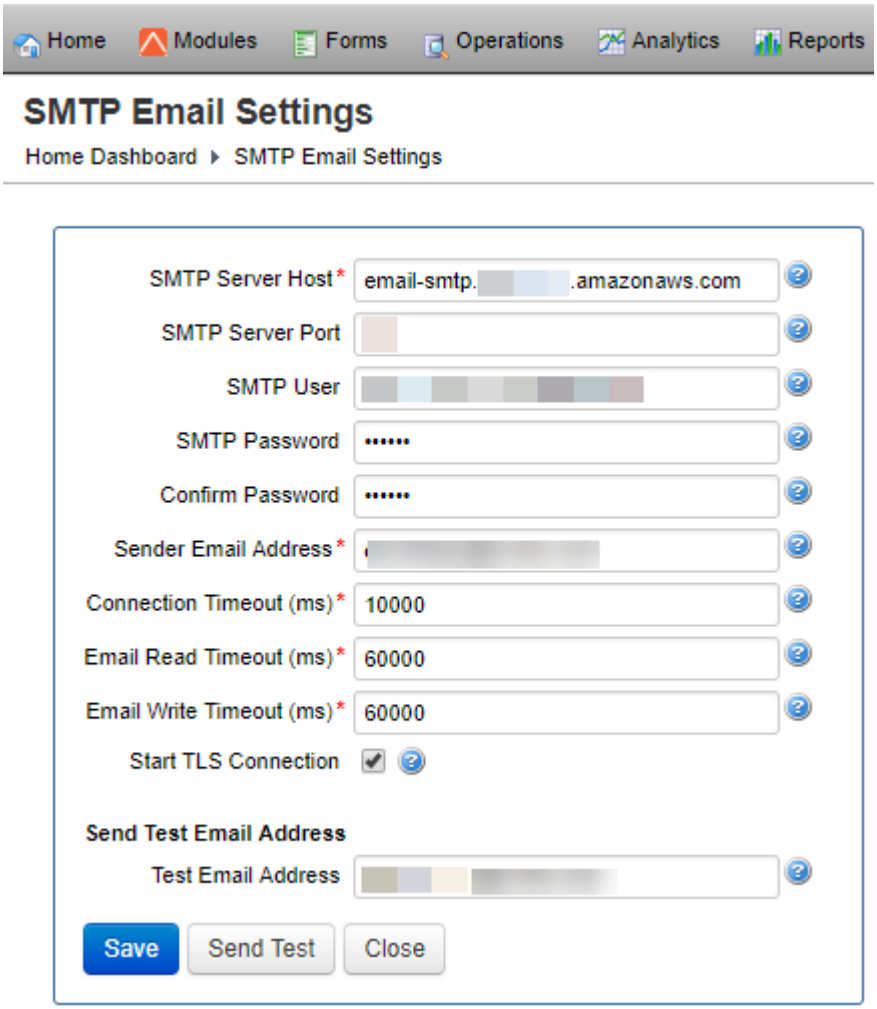
Configure a SMTP Server

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager requires you to configure an SMTP server before you can start sending queued emails notification emails. The SMTP server can be a local one included with the installer or an external one, such as **Amazon Simple Email Service (SES)**[1].

To configure the SMTP server:

1. Select **System > SMTP Email Settings**.



2. Provide an SMTP server host name in the **SMTP Server Host** field.
3. Specify a port number in the **SMTP Server Port** field.
4. Enter an SMTP email server user name in the **SMTP User** field.
5. Choose a strong SMTP email server user password in the **SMTP Password** field.

---

[1]Amazon Simple Email Service (SES) is a pay-per-use service that allows you to build in email functionality into an application that you are running on AWS.

6. Provide the default email sender address in the **Sender Email Address** field.
7. Specify the email server connection timeout in the **Connection Timeout (ms)** field.
8. Specify the email server read timeout in the **Email Read Timeout (ms)** field.
9. Specify the email server write timeout in the **Email Write Timeout (ms)** field.
10. Select the **Start TLS Connection** checkbox to enable **TLS**[1] connection to the SMTP server. This sets the **`mail.smtp.starttls.enable=true`** property. TLS version depends on a 3-rd party library used by Manager as well as recipient's mail server's capability.

    > Check that the SMTP server supports TLS before enabling this setting, otherwise emails won't be delivered and you'll receive an error.

11. Provide the test email address in the **Test Email Address** field to send a test to.
12. Click **Send Test** to verify SMTP configuration by sending an email to the **Test Email Address**.
13. Click **Save** to update the changes.

Next, learn how to view the email queue.

[1] Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network.

Create an Email Queue Item

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows a Manager administrator to create emails in the email queue, which are automatically sent and tracked.

To send an email, you need to add it to the email queue first:

1. Select **System > Email Queue**.
2. Click **New**.
3. Select a space from the dropdown list. It populates the **Available Users** list with all users associated with this space.
4. Select one or more email addresses from the **Available Users** and click **>** to add them as recipients. To add all users, click **>>**. Click **<** or click **<<** to remove selected users. Add additional email addresses into the **CC** and **BCC** fields.
5. Edit the **From** field with a custom email, if required, otherwise the default value of the **Sender Email Address**, defined in the **SMTP Email Settings**, is used.
6. Edit the **Reply To** address if it is different to the **From** address. This address is used when a user replies to an email sent out via the email queue.
7. Enter the email message into the **Body for HTML** email content or the **Plain Body** for plain text content.
8. Click **Send** to add this email to the email queue, which sends it. There is a maximum of two automatic retries in case of any errors.

Next, learn how to view the email queue.

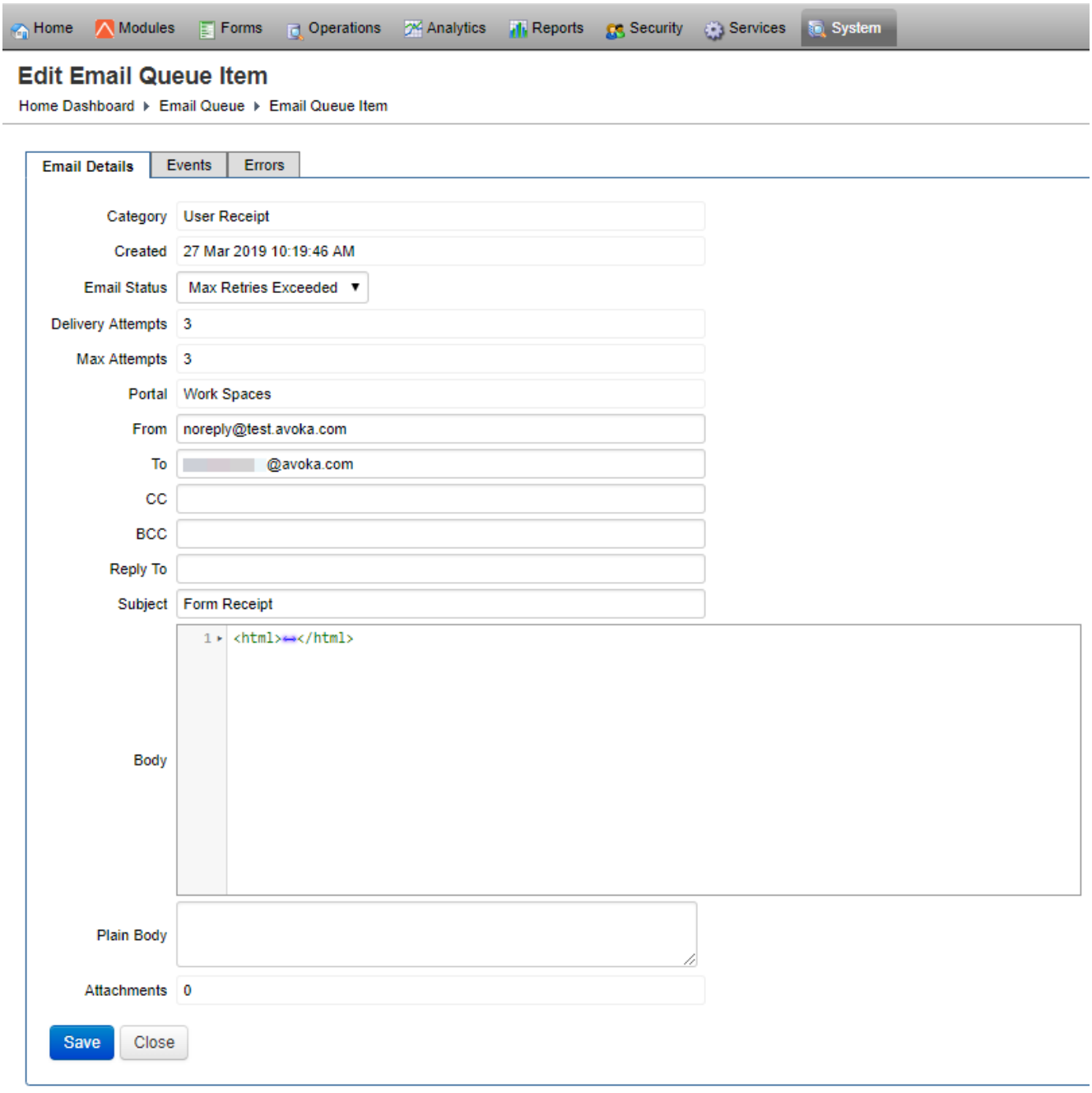
Edit an Email Queue Item

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to edit an email that was added to the email queue.

To edit an email:

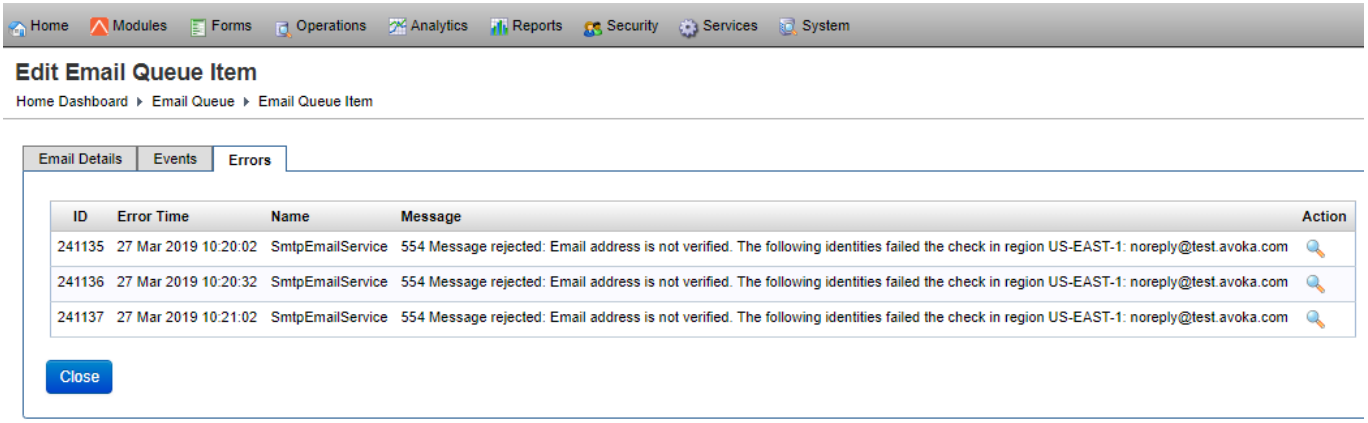
1. Select **System > Email Queue**.
2. Locate an email and click **Edit**.
3. Click the **Email Details** tab to edit some of the fields such as the recipients and email body.



4. Select **Ready** from the **Email Status** dropdown list to send this email again, which can resolve email delivery issues when an email has been unsuccessfully resent several times and has exceeded the maximum number of retries.
5. Click **Save** to update the changes.

Click the **Events** tab to see any events associated with this email delivery.

Click the **Errors** tabs to see any errors that occurred when attempting to send the email. This can be useful for debugging email failures.



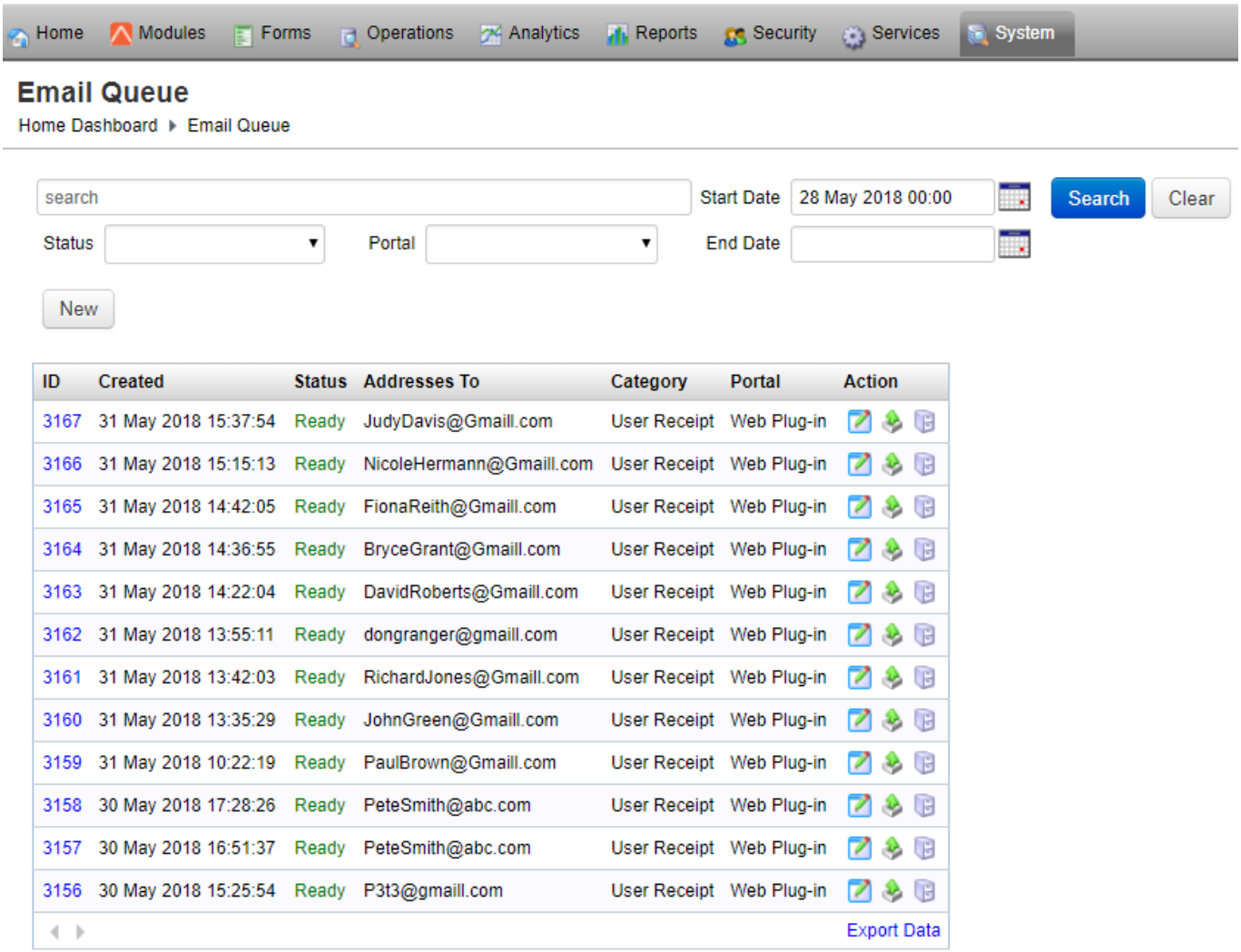
Next, learn how to view email queue items.

View the Email Queue

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides the email queue to send outgoing emails.

To view the list of all emails that has been sent, select **System > Email Queue**.



To filter or search the email queue by one or more criteria, specify the following settings and click **Search**:

- A part of the email, for example an email address.
- Status – select from the following list: Not Ready, Ready, Completed, Error, Max Retries Exceeded or Purged.
- Portal – select a form space.
- Start Date
- End Date

Click **Clear** to reset the search criteria.

You can also perform the following actions:

- Create a new email.
- Edit an email.
- Click **Send Email Now** to resent the email manually. An administrator can do this task after identifying and fixing the email delivery issues.
- Click **View Submission Details** to check the transaction detail.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to create email queue item.

Data Retention Management Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager has a high volume of user form transactions and associated resources, such as data records and log files, that amounts to significant data even during a rather short period of operation. This can have a detrimental effect on the overall system performance, which, in turn, can negatively impact user experience and various 3-rd party integrations. To maintain the system optimal performance over time, Manager provides data storage, retention and purging capabilities that you can adjust to your company's data requirements. This is instrumental in achieving the following:

- Control the growth of database tables. In Manager, the Transaction table is a heavy weight table used to store a lot of information about in progress transactions and is not designed for long term storage. When a transaction is completed an entry is created in the Transaction History table which is designed to provide a long term audit trail. Purging data and records quickly from the Transaction table will help maintain performance.
- Maintain high system performance.
- Purge transaction form data which contains form users' **Personally Identifiable Information**[1]. Form transactions can include sensitive information either business or personal depending on how the transaction have been designed. Purging this data after delivery provides a stronger security story.

Data retention management is an automated process - what you need to do is to adjust the default data retention configuration and periodically review it to make sure the data is purged as per your **SLAs**[2].

Manager purges the following data:

- Transaction Data - the forms XML, Data Extracts, Attachments and Receipts. This will include the majority of the users personal information.
- Transaction Records - the actual record from the Transaction table. The main reason to keep this record in place for short period of time is for troubleshooting.
- Transaction History Records - the audit log of transaction activities. For systems with high throughput there is a argument for archiving these records for long term storage and purging these records from Manager.
- System Logs - numerous areas of system logs whose purge time can the controlled separately for troubleshooting purposes.

Manager collects data retention information for each transaction including whether any PII data has been purged or is due to be purged and details about the data retention stage reached in the life cycle of this transaction.

