

Journey Platform

DOCUMENTATION



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

Welcome to the Temenos Journey Platform Documentation! **72**

Topics 72

Badges 72

Version agnostic content 73

Copy URL button 73

Documentation Navigation Options 73

Search 73

Search filters 75

Google Search 75

Navigation Bar 75

Quick Menu 75

Table of Contents 76

Resources 77

Feedback 77

What's Next? 78

Temenos Journey Platform Overview **79**

AWS Cloud Deployment 81

Azure Cloud Deployment 82

Products 82

Cloud hosted Journey Platform VDC Editions **85**

Introduction to Organizations **88**

Journey Executive Reading Path **90**

Journey Starter Reading Path **92**

Journey Manager Overview **95**

Journey Maestro Overview **99**

Journey Exchange Overview **102**

Contacting the Exchange Team **104**

Exchange Package Architecture **105**

Exchange Development Schedule **106**

Exchange Partnerships **107**

Exchange Developer Resources **108**

Journey Workspaces Overview **109**

What is Workspaces? 109

Who is Workspaces for? 109

Getting Started 110

Configuration 110

Browser Support 110

What's new? 111

Journey Analytics Overview **114**

Analytics Views 114

Journey Analytics User Interface 115

Terminology 116

Transaction Status 116

Strict No PII Policy 116

Architecture 116

Setup and User Administration 117

Retail DAO Overview **118**

Audience 119

Areas of Value 119

Prerequisites 119
What Remains to Be Done 120
Adobe Reader PDF Form 121
Adobe Reader Not Found 121
Adobe Reader Update Required 121
Adobe Reader JavaScript is Disabled 121
For more information on particular browsers, see: 122
Unsupported Browsers 123
Google Chrome 123
To disable Chrome plugins 123
Microsoft Edge 124
View PDF Documents in Chrome 125
Chrome Troubleshooting 126
Check for updates for Chrome, Adobe Reader and Acrobat 126
Check your Adobe Reader and Acrobat settings 126
Repair your Adobe Reader or Adobe Acrobat installation 127
Uninstall either Adobe Reader or Adobe Acrobat installation 127
Windows 127
Mac 127
Uninstall and Reinstall Adobe Reader or Acrobat 127
View PDF Documents in Firefox 128
Using the built-in PDF viewer 128
Using a PDF reader plugin 128
If plugin is not working 129

Testing the Adobe Reader Plugin 130

If PDF files are downloaded instead of displayed, try these solutions 130

View PDF Documents in IE 8 and IE 9 ..... **131**

Using an Add-on to view PDF documents within an IE8 browser window 131

1. Make sure that the system meets the requirements for the Acrobat product you use. 131

2. Check your IE add-on 132

3. Make sure that the ActiveX folder contains files required for web browsing. 134

4. Use a supported version of the browser. 134

5. How to turn on and off add-ons 134

6. To delete Active X controls 134

View PDF Documents in IE 10 ..... **136**

Using the built-in PDF viewer 136

Check your Browser Settings 136

Still having Issues? Try these steps 137

Method 1: Disable Add-Ons 137

Method 2: Reset security settings for Internet Explorer 137

Method 3: Run the Internet explorer performance troubleshooter and check if it helps. 138

Method 4: Reset Internet Explorer 10 138

Method 5: uninstall the Adobe Acrobat Reader and install the latest version. 138

View PDF Documents in Safari **139**

System requirements 139

Configuring Acrobat to display PDF files in Safari 140

Troubleshooting issues in Safari 140

Known Issues 140

Reader's Preferences > Internet > Display PDF in Browser is always dimmed 140

Uninstalling an older version of Acrobat or Reader after Acrobat X or Reader X has been installed removes the AdobePDFViewer plug-in 140

Older versions of the AdobePDFViewer plugin can overwrite newer versions of the plug-in 140

View PDF Documents in Windows 8 141

Using Acrobat Reader to be the Default PDF viewer 141

TransactField App Overview 143

Getting Started with TransactField App 145

Installation 145

Configuration 146

Log in 148

Self-registration 148

Credentials provided to you 148

Use your existing organizational credentials 148

How to Use TransactField App 150

Synchronizing 151

Disconnected 151

WI-FI 152

Navigating around TransactField App 152

Forms in TransactField App 157

Types of Forms 157

Widgets 157

Fill out a Form 157

Task List in TransactField App 161

Using the Task List 161

Use a Build-in Camera in TransactField App 165

Barcodes 165

Supported Formats 167

UPC-A and UPC-E 167

Code 93 168

Codabar 168

Data Matrix 168

EAN-8 and EAN-13 169

Code 128 169

RSS-14 169

Aztec 169

Code 39 170

ITF 170

RSS Expanded 170

PDF 417 171

QR Code 171

View Form History in TransactField App 172

Using the History folder 172

Viewing Options 172

Multiple Languages in TransactField App 174

TransactField App Security 177

Accessing Security 177

Form Security 177

Configure TransactField App **178**

Client Settings 178

System Settings 180

Disable the Standard Submit or Attachment Button 183

Configure Form Tiles **185**

Extract a Customer Email from Form Data to Display it in a Task List 185

Configure the form XML data extracts in Journey Manager: 185

Configuing the field for TransactField App 185

Synchronize the Device with Manager 186

Extract a Date Field and Format it 186

Customize a Title of a Tile 187

Hide Default Fields 188

Configure the History Screen Tiles Differently to the Task List and Saved Form Tiles 188

Reference 188

Supported Views 188

Template Values 189

Special date value handling 189

Composer User Guide **192**

Getting Started in Composer **193**

Login Page 193

Welcome Page 193

Recent Forms 193

Actions 194

Product Information 195

News 195
Home Menu Choices 195
Workspace 195
Access Control 196
System 196
Top of Page Links 196
Options 196
Feedback 198
Online Documentation 198
Logout 198
Accessibility for Composer Developers 199
Checklist: 199
1. Use sequential page navigation 199
2. Use section headers 200
3. Mark-up lists 200
4. Add alternate text to images 201
5. Do not use background images 202
6. Use fieldsets 202
7. Use caption placement top 203
8. Use described by 204
9. Check color contrast 205
10. Do not use placeholder text 205
11. Do not replace field captions with text 205
12. Do not disable anything 206

The Composer Workspace **207**
The Hierarchy and Environment 207
Workspace Hierarchy 207
Merits of this Structure 208
The Composer Hierarchy 208
Accounts 208
Organizations 208
Projects 210
Creating a Form 211
Composer Form Revisions **212**
Manage Revisions Functionality 212
Accessing the Form Revision Dialog 212
Menu 212
Manage Form Revisions Dialog 212
Form Revision Options 213
Create 214
Make Active 215
Delete 216
Rename 217
View XML 217
Read-Only Option 218
Creating a Revision at Form Publish 219
Form Designer **221**
The Wireframe View 221

Structure and Preview 221
Dragging and Dropping on the wireframe. 222
Widget Palette 222
Overview of Widgets 222
Organization of the Palette 222
Assistants 222
Field Types 222
Favorites 222
How to Use Widgets to Create Fields on Forms 223
Widget Types 223
Simple Widgets 223
Blocks 223
Sections 223
For More on Widgets 223
Structure Panel 223
Standard Mode 224
Showing the relationship between the Structure Panel and the Wireframe 224
Advanced Mode 225
Standard Mode vs Advanced 226
Nuts & Bolts vs Content 226
Composer Framework (Advanced Mode Only) 226
How to Place a New Field into the Structure 226
Drag and Drop 226
Shortcut for Power Users 229

Menu Bar 229
Layout 230
Assistants **231**
Adding Assistants to a Form 231
Assistants 231
Checkboxes 231
Using the Checkbox Assistant 232
The resultant struture 234
Radio Buttons 235
The "Add a Radio Button Group" wizard dialog 236
The structure the "Radio Button Assistant" adds to the form 236
Dynamic Data 237
Sections 237
The Section Assistant wizard 237
Tables 238
Content Blocks 239
Predefined Blocks **240**
249
Tables & Repeats **250**
Static vs Dynamic 251
Editing Table Structure 255
Adding a Footer 256
Repeats 259
Radio Buttons 259

Templates **260**

Overview 260

Out of the Box 261

Creating a New Form 262

Choosing and previewing templates in wireframes withint the New Form wizard. 262

Basic Navigation Types 263

Standard Navigation 263

Accordion Navigation 263

Wizard Navigation 263

strictly in sequence 263

LH Lists 263

menu selection 263

restricted menu selection 264

Combinations of the Above 264

Styling 264

How to Change the Template 264

Right-clicking on the top of the Structure Panel tree 264

Samples of Templates and Styles 265

A basic template, and how choice of color or theme changes the appearance. 266

Same form, different color scheme 267

Same form, but with a more elaborate template. Note the Preview gives a more accurate reprentation of the form's appearance. 268

Edit Properties **270**

Invoking the Edit Properties Dialog 270

Structure 271
Edit Properties dialog for a tab with a Left Hand menu 272
Notation Convention Used 272
The Filter and View Buttons 273
The Properties Tab 273
The Other Tabs 275
Sections Blocks 276
Field Layout Control 277
Problems with explicit position layouts 277
Layout Managers and Hints 277
The Layout Panels 278
Layout Constraints 279
The Effect of the Different Layout Managers 282
Flow — Top to Bottom 282
Flow — Left to Right 283
Multiple fields per line 284
Horizontal Layout 284
Two-column forms 284
Two Column Layout 285
Grid Layout 285
Grid layout with 3 radio buttons respectively 100%, 60mm and 100% 286
Property editor for the parent block 286
Tweaking the element with "Edit Properties -> Layout -> Layout Data" 287
Grid layout where each of 5 columns set to 100% (Note: the caption on the 4th column) 287

Fillers, Springs and Page Breaks 287
Field Styling 289
Using Stylesheets 289
Stylesets 289
Typed Styles 289
Named Styles. 290
Manual Overrides 291
Custom Styling 294
2 sample "HTML Custom Styling" panels 294
A selected text field in preview 294
Field Data 295
Overview 295
Data properties 295
Data Model 295
Data Properties 295
Data Panel 296
Initial Field Value 299
Data Format 299
Data Display Format 299
Data Clearing Policy 299
Values 299
Dropdown Data Panel 300
Dropdown Data Panel (Advanced Mode) 300
Decimal Options Panel 301

The Data Model Tab 302
Scripting 302
Overview of Scripting 302
The Composer Scripting Language 303
The Rules Tab 303
"Rule" Properties 303
"Policies" 303
Editable Policy 303
Editable Clearing Policy 304
Visibility Policy 304
Validation Policy 304
Messages 304
Actions 304
Checkboxes 305
Responsive Rulesets 305
The Rule Editor 306
The Script Editor 307
The Specification Tab 309
Dependencies 310
Overview 310
Inspecting Triggers 311
Inspecting Dependencies 312
Preview & Publishing 313
Renditions 313

Publishing to TM 314

Navigation **315**

The Chrome of Navigation 315

The Simplest Chrome: None 316

Button Navigation 317

Left Hand Menus 318

LH Menu Variants (Unrecommended) 320

Top Menus 321

2-Level Menus 321

Mobile Menu Chrome 322

Navigation Choices 322

Unconstrained vs Constrained 323

Visible vs Hidden Navigation Choices 323

Validation, Errors and Navigation 323

Abandonment 323

Portal Chrome 324

Sections 324

Introduction 324

Maguire Template 324

Adding a Section 325

The Section Assistant wizard. 325

Section Levels 325

Maguire's Use of Sections 325

Maguire Template use of Level 2 Sections 325

Wizards 325
2-Level Navigation 326
MaGuire template top 2-level menu 326
The Navigation Menu in Composer 326
Features common to all templates 326
Maguire Exclusive Features 328
Headers and Footers 329
The Simple Approach 330
The Modern Approach 333
Maguire-style Navigation 334
Footers 336
Recolor and Hide on Print 337
Layout 338
Legacy Layouts 338
Overview 338
Keep it simple 339
Let Composer do the work 340
Use Percentages, not Fixed Width layouts 340
Use Field Styling 340
Checkboxes and Radio Buttons 340
Units and usage tips 340
Caption Top or Caption Left 340
Mixing Checkboxes or Radio Buttons with Textfields on a line 341
Boxy Layout 341

An example of a Boxy layout 342
Layout Issues 344
Measurements 344
Recommendations 345
Devices vs Desktop 345
Delivery to Mobile Devices 346
Two Modes of Delivery 346
Mode 1: The Web Browser 346
Mode 2: Via Native App 347
Which Mode is Superior? 347
Responsive Layout 347
Introduction 348
Composer use of Thresholds 349
Previewing Responsive Behavior 350
Grid Layout Caveats 353
Actions 354
Wrap 354
Wrap and fill 354
Fill 354
Hide 354
Hide (no media support) 354
Show 354
In Line Validation Methods 354
Tool tips 355

Submissions & Attachments **356**

File Uploading 356

Defining Attachments in Composer 356

TM Attachment Table 356

TM Attachment Rule 356

TM Submission Block 357

How this structure renders in preview. 358

Resuling "Submission Attachments" page. 358

Delivery of Attachments 358

Email Acknowledgements 359

Signatures 359

"Signatures are not required at this time. After you have submitted this form, print the receipt, sign it and send it in." 359

Wet signature form after submission. 360

Signature Pad Widget [HTML] 360

The signature canvas. 361

Notes 361

Dynamic Data **362**

Overview 362

What is Dynamic Data? 362

Dynamic Data in Action 362

The Dynamic Data Assistant 363

Standard Block 363

Repeatable Block 363

Table 364
The Dynamic Data Assistant wizard dialog. 365
Dynamic Data Assistant creates the following strutcure and wireframe. 366
Preview of the form with fake data generated by the Dynamic Data Assistant, desktop vs narrow mobile 367
AutoSuggest 368
Overview 369
Reference Data Lookup Block 372
CSV Data With Headings 373
Panels in "Properties -> Data" for the "Type" node (a TM Reference Data Lookup block) 375
CSV Data Without Headings 376
Notes on Auto Suggestion _controller Data Configuration Parameters 376
"Hide Standard Address Block" in VEDA Block Widget 377
Sequence Numbers 378
Internet Explorer 8 Issue 378
Pagination 379
Adding a Page Navigation Block 380
2nd page of the page navigation output (using fake data). 380
TransactField Mobile App 381
About the App 381
The Avoka TransactField App's main user interface 382
How to Publish a Form to the Mobile App 382
Mobile Camera Support 382
The Standard "Take Picture" Button 383

Very simple form with a camera button 384
The form in the Mobile App, and the effect of the "Take Picture" buitton 385
Suppressing the Attach and Submit Tabs 386
Restricting Attachment File Types in the App 387
Multi-channel **388**
Overview 388
Authenticated Users 388
With the Maguire Template 388
With Other Templates 388
The Task List 389
TransactField App 390
Anonymous Users 390
Form Sharing **391**
Overview 391
Minimum Prerequisites for Form Sharing 392
User Authentication Settings 392
Simple Case using Prefabricated Blocks 393
Why Hide the Sharing Block? 399
Sharing in Collaborations 399
Composer Form Parts **400**
Overview 400
Setting Up a Form Parts Project 401
Creating Master Forms and Part-Forms 402
Creating Parts within the Part Forms 403

Combining the Parts to Create a Master Form 406
Opaque and Transparent Parts 408
Parts and Versioning 409
Cross-part rules 410
Simple - All Cross-part Rules in the Master Form 410
Normal - Each Part References Other Parts it Requires. 410
Advanced - Model/View/Controller 412
Usability & Accessibility 415
General Statement 415
FAQs 416
Question: 416
Short Answer: 416
Long answer: 416
Tab Order 416
Privacy 418
General Overview 418
Privacy Concerns with Hidden Fields 418
Clear Hidden Data on Submit 418
Overriding 419
Clear Data when Hidden 419
When to Clear Data 419
Validation 420
Overview 420
Fixed Length 420

Regular Expression 420
Validation Script 421
Icon in Script Editor to access Composer classes listing 421
Validation and Hidden fields 422
Validation Messages 422
Validation settings for a text field 423
Resultant Validation messages on the Desktop 423
In Line Validation 423
Inline Validation Settings 424
Some of the important form-wide HTML Inline Validation options 424
Inline Validation Example 425
Wireframe of form used to demonstrate Inline Validation 425
Desktop Inline Validation Method [HTML] 425
None, Desktop 426
Inline Validation: Desktop, None 426
Per Field Block, Desktop 427
Inline Validation: Desktop, Per Field Block 428
Per Field, Desktop 428
Inline Validation: Desktop, Per Field 429
Per Section, Desktop 429
Inline Validation: Desktop, Per Section 429
Mobile Inline Validation Method [HTML] 430
None, Mobile 430
Mobile: Inline Validation method = None 430

Per Field Block, Mobile 431
Per Field, Mobile 432
Per Section, Mobile 433
Analytics **434**
Overview 434
Composer's Role in Analytics 435
Background Saving 435
The Cancel Button in Maguire Forms 436
The Abandon Widget in non-Maguire Forms 437
Scoring 440
Field Types **443**
Rich Text Fields 443
Rich Text Editor 444
Tag Reference 446
Character tags 447
Radio Buttons 449
Creating Radio Button Groups Manually 449
Mandatory Radio Buttons 449
Calculating or Using the value of a Radio Button Group 450
Dropdown Menus 450
Display Values 450
Values 451
Cascading Dropdowns 452

The Cascading Dropdown Manager predefined block dialog, with the XML pasted into the data text-box. "Save" creates the structure shown. 455
The dialog for creating the "model" level cascading dropdown list. 455
The final structure on the form. 456
Preview of the cascading dropdowns. 456
Prefilled Dropdown Lists 456
Prefilled Dropdown Lists from Repeatable Areas 457
Repeats **458**
Overview 458
Advanced Scripting **459**
JavaScript in Composer 459
Syntax 459
"Me" Instead of "This" 459
Other Departures from Standard JavaScript 459
Classes and Functions 460
Syntax Checking 460
Composer Framework 461
The JavaScript Libraries 461
The Public API 461
Documentation 461
Calls 461
Using JavaScript's Built-in Functions 462
Parameters 462
In the Script Editor 462

Edit Parameter Data 462
Parameter Types 463
Node Reference 464
Obtaining the Fullpath 464
Obtaining the UID 464
Data Types 464
Overview 464
JavaScript Types 464
String 465
Number 465
Boolean 465
Undefined 465
Null 465
Raw Data 465
Conversions 465
Explicit Conversions 465
Implicit Conversion 466
Comparisons 466
Dates 466
Date Formats and Patterns 466
Recommendation for Dates 468
Advanced Date Output Formats 468
Regular Expressions 470
Overview 470

RegEx References 471
How to Use Regular Expressions 471
g 471
i 471
m 471
Using Property Formulae 471
Overview 471
$BUILD 472
$CHAR 472
$DATA 472
$ENCODE 472
$FIELD 473
$FIND 475
$FLOAT 475
$FORM 475
$GENERATION 475
$IF 476
$INFO 476
$LIST 477
$LOCALE 477
$LOOKUP 477
$MATH 477
$RULE 478
$SEQUENCE 478

$STRING 478
$TEST 479
Optimizing Performance **480**
Overview 480
Form Generation 480
Form Rendering 481
Speeding up Generation and Rendering 481
Optimization Strategies 481
Tuning Dependencies 481
Inspecting Dependencies 481
My Rule Dependencies 481
Triggered Rules 482
Dynamic vs Static Dependencies 485
Trigger Conditions 488
Trigger at Initialization 491
Phase Changes 492
Additional Events 492
Caveats 493
Business Rules **494**
GeneralPurpose 494
Error 494
Warnings 494
General Purpose 494
Configuration Settings 494

Visibility Rule 495
Visibility Policy 495
Additional Event LIsteners 495
Business Rule Trigger 495
The Relationship Between Nominated Widgets and the Rule 495
The default 496
Rule fires on change of any observed widget; as a result, the rule could fire many times and make the form laggy 496
Rule fires less and only on a nominated form event or events; results in a more responsive form. 497
Standard Composer Events 497
Error 498
Focus Field Target 498
Clearing Policies 498
Displaying Errors 499
Warning 499
Mandatory Checkbox Block 499
Mandatory Block 499
Library Advanced Features 500
Overview 500
Composer Account Administration 501
How Libraries Relate to Organizations and Projects 501
Inheritance 501
Visibility 501
Ordinary Libraries at the Account Level 501

Compatibility Libraries 502
The Organization's Internal Library 502
Search Path Inclusion at Organization Level 502
Templates 502
Visibility Summary 502
Inheritance 504
The Cascade 504
Search Path Order 505
Custom Types 506
Localization Keys 506
Non-editable 506
Editable 507
Style Packs 511
Inheritance Within Forms 512
Overview 512
Widget Hierarchy 512
Inheritance in blocks 513
Starter Content 513
Resources 513
Custom CSS 514
Custom JavaScript Libraries 514
Templates 515
Overview 515
Stylesheet Modification 515

Location of the Stylesheet Modifications 515
Scope of Stylesheet Modifications 515
Editing the Elected Stylesheets 516
Template Editor 516
Style Options tab 517
Editing the XML Source of the Template 518
Localization 518
Overview 518
Activating Localization 518
Switching Template 519
Activating a Template 519
Setting Target Locales 520
Generating Keys 521
Generating the Properties File 521
Use Source Language Values 521
Use Google Translate Values 521
Use Blank Values 521
Save Non-form Properties to Global Properties File 522
Deployment Options 522
Stylesheets 523
Overview 523
Typed Styles 523
Named Styles 524
Stylesheet Categories 524

Editing Stylesheets 525
Customizing Stylesheets 527
Form Designer Advanced Features **528**
Overview 528
Bulk Editor 528
Property Sets 529
Artifacts Tab 529
Data Model Tab 530
XML Source Tab 531
Localization Tab 531
Attachment Fields Overview **532**
Finding the Attachment Field in the Palette 532
Configuring the Attachment Field 532
Initial Config Options 532
Basic Mode Config Options 533
Advanced Mode Config Options 534
Testing in Preview Mode 535
The User Experience 535
Interacting with the Attachment Field 535
Drag and drop support 536
Read-Only Behavior 536
Exposing Attachments in Review & Approval Style Collaboration Jobs 536
Support Notes 537
Obtain Form Details **538**

Restore the Environment **540**

Composer Account Administration **544**

Aim of this Guide 544

The Workspace **545**

Creating New Organizations **546**

Creating New Projects **548**

Creating & Importing Libraries **549**

Export Options **551**

Inspecting an Organization's Libraries 552

Maintaining Libraries **553**

Sharing Libraries with Accounts 553

Assigning Libraries to organizations from the Workspace 553

Assigning and Maintaining Stylesheets 554

Maintaining Custom Types 557

Maintaining Templates 559

Assigning Libraries 562

Maintaining Property Sets **563**

Publishing & Impact Analysis **565**

Publish Targets 566

Impact Analysis 566

Creating Users **568**

Client Tier Users 568

System Tier Users 569

Creating Roles **570**

Allocating Users **579**

Creating New Accounts **581**

Deployment Properties Setting **582**

Display Important Information on the Composer Login Page 582

Maintaining Help, News, and Tips **584**

Organization Management **587**

Release Management **589**

Viewing the Logs **591**

Managed Datapacks **593**

Background and strategic fit 593

Features of using Managed Datapacks 593

User interaction and design 593

Managed Support for BizPacks **595**

Initial Allocation of Bizpack Support 595

Importing the Bundled Packs 595

Gaining Access to the BizPack Libraries 595

Removing Access to the BizPack Datapacks 596

Deprecated Widgets 596

The Composer Palette 596

Managed Support for Maguire **598**

Composer Ant Support v4.5.0 ..... **599**

Requirements 599

Setting up your Ant build file 599

Composer Ant Tasks 600

sfc-export-account 600
Task Parameters 600
Example 601
sfc-export-org 601
Task Parameters 601
Example 601
sfc-form-spec 601
Task Parameters 601
Examples 602
sfc-import 603
Task Parameters 603
Example 603
sfc-publish 603
Task Parameters 603
Example 604
sfc-server 604
Task Parameters 604
Example 605
Composer On-premise Upgrade Guide (for major release) **606**
Upgrade Steps 607
Follow up Actions 608
1) Updating the Service Pack(s) (Not required for major releases) 608
2) Adding Maguire Packs 608
3) Activating Bizpacks 609

4) Setting the current release 610
5) Clients not on the latest release 611
Rollback 612
Upgrading from an older version 612
Steps to upgrade 612
Composer On-premise Upgrade Guide (service pack release only) 613
1) Updating the Service Pack(s) 613
2) Adding Maguire Packs 613
3) Setting the current release 614
Composer Attachment Definitions 616
Transaction Manager 616
Pros 616
Composer Form 616
TM Attachment Rule 616
Pros 617
Cons 617
TM Attachment Table 617
Composer Attachment Restrictions 618
Transaction Manager - Document Types 618
Composer Attachment Time 619
Composer Attachments in Repeats 620
Transact for Salesforce App 621
User Interface Design 622
Styling 622

Navigation 622
Information Architecture 623
Integration Approach 624
UI 624
APIs 624
Authentication / Identity Management 624
Data Security 625
Data Access Controls 625
Off-Platform Data Storage 625
Salesforce Organization Impacts 626
Licensing 626
Data Storage Requirements 626
API Usage Volumes 626
Batch Processing 627
Reporting 627
Integrate Salesforce with Manager 628
Install the Transact for Salesforce App 628
Create the API User Account in Salesforce 628
Create the Manager Connected App in Salesforce 629
Configure Permitted Users for the Connected App 630
Create the API Named Credential in Salesforce 630
Information required by the Journey Manager Team 631
Enable Access to the Transact for Salesforce App 633
Manage User Licenses 634

Retrieve the Consumer Key and Secret for the Salesforce App **635**

Retrieve the Security Token for the Salesforce User Account **636**

Remove Password Expiry on a Salesforce User Account **637**

Creating a 'Password Never Expires' Permission Set 637

Adding the 'Password Never Expires' Permission Set to a User Account 637

Configure the Transact for Salesforce App UI in Salesforce **638**

Create the Visualforce Page 638

Visualforce Markup Attributes 640

Add the Visualforce page to the Page Layout 642

Configure Transact for Salesforce App **643**

Create the API User Account in Manager 643

Configure the Salesforce API Client Details 644

Import the Manager Extension Packages 644

Configure the Salesforce User Space 644

Configure the Security Manager 645

Configure the Delivery Service and Delivery Channel 645

Make Forms Available for Salesforce Users **647**

Group Forms Using Form Categories **648**

Configure Salesforce Delivery Channel **649**

Completed Transaction Delivery 649

Abandoned Transaction Delivery 649

Saved Transaction Delivery 649

Configure a Prefill Generator in Salesforce **650**

Create the Prefill Generator Record 650

Managing Form Versions 650
JSON Configuration Format 651
Supported Parameters 652
Object Type Definition 652
Dealing with Complex Types 653
The includeFields Directive (Required) 653
The includeRelated Directive (Optional) 653
The includeCustom Directive (Optional) 654
XML Prefill Data Generation 654
Custom Prefill Generators 657
Populate a Form with Salesforce Prefill Data 658
Configure a Delivery Processor in Salesforce 659
Create the Delivery Processor Record 659
JSON Configuration Format 660
Object Type Definition 662
The matchFields Directive (Optional) 663
The insertFields Directive (Required) 664
The updateFields Directive (Required) 664
The sourceFields Directive (Required) 664
The relations Directive (Optional) 665
XML Consumption 665
Custom Delivery Processors 666
Restrict Access for Selected Forms to User Groups 668
Create the Form Group 668

Add Users to the Form Group 668

Assign Forms to the Form Group 668

Enable Access to Restricted Forms 669

Receive Administrator Alerts when Delivery Processing Fails 670

Transact for Salesforce App Troubleshooting 671

Content cannot be displayed: You do not have sufficient privileges to access the page: /apex/Avoka_Transact_Lead 671

Problem 671

Solution 671

Reference 671

Could not identify SObject type <X> 671

Problem 671

Solution 672

Delivery Error: No such column 'avoka__Delivery_Trigger_Type__c' on sobject of type avoka__Transact_Delivery__c 672

Problem 672

Solution 673

Delivery Error: The requested resource does not exist 674

Problem 674

Solution 674

Delivery Processing Error: Field <X> on <Y> is not CREATEABLE by <Z> 674

Problem 674

Solution 675

Insufficient Privileges: You do not have the level of access necessary to perform the operation you requested... 675

Problem 675
Solution 675
Missing "Reset My Security Token" Option 676
Problem 676
Solution 676
Post Request: 403 Forbidden - You don't have permission to access /manager/secure/GroovyServiceServlet on this server 676
Problem 676
Solution 676
Salesforce API Authentication Failure (Invalid Grant) 676
Problem 676
Solution 676
Sorry, you are not licensed to use the Transact solution 677
Problem 677
Solution 677
Transact for Salesforce App Installation 678
Service Definition Import Options 679
Security Manager Import Options 679
Installation Assets Version 1.4 681
New Features 681
Issues Resolved 681
Installation Assets 681
Installation Assets Version 1.5 682
New Features 682

Issues Resolved 682
Installation Assets 682
Installation Assets Version 1.6 ..... **684**
New Features 684
Issues Resolved 684
Installation Assets 684
Installation Assets Version 1.8 ..... **685**
New Features 685
Issues Resolved 685
Installation Assets 685
Installation Assets Version 1.10 ..... **686**
New Features 686
Issues Resolved 686
Installation Assets 686
Installation Assets Version 1.11 ..... **688**
New Features 688
Issues Resolved 688
Installation Assets 688
Transact Integration Gateway Overview **689**
Overview 689
Usage and Deployment Models 689
Delivery Integration 690
Service API Gateway 691
Gateway Security 692

Reference Data Integration 692
Transact Integration Gateway Setup **694**
Prerequisites 694
Transact Client SDK 694
Installation 694
Configuration 695
Delivery Config 696
Proxy Config 698
Services Config 700
Gateway Services Config 702
Scheduler Config 703
Storage Config 704
Transact Integration Gateway Development **706**
Groovy Development 706
Security Restrictions 706
Script Development 706
Delivery Service Script 707
Gateway Service Script 710
Scheduled Service Script 711
Adding JAR Libraries 712
Logging 713
Transact Manager Previous Releases **715**
Transact Fluent SDK Overview **716**
Benefits of Using Groovy in Journey Manager **718**

Fluent Groovy Services vs Groovy Services **722**

Fluent Groovy Service 722

Groovy Service 724

Configuration Example 725

Fluent API REST Service **727**

Requests 727

Responses 729

Security 729

Class Names 729

About Journey Platform New Releases **734**

Major Release 734

Maintenance Release 734

Release Notes Sitemap 734

What's New 734

What's New in 21.11 ..... **736**

Journey Maestro New Features **737**

Journey Manager New Features **738**

Journey Analytics New Features **739**

Journey Workspaces New Features **740**

What's New in 21.05 ..... **741**

Journey Maestro New Features **742**

Journey Manager New Features **743**

What's New in 20.11 ..... **744**

Journey Maestro New Features **745**

Journey Manager New Features **746**

SpringboardSmall Business (SMB) New Features **747**

What's New in 20.05 ..... **748**

Journey Maestro New Features **749**

Journey Manager v20.05 New Features **750**

Journey Analytics New Features **751**

Journey Workspaces New Features **752**

Exchange New Features **753**

Springboard v3.0.0 New Features **754**

What's New in 19.11 ..... **755**

Journey Maestro New Features **756**

Journey Manager New Features **757**

Journey Analytics New Features **758**

Journey Workspaces New Features **759**

Exchange v19.11 New Features **760**

What's New in 19.05 ..... **761**

Journey Maestro New Features **762**

Journey Manager New Features **763**

Journey Analytics New Features **764**

Journey Workspaces New Features **765**

Exchange New Features **766**

Journey Manager Features **767**

Main Features 767

Features by Release 767

22.04 768
21.11 768
21.05 768
20.11 769
20.05 769
19.11 770
19.05 771
18.11 771
18.05 773
17.10 773
5.1 774
5.0 776
4.3 776
Maestro Features **778**
Main Features 778
Features by Release 778
22.04 778
21.11 779
21.05 779
20.11 780
20.05 780
19.11 780
19.05 781
18.11 781

Journey Analytics Features **782**

Main Features 782

Features by Release 782

21.11 782

21.05 783

20.05 784

19.11 784

19.05 784

18.11 785

Temenos Journey Platform Features Compatibility **786**

Manager 786

Maestro 786

Composer 787

Product Suite Common Vulnerabilities and Exposures **788**

Journey Platform for Developers **789**

Journey Maestro Design Environment 789

Open UX Environment 790

Integration 791

Native Integration 791

3rd Party Integration 791

Software Development Technologies **792**

AngularJS 792

JavaScript 792

LESS 792

HTML5 793
JSON 793
CSS 793
CSV 793
XML 793
Journey Platform Integration 795
Integration Areas 795
Transactions Discovery 795
Data Pre-population 796
Data Entry Streamlining 796
Execution 797
Save and Resume 797
Delivery 798
Authenticated and Unauthenticated Transactions 798
Extension Points 799
Calling External Services and API 803
Web Site Integration and Single Signon 804
Form Level Integration 805
Five Journey Platform Integration Areas 806
Delivery 806
Form Discovery 809
3 Pre-Population - Accelerating Data Entry 810
Streamlining Data Entry 810
Execution 811

The Economics of Integration 813
Why do we do integration? 813
Assess the Value of Integration 814
Consider Deferring Integration 814
Consider Leveraging Existing Paper/image-based Processes 814
Hidden Costs of Process Re-engineering 815
Google Analytics Virtual Page Integration 816
Add GA Code to Head of Page 816
Create Business Rule to call the ga method on page change 816
Continuous Integration Overview 820
Continuous Integration Workflow 822
CI Version Control Systems 823
CI Servers 824
CI Build Systems 825
CI Dependency Management Systems 827
CI Application Packages 828
Main JSON 829
app-package-def.json 829
Delivery Channels JSON 829
delivery/rest-service-delivery-def.json 829
Forms JSON 830
forms/CCA-MAESTRO-AP/cca-maestro-form-def.json 830
forms/CCA-MAESTRO-AP/locale-de.json 834
Properties 834

properties/property-def.json 834
Service Connections 835
connections/connection-def.json 835
Services 836
services/helloworld/service-def.json 836
services/helloworld/HelloWorld.groovy 837
Support **838**
support HANDBOOK 839
SUPPORT PORTAL 839
POLICIES 839
Resources 840
SYSTEM REQUIREMENTS 840
LEGACY SYSTEM REQUIREMENTS 840
Journey Platform Support Handbook **841**
Roles and responsibilities **842**
Support definitions **843**
Platform Support 843
Definition 843
Example of Platform Issues 843
Out of Scope 843
Development Assistance 843
New Feature Requests 843
Severity level definitions **844**
Severity 1 844

Severity 2 844
Severity 3 844
Response times and update targets **845**
Response Times 845
Update Targets 845
Business Hours Definition 845
After Hours Support 845
Contacting Support **846**
Support Centers 846
Business Hours 846
After Hours Support 846
Temenos Support Portal 846
Accessing the Temenos Support Portal 846
Username and Password 847
Open a new ticket 847
Describe your issue 848
Summary 848
Issue Description 848
View existing tickets 848
Ticket statuses explained 849
Lifecycle of a support ticket **850**
Step 1 850
Open Support Ticket 850
Support Ticket Severity 850

Step 2 850
Engagement with Support Team 850
Step 3 850
Close a Support Ticket 850
Issue Resolution 851
Step 4 851
Customer Satisfaction Survey 851
Step 5 851
Reopening a support ticket 851
Escalating a support ticket **852**
Escalating support cases 852
How do I escalate the visibility of a support ticket to management? 852
What happens after an escalation request is made? 852
Time-Based Escalation 852
Product release definitions **854**
Release terminology 854
Major release 854
Maintenance release 854
End-of-Life Policy 854
Getting more information **855**
Documentation website 855
Resources website 855
Announcements 856
Release notes 856

Q&A 856
Ideas and New Feature Requests 857
Journey Platform End-of-Life Policy 858
Feature and Maintenance Releases 858
Latest Release Notes 859
Feature Lists 859
End-of-Life Schedules 859
Journey Manager 859
Supported versions 859
Unsupported (EOL) versions 859
Journey Maestro 860
Supported versions 860
Unsupported (EOL) versions 860
Strategy for Continued Journey Maestro Support after AngularJS End-of-Life from 31 December 2021 861
Transact Composer 861
Unsupported (EOL) versions 861
TransactField App 862
Unsupported (EOL) versions 862
Journey Platform Deprecation Schedule 863
Reset Support Portal Password 865
Changes to Our Product Names 867
Name Changes from the original Knowledge Base 867
Journey Platform System Requirements 869
Legacy 869

Journey Manager System Requirements **870**

Cloud 870

On Premise 870

Browser 870

Operating Systems 871

Databases 871

Hardware 872

Journey Maestro System Requirements **873**

Cloud 873

Browser 873

Recommended Hardware 873

Journey Platform Forms System Requirements **874**

Desktop Browsers 874

Legacy Desktop Browsers 874

Mobile Browsers 875

Mobile Browsers Requirements **876**

Transact Composer Requirements **877**

Cloud 877

Adobe Flash 877

Browser 877

Recommended Hardware 878

Transact Integration Gateway System Requirements **879**

Certified Operating Systems 879

Recommended Hardware 879

TransactField App System Requirements **880**

Legacy Browsers Requirements **881**

Journey Platform Persona Overview **882**

Implementation Team 882

Business as Usual (BAU) Team 883

Maestro Form Builder **885**

Example tasks of a Form Builder include: 885

Prerequisite Skillset of a Form Builder 886

Objectives and Goals of a Form Builder 886

Journey Solution Architect **887**

Example tasks of a Journey Solution Architect include: 887

Prerequisite Skillset of a Solution Architect 888

Objectives and Goals of a Solution Architect 888

Journey Platform Developer **889**

Example tasks of a Platform Developer include: 889

Prerequisite Skillset of a Platform Developer 890

Objectives and Goals of a Platform Developer 890

Maestro Template Designer **891**

Example tasks of a Template Designer include: 891

Prerequisite Skillset of a Template Designer 892

Objectives and Goals of a Template Designer 892

Journey Application Tester **893**

Example tasks of a JourneyApplication Tester include: 893

Prerequisite Skillset of a Journey Application Tester 893

Objectives and Goals of a Journey Application Tester 894

Journey Analytics User 895

Prerequisite Skillset of Journey Analytics 895

Objectives and Goals of Journey Analytics 895

Journey Implementation/Project Manager 896

Example tasks of a Journey Implementation/Project Manager include: 896

Prerequisite Skillset of a Journey Implementation/Project Manager 896

Objectives and Goals of a Journey Implementation/Project Manager 897

Journey Champion/System Owner 898

Prerequisite Skillset of a Journey Champion/System Owner 898

Objectives and Goals of a Journey Champion/System Owner 898

Journey System Administrator 899

Example tasks of a Journey System Administrator include: 899

Prerequisite Skillset of a Journey System Administrator 899

Objectives and Goals of a Journey System Administrator 899

Journey System Manager/DevOps 900

Example tasks of a Journey System Manager/DevOps include: 900

Prerequisite Skillset of a System Manager/DevOps 900

Objectives and Goals of a System Manager/DevOps 900

Journey Support Team 901

Example tasks of a Journey Support Team include: 901

Prerequisite Skillset of the Support Team 901

Objectives and Goals of a Support Team 901

Instructor-Led Training and eLearning overview 902

Instructor-Led Training ILT **903**

eLearning **904**

Available eLearning Courses 904

eLearning Exercises **905**

Version 5.1 905

Version 17.10 905

Help **906**

Learning Center Help Guide **907**

New consolidated Journey Platform resources website 907

Access the online courses at the new website - journey.temenos.com 907

Access the online courses at the original location - learning.avoka.com 907

Navigate through the Temenos Learning Center 910

Home page and menu bar 910

Status icon 910

Course name 910

Launch video 910

eLearning Exercises for Maestro v17.10 ..... **912**

Pre-requisite 912

912

eLearning Exercises for Maestro v17.10 912

Whats new? 912

Help 913

Exercise Setup - Importing the Maestro v17.10 Exercise Files **914**

Step 1: Download the Exercise Project 914

Step 2: Navigate to the Maestro Organization 914
Step 3: Import Project 914
Step 4: Set the Default Initial Version and Default Template 915
Assign an Entity to Multiple Items Exercise v17.10 916
Step 1: Add a new Section and Components 916
Step 2: Assign an Entity to multiple items 916
Extra Help 916
Create an Entity in the ID Field Exercise v17.10 917
Step 1: Create a Form 917
Step 2: Create an Entity 917
Step 3: Assign an Entity 917
Step 4: Create a nested Entity 918
Extra Help 918
Create an Entity in the Entity Panel Exercise v17.10 919
Step 1: Create an Entity 919
Step 2: Create a nested Entity 919
Extra Help 919
Delete an Entity Exercise v17.10 920
Step 1: Delete an Entity 920
Extra Help 920
Duplicate IDs Exercise v17.10 921
Step 1: Add a component with a duplicate ID and Review 921
Step 2: Add a component with a duplicate ID and Accept 921
Extra Help 922

ID Management Exercise v17.10 ..... **923**

Step 1: Create a new form 923

Step 2: Add a new Text Field component 923

Step 3: Update the Text Field component 923

Extra Help 924

Impact Analysis Exercise v17.10 ..... **925**

Step 1: Open the Order form 925

Step 2: Review the current references 925

Step 3: Add a new Text Field that references First Name 925

Step 4: Review the updated references 925

Extra Help 925

Semantic Versioning Exercise v17.10 ..... **926**

Step 1: Update the 1.0-develop version 926

Step 2: Create a Tagged version (based on 1.0-develop) 926

Step 3: Open the Tagged version 926

Step 4: Update the 1.0-develop version 926

Step 5: Create another Tagged version (based on 1.0-develop) 927

Extra Help 927

Upgrade a Project Exercise v17.10 ..... **928**

Step 1: Import a Legacy project 928

Step 2: Upgrade the project 928

Extra Help 928

eLearning Exercises for Maestro v5.1 ..... **929**

Pre-requisite 929

929

eLearning Exercises for Maestro v5.1 929

Help 930

Exercise Setup - Importing the Maestro v5.1 Exercise Files 931

Step 1: Identify the Training Environment 931

931

Step 2: Download the Exercise Project 931

Step 3 Import the Exercise Project 931

Result 932

Optional Step 4 Importing Legacy Projects 932

Step 5 Update Release Version 933

Extra Help 934

Importing a Project 934

Calculation Rules Exercise v5.1 937

937

Step 1 Create a New Form 937

Step 2 Add Components to Page One 937

Step 3 Populate a Component Based on Entered Data 938

Step 4 Add Show if Rules 939

Step 5 Change the Clear Hidden Data Option 939

Step 6 Add an Order Details Section to the Form 939

Step 7 Add a Calculation Rule to the Item Total 940

Step 8: Add a Calculation Rule to the Sub-Total Component 940

Step 9: Add a Calculation Rule to the Total Due Component 941

Extra Help 941
Create a Show If Rule 941
Combine entered data into a component 943
Clear Hidden Data Immediately 943
Create a Mathematical Calculation Rule 944
Create a Calculation Rule for Repeated Items 944
Compatibility Mode Exercise v5.1 946
Step 1: Import Compatibility Mode Project 946
Step 2: Update the Project to Maestro 5.1 947
Extra Help 947
Import project 947
Change Release Version 948
Create a New Form Exercise v5.1 950
Step 1: Create a New Form 950
Step 2: Add Components to Page One 950
Step 3: Add Components to Page Two 952
Step 4: Save the Form 953
Extra Help 953
Component Properties 953
Radio Button and Dropdown Options 955
Arrangement of Radio Button Options 956
Creating Rules Exercise v5.1 958
Step 1 Create a New Form 958
Step 2 Add Components to Page One 958

Step 3 Add a Rule to Billing Address 959
Step 4 Add a Rule to the Checkbox 960
Step 5 Add Components with a Mandatory If Rule 960
Extra Help 961
Create an Editable If Rule 961
Create a Change Rule 962
Create a Mandatory If Rule 963
Creating Shared Components Exercise v5.1 965
Step 1 Open Create Component Begin 965
Step 2 Setup a Repeating Block and Add Components 965
Step 3 Create Visibility Rule 966
Step 4 Wrap in block 966
Step 5 Create a Shared Component 968
Step 6 Edit a Shared Component 969
Step 7 Publish Updated Component 970
Extra Help 971
Using a Repeating Block Template 971
Create a Show if Rule 972
Wrap Components in a Block 974
Create Shared Component 976
Layout Exercise v5.1 979
Step 1: Open the Existing Form Layout Begin. 979
Step 2: Adjust the Layout of Title, First Name and Last Name 979
Step 3: Adjust the Employment Details Components 980

Step 4: Adjust the Living Expenses Components 980
Step 5: Add a New Component to the Living Expenses Section 981
Extra Help 981
Change the layout of components 981
Change the position of components (using Offset) 982
Add a Currency Field 983
Pages and Sections Exercise v5.1 985
Step 1: Create a New Version 985
Step 2: Update the Names of the Existing Pages and Sections 985
Step 3: Add a new section to the Applicant Details page 986
Step 4: Add a New Page with a Section 986
Step 5: Rearrange the Components 987
Step 6: Build the TM Form Version 988
Extra Help 988
Add a new page 988
Add a new section 989
Additional Text 990
Rearrange Components 990
Repeating and Shared Components Exercise v5.1 993
Step 1 Open the Existing Form 'Repeating and Shared Components Begin' 993
Step 2 Add a New Page with Sections to Your Form 993
Step 3 Add Components to the Page 994
Step 4 Add a Shared Component 995
Extra Help 996