---

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

[2]A service-level agreement (SLA) is a commitment between a service provider and a client. Particular aspects of the service – quality, availability, responsibilities – are agreed between the service provider and the service user.

You can set the data retention and storage settings globally and override important settings for individual organizations.

Data Retention Configuration - Global Level

In addition to data retention, you can also configure submission data storage. This means that you can choose how and where submission-related data, such as submission data, attachments and submission history data for saved forms, is stored and retrieved by Manager.

By default, submission-related data is stored in a Manager database. However, it can be configured and changed at runtime as well. The functionality is encapsulated in a set of service definitions of type Submission Data Storage. The storage service that is marked as the default for this service type is used when storing submission data.

Data Retention Configuration - Organization Level

You can override important form-related settings for each organization individually.

The data retention policies you can configure are the maximum age of form submissions in general as well as delivered (including undeliverable) and saved (including abandoned) submission data. If you would like the global settings to apply for this organization, select "Use system default".

You can also set the data storage encoding for the organization if it is different to the system-wide setting. We recommend that you use the **Compressed / Encrypted** option for maximum security and minimal storage space requirements.

Finally, for additional security you can set a rollover interval for the security key for the organization. The security key is used by Manager to encrypt submission-related data. All encrypted data is associated with its security key, so data encrypted before a security key change will still be accessible. If you choose a value other than "Never", a new security key will be automatically generated by a background job when required.

Manager has two modes of data retention policy, strict and relaxed, that determines the length of time transactional data is retained. The selected policy can be customized at the environment organization, or form levels.

> It's recommended to use the strict policy mode, which enforces less transactional, historical and log data retention as opposed to the relaxed policy mode.

Manager uses various scheduled jobs that automatically purge transactions that have reached their maximum age.

You can check the data retention summary, which is useful for monitoring the overall performance of the data retention management and assists in diagnosing and tuning of any data retention management issues.

Next, learn about data retention policies.

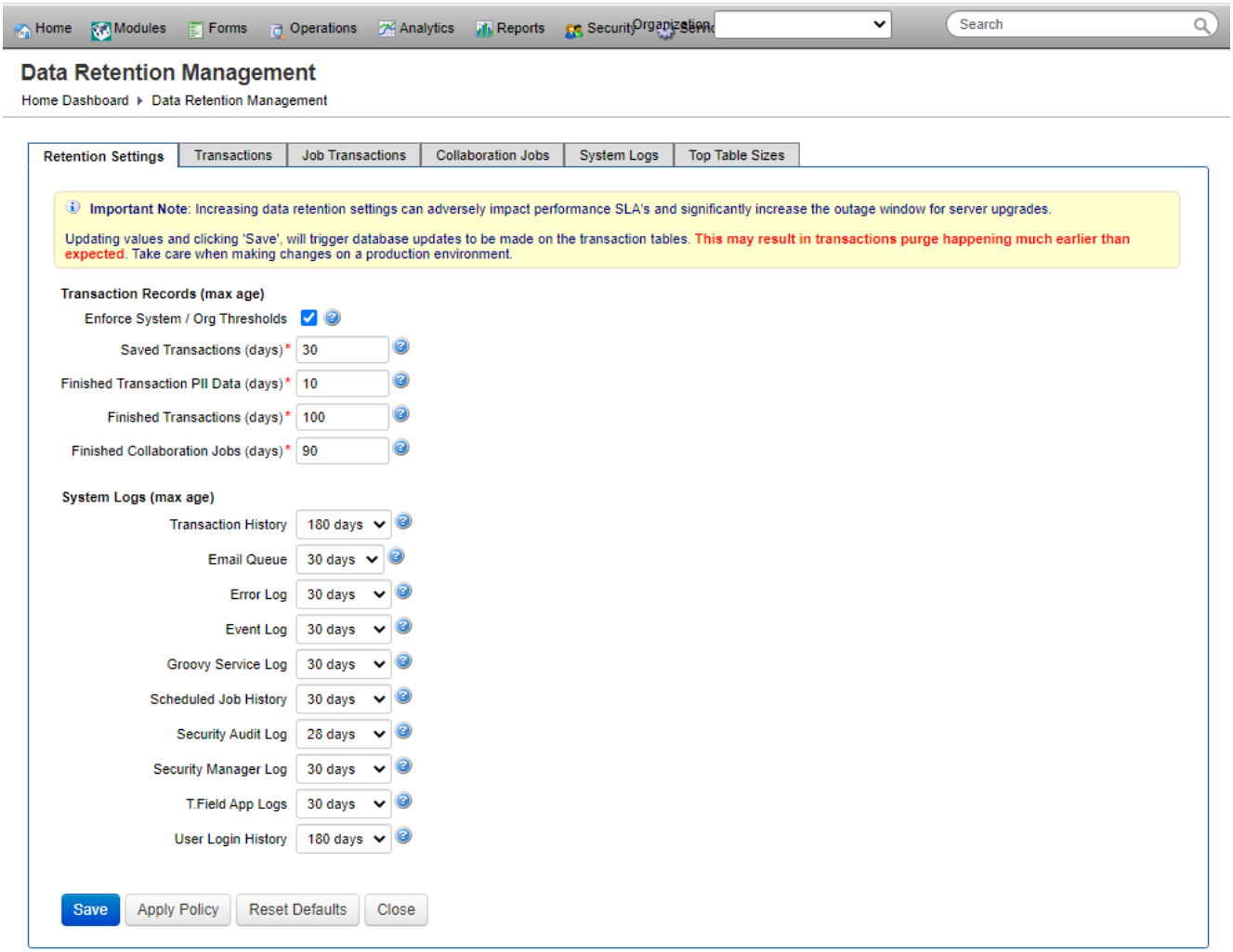
Configure Data Retention

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager comes with highly customizable Data Retention Management that allows you to implement various customer requirements pertaining to their data retention policies.

To configure data retention:

1. Select **System > Data Retention Management**.



You must have the Service Edit permission to view the Data Retention Management tab.

2. Select the **Retention Settings** tab to configure transaction records and system logs retention settings.
3. Configure Transaction Records. Transaction records are created the moment a user opens a form. The status of the form changes as the user saves, cancels, and submits the form. Each transaction corresponds to the user entered data, and is associated with other details, such as submission history, submission properties, data extracts, and attachments. In relation to data retention, transactions incur two major concerns:

- Transactions are continually generated and contain a large amount of data. This can lead to large database sizes and degraded performance if data is not purged regularly.
- Transactions contain user-entered Personally Identifiable Information. This can pose a security risk. Manager comes with several data security measures. It is recommended to ensure that PII is purged as soon as transactions have finished.

  Generally, you want to keep your purge settings strict. Using the strict mode will ensure that transaction data is purged regularly, maximizing your system's performance and security. Manager supports this approach by a frequent job that checks for and purges any PII data and completed submissions that had been completed more than the maximum age time frame (according to your data retention management settings). The transactions eligible for deletion are completed transactions (successful submissions) and abandoned transactions. That is, the Manager must recognize that the transaction is in a finished state.

4. The **Enforce System/Org Thresholds** checkbox determines whether to constrain the settings that can be overridden on the organization or form level, so select it to prevent setting a more lenient override for a corresponding setting at the organization or form level. Furthermore, where there is a stricter corresponding setting at the organization level, you will be prevented from setting a more lenient corresponding override at the form level. For example, if the Enforce System/Org Thresholds checkbox is ticked and the global Finished Transaction PII Data policy is set to 14 days, you cannot enter values greater than 14 days for the Finished Transaction PII Data policy on the organization or form level. In addition, if an organization's value of the Finished Transaction PII Data policy is 10 days, you cannot configure any of this organization's forms to have a Finished Transaction PII Data policy value greater than 10 days.

   Thresholds are only relevant to the Saved Transactions and Finished Transaction PII Data settings.

   Even if **Enforce System/Org Thresholds** is selected, changes to global or organization policies DO NOT affect the values set on lower levels. For example, if an organization has a Finished Transaction PII Data policy value of 14 days and you change the global Finished Transaction PII Data policy value to 7 days, the organization value will remain the same until the organization settings are edited. This avoids problems around accidental changes to global policies trickling through the entire system.

   The following settings will initially be defaulted according to your environment's data retention policy mode, strict or relaxed. The individual settings may be customized as explained.

5. The **Saved Transactions** setting allows you to specify the number of days after which a transaction that was explicitly saved by a user is automatically abandoned by the system. The number of days before abandonment will reset each time the form is saved by the user. This means that the allowable length of time between when the form is first opened and its eventual submission may be extended by the user resuming the form regularly to complete and save the entered data.

Once a saved transaction is abandoned, it is considered a finished transaction like when a user completes and successfully submits a form. Note that the transaction will not be deleted at this point, but as a finished transaction, the countdown for PII data deletion and submission record deletion starts ticking. This setting can be customized on both the organization and form levels.

Be aware that 24-hour automatic abandonment of transactions can happen even they are rendered but never actually captured any data, or transactions that are never user-saved, even though they are background-saved, for example, on page navigation.

6. The **Finished Transaction PII Data** setting determines how many days personally identifiable submission data (XML data, attachment data, submission extract data and submission properties) will be kept once a transaction has finished (completed or abandoned). This data takes up a large amount of space and poses some security risk due to containing user entered data. The transaction record itself will stay around after data deletion (see the next setting below). This setting can be customized on both the organization and form levels.
7. The **Finished Transactions** setting determines how many days transaction records and any associated details will be kept once a transaction has finished. If PII data has been deleted for a submission, the header records remain until this policy kicks in, allowing administrators to find the submission in the logs.
8. Select the number of day from the **Finished Collaboration Jobs** dropdown list to specify how long collaboration jobs and their submissions are kept. The default is 90 days.
9. The **Transaction History** setting determines how long transaction history records are kept, counting from the time the submission is started. The default is 180 days.
10. **Email Queue** setting determines how long email queue entries is kept, counting from the time that the email was created. | 18.05 .
11. **Error Log** setting determines how long error log entries and data that are not associated with a submission will be kept, counting from the time that the error occurred.
12. **Event Log** setting determines how long event log entries that are not associated with a submission will be kept, counting from the time that the event was logged.
13. **Groovy Service Log** setting determines how long log entries for Groovy service invocations will be kept, counting from invocation time.
14. **Security Log Audit** setting determines how long audit log entries are kept, counting from the time the log was created. The audit log can be crucial to tracking down changes made by administrators, so on non-test servers, we recommend that you allow adequate time before purging this log.
15. **Schedule Job History** setting determines how long Manager keeps scheduled job history entries, which are created every time a scheduled job runs.
16. **User Login History** setting determines how long events relating to user session, such as login, logout, session expiry, and account lock etc., are kept.
17. Click **Save** to update the changes.
18. Click **Apply Policy** run a purge now.

Open the event log and search for events using **data retention** key word to see the actual number of purged transactions of each category.

Next, learn how to configure organization data retention.

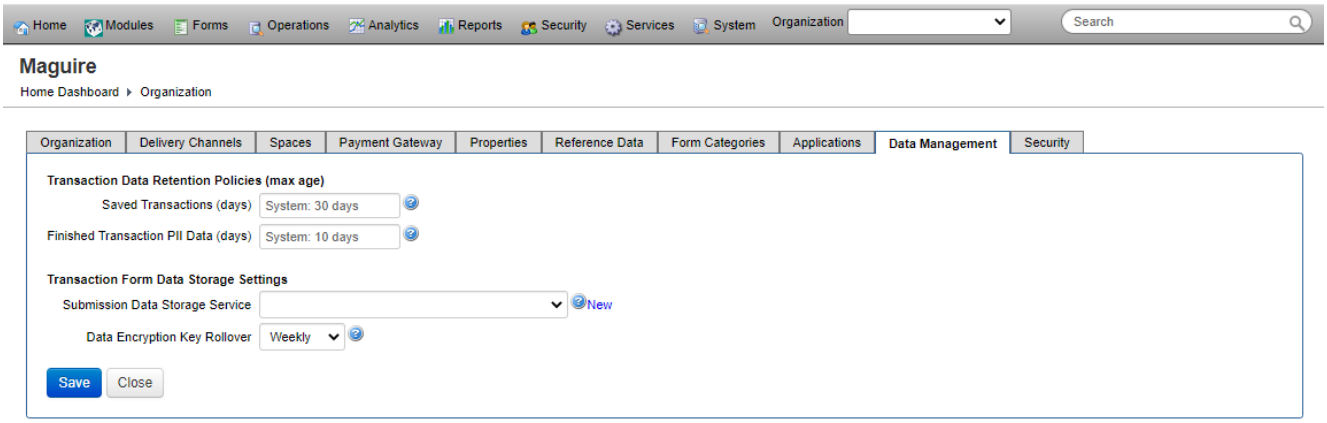
Configure Organization Data Retention

Journey Manager (JM) | System Manager / DevOps | 19.05

By default, Manager applies global data retention configuration defined for the entire server to its organizations, unless an organization has its own data retention configured. However, you can customize retention policies for all transactions associated with forms, for example, saved and finished transaction's **PII**[1] data.

To configure organization data retention:

1. Select **Forms > Organizations**.
2. Locate an organization and click **Edit**.
3. Select the **Data Management** tab to check and edit data retention policies.



You can only update these settings if the **Enforce System/Org Thresholds** checkbox in the Retention Settings tab is selected and no values have been entered for each of these settings.

4. Edit **Saved Transactions** with a value which doesn't exceed the **Saved Transactions** organization threshold, if one exists at the organization level (from the Data Management tab for the related organization), otherwise at the global level (from the Retention Settings tab). In this example, the value must not exceed the system active threshold of 30 days. Note this setting can also be configured on the Abandonment tab.
5. Edit **Finished Transaction PII Data** with a value which doesn't exceed the **Finished Transaction PII Data** organization threshold, if one exists at the organization level (from the Data Management tab for the related organization), otherwise at the global level (from the Retention Settings tab). In this example, the value must not exceed the organization active threshold of 7 days.

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

However, when the Enforce System/Org Thresholds checkbox in the Retention Settings tab is not selected, no such input prompts are provided, and different constraints apply. From the Details tab, if you wish to override the global and/or organization Data Retention Management settings, enter custom values for:

- Edit **Saved Transactions** with a value which doesn't exceed the maximum value for the Saved Transaction setting as determined by the data retention management policy. That is, the value cannot exceed 180 days for a strict mode and cannot exceed 730 days (2 years) for a relaxed mode.
- Edit **Finished Transaction PII Data** with a value which doesn't exceed the maximum value for the Finished Transaction PII Data setting as determined by the data retention management policy. That is, the value cannot exceed 30 days for a strict mode and cannot exceed 730 days (2 years) for a relaxed mode.

6. Select a submission data storage service from the **Submission Data Storage Service** dropdown list.
7. Select a submission data encryption key rollover policy from the **Data Encryption Key Rollover** dropdown list specifying the following available policies:

- Never
- Annually
- Monthly
- Weekly. This is the default option
- Daily

8. Click **Save** to update the changes.

Next, learn how to configure form data retention.

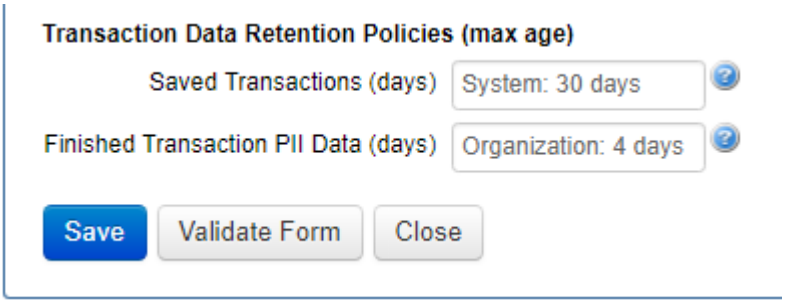
Configure Form Data Retention

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager applies global data retention policies, defined for the entire server or individual organizations, to its forms. However, you can customize transaction retention policies on the form level, for example, saved and finished transaction's **PII**[1] data. These settings then apply to all submissions made for this form.

To configure form data retention:

1. Select **Forms > Forms**.
2. Locate a form and click **Edit**.
3. Select the **Details** tab to check and edit data retention policies in the **Transaction Data Retention Policies** section.



You can only update this configuration if the **Enforce System/Org Thresholds** checkbox in the Retention Settings tab is selected and no values have been entered for each of these settings.

4. Edit **Saved Transactions** with a value which must not exceed the configured **Saved Transactions** organization's threshold, or the global threshold. In this example, the value must not exceed the system active threshold of 30 days.

You can also configured the saved transaction threshold from the Abandonment tab of a form.

5. Edit **Finished Transaction PII Data** with a value which must not exceed the **Finished Transaction PII Data** organization's threshold, or the global threshold. In this example, the value must not exceed the organization active threshold of 4 days.

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

If the **Enforce System/Org Thresholds** checkbox in the Retention Settings tab is not selected, no default values are provided, and different constraints apply. To override the global or organization data retention configuration:

- Edit **Saved Transactions** with a value which must not exceed the maximum value for the Saved Transaction setting of the data retention management policy, which is 180 days for a strict mode and 730 days (2 years) for a relaxed mode.
- Edit **Finished Transaction PII Data** with a value which must not exceed the maximum value for the Finished Transaction PII Data setting of the data retention management policy, which is 30 days for a strict mode and 730 days (2 years) for a relaxed mode.

6. **Delete Form Data Extracts** for relaxed mode only. | Removed in 5.1.
7. Click **Save** to update the changes.

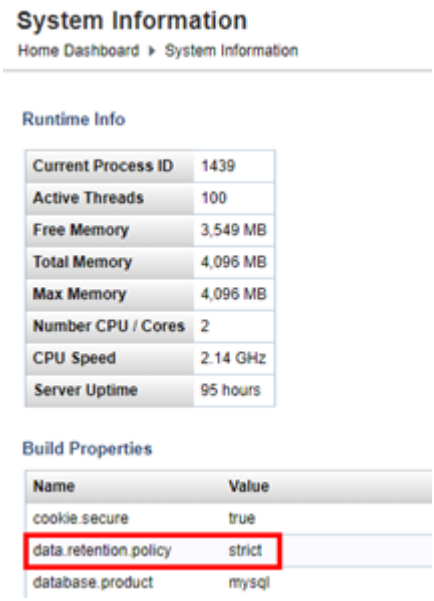
Next, learn how to configure organization data retention.

Data Retention Policies

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides you with two modes of data retention policy, **strict** and **relaxed**. The selected policy determines the default data retention management configuration at the environment level, which are used throughout the Manager server, unless data retention is configured at the organization or form level. The policy mode is selected during a Manager installation or upgrade.

To check the data retention policy mode of your environment, select **System > System Info**, go to the **Build Properties** section and look for **`data.retention.policy`**.



It's recommended to use the strict mode, which constrains the potential amount of data retained within Manager by restricting the maximum age of transaction data. It prevents problems related to long data retention such as:

- Database growth
- Performance degradation
- Prolonged storage of sensitive **PII**[1]

Both the strict and relaxed modes apply the global default settings to data retention configuration. To view and configure the default settings, follow one of the links below:

- Global data retention configuration
- The Organization's Data Management tab for a specific organization
- The Form's Details tab for a specific form

---

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

The default settings can be individually configured with a value that can't exceed the maximum value defined for each property.

> You can't change Submission Data Extracts and Submission Properties default values.

The differences between the strict and relaxed modes are outlined below for the following data types:

Saved Transactions

This setting determines how long to keep user saved transactions before they are abandoned by Manager and become finished transactions. The Manager environment setting is initially set to the default, but you can change it to any value that doesn't exceed the maximum. Organization and form level settings can be used to override the environment settings. However, they cannot exceed the maximum value.

- Strict Mode: default is 30 days, maximum is 180 days.
- Relaxed Mode: default is 30 days, maximum is 2 years.

Finished Transactions with PII Data

This setting determines how long to keep data entered by the user before it is deleted. The Manager environment setting is initially set to the default, but you can change it to any value that doesn't exceed the maximum. Organization and form level settings can be used to override the environment settings. However, they cannot exceed the maximum value.

- Strict Mode: default is 7 days, maximum is 30 days.
- Relaxed Mode: default is 7 days, maximum is 2 years.

Finished Transactions

This setting determines how long to keep finished transactions before they are deleted. Finished transactions include transactions that have been marked as abandoned or completed. The Manager environment setting is initially set to the default, but you can change it to any value that doesn't exceed the maximum. Organization and form level settings can be used to override the environment settings. However, they cannot exceed the maximum value.

- Strict Mode: default is 90 days, maximum is 2 years.
- Relaxed Mode: default is 90 days, maximum is 5 years.

Finished Collaboration Jobs

This setting determines how long to keep finished collaboration jobs before they are deleted. The Manager environment setting is initially set to the default, but you can change it to any value that doesn't exceed the maximum . Organization and form level settings can be used to override the environment settings. However, they cannot exceed the maximum value.

- Strict Mode: default is 90 days, maximum is 2 years.
- Relaxed Mode: default is 90 days, maximum is 2 years.

Submission Data Extracts

This setting determines how long to keep submission data extracts before they are deleted. You can't override this setting, unless it's Manager version prior to 17.10 and it's in the relaxed mode.

- Strict Mode: default is 7 days, maximum is 30 days.
- Relaxed Mode: default is 7 days, maximum is 2 years.

Submission Properties

The setting determines how long to keep submission properties before they are deleted. You cannot override this setting, unless it's Manager version prior to 17.10 and it's in the relaxed mode.

- Strict Mode: default is 7 days, maximum is 30 days.
- Relaxed Mode: default is 7 days, maximum is 2 years.

When you configure data retention policies in Manager, you need to consider the following:

- Optimal conversion rate – a user must be given an adequate time to return to and complete a transaction.
- System efficiency – a lean database makes a Journey Manager a more responsive system.
- Privacy – **PII**[1] data must be retained for the shortest possible time to minimize any potential security risks.

> The Journey platform is intended to be a system of engagement rather than a system of records. That's why both the relaxed and strict data retention policies have been shortened over time.

Manager keeps data only for as long as it is required within the transaction life cycle. It is recommended to use the strictest possible data retention management policy to avoid excessive data base growth and potential problems during any recurring Manager upgrades.

---

[1]Personally Identifiable Information (PII) is information about an individual that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked to an individual.

However, often it is a balancing act between maintaining a lean, well performing and responsive system, while providing an adequate time frame for users to return to and complete their submitted forms. After sufficient time, given to users to complete their transactions, has elapsed, all incomplete transactions are automatically abandoned.

PII data is automatically purged from the transactions soon after the user has submitted their form to minimize any potential security risk. It's recommended to use the strict data retention policy for any production Manager instances.

Next, learn about data retention schedule.

Data Retention Schedule

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager comes with a mechanism for processing transactions and managing data retention. It executes the Transaction Processing scheduled job in the background every 5 minutes, which purges up to 500 records each time. This keeps data from accumulating and spreads out the processing load over time.

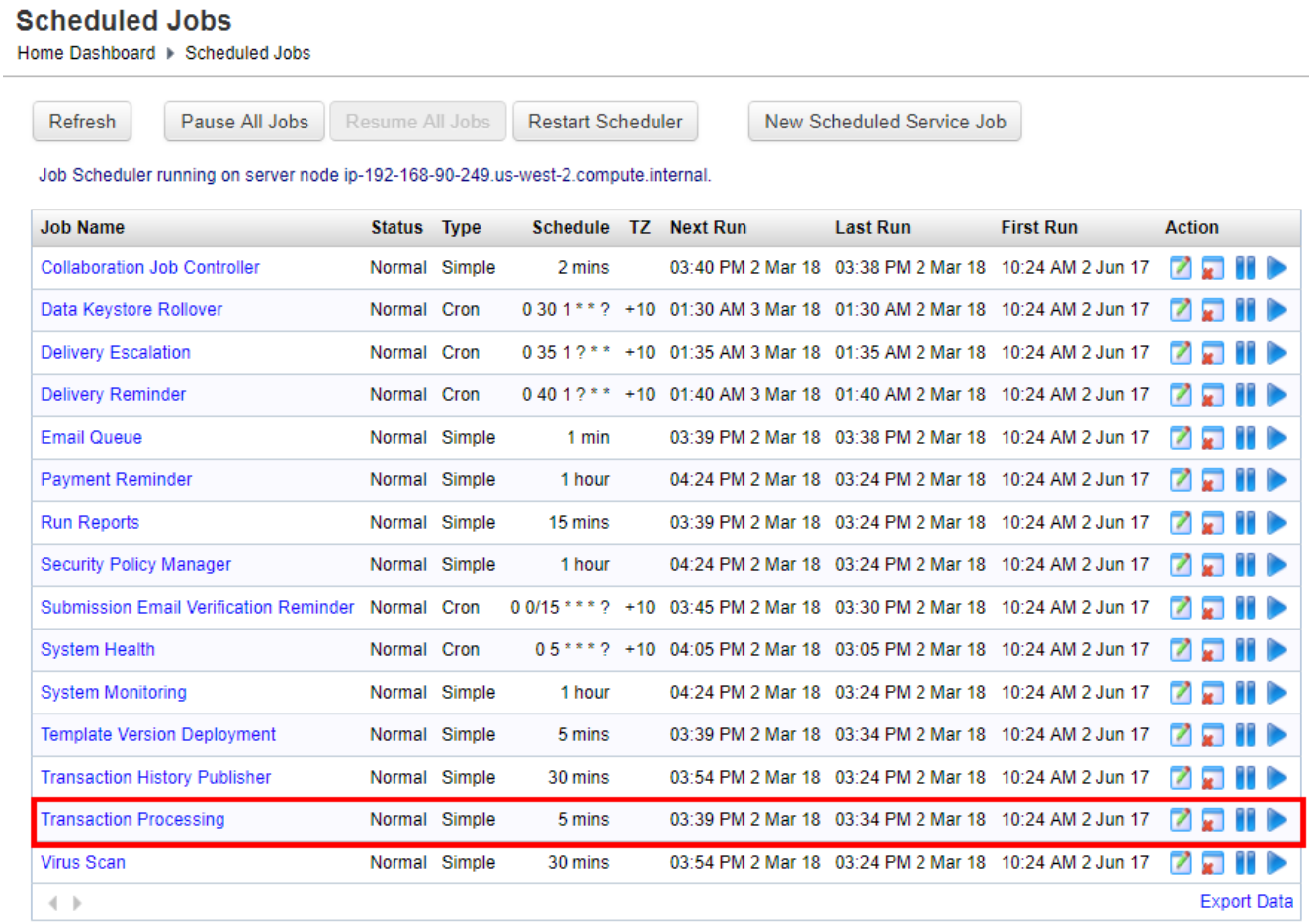
Transaction Processing scheduled job performs the following important role in the transaction life cycle:

- Delivery
- Task expiry
- Abandonment
- Data purging by calling the Transaction Processor and the Data Retention Management services that purge data.

To change the maximum number of purged records, see the Transaction Processor Service and the Data Retention Management Service documentation.

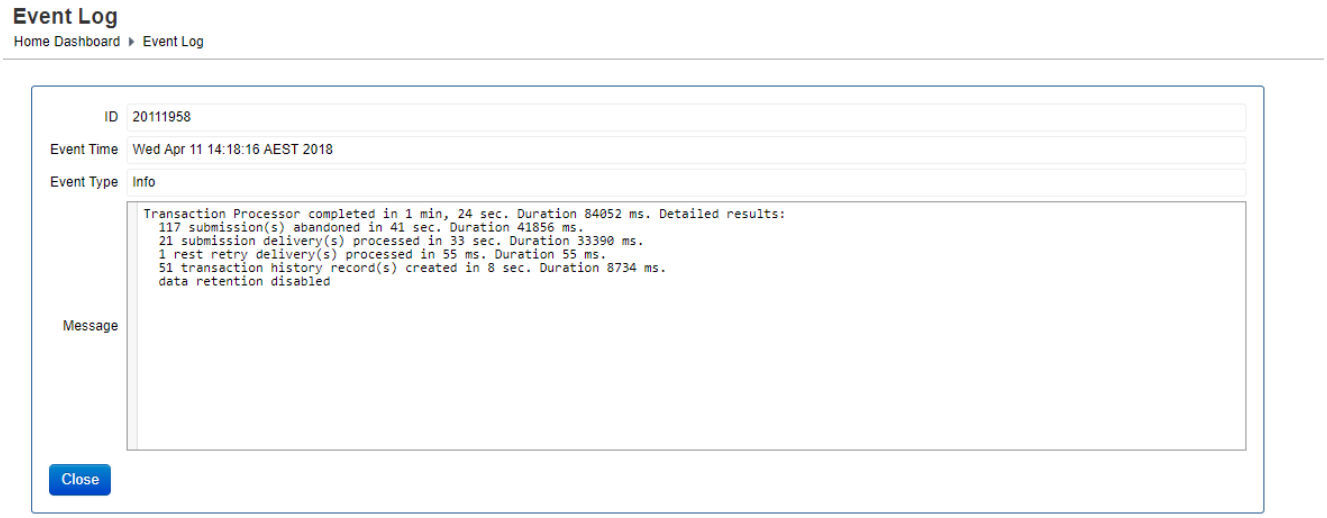
To view the Transaction Processing scheduled job as part of all scheduled jobs:

1. View scheduled jobs
2. Search for Transaction Processing to see the following scheduled jobs details.

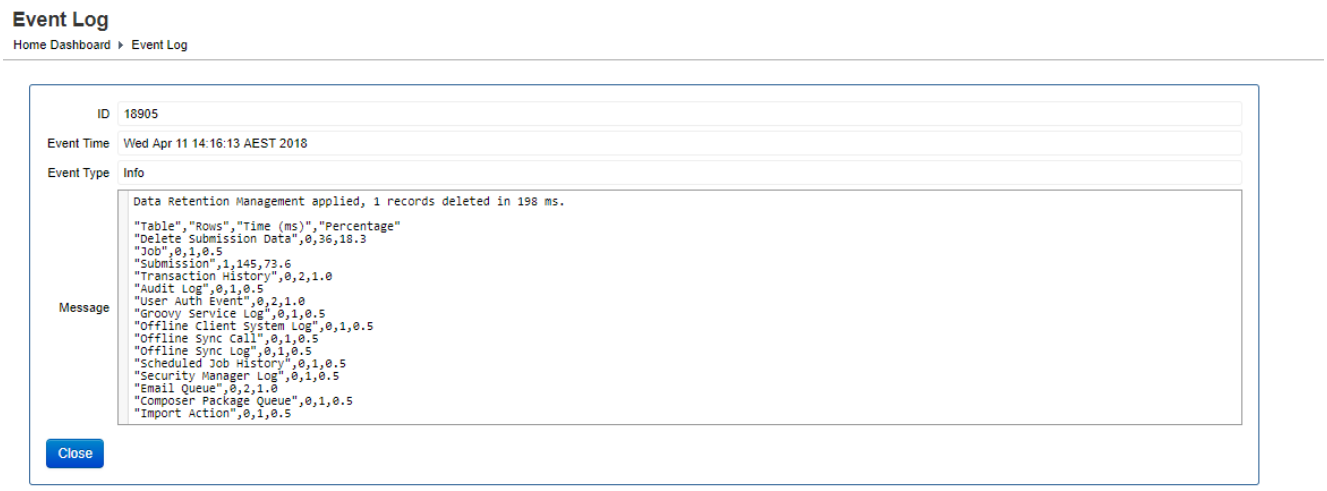


Data Retention Management audit information is captured in the Event Log as shown below:

- Transaction Processor service audit details.



- Data Retention Management service audit details.



Next, learn how to view transactions data retentions summary.

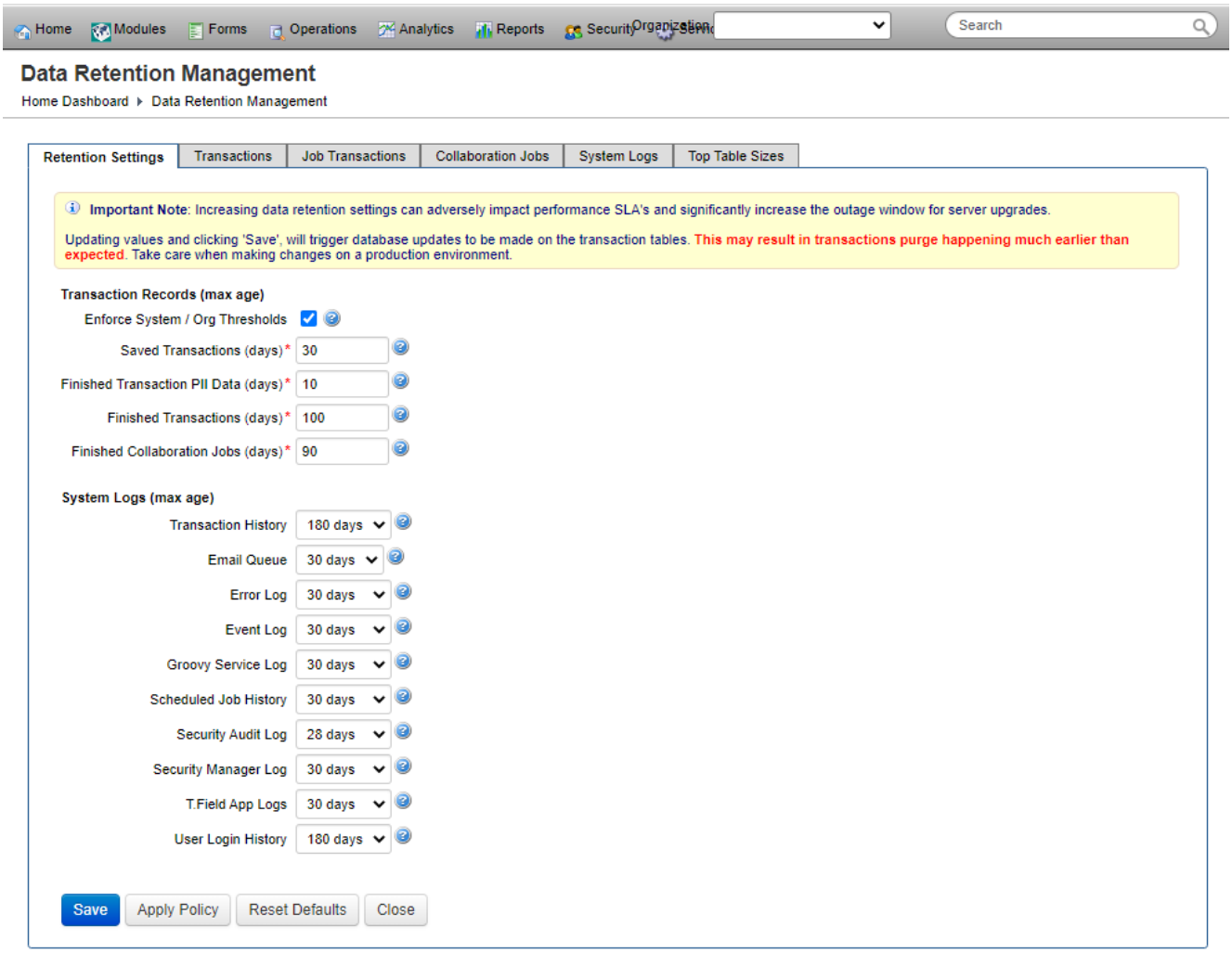
Manual Data Purge

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager continuously applies data retention policies to all relevant data and purges it accordingly. However, you can purge data manually, if you must delete old data to free up server resources such as OS file system or a database space.

To trigger a data purge manually:

1. Select **System > Data Retention Management**.
2. Click the **Retention Settings** tab to verify the current settings and adjust them so the manual purge removes data that needs to be removed.