Add a new Page and Section 996

Using a Repeating Block Template 996

Using a Shared Component 998

Rules without Scripting Exercise v5.1 1000

Step 1 Open the existing form Rules without Scripting Begin. 1000

Step 2 Create a Rule for Income Details 1000

Step 3 Calculate the Total of the Four Expenses in the Expense Details Section 1000

Step 4 Add the Driver License Components to the Your Details Section. 1001

Step 5 Create a Rule for Driver License 1001

Extra Help 1002

Create a Show If Rule 1002

Create a Script Rule 1004

Wrap Components in a Block 1005

Two Level Navigation Exercise v5.1 1008

Step 1 Open Two-Level Navigation Begin 1008

Step 2 Add Page Groups to the Form 1008

Step 3: Rename Pages 1009

Step 4 Set Second Level Navigator to Equal Width 1009

Extra Help 1009

Using a Page Group 1010

Set the Second Level Navigator to equal widths 1010

Validation Rules Exercise v5.1 1013

Step 1 Open Validation Rules Begin 1013

Step 2 Create a Valid If Rule 1013

Step 3 Create a Regular Expression Validation Rule 1014

Extra Help 1015

Create a Valid if rule 1015

Create a Regular Expression rule 1016

eLearning Exercises for Maestro v5.0 1018

Pre-requisite 1018

1018

eLearning Exercises for Maestro v5.0 1018

Help 1018

Exercise Setup - Importing the Maestro v5.0 Exercise Files 1020

Step 1 Identify the Training Environment 1020

Step 2 Download the Exercise Project 1020

Step 3 Import the Exercise Project 1020

Result 1021

Optional Step 4 Importing Legacy Projects 1021

Calculation Rules Exercise v5.0 1022

1022

Step 1 Create a New Form 1022

Step 2 Add Components to Page One 1022

Step 3 Populate a Component Based on Entered Data 1023

Step 4 Add Show if Rules 1024

Step 5 Change the Clear Hidden Data Option 1024

Step 6 Add an Order Details Section to the Form 1024

Step 7 Add a Calculation Rule to the Item Total 1025

Step 8: Add a Calculation Rule to the Sub-Total Component 1025
Step 9: Add a Calculation Rule to the Total Due Component 1026
Extra Help 1026
Create a Show If Rule 1026
Combine entered data into a component 1028
Clear Hidden Data Immediately 1028
Create a Mathematical Calculation Rule 1029
Create a Calculation Rule for Repeated Items 1029
Create a New Form Exercise v5.0 1031
Step 1 Create a New Form 1031
Step 2 Add Components to Page One 1031
Step 3 Add Components to Page Two 1033
Step 4 Save the Form 1034
Extra Help 1034
Component Properties 1034
Radio Button and Dropdown Options 1036
Arrangement of Radio Button Options 1037
Creating Rules Exercise v5.0 1039
Step 1 Create a New Form 1039
Step 2 Add Components to Page One 1039
Step 3 Add a Rule to Billing Address 1040
Step 4 Add a Rule to the Checkbox 1041
Step 5 Add Components with a Mandatory If Rule 1041
Extra Help 1042

Create an Editable If Rule 1042
Create a Change Rule 1043
Create a Mandatory If Rule 1044
Creating Shared Components Exercise v5.0 ..... **1046**
Step 1 Open Create Component Begin 1046
Step 2 Setup a Repeating Block and Add Components 1046
Step 3 Create Visibility Rule 1047
Step 4 Wrap in block 1047
Step 5 Create a Shared Component 1049
Step 6 Edit a Shared Component 1050
Step 7 Publish Updated Component 1051
Extra Help 1052
Using a Repeating Block Template 1052
Create a Show if Rule 1053
Wrap Components in a Block 1055
Create Shared Component 1057
Layout Exercise v5.0 ..... **1060**
Step 1: Open the Existing Form Layout Begin. 1060
Step 2: Adjust the Layout of Title, First Name and Last Name 1060
Step 3: Adjust the Income Details Components 1061
Step 4: Adjust the Expenses Details Components 1061
Step 5: Add a New Component to the Expense Details Section 1062
Extra Help 1062
Change the layout of components 1062

Change the position of components (using Offset) 1063

Add a Currency Field 1064

Pages and Sections Exercise v5.0 1065

Step 1 Create a New Version 1065

Step 2 Update the Names of the Existing Pages and Sections 1065

Step 3 Add a new section to the Applicant Details page 1066

Step 4 Add a New Page with Two Sections 1066

Step 5 Rearrange the Components 1067

Step 6 Publish the Form 1069

Extra Help 1069

Add a new page 1069

Add a new section 1070

Additional Text 1071

Rearrange Components 1071

Repeating and Shared Components Exercise v5.0 1074

Step 1 Open the Existing Form 'Repeating and Shared Components Begin' 1074

Step 2 Add a New Page with Sections to Your Form 1074

Step 3 Add Components to the Page 1075

Step 4 Add a Shared Component 1076

Extra Help 1077

Add a new Page and Section 1077

Using a Repeating Block Template 1077

Using a Shared Component 1079

Rules without Scripting Exercise v5.0 1081

Step 1 Open the existing form Rules without Scripting Begin. 1081
Step 2 Create a Rule for Income Details 1081
Step 3 Calculate the Total of the Four Expenses in the Expense Details Section 1081
Step 4 Add the Driver License Components to the Your Details Section. 1082
Step 5 Create a Rule for Driver License 1082
Extra Help 1083
Create a Show If Rule 1083
Create a Script Rule 1085
Wrap Components in a Block 1086
Two Level Navigation Exercise v5.0 1089
Step 1 Open Two-Level Navigation Begin 1089
Step 2 Add Page Groups to the Form 1089
Step 3: Rename Pages 1090
Step 4 Set Second Level Navigator to Equal Width 1090
Extra Help 1090
Using a Page Group 1091
Set the Second Level Navigator to equal widths 1091
Validation Rules Exercise v5.0 1094
Step 1 Open Validation Rules Begin 1094
Step 2 Create a Valid If Rule 1094
Step 3 Create a Regular Expression Validation Rule 1095
Extra Help 1096
Create a Valid if rule 1096
Create a Regular Expression rule 1097

Exercise_AssignEntityMultipleItems 1099

Exercise_CreateEntity_EntityPanel 1101

Exercise_CreateEntity_IDfield 1103

Exercise_DeleteEntity 1106

Exercise_DuplicateIDs 1107

Exercise_IDManagement 1109

Exercise_ImpactAnalysis 1110

Exercise_SemanticVersioning 1112

Exercise_UpgradeProject 1115

Exercise - Assign an Entity to multiple items 1116

Exercise - Create an Entity (Entity panel)_2 1118

Exercise - Create an Entity (ID field) 1120

Exercise - Delete an Entity 1123

Exercise - Duplicate IDs_3 1124

Exercise - ID Management_2 1126

Exercise - Impact Analysis_2 1127

Exercise - Semantic Versioning 1129

Exercise - Upgrade a project_2 1132

Glossary 1133

Welcome to the Temenos Journey Platform Documentation!

Journey Platform | All Personas | 5.1 & Higher

We are here to help you start using our documentation website in the most efficient way so you can quickly and easily find the wealth of information on all our Temenos Journey platform products. We have written our documentation in an intuitive way, but it's worth a few minutes of your time to become familiar with the documentation layout and important features of the website, such as:

- Search
- TOC
- Quick Menu
- Navigation Bar

Topics

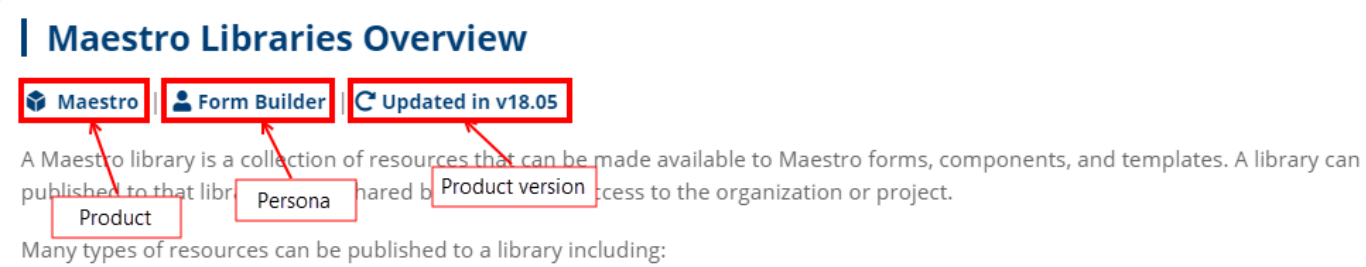
We use **topics**[1] to write and organize our documentation. Each topic is a self-contained article describing a particular concept or a procedure that can stand on its own and give enough information to a reader as well as provide links to another related topic to provide a learning path. A topic is designed to minimize repetition across pages and ensure there's a single source of truth for every concept and feature.

Badges

Every topic begins with a title that lets you know what you're looking at. Links between topics let you jump straight to related content. Most topics include a row of badges, located above the content, that provide information about the topic, including:

- Product the topic relates to. Usually a topic describes one product, but sometimes it can cover several, for example, Manager and Maestro.
- Persona the topic is relevant for. It helps identify reading audience and create various learning paths.
- Product version the topic is compatible with and whether a feature described in this topic was created, updated or removed.

For example, the topic below - the Maestro Libraries Overview - relates to the Maestro product, is relevant for Form Builders, and has been updated with content for Maestro v18.05.



[1]Topic-based authoring is a modular approach to content creation where content is structured around topics that can be mixed and reused in different contexts.

We continuously add, update and enhance its content, so there is more for you to read and search. To make it easier for you to navigate our content, we have tailored it to certain personas related to how you use the Journey platform. Read understanding personas to find out what your Journey persona is.

Version agnostic content

This documentation covers all Journey platform products including their different versions. We have written it in such a way to make it version agnostic, meaning all users have access to the same content regardless of the product version they use. When there is a product or feature difference between versions, it is noted within the documentation. This allows the number of versions of each topic to be minimized, which reduces duplication of content that doesn't change from one version to the next, and makes it easier for you to find what you need. The topic layout is designed to make it clear which product versions the topic relates to. Each topic has a version badge identifying the product versions in which the topic was most recently updated. For example, the **Maestro Libraries Overview** topic above was updated for Maestro v18.05.

Our documentation is intended to describe the most recent version of the Journey platform. Any version-specific content prior to the most recent version will be identified either at the topic level, using badges as described above, or inline.

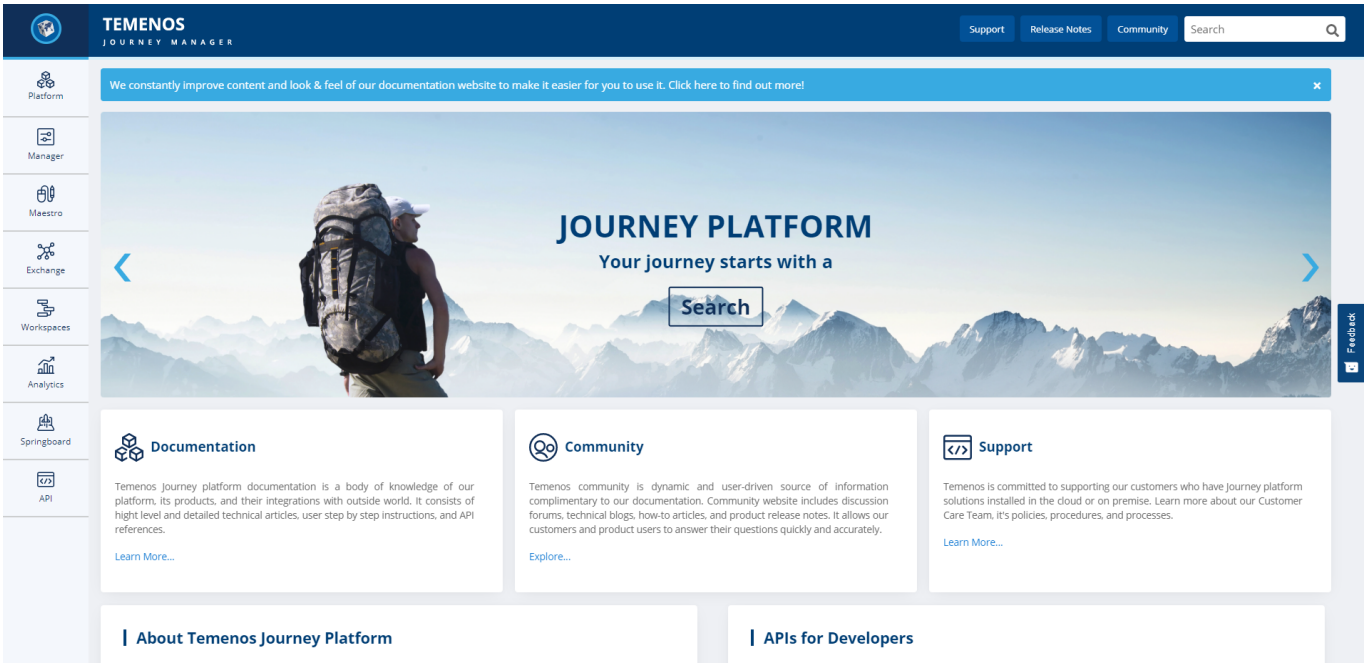
Copy URL button

Each topic has the **Copy Url** button  that allows you to copy the topic's Url, so you can share the page via social networks or email.

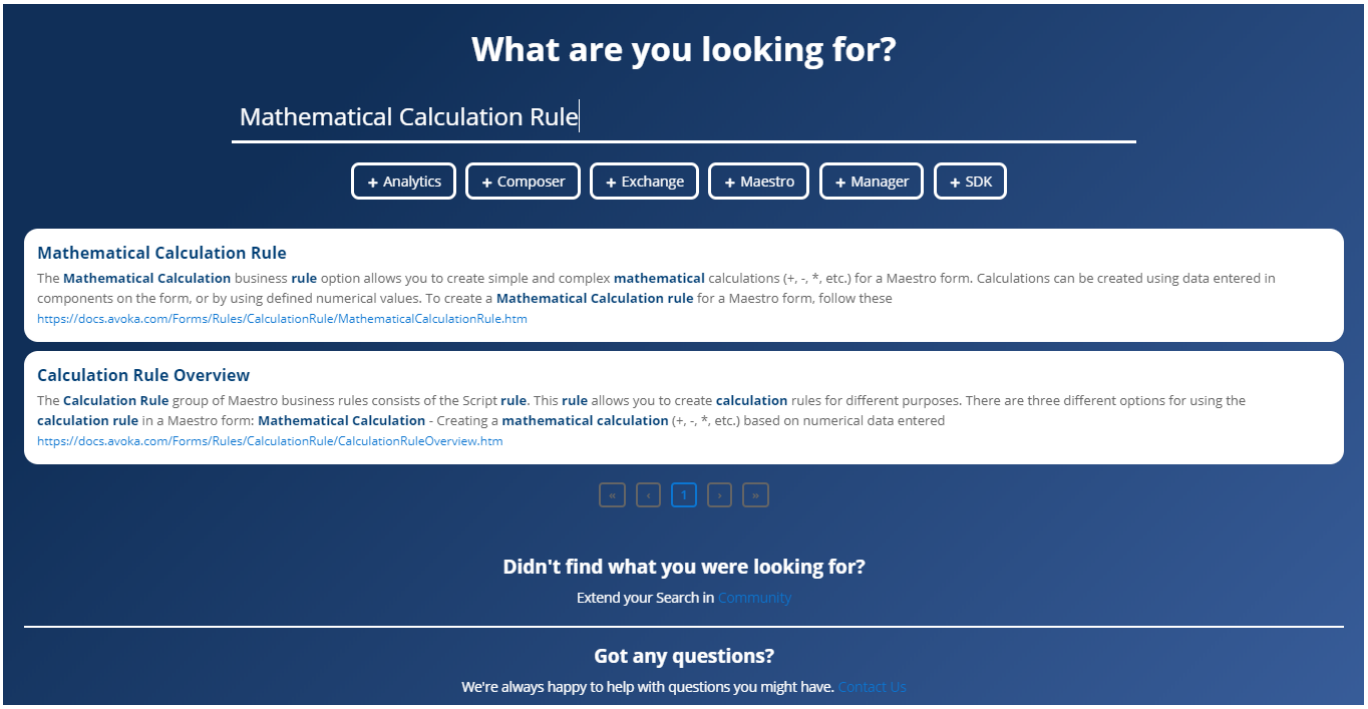
Documentation Navigation Options

Search

We believe that searching our documentation is the fastest and most flexible way to navigate its content to find what you need. When you open our home page, the search is right there and ready to go.



Click in the **Search** button and start typing what you're looking for, for example, maestro overview. As you type, the search will list the matching results, so you can stop when you see the match and click the link to read a topic.



Each search match displays a topic title, a paragraph where the search keyword or phrase is found, and the URL of the topic.

It's common that the search returns many pages of results, so you need to scroll through the page to look for the exact match and use the pagination at the bottom of the search window to go to the next page. For example, click **2** to go to the second page of the search results.

Search filters

The search output can be overwhelming at times, especially if you're not sure what you're looking for. In this case, you can narrow the search result by selecting one or several filters from the **Filters** box. Filters represent our products. For example, you should click the **Maestro** filter to show only articles related to Journey Maestro product.

Every page of our documentation has a search bar in the top right corner, so you are never far away from it. Type your search query in the box or select one from the previous searches and press **Enter.**

If you can't find what you're looking for in our documentation, click **Resources** (top-right corner) to continue searching there. The resources search works similarly to the documentation search, and you can filter its results as well.

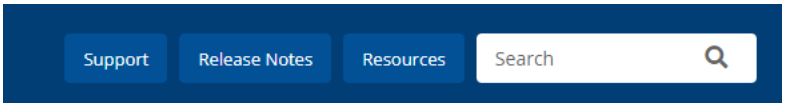
Google Search

You can also search our Journey Platform documentation straight from Google! Google indexes the docs.avoka.com website every now and then, so all you need to do is to run a query like this **site:docs.avoka.com "whatever you are looking for"** from your browser. You can do the same for the resources website.

Navigation Bar

There are a number of ways that you can find the information you need - you can use the documentation search, view one of the persona or feature user paths, or browse the table of contents.

At the top of the page, the **Navigation Bar** provides quick access to the most relevant topics of the selected part of the documentation. The Navigation Bar is accessible from any topic you're reading, so you can use it as a shortcut to another part of the documentation. For example, click **Overview** to open the **Temenos Journey Platform Overview** topic.

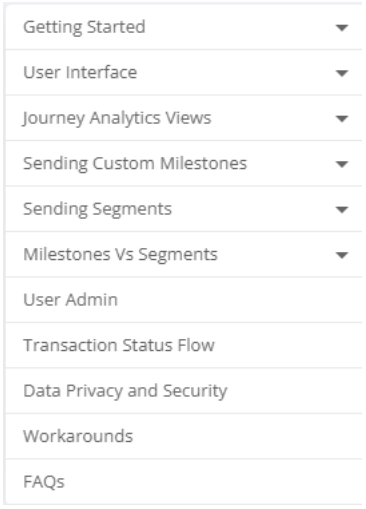


Quick Menu



On the left side of the page, the **Quick Menu** provides direct access to each product section.

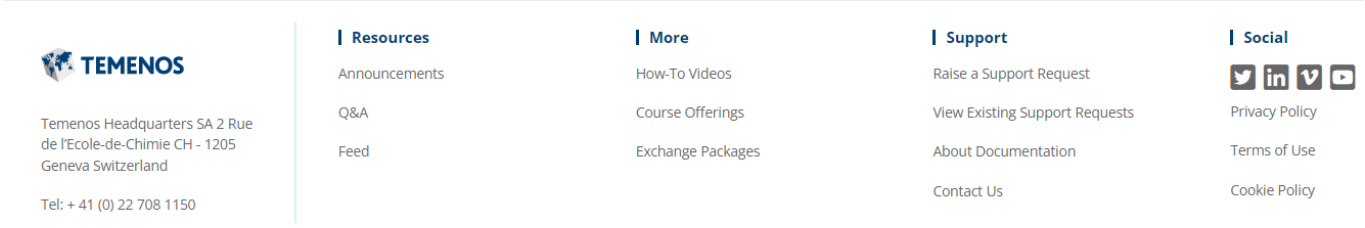
Table of Contents



On the left side of the page, the table of contents shows where you're at in relation to other topics in that product section. You can expand or collapse each section of the table of contents to control what's displayed.

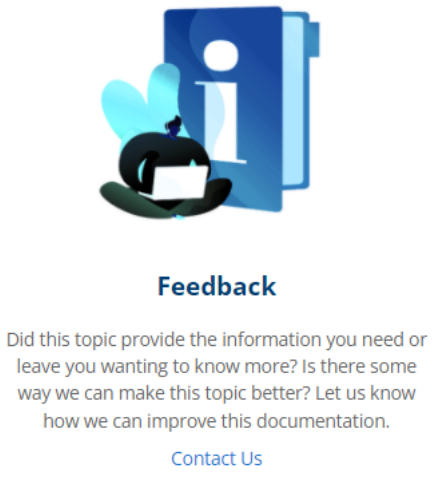
Resources

The footer on every page has useful links to our resources, support and social media.



Feedback

Every topic has a **Feedback** tile with a link to Contact Us via email. Use this to provide your feedback if any improvements or fixes are required.



What's Next?

Now, we would like to invite you to learn about customer journeys you can create with our Journey platform. Read Temenos Journey platform overview to get you started!

Temenos Journey Platform Overview

Journey Platform

The Temenos Journey platform is a **system of engagement**[1] that enables banks to create a seamless orchestration between the acquisition of customer data and the back-end **system of record**[2]. It lives in between your customer acquisition channels and your back-end systems, usually alongside your existing web CMS platform. This allows the Journey platform to maintain the loose coupling between the customer channels coming in from multiple sources, and the back-end systems without needing major architectural changes to the existing infrastructure.

The Journey platform solves the following key architectural challenges in building **omnichannel**[3] customer acquisition and **onboarding**[4] systems:

- Interoperability with existing systems of record.
- Interoperability with 3rd party systems for actions like ID verification, risk scoring, and fraud detection.
- Creation and maintenance of functionality common to any acquisition process, such as security, scalability, save & resume, receipt generation, attachments, and bundling of multiple forms.
- Behavioral Analytics for measurement and continuous improvement of the **CX**[5].

The Journey platform supports all the key facets of the customer acquisition process, by providing features to the front-end channels as functions, and simplifying the omnichannel experience for the end consumer. It delivers the ability to maintain a loose coupling of the state, user data, and input actions of the consumer to the back-end system, which allows rapid deployment of customer initiatives without complex and costly changes to existing systems.

---

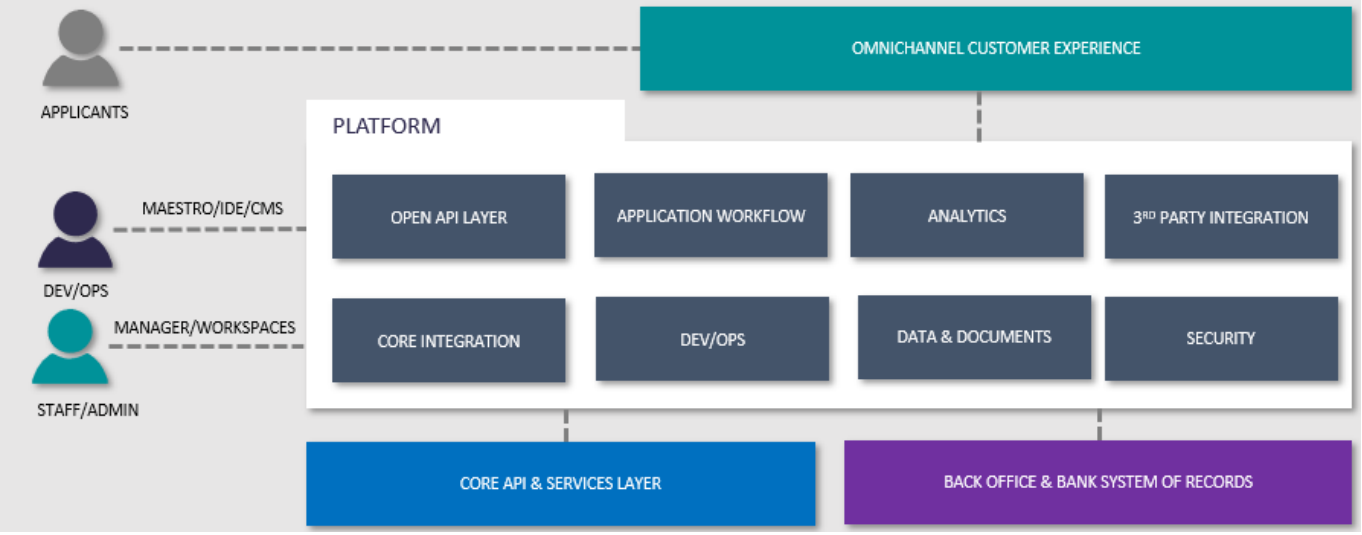
[1] Systems of Engagement (SoE) is the technology used by an organization to help facilitate and orchestrate the customer journey via more personalized, seamless interactions across the various touchpoints. These include social media channels, email marketing platforms, mobile apps, and content management systems.

[2] System of Record (SOR) is an ISRS (information storage and retrieval system) that is the authoritative source for a particular data element in a system containing multiple sources of the same element. To ensure data integrity, there must be one -- and only one -- system of record for a given piece of information.

[3] Omnichannel is a service delivery model that integrates different interaction points for customers such as online, by phone or in store. Omnichannel Banking offers banking access across these three methods.

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

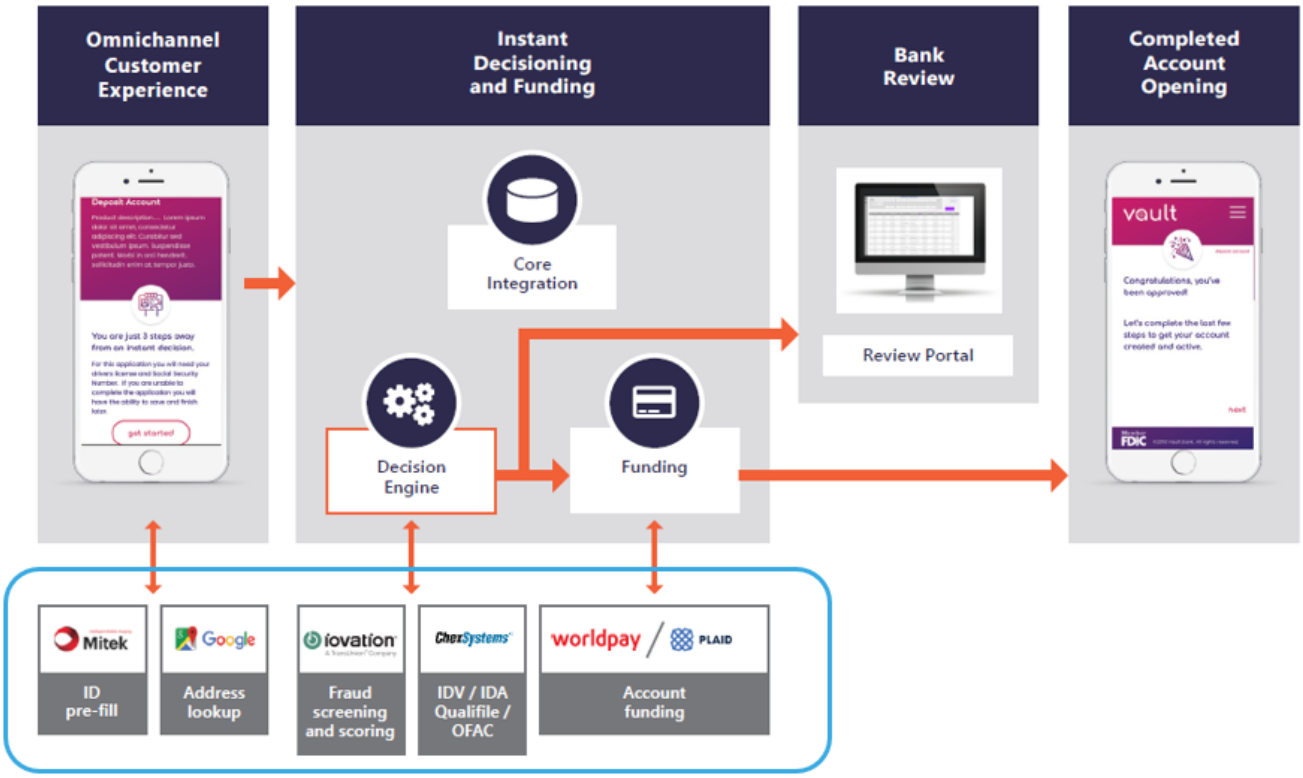
[5] Customer Experience (CX) is an aggregate of a customer's A-Z experience with a brand. It is not a snapshot in time, such as a key touchpoint or a critical interaction.



The Journey platform acts as a **PaaS**[1], providing a service platform on which banks can deploy their web applications. Inside the platform, there are multiple services that run independently of one another maintaining the platform functionality, such as security, front-end services, management, and reporting and operational monitoring. These services can be independently scaled and adjusted to fit the need of the customer's applications. The PaaS architecture allows rapid and simple deployment and scaling of services needed to deliver complex user experience orchestrations.

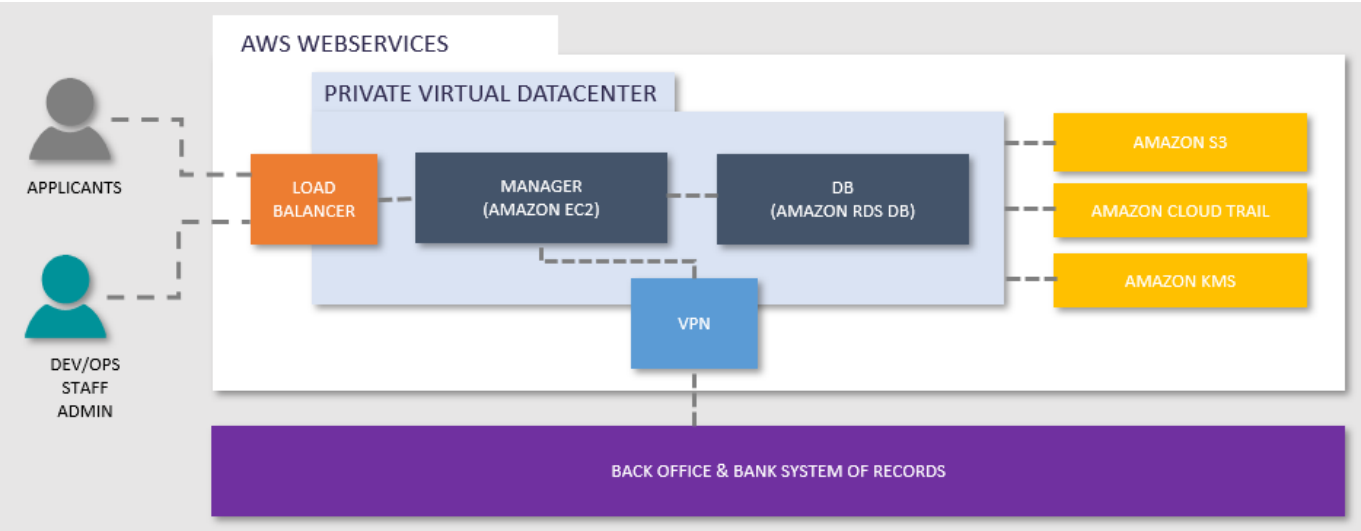
The example of a deposit account opening application built on the Journey platform is shown below:

[1]Platform as a Service (PaaS) is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.

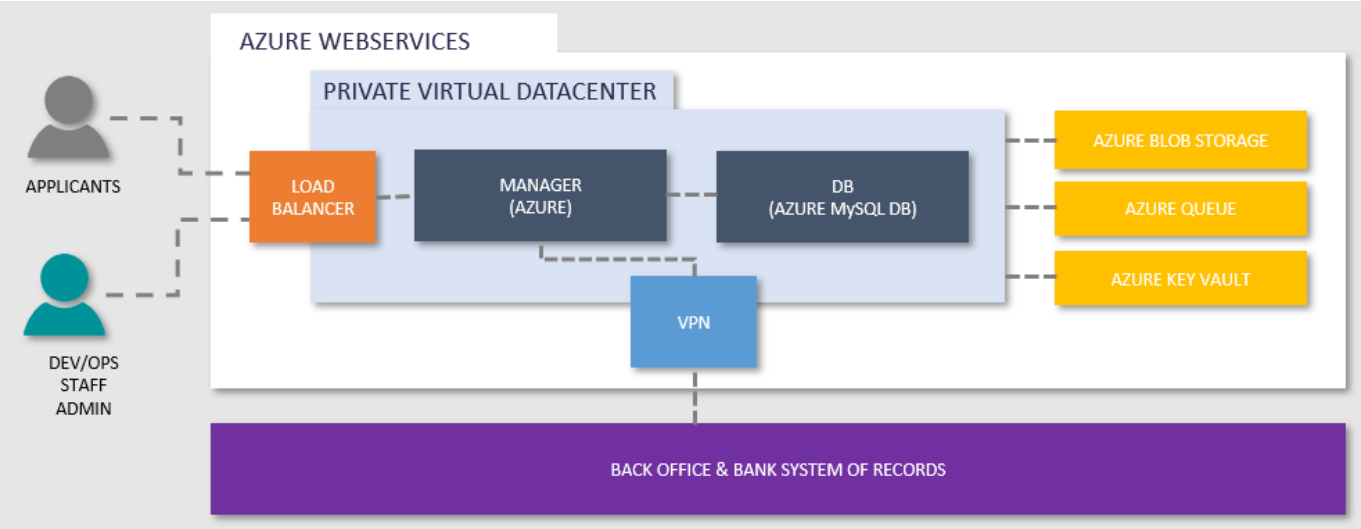


The Journey platform is mostly deployed as a cloud-based system and configured as a private secure environment for each bank. End users engage in the customer acquisition journey through the Journey platform's servers and application data is exchanged to and from bank systems of record via API services. The Journey platform configuration on the cloud provides for the highest levels of security, reliability, scalability and data integrity. We support both AWS and Azure clouds, as shown below.

AWS Cloud Deployment



Azure Cloud Deployment



Interacting with the Journey platform PaaS is made simple through the ability to make API calls from the front-end framework of choice, meaning that the complexity of building multiple platform services is reduced to a set of standard REST API calls. This allows the bank development team to dictate the terms of engagement with the platform, choosing to either leverage in-house skills to build experiences or using our tools for building customer interaction workflows, saving time and expense against having to build the entire set of services from scratch.

Products

The Journey platform incorporates the following products, also known as modules, to design, manage, and optimize the customer experience across channels without impacting back-end systems:

Journey Maestro is the **HTML5**[1], design environment that allows business teams to build sophisticated and elegant web form applications.

Journey Manager is the enterprise management system that hosts the web form applications and enables sophisticated customer interaction and 3rd party system integration between form transactions and back-end business processes.

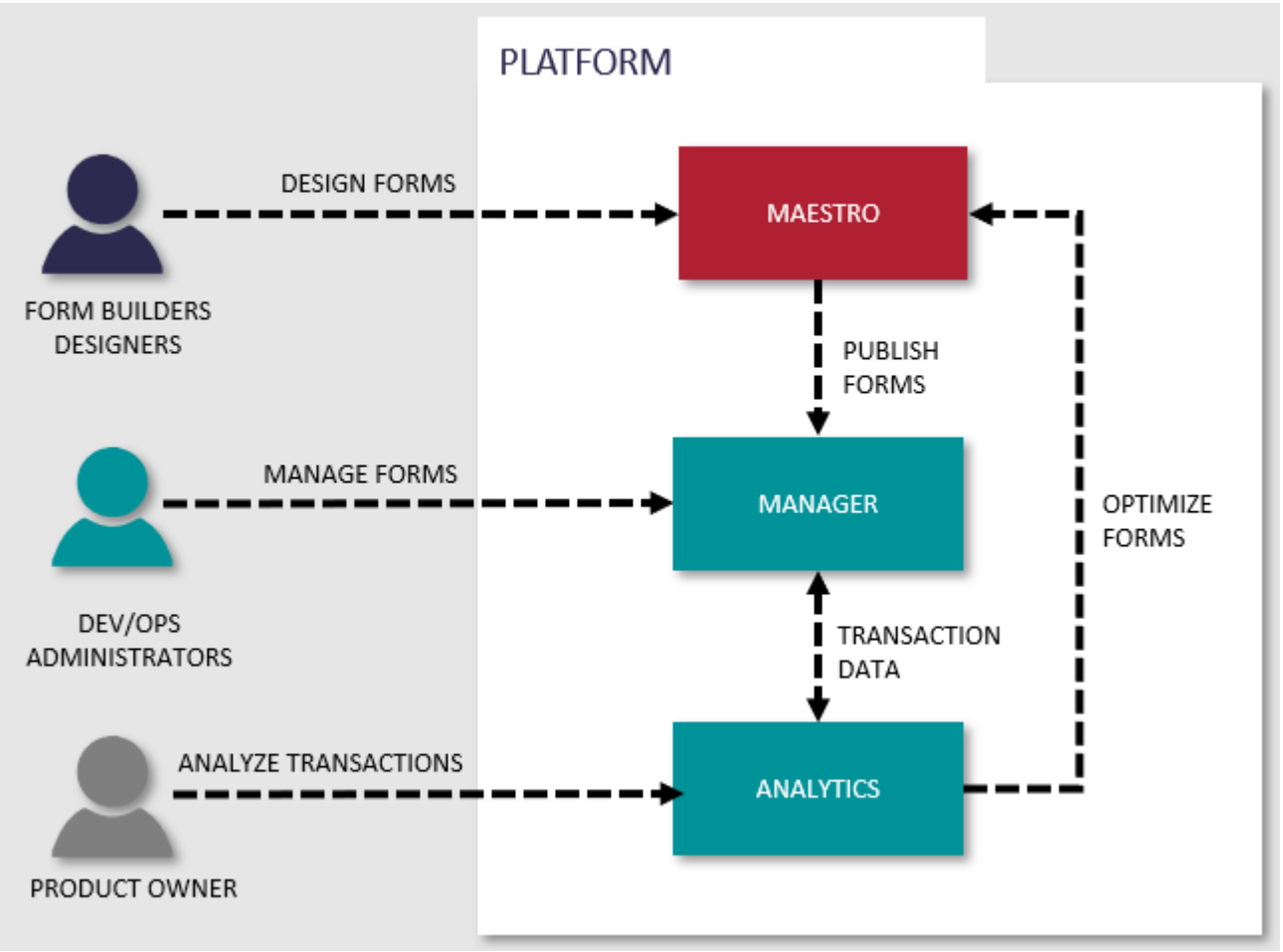
Journey Exchange is a catalog of pre-integrated FinTech applications to extend the power of the Journey Platform. These third party integration packages are made available to our Journey Platform customers for use in their online customer applications.

[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.

Journey Workspaces is an integral module of the Temenos Journey Platform that enables your organization to improve your customers' experience with onboarding assistance, rapid resolution, and communication of progress, regardless of the channel they are applying from. Out of the box, Workspaces includes the features you need to accomplish your business objectives.

Journey Analytics captures customer behavior to drive detailed form transaction analytics on how to reduce friction and abandonment within the web form applications. It allows you to watch transaction progress and monitor completion rates, bounce rates, and submission volume for different devices and forms. You can test new ideas with the standard Form A/B Testing in Manager or the more granular metrics with Custom Reports in Journey Analytics to instantly monitor results. Then when changes are needed, adjustments are data-driven similar to a web content manager, no coding changes required. This allows rapid deployment in a few clicks, no need for waiting on test and release cycles.

All our products are designed and built to be easily integrated with each other. The typical set up is shown below.



You can also integrate Manager with Salesforce using our API or use Open UX to design and create forms using your favorite IDE.

As you can see, the Temenos Journey Platform is based on various architectural and design principals. Most of these aspects are not covered in this overview article due to its complexity and sensitivity.

Next, learn more about platform integration and platform development.

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

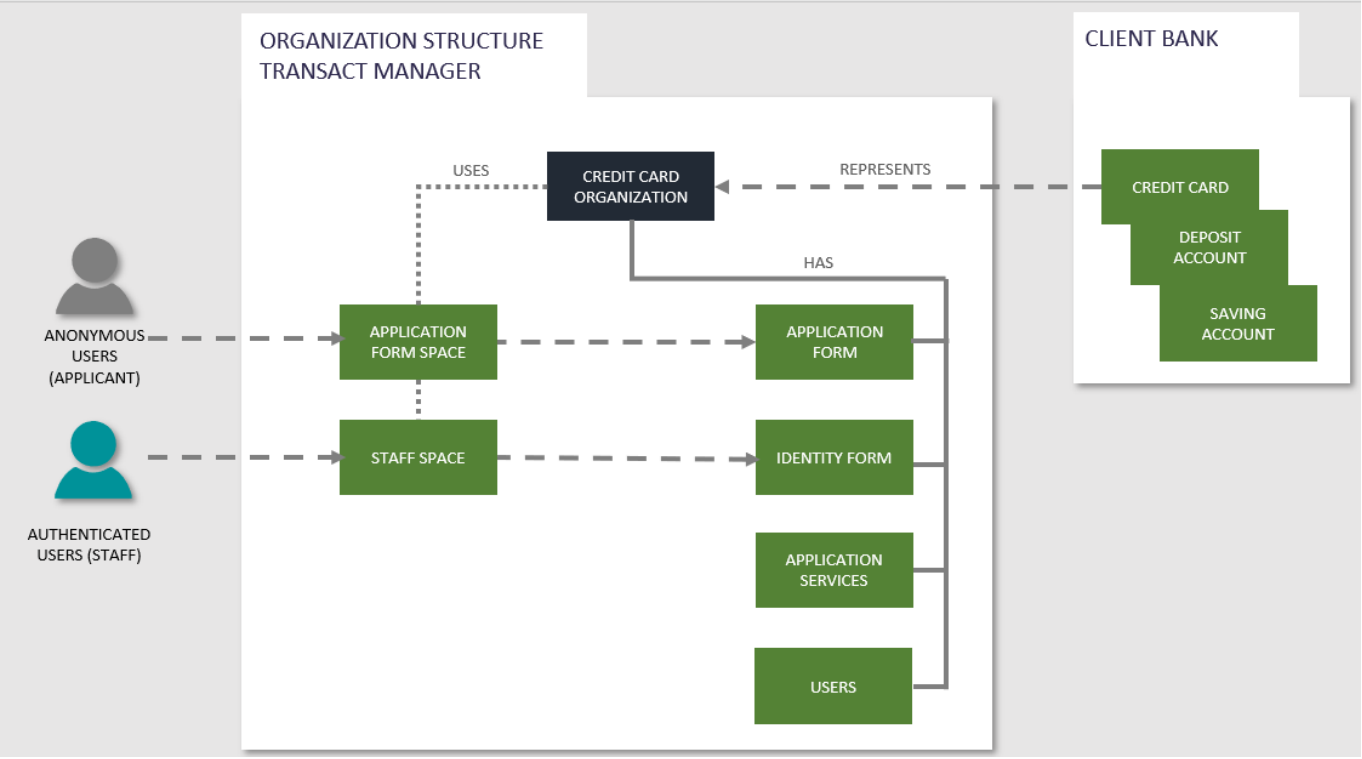
If you have any further questions or you need more information including pricing, contact your Customer Success Manager.

Next, learn about the Journey platform system requirements.

Introduction to Organizations

Journey Platform

An organization is a concept within the Journey platform that allows you to represent your business such as a company, a department, a branch within the company, a government organization, or a product. For example, if a client bank uses credit card application process, you can create a corresponding structure in Journey Manager, as illustrated below.



You can use organizations to:

- Create a well-defined structure of your assets (forms and PDF receipts) and resources (libraries and templates) within an organization, manage and migrate them across different environments to speed up form development life cycle.
- Have different core module configuration on the organization, system, form, and project levels to allow you to change the way forms appear and operate.
- Install new modules into your organization to integrate with other external systems.
- Provide secure users access to the resources within the organization and assign them different roles to perform certain activities.
- Encapsulate development and testing of forms relevant to your organization.
- Focus on monitoring, collecting and analyzing only your organization's runtime data to guarantee and improve forms user journey.

Every Jotform platform's core module uses the concept of organizations, but in slightly different ways. The brief summary is shown below with the links to detailed information for each module:

- Manager organizations own forms and their assets.
- Maestro organizations hold projects, widgets and libraries.
- Exchange uses organizations to allow you to install modules only there.
- Jotform Analytics uses organizations to control which form statistics users can view.

Next, learn about Manager.

Journey Executive Reading Path

Journey Platform | Executive

As an executive, you have little to no free time to grasp a new concept or understand a new product, so we have short-listed topics you need to check to get to know our Temenos Journey platform and its products.

First of all, this is all about customer **onboarding**[1], creating easy and smooth customer journeys, helping them to sign up for products they want to. This is not an easy task, but we have expertise and the Temenos Journey platform to make it happen. Our platform enables you to create and manager these customer onboarding journeys where each journey is a single dynamic HTML form or a form bundle that navigate customers through each onboarding workflow. You can think of our platform as a fast and reliable framework to build, implement and deploy these forms and workflows.

The Journey platform is a suite of core products working together to enable customer onboarding journeys. Journey Manager is the most important product and the heart of the platform. It's a complex application web server, written in **Java**[2] and **Groovy**[3] and deployed on a cloud or on-premise, that runs all other products, modules and bespoke solutions. Of course, it's not all that Journey Manager does - it has many other great features, such as form user data prefill, save and resume, and receipt generating to name a few.

The next product is Journey Maestro. It allows you to design, build and deploy application forms using the latest **JavaScript**[4] technologies.

Then, we have Journey Analytics that gives you a powerful tool to analyze customer journey experiences, so you can adjust your forms and business logic to continuously improve user experience and reduce the user abandonment.

The list wouldn't be complete without Transact Exchange, which provides a wealth of 3rd party **FinTech**[5] services, so you can easily plug them into your solutions using **Groovy**[6] scripts.

---

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

[2] 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.

[3] 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] 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.

[5] 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.

[6] 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.

To make your life easier, we have developed Journey Workspaces, which is the user-friendly, feature-rich business portal for managing application forms and workflows. We've purposely developed the collaboration jobs framework to help you build and manage complex workflows without the need to use **Business Process Modeling**[1] tools.

And, of course, there's Journey Springboard, the out of the box Deposit Account Opening (DAO) application for the US market, which you can use as the best practice to implement your own custom solutions.

Now, this was the brief introduction on our platform. We hope you've clicked a few links above, so you got some good insights. We understand it may be overwhelming at first, but don't worry, you can use our documentation search and the table of contents to find out more about each product and how they may benefit you.

Next, learn more about the starter path.

[1]Business Process Modeling Notation (BPMN) is a method of illustrating business processes in the form of a diagram similar to a flowchart.

Journey Starter Reading Path

Journey Platform | Starter

As a starter, you want to understand quickly a few key concepts about our Temenos Journey platform and its products, so you can start using them.

Our platform is rich in functionality and therefore complex, so it takes some time to get your head around it. We recommend you begin with the executive path first, as you only need to read a handful of articles and understand a few key things.

The first product to learn about is Journey Manager, because it's the most important product that other products utilize. Journey Manager has to be installed and configured before you can start using it. You can install the product yourself, or ask your IT department to do it for you. However, Journey Manager is often installed and run on a cloud, so you only need to log in using credentials provided to you.

Your Journey Manager experience starts with the home dashboard. It is abundant with useful system information, recent forms and services, and transaction history, so it may be tricky to decide where to go next. We suggest you look at organizations, as this is the first building block you need to create and configure before you can deploy any forms. Then, check what form spaces are and whether you have permissions to access them. Form spaces host forms and control user accessibility using groups and roles. Manager comes with the advanced **SSO**[1] security configuration, including **LDAP**[2] and **ADFS**[3] to name a few, and you need to be quite familiar with these technologies to set it up.

One of the best features of Journey Manager is Groovy services, which you can use as they are or write your own. This gives you unbounded possibilities to implement various solutions, but yet you have to know how to write **Groovy**[4] scripts. We recommend you check Transact Fluent SDK overview and benefits of using Groovy before you dive into writing your own stuff. It's worth looking at the standard services, as they may already address your needs, plus they are professionally written and thoroughly tested. This is especially the case for the 3rd party service integration that is usually complex and therefore time consuming part of application development. Luckily for you, we have Transact Exchange that comes with dozens of packages you can easily install and use to enhance you **FinTech**[5] applications.

---

[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]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.
[3]Active Directory Federation Services.
[4]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.
[5]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.

Manager includes a powerful framework, the collaboration jobs, for building onboarding workflows. This is a quite complex area and requires significant knowledge of our products, but it's worth reading a few articles to understand what you can possibly achieve with this tool.

Journey Manager is the developer friendly product and comes with extensive logging and troubleshooting functionality, so you can debug your services and configurations to tune and fix them. Additionally, it allows you to create your solution in a development environment and then propagate it to another environment and finally to production using our multi-environment support. Journey Manager also supports the horizontal scaling, meaning you can add server nodes to cater for your production needs and demands. We recommend running several server nodes, with at least one dedicated node to generate and render receipts.

Journey Manager has a great deal of system configuration that is paramount to keep the product running. We recommend starting with outgoing email service and data retention management. While Journey Manager is designed to sustain high load of transactions, it requires some maintenance and operational support every now and then and when an issue occurs. There's a lot to learn here, so we suggest you start with submission life cycle and transaction details.

The next product you need to understand is Journey Maestro. It is extremely rich in functionality and has user-friendly interface allowing you to build sophisticated and elegant web form applications based on modern technologies, such as **HTML5**[1] and **JavaScript**[2]. We recommend looking at forms and components first, then get your head around libraries and entities, as these are enablers for best UX design principles and software reuse. This content is rather complex and requires some knowledge of HTML, JavaScript and UX design. When you've mastered form creation, you can learn how to build them in Maestro and then deploy them to Journey Manager, where they will be run.

Our Journey platform enables you to apply **SDLC**[3] to the process of building, testing and running your form applications. Journey Analytics fulfills a very important role in this development cycle, as it allows you to analyze customer journey experiences that are based on your forms. There's a lot of theory behind how we collect, analyze and present data, but we recommend you start with the product's dashboard view and then follow the links as you read. If you're after a few simple reports and A/B testing caparisons, we suggest you look at Journey Manager's analytics, which is a simplified version of Journey Analytics. Nevertheless, this tool gives you a good insights on transaction trends, such as completion and conversion rates, form abandonment and bounce rate, that are the key metrics for form improvement to try to make the customer journey frictionless.

---

[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]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.
[3]Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares.

As you can see, there's a wealth of information generated during user interaction with forms. It's extremely difficult to manage it without our Journey Workspaces module, which provides key features for bank staff to provide great customer service. We recommend starting with Workspaces UI Tour and Workspaces Core Concepts to gain an understanding of Workspaces fundamentals. One of our newest products is Journey Springboard, the out of the box Deposit Account Opening (DAO) application for the US market, which enables banks to rapidly deliver an outstanding digital customer acquisition journey using a pre-built solution on an extensible platform that grows with the bank. You can use it as a best practice, pre-built custom solution to get to market faster with acquisition journeys. The Springboard DAO solves the complex challenges of the deposit account opening process, including compliance and risk management requirements. Through pre-integrated connections to proven FinTech providers, Springboard DAO addresses the critical issues of fraud screening, ID verification, risk mitigation, data pre-fill, and funding.

Now, while this was more a detailed introduction on our products and the development practices we use, there's much more for you to read and learn. You can always go to our documentation search or navigate the table of contents to find out more about each product.

Next, learn about the Platform Developer path.

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.

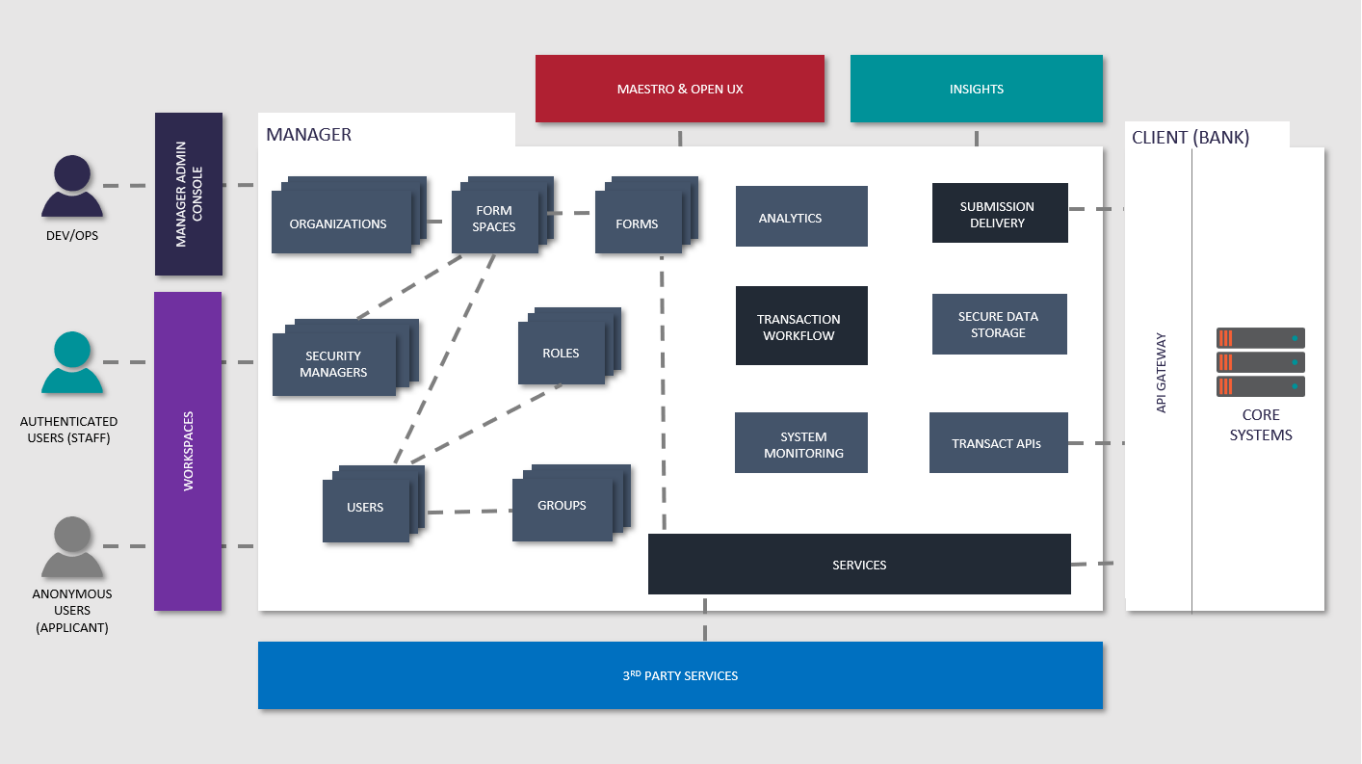
[2]Customer Experience (CX) is the product of an interaction between an organization and a customer over the duration of their relationship. This interaction is made up of three parts: the customer journey, the brand touchpoints the customer interacts with, and the environments the customer experiences (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 Maestro Overview

Maestro | Form Builder | Platform Developer | 5.1 & Higher

Journey Maestro is a tool for designing and building application forms that implement various customer onboarding journeys. Maestro is a product suite of the Temenos Journey platform, so it is seamlessly integrated with all other products to give you a maximum edge in creating web-based apps using latest technologies.

Maestro allows anyone with moderate technical expertise and skills to design and configure web-based forms using the built-in Maestro editor. The Maestro editor is a **WYSIWYG**[1] editor so you can see how a form will look like while creating it. You can also preview the form in different device emulators or render it as a PDF or HTML receipt. The Maestro editor comes with the rich text editor to give you a flexibility of using various content formatting.

The forms that Maestro produces are single-page AngularJS applications with responsive designs, and can be previewed in formats suitable for smartphones, tablets, and desktop computers. A single design runs on all devices, takes the screen size into account, and gives an optimal experience without creating versions for each device.

Maestro provides you with the following features to streamline your form development process:

- Drag-and-drop pre-built components directly into the WYSIWYG editor for rapid form creation
- Create custom components from scratch or by combining existing components
- Use templates to maintain a consistent look and feel across a large number of forms
- Generate receipts based on the design of a form or design custom receipts
- Implement save and resume functionality without writing any code
- Define rules that run arbitrary JavaScript code when events, such as a click, occur
- Maintain multiple translations of a single form

For more information, see the complete list of Maestro features.

Maestro uses the following entities to enable you to design and build forms:

- Organization - contains projects and organization level libraries
- Project - contains forms, components, templates and project level libraries
- Form - contains user data captured using form components
- Template
- Receipt
- Component
- Library

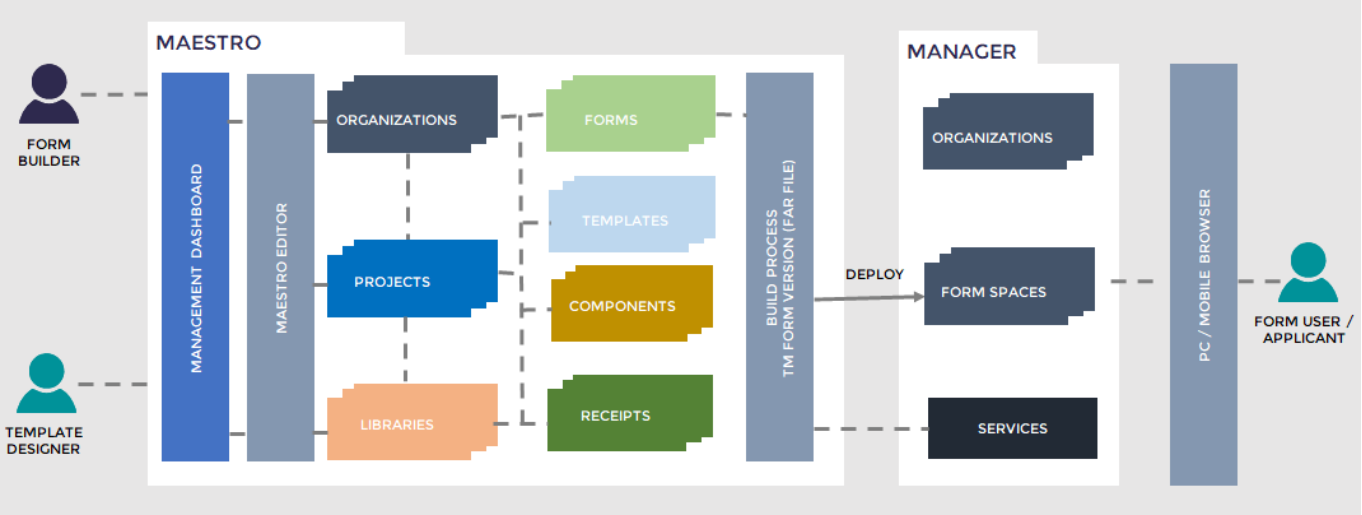
---

[1]WYSIWYG, an acronym for What You See Is What You Get, is a system in which editing software allows content to be edited in a form that resembles its appearance when printed or displayed as a finished product, such as a printed document, web page, or slide presentation.

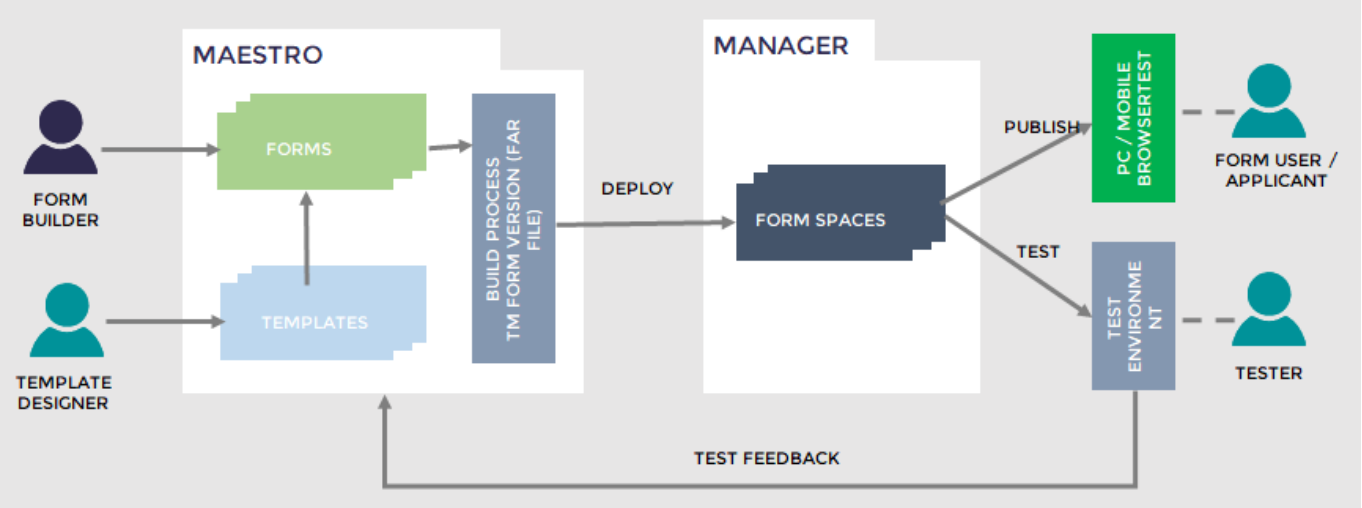
Organizations, projects and forms make up the three-tired hierarchy that allows you to minimize configuration effort yet maximize development efficiency by using a correct level or scope while defining and configuring a form.

Maestro relies on Journey Manager to host forms, process user inputs and store all relevant information to ensure a form user gets the best onboarding experience. Therefore, Manager has the similar concept of entities as Maestro, plus it also includes back-end configuration and data that are not exposed to web-based forms.

The diagram below shows the main entities of Maestro, communication with Manager and interaction with form users (applicants), Form Builders and Template Designers.



A typical Maestro form goes through several stages of the customer journey experience, such as, Prefill, Data Capture, Save and Resume, Payments, Notifications, and Collaboration between various participating or approving parties. Some of these stages have an impact on a form's design. Let's explain it with the diagram below, which outlines the development workflow that Maestro provides to Form Builders and Template Designers to create a customer journey.



This workflow begins with a template designed in Maestro by a Template Designer. Once the template is created, Form Builders can start work on new forms based on the template, taking advantage of the common features built into the template and maintaining compliance with brand guidelines. The Template Designer can continue adding to the template incrementally, even while Form Builders are creating forms based on the template, without affecting any forms in progress.

Once a form is implemented, the Form Builder can build the form and import it into Manager. After importing, the form can be tested across multiple environments and, if needed, further improvements can be made before the form is published into a production environment for customers to use.

Next, learn about Management Dashboard.

Journey Exchange Overview

Exchange | Platform Developer | v5.1 & Higher

The Journey Exchange, formerly known as the Avoka Exchange, is a catalog of pre-integrated **FinTech**[1] applications, such as Mitek, FIS, Mastersoft, and LexisNexis, to extend the power of the Journey Platform.

These third party integration packages are made available to our Journey Platform customers for use in their online customer applications. A list of published and available packages per region is maintained at: Exchange Packages

The Exchange catalog has been integrated into Journey Manager from v18.05 to allow you to easily download Exchange packages directly into your Manager environment.

You will need the Exchange permissions to be able to do this. However, for all earlier versions of Manager, a zip file gives you access to the Exchange package services and Maestro assets. To find out more, refer to:

- Access to Exchange Packages
- Package Installation
- Exchange Permissions
- Package Configuration

Watch this video for an introduction to the Exchange.

[1]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.

Contacting the Exchange Team

You can contact the Exchange Team to suggest an idea or raise a support request through the Customer Care portal:

Got an idea? Need to know how to log an idea?

Got an issue? Need to know how to log a defect or issue?

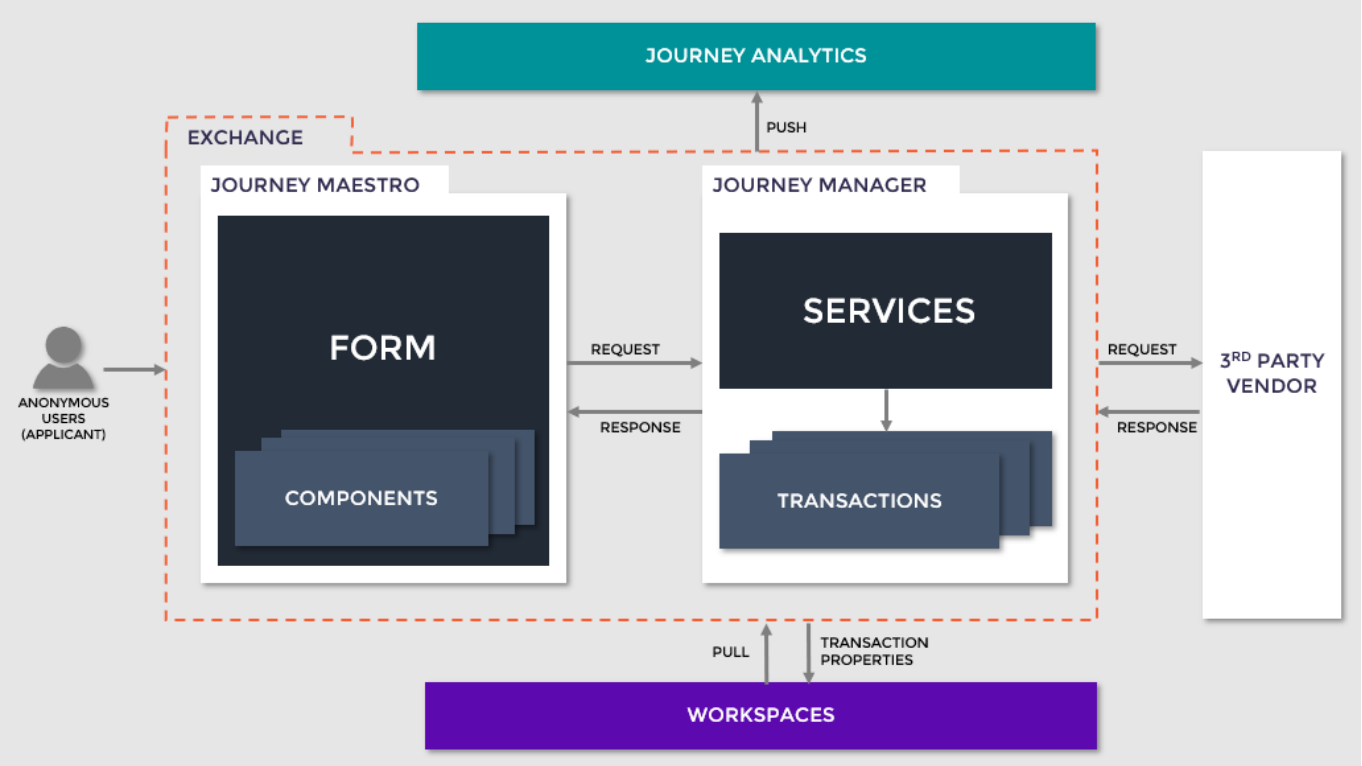
1. Ensure you're using the original Exchange package (not modified services or Maestro assets).
2. Include your Manager, Maestro and Exchange package versions.
3. Detail the steps to help us to reproduce the issue.

Exchange Package Architecture

All Exchange packages are based on the same architecture and interact with Manager and Maestro in the same manner.

As the user interacts with an Exchange package implemented in their application (form), the third party services are called via Manager, and these events are recorded in Journey Analytics, the platform's analytics tool, to be further analyzed in the future.

The transaction properties of the user's application are also sent to the Workspaces portal, so the application life cycle can be reviewed by the Workspaces users.



Exchange Development Schedule

More Exchange packages are always being developed.

Packages being considered for development are here: Development Schedule

Exchange Partnerships

Interested in becoming an Exchange Partner and publishing your own solutions?

The Exchange Partner Program is designed to allow your team to participate in the Avoka Exchange service by developing, certifying and publishing your own solutions. Find out what is involved:

- The Exchange Publishing Process
- Exchange Certification
- Developer Resources

Exchange Developer Resources

Looking for answers as you develop? Let us assist you to get your Exchange package built, certified and published.

We've got some important tips and guidelines that are a great place to start: Developer Resources

Journey Workspaces Overview

Workspaces | Workspaces User | 18.11 | 21.11

Efficient resolution and assistance for customers is an essential part of omni-channel customer acquisition. Journey Workspaces, powered by Journey Manager, is an integral module of the Temenos Journey Platform that enables your organization to improve the quality of customer experience with onboarding assistance, rapid resolution, and communication of progress, regardless of the channel they are applying from.

What is Workspaces?

Journey Workspaces is a highly configurable and feature-rich business portal for banking staff, designed to support resolution and customer service activities related to account opening and onboarding directly in the account opening process. Workspaces supports a digital workflow for in-bound applications that enables bank staff to initiate, review and resolve customer applications, and provide great customer service. Bank staff are offered an efficient method of receiving, tracking, organizing, and resolving pending applications that improves both their productivity and the accuracy of the application review process.

In the Workspaces portal, you can search and filter to find an application then view all the key application details, enabling bank staff to provide a richer customer assistance experience and make an informed decision about an application's outcome. You can filter and assign applications for review based on criteria such as workloads, decision status, risk, or incomplete application data. Application assignment features enable managers to control staff workloads, helping to reduce the onboarding delays that can lead to customer dissatisfaction. Throughout the account opening process, Workspaces captures notes and user- and system-initiated actions that document an easy-to-understand audit trail of the application life cycle. With visibility into all the application information at your fingertips, you can quickly resolve pending applications with a final decision and communicate back to the applicant to increase customer acquisitions and improve customer service.

Out of the box, Workspaces comes with a collection of thoughtfully designed, pre-tested templates for several role-based spaces that facilitate processing, helpdesk, and application management activities as well as kick-starting applications on behalf of customers and monitoring their progress. These templates represent common work scenarios and take advantage of third-party integrations to add meaningful features. The templates accelerate the installation and configuration process and can also be used as the basis for your own custom Workspaces experience.

Who is Workspaces for?

Workspaces provides features for several user types or purposes.



Processing Staff Workspaces offers review capabilities for pending applications that need manual action.

 Support	**Helpdesk Staff**	Workspaces allows support staff to find any application that needs further assistance quickly and easily.
 Assist	**Assisted Channel Staff**	Workspaces allows assisted channel staff to kick start the application process and track application progress.
 Manage	**Managers / Supervisors**	Workspaces provides tasks management capabilities for managers.
 Applications	**Authenticated Applicants**	The Workspaces Applicant space offers a self-service portal for managing application creation, follow-ups, and keeping track of application progress.

To learn more about the Workspaces Applicant space, see Workspaces Applicant Space Overview, The Applicant Space Search Screen, and The Applicant Space Details Screen.

Getting Started

Configuration

Workspaces is a highly configurable product, so we provide a set of default configurations to get you up and running faster. You can download the Workspaces distribution, including the default configurations. More information is available in the Workspaces technical documentation to help you configure your Workspaces portal to meet your needs.

Browser Support

Workspaces has been designed for and tested against the following web browsers.

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Microsoft Internet Explorer 11 (limited support - see **IE11 Support** below)

For the best experience, we recommend that you use the latest version of Chrome, Firefox, or Edge, and that you keep your browser updated, especially through the application of security patches.

While you may be able to access Workspaces using other browsers, there's no guarantee that everything will work, and you may not enjoy the great user experience available when using a supported browser. We strongly recommend that you use a supported browser for the best Workspaces experience.

IE11 Support

Workspaces is supported on Internet Explorer 11 (IE11), commencing from Workspaces v18.11.4. Workspaces is feature-compatible with IE11; however, the user experience on IE11 may not be as great as when using a modern browser due to limitations imposed by IE11.

What's new?

Here's some information to help you discover the great new features and improvements in each Workspaces release.

21.11 20.05 19.11 19.05 18.11

Here's what's new and improved in Workspaces 21.11.



Improved build and deployment process	An improved build and deployment process for Workspaces is now available from the Product Artifactory.
Applicant Space template	A fully configurable workspace for authenticated non-bank users to manage the application journey.
Support for Job Properties	Support for configuring the Key Info card and custom cards using the application's job properties.
Localization improvements	Support for switching the language within Workspaces, and other minor enhancements to the localization framework.
Search improvements	Configure search to include fields that are not part of the item list data.

Here's what's new and improved in Workspaces 20.05.



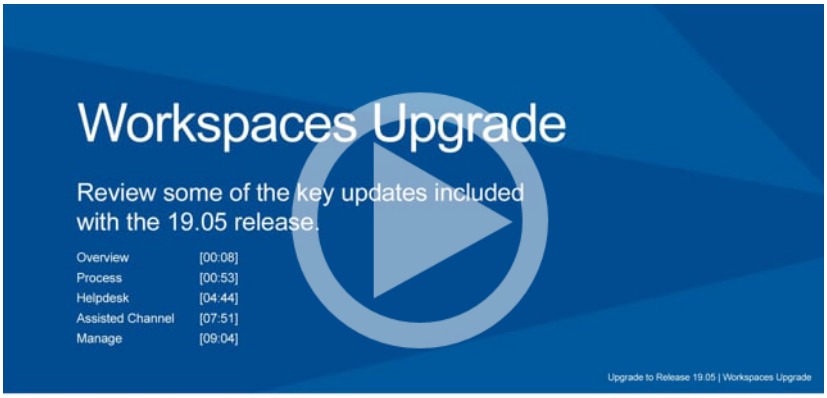
Improved build and deployment process	An improved build and deployment process for Workspaces that supports seamless integration to your DevOps practices such as CI/CD.
Task-specific actions	Task-specific actions are presented alongside each task in an application to give you confidence you are acting on the right task.
Focused search	A new focused search option that lets you search on a specific application data item and get results faster.

Improved UI / UX	A fresh new user interface for an improved user experience.
Improved custom card configurations	Improved custom card configuration including permission control and an option to hide empty/null data.
Improved session logout	Automatic session logout after a defined period of inactivity.
Date picker enhancements	Enhancements to the date picker to support single date selection and custom date formats.
Enhanced search/-filter behavior	New search/filter behavior on the List pages: the latest search/filter is retained in each space until cleared or the session ends.
Enhanced application assignment	The assignee full name is displayed when assigning a task to a user.
Enhanced application listing	An enhanced view configuration extending control over line wrapping in data tables.
Enhanced documents panel	Attachment name is now included for each item in the Documents panel.

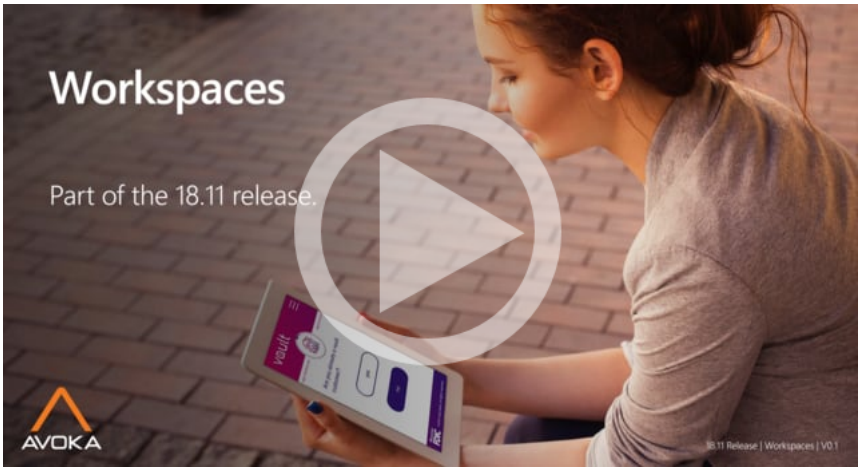
Watch this video to discover the great new features and improvements in Workspaces 19.11.



Watch this video to discover the great new features and improvements in Workspaces 19.05.



Watch this video to discover the great new features and improvements in Workspaces 18.11.



Journey Analytics Overview

Journey Analytics | Analytics User | Latest Version

Journey Analytics, formerly known as Transact Insights, is a purpose-built behavioral analytics module that provides additional extensive analytics on your applications hosted on the Temenos Journey Platform.

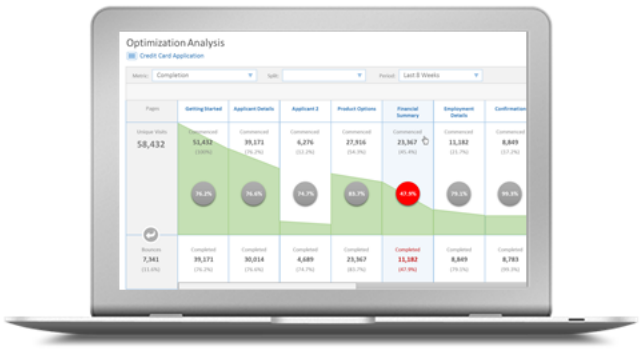
Once set up, Journey Analytics aims to assist customers to track the performance of their applications on their Journey Platform from an analytical perspective. Journey Analytics provides a wide variety of analytical views that capture completion analysis for one or more applications. As well as these features, Journey Analytics also supports rich filters that allow customers to understand the device type, browser and other attributes of the devices used to complete applications.

Journey Analytics must be enabled for Maestro, Composer or Open UX applications in Journey Manager to be able to collect Journey Analytics data.

A key value Journey Analytics offers is the behavioral analytics to assist our customers to understand user engagement with the form, the effort it takes to submit an application, and where users are abandoning their applications.

These fine-grained metrics help to identify abandonment hotspots in the customer application and further explore problematic fields to:

- Reduce the effort it takes the user to complete an application.
- Improve the user experience and **onboarding** [1] process.
- Optimize the application to drive conversion and completion.



Analytics Views

Journey Analytics now offers eight main analytical views for both application performance analytics and user behavioral analytics:

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

1. Dashboard View – gives a high-level overview of application performance providing key metrics on completion, device distribution, completion times and recent completion history.
2. Timeline View – provides a trend of application statuses over a selected period. In addition, you can aggregate the data at various levels and split the data by device type, browser, application version and other options.
3. Dropoff View – provides behavioral analytics capturing key insights on user interaction with the application. It includes section and field level metrics that help to identify abandonment hotspots, problematic fields, and application flow.
4. Field Analysis View - provides detailed field-level statistics for each section of the application being analyzed. It also helps to identify abandonment hotspots by aggregating all field level behavioral data for all transactions during the selected time frame. In v18.05 and earlier this view is accessed through the Dropoff View.
5. User Journey View – this view provides a map of custom milestones implemented in the application and how users are progressing through them towards completing the application. It also provides insight into all possible journey paths users are taking to complete transactions, and allows customers to perform segmentation analysis of the users based on the data sent by the Segment APIs from the application form or Journey Manager. Introduced with Journey Manager 18.05.
6. Collaboration Job View - this view uses a node graph to represent the Collaboration Job workflow. It provides a view into key metrics at the job, step, action, section and field level. Introduced with Journey Manager 18.11.
7. Export View - this feature allows you to export analytics reports to CSV, with the ability to select the granularity of your reports and retrieve exports for multiple applications in a single file. Introduced with Journey Manager 19.05.
8. Custom Reports View - this view provides a customizable dashboard that makes it easy to compare the performance of your application versions, across many different metrics in a single view. Multiple analytics can be displayed in the form of report cards that can be arranged and resized in any location on the screen. Introduced with Journey Manager 19.05.

Journey Analytics User Interface

The Journey Analytics User Interface is common to all analytical views. Use the interface to select the required analysis views, select the timeframe for analysis, and set global filters and preferences.

From Journey Manager 18.11 a new user interface called the Visualizer was introduced, which is a modern user interface that brings together modern best practices to deliver a refreshed user experience. For customers on Journey Manager 18.11 and later, all analytical views will be seen through this refreshed user interface. For customers on Journey Manager 18.05 and earlier, the original legacy Dashboard user interface will display. Examples of both user interfaces are available in this documentation.

Terminology

Journey Manager forms or applications designed in Maestro, Composer or Open UX are referred to as applications in Journey Analytics, and once a user submits the application in the browser it becomes a completed transaction in Journey Manager.

Transaction Status

Transactions on Journey Manager may progress through any of the following statuses; Open, Bounced, Started, Saved, Completed, Abandoned.

Strict No PII Policy

Transactional data entered by users is not stored in Journey Analytics, however, Journey Analytics does store information about the transaction status and user behavior events from the application. This behavioral data is stored and used to populate all the Journey Analytics analysis views.

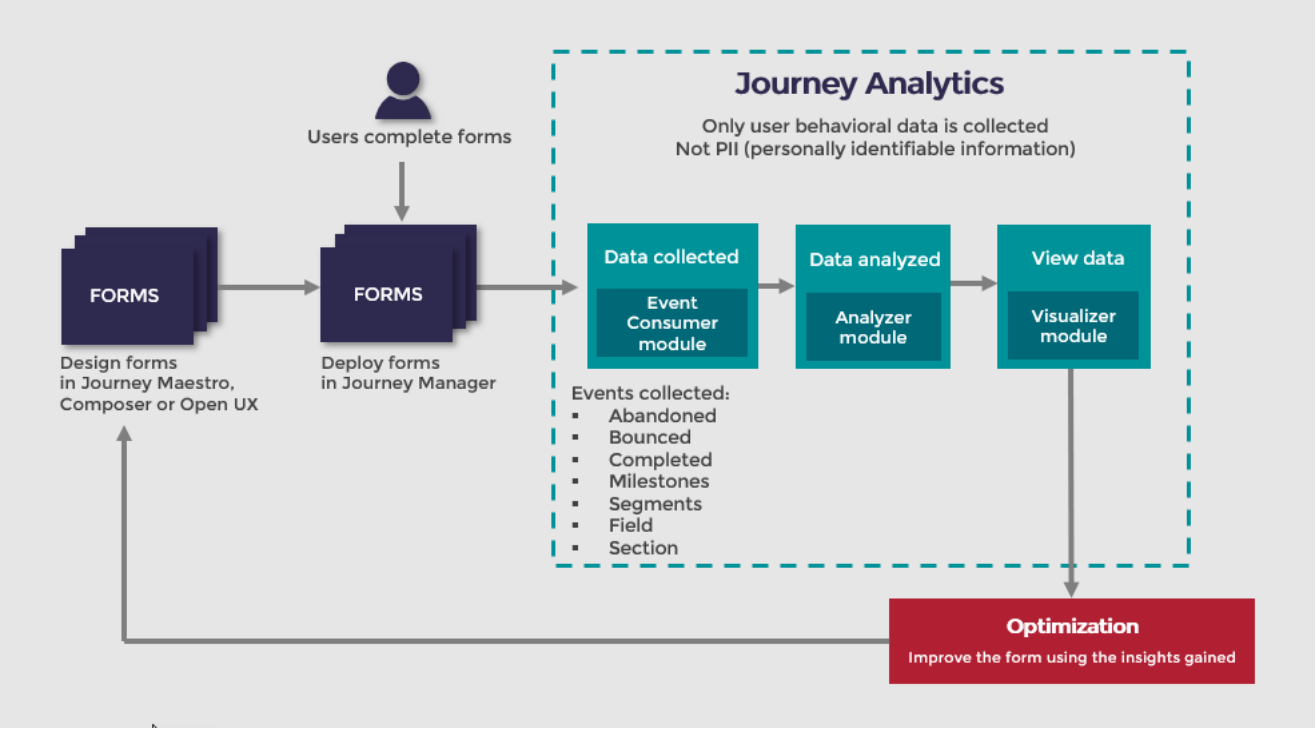
The transaction status and user behavior events DO NOT include any user input data itself and hence Journey Analytics DOES NOT collect any **Personally Identifiable Information**[1] (PII). The Journey Analytics service has been explicitly designed to NOT record or store user input data and PII.

You may verify this by using the browser's F12 tool to monitor the network traffic of an Journey Analytics enabled application. Journey Analytics employs all possible mechanisms and strategies to NOT store data relating to PII. For more information, see Journey Analytics Data Privacy and Security.

Architecture

Journey Analytics is built and hosted on the Google Cloud Platform (GCP) infrastructure. It is a fully managed service built using the GCP. The image below describes the high-level architecture of Journey Analytics and how it fits into the Journey platform.

[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.



Journey Analytics collects events from two sources – Journey Manager and the browser where the application is rendered.

Journey Manager sends transaction status events and applications rendered in the browser send user interaction events. These two sources support the detailed statistical analysis of user behavior within applications.

The user interaction events from the browser include information such as - field visit, field completion, validation error, section navigation etc. These analytical events are directly comparable to the Google Analytics or Adobe Analytics events recorded by these analytics services. For more information, see Journey Analytics Data Privacy and Security.

Setup and User Administration

Some setup and user administration is required to use Journey Analytics. Users must have the Journey Analytics user role and space added to their Journey Manager user accounts to be able to access Journey Analytics.

Retail DAO Overview

Springboard | Form Builder | Retail DAO 3.0

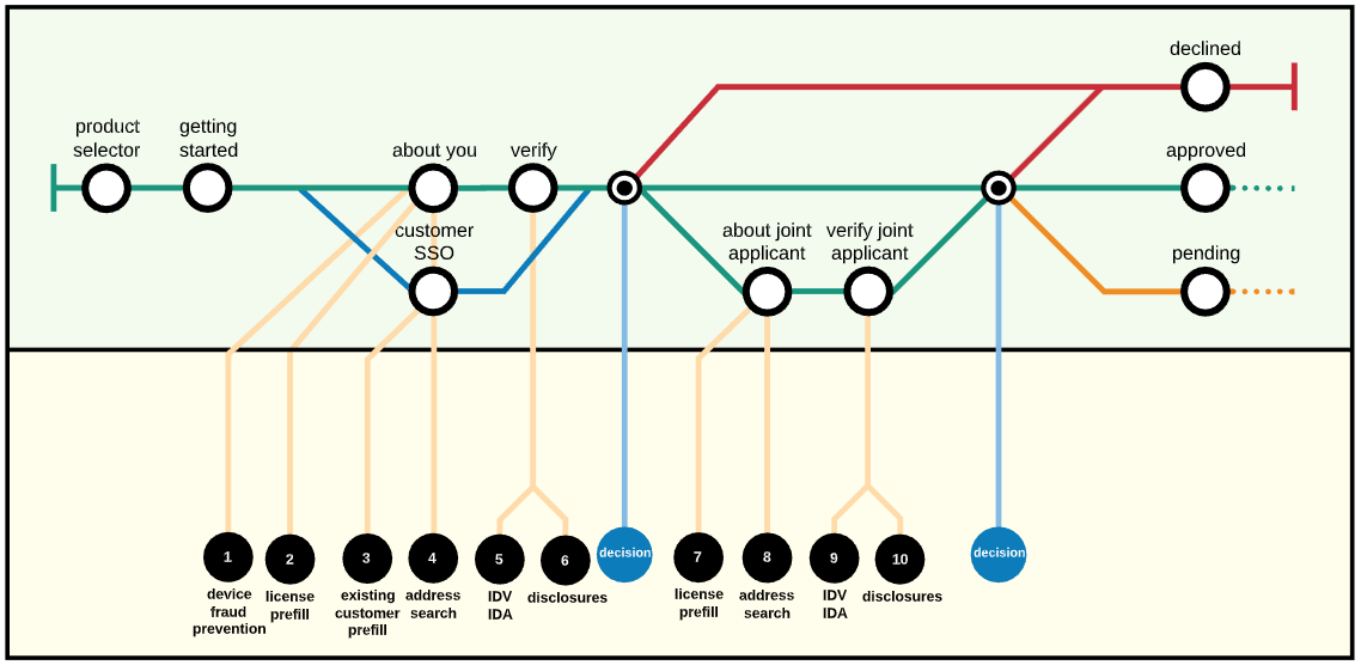
The Small Business Springboard Solution for US Deposit Account Opening, Springboard **DAO**[1] for short, is a ready-to-use solution to enable **onboarding**[2] of new and existing customers for US Direct Deposit Accounts (**DDAs**[3]). Springboard is a starting point for customer specific DAO projects, which requires significantly less development than a typical Journey project.

This is achieved by pre-building most of the typical components, required to support US DAO solutions on Journey Manager, such as:

- Maestro template and forms.
- Transact Functions including third-party integration.
- Standard workflows and collaboration jobs to support manual review and approval activities.

For more information on each of the components supplied with the solution, see components.

A typical Springboard application flow is shown below.



[1]Deposit Account Opening (DAO). DAO bank forms allow users to open Checking, Savings and Money Market Accounts.

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

[3]Direct Deposit Account (DDA) is simply a checking or savings account which offers the ability to send and receive funds electronically.

Audience

This documentation is designed to assist your implementation team members with the initial setup of their local development environment, deployment of solution artifacts and configuration of standard solution components.

This document is also intended for all project team members:

- Project Manager
- Business Analyst
- Solution Architect
- Technical Lead
- Developers
- Quality Assurance Team

Areas of Value

Springboard provides value to Client Services teams and our clients in several ways:

- Offers our clients a vision of what a good **DAO**[1] solution should look like.
- Offers our clients a head start on developing a custom DAO solution by leveraging our base Springboard solution.
- Serves as Temenos' current reference architecture for a Journey Manager / Journey Maestro DAO solution.
- Offers Client Services teams a framework for implementing a custom DAO solution.

Prerequisites

To start using Springboard and get the most out of it, you need to be familiar with:

- The Temenos Journey platform
- Journey Manager's services
- Maestro
- Narratives
- SCM
- Git

More information is available in our online documentation and on our resources website. We also offer instructor-led and online training (authentication required) on developing Journey platform solutions.

[1]Deposit Account Opening (DAO). DAO bank forms allow users to open Checking, Savings and Money Market Accounts.

What Remains to Be Done

While large portions of the solution are pre-built there is still work required for every customer specific implementation. Typically, the bulk of the work for a client should be focused into three main areas:

- Configuration/Setup - such as style/brand, labels, configurable options, deploying and testing components, testing integrations, etc.
- Core Banking Integration - to support account and customer creation, as well as other critical onboarding activities
- Customization - which includes specific modifications to the project scope, which have been added to the project SOW or via CRs

Next, learn about Springboard architecture.