3. Click **Apply Policy**. Manager executes the Transaction Processor service to purge data and displays a report with actual number of records been purged.

Next, learn about data retention policies.

Data Retention Summary

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager keeps track of how much data is currently stored for each server instance and breaks it down by different categories. This allows an administrator to choose the most appropriate data retention policy and verify whether data has been purged as expected or not.

Manager provides the following data retention summaries:

- Transactions
- Job transactions
- Collaboration jobs
- System logs
- Top table sizes

Next, learn about Data Retention Schedule.

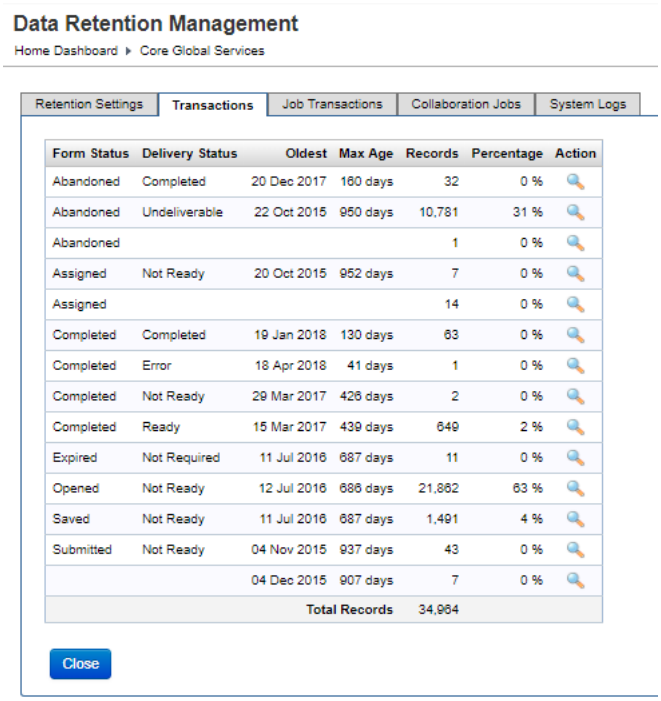
Data Retention Summary: Transactions

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager allows you to view data retention summary of transactions.

> Transactions don't include submissions associated with collaboration jobs. | 18.05

To view transactions data retention summary, select **System > Data Retention Management** and click the **Transactions** tab. The data usage summary is grouped by **Form Status** and **Delivery Status**.



Form Status	Delivery Status	Oldest	Max Age	Records	Percentage	Action
Abandoned	Completed	20 Dec 2017	160 days	32	0 %	
Abandoned	Undeliverable	22 Oct 2015	950 days	10,781	31 %	
Abandoned				1	0 %	
Assigned	Not Ready	20 Oct 2015	952 days	7	0 %	
Assigned				14	0 %	
Completed	Completed	19 Jan 2018	130 days	63	0 %	
Completed	Error	18 Apr 2018	41 days	1	0 %	
Completed	Not Ready	29 Mar 2017	426 days	2	0 %	
Completed	Ready	15 Mar 2017	439 days	649	2 %	
Expired	Not Required	11 Jul 2016	687 days	11	0 %	
Opened	Not Ready	12 Jul 2016	686 days	21,862	63 %	
Saved	Not Ready	11 Jul 2016	687 days	1,491	4 %	
Submitted	Not Ready	04 Nov 2015	937 days	43	0 %	
		04 Dec 2015	907 days	7	0 %	
			Total Records	34,964		

- **Oldest** - the date of the oldest transaction for this group.
- **Max Age** - the maximum age of the oldest transaction for this group.
- **Records** - the number of transaction records for this group.
- **Percentage** - the percentage transaction records for this group of the total number of transaction records.

A form status can be one of the following:

- Abandoned - transactions that have been abandoned by the user, or based on the data retention policy.
- Assigned - a task which is waiting to be completed by a user.
- Completed - transactions that have been successfully submitted and contain all required data.
- Opened - transactions that have been started by a user. When a transaction is created by a user (by starting a new form), the initial form status is set to opened. This status will be changed once the user saves, cancels, or submits the transaction.
- Saved - transactions that have been saved by the user or automatically by the system.
- Submitted - transactions that have been submitted but may require additional information (attachments, payment, etc.). Once the required data is collected, the transaction status will change to Completed. This status is not common for Maestro forms as all information, such as attachments and payments, are collected prior to the user submitting the form. Most Maestro forms would skip the Submitted status and be marked as Completed after the form is submitted by the user. This is a legacy feature.
- Expired - a form used by this transaction has expired and is no longer available.

A delivery status can be one of the following:

- Completed - a transaction has been successfully delivered, based on the configured delivery channel.
- Undeliverable - a delivery is not possible. Check the error logs for further details of the error.
- Not Ready - a transaction is not ready for delivery yet. This is the start state before a transaction has been completed by the user. Generally, the receipt render service will kick the transaction from this state to the Ready state when the receipt is rendered.
- Not Required - a delivery is not required. It may be that this form is part of a collaboration job and the delivery is to be handled separately say as a consolidation of the transaction details from all the associated forms.
- Ready - a transaction is ready to be delivered. A transaction will be marked as Ready until the delivery channel is processed, or it may remain marked as Ready if a delivery channel is not configured.
- In Progress - a delivery has been started but is waiting to be completed.
- Paused - a transaction delivery is paused. This transaction's status can be changed by some other means, for example by a Groovy Scheduled Task.
- Error - a transaction delivery has failed and is waiting to be retried.
- Sent Email - a secure email has been sent and now a transaction is waiting for the user to log in and process it.
- Pending

Only transactions with the following combinations of statuses are purged:

- Form Status: Completed or Abandoned,
- Delivery Status: Completed, Undeliverable or Not Required.

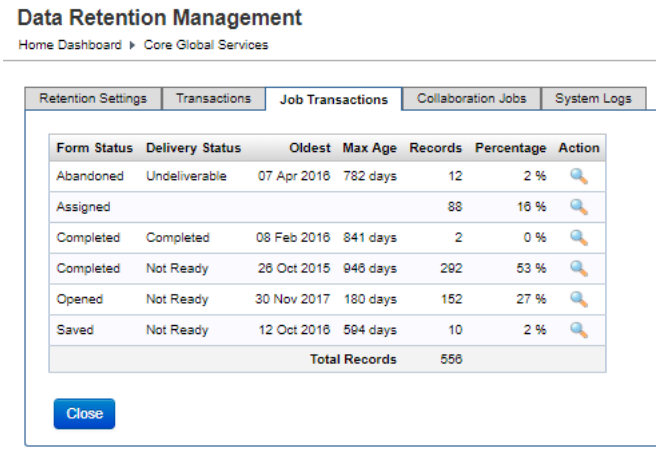
Next, learn how to view job transactions data retention summary.

Data Retention Summary: Job Transactions

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager allows you to view data retention summary of job transactions.

To view data retention summary, select **System > Data Retention Management** and click the **Job Transactions** tab. The data usage summary is grouped by **Form Status** and **Delivery Status**.



- **Oldest** - the date of the oldest transaction for this group.
- **Max Age** - the maximum age of the oldest transaction for this group.
- **Records** - the number of transaction records for this group.
- **Percentage** - the percentage transaction records for this group of the total number of transaction records.

A form status can be one of the following:

- Abandoned - transactions that have been abandoned by the user, or based on the data retention policy.
- Assigned - a task which is waiting to be completed by a user.
- Completed - transactions that have been successfully submitted and contain all required data.
- Opened - transactions that have been started by a user. When a transaction is created by a user (by starting a new form), the initial form status is set to opened. This status will be changed once the user saves, cancels, or submits the transaction.
- Saved - transactions that have been saved by the user or automatically by the system.
- Submitted - transactions that have been submitted but may require additional information (attachments, payment, etc.). Once the required data is collected, the transaction status will change to Completed. This status is not common for Maestro forms as all information, such as attachments and payments, are collected prior to the user submitting the form. Most Maestro forms would skip the Submitted status and be marked as Completed after the form is submitted by the user. This is a legacy feature.
- Expired - a form used by this transaction has expired and is no longer available.

A delivery status can be one of the following:

- Completed - a transaction has been successfully delivered, based on the configured delivery channel.
- Undeliverable - a delivery is not possible. Check the error logs for further details of the error.
- Not Ready - a transaction is not ready for delivery yet. This is the start state before a transaction has been completed by the user. Generally, the receipt render service will kick the transaction from this state to the Ready state when the receipt is rendered.
- Not Required - a delivery is not required. It may be that this form is part of a collaboration job and the delivery is to be handled separately say as a consolidation of the transaction details from all the associated forms.
- Ready - a transaction is ready to be delivered. A transaction will be marked as Ready until the delivery channel is processed, or it may remain marked as Ready if a delivery channel is not configured.
- In Progress - a delivery has been started but is waiting to be completed.
- Paused - a transaction delivery is paused. This transaction's status can be changed by some other means, for example by a Groovy Scheduled Task.
- Error - a transaction delivery has failed and is waiting to be retried.
- Sent Email - a secure email has been sent and now a transaction is waiting for the user to log in and process it.
- Pending

Only transactions with the following combinations of statuses are purged:

- Form Status: Completed or Abandoned,
- Delivery Status: Completed, Undeliverable or Not Required.

Next, learn how to view collaboration job data retention summary.

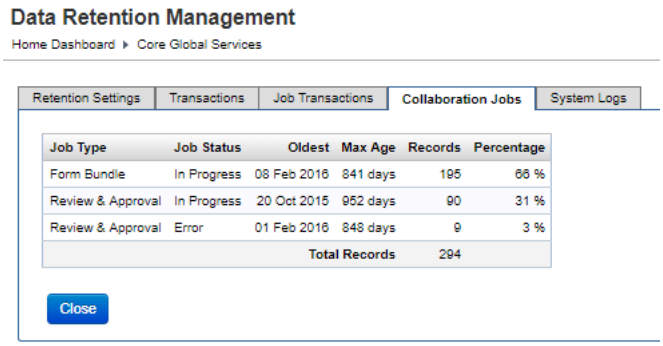
Data Retention Summary : Collaboration Jobs

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager allows you to view data retention summary of collaboration jobs.

To view collaboration job transactions data retention summary:

1. Select **System > Data Retention Management.**
2. Click the **Collaboration Jobs** tab to see the collaboration job data usage:



- Job Type, for example, Form Bundle, or Review and Approval.
- Job Status.

  A job status can be one of the following:

- In Progress - the collaboration job is active and in progress.
- Completed - the collaboration job has been completed. This happens when a endpoint step has completed.
- Expired - this happens when an endpoint expiry collaboration job has been automatically expired by Journey Manager.
- Cancelled - the collaboration job has been manually canceled by an administrator.
- Error - the collaboration job has a critical or unrecoverable error that requires an administrator intervention before it can be run again.
- Paused - the collaboration job has been paused temporarily.
- **Oldest** - the date of the oldest transaction for this group.
- **Max Age** - the maximum age of the oldest transaction for this group.
- **Records** - the number of transaction records for this group.
- **Percentage** - the percentage transaction records for this group of the total number of transaction records.

The list is grouped by Job Type and Job Status.

3. Click **Close**.

Only collaboration jobs with the Completed, Cancelled, and Expired combinations of statuses are purged:

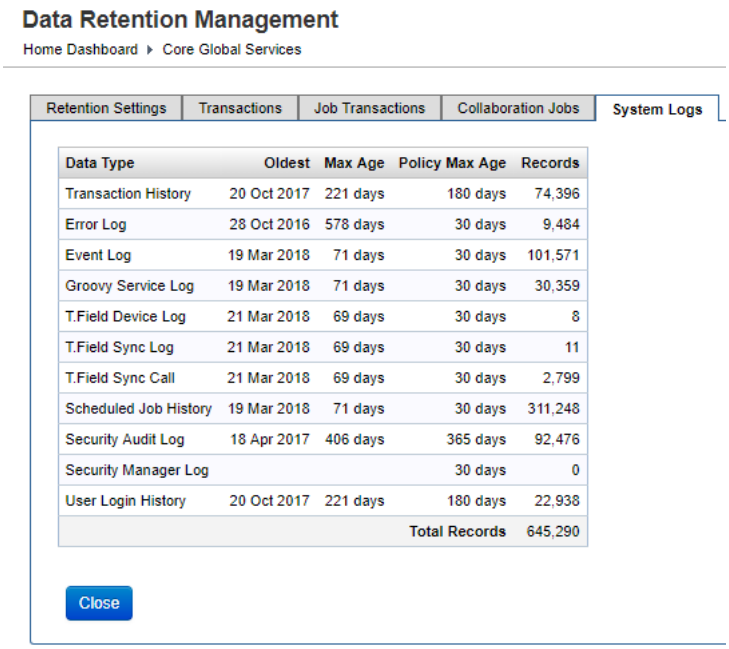
Next, learn how to view system log data retention summary.

Data Retention Summary: System Logs

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view data retention summary of system logs.

To view system logs data retention summary, select **System > Data Retention Management** and click the **System Logs** tab. The data usage summary is grouped by log file type.



Data Type	Oldest	Max Age	Policy Max Age	Records
Transaction History	20 Oct 2017	221 days	180 days	74,396
Error Log	28 Oct 2016	578 days	30 days	9,484
Event Log	19 Mar 2018	71 days	30 days	101,571
Groovy Service Log	19 Mar 2018	71 days	30 days	30,359
T.Field Device Log	21 Mar 2018	69 days	30 days	8
T.Field Sync Log	21 Mar 2018	69 days	30 days	11
T.Field Sync Call	21 Mar 2018	69 days	30 days	2,799
Scheduled Job History	19 Mar 2018	71 days	30 days	311,248
Security Audit Log	18 Apr 2017	406 days	365 days	92,476
Security Manager Log			30 days	0
User Login History	20 Oct 2017	221 days	180 days	22,938
			Total Records	645,290

- **Oldest** - the date of the oldest transaction for this group.
- **Max Age** - the maximum age of the oldest transaction for this group.
- **Policy Max Age** - the policy maximum age as specified in the data retention settings for your server instance.
- **Records** - the number of transaction records for this group.
- **Percentage** - the percentage transaction records for this group of the total number of transaction records.

System log records associated with a transaction are normally purged based on the standard data retention policy, as long as the transaction had already been purged. If not, it must wait for the transaction to be purged. If there are existing old transactions, then it is possible that the related system log entries will show the **Max Age** being greater than the **Policy Max Age**.

Next, learn how to view top table size data retention summary.

Data Retention Summary: Top Table Sizes

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view current top table sizes data summary.

This information is only available if a Manager instance uses a **MySQL**[1] DB.

To view top table sizes data retention summary, select **System > Data Retention Management** and click the **Top Table Sizes** tab to see the following data usage summary of 19 largest tables listed by their size in descending order.

Data Retention Management

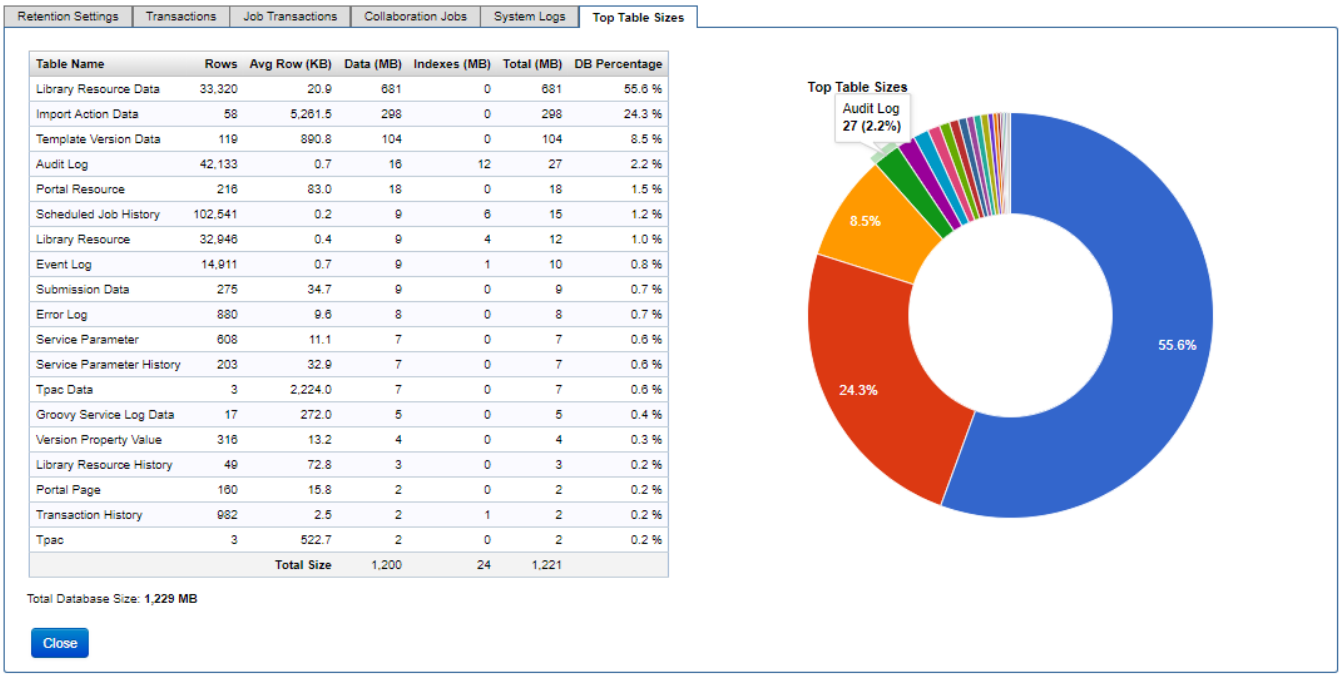


Table Name	Rows	Avg Row (KB)	Data (MB)	Indexes (MB)	Total (MB)	DB Percentage
Library Resource Data	33,320	20.9	681	0	681	55.6 %
Import Action Data	58	5,261.5	298	0	298	24.3 %
Template Version Data	119	890.8	104	0	104	8.5 %
Audit Log	42,133	0.7	16	12	27	2.2 %
Portal Resource	216	83.0	18	0	18	1.5 %
Scheduled Job History	102,541	0.2	9	6	15	1.2 %
Library Resource	32,946	0.4	9	4	12	1.0 %
Event Log	14,911	0.7	9	1	10	0.8 %
Submission Data	275	34.7	9	0	9	0.7 %
Error Log	880	9.6	8	0	8	0.7 %
Service Parameter	608	11.1	7	0	7	0.6 %
Service Parameter History	203	32.9	7	0	7	0.6 %
Tpac Data	3	2,224.0	7	0	7	0.6 %
Groovy Service Log Data	17	272.0	5	0	5	0.4 %
Version Property Value	316	13.2	4	0	4	0.3 %
Library Resource History	49	72.8	3	0	3	0.2 %
Portal Page	160	15.8	2	0	2	0.2 %
Transaction History	982	2.5	2	1	2	0.2 %
Tpac	3	522.7	2	0	2	0.2 %
		Total Size	1,200	24	1,221	

Total Database Size: **1,229 MB**

Close

[1]MySQL is an open-source relational database management system (RDBMS). It's an Oracle-backed open source relational database management system (RDBMS) based on Structured Query Language (SQL). MySQL runs on virtually all platforms, including Linux, UNIX and Windows. Although it can be used in a wide range of applications, MySQL is most often associated with web applications and online publishing.