Adobe Reader PDF Form

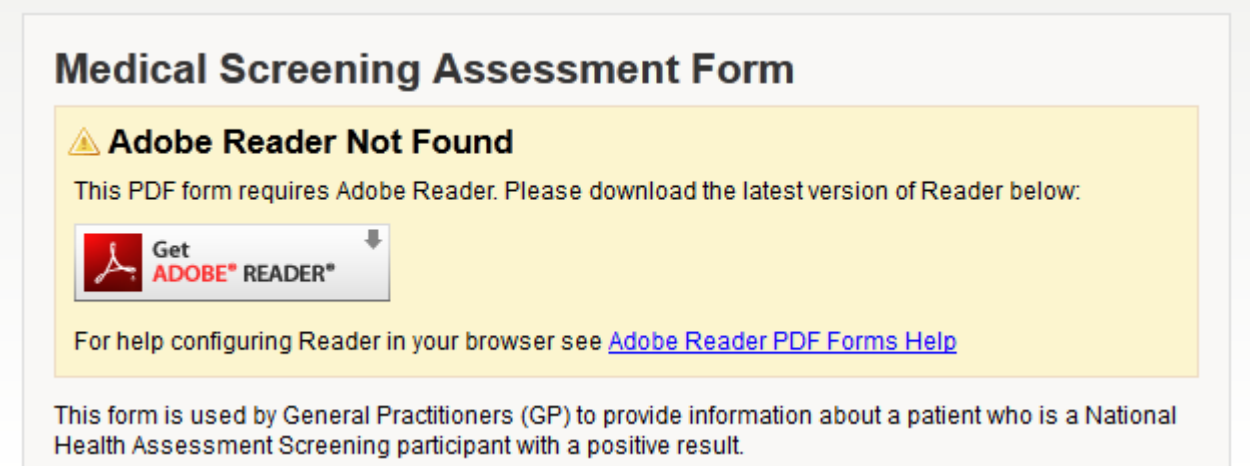
Journey Platform | Form Builder | v5.0

This article provides up to date information on how to get PDF Forms and Adobe Reader working on your computer.

If you are using an Apple iPad, Android or Windows RT device, you will need to use a PC or Mac computer to complete your form. Currently XFA PDF forms are not supported on these tablet devices.

Adobe Reader Not Found

This message is displayed when your browser could not find the Adobe Reader plug-in. This may be due to Adobe Reader not being installed on your computer or the plug-in is not configured in your browser.



To install Adobe Reader please download the free installer from Adobe at http://get.adobe.com/reader/. We always recommend you use the latest version of Adobe Reader so you have the latest security updates have the best user experience. It is recommended that you don't use any versions before Reader 9 as these older versions are not supported by Adobe, and will not be updated to address the latest security threats.

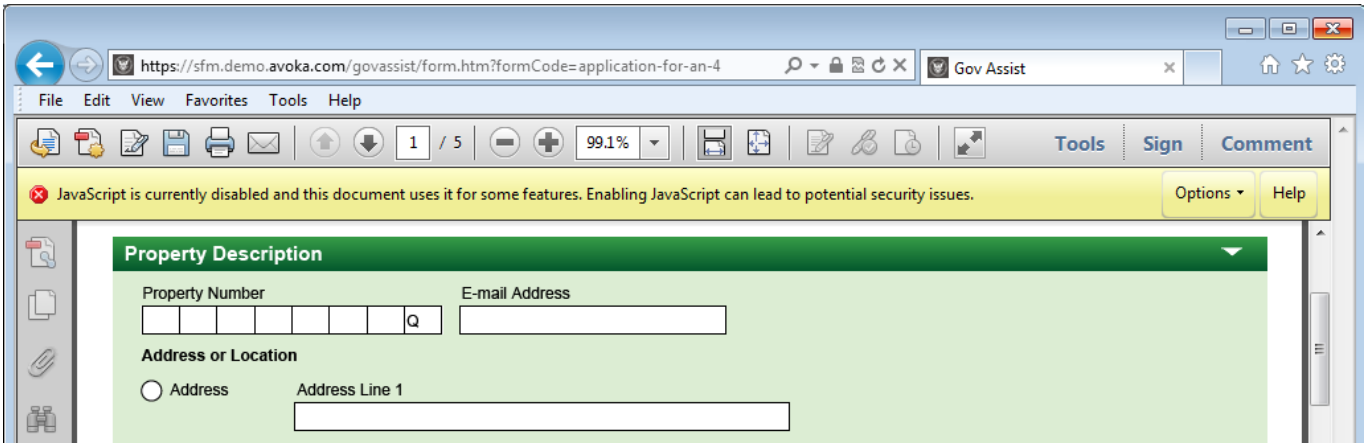
If you have Adobe Reader installed by its not available in your browser please check the following browser settings.

Adobe Reader Update Required

If this message is displayed the PDF form requires a more up to date version of Adobe Reader. Please install the latest version of Adobe Reader from: http://get.adobe.com/reader/

Adobe Reader JavaScript is Disabled

If you open a PDF form and the following message ribbon is displayed then your Adobe Reader is configured to disable JavaScript.



Most PDF forms will not work without **JavaScript**[1] and you will need to enable it to complete your form.

To enable JavaScript:

1. Click on the Options ribbon menu and select the item 'Enable JavaScript for this document always'
2. In the 'Add to Privileged Location' click on the OK button

For more information on particular browsers, see:

- Unsupported Browsers
- View PDF Documents in Chrome
- View PDF Documents in Firefox
- View PDF Documents in IE8 and IE9
- View PDF Documents in IE10
- View PDF Documents in Safari
- View PDF Documents in Windows 8

[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.

Unsupported Browsers

Journey Platform | Form Builder | v5.0

A number of browser now no longer support Adobe Reader plugins and as such don't support in browser PDF forms. The browsers are listed below:

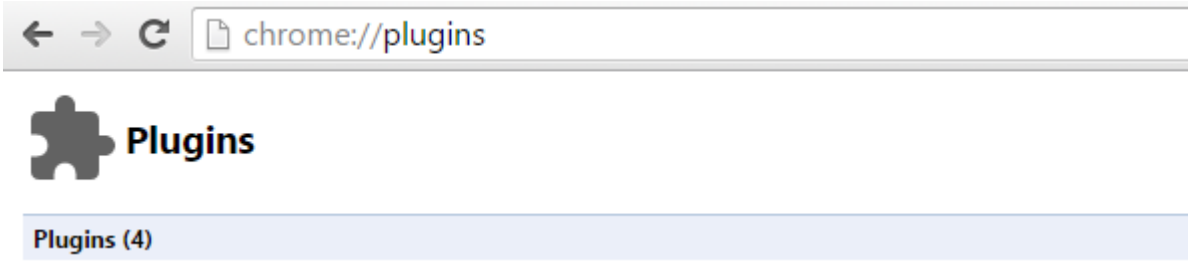
Google Chrome

Google chrome now no longer supports NPAPI (Netscape Plugin API), which Adobe Reader utilized to run from within the browser. To use PDF forms now If you can try one of the following solutions:

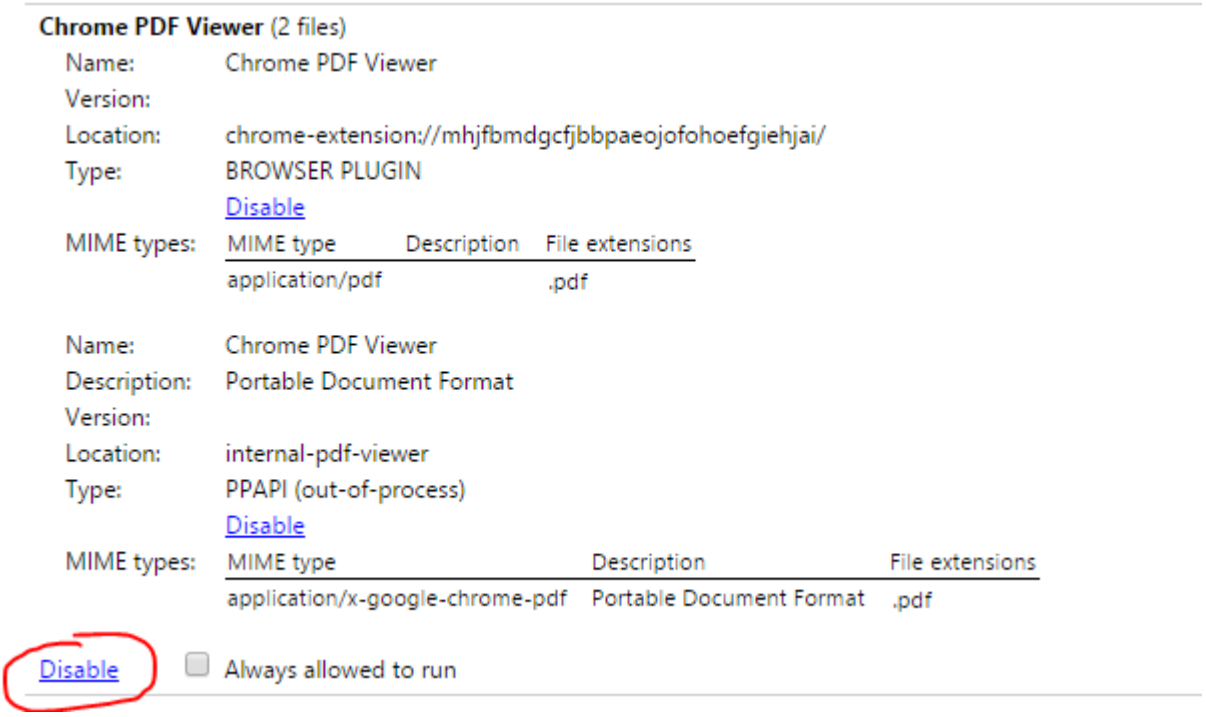
- Use a different browser, for example, Firefox or Internet Explorer.
- Disable reader, Chrome's default PDF viewer, so it will force Chrome to open the PDF in the default application for the PC, which can be set as Adobe Reader.

To disable Chrome plugins

1. Open Chrome and in the address bar type chrome://plugins



2. Click **Details** on the right hand side of the page.
3. Scroll down to Chrome PDF Viewer and click **Disable** at he bottom of the plugin information.



4. PDFs will now open to the default App on your machine.

Microsoft Edge

The new Edge browser does not have any support for ActiveX plugins, which is what Adobe Reader uses in its browser plugins. Edge will open PDFs in the default PDF viewer app for your PC. If you still have troubles with PDFs in Edge, it is recommended that you use another browser, for example, Firefox.

Next, learn how to view PDF documents in Chrome.

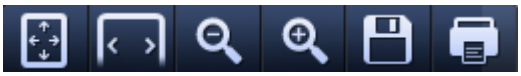
View PDF Documents in Chrome

Journey Platform | Form Builder | v5.0

As we use PDF format for our application forms, you will need to get Adobe Reader working on your computer or in your browser. This article provides instructions how to use Google Chrome, a popular browser, to open PDF Forms.

By default, Chrome uses a built-in PDF viewer to open PDF files, so when you click a PDF link on a web page, the file opens directly in your browser.

The following controls are available in the lower-right area of the browser window:

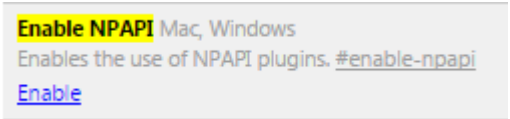


If you've disabled the built-in PDF viewer, Chrome will use Adobe Acrobat or Adobe Reader to display PDF files.

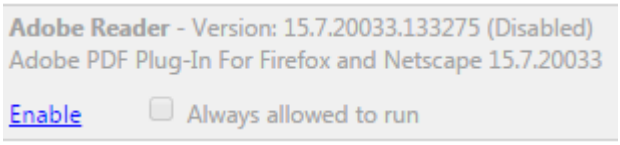
Adobe Reader runs inside Chrome as a NPAPI (Netscape Plugin API) plugin. Google is phasing out support for NPAPI plugins from the Chrome browser. Since April 2015 (Chrome 42) NPAPI plugins have been disabled by default. It is possible to re-enable them via an override however from September 2015 (Chrome 45) this override will be removed and NPAPI support will be permanently removed from Chrome.

To re-enable NPAPI:

1. Enter: **`chrome://flags/#enable-npapi`** in the browser's address bar.
2. Look for the following Enable NPAPI entry.



3. Click **Enable**.
4. Restart Chrome and then enter **`chrome://plugins/`** in the browser's address bar.
5. Look for the following Adobe Reader entry.



6. Click **Enable**. This makes Adobe Reader the default PDF viewer inside your Chrome and consequently disables the Chrome PDF Viewer.

> Alternatively, you can set any of the plugin Enterprise policies, such as **EnabledPlugins** or **PluginsAllowedForUrls**, which temporarily re-enable NPAPI support.

After NPAPI support is removed from Chrome, if you need to use Adobe Reader inside a browser, you must use another browser until a replacement plugin becomes available for Chrome.

If you have enabled both NPAPI support and either the Adobe Acrobat or Adobe Reader plugin, but you are still experiencing problems please follow the following steps.

First check if the issue is specific to Chrome. Try opening PDFs in another browser on your computer, such as Firefox. If you experience similar problems viewing PDFs in another browser, the issue is likely not related to Chrome, so you need to visit the Adobe support site for additional help.

Chrome Troubleshooting

If you are still having issues viewing PDFs and it is specific to Chrome, try the following solutions in the order that they arere listed:

Check for updates for Chrome, Adobe Reader and Acrobat

1. Checking for Chrome updates here.
2. Check for Adobe Reader or Acrobat software updates here.

Check your Adobe Reader and Acrobat settings

Follow the instructions below to confirm that either Adobe Reader or Adobe Acrobat is configured to display PDFs in the browser.

> Only one of the programs should be set to preview PDFs in the browser, not both.

1. In either Adobe Acrobat or Reader, select **Edit > Preferences**.
2. Click **Internet**.
3. Select the **Preview PDF in browser** or **Display PDF in browser** checkbox. If the checkbox is already selected, clear it, click **OK**, then go **Preferences** and select it again.
4. Click **OK**.

5. Follow the same steps in the other program, but make sure that the **Preview PDF in browser** checkbox is not selected.

Repair your Adobe Reader or Adobe Acrobat installation

1. In Adobe Reader, select **Help > Repair Adobe Reader Installation**.
2. In Adobe Acrobat, select **Help > Repair Acrobat Installation**.

Uninstall either Adobe Reader or Adobe Acrobat installation

If you have both Adobe Reader and Adobe Acrobat installed, they may conflict with each other. Try uninstalling one of the applications.

Windows

1. Use the **Add or Remove Programs** function in your Windows Control Panel to remove either Adobe Reader and Acrobat from your computer.
2. Restart your computer after you have completed uninstalling the program.

Mac

1. Drag either Adobe Reader or Acrobat from the Applications folder to the **Trash**.
2. Remove AdobePDFViewer.plugin from the **Library > Internet Plug-Ins** folder.

Uninstall and Reinstall Adobe Reader or Acrobat

1. Follow the steps from the section above to uninstall all versions of these programs.
2. Visit http://www.adobe.com/downloads/ to reinstall the latest version of Adobe Acrobat or Reader.

Next, learn how to view PDF documents in Firefox.

View PDF Documents in Firefox

Journey Platform | Form Builder | v5.0

This article provides up to date information on how to get PDF Forms and Adobe Reader working on your computer.



Using the built-in PDF viewer

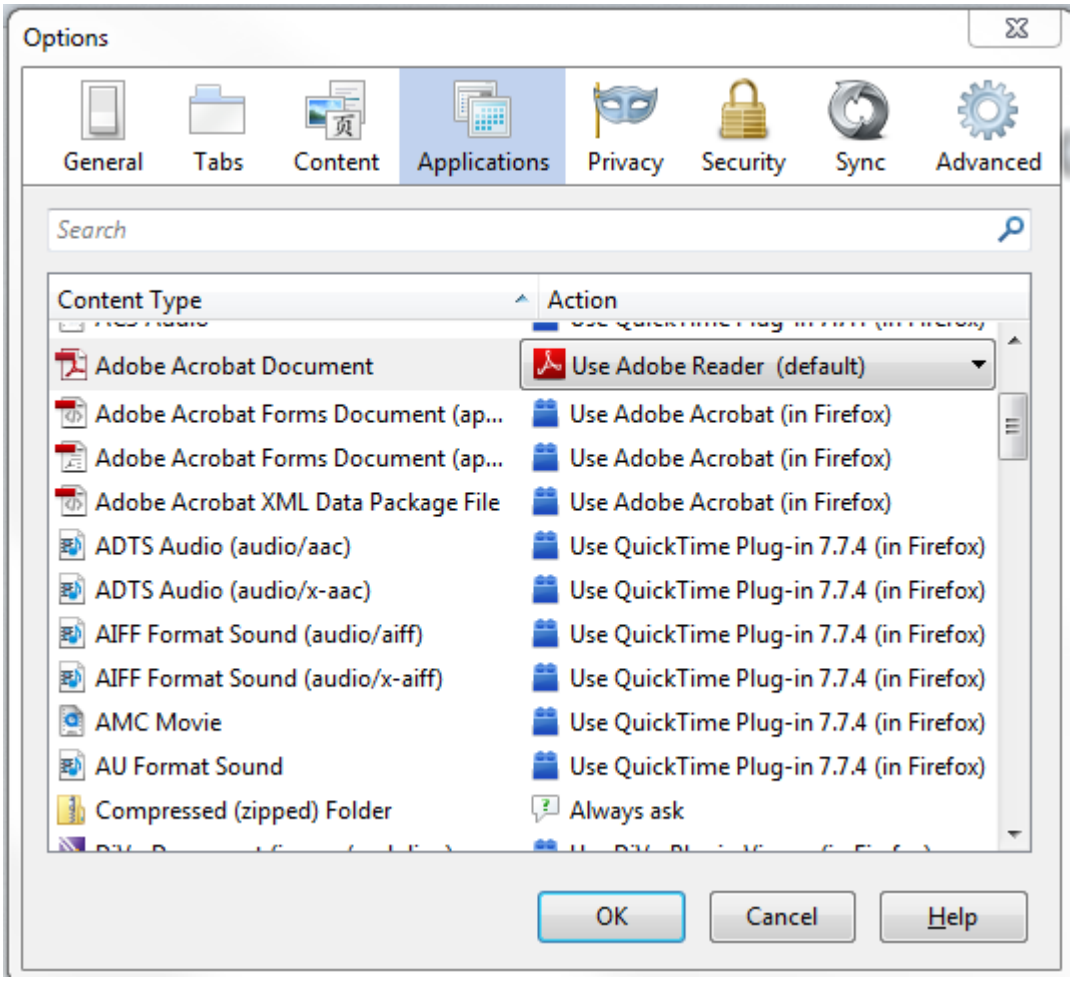
Firefox now includes a built-in PDF viewer that allows you to view almost all PDF files found on the web without a plugin. The built-in PDF viewer is enabled by default.

Using a PDF reader plugin

To check what your Firefox browser is set to, go to the **Tools Menu > Options** and click on the **Applications** tab

To change from using the built-in PDF viewer to your PDF reader plugin:

1. Click **Firefox** at the top of the Firefox window and select **Options.**
2. Select the **Applications** panel.
3. Click **Find Portable Document Format (PDF)**.
4. Select a desired PDF reader from the dropdown list, for example, **Use Adobe Reader (default)** in the **Action** column .



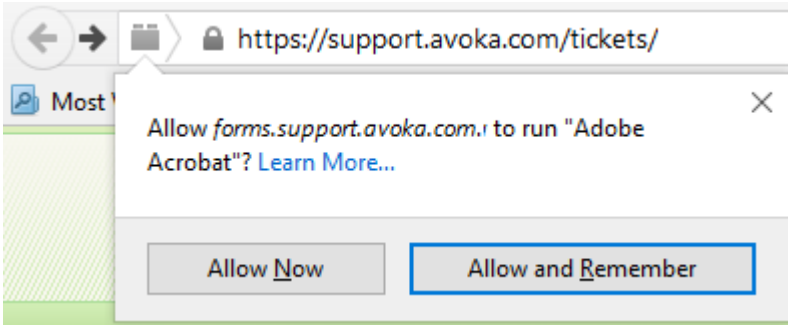
5. Click **OK**.

The Firefox Software Update feature does not update installed plugins. For information about security and stability issues affecting Adobe Reader, see the Adobe security bulletins and advisories documentation.

To check if you have the latest versions of all your plugins, see plugin check documentation.

If plugin is not working

Firefox can sometimes block plugins from working on certain web pages. If the PDF is not rendering check the plugin is allowed, click the plugin icon in the browser address bar and then click **Allow and Remember**.



Testing the Adobe Reader Plugin

To test whether you are able to open PDF files, see Mozilla Privacy Policy PDF documentation.

If PDF files are downloaded instead of displayed, try these solutions

To reset download actions:

1. Open your profile folder.
2. Click **Firefox** at the top of the Firefox window.
3. Go to the **Help** menu and select **Troubleshooting Information**.
4. Under the **Application Basics** section, click **Show Folder**. A window with your profile files will open.

> If you are unable to open or use Firefox, follow the instructions in Finding your profile without opening Firefox.

Next, learn how to view PDF documents in IE8 and IE9.

View PDF Documents in IE 8 and IE 9

Journey Platform | Form Builder | v5.0

This article provides information on how to get PDF Forms and Adobe Reader working on your computer.



Internet Explorer is supported for use with Acrobat and Adobe Reader version 7.0.8 and higher. For versions of Acrobat earlier than 7.08, most functionality will work, but there may be issues with forms workflows.

If you run into any sort of issue, then you should upgrade to the latest version of Acrobat Reader (and if possible your browser version).

Internet Explorer 8 makes it more visible to the user which add-ons are enabled and disabled. This is done from the tools menu by selecting Manage Add Ons.

Using an Add-on to view PDF documents within an IE8 browser window

When you install Acrobat or Adobe Reader on a system that includes a browser, you automatically configure the browser to open PDF files within the browser window. Acrobat and Adobe Reader install ActiveX plug-in files that allow you to use Internet Explorer or AOL to open PDF files.

If either application is installed, but PDF files don't open in the browser window, then work through the following steps to configure the browser:

1. Make sure that the system meets the requirements for the Acrobat product you use.

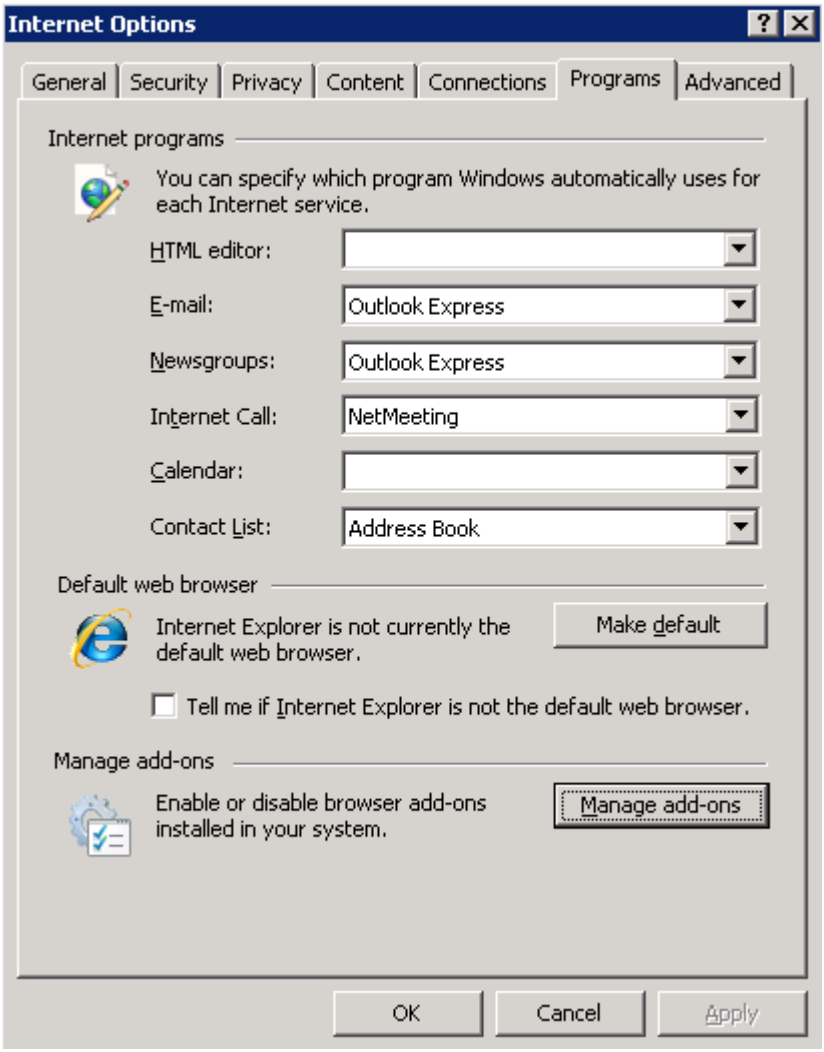
Acrobat may not run correctly if the system doesn't meet the following requirements:

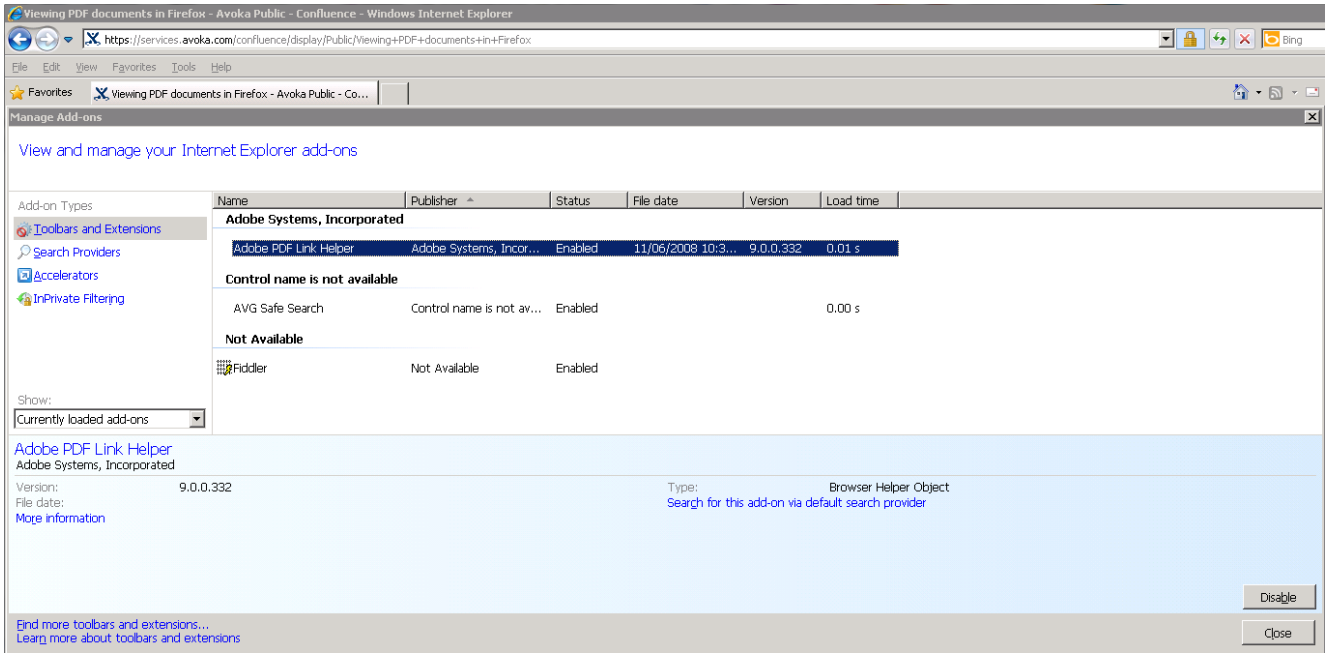
- Intel Pentium III processor or equivalent
- Microsoft Windows 2000 with Service Pack 4 or Windows XP Professional, Home Edition, or Tablet PC Edition with Service Pack 2
- Microsoft Internet Explorer 6.0
- 256MB of RAM (512MB recommended)
- 860MB of available hard-disk space; cache for optional installation files (recommended) requires an additional 460MB of available hard-disk space.
- 1,024x768 screen resolution

2. Check your IE add-on

Internet Explorer 8 makes it more visible to the user which add-ons are enabled and disabled. This is done from the tools menu by selecting Manage Add Ons.

1. Click **Start**.
2. Type **Internet Explorer** and select **Internet Explorer** from the list.
3. Click **Tools** and select **Manage** add-ons.
4. Click **Toolbars and Extensions** under Add-on Types.
5. Select one of the following views of your add-ons under **Show**:
   - All add-ons - to display a complete list of the add-ons on your computer.
   - Currently loaded add-ons - to display only those add-ons that were needed for the current webpage or a recently viewed webpage.
   - Run without permission - to display add-ons that were preapproved by Microsoft, your computer manufacturer, or a service provider.


Internet Options
General
Security
Privacy
Content
Connections
Programs
Advanced
Internet programs
You can specify which program Windows automatically uses for each Internet service.
HTML editor:
E-mail:
Outlook Express
Newsgroups:
Outlook Express
Internet Call:
NetMeeting
Calendar:
Contact List:
Address Book
Default web browser
Internet Explorer is not currently the default web browser.
Make default
Tell me if Internet Explorer is not the default web browser.
Manage add-ons
Enable or disable browser add-ons installed in your system.
Manage add-ons
OK
Cancel
Apply


Viewing PDF documents in Firefox - Avoka Public - Confluence - Windows Internet Explorer
Manage Add-ons
View and manage your Internet Explorer add-ons
Add-on Types
Toolbars and Extensions
Search Providers
Accelerators
InPrivate Filtering
Name
Publisher
Status
File date
Version
Load time
Adobe Systems, Incorporated
Adobe PDF Link Helper
Adobe Systems, Incor...
Enabled
11/06/2008 10:3...
9.0.0.332
0.01 s
Control name is not available
AVG Safe Search
Control name is not av...
Enabled
0.00 s
Not Available
Fiddler
Not Available
Enabled
Show:
Currently loaded add-ons
Adobe PDF Link Helper
Adobe Systems, Incorporated
Version: 9.0.0.332
File date:
More information
Type: Browser Helper Object
Search for this add-on via default search provider
Disable
Find more toolbars and extensions...
Learn more about toolbars and extensions
Close

6. Click **Close**.

3. Make sure that the ActiveX folder contains files required for web browsing.

To open PDF files in a browser window, the ActiveX folder must contain the following files: AcroPDF.dll, AcroIEHelper.dll, and pdfshell.dll.

The ActiveX controls should be located in: Program Files\Common Files\Adobe\Acrobat\ActiveX

If any of these files are missing, then repair Acrobat or Adobe Reader by choosing Help > Repair Acrobat Installation, or reinstalling the application.

4. Use a supported version of the browser.

Make sure that you use a version of your web browser that Acrobat and Adobe Reader support.

Supported browsers for Adobe Acrobat 8, Adobe Reader 8 are Internet Explorer 6.0, 6.0 SP1, Internet Explorer 7.0, Firefox 1.5 or later, Mozilla 1.7, and AOL 9.

5. How to turn on and off add-ons

1. Click **Start**.
2. Type **Internet Explorer** and select **Internet Explorer** from the list.
3. Click **Tools** and select **Manage** add-ons.
4. Click **All add-ons** under **Show**.
5. Click an add-on, and then do one of the following:
   - Click **Enable** - to make the add-on available for use in the browser.
   - Click **Disable** - to turn the add-on off so it can't be used in the browser.
6. Repeat step 4 for each add-on that you want to turn on or off.
7. Click **Close**.

Before you turn off a browser add-on, keep in mind that some webpages, or Internet Explorer itself, might not display properly if an add-on is turned off. We recommend that you only turn off an add-on if it repeatedly causes Internet Explorer to close.

6. To delete Active X controls

You can only delete ActiveX controls that you have downloaded and installed. You can't delete ActiveX controls that were pre-installed or add-ons of any kind, but you can turn them off.

1. Click **Start**.
2. Type **Internet Explorer** and select **Internet Explorer** from the list.
3. Click **Tools** and select **Manage** add-ons.

4. Click **Downloaded controls** under **Show** to display all ActiveX controls.
5. Click the ActiveX control you want to delete, and then click **More information**.
6. Click **Remove**. Administrator permission required If you're prompted for an administrator password or confirmation, type the password or provide confirmation.
7. Repeat these steps for every control you want to delete.
8. Click **Close**.

For more information, see http://windows.microsoft.com/is-IS/windows7/How-to-manage-add-ons-in-Internet-Explorer-9.

Next, learn how to view PDF documents in IE10.

View PDF Documents in IE 10

Journey Platform | Form Builder | v5.0

This article provides information on how to get PDF Forms and Adobe Reader working on your computer.



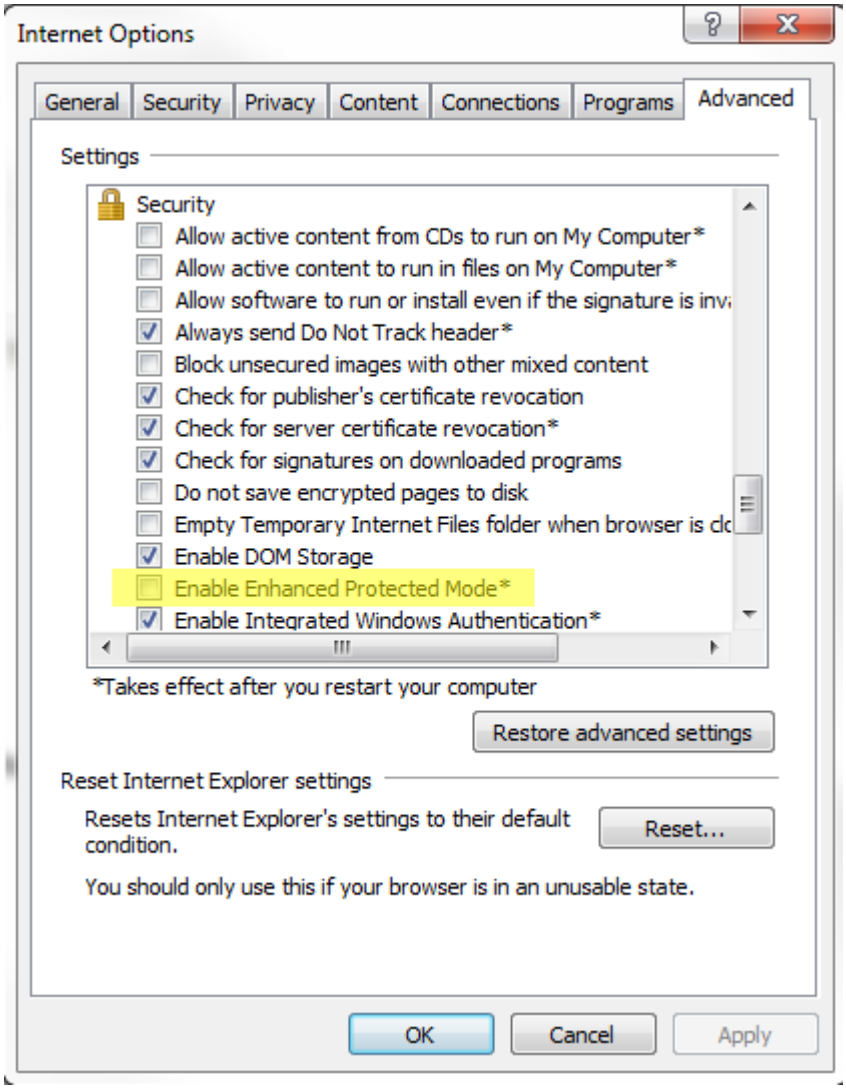
Using the built-in PDF viewer

Internet Explorer includes a built-in PDF viewer that allows you to view almost all PDF files found on the web without a plugin.

If your PDFs are opening in a native application such as Acrobat Reader, check the IE settings as described below.

Check your Browser Settings

1. Open the IE browser.
2. Select **Tools Menu > Internet Options.**
3. Click the **Advanced Tab.**
4. Scroll to the **Security** section and clear the **Enable Enhanced Protected** checkbox
5. Restart your computer and PDFs should now open in your IE browser.



6. Click OK.

Still having Issues? Try these steps

Method 1: Disable Add-Ons

1. Click **Tools** and select **Manage add-ons**.
2. Disable the add-ons by clicking them one at a time to highlight them and then click on disable button.
3. Re-enable the add-ons one by one and check with which Add-on you get this error message.
4. Disable the Add-on causing the problem.

Method 2: Reset security settings for Internet Explorer

If you configure security settings to be too restrictive, you may prevent Internet Explorer from displaying certain Web sites. To determine whether an issue is caused by overly restrictive security settings, revert to default security settings. To do this, follow these steps:

1. Open Internet Explorer.
2. Click **Tools** and select **Internet Options**.
3. Click the **Security** tab.
4. Click **Reset all zones to default level**
5. Click **OK**.

Method 3: Run the Internet explorer performance troubleshooter and check if it helps.

1. Press the 'Windows + W' key on the keyboard.
2. Type troubleshooter in the search box and then press enter.
3. Select **Troubleshooting**.
4. Click on view all and select Internet explorer performance.
5. Click next and follow the on-screen instructions.

Method 4: Reset Internet Explorer 10

1. Drag the mouse to the bottom right corner and click on search.
2. Click **Settings** and type **Internet Options**.
3. Click **Internet Options** and select **Advanced** tab.
4. Click **Reset.**
5. Click **OK**.

Please note that running Reset Internet Explorer Settings will reset all user-defined settings including those set by installed extensions, toolbars and other add-ons to IE Defaults. This includes all Security, Privacy and Zone settings. Also this will clear browsing history, delete all temporary Internet Files, cookies, form data and especially all stored passwords.

Method 5: uninstall the Adobe Acrobat Reader and install the latest version.

Download the latest version at http://get.adobe.com/reader/otherversions/

If you have multiple Adobe Readers installed, uninstall them all then install the latest version. Adobe doesn't support multiple versions of Reader or Acrobat on the same computer. Multiple versions simultaneously installed can lead to software conflicts and errors. Uninstall all versions of Reader or Acrobat and then reinstall only one version of the product.

Next, learn how to view PDF documents in Safari.

View PDF Documents in Safari

Journey Platform | Form Builder | v5.0

This article provides up to date information on how to get PDF Forms and Adobe Reader working on your computer.



The Adobe Reader and Acrobat stand-alone applications are unaffected and Safari 5.1 renders PDF documents natively.

However, the Adobe Reader and Acrobat browser plug-ins before 10.1.3 don't function as expected in LiveCycle and Acrobat workflows that require either plug-in to render PDF documents in Safari 5.1.

Adobe Reader for XFA PDF forms is not available on iPad or iPhone devices.

System requirements

Acrobat X and Adobe Reader X support viewing PDF files inside the Safari browser on Mac OS X.

For more information about system requirements, click these links for your product: Acrobat X Pro, Acrobat X Standard, Adobe Reader X.

Adobe PDFViewer for Mac OS X requires the following system requirements:

- Safari 4 for Mac OS X 10.5.8 and Mac OS X 10.6.4
- Safari 5 for Mac OS 10.6.4
- Acrobat and Reader via AdobePDFViewer Safari plug-in are 32/64-bit (64-bit available in Safari 10.6)

If you don't have Acrobat or Reader, or haven't installed the PDFViewer plug-in, then Safari shows PDF files using the Mac OS X native PDF support. Support for many Adobe PDF workflows, such as the collaboration jobs or many forms and security features, is unavailable.

Configuring Acrobat to display PDF files in Safari

The AdobePDFViewer plug-in is used to display PDF files in Safari using Acrobat and Reader. This plug-in is installed as part of the Acrobat X or Reader X installation.

The location of this plug-in on a Macintosh is **/Library/Internet Plug-ins/AdobePDFViewer.plugin**.

Troubleshooting issues in Safari

Options include the following:

- Continue to use Safari 5.0.x and Mac OS X 10.6.
- Upgrade to Adobe Reader or Acrobat version 10.1.3 or later.
- Save the PDF file from Safari to the computer, and then open it directly in Adobe Reader or Acrobat.

For more information, see http://helpx.adobe.com/acrobat/kb/troubleshoot-safari-plug-acrobat-x.html.

Known Issues

Reader's Preferences > Internet > Display PDF in Browser is always dimmed

- To disable Safari integration, delete the AdobePDFViewer.plugin from **/Library/Internet Plug-ins**.
- To reenable Safari integration, rerun the installer over the current installation. Start the Acrobat/Reader X installer from the original media and follow the prompts to complete the installation.

Uninstalling an older version of Acrobat or Reader after Acrobat X or Reader X has been installed removes the AdobePDFViewer plug-in

If Acrobat X or Reader X is installed alongside an older version of Acrobat or Reader, removing the older version removes the AdobePDFViewer plug-in.

To re-enable Safari integration, rerun the installer over the current installation. Start the Acrobat/Reader X installer from the original media, and then follow the onscreen prompts to complete the installation.

Older versions of the AdobePDFViewer plugin can overwrite newer versions of the plug-in

Having multiple versions of Acrobat or Reader installed on the same system can cause plug-in conflicts. For example, suppose you have both Acrobat 9 and Acrobat X on your system and Acrobat 9 gets updated. In this case, the Acrobat 9 version of the AdobePDFViewer plug-in can overwrite the Acrobat X version of the plug-in. To correct the issue, follow these steps:

- Remove the previous version of Acrobat or Reader.
- Reinstall Acrobat X or Reader X from the original media.

Next, learn how to view PDF documents in Windows 8.

View PDF Documents in Windows 8

Journey Platform | Form Builder | v5.0

This article provides information on how to get PDF Forms and Adobe Reader working on your computer.



To make PDF forms work in Windows 8 you need to make Adobe Reader the default PDF Reader, as the Reader provided by Microsoft Windows does not support PDF forms.

Once you have Adobe Reader installed and configure please access forms with Internet Explorer in the Windows Desktop UI mode. To switch to Desktop mode press the keys `Windows + D`.

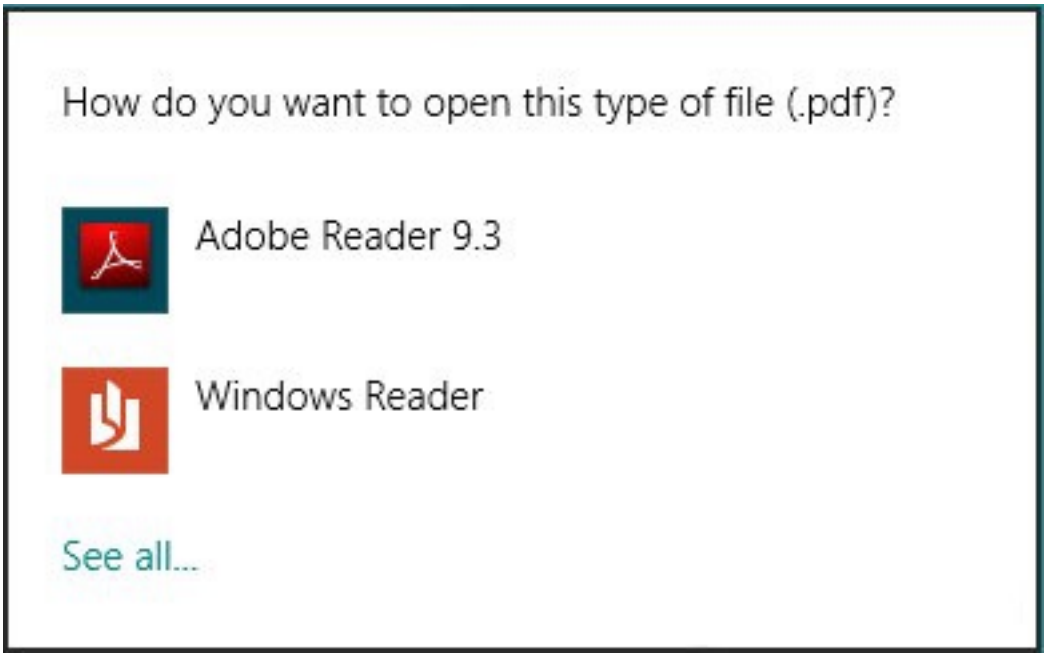
The Adobe Reader plugin is not compatible with Internet Explorer running in start screen or Metro UI mode.

To make Adobe Reader the default PDF Reader please follow the instructions below:

Using Acrobat Reader to be the Default PDF viewer

Windows 8 doesn't associate Acrobat Reader as default application for PDF files after installing it. If you want to open all your PDF files using Acrobat Reader by default, follow the following steps:

1. Right click on any PDF file and click **Properties**.
2. Click **Change** against the **Open With** option to open a new window which shows valid applications associated with PDF files.



3. Click **Adobe Acrobat**.

Once you have Adobe Reader installed and configure please access forms with Internet Explorer in the Windows Desktop UI mode. To switch to Desktop mode press the keys `Windows + D`. The Adobe Reader plugin is not compatible with Internet Explorer running in start screen or Metro UI mode.