- **Rows** - the number of transaction records for this table.
- **Avg Row (KB)** - the average row size in kilobytes for this table.
- **Data (MB)** - the data size in megabytes for this table.
- **Indexes (MB)** - the indexes size in megabytes for this table.
- **Total (MB)** - the total data size in megabytes for this table.
- **DB Percentage** - the percentage of total data size in megabytes for this table of the total megabytes for the top table sizes.

Next, learn how to view transactions data retention summary.

Server Nodes Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager uses a scalable architecture based on server nodes. Each server node corresponds to a **Java Virtual Machine**[1] (JVM) instance where a Manager server is installed. Each server nodes can be configured to host a combination of the Manager console, host form spaces, and operate as background PDF receipt generators.

Manager allows you to configure a single server node to host multiple servers. For example, a single server node can be configured to host both the form space server and the Manager server. Receipt generation is handled by an individual server node, but you can change this configuration if required.

Manager creates default server nodes when it is initially installed. For example, a server node dedicated to PDF receipt generation is created when Manager is installed. Though these server nodes are installed automatically, it is still important that you check and confirm the configuration (including the IP address and credentials) of every server node to ensure that each of them operates as expected in the Manager environment.

All server nodes share the same database.

Next, learn how to view server nodes.

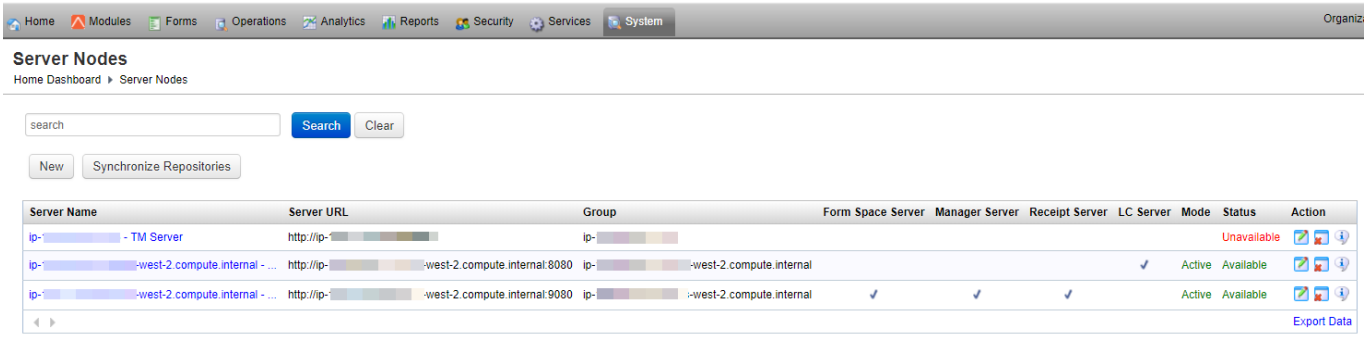
[1] Java Virtual Machine (JVM) is a software-based machine that runs Java programs. It can be installed on several different operating systems, including Windows, OS X, and Linux. JVMs allow Java apps to run on almost any computer.

View Server Nodes

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to view, create, configure, remove, and test server nodes.

To view the list of server nodes, select **System > Server Nodes**.



To search for a server node, type a text relevant to its configuration, for example 9080, and click **Search**.

Click **Clear** to reset the search criteria.

Each server node contains the following information:

- Server Name – that uniquely identifies the server node.
- Server URL – the URL that directs to the server node.
- Group – the local data server group to where the server node belongs.
- Form Space Server – this is a form space server that handles and hosts web traffic and form spaces.
- Manager Server – this is a Manager console server that supports development and operations staff.
- Receipt Server – this is a background PDF receipt generation server.
- LC Server – this is an Adobe LiveCycle Form server. This is shown only if the LiveCycle component is installed on this server.
- Mode - the mode of the server. This is shown only if there are more then one server nodes configured.

  A server node mode can be one of the following:

- Active
- Standby
- Inactive
- Status – the status of the server. This is shown only if there are more then one server nodes configured.

  A server node status can be one of the following:

- Available
- Unavailable
- Action – that can be: Edit, Remove or Check Health Status.

Click **Synchronize Repositories** to synchronize the LiveCycle repositories manually. Active and standby server nodes do it automatically.

You can also do the following from this page:

- Create a server node
- Configure a server node
- Remove a server node

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

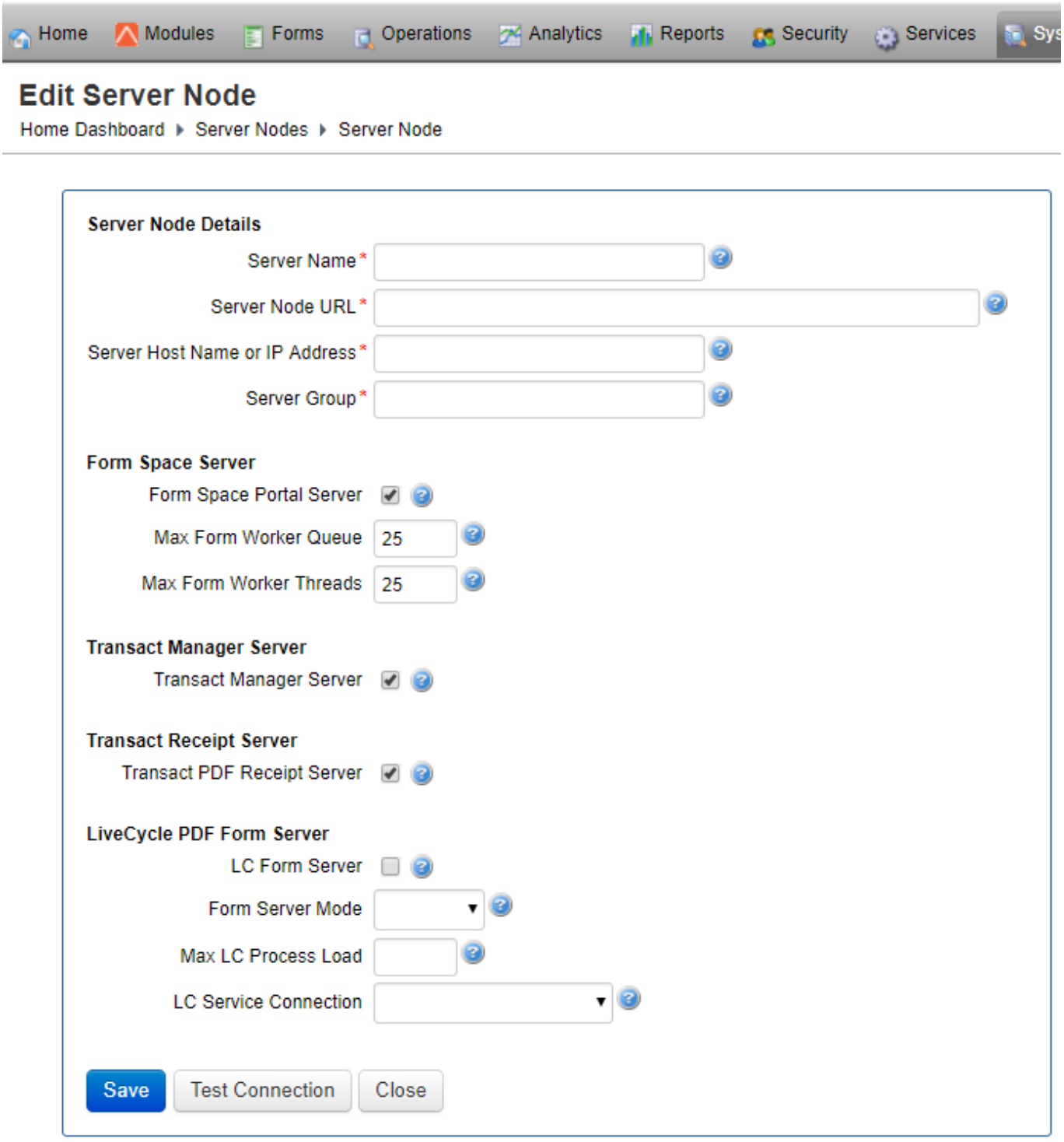
Create a Server Node

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager creates server nodes during its installation, but you can add new server nodes at any time when required.

> It's recommended to create server nodes for specific purposes such as to host form spaces or generate PDF receipts.

To create a new server node, select **System > Server Nodes** and click **New**.



Next, learn how to configure a server node.

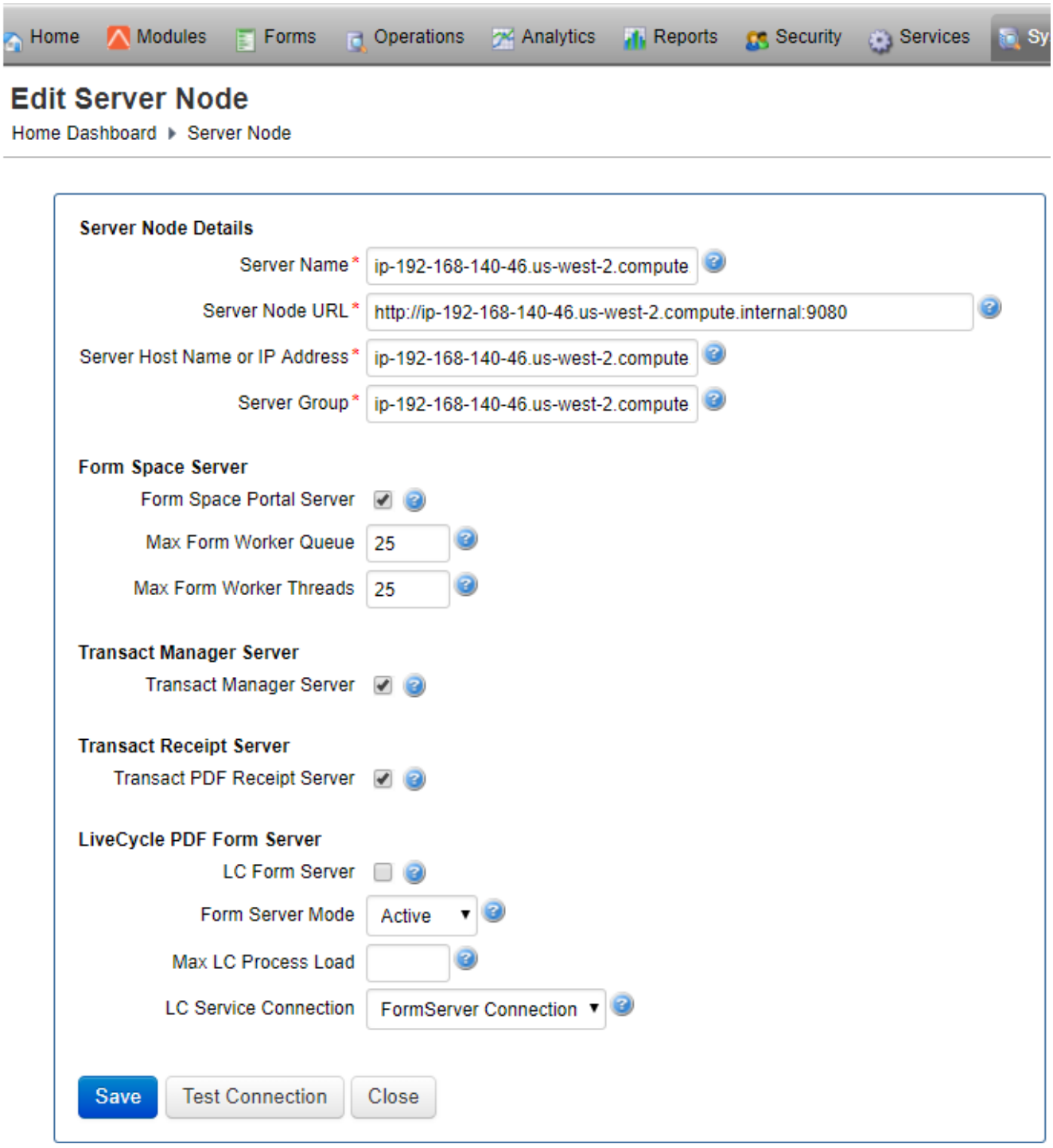
Configure a Server Node

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to create and configure server nodes.

To configure a server node:

1. Select **System > Server Nodes**.
2. Locate a service node and click **Edit**.



3. Provide the unique server name and server node URL.
4. Select the **Form Space Portal Server** checkbox if a server node handles public facing Internet traffic. It is not selected by default.
5. Change the **Max Form Worker Queue** value, which is a maximum length of the concurrent PDF form worker queue before throttling is imposed. The default is 25. | 18.05
6. Change the **Max Form Worker Threads value**, which is a maximum number of the concurrent PDF form worker threads. The default is 25. | 18.05

7. Select the **Journey Manager Server** checkbox to mark it as a Manager console server that supports development and operations staff.
8. Select the **Transact PDF Receipt Server** checkbox to mark it as a background PDF receipt generation server.
9. Select the **LC Form Server** checkbox if the server node is an Adobe LiveCycle Form Server.
10. Select the server node mode from the **Form Server Mode** dropdown list:
    - **Active** – if the node handles Form requests and keeps LiveCycle repository synchronized
    - **Standby** – is a hot standby server, which doesn't handle requests, but keeps the LiveCycle repository synchronized.
    - **Inactive** – doesn't handle form request neither keep the LiveCycle repository synchronized.

    To deactivate a server node for a server node, change its status to **Inactive**.

11. Change the **Max LC Process Load** value, which is the maximum concurrent LiveCycle process load (%) before throttling is imposed. The default is 16. | 18.05
12. Select the **LC Service Connection** from the dropdown list, which represents the LiveCycle Form Server local EJB connection detail:
    - FormServer Connection. | 18.05
13. Click **Test Connection** to establish a network connection to the server. The expected result is a successful message such as:
    `'ip-192-168-140-46.us-west-2.compute.internal - TM Server' health status - CPU: 5% Executing Processes: null Errors: null Update Time: 10:37:58 AM`.
    In case of an error message, check the server node configuration and re-test it again. The error message looks like:
    `'ip-192-168-140-46 - TM Server' is unavailable`. | 18.05
14. Click **Save** to update the changes.

You can configure a server node to host one or several Manager servers.

Next, learn how to remove a server node.

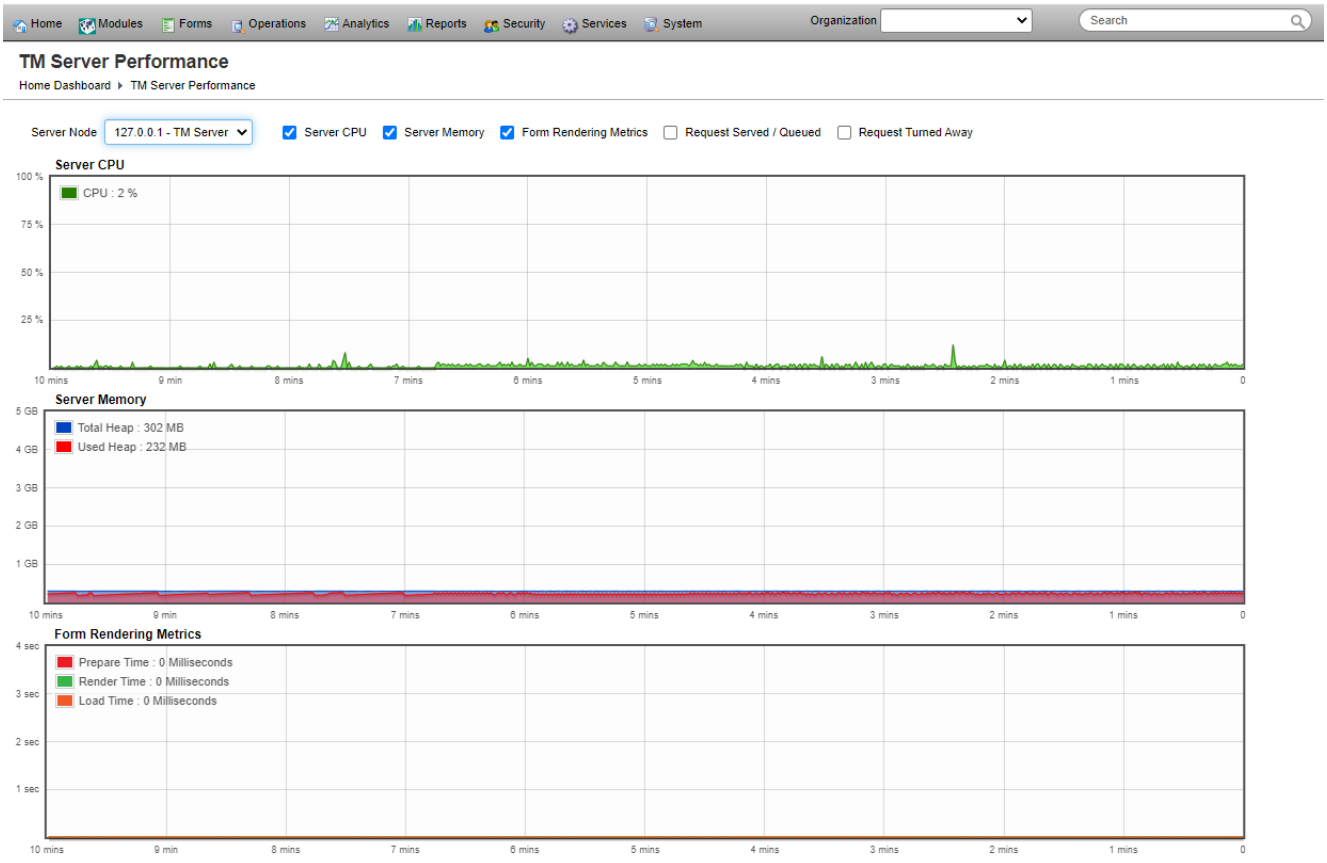
Monitor Server Node Performance

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager provides graphical visualization of a server node real-time performance, which you can use to troubleshoot server performance issues.