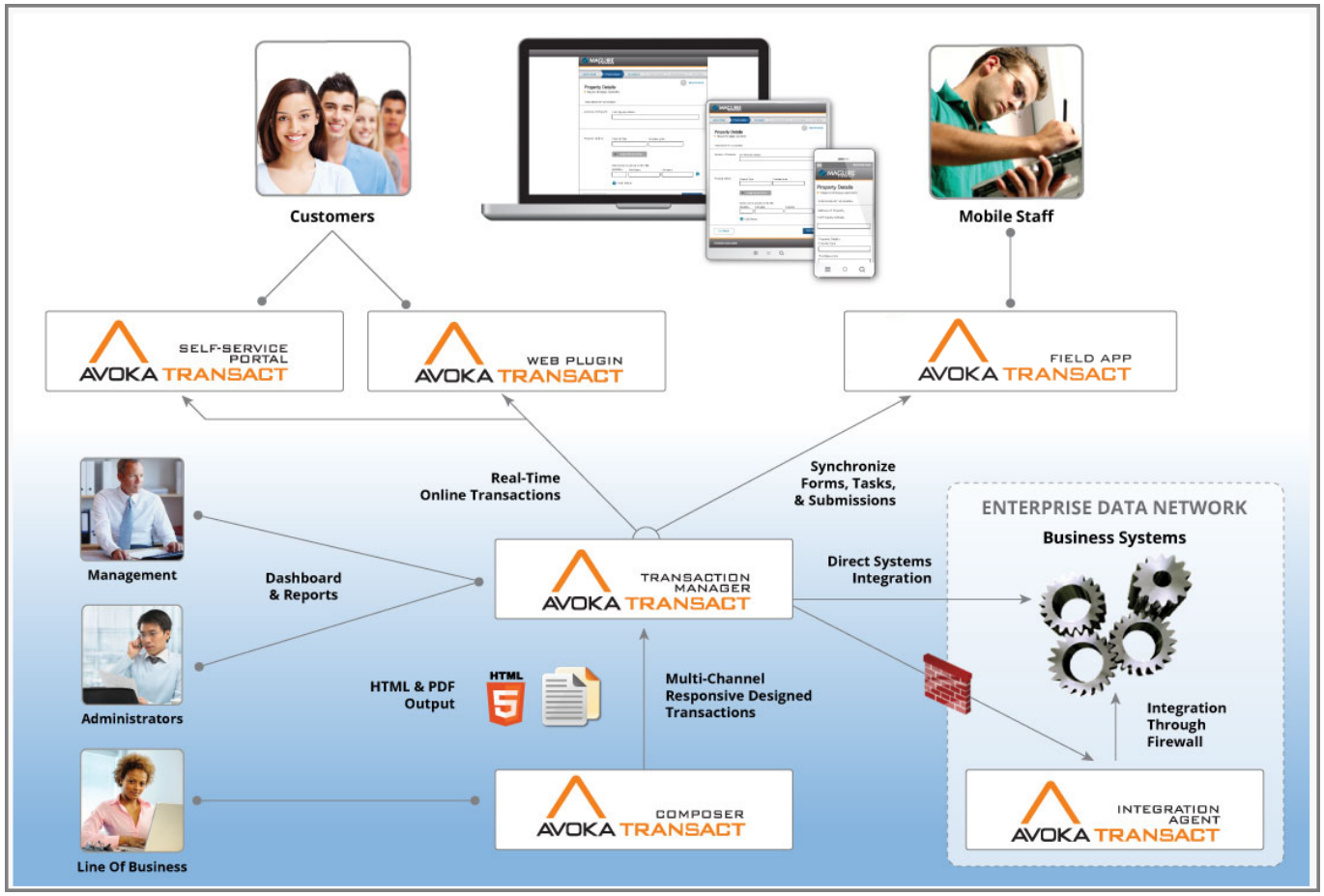
Next, learn about unsupported browsers.

TransactField App Overview

TransactField App | Deprecated in 21.11

TransactField App works with the Temenos Journey Platform and enables mobile workers to do business whenever, and wherever, they require. It offers robust mobile capabilities and improves mobile workforce productivity by optimizing on-site data entry and making it easy for organizations to deliver quality service and measurable results, as well as speed up billing. Additionally, organizations can link the TransactField App to ERP[1] and CRM[2] systems to deliver work tickets, inspection jobs, and report forms to a mobile workforce – anytime, anywhere.

TransactField App runs on Apple and Android devices, Microsoft Windows desktop and tablet and synchronizes them with mobile data capture applications that are built using the Temenos Journey Platform. Interaction between TransactField App and the Journey platform is illustrated with the diagram below.



[1]Enterprise resource planning (ERP) is the integrated management of main business processes, often in real-time and mediated by software and technology.

[2]Customer relationship management (CRM) is an approach to manage a company's interaction with current and potential customers. It uses data analysis about customers' history with a company to improve business relationships with customers, specifically focusing on customer retention and ultimately driving sales growth.

The TransactField App delivers the following capabilities to mobile workers:

- Allows mobile workers to take their forms with them out in the field and work either online or offline.
- Synchronizes SmartForms automatically to the device.
- Stores submitted form data temporarily on the device, completing the submissions when the user comes online.
- Enables forms to make use of device capabilities, including camera, geolocation (GPS), and scribble signature.
- Enforces security, including password protection and encrypted data on the device and over the wire.
- Provides full control over which forms go to which users through permissions and roles.
- Synchronizes to all apps when a new form or an updated form is deployed.

The following form transaction features supported by TransactField App and Self Service Portal. This table should help guide business on where to deploy their form transaction applications.

Form Transaction Features	TransactField App	Web Portal
Offline Support	Y	N
Authenticated Access	Y	N
Anonymous Access	N	Y
Authenticated Save	Y	Y
Anonymous Save	Y	Y
Form Tasks	Y	Y
Form Group Access Control	Y	Y
Landing Page	N	Y
Terms & Conditions	N	Y
Attachments	Y	Y
Box Integration	Y	N
Card Payments	N	Y
eSignature and Wet Signatures	N	Y
Scribble Pad Signature	Y	Y
Email Verification	Y	Y
Prefill Service	Y	Y
Dynamic Data Service	Y (Available only when authenticated)	Y (Available only when authenticated)
PDF Form (XFA)	N	Y

The Self Service Portal does not support working offline or working in environments with very limited internet connectivity.

Next, learn how to get started with TransactField App.

Getting Started with TransactField App

TransactField App | Deprecated in 21.11

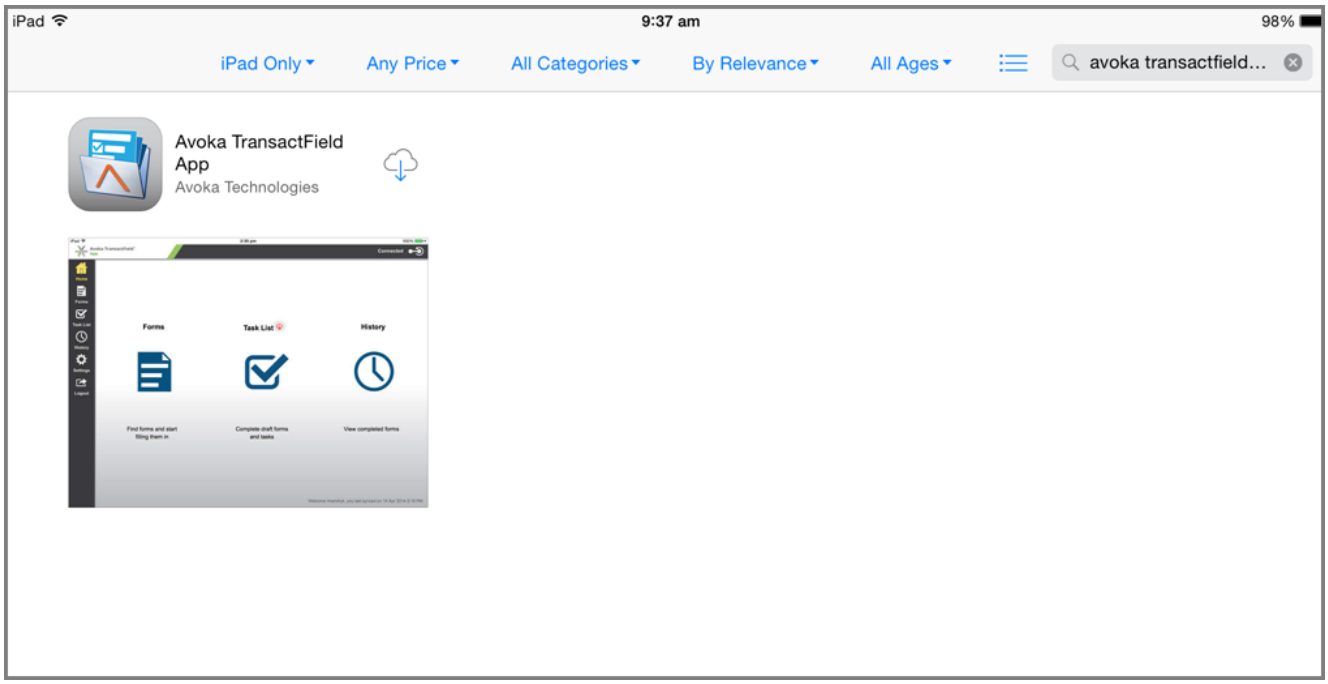
To use TransactField App, you need to install it on your device or a computer.

Installation

You can download and install TransactField App from the Apple App Store, Google Play Store, and via standalone desktop version that runs on Windows 10.

To install TransactField App:

1. Locate the TransactField App application on the appropriate online application store.



2. Click **TransactField App** for more details, reviews and related information.
3. Click **Install ("Free")** to install the application.
4. Click **TransactField App** to open the application.

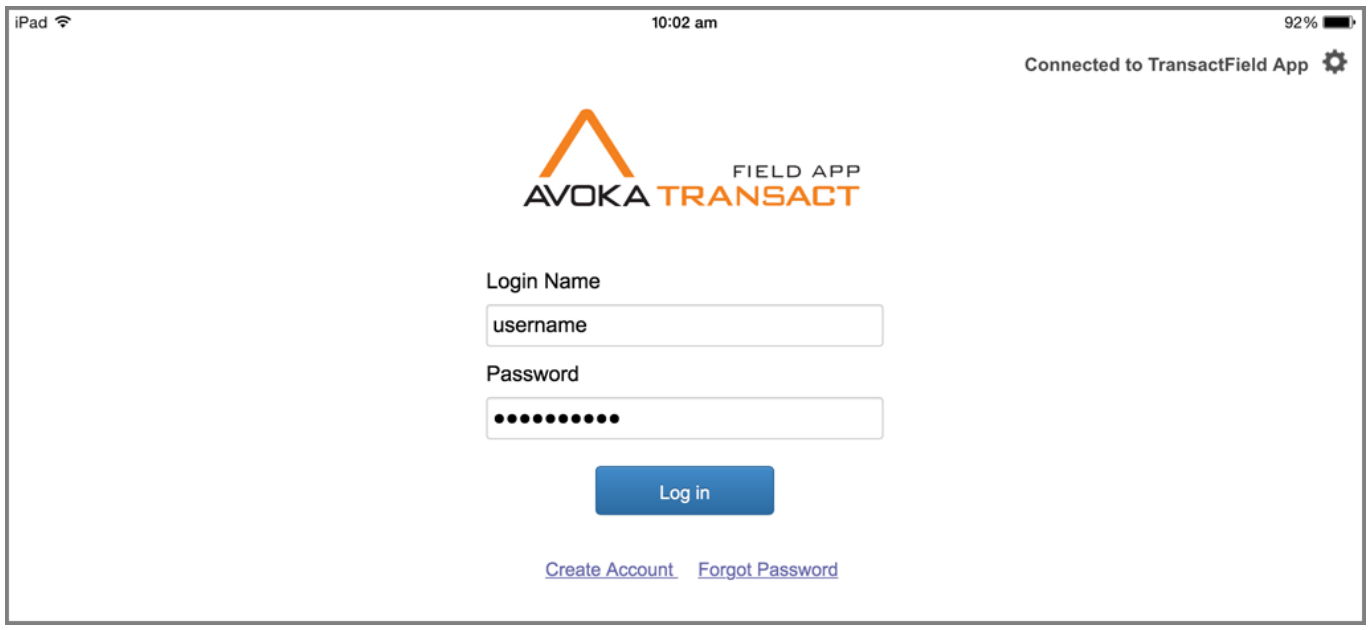


5. Configure the application.

Don't uninstall TransactField App from the device, because it will delete all local not synchronized data. If you need to uninstall it, make sure you don't have any outstanding not synchronized work on your home screen. You should update TransactField App from the app store instead.

Configuration

When you first launch TransactField App, you will be presented with the login screen as shown below.

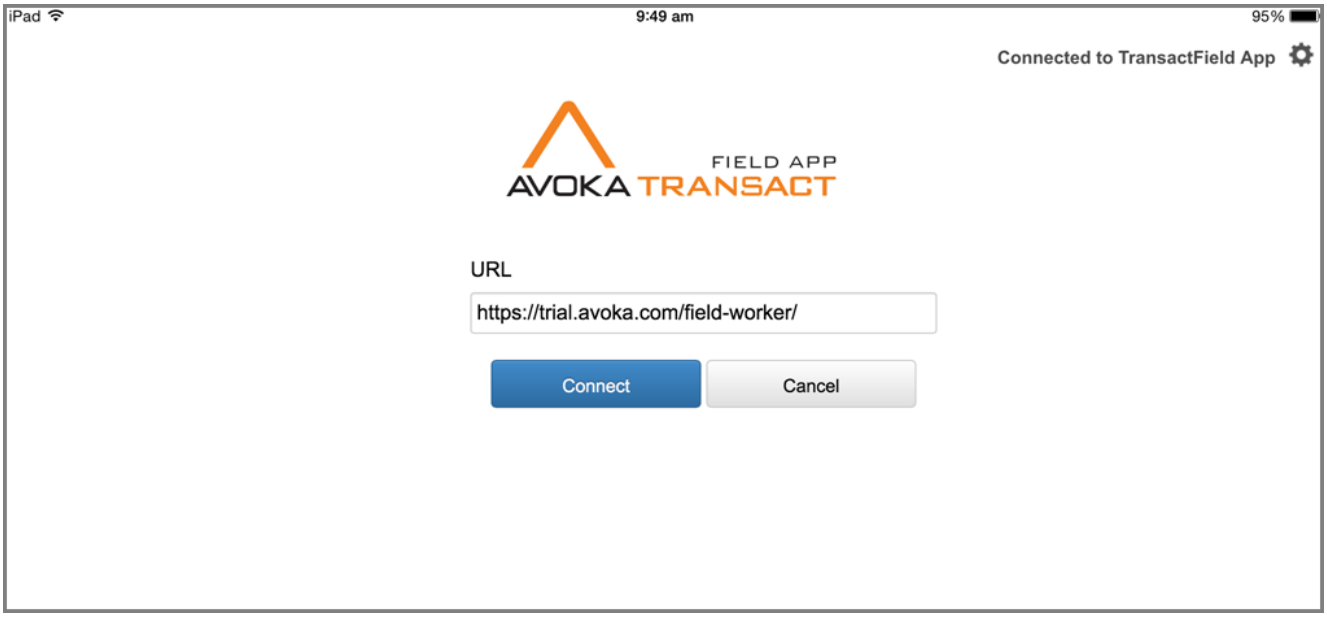


If you can't see the **Create Account** button, refer to System Settings.

Each organization using TransactField App has their own Journey Manager server, which stores all the forms and submission and other relevant data. When you first install TransactField App, it is configured to use a standard Trial Server. You will need to configure it to point to your organization's Manager server. Your system administrator will send you information about which server you need to use.

To configure which Manager server to connect to:

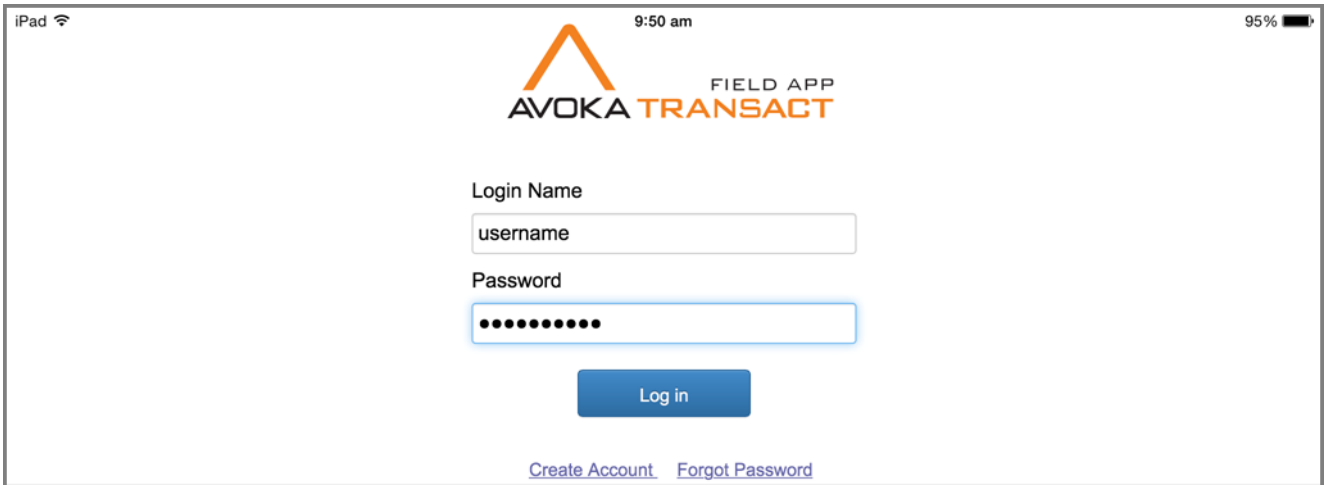
1. Click **Connected to**.



2. Enter the URL of your server.

The default for users evaluating the app on trial is https://trial.avoka.com/field-worker/.

3. Click **Connect**.
4. Provide your Login Name and Password and click **Log in** or click **Create Account**.



You will only need to configure the server once - TransactField App will remember the configuration.

System Administrators can choose from a range of languages.

Don't purge the device or change portal URL - this will delete all data from the device so you can not recover all unsynchronized data.

For more information on TransactField App configuration, see configure TransactField App.

Log in

After launching TransactField App, you need to log into the server. There are several different ways to obtain your user-name and password.

Self-registration

If security settings are configured, you may be allowed to create your own credentials in the system, as described below:

1. Click **Create Account**.
2. Enter your username and password.

Credentials provided to you

Your system administrator have created an account for you already and have l provided a username and password to you:

1. Enter your username and password.
2. Click **Log in**.

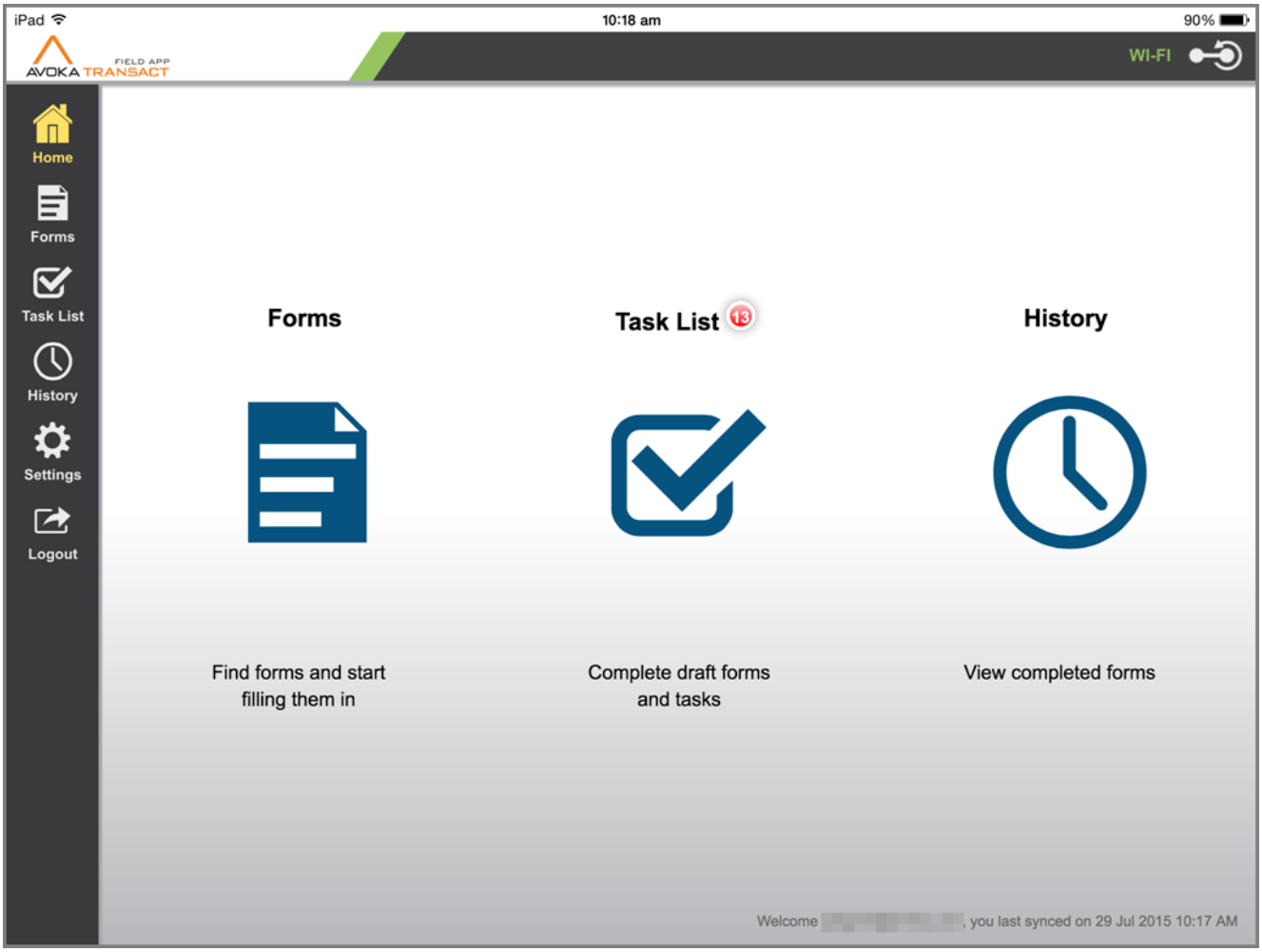
Ensure that you are online the first time you log in, as TransactField App caches your credentials so you can work offline disconnected from the Manager server.

Use your existing organizational credentials

In some cases, you can be configured to reuse your existing organization credentials:

1. Enter the usual username and password that you use when logging into your organizational network or a computer.

2. Click **Log in** to access the TransactField App home screen. as shown below.



You will be asked to login again after closing TransactField App and reopening it, or after it has timed out.

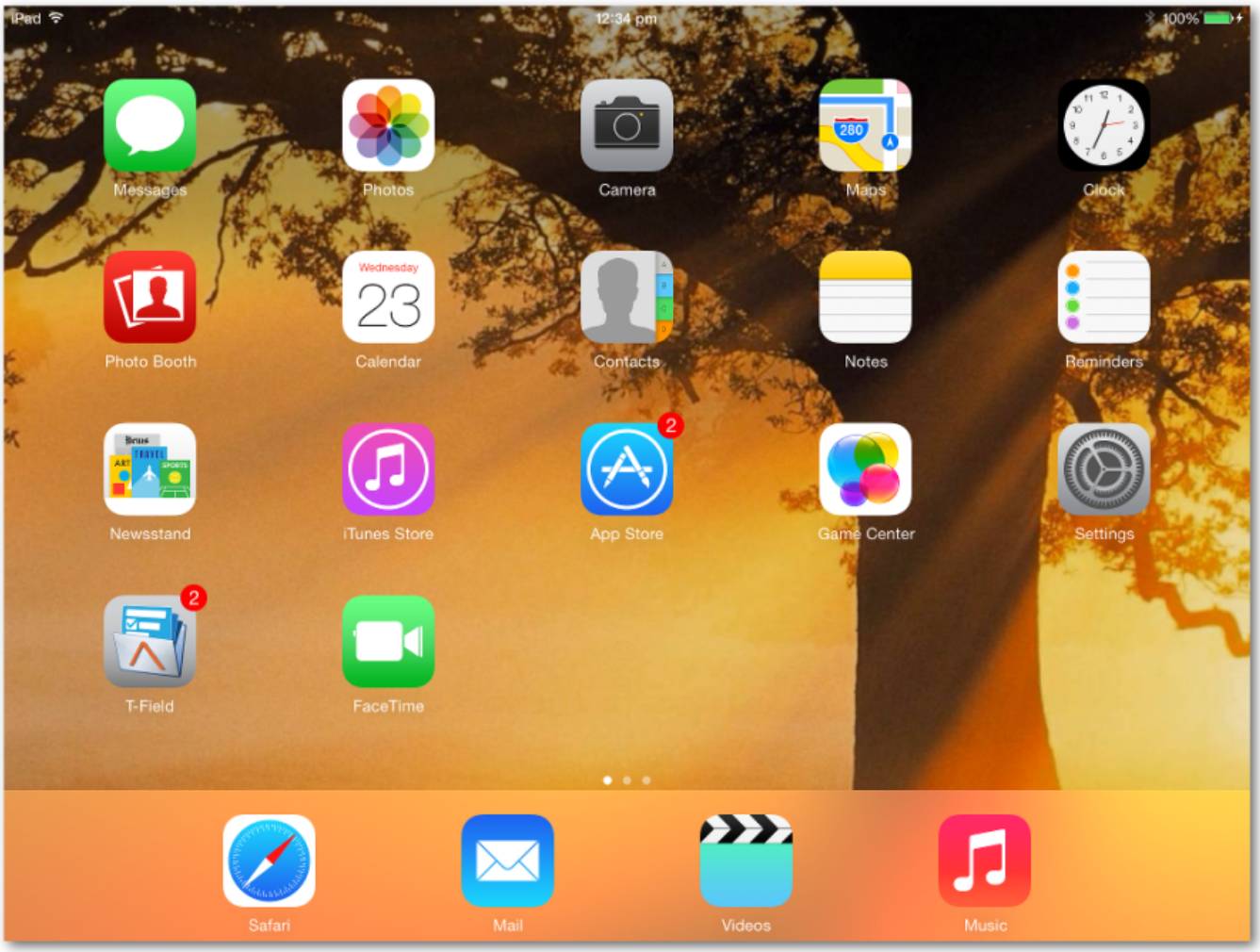
Next, learn how to use TransactField App.

How to Use TransactField App

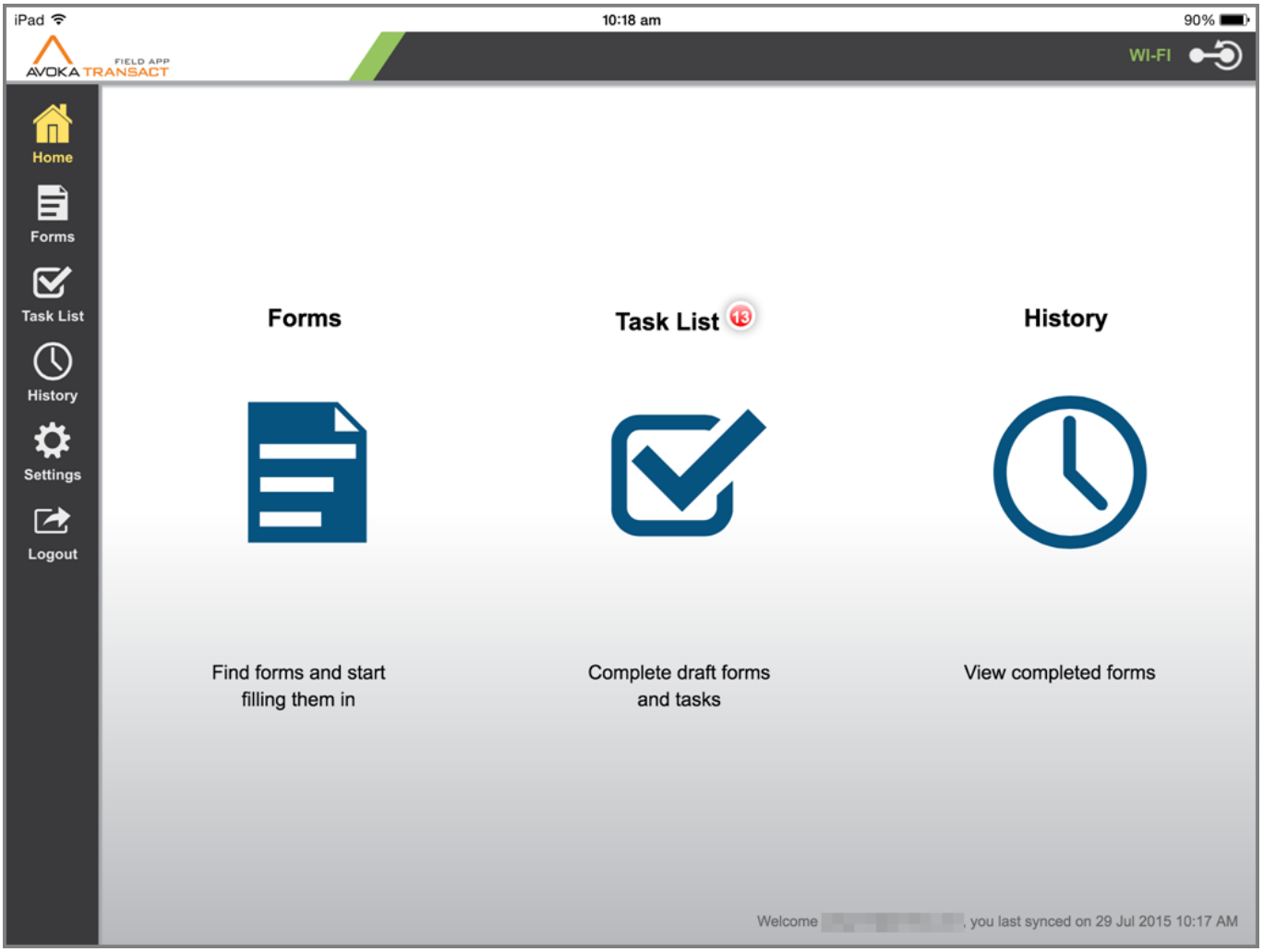
TransactField App | Deprecated in 21.11

TransactField App is used to collect some data remotely using online forms.

Once you have installed the app on your mobile device, it is accessed by selecting the icon on your mobile device, as shown below:



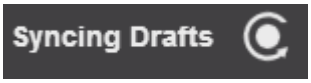
You will be directed to the home screen in TransactField App.



If you are online, the information on TransactField App will automatically synchronize with your server. The online status icon (found in the top right hand corner of your screen) is accessible from within any screen in TransactField App. The app will either be shown in one of the following states:

Synchronizing

Synchronizing - the app is currently synchronizing with the server (several functions will be disabled till it is finished). The icon is a rotating animation and various messages display beside the icon, such as "Synching Profile", "Completing Sync", "Downloading n Templates" or "Downloading n Prefills".



Disconnected

Disconnected - the app can not currently access the server. Tap on the icon to re-establish the connection and to re-sync.



WI-FI

When connected the type of connection is displayed. This will be one of WI-FI, Cell or Connected (for Cable connections). Tapping on the icon will re-sync the app with the server



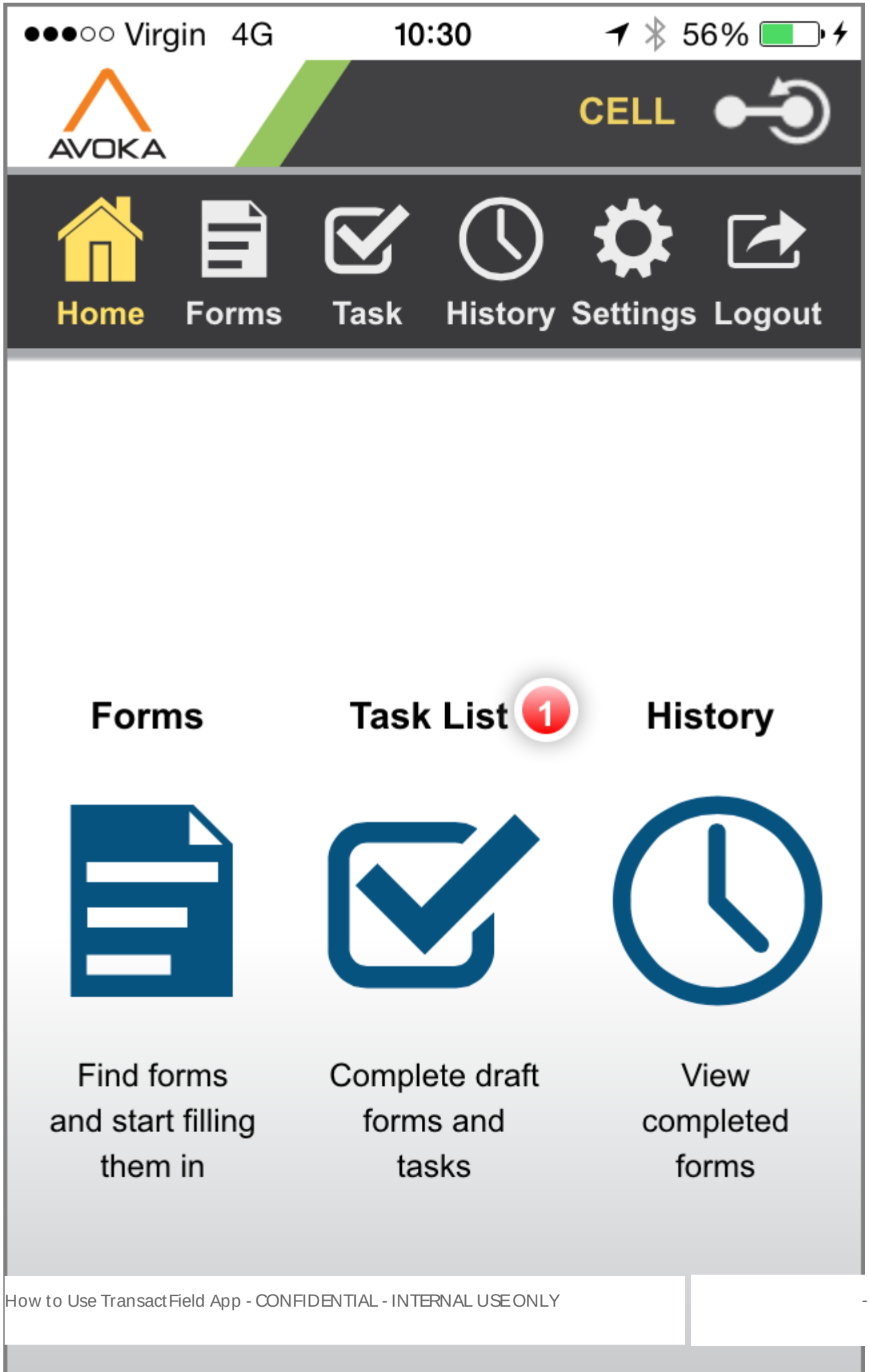
TransactField App will automatically synchronize with Journey Manager after a specified number of seconds (predefined in Journey Manager and visible in the app under "Settings -> Preferences -> Auto Sync Interval") if you are connected to the server.

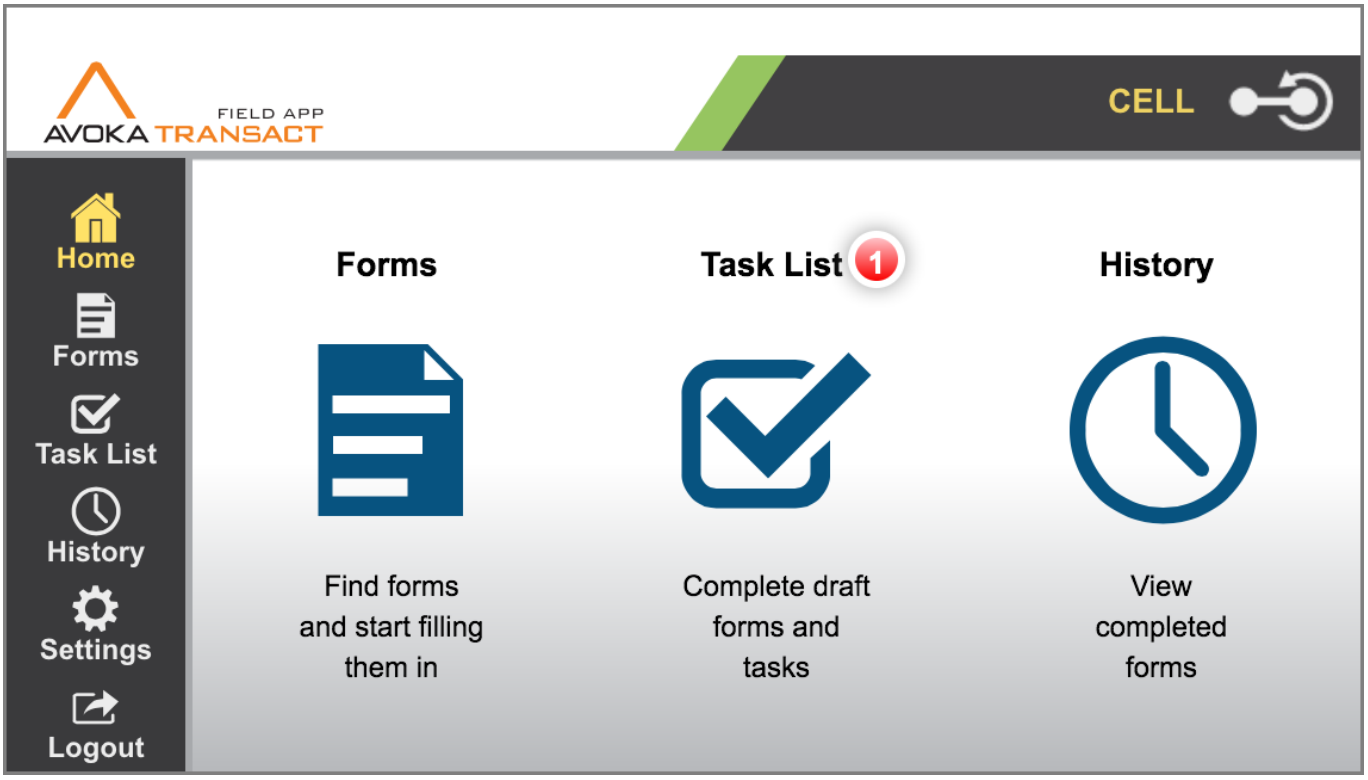
However clicking on the status icon allows you to manually synchronize TransactField App.

> If you have a form open, Transact Mobile App will not synchronize. When you close the form, it will resume synchronizing.

Navigating around TransactField App

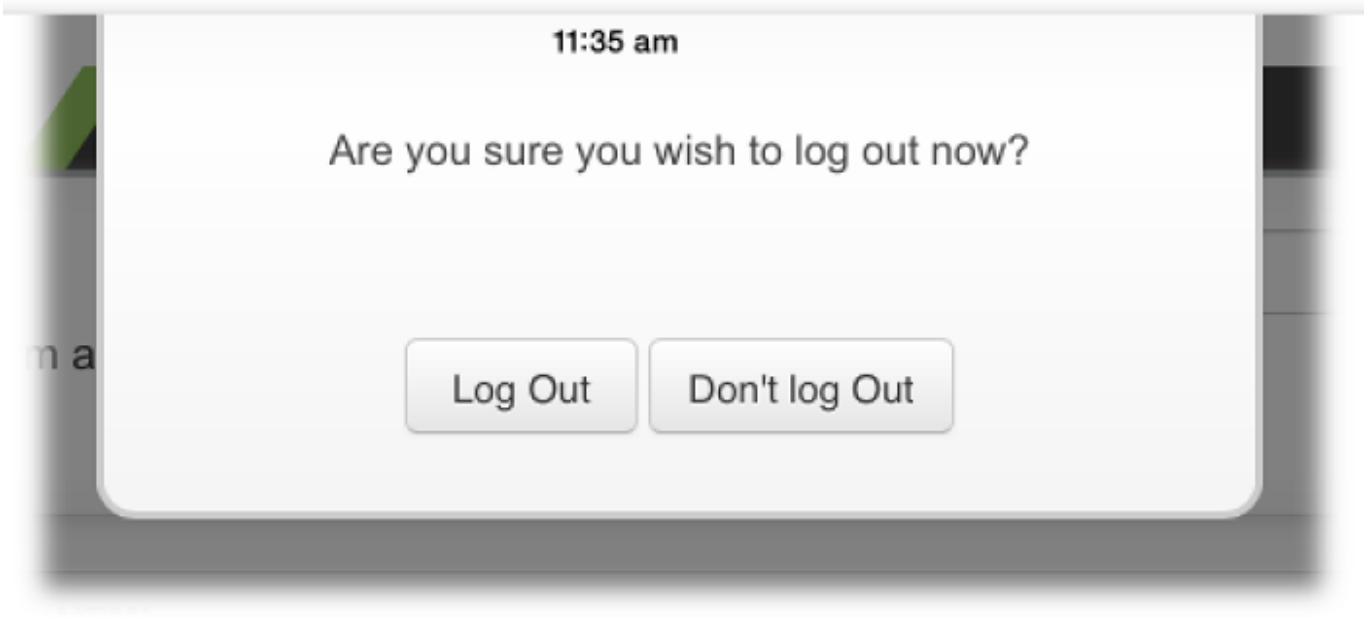
When you open TransactField App, you will be directed to the Home screen. The Home screen gives you access to the various capabilities within the application.


Virgin 4G
10:30
56%
AVOKA
CELL
Home
Forms
Task
History
Settings
Logout
Forms
Task List
1
History
Find forms and start filling them in
Complete draft forms and tasks
View completed forms

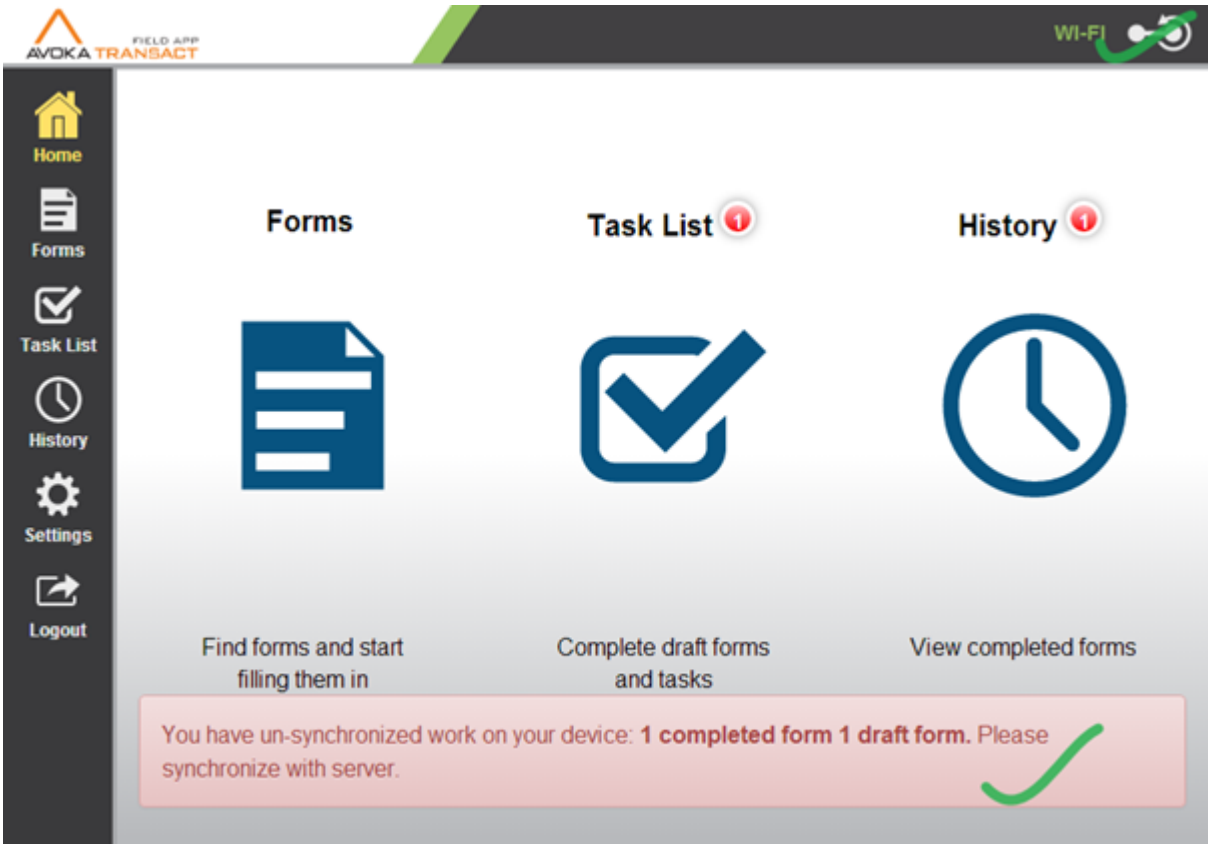


The Home screen consists of a menu (which will either be located across the top of the screen or to the left of the content area depending on the device type and orientation) and the content area where the 'Forms', 'Task List' and 'History' icons are visible by default.

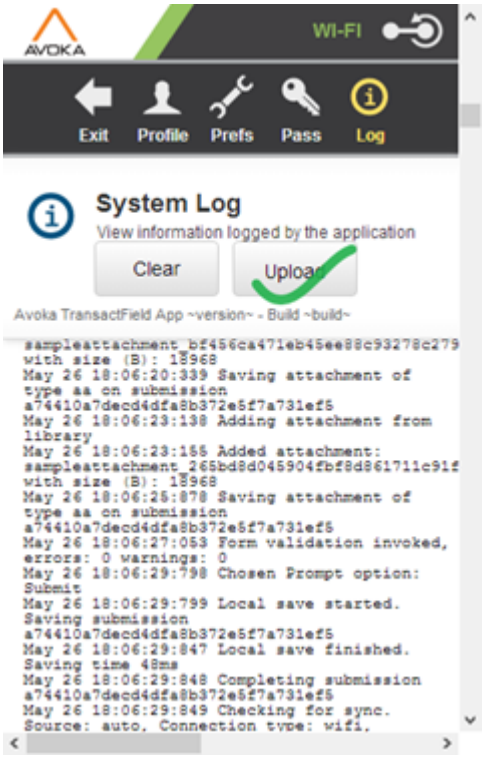
- Forms lists those forms you can access or have been allocated through Journey Manager.
- Task List shows your partially completed forms or those passed on to you for completion.
- History lists submitted forms and their receipt numbers. You cannot access forms after their submission.
- Settings is for making configuration changes to the TransactField App.
- Logout is for exiting the TransactField App.