To view server node performance information:

1. Select **Operations > JM Server Performance**.
2. Choose a server node from the dropdown list.



3. Select one of the following performance metrics to visualize:
   - Server CPU
   - Server Memory
   - Form Rendering Metrics
   - Request Served / Queued
   - Request Turned Away

The selected performance graphs have 10 minutes time span, which can't be changed.

Next, learn how to view server nodes.

Remove a Server Node

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to remove a server node.

> It is recommended to change a server node mode to inactive rather than remove it.

To remove a server node:

1. Select **System > Server Nodes**.
2. Locate a server node.
3. Click **Remove**.
4. Click **OK** to confirm your decision.
5. Verify the server node has been removed from the list of server nodes.

Next, learn how to create a server node.

System Info

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager allows you to see some important server's information, which you can use to troubleshoot server performance issues.

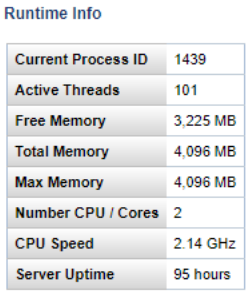
To view the server's system information, select **System > System Info**.

The system information is broken down into the following sections:

- Runtime Info
- Build Properties
- System Properties

Runtime Info

This section shows the current runtime load of the server, which is useful to check the server performance.

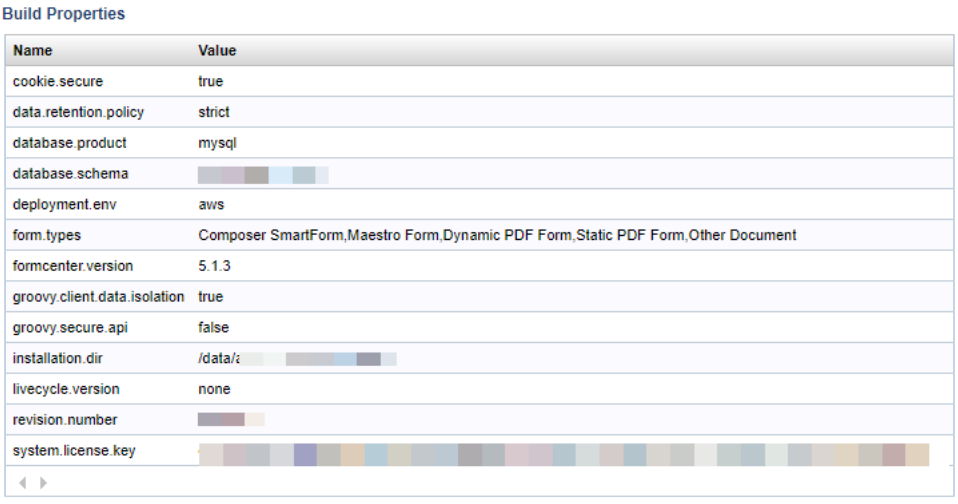


Runtime Info

Current Process ID	1439
Active Threads	101
Free Memory	3,225 MB
Total Memory	4,096 MB
Max Memory	4,096 MB
Number CPU / Cores	2
CPU Speed	2.14 GHz
Server Uptime	95 hours

Build Properties

This section shows the server build and installation configuration.



Build Properties

Name	Value
cookie.secure	true
data.retention.policy	strict
database.product	mysql
database.schema	
deployment.env	aws
form.types	Composer SmartForm,Maestro Form,Dynamic PDF Form,Static PDF Form,Other Document
formcenter.version	5.1.3
groovy.client.data.isolation	true
groovy.secure.api	false
installation.dir	/data/
livecycle.version	none
revision.number	
system.license.key	

Use the **deployment.env** property to check the deployment mode of Manager, which was chosen during its installation.

System Properties

This section shows the server **Java**[1] and **Groovy**[2] configuration.

System Properties

Name	Value
RUN_UNDER_ECLIPSE	false
[Standalone]	
awt.toolkit	sun.awt.X11.XToolkit
cayenne.server.contexts_sync_strategy	false
com.amazonaws.services.s3.enableV4	true
com.avoka.component.httpclient.connectTimeout	10000
com.avoka.component.httpclient.socketTimeout	60000
com.avoka.core.cayenne.query.cache.enabled	true
com.avoka.core.cayenne.query.cache.size	2048
com.avoka.core.cayenne.query.cache.timeout	30
com.avoka.core.cayenne.query.profiling.detail	false
com.avoka.core.cayenne.query.profiling.enabled	false
com.avoka.core.cayenne.query.profiling.threshold	30000
com.avoka.core.cayenne.query.timeout	600
com.avoka.core.groovy.script.cache.timeout	12
com.avoka.core.util.XmlUtils.enableDoctypeProcessing	false
com.avoka.core.util.XmlUtils.stripSurrogateCharacters	true
com.avoka.fc.core.servlet.backgroundSaveRevisionCheck	true
com.sun.jndi.ldap.connect.pool.protocol	plain ssl
com.sun.jndi.ldap.connect.pool.timeout	300000
1-20 of 109	

Use the **user.timezone** to check the server's timezone, which is important for correct database configuration, see configure timezone for MySQL database.

In some cases, you need to change some of the System Properties to implement your specific requirements or adjust Manager performance. You can do this via the Fluent API. For example, the Fluent API's **HttpRequest** base class has **setSocketTimeout()** and **setConnectTimeout()** methods, so you can set timeout values after creating a **GetRequest** or **PostRequest**. However, in most cases, you need to contact your System Administrator to request the change.

Next, learn how to check system health.

---

[1] Java is a programming language that produces software for multiple platforms. When a programmer writes a Java application, the compiled code (known as bytecode) runs on most operating systems (OS), including Windows, Linux and Mac OS. Java derives much of its syntax from the C and C++ programming languages.
[2] Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across the Journey platform, Groovy is used to create services in Manager. These services are primarily used to create plug-in style systems that interact with Manager.

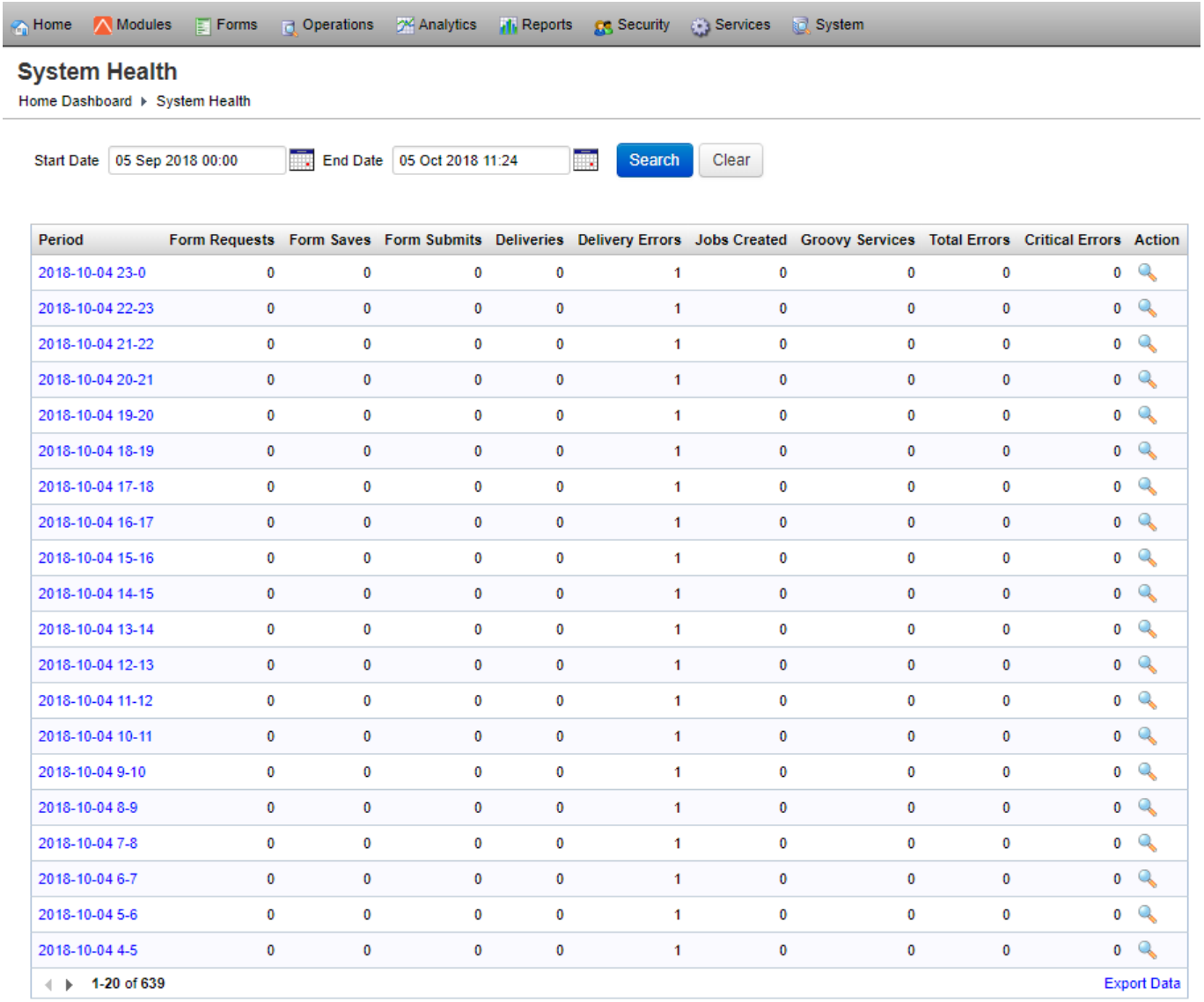
System Health

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager allows you to view server's system health information that includes transaction statistics, system info, and error log.

To view system health, select **System > System Health**.

System health records are shown page by page, with the maximum number of records per page configured in the users preferences.

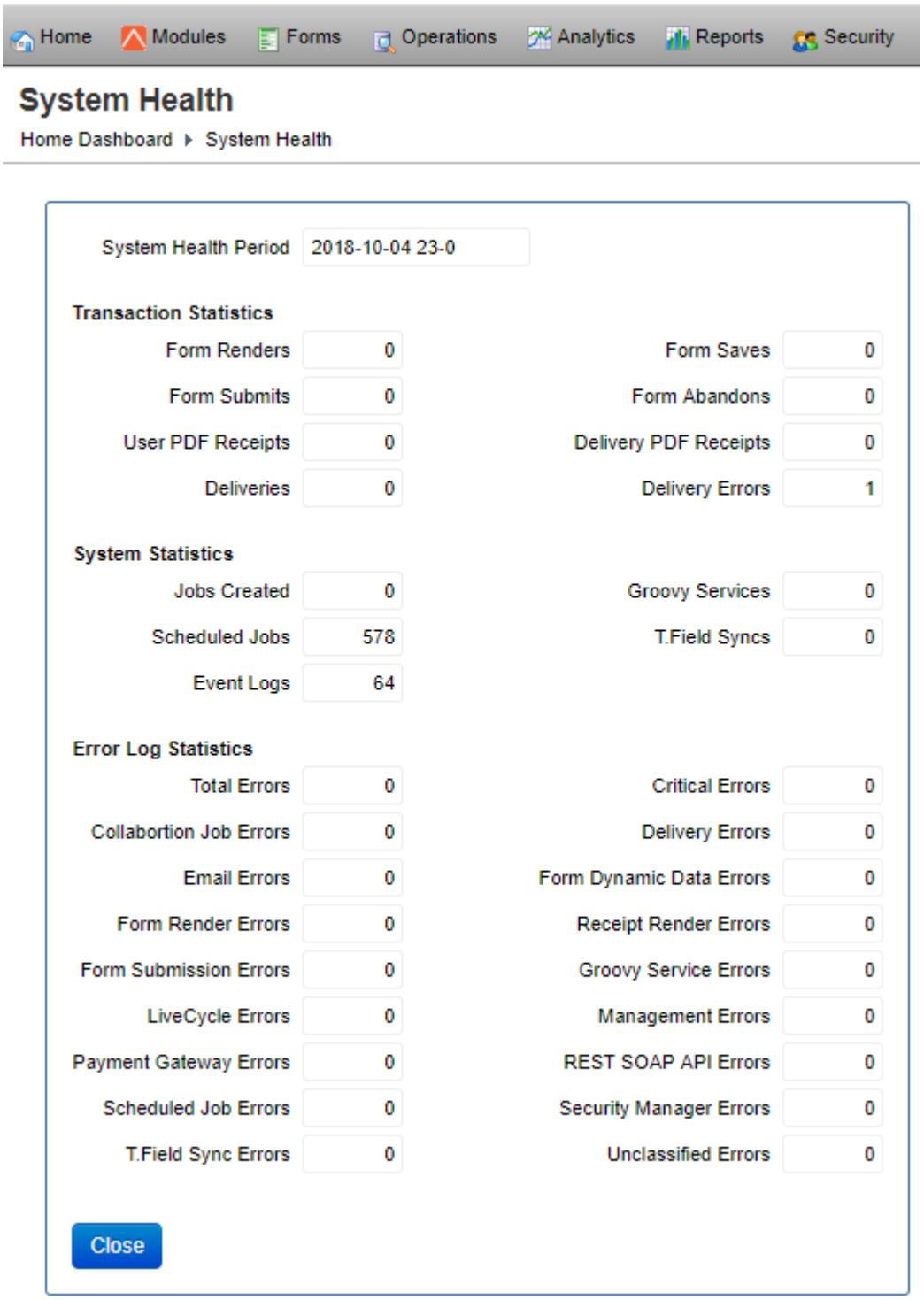


To filter or search system health records by one or more criteria, specify the following settings and click **Search**:

- Start Date.
- End Date.

Click **Clear** to reset the search criteria.

To view additional information, click **View**. The example below shows system health for a specified period.



Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

Next, learn how to check database version log.

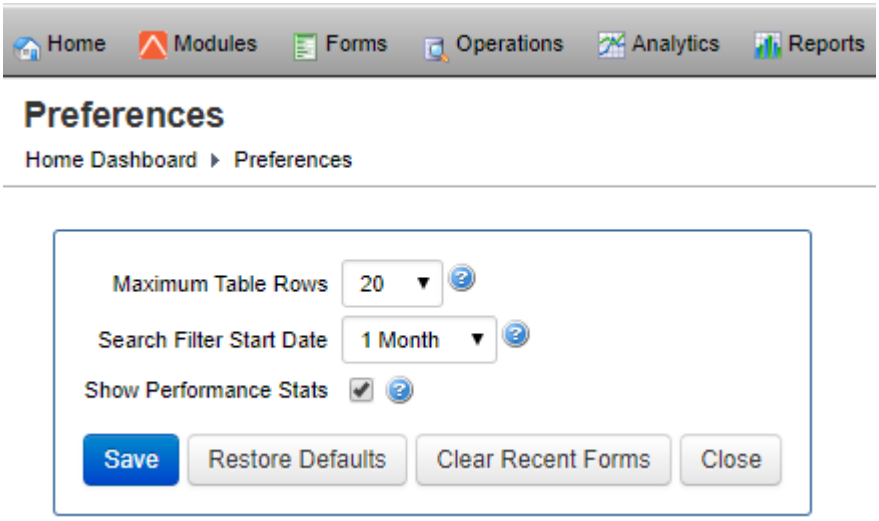
System User Preferences

Journey Manager (JM) | System Manager / DevOps | 18.05

Manager allows you to customize the user's preferences.

To configure the user's preferences:

1. Select **System > Preferences**.



2. Select the maximum number of table rows to display in one page from the **Maximum Table Rows** drop-down list. Default is 20.
3. Select the default search start date filter criteria from the **Search Filter Start Date** dropdown list. Default is 1 Month.
4. Select the **Show Performance Stats** checkbox to show performance statistics in the form requests, Groovy service log, and scheduled job history. It is selected by default.
5. Click **Restore Defaults** to restore your preferences to the default Manager preferences.
6. Click **Clear Recent Forms** to clear your form history.
7. Click **Save** to update the changes.

Next, learn how to check Manager system info.