Check TransactField App is Connected - this is displayed with indicator in top right corner (WI-FI, CELL, CABLE). When disconnected for a long time, attempt to connect device to network to minimize the risk of losing your work.



If you have synchronization errors and you have outstanding work (Home screen red warning box) even after several sync attempts, upload detailed log, as displayed below, and contact your administrator.



You can check the TransactField App synchronization and device logs in Journey Manager.

Next, learn about forms in TransactField App.

Forms in TransactField App

TransactField App | Deprecated in 21.11

TransactField App allows you to access and view a list of your forms.

Types of Forms

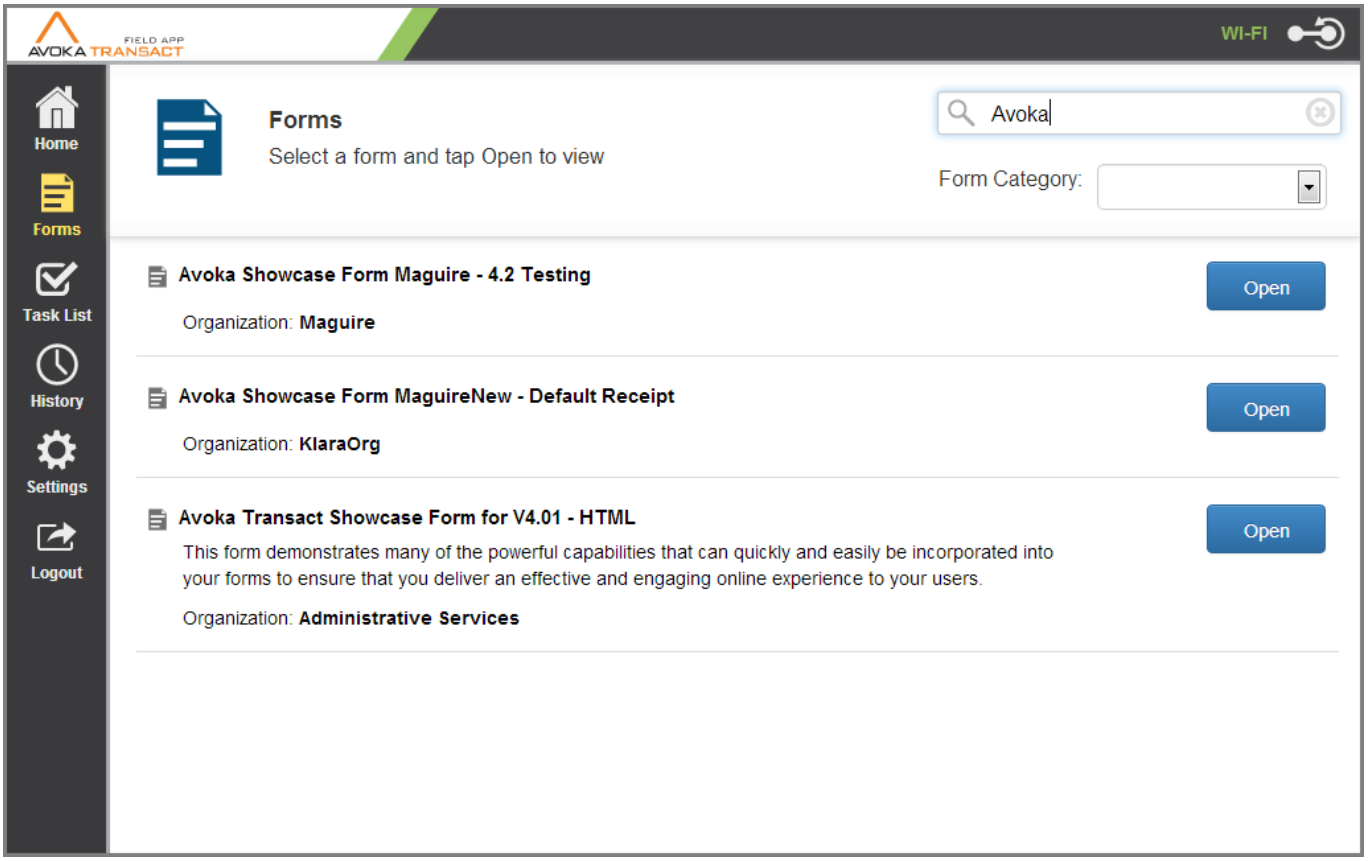
- Some forms may only require a handful of fields to gather information, while others will require a complex structure that changes the face of the form depending what the user inputs.
- In the "Top to Bottom" form structure, fields are simply laid out one below the previous one and navigation occurs by simply 'swiping your finger' up and down the screen.
- In a form containing a wizard structure, form pages contain navigation buttons typically 'Next' and 'Previous' to move between pages. There might also be a menu on the left or top side of the form which allows an alternate way to navigate among form pages.
- The interaction with the form should be the same all devices. However, the form may re-arrange itself depending on the width of the screen being used: this is called responsive behavior.

Widgets

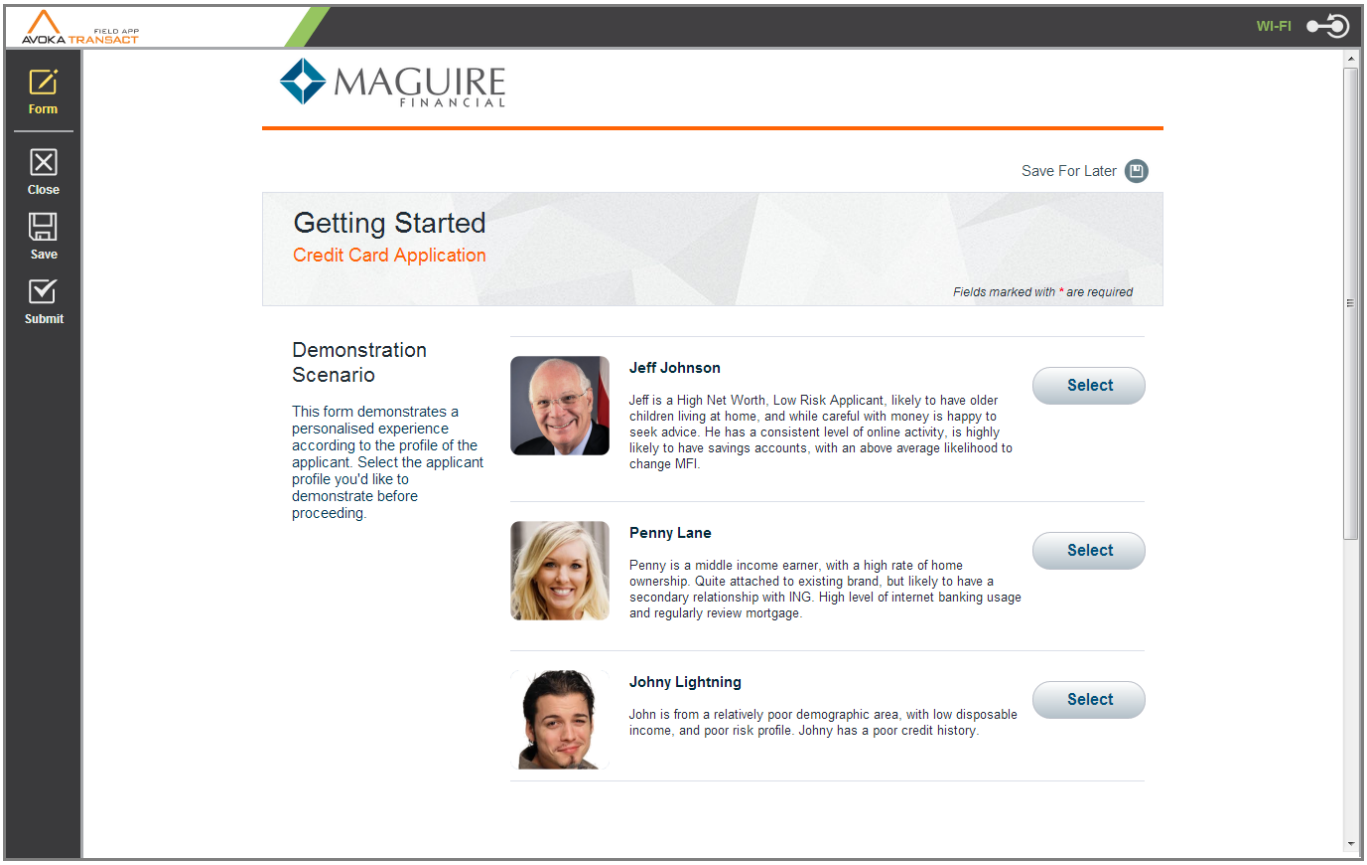
- Most forms will contain the basic widget types, e.g.: a text field which enables the form filler to enter information and radio buttons or check boxes to enable the form filler to make choices.
- Some widgets, such as the date picker and radio buttons, have the same functionality on all devices but display different on mobile devices.
- Other widgets may only be used on a mobile device. These include:
  - The camera widget with gallery integration which allows the mobile form filler to easily capture a picture, such as a damaged vehicle, a patient's injury, a driver's license, or a passport. Pictures are submitted as part of the form submission. The camera widget only works on devices that have a camera.
  - The scribble signature field that allows signatures created with a finger or stylus to be collected right on the screen.

Fill out a Form

Selecting Forms either from the tab menu or the home screen, directs you to a list of accessible forms with brief descriptions, if available.



Tapping on Open takes you to the form's first page. The tab menu may also change to form-specific functions.

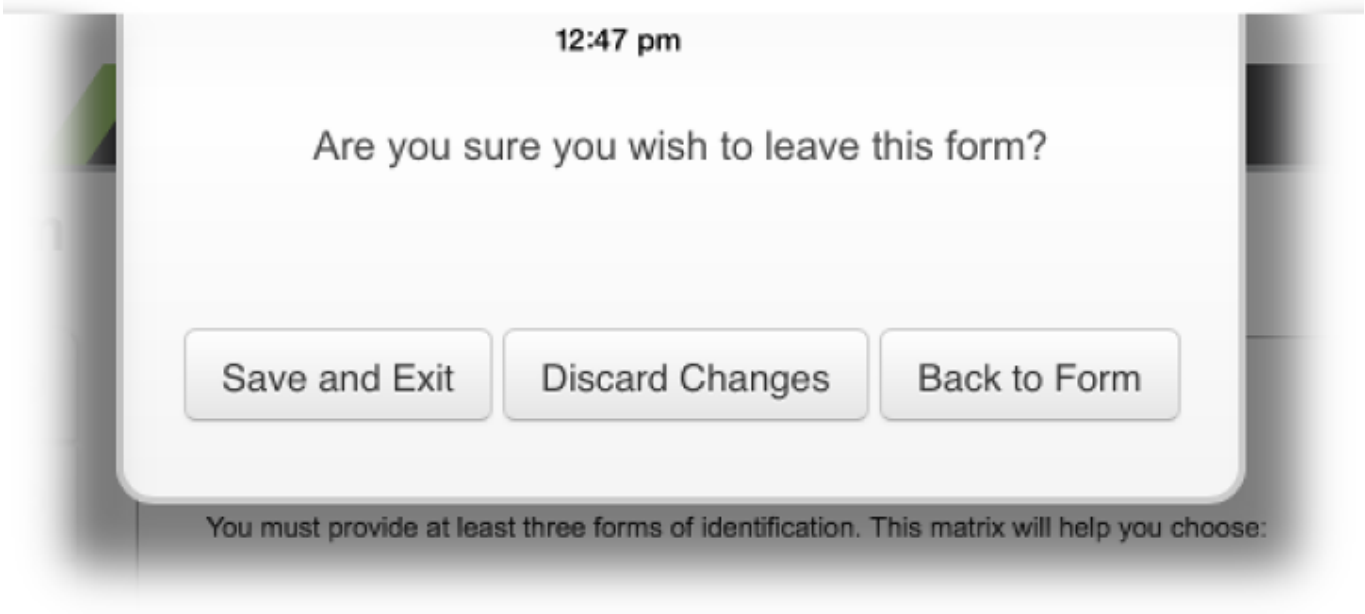


The Submit icon is selected to complete the form. If the user is online, the form will be submitted and an entry goes in the History tab, together with a Reference number. If you are not connected, it is shown in the Task tab, with the status Pending. It moves to History after a successful synchronization.

It is possible to suppress the submit button in the Mobile App and only submit via a form submit button or visa-versa. This can be achieved through configuration settings made in Composer or in Transaction Manager.

TransactField now supports in-form attachments. The form is setup with the supporting widget. When selected the attachments can be sourced directly from the device's internal camera, from the internal camera roll, or from your corporate Box cloud service.

The 'Close' icon allows the form filler to exit the form with various options:



- Save and Exit saves the form into your Task List for future completion.
- Discard Changes does not save any changes made.
- Back to Form takes you back to the open form.

Next, learn about tasks in TransactField App.

Task List in TransactField App

TransactField App | Deprecated in 21.11

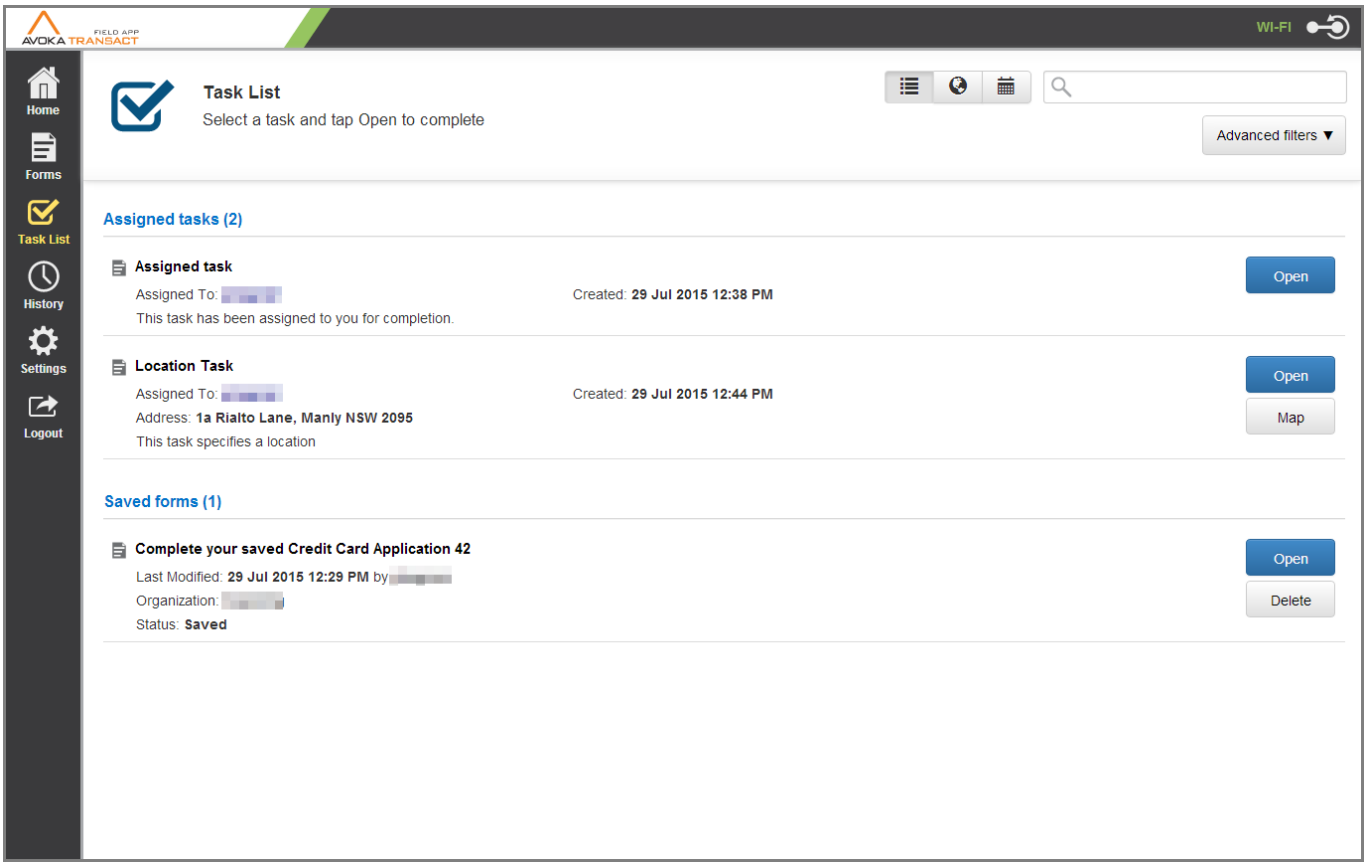
Task List contains any partially-completed forms, which can include:

- Forms started by you and then saved for later submission.
- Pre-populated forms assigned to you for completion.
- Submitted forms that have not been synchronized.

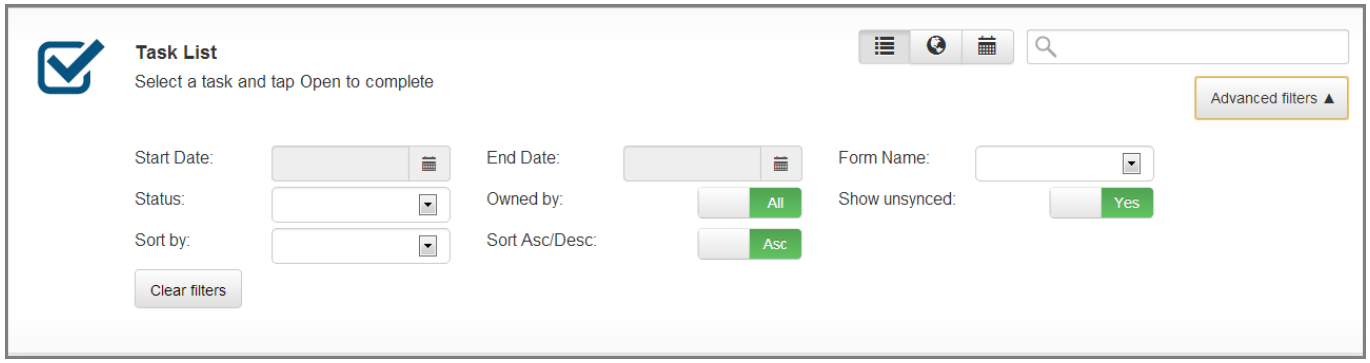
Using the Task List

There is a maximum number of tasks that can appear in the 'Task List'. This is set in Journey Manager by your administrator.

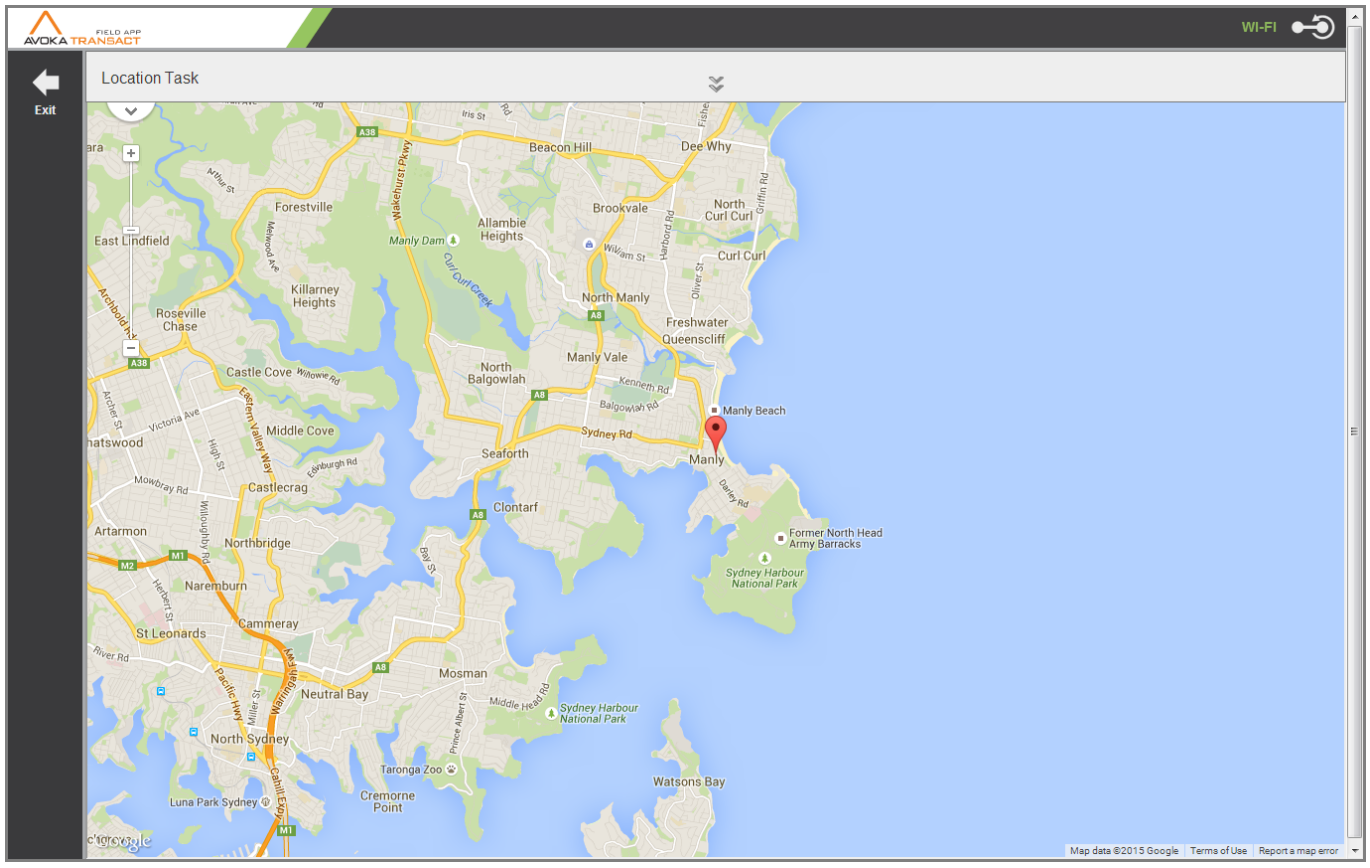
Task List Sections: The following screen shows that the user has 2 assigned tasks and one saved form in their queue.



Filtering: Tasks can be filtered using the Advanced filter options.



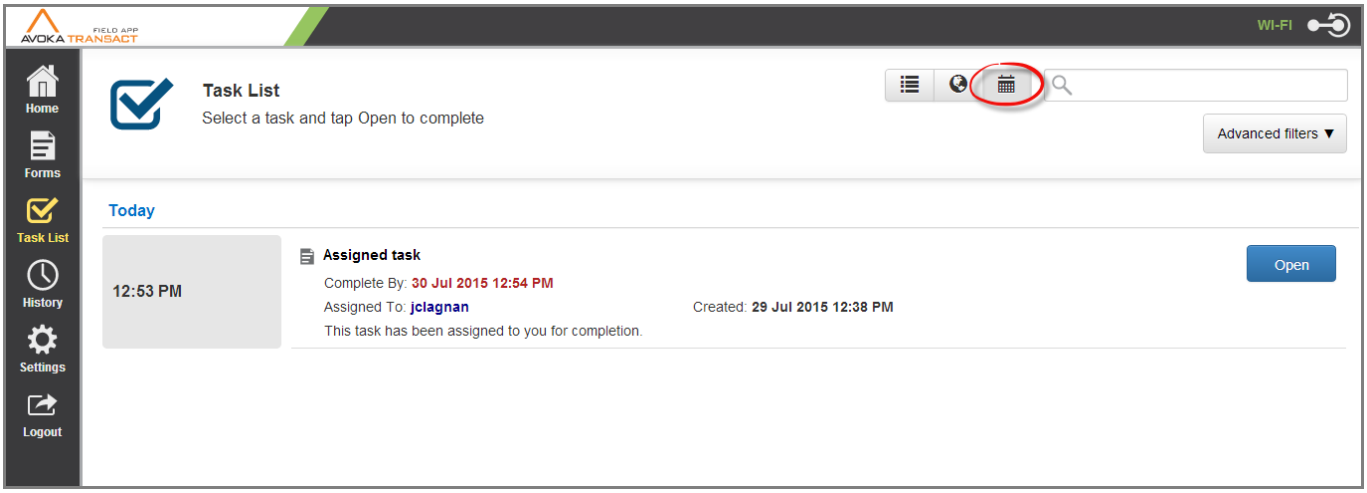
Location options: Tasks can be specified with a location. This will introduce a map option to the task and will allow the map to be shown to the user when selected.



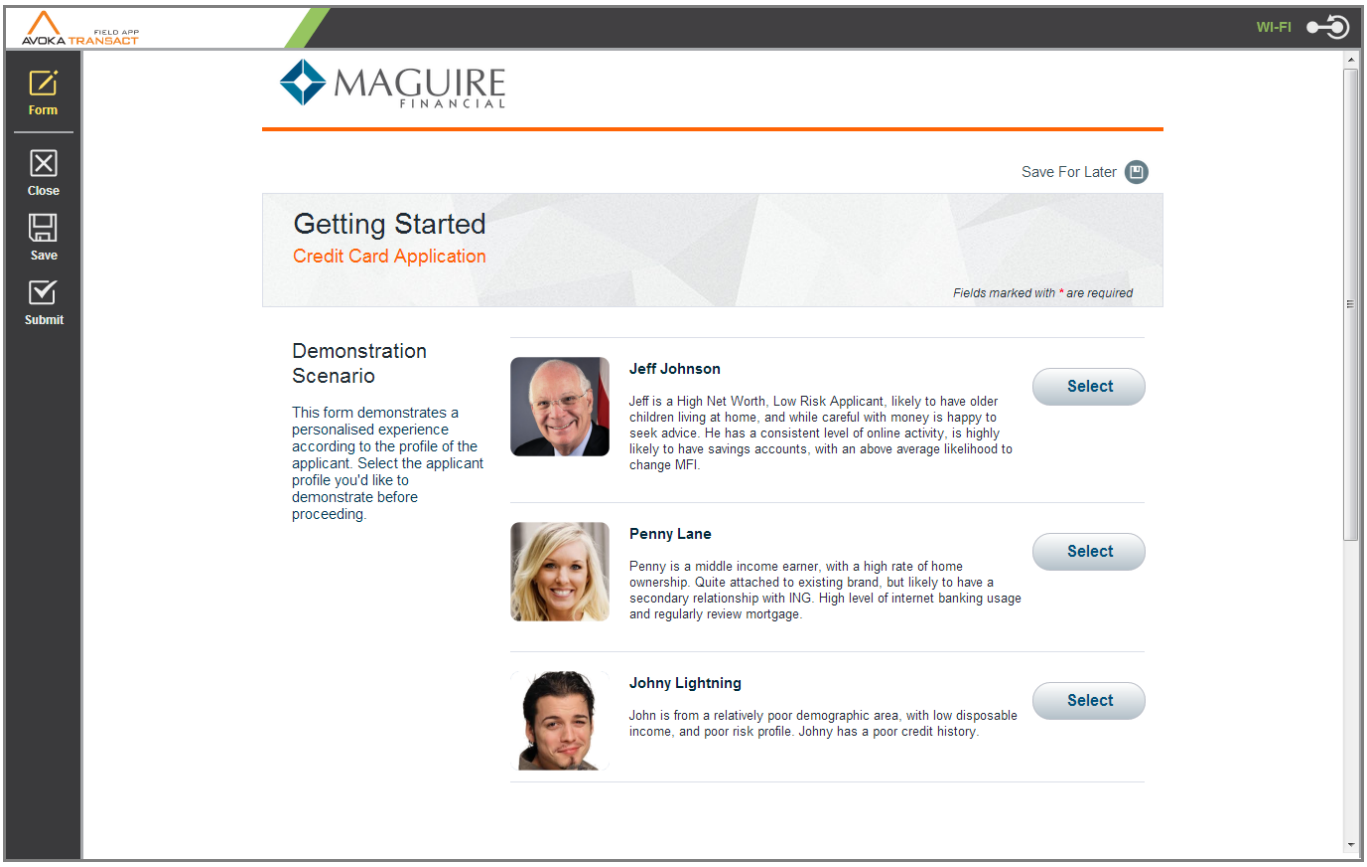
Task scheduled/expiry dates: A task can have dates assigned to it. They are shown in the task tile as below.



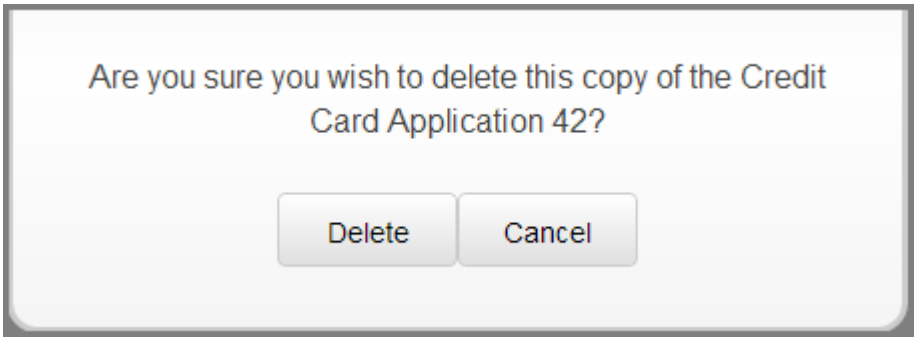
The date filter icon will show the scheduled tasks in time order.



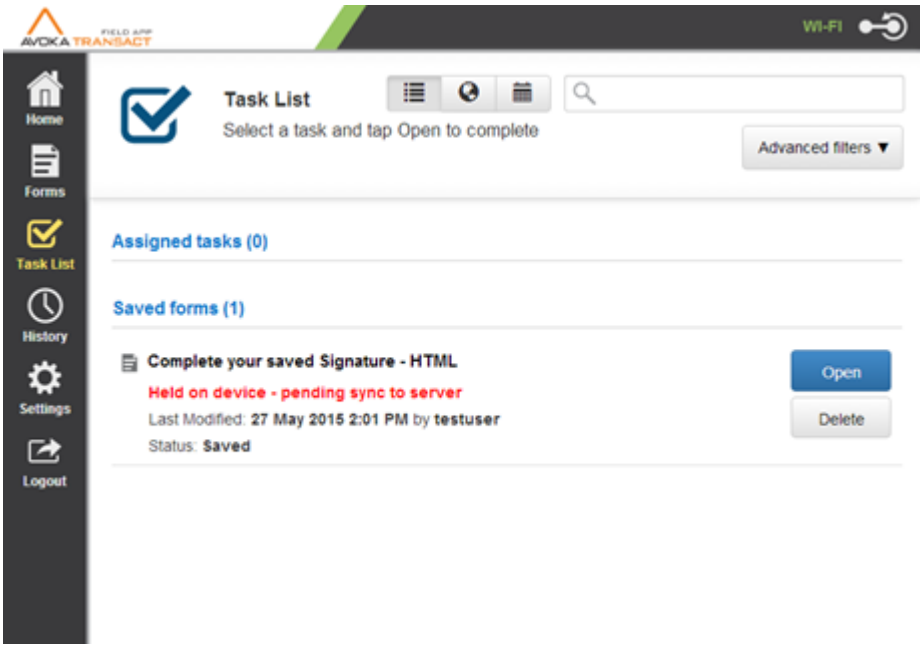
Task Open button is for editing the form.



Task Delete displays this confirmation message:



Check your home screen, task list and history page for outstanding items. As displayed on screen-shots you can easily see if there is any outstanding work. In this case tap on sync icon in top right corner or on red warning box on home screen to synchronize.



Next, learn about a built-in camera in TransactField App.

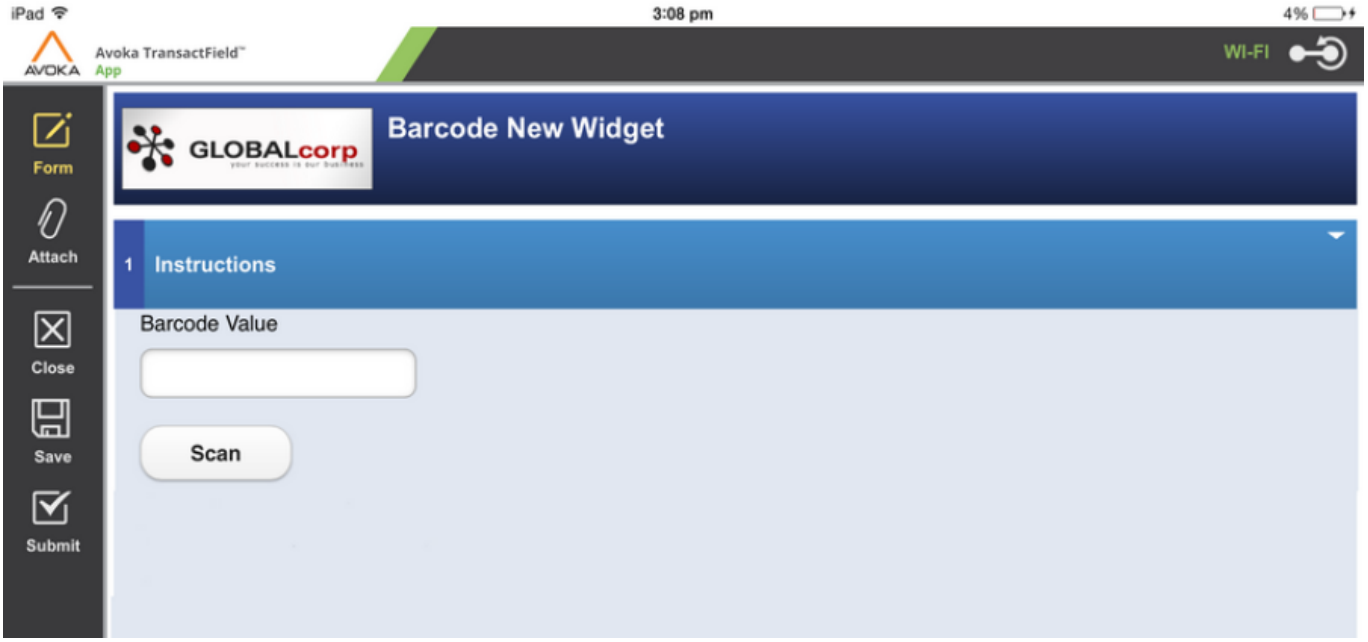
Use a Build-in Camera in TransactField App

TransactField App | Deprecated in 21.11

There are some form functions available only through native apps, the inbuilt camera being one of them. HTML forms can have the user upload images from the device's camera roll as attachments, but the mobile app can go further and have the user snap a photo from the form. The mobile App can also handle bar codes in a way not available to HTML forms accessed through a web browser.

Barcodes

The Temenos Journey platform allows a form to have a **barcode**[1] button, so a device camera reads a barcode and its data is returned to the form. This eliminates the need for a user to submit the form for the barcode to be deciphered, because it is now done within TransactField App.



The barcode must be positioned over the semi-transparent red line, the device must be moved close enough that the image focuses and when the software finds that it can read the image, the user is taken back to the form with the **Barcode Value** field filled in with the value of the code.

The barcode scanner module behaves slightly differently in Android and Windows phone devices: there may not be the red line display (to simulate a laser scanner) and the camera's focus may operate differently.

[1]Barcode is a machine-readable code in the form of numbers and a pattern of parallel lines of varying widths, printed on a commodity and used especially for stock control.


iPad
11:13 am
92%
1234567890
Cancel



Note: for iPads, iPhones and iPod Touches — the camera's auto-focus software does not work in the usual way, by tapping on a point in the preview window. The only real control, other than physically moving the whole device, is the "Cancel" button at the bottom of the window. The user does not have a shutter control to take the picture in the usual way. Instead the App goes back to the form when it has successfully read the barcode.

Supported Formats

TransactField App supports the following **AIM**[1] formats:

UPC-A and UPC-E

The Universal Product Code or UPC barcode was the first bar code symbology widely adopted. Its birth is usually set at April 3, 1973, when the grocery industry formally established UPC as the standard bar code symbology for product marking. Foreign interest in UPC led to the adoption of the EAN code format, similar to UPC, in December 1976.

[1]Association for Automatic Identification and Mobility (AIM) is an industry trade group that developed and standardized bar codes, Automatic identification and data capture.



Code 93

Code 93 offers higher information density for alphanumeric data than either Code 39 or Code 128. Code 93 is used primarily by Canada Post to encode supplementary delivery information. Every symbol includes two check characters.



Codabar

Codabar can encode the digits 0 through 9, six symbols (-:.$/+), and the start/stop characters A, B, C, D, E, * , N, or T.



Data Matrix

Data Matrix is a two-dimensional matrix symbology containing dark and light square data modules. It has a finder pattern of two solid lines and two alternating dark and light lines on the perimeter of the symbol. A two-dimensional imaging device such as a CCD camera is necessary to scan the symbology. Data Matrix is used for small item marking applications using a wide variety of printing and marking technologies.

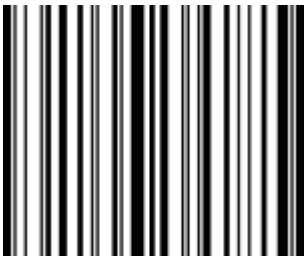


EAN-8 and EAN-13

See the UPC variants above

Code 128

Code 128, a very high-density barcode symbology, is currently used extensively world wide in shipping and packaging industries. The symbol can be as long as necessary to store the encoded data. It is designed to encode all 128 ASCII characters, and will use the least amount of space for data of 6 characters or more of any 1-D symbology.



RSS-14

RSS (Reduced Space Symbology) is now called GS-1 Databar. See http://www.databar-barcode.info/ Example: Databar-14:



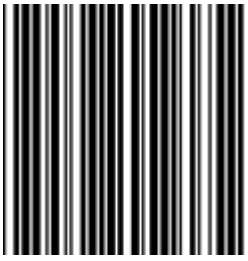
Aztec

Partially supported.Aztec Code is a two-dimensional matrix symbology containing dark and light square data modules. It has a finder pattern of concentric square rings centered on a single dark module located in the center of the symbol. A two-dimensional imaging device such as a CCD camera is necessary to scan the symbology. Aztec Code is used for small item marking applications using a wide variety of printing and marking technologies, and is also well suited for displays of cell phones and other mobile devices.



Code 39

Code 39, pioneered by the defense and automotive industries, is an alphanumeric bar code. The symbol can be as long as necessary to store the encoded data. It is designed to encode 26 uppercase letters, 10 digits and 7 special characters. It can be extended to code all 128 ASCII characters by using a two character coding scheme.



ITF

ITF barcodes, also known as Interleaved 2 of 5 barcodes, encode 14 numeric digits and are generally used for the packaging level of products. Since they can deal with high printing tolerances, ITF is a good choice when barcodes need to be printed on corrugated cardboard.



RSS Expanded

RSS Expanded is capable of storing 74 numeric or 41 alphabetic characters and is normally used in point of sale retail applications. See http://www.bardecode.com/en1/rss-expanded-examples/

We support most variants.

PDF 417

Current support needs further work. PDF417 is a two-dimensional, multi-row symbology designed to be scanned by laser scanners and linear CCD scanners and used to encode data files with hundreds or thousands of characters in a laser scannable symbol. PDF417 is used in a variety of applications, primarily transport, identification cards, and inventory management.



QR Code

QR Code is a two-dimensional matrix symbology containing dark and light square data modules. It has position detection patterns on three of its four corners and features direct encodation of the Japanese Kana-Kanji character set. A two-dimensional imaging device such as a CCD camera is necessary to scan the symbology. Although initially used for tracking parts in vehicle manufacturing, QR Codes are now used in a much broader context, including both commercial tracking applications and convenience-oriented applications aimed at mobile phone users (known as mobile tagging).

This is the recommended barcode format. Avoka testing shows it is the most suitable for mobile.



Next, learn how to view history in TransactField App.

View Form History in TransactField App

TransactField App | Deprecated in 21.11

The History folder contains a list of submitted and synchronized forms and the current status of the submission. The status may be updated by downstream processes to indicate the progress of your submission as it flows through the organization.

Using the History folder

The History List is limited to a maximum number of submissions (predefined in Journey Manager).

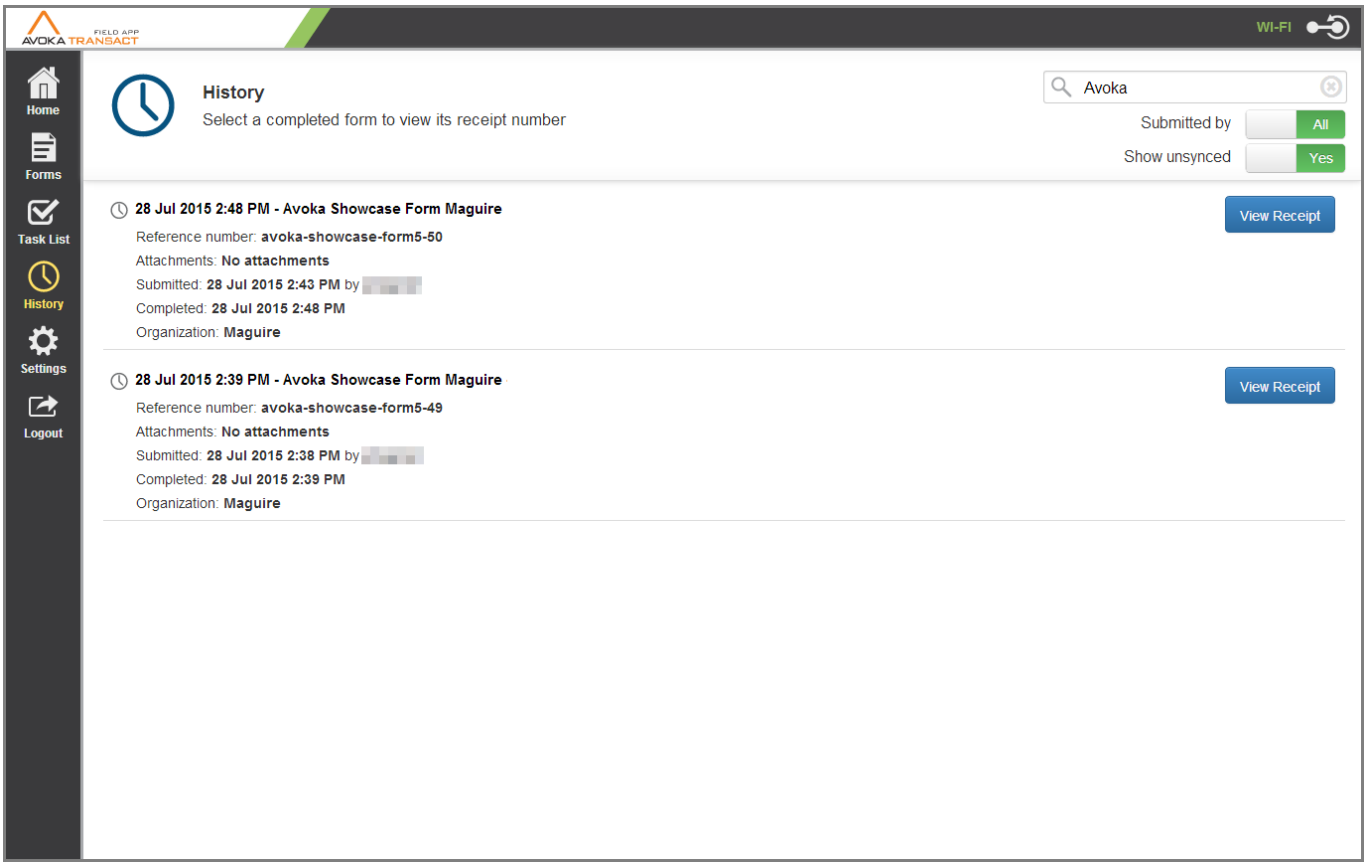
Viewing Options

The "Search" Field at the top right of the screen filters forms by input keywords. In the screen below we have used "Avoka" as the search criteria.

There are two filter switches to further assist when many items exist:

'Submitted by', defaulted to "All", it allows users 2 options: 'Me' and 'All'. This filters which forms become visible.

'Show unsynced' defaulted to "Yes". This allows a user to exclude any unsynchronized items from the view



Reference Number: will contain the system reference number for the submission e.g. child-protection-rep1.

Total Payment: will contain either 'No payment' (not required in the form) or a payment amount.

Attachments: will contain 'No attachments' (not required in the form) or a list of attachments that were uploaded to the form.

Submitted: will contain the date and time of form was submission (whether online or offline) and when the submission actually completed.

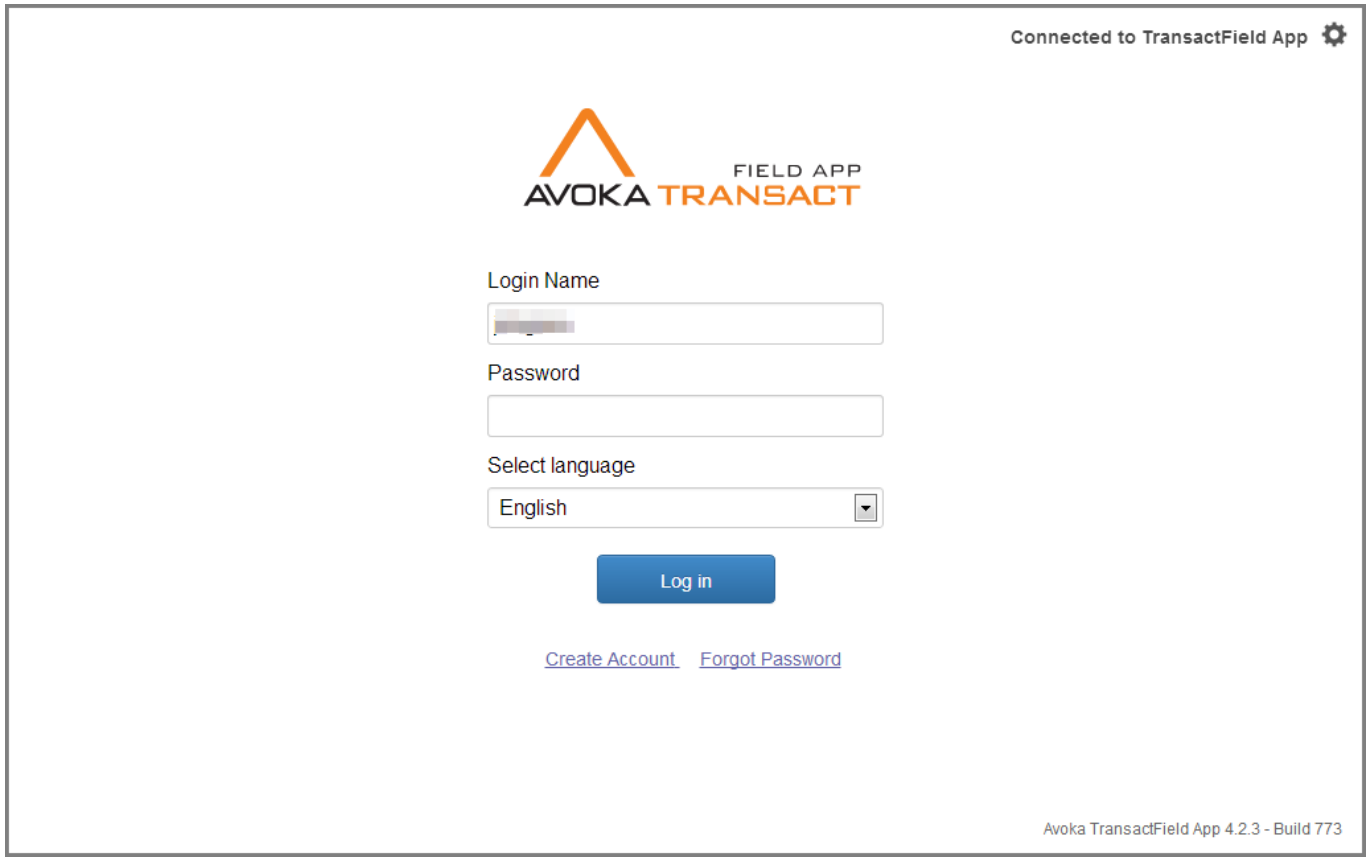
The 'View Receipt' button may, or may not, be seen in the History listing (predefined in Journey Manager). Clicking on it, will open the mobile device's web browser to display your receipt. Note that you will have to login using the same credentials used to login to the Mobile App. This connects the browser with the Mobile App.

Next, learn about multiple languages in TransactField App.

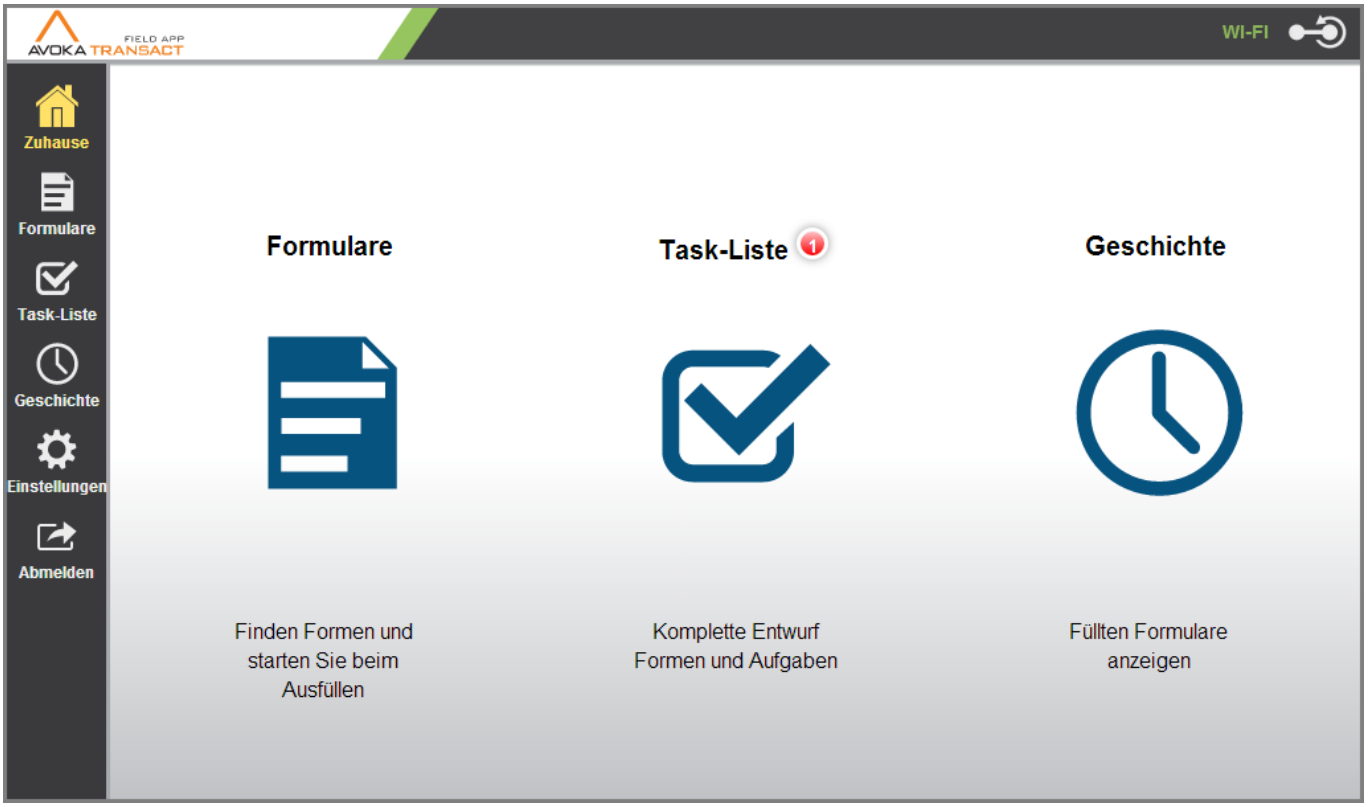
Multiple Languages in TransactField App

TransactField App | Deprecated in 21.11

Journey Manager can be configured to support multiple languages, which is called localization. You can log in using localization support.




FIELD APP
English
German
Czech
Portuguese
Chinese
German
einloggen
Konto erstellen
Passwort vergessen


AVOKA TRANSACT
FIELD APP
WI-FI
Zuhause
Formulare
Task-Liste
Geschichte
Einstellungen
Abmelden
Formulare
Task-Liste
Geschichte
Finden Formen und starten Sie beim Ausfüllen
Komplette Entwurf Formen und Aufgaben
Füllten Formulare anzeigen

With the Manager server configured to enable localization, the login screen will display a selection of languages. The actual choice depends on the localization settings of the server. Selecting a language other than the default changes the chrome of TransactField App, namely all surrounding navigation links and icons such as the left-hand menu on a device in portrait orientation, as shown above.

The form contents are not automatically translated: each of the forms must be localized in Composer (See the Composer User Guide on how to do this). Each form though, can be localized for several languages and served according to the user's language selection.

Journey Manager can be configured to display the app in any language of your choice. There is an automatic translation facility in Manager, based on the Google Translate platform. Avoka strongly advises to check all the text fragments in the app with fluent speakers in the target language to ensure the machine translations are accurate.

Next, learn about security in TransactField App.

TransactField App Security

TransactField App | Deprecated in 21.11

This topic discusses the various security measures within TransactField App. For more information, see our security architecture document.

Accessing Security

The TransactField App has various levels of security:

- The first time you log in, you must be online, and your user-name and password are verified against Journey Manager.
- On subsequent logins:
  - If you are online, you will always be authenticated against Journey Manager.
  - If you are offline, you can still log in and be authenticated against an encrypted stored copy of your credentials.

Note: This means that if your server password has changed, you must continue using your old password offline until you log in online.

- If your session is inactive for a specified length of time (predefined in Journey Manager), the session will time out and you will be required to log in again.
- If you fail to log in successfully after a specified number of times (predefined in Journey Manager), the TransactField App data (including data not saved back to the server) on your mobile device will be permanently deleted.
- All the TransactField App data is securely stored with 128-bit encryption.
- All communication is through the 'https' protocol to avoid 'man-in-the-middle' attacks.

Form Security

In Journey Manager, administrators can set up 'Form Work Group Support' which impacts the stage at which a group of users may access a form in TransactField App.

The stages of a form include one or more of the following:

- New forms - visible in the 'Forms' listing.
- Saved or Assigned Forms - visible in the 'Task List'.
- Completed Forms - visible in the 'History' listing

Next, learn how to configure TransactField App.

Configure TransactField App

TransactField App | Deprecated in 21.11

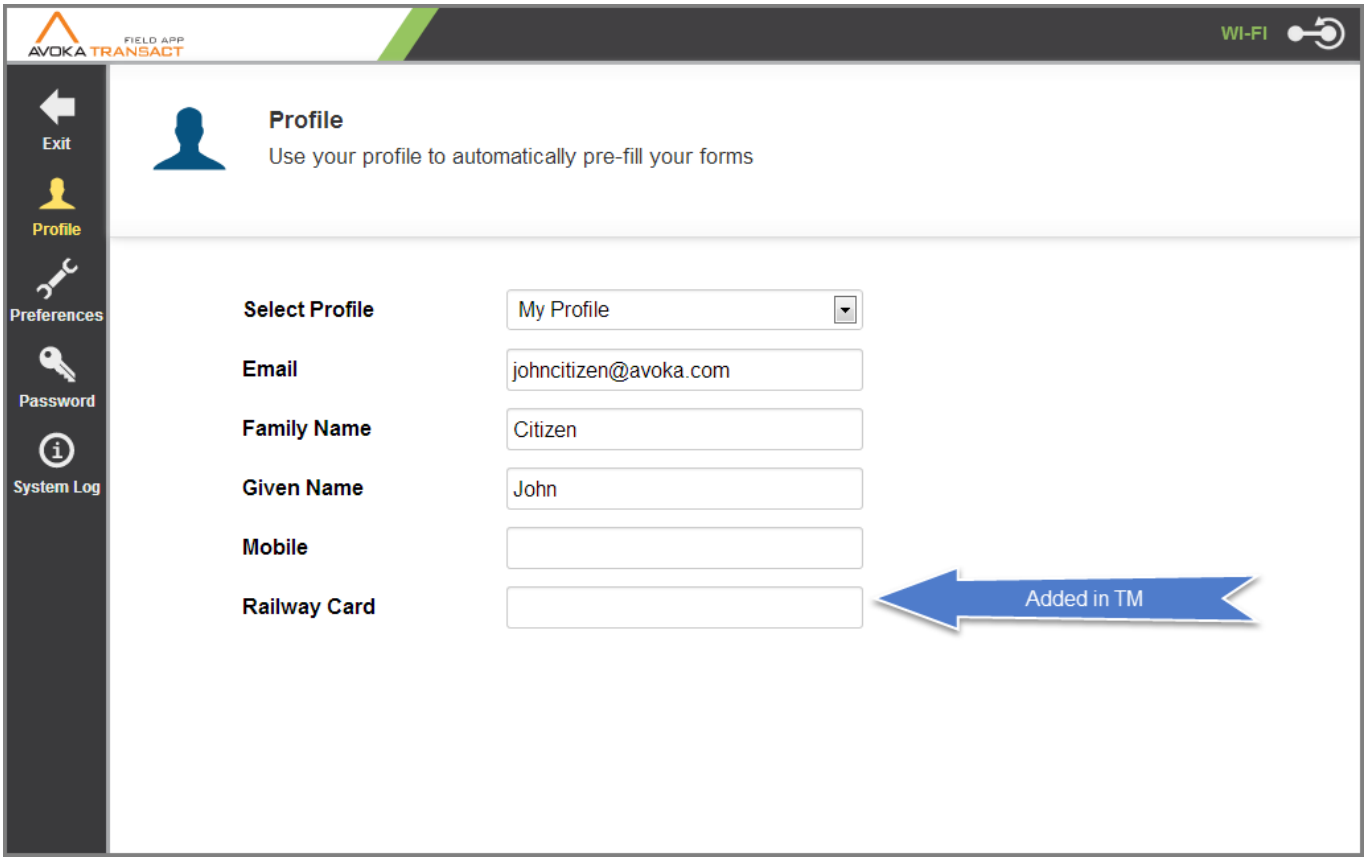
You can configure TransactField App using:

1. Client settings in the Settings tab of TransactField App, which you can modify.
2. System settings that are configured by a Journey Manager administrator. You can't change these settings in TransactField App. For more information, see configure form tiles.

Client Settings

Click **Settings** to change configuration in TransactField App.

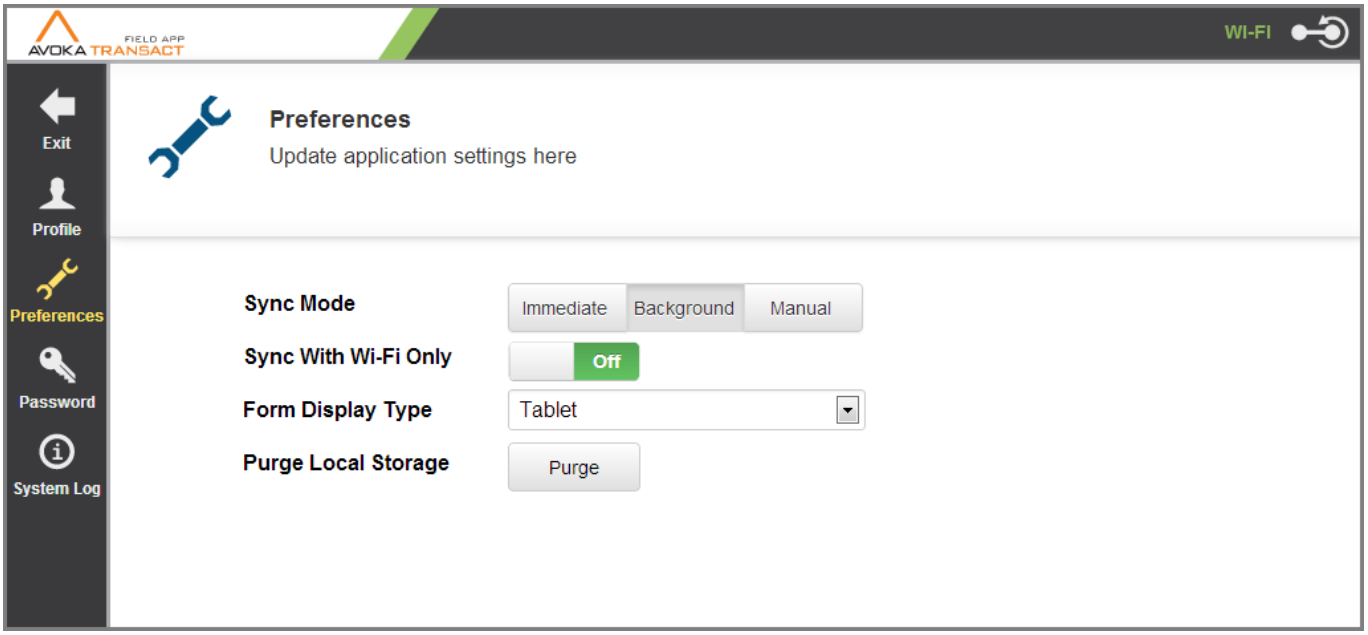
Click **Profile** to change configuration of the user profile details.



This will cause the following to take place:

- Changed profile information is updated in Journey Manager next time you are connected to it.
- The information that you see on this page may vary depending on the configuration settings that your administrator has set up.
- It is also possible for you to configure multiple profiles, however this feature is rarely enabled.

Click **Preferences.**



- Click **Sync Mode** to change it to:
  - **Immediate** — which is how previous versions of the app used to synch. Synchronization happens after every form save, submit or delete and you could not open any form for several minutes until synching finished. An immediate sync will always occur when you login for the first time, change servers or after a local storage purge.
  - **Background** (default) — Synchronization now happens at regular time intervals (default is 30 seconds), not every time forms change as per the previous option. You can also delete multiple forms without the need to wait for synchronization after the deletion of each form.
  - **Manual** — synchronization happens only when you initiate it by tapping on the sync button.This mode is for situations when you are working offline, for example, and don't want your work being frequently synchronized.
- Click **Sync With Wi-Fi Only** to conserve mobile data bandwidth.
- Select a form display type from the **Form Display Type** dropdown list, which can be changed from **Tablet** though this is not recommended if you are on a mobile device.
- Click **Purge Local Storage** to delete TransactField App data stored on your mobile device. This should be used with caution and only as a last resort when corruption of local data may have occurred.

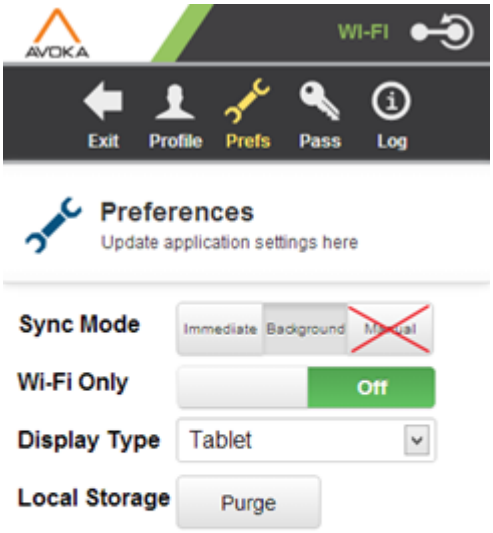
Click **Password** to change your password.

After you click **Change**, you will be taken to the TransactField App login screen.

Click **System Info** to view useful information to diagnose problems within TransactField App.

Click **Upload** to populate the TransactField App Client Log in Journey Manager the next time you are connected to the server.

Don't turn sync mode to manual without a good reason. Background sync mode is the default setting and will synchronize the app with the server regularly (usually every 30 seconds depending on Manager setting). It's important to have your work regularly backed up. Manual sync mode can be used temporarily under very unstable network conditions. Ensure you switch back to Background sync mode in regular conditions.



System Settings

There are a number of server side TransactField App Portal Properties which can be configured by system administrators.

Setting	Description	Default
Auto Sync Interval	The auto sync interval in seconds during which a sync action is performed.	30
Box Client Id	The client key for Box	<key>
Box Client Secret	The secret key for Box	<key>
Client Auto Save Interval	Interval in seconds specifying how often client automatically saves open forms.	60
Client Delete DB After Max Login Attempts	The TransactField App should delete the local database after the maximum number failed logins have been attempted.	3
Client Deprecated Versions	Comma separated list of T-Field	string

	versions which will be forced to update.	
Client HTTP timeout	Client HTTP request timeout in seconds.	180
Client Group Filter Default	The client task list and history group filter default setting.	Group
Client Log Length	Number of lines which will be persisted in log file on device.	1000
Client Log Location	Settings which turns on location logging in sync message. When enabled, sync message will include user latitude and longitude.	False
Client Session Timeout	The user inactivity duration in minutes after which the client is logged out. The default zero value means do not log the user out.	5 Minutes
Client Sync Mode	The predefined syncing mode for users. If Editable, user can choose the mode on device.	Editable
Client Sync WIFI Only	Settings which allow client syncing on Wifi connection only. If Editable, user can choose setting on device.	Editable
Client Use Native XML Parser	Use native XML parser on iOS devices.	True
Email 2FA Message	Email message for 2FA user account enrolment.	
Email 2FA Subject	Email subject for 2FA user account enrolment.	
Email Form Receipt Subject	Email subject for submitted form receipt email.	"Form Receipt"
Email Form Receipt Message	Email message for submitted form receipt email.	"You recently submitted a <form name> at <portal name>. Your submission reference number is: ..... Please see the attached PDF for a copy of your form submission. "
Email Lost Password Subject	Lost password reset email subject.	"Transaction Manager Lost Password"
Email Lost Password Message	Lost password reset email message.	"Thank you, an email with your new password will be sent to

		you in a few minutes. Please check your email SPAM folders if you don't receive an email, also note if you don't have a valid account you will not receive an email."
Email Submission Status Message	Email subject for submission processing status update	#if ($jobProcessMessageText) ${jobProcessMessageText} #else ${user.fullName} your form tracking code ${submission.trackingNumber} status has been updated to: ${submission.processingStatus} #end
Email Submission Status Subject	Email subject for submission processing status update	${submission.formName} Update
Email Template HTML	Email border template HTML	HTML template code
Email Template Mode	Specify the location of the Email HTML border template to use.	"Use Email Template HTML"
Forget Password Message	Password reset confirmation message to user.	"Thank you, an email with your new password will be sent to you in a few minutes. Please check your email SPAM folders if you don't receive an email, also note if you don't have a valid account you will not receive an email."
Form Load Timeout	Client maximum time allowed to load a form in milliseconds.	120000
Google API Key	Google API Services Access Key	
Max Attachment Pixel Dimension	The maximum height or width (whichever is bigger) in pixels. Otherwise the image will be re-sampled to the same aspect ratio with the maximum dimension the specified value. The default value, "0", means that images will not be resampled. Note: unresampled images in viewers that do not honour EXIF data, Windows Photo Viewer for example, may need rotation to be in the correct	800

	orientation. The resampling feature in Transaction Manager will take EXIF orientation into account.	
Max Client History Submissions	The max number of Submission History records the client can download from the server.	60
Max Client User Tasks	The max number of Task records the client can download from the server.	30
Sync Logging Level	The server sync logging level. If 'Detail' enabled then all sync calls will be logged.	Summary

A number of these properties control the TransactField App's behavior and the client synchronizes with these:

- Auto Sync Interval
- Client Sync Mode
- Client Sync WiFi Only
- Client Delete DB After Max Login Attempts
- Client Session Timeout
- Max Client User Tasks
- Max Client History Submissions

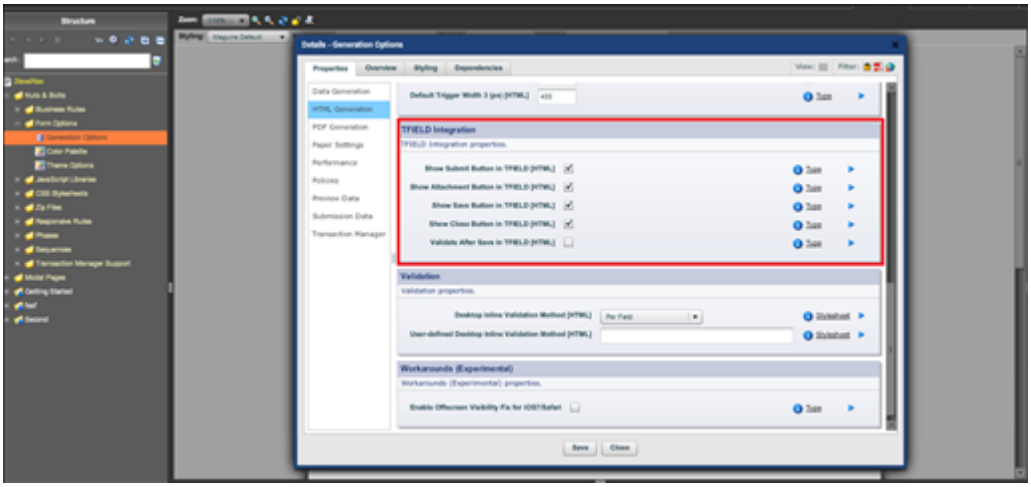
Disable the Standard Submit or Attachment Button

To disable the standard Submit or the Attachment button, go to **Nuts & Bolts > Form Options > Generation Options > Properties > HTML Generation > TFIELD**.

Why would you use these options:

If a form has no mandatory fields, it would be useful to place the Submit button at the end of the form to prevent the form filler submitting the form too early. In this case, you would disable the mobile app submit button by unchecking the option, and place a submit button on the form.

If you had a form with no attachments, it is useful to remove the attachment button from the application to minimize confusion. Additionally, if you have elected to use the camera or gallery widget button in the form, and prefer your users to use the in form options, you may wish to un-check this option.



Next, learn how to use TransactField App.

Configure Form Tiles

TransactField App | Deprecated in 21.11

TransactField App allows you to configure form tiles and data extracts from forms. Form tiles custom information is useful for faster navigation, filtering and sorting of saved tasks, forms and history items, especially when there are many items. This is mostly accomplished from Journey Manager. You can configure TransactField App to achieve the following.

Extract a Customer Email from Form Data to Display it in a Task List

To show a custom field on a TransactField App tile from a form:

Configure the form XML data extracts in Journey Manager:

1. Log on to Journey Manager
2. Select **Form > Data Config > Form Data Extract Mapping**
3. Name the field **Email** and enter the **XPath**[1] to the field in form data XML

Configuing the field for TransactField App

TransactField App needs to know what is available and how it should be displayed in the form tiles

[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).

- A tiles.json file needs to be created. This shows the minimal configuration to display the extracted email field for the appropriate form. The field will be displayed with "My Email" label.

```
[
	{
		"formCode" : "avoka-showcase-form5",
		"views" : [{
			"type" : "saved",
			"fields" : [{
				"label" : "My Email",
				"value" : "formDataMap.email"
			}]
		}]
	}
]
```

- In the example above the field is named "Email" in Manager and is accessed in tiles.json file as formDataMap.email. Note that Extract Field Names are transformed to camel case notation so a field called "Delivery Email" would be accessed as formDataMap.deliveryEmail
- the last step is to upload tiles.json file to Manager in T-Field portal

Synchronize the Device with Manager

This will download data extract rules and the **`tiles.json`** to device.

- open the form and enter e-mail address into mapped field
- save and close the form
- The email address from the form will be displayed in the tile as below

Extract a Date Field and Format it

Date fields need special handling as they are encoded in XML in a fixed format. To display them in more user friendly format follow these steps:

- Extract date field from Manager
- Name it Date by following this instruction: **Form > Data Config > Form Data Extract Mapping**
- Update the **tiles.json** to display the new field on the tile, for example a **Visit Date** label, as shown below

```
[
	{
		"formCode" : "avoka-showcase-form5",
		"views" : [{
			"type" : "saved",
			"fields" : [{
				"label" : "My Email",
				"value" : "formDataMap.email"
			}, {
			"label" : "Visit Date",
			"value" : "formDataMap.date",
			"dateFormat" : "EEE, d MMM yyyy"
			}]
		}]
	}
]
```

- The **dateFormat** field specifies date format string, which is similar to Java's SimpleDateFormat.

Customize a Title of a Tile

TransactField App shows a form name in the title by default . You can change it with an expression in **tiles.json** file by adding the following line **"title" : "Customer {{formDataMap.email}} visited on {{formDataMap.date|date:'EEE, d MMM yyyy HH:mm:ss '}}"**.

```
[
	{
		"formCode" : "avoka-showcase-form5",
		"views" : [{
			"type" : "saved",
			"title" : "Customer {{formDataMap.email}} visited on {{formDataMap.date|date:'EEE, d MMM yyyy HH:mm:ss '}}",
			"fields" : [{
				"label" : "My Email",
				"value" : "formDataMap.email"
				}, {
				"label" : "Visit Date",
				"value" : "formDataMap.date",
				"dateFormat" : "EEE, d MMM yyyy"
			}]
		}]
	}
]
```

Hide Default Fields

- Default fields such as "Status", "Organization", "Last Modified" or all task related fields can be hidden.
- This can be achieved by specifying **`hiddenFields`** in the **`tiles.json`**, as shown below.

Configure the History Screen Tiles Differently to the Task List and Saved Form Tiles

- Separate templates can be created for saved forms, tasks, and history items.
- These require a separate **`view`** in the **`tiles.json`**, as shown below.

Reference

- Default fields such as "Status", "Organization", "Last Modified" or all task related fields can be hidden.
- This can be achieved by specifying **`hiddenFields`** in the **`tiles.json`**.

Supported Views

- In the **`tiles.json`** file for each form, multiple view types can be specified. If a view without type exists, it will be used as default for all views. Any subsequent views will override the default.
  - saved
  - task
  - history

Template Values

- TransactField App allows access to information not only from submission xml but also from submission and form metadata:
- formDataMap
- form
- submission

Special date value handling

- dates are by default stored as yyyy-MM-dd in submission xml
- to allow for more friendly formats in form tiles, users can specify **`dateFormat`** for fields and also for titles (see tiles.json example)
- date format uses angular date filter for formatting which is very close to Java SimpleDateFormat format but doesn't support few options (era, long time zone name):
- https://docs.angularjs.org/api/ng/filter/dat
- https://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html

Detailed Example of the **`tiles.json`** file

```
[
	{
		"formCode" : "transact-showcase-fm",
		"views" : [{
			"title" : "Saved title: Customer {{formDataMap.deliveryEmail}} visited
{{formDataMap.myDate|date:'EEE, d MMM yyyy HH:mm:ss Z'}}",
			"type" : "saved",
			"fields" : [{
			"label" : "Delivery Email",
			"value" : "formDataMap.deliveryEmail"
		}, {
			"label" : "Form Name",
			"value" : "submission.formName"
		}, {
			"label" : "Visit Date",
			"value" : "formDataMap.myDate",
			"dateFormat" : "EEE, d MMM yyyy HH:mm:ss Z"
		}, {
			"label" : "Address",
			"value" : "formDataMap.address"
		}],
		"hiddenFields" : ["Last Modified", "Status", "Attachments", "Submitted"]}, {
			"title" : "Task title Customer visited {{formDataMap.myDate|date:'dd MM yyyy'}}"
			"type" : "history",
			"fields" : [{
			"label" : "Delivery Email",
			"value" : "formDataMap.deliveryEmail"
			}, {
			"label" : "Form Name",
			"value" : "submission.formName"
			}, {
			"label" : "Visit Date",
			"value" : "formDataMap.myDate",
			"dateFormat" : "dd.MM.yyyy"
			}, {
			"label" : "Form Version",
			"value" : "form.currentVersionNumber"
			}
			],
			"hiddenFields" : ["Last Modified", "Status", "Attachments", "Submitted"]
		}]
	}
]
```

You can hide the following fields:

```
[
        /**drafts*/
        "Last Modified",
        "Organization",
        "Status",
        /**tasks*/
        "Scheduled",
        "Complete By",
        "Assigned To",
        "Created",
        "Address",
        "Latitude/Longitude",
        "Message",
        /**history*/
        "Processing Status",
        "Status Updated",
        "Reference number",
        "Attachments",
        "Total Payment",
        "Submitted",
        "Completed"
]
```

Next, learn how to use TransactField App.

Composer User Guide

Composer | Form Builder | v4.3 & Higher

This User Guide aims to describe the available Composer functionality in detail. It is comprised of descriptions of Composer functionality as well as "How To" articles that step through set up procedures. Links and references are provided throughout the guide to other Avoka Knowledge Base Spaces that describe the functionality in more detail.

Throughout this administration guide, we use the notation "**->**" to indicate a location in the interface to carry out a task.; for example: "Access Control -> Users -> New System User"

Angle brackets ("**< >**") indicate the general name, for example "<Account>" for "Foobar Account" or "My Account" or whatever.

Next, learn how to get started with Composer.

Getting Started in Composer

Composer | Form Builder | v4.3 & Higher

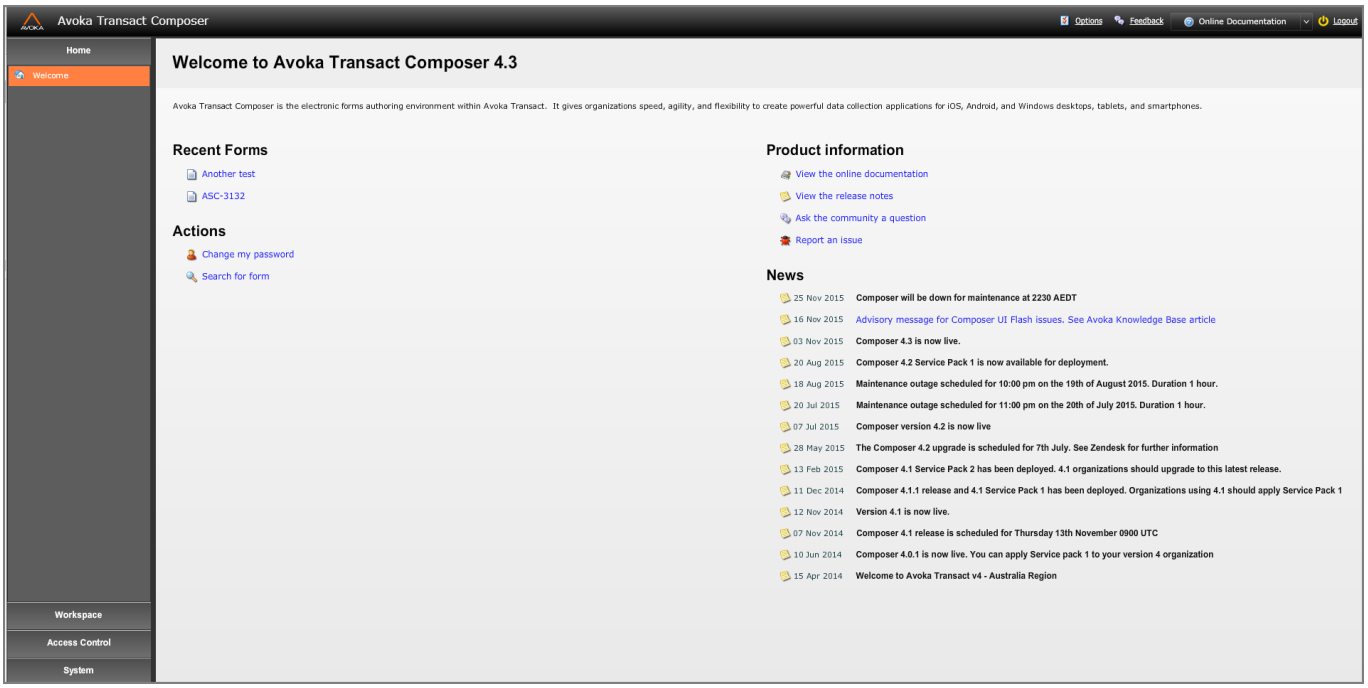
Before you can start using Composer, you must login in. There are various user roles and there is a well defined hierarchy of documents (or forms) into Organizations and Projects that define user access. For more information on Access Control, see Composer Administration Guide.

Login Page

The composer login page presents the user with a standard user name and password dialogue as well as several links to assist the user in getting started. One of the link is the request password reset that sends an email containing a link with a token to an email address specified in the User name field so a user can reset a password. This is shown when the user tries to reset the password.

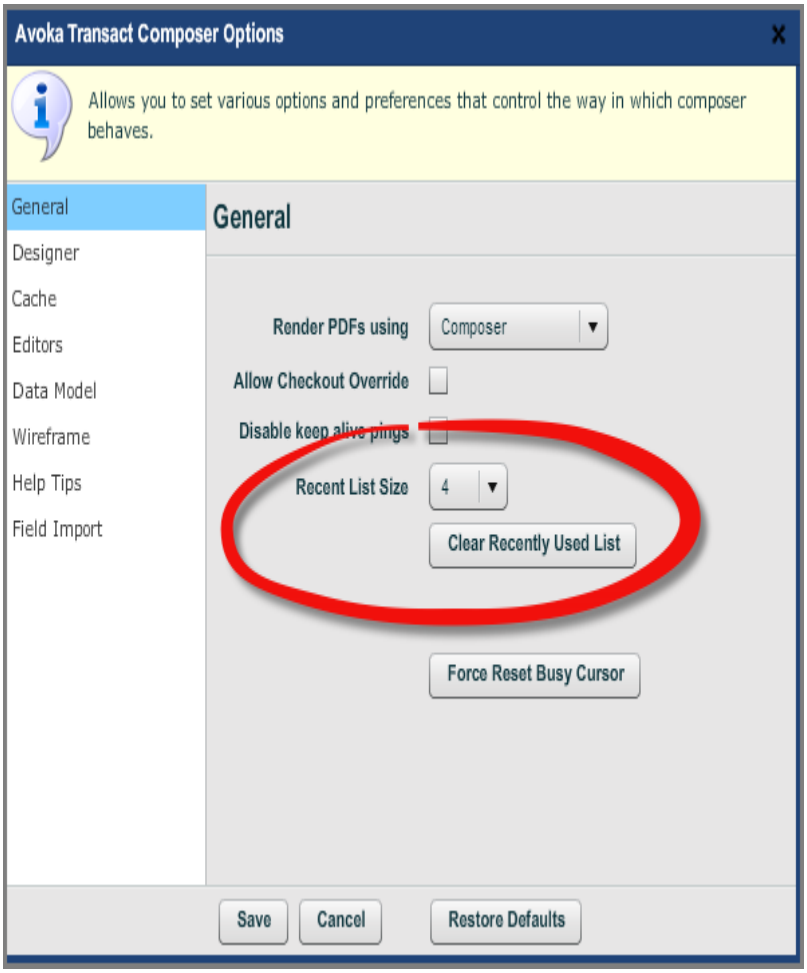
Welcome Page

The main area of the Welcome page is divided into sections described below. These are intended to make it easy for users to navigate around the product. This example is for an administrator and thus may contain options not available to other users.



Recent Forms

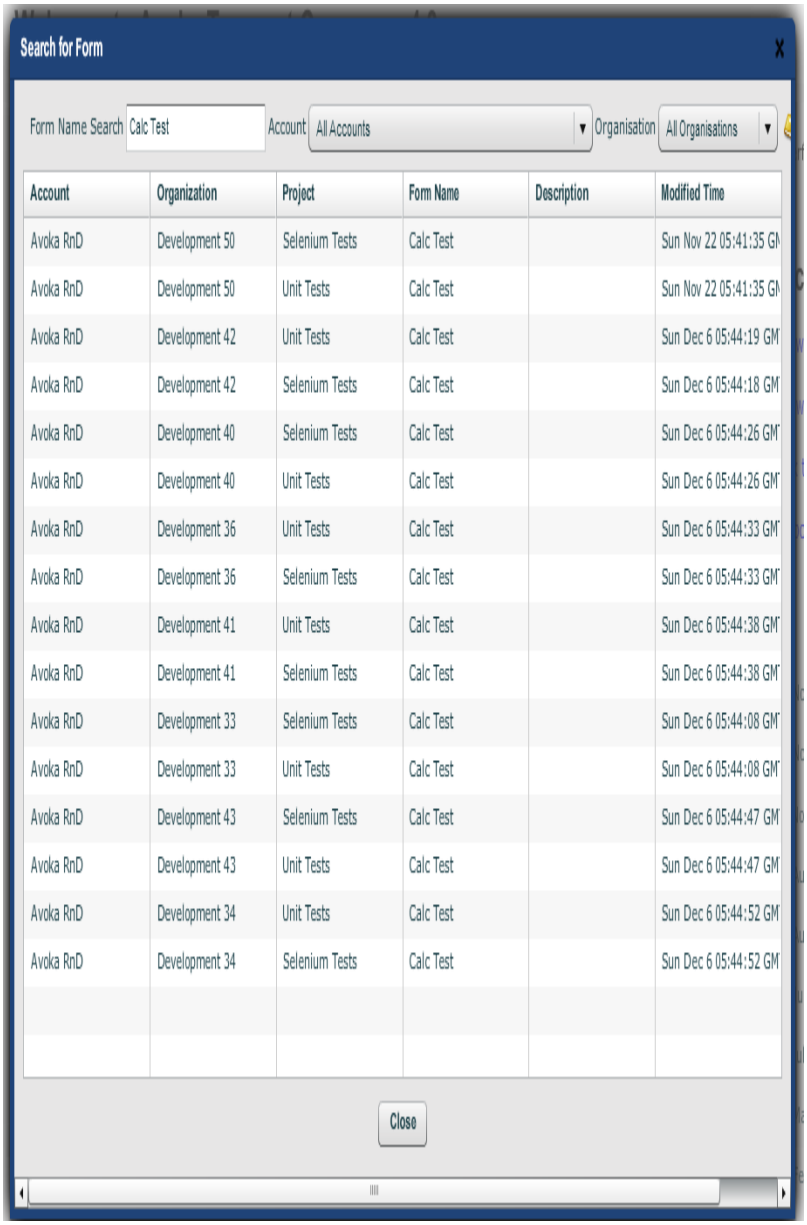
This section retains a list of most recently used forms. These are kept across sessions. The default saves the last 4 forms viewed but can be configured to store up to 10. The configuration can be changed from the "Options=>General" link at the top of the Composer Screen.



Actions

This is where you can change your password or search for a form.

Form Search allows a user to search for a forms in Accounts and Organizations they have permission to access.



Product Information

These are quick links to documentation, releases notes, Avoka Questions and the Avoka Support Portal.

News

The Composer administrator can post news of releases, advisories or other important information. This is done via System=>Deployment .

Home Menu Choices

Workspace

The heart of Composer's user interface where all the design work takes place.

Access Control

Used to control Composer Security and is only visible to users with the appropriate access levels.

System

This is used to configure Composer and is only visible to users with the appropriate access levels.

Top of Page Links



Options

The Options dialog allows a user to set the default configurations for their Composer session. The following table lists the options:

Category	Function	Default	Description
General			**These are General Configuration Options**
	Render PDFs using	Composer	Options: Composer, Adobe LiveCycle, XDP
	Allow Checkout Override	Off	Setting that will ignore the form checkout status on Open/Preview only. Save/Publish will honor the checkout state.
	Disable keep alive pings	Off	By default Composer pings the server to avoid the session being closed.
	Recent List Size	4	The number of Recently Accessed Forms retained on the Welcome Screen
	Clear Recently Used List	-	Clears the List
	Force Reset Busy Cursor	-	Occasionally the Cursor can remain as busy. This will reset the cursor back to its original state.
Designer			**Options to Configure the designer screens**
	Show Internal Fields	Off	By default internal fields are hidden on form open. This will override that setting.
	Show Names on Structure Tree	Off	By default the field label is shown in the tree. This setting will switch to showing the Internal Name of the Field.
	Enable Parts on Standard Projects	Off	This will enable the Form Parts functionality in standard projects rather than requiring a Parallel Development Project (version 4.2 or later).
	Expand Transparent	Off	This makes the contents of imported trans-

	Parts		parent parts visible in the structure tree (version 4.2 or later).
Cache			**Allows the manipulation of the local Cache size. This can assist with performance of the UI**
	Organization Cache Size	3	How many Organizations can be cached
	Current Cache Contents	-	Show the Organizations that are in the Cache
	Clear Cache	-	Clears the current organization cache
	Show Load Times	Off	This setting will display a pop up with the form load time in milliseconds.
Editors			**Configuration settings when the editor dialogs are opened.**
	Show Advanced Properties	Off	By selecting this option the advanced properties of the field are shown
	Default Units	Form Units	Allows selection of default units in the editor
	Hide Script Editor Tree at startup	Off	Forces the script editor tree to NOT be built at startup.
Data Model			
	Link Data Model Trees	On	Links tree selection in data model view.
Wireframe			**Controls how the wireframe looks and works.**
	Default Zoom Factor	110%	Changes the default wireframe zoom. This can be adjusted when the form is open.
	Show Measurement Tooltips	Off	Shows approximate measurements of a field dimensions as a tooltip in the wireframe
	Freeze Wireframe	Off	Will set off wireframe redraws
	Use embedded fonts	On	Controls whether the wireframe will use its embedded font
	Auto Refresh Problems/Dependencies	On	When selected the wireframe will auto refresh Problems and dependencies automatically
	Check Errors Before Preview	Off	
	Show Wireframe on Form Open	On	This option can restrict the wireframe from being shown at form open. This can assist with opening times on large forms.
	Show Empty Block Border	On	Setting it on will show a border around an empty block. This makes it easy to drop in items.
Help Tips			**Controls whether Help Tips are displayed**
Field			**Settings used with the Field Import Func-**

Import		tion
Show Field Import Dialog	On	
Create New Field Instead of Updating Existing Field	On	

Feedback

This link shows the dialog that provides information that includes form details, the Composer build used, the search path details and the browser details.
This information can be copied and pasted into support Jras or emails.

Online Documentation

This links to this guide in HTML. It is also a dropdown menu (click on the arrow to the right of "Online Documentation") and you have the following choice:

- Online Documentation - The link is set under System=>Deployment properties
- Release Notes - Link to Avoka knowledge base. The link is set under System=>Deployment properties
- Javascript Documentation (giving the Composer Framework — see Scripting)
- About Avoka Transact Composer (Specifies, version number, build date and number and build environment)

Logout

Logs the user out.

Accessibility for Composer Developers

Composer | Form Builder | v4.3 & Higher

Accessibility can be considered the degree to which electronic services and products can be accessed by the largest number of users possible. Although, often focused on disabled people using assistive technology including screen-readers, this is relevant to all people, including those that are on slow broadband, using non mainstream operating systems, using a mobile device or speaking non-native languages.

Since 1999 the primary international standard for website accessibility has been the Web Content Accessibility Guidelines (WCAG) developed by the W3C. This has formed the basis for accessibility legislation around the world, including Section 508 in the US and the 2000 Government Online Strategy in Australia.