System Licensing

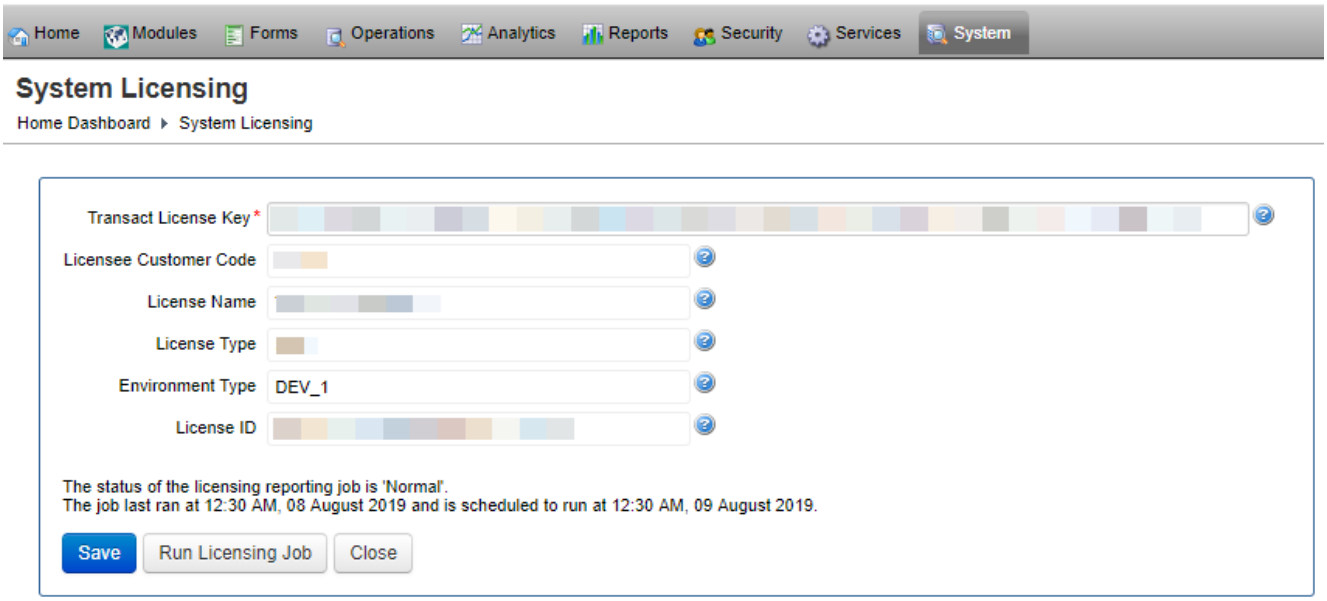
Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager is a licensed product, so you must have a valid license to run it. Our licensing model includes an automated transaction licensing report, which Manager generates periodically and sends it to our servers. The report contains daily summaries on the number of submissions, broken down by organizations and other metrics, such as number of completed transactions. For this to work, you must configure Manager with licensing details, which we provide to you.

> You must have an Administrator role to view and configure the license details.

To view and configure the system licensing details of your server:

1. Select **System > System Licensing**.



2. Enter the **Transact License Key** provided to you by Temenos. Manager will populated all other read-only fields below from this license key, so you don't need to enter any additional information.

> A separate license key must be entered for every Journey Manager instance as well as every Journey Manager version upgrade of the instance.

3. The **Licensee Customer Code** is provided by Temenos, for example, **My Company**.
4. The **License Name** is the name of your Journey Manager system, for example, **My Company UAT**.
5. The **License Type** is a type of the license of your Manager system.
6. The **Environment Type** you are licensed for, for example, **test**, **UAT**, or **production**.

7. The **License ID** is the unique code identifying your Manager system for licensing purposes.
8. Click **Save** to update the changes.
9. Click **Run Licensing Job** to run the transaction licensing report manually and upload the data to Temenos. You check the details of the last transaction licensing job, its status, time it was run, and when the next scheduled job will be run.

   > By default, the transaction licensing report is run nightly.

Next, learn how to check system health.

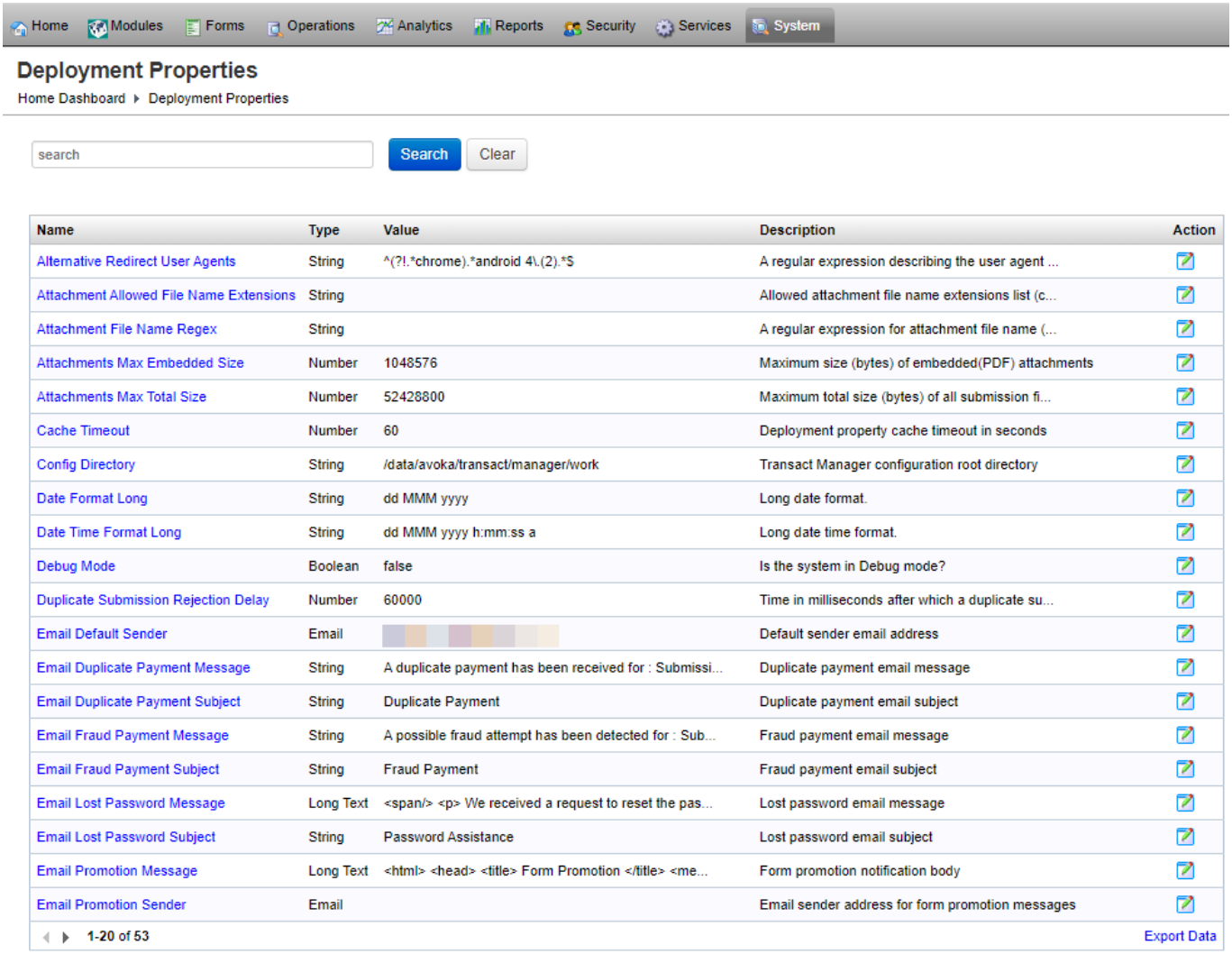
Deployment Properties

Journey Manager (JM) | System Manager / DevOps | 20.11

Manager comes with a set of deployment properties that allow you to control and customize applications' functionality without the need to rebuild them. You can view and configure deployment properties. Each property is initialized to its default value during Manager installation, but you can change it. However, you must understand the impact of the change as it may affect various parts of Manager.

To view the deployment properties, select **System > Deployment Properties**.

Deployment properties are shown page by page, with the maximum number of records per page configured in the users preferences.



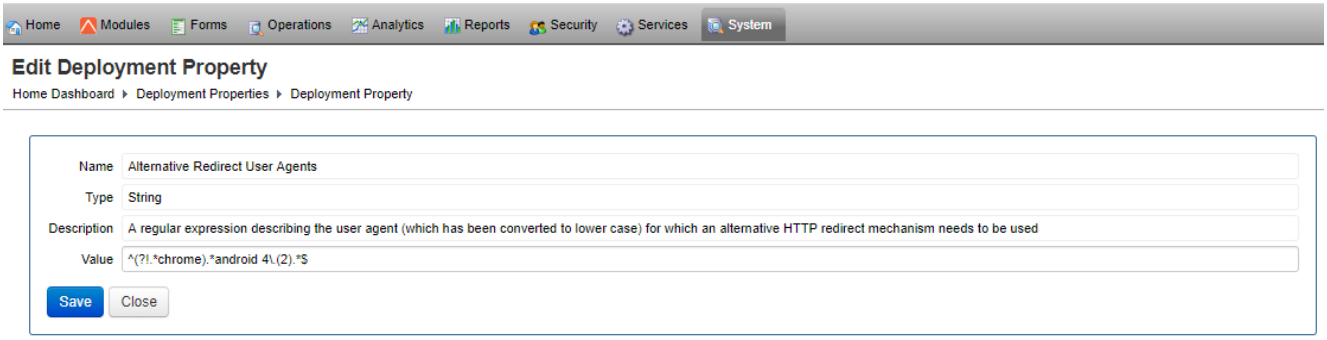
To filter or search deployment properties, enter the property name or part of the name and click **Search.**

Click **Clear** to reset the search criteria.

Click **Export Data** to export the displayed data as a Microsoft Excel (XLS) file.

To configure a deployment property:

1. Locate a deployment property and click **Edit**.



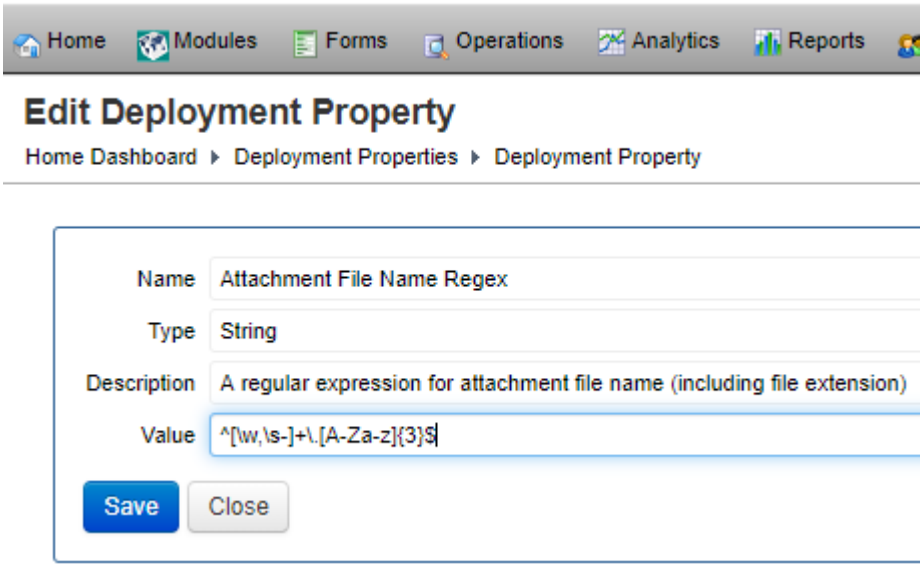
2. Update the property value in the **Value** field.
3. Click **Save** to update the changes.

Default Deployment Properties

Manager has the following default deployment properties:

Property Name	Description
Alternative Redirect User Agents	Describes a set of user agents requiring alternative HTTP redirect mechanisms
Attachment Allowed File Name Extensions	Allowed attachment file name extensions list (comma separated list)
Attachment File Name Regex	A **regular expression**[1] (regex) for validating a file name, including a file name extension, when a customer uploads an attachment. This is the system level deployment property. If it doesn't exist, then the attachment functionality operates as normal. If the regex fails, the attachment upload operation fails. You can use a standard regular expression, for example, `^[\w,\s-]+\.[A-Za-z]{3}$`

[1] A regular expression (regex) is a sequence of characters that define a search pattern. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation.



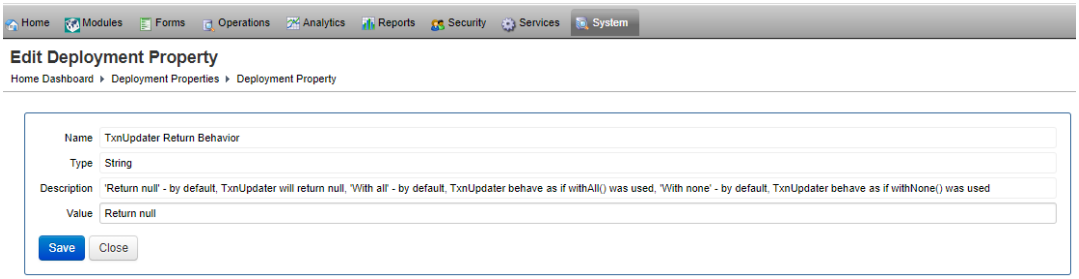
| 20.11

Attachments Max Embedded Size	The maximum size of a single embedded (PDF) attachment
Attachments Max Total Size	The global maximum combined size of all attachments for a single submission. Contact your System Administrator or cloud service team to request this value changed.
Cache Timeout	Deployment property cache timeout in seconds
Config Directory	Manager configuration root directory.
Date Format Long	Long date format.
Date Time Format Long	Long date time format.
Debug Mode	System is in Debug mode, if it's set to true.
Duplicate Submission Rejection Delay	Time in milliseconds after which a duplicate submission is rejected.

Email Default Sender	Default sender email address for emails sent by Manager.
Email Duplicate Payment Message	Body text template for emails notifying administrators of a duplicate payment.
Email Duplicate Payment Subject	Subject template for emails notifying administrators of a duplicate payment.
Email Fraud Payment Message	Body text template for emails notifying administrators of a possibly fraudulent payment.
Email Fraud Payment Subject	Subject template for emails notifying administrators of a possibly fraudulent payment.
Email Lost Password Message	Lost password email message.
Email Lost Password Subject	Lost password email subject.
Email Promotion Message	Form promotion notification body.
Email Promotion Sender	The email address for the promotion notification messages.
Email Promotion Subject	Form promotion notification subject.
Email Report Message	Report delivery email message.
Email Report Sender	Report delivery sender email address.

Email Report Subject	Report delivery email subject.
Email Submission Status Message	Submission processing status update email message. You can overwrite it on the organization and form level by using organization and form version properties.
Email Submission Status Subject	Submission processing status update email subject. You can overwrite it on the organization and form level by using organization and form version properties.
Email Task Notification Message	Task notification body.
Email Task Notification Subject	Task notification subject.
Email Task Sender	The email address shown as the sender for task notification messages.
Email Welcome User Message	User welcome email body.
Email Welcome User Subject	User welcome email subject.
Enable REST/Web Service Delivery	Enable REST and web service delivery for submissions.
Enable Script Submissions	Enable test submission scripts by switching off submission request log checks.
Enable Web Service Delivery	Enable web service delivery. This can be used to temporarily suspend web service delivery, for example, during high load or web service maintenance.

Groovy Logging Enabled	Enable logging of Groovy service calls.
Groovy Script Timeout	Groovy script execution timeout in milliseconds.
HTTPS Port	Port on which HTTPS operates.
LC Form Servers Deployed	If this instance of Manager does not use any Adobe LiveCycle servers, set this property to "false"
LC Form Servers Load Balanced	Enable this setting to use load balancing when assigning tasks to Adobe LiveCycle nodes.
LiveCycle Repository Domain	Relative location of LiveCycle repository domain for form version publishing.
Online Save Enabled	Option to globally disable Online Saving of forms. If set to true, the individual form's setting will be used.
Report Internal URL	Location where reports are run internally by the application, that is **http://-localhost:9080/webreport**
Report Reschedule Delay	Amount of time in minutes to wait before rescheduling a running report.
Server Real Time Health Monitoring	Enable real-time server health monitoring. Set it to false to disable the Manager server metrics on your local server instance temporarily.
SMTP Host	SMTP email server hostname (**mail.smtp.host**).
SMTP Password	SMTP email server authentication Password (**mail.smtp.password**).
SMTP Port	SMTP email server Port (**mail.smtp.port**).
SMTP User	SMTP email server authentication User (**mail.smtp.user**).

Supported Delivery Methods	Supported delivery methods.
TxnUpdater Return Behavior	This allows you can specify the default behavior of the **TxnUpdater** to control the unforeseen performance side affects of the previous change to the **TxnUpdater** when asked to return a **Txn** object. Now, you can specify the following options for TxnUpdater Return Behavior: • **Return null** - **TxnUpdater** will return null. This is the default option. • **With all** - **TxnUpdater** behaves as if **withAll()** was used. • **With none** - **TxnUpdater** behaves as if **withNone()** was used.  | 19.05

Important Deployment Properties

You should modify the following deployment properties after setting up your Manager:

Directories

Manager uses directories to store various objects such as attachments and report files. By default, these files are stored in a sub-folder of the folder where the Manager server is installed. You can adjust this by setting the **Config Directory** property to any directory you want. However, make sure that the service has read and write access to the directory.

Edit the property **LiveCycle Repository Domain** and enter the path, relative to the LiveCycle repository, where Manager should publish its forms. If multiple Manager instances are sharing the same LiveCycle repository, make sure to enter a unique path, for example, **/tm-test-thirdparty/JourneyManagerRepository/**.

SMTP Server

If you want Manager to send emails, for example, for delivery purposes, you have to configure the SMTP settings.

Duplicate Form Submissions

Manager offers some options to control duplicate submissions, which are multiple submission attempts of a requested form. This can happen either because the user attempts to submit the same form multiple times (by double-clicking the submit button or saving the form offline and then opening multiple copies of it) or commonly in system testing, where scripts are used to automate a number of submissions.

By default, Manager blocks duplicate submissions when they are more than a certain time interval apart, which is configured using the **Duplicate Submission Rejection Delay** property. If duplicate submissions occur within a certain short period, Manager assumes that they originate due to accidental double-clicking of the submit button and ignores the duplicates.

To enable duplicate submissions , for example, for load testing, change the **Enable Script Submissions** deployment property to **true**.

To customize the message displayed to form space users when a duplicate submission is rejected, use the in-built form property **Submission Duplicate Message**.

Journey Analytics Deployment Properties

Manager provides you with the following Journey Analytics deployment properties:

Property Name	Description
Insights Application ID	Journey Analytics Application ID. This should be **PROD** for production instances. Must be one of **PROD**, **DEV**, **DEV1**, **DEV2**, **DEV3**, **LOCAL**, **TEST**, **TEST50** or **STAGING**.
Insights Enabled	Enable Journey Analytics.
Insights Version	Journey Analytics version identifier. Leave blank for the default version.

Next, learn how to configure modules.

Enable Deprecated Features

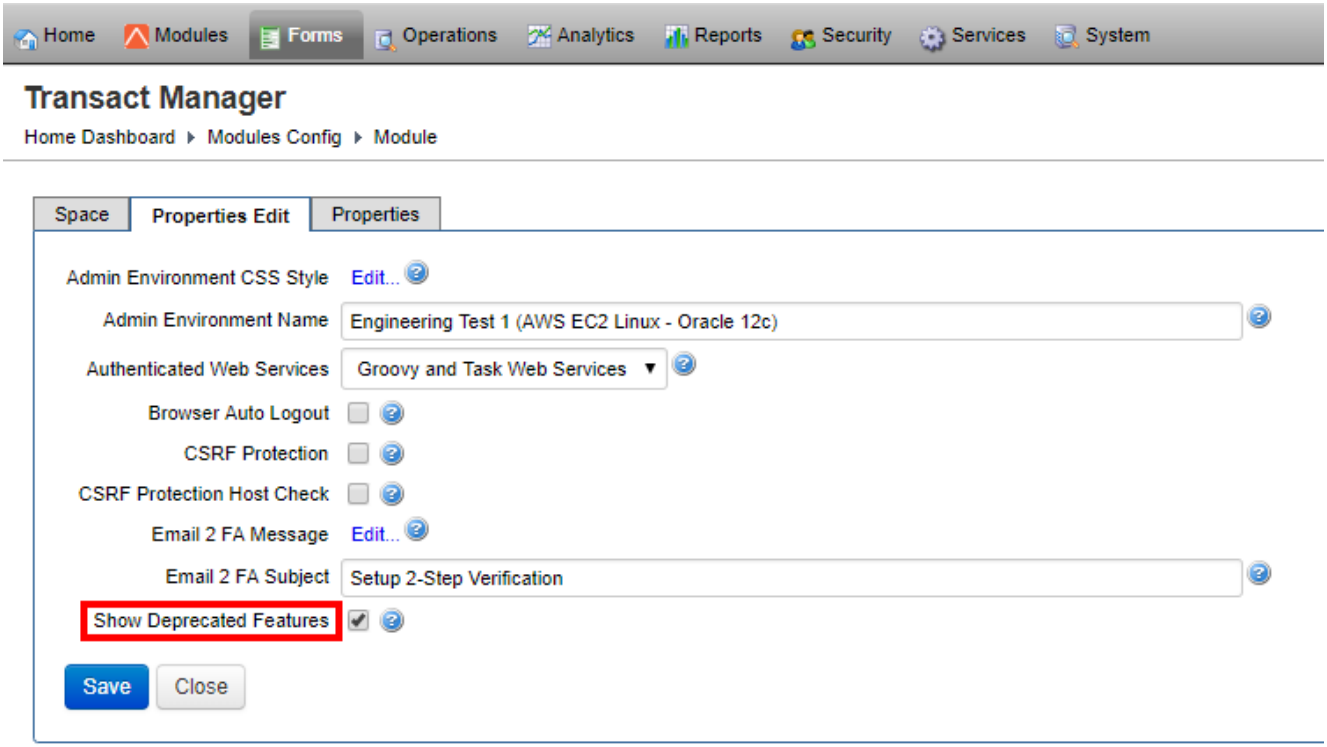
Journey Manager (JM) | System Manager / DevOps | | 21.11

Manager allows you to enable deprecated features, if you depend on them in your current installation and configuration.

> Manager 18.05+ removes deprecated features of Manager module by default. You can, however, enable these features if they're required.

To enable deprecated features:

1. Select **System > Modules Config**.
2. Locate the **Transact Manager** module and click **Edit**.
3. Select the **Properties Edit** tab.



4. Select the **Show Deprecated Features** checkbox.
5. Click **Save** to update the changes.

The list below identifies the current deprecated features that support the **Show Deprecated Features** property.

Version	Deprecated Features	Details

18.05	Forms menu options.	New Form Wizard (also the New button from the Forms page is removed). Document Types, Metadata Tag Definitions.
18.05	Specific form selection.	The Form Promotion tab from the form dashboard. For a selected version of the form, the Form Tags tab.
18.05	The Organizations option from the Forms menu bar when a specific organization is selected.	The Form Tags tab. Switch to the Delivery Channel tab. By adding or editing a delivery channel where the Delivery Method is REST Service, the PDF Receipt Embed Files checkbox is removed.

The following deprecated features can't be enabled using the **Show Deprecated Features** property.

Version	Deprecated Features	Details
21.11	TransactField App functionality removed	Removed support for the TransactField App product as it has reached End-Of-Life, which also includes T.Field Device Log and T.Field Sync Log.
21.11	FOP removed	FOP receipt service has been removed.
21.05	Remove the PDF export option from licensing reports	Remove the PDF export option (the **Download PDF** button) from the Transaction and Job Licensing reports because it uses **iText**[1], which requires a commercial license. The Excel spreadsheet export is still supported.
21.05	Wet Signature Template	LiveCycle Wet Signature receipt generation removed.

Next, learn about all Journey Manager features.

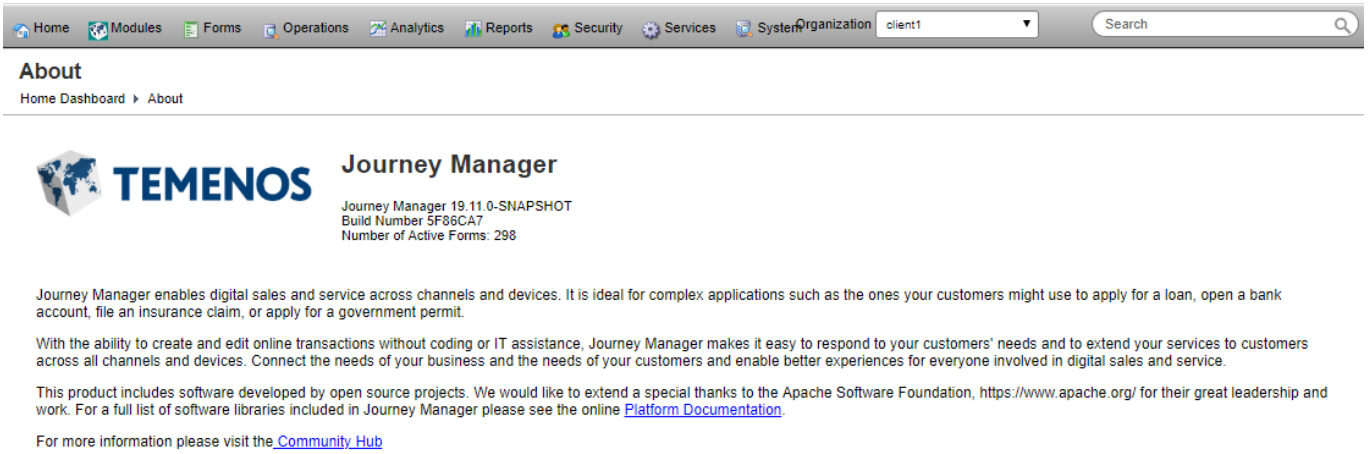
[1] iText is a library for creating and manipulating PDF files in Java and .NET.

About

Journey Manager (JM) | System Manager / DevOps | 19.05

Journey Manager allows you to view the product's basic information, such as a product **version**[1], a build number and number of active forms. You may want to use this information when updating Manager software or contacting our customer service to troubleshoot any issues.

To view this information, select **System > About**.



Click **Platform Documentation** to open out documentation website in a new browser window.

Click **Community Hub** to open our community website in a new browser window.

Next, learn how to check system health.

[1]Semantic Versioning is a formal convention for specifying compatibility using a three-part version number: major version; minor version; and patch. The patch number is incremented for minor changes and bug fixes which do not change the software's application programming interface (API). The minor version is incremented for releases which add new, but backward-compatible, API features, and the major version is incremented for API changes which are not backward-compatible. For example, software which relies on version 2.1.5 of an API is compatible with version 2.2.3, but not necessarily with 3.2.4.

Multi-Environment Support

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Typically, you need several Manager instances to develop, test, and deploy a Maestro form into production. These are often referred to as environments.

An environment can be hardware, a **virtual machine**[1], or a **cloud**[2]. Several Manager instances can be installed and run on the same or different environments.

As you develop and enhance Journey Maestro forms and Manager configurations, you promote or migrate them from one environment to another until they are in the production environment. Normally, you implement a form in the development environment, test it in one or several test environments, and finally deploy it in the production environment.

Any Manager instance allows you to migrate forms and organizations from one environment to another. Additionally, you can emulate form migration by using the Manager Form Test Mode feature.

Next, learn how to promote your Manager to another environment.

---

[1] A virtual machine (VM) is a software program or operating system that not only exhibits the behavior of a separate computer, but is also capable of performing tasks such as running applications and programs like a separate computer. A virtual machine, usually known as a guest is created within another computing environment referred as a "host." Multiple virtual machines can exist within a single host at one time.

[2] Cloud computing is shared pools of configurable computer system resources and higher-level services that can be rapidly provisioned with minimal management effort, often over the Internet. Cloud computing relies on sharing of resources to achieve coherence and economies of scale, similar to a public utility.

Multi-Environment Migration

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Journey Manager allows you to migrate or promote its assets from one environment to another. You must migrate them in the following order:

1. Services
2. Form spaces
3. **Security managers**[1]
4. Users
5. Roles
6. Organizations
7. Forms
8. Groups

For example, to migrate a space and an organization with all its forms from one environment to another, you must import them in this order:

1. A form space
2. An organization
3. Forms

You should review the server and environment specific configurations of the component before the migration, because it might need to be updated in the target environment. This is especially true for the following components:

- Services that contain endpoint URLs.
- Spaces that contain the context path.

[1]Security managers are the Journey Manager's components that are responsible for User Authentication, User Authorization, User Self-Registration, and Session Management. Security mangers are designed to handle various sets of users requiring access to different form spaces and can be associated with one or more form spaces. Users can be external or internal, and they can require access to form spaces or a combination of form spaces and modules in different environments. You can find security managers under Security > Security Managers.

Manager Deployment and Rollback Procedure

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

This guide covers a customer solution deployment and rollback specifics for an initial deployment and environment establishment. It provides a list of appendices, which is designed to provide any customer specific information, that need to be completed by the customer prior to any solution deployment or rollback.

Deployment

It is recommended to create a new version of the configuration and keep a previous one, if possible. Then you can make use of the new configuration by assigning it to your space in the Spaces tab. It will allow you to roll back to the previous configuration quickly and reliably.

Security Managers

Promote the SSL Certificate Store security manager definition as per the Journey Manager security manager environment migration process.

1. Generate the SSL trust Java keystore as outlined in Appendix - SSL Keystore Generation
2. Generate the SSL private Java keystore as outlined in Appendix - SSL Keystore Generation
3. Navigate to the Security Manager Certificates tab.
4. Upload the trust keystore as the Validation Certificate File.
5. Upload the private keystore as the Java Keystore File.
6. Enter the private key alias and password details into the Private Key fields.
7. Navigate to the Security Manager Parameters tab.
8. Enter the trust keystore password details into the truststorePass parameter.

Organisations

Promote the organisation definitions as per the Journey Manager organisation environment migration process. The list of organization definitions can be found in Appendix - Organizations Environment.

Form Spaces

Promote the form spaces definitions as per the Journey Manager form space environment migration (high availability) process. The list of form spaces definitions can be found in Appendix - Form Spaces Environment.

1. Navigate to the Form Space Space tab.
2. Update the Context Path URL field as per Appendix - Form Spaces Environment.

Organisation Applications

Promote the organisation applications as per the Journey Manager organisation application environment migration process. The list of organisation applications can be found in Appendix - Organization Applications Environment.

Service Connections

1. Navigate to a service connections definitions in Journey Manager. The list of form service connections definitions can be found in Appendix - Service Connections Environment.
2. Update the **Endpoint** as per Appendix - Service Connections Environment.

Verification

Execute post-deployment verification checks as per Appendix - Verification.

Rollback

You can make use of the previous configuration by assigning it to your space in the Spaces tab. It will allow you to roll back to the previous configuration quickly and reliably.

Security Managers

Revert to the previous security manager version or a backup copy as per the Journey Manager security manager environment migration process.

Organisations

Revert to the previous organisation version or a backup copy as per the Journey Manager organisation environment migration process.

Form Spaces

Revert to the previous form space version or a backup copy as per the Journey Manager form space environment migration (high availability) process.

Organisation Applications

Revert to the previous organisation application version or a backup copy as per the Journey Manager organisation application environment migration process. Please refer to the application names listed in the Deployment chapter.

Service Connections

1. Navigate to a service connections definitions in Journey Manager. The list of form service connections definitions can be found in Appendix - Service Connections Environment.
2. Update the **Endpoint** as per Appendix - Service Connections Environment.

Verification

Execute post-deployment verification checks as per Appendix - Verification.

Appendix - SSL Keystore Generation

Document the process to generate SSL trust and private keystore files here.

Appendix - Organizations Environment

Document the process to update Organization configuration.

If you do not have a change control in place, document the current (working) and the new configuration here, so this information can be used during the rollback.

Appendix - Form Spaces Environment

Document the process to update Form Spaces configuration, e.g. Context Path URL.

Appendix - Organization Applications Environment

Document the process to update Organization Applications configuration.

Appendix - Service Connections Environment

Document the process to update Service Connections configuration, e.g. Endpoint URL.

Appendix - Verification

Document the process to verify the deployment and rollback success.

Rollback Overview

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

Manager is a complex and highly configurable software, which allows you to create, import and update majority of its components, such as an organization, form space, form version, service, user, role, group, and security managers. Most of the changes include customized Groovy scripts and configuration files, for example, collaboration jobs, that can be long and complex and are deployed manually. Even though, developers and testers do their best to ensure there are no errors or flaws in these changes, errors do happen and we have to deal with them. Manager is designed to deal with these possible error scenarios, but each component does it differently. The general approach is to rollback the faulty changes to the previous working version.

You can rollback the following components:

- Forms and applications
- Configurations
- Security Managers
- Organizations
- Forms spaces
- Service Connections
- Services

Next, learn about the form dashboard.

Rollback Configuration

Journey Manager (JM) | System Manager / DevOps | 5.1 & Higher

If the results of the triage indicate that some configuration is incorrect, for example, someone has put wrong details or updated it with a typo, you can try to rollback to a previous working configuration. There is no easy way to revert to previous configurations in Journey Manager, so each configuration change may have its own rollback procedure. You must relay on your change logs to find the previous working configuration and then do it manually via Manager.

It is recommended to version any new configuration changes, for example, a new security manager v2, and keep the previous ones. Then you can make use of the new configuration by assigning it to your form space. It will allow you to roll back to the previous configuration quickly and reliably.

To give a practical example, let's assume you need to change your current Local Security Manager. Instead of changing its configuration directly, you create a new Local Security Manager version 2 and populate it with the configuration of the current one. Then you update some specific configuration according to your requirements and assign the new security manager with your space. If you discover later than the new configuration stopped working by some unknown reasons, you can easily roll back to the previous Local Security Manager version 1 by re-assigning it to your form space.

Even if you have managed to fix a problem by rolling back a configuration change, you should always ensure that you determine why it changed, and how to prevent it being changed again. In particular, application-level configuration changes may be overwritten when a new version of the application is deployed. This type of change needs to be updated in the application at the source-code level.

> In some cases, you may need to contact our Customer Care team to get help in restoring your Manager configuration.

You can also use the configuration rollback as a way to promote or migrate it from one to another Manager server, for example, in different environments.

Next, learn how to promote the Journey Manager to another environment.

Manager FAQ

Journey Manager (JM) | System Manager / DevOps | 21.05

General

How to verify 2FA PIN from Amazon SNS was sent using SMS?

You can check the transaction properties to see if the **MFAToken** exists there. This property should give a created date as well (**createdDate**).

How to redirect a form user to an external URL instead of a Journey Manager form?

You can set up a service on the specific form which throws a **RedirectException** with a redirect URL under the right conditions. A code snippet is shown below.

```
import com.avoka.tm.util.RedirectException

//....
if (!params.request.getParameter("email")){
    throw new RedirectException("http://www.google.com")
}

//...
```

If you need to submit a form at that point in time, you can use the Confirmation Page URL property.

How to add a Java library to Journey Manager?

If implementing your new business functionality requires a new Java library, which is not part of the product release, you have the following options:

- Try to use an existing library that is part of the product release
- Request us to add the library, so we will evaluate if this library provides an essential missing functionality and then add it to the new release
- Request us to whitelist the library, so you can use it in the new release. For more information, see **Secure Java Package Whitelist** in your SDK version.

-

How to turn off the auto-confirmation email?

If you don't want auto-confirmation emails to be sent to users, you need to set the **Send Confirmation Email to User** option to **None** for your form in Manager. You also need to clean the **Auto-Send Confirmation Emails** checkbox in the Form Options of your form in Maestro.