Many Avoka clients care about accessibility because it is good for society and, in some sectors, compliance has been legislated.

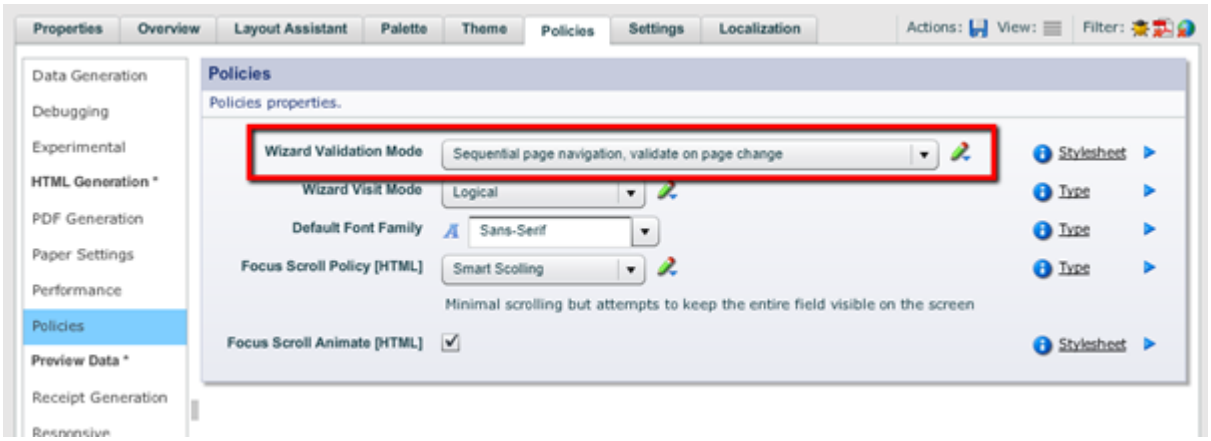
Avoka Composer provides as much of the basic, field-level accessibility as possible automatically. For instance, linking a field's caption text with its input. This document provides a guide of the additional tasks that a Composer form developer must perform to achieve maximum accessibility.

Checklist:

1. Use Sequential page navigation
2. Use Section headers
3. Mark-up lists
4. Add alternate text to images
5. Do not use Background Image
6. Use Fieldsets
7. Use Caption Placement Top
8. Use Described By
9. Check color contrast
10. Do not use Placeholder Text
11. Do not use a separate Text for field captions
12. Do not disable anything

1. Use sequential page navigation

Structure > [Root] > Edit Properties > Policies > Wizard Validation Mode

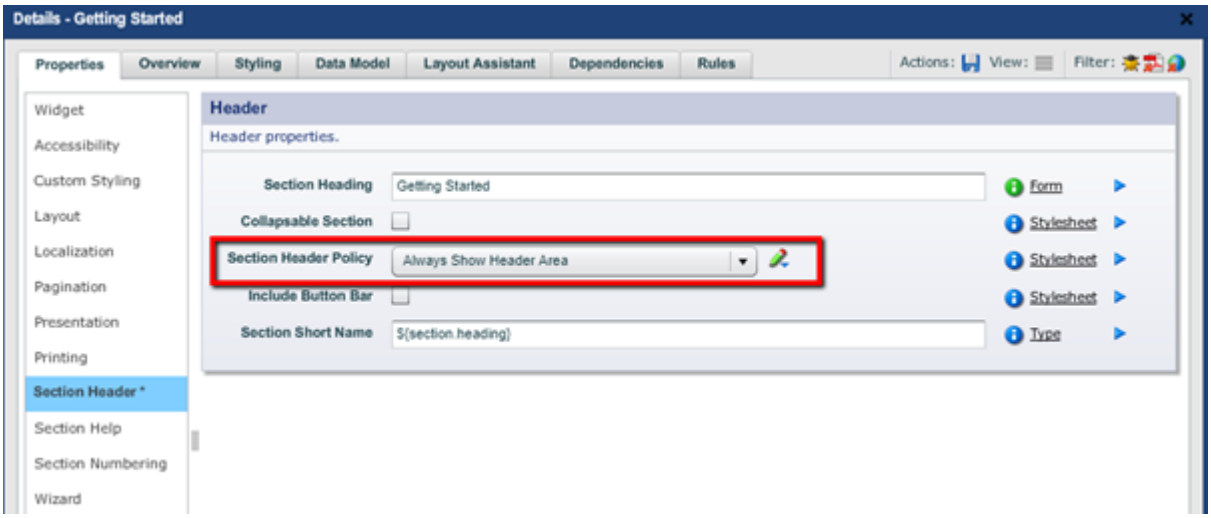


This policy property defines how validation and navigation operates for the form as a whole. Sequential page navigation is better for accessibility as it provides a simpler experience where users navigate through the form sequentially, one page at a time, and does not allow them to progress until the current page passes validation.

Unconstrained page navigation allows the user to navigate to any page, in any order which can be confusing (particularly on mobile devices) and does not perform full form validation until they attempt to submit from the final page. Having to then potentially correct issues across multiple pages can be a challenge.

2. Use section headers

Structure > Section > Edit Properties > Properties > Section Header > Section Header Policy

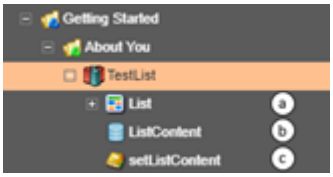


Correctly marked-up headings are provided automatically by the form header and sections at levels 1, 2 and 3. Use section headers and do not manually create them with text objects.

3. Mark-up lists

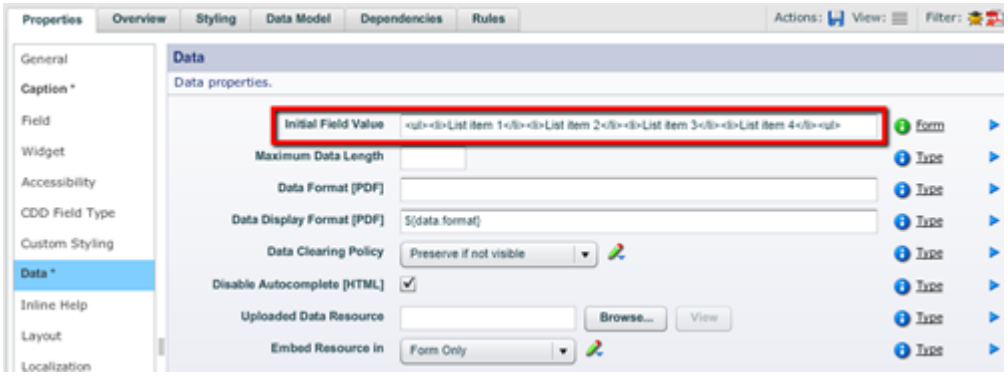
Lists that are created in the rich text editor are visually represented as a list but are not marked-up correctly in the HTML. See the example below that demonstrates a technique for providing marked-up lists.

Example list structure:

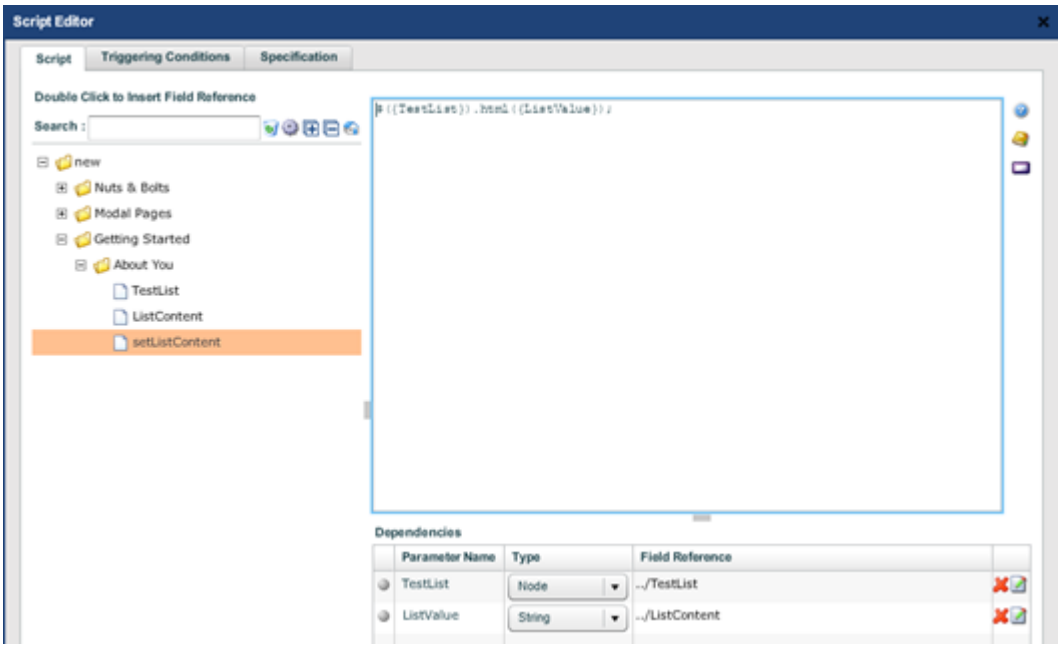


a) TestList - Create a rich text placeholder for the list with no content

b) ListValue - Create a data field with the list HTML mark-up as the initial field value:

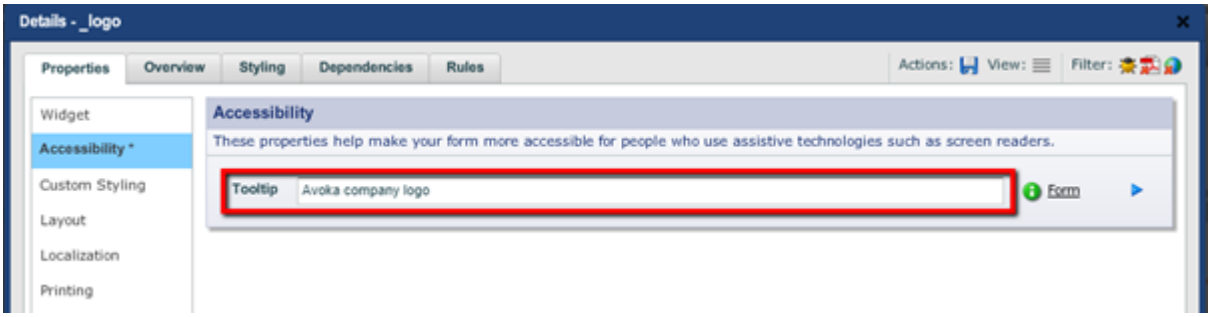


c) setListContent - Create a general purpose business rule that triggers on Initialization and uses JQuery to set the ListValue HTML onto the TestList:



4. Add alternate text to images

Image > Edit Properties > Accessibility > Tooltip



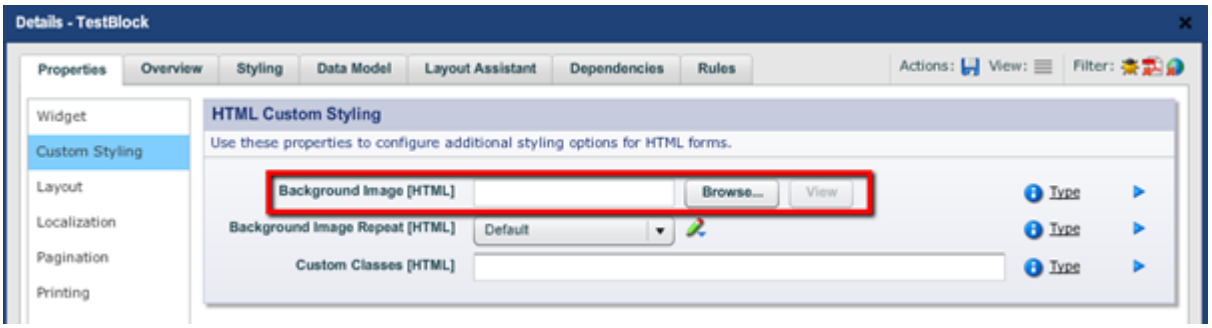
Images can be informative or decorative. The form developer must use their judgement on whether an image should be treated as decorative or informative:

Decorative images do not convey information to the user and are typically used to make the form more visually attractive by adding visual components such as separators and textured backgrounds. An image is also considered decorative if it conveys information that is also fully provided by surrounding text. Decorative images should not have alternate text.

Informative images do convey information to the user that is not provided by surrounding text and should have alternate text that conveys the meaning or content of the image. The alternate text need not be a literal description of the image and should be as succinct as possible.

5. Do not use background images

Component > Edit Properties > Properties > Custom Styling > Background Image [HTML]

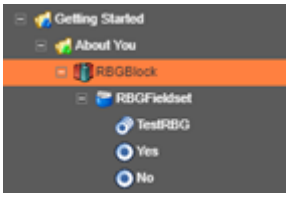


Informative images should not be used as a background image as it is not possible to provide them with alternate text and some browsers remove background images when put in high contrast mode. It is fine to use decorative images as a background image.

See section 4. Add alternate text to images, for a definition of informative and decorative images.

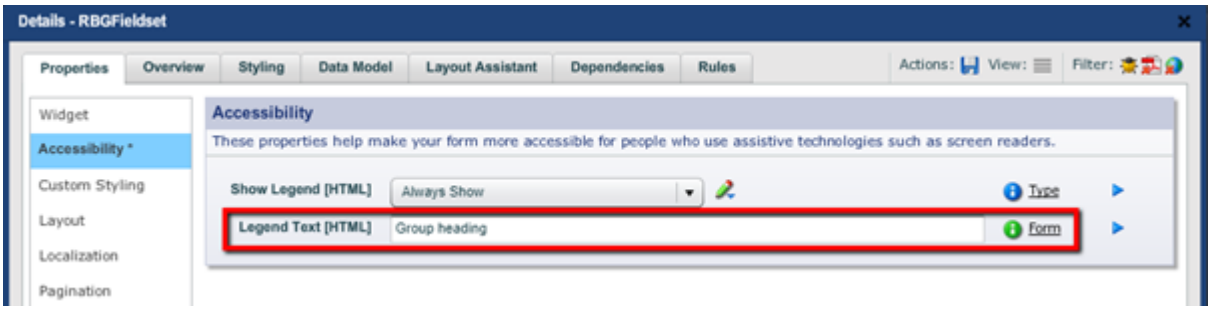
6. Use fieldsets

Fields that are not stand-alone and only make sense in the context of a group of fields should be contained within a fieldset. For instance radio buttons, telephone region code and number and address fields:



A meaningful description of the group should be provided in the fieldset's Legend Text which is usually displayed on-screen as a heading:

Fieldset > Edit Properties > Properties > Accessibility > Legend Text [HTML]

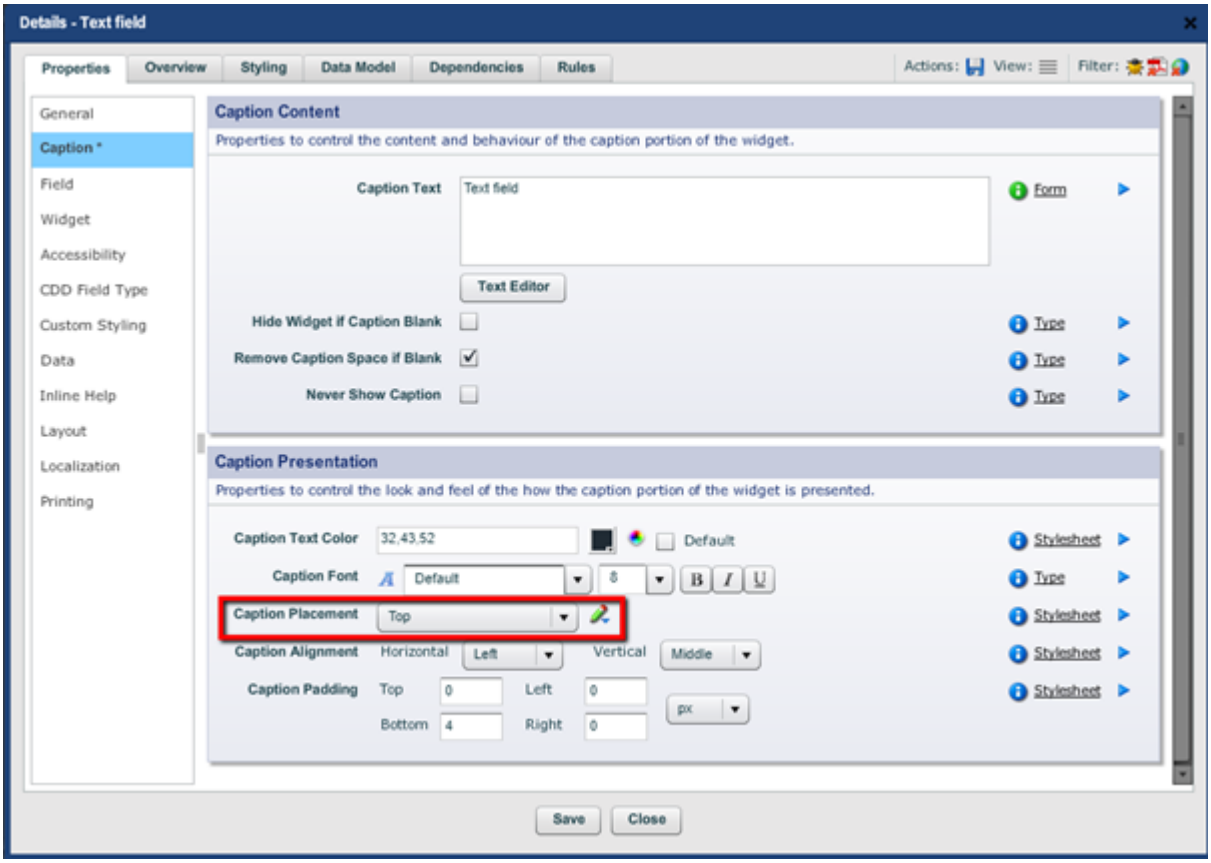


The legend text will be used by assistive technology to inform the user of context for fields within the group. For instance, tabbing to a radio button whilst a screen-reader is running might inform the user that they are currently focused on a radio button with caption 'Yes' for the fieldset 'Would you like to purchase insurance?'.

Without the fieldset, the user would only be informed that they are focused on a radio button with caption 'Yes' which is clearly insufficient. To avoid repetition, the majority of assistive technology will only inform the user of the fieldset legend for the first field focused within the group, regardless of which field that is i.e. not necessarily the first field in the group by position.

7. Use caption placement top

Field > Edit Properties > Properties > Caption > Caption Placement

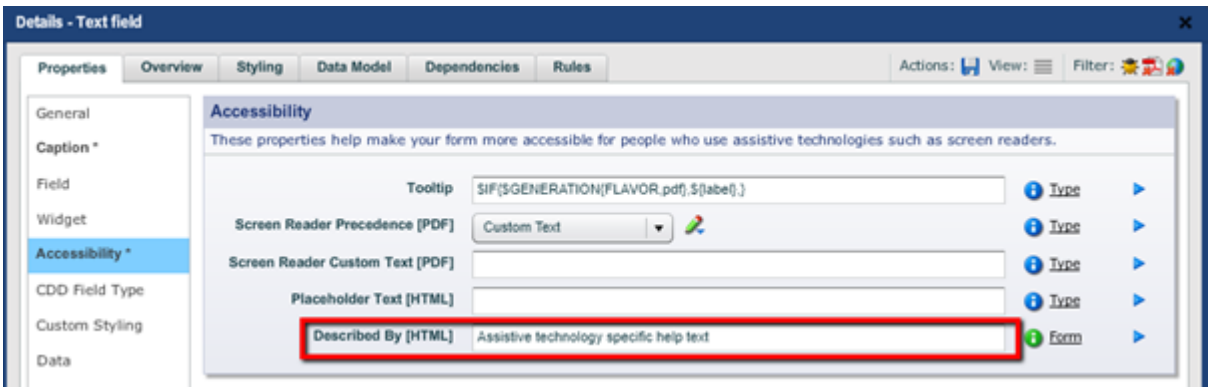


'Caption placement top' is best for accessibility as it keeps the caption closest to its field. In particular, this makes it easier for users of magnifier software to locate the field associated with a caption without having to move the cursor too far and reduces the chances of finding the wrong input by mistake.

Top placement also provides the best scan line between captions and inputs for fields. A scan line refers to eye movement along a straight line as a sighted user browses a form or searches for a specific field. Other caption placement results in additional, unnecessary movement as the sighted user's eye jumps around between captions and inputs that are less aligned.

8. Use described by

Field > Edit Properties > Properties > Accessibility > Described By [HTML]



Use the 'described by' property on fields to provide additional information to users of assistive technology that is not required to be displayed on-screen for sighted users. This is a useful technique for helping users of assistive technology work with non-standard fields where the behaviour is obvious to sighted users e.g. a field will auto-forward focus to the next field once it has been filled-out and so the user need not tab.

'Described by' cannot be used to reference another visible component on the form.

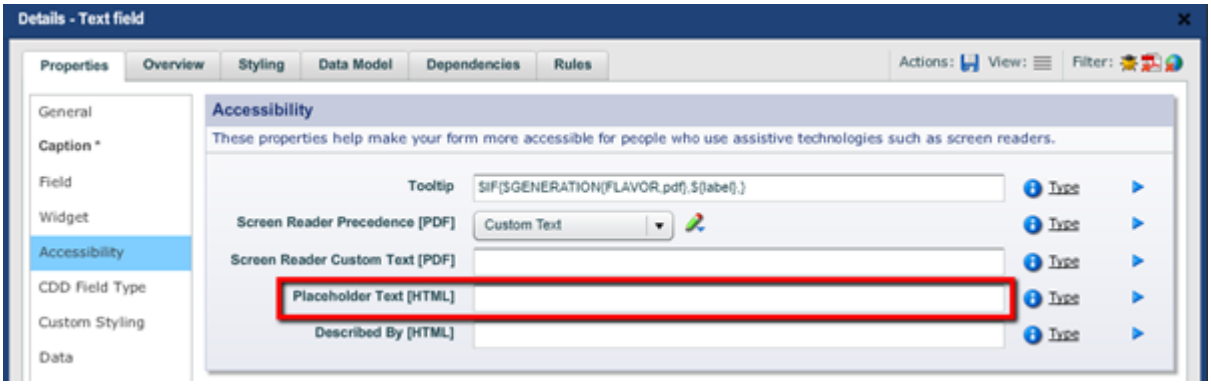
9. Check color contrast

Colors used in a form must meet accessibility requirements for contrast of foreground and background objects for partially sighted and color-blind users.

You can check your form using a contrast tool such as the colour contrast analyser from Vision Australia.

10. Do not use placeholder text

Field > Edit Properties > Properties > Accessibility > Placeholder Text [HTML]

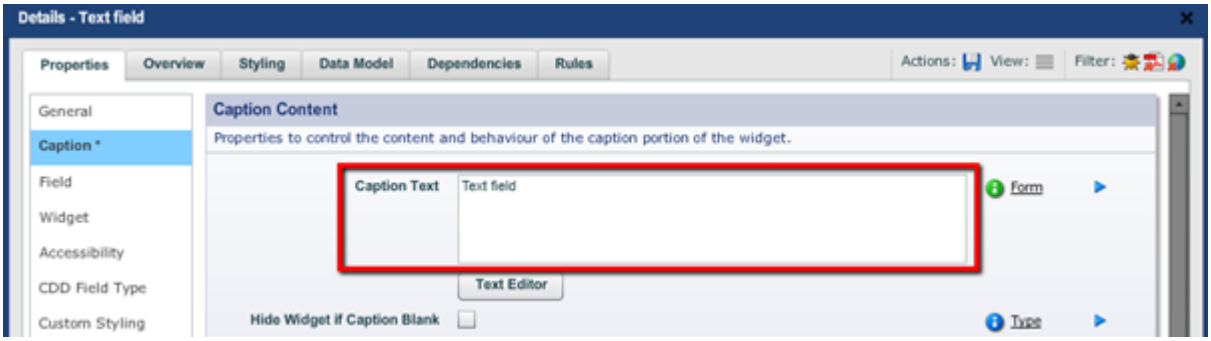


Placeholder text is the light grey text that is displayed in a field until data is entered at which point it is removed.

Placeholder text is not accessibility compliant due to the lack of consistent support in assistive technology. Some assistive technology ignore placeholder text entirely. Some always notify the user of the placeholder text regardless of whether there is data in the field or not and makes no differentiation between the placeholder text versus the entered data. Only a small proportion of assistive technology do the correct thing in only notifying the user of the placeholder text when it is visible on-screen.

11. Do not replace field captions with text

Field > Edit Properties > Properties > Caption > Caption Text



It can be tempting to hide the built-in field caption and create a replacement with a separate text object in order to achieve layouts such as a caption that is wider than its field.

Although, this approach may look identical on-screen to sighted users, this does not provide the browser functionality where selecting a caption sets focus to its field.

To resolve the specific case of long captions, it is better for accessibility to use a combination of shortening the caption, widening the field and moving non-critical information from the caption to a popover tooltip or a message box located close to the field.

12. Do not disable anything

Disabled form content is problematic for accessibility and should be avoided.

Typically, disabled content is visible on-screen for sighted users but not active and visually indicated as disabled with grey-out styling. Disabled content is not discoverable for uses of assistive technology which results in a mismatch of information. For instance, in the case of disabling the continue button for a page until certain actions are performed a sighted user can see that the button is disabled and may be able to deduce that they need to perform an action before it will be enabled.

However, a user of assisted technology will have no knowledge of the existence of the continue button at all which is likely to lead to confusion of what is expected for them to progress.

In this case, it is better for all users to keep the continue button enabled and inform the user what is expected of them with error messaging if they attempt to progress without completing necessary actions.

The other common alternative approach to disabling components is to hide them.

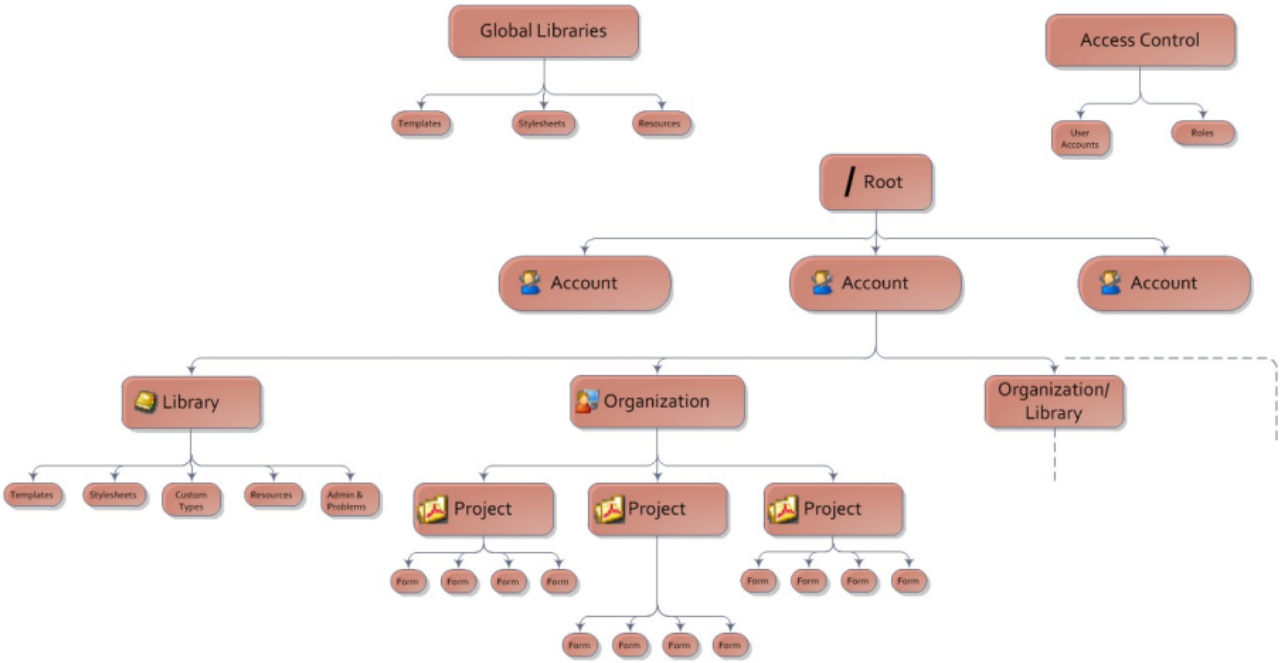
The Composer Workspace

Composer | Form Builder | v4.3 & Higher

The Workspace in Composer is the container for all forms and their environments. Libraries are integrated into the workspace and their allocation and sharing across projects are now dynamic and efficient. The implementation of compatibility with older versions, of patching and of service pack installation is all available to authorized users.

The Hierarchy and Environment

Workspace Hierarchy



The Workspace divides into "Accounts". Accounts can have several "Organizations". Each Organization has several "Projects". Each Project has its set of forms. In this model, forms are not dynamically shared between projects but can be between Organizations.

When designers require personal workspaces, they do this at the Project level.

All forms can be saved to version control and can be rolled back to a previously versioned revision, if required. If designers delete forms, they are actually not deleted from the Composer database and can be recovered.

If a form is open in Composer by one designer, other designers are locked out. There is a form lock override available however this needs to be used with caution to ensure changes are not lost.

Each Account can have a number of Libraries which are dynamically shared at the Organization and Project levels. Administrators are free to allocate whole Libraries as they see fit, or the components of Libraries.

Merits of this Structure

Composer caches Organizations allowing faster navigation times around the Workspace. By default, Composer caches 3 organizations. This is configurable using the Options=>Cache.

Both Global Libraries and Access Control sit above the Account hierarchy. Global Libraries can be shared dynamically across Accounts, allowing — if desired — patching to these Global elements without intervention by end users or administrators.
Users can be given access to several Accounts, and within those be given access to different functions. Libraries are shared at the Account level, and the one Account can have several Libraries so that branding can be controlled across the Enterprise in both similarity and difference from department to department.

The Composer Hierarchy

Accounts

An Account is the top level of the Composer tree hierarchy. It holds one or more Organizations which in turn can contain one or more projects. Most users only belong to one Account.

The screen below shows the Account "Avoka RnD". It houses several Development Organizations.

Organizations

Organizations are a way of splitting up the account into entities that match the client's business. For example this may be by department or distinct business units. Users can belong to one or more organizations depending on the way their permission set. The following screen shot shows the organization "Development 42". It has expanded to display its underlying projects.

There are several capabilities that are accessed at the organization level.

Asset Type	Description
Projects	A grouping for forms. To create a new project, click on the Create New Project button.
Style Sheets	A list of style sheets available for your organization. Many of the standard style sheets will be inherited from Composer default style sheets, and these cannot be modified. If you want to modify one of these style sheets, you must first make a copy of it. For an explanation of style sheets, see this topic: Overview

Custom Types	Custom types are the widgets or blocks that have been created for your organization. You can modify or delete these types if you have the required privileges. Some types available to your organization are not modifiable. This are either types that are inherited from Composer defaults, or complex types that are not editable using the standard block editor. You can view these types by clicking the "Include non-editable types" checkbox. **Warning**: Be very careful of modifying or deleting custom types, as they may be in use by multiple forms. Always perform an impact analysis before modifying. If in doubt, make a copy of the type rather than modifying it. For more information on Impact analysis, see this topic: Impact Analysis
Resources	A resource can be any file used in your forms. Typically these are images and JavaScript libraries.
Templates	A template allows you to customize the form-design experience and the forms that are produced by Composer. This is for advanced users only.
Administration	This contains the next 3 sub-tabs **Assigned Users** A list of users assigned to this organization. Administrators can create new users, add and remove users from the organization, and change their roles. Users are global to the Composer instance. **Note**: Depending on the way that your organization is set up, you may need to add users to individual projects as well as to the organization. **Note**: It is strongly suggested to use a user's email address as their user-id. This avoids possible name conflicts. **Property Sets** This is a list of the set of properties that can be used on the Bulk Editor page. **Settings** This tab displays several settings about the current user and organization, including: • publish targets • the ability to perform impact analysis on widgets, blocks, style sheets, and templates.
Problems	This shows any issues in the organization.

Projects

A project is a convenient container of a number of forms - usually forms get created in groups or batches, and a project is a simple way to group them. Restrictions can be placed on which users have access to each project. Click on the Create New Project button at the organization level to create a project, and follow the wizard steps.

There are several capabilities that are accessed through project level tabs:

Asset	Description
Forms	A list of forms in this project. You could double-click on the form in the list in order to open it. You can perform a number of actions on a form: Open Form: allows form editing. You can do this by clicking the icon, or by double-clicking. Open Form - No Wireframe: Opens a form for editing with no wireframe shown. This is especially useful for large forms that may take a long time to open. Update Form Details: Modify a form's properties, including its name and description. Save As: Make a copy of a form. Delete Form. Preview Form. The rendition types and defaults are controlled by the template being used. Unlock form. (Administrators only.) This allows a form that has been opened by another user to be unlocked. Use this with caution, as the current user will no longer be able to save their changes. Publish form. Publish the form to Transaction Manager or as a zip file. View Form Statistics. Various statistics about the form such as how many fields, dependencies and so forth. Manage Form Revisions: This can be used to create new and use old form revisions, for example, revert to a previous version. Generate Form Specification. **Note**: You cannot open a form that is currently being edited (i.e. "checked out") by another user, who is identified in the "Checkout by" column. **Note**: if you open a form in a Read-only organization, you will be able to modify the form, but not save it.
Assigned Templates	The templates available to this project. Advanced users only.

Assigned Users	The users that are assigned to this project. The user must first be assigned to the organization.

Creating a Form

To create a new form, click on the Create New Form button in the project's toolbar, and follow the wizard's instructions. The wizard content can change depending on the template that has been selected. The Avoka Maguire template adds a screen where the form developer can choose various functions that can be included directly in the form.

Here are the steps to the New Form Wizard:

Step	Description
1. Name and Description	Enter the name and description of your form. Choose the name carefully, as this will appear in the form header by default (this can be changed if desired). The description is not used within the form, and is used only to describe the form to other users of Composer.
2. Template	Select a template. This will vary based on how your organization has been set up. Most organizations will only have a single template.
3. Additional Values	Select additional values, such as the heading lines of your form. These will vary depending on how your template has been configured. Some of the fields entered here will appear on the header or footer of your form, and some options may control they way in which your form is generated. There are a number of special values that may be used: • $FORM{NAME} - this is the name of your form as you typed it in the first page of the wizard, and as it appears in the list of forms in your project. Using this will ensure that the name of your form in Composer matches the name at the top of your form. However, this may not always be what you want, so feel free to delete this formula, or change it later. • $GENERATION{DATE} - this is the most recent date and time at which the form has been generated. This is used as the default form identifier, because it will always increase whenever a new version of the form is generated. However, you may choose to set your own form identifier, such as "VA 22-1995".

Click Finish to complete the wizard, and your form will open for editing.

Composer Form Revisions

Composer | Form Builder | v4.3 & Higher

The Composer Form Revision functionality allows an organization to create and manipulate revisions of a form.

It offers the ability to:

- Activate any revision belonging to a form.
- Make a revision read only.
- View the XML of any revision thus eliminating the need to activate a revision temporarily to view it.
- Assign a name to a revision. This can be used for versioning revisions. The name assigned is also shown in the form list.
- Create a revision when the form is published.

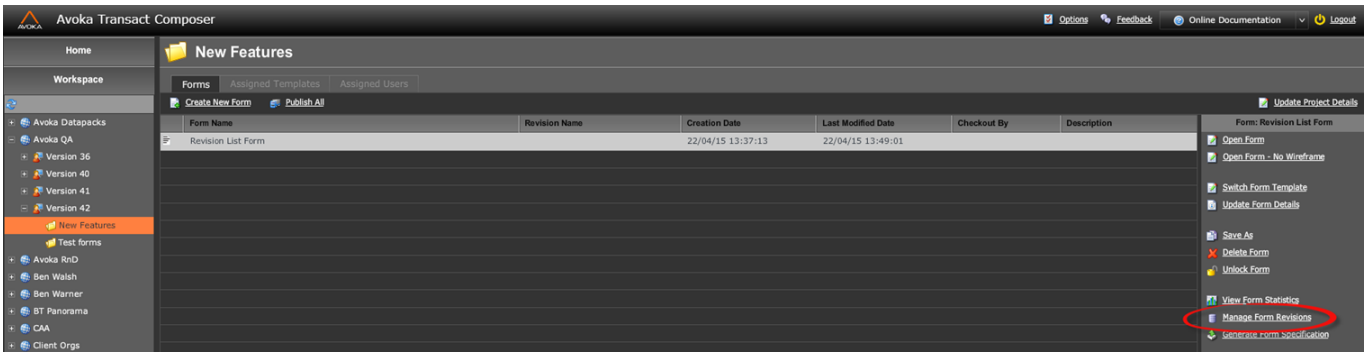
Manage Revisions Functionality

Accessing the Form Revision Dialog

The Manage Form Revision functionality is accessed from the form list screen by selecting a form in the list and choosing the option on the right hand menu.

Menu

The screenshot shows the "Revision List Form" with the "Manage Form Revisions" option circled.



Screen 1 - Form List Screen showing Form Menu items.

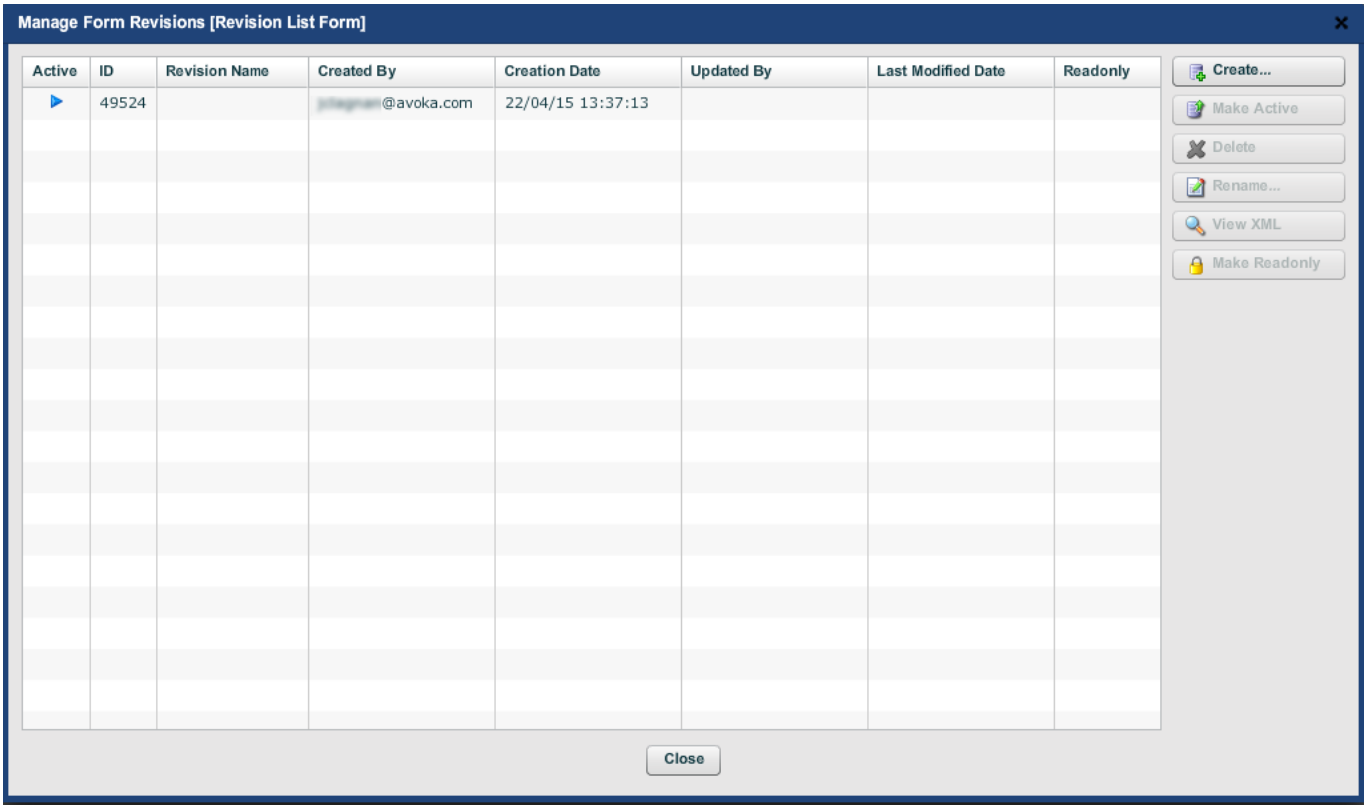
Manage Form Revisions Dialog

A newly created form will only have one form revision, however a form can have multiple revisions. The Manage Revision dialog is shown in Screen 2

The dialog shows the following columns (all columns can be sorted):

- Active - The indicator will appear against the form revision that is currently active. When the form is opened it will be this revision that will be used.
- ID - The identification number of the form revision. This is system generated.
- Revision name - This is a freetext field, non-unique name that can be updated using the Rename Option on this screen.
- Created by: User id that created the form revision
- Creation Date: Date and time when revision was created
- Updated By: User id that updated the form revision
- Last Modified Date: Date/Time when form revision was last modified.
- Read Only: This column will show a lock symbol when the revision has been made read only. This means that if the form is made active it can not be modified.

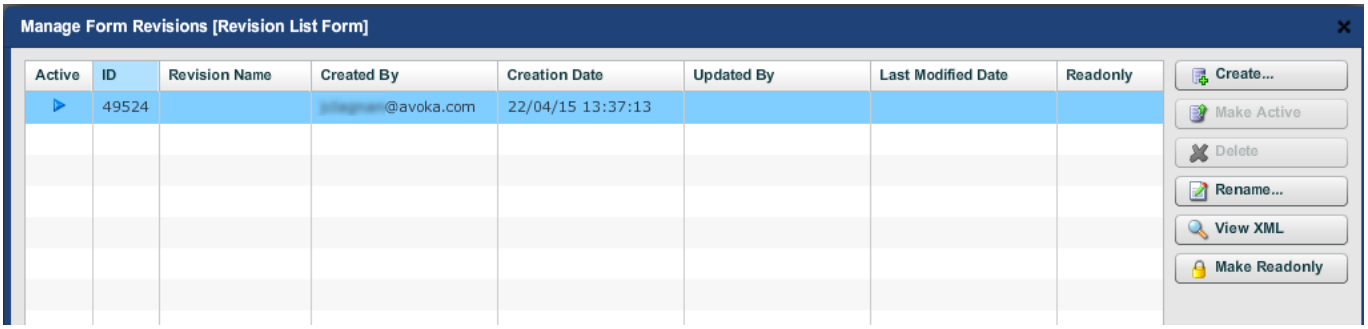
Sorting: All columns can be sorted in ascending or descending order.



Screen 2 - Revision Dialog

Form Revision Options

When a revision is highlighted several options are available. Some options are disabled depending on the state of the particular revision.



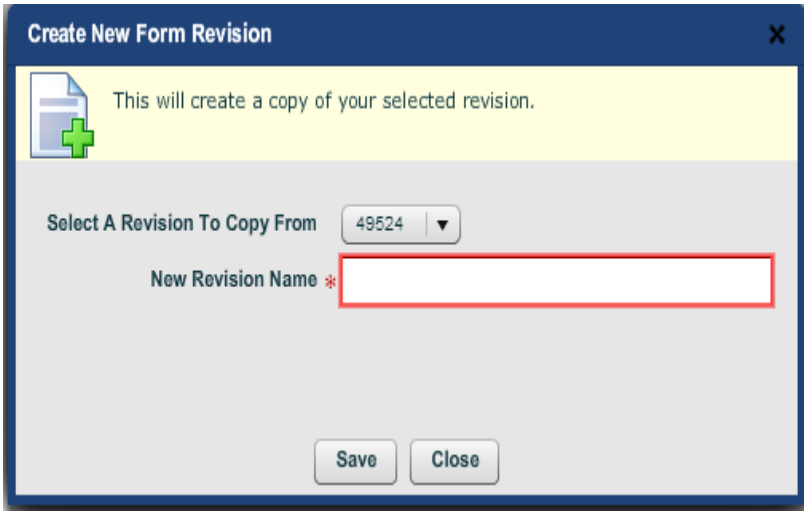
Screen 3 - Revision Dialog: Options highlighted

Create

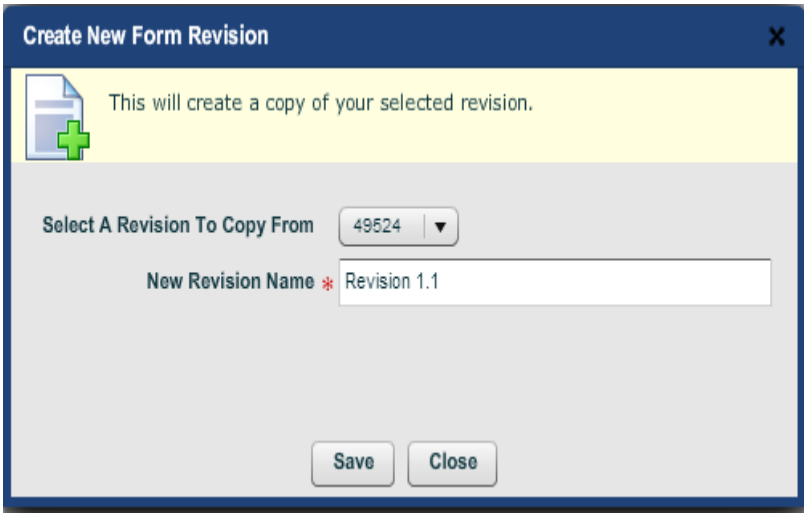
The create option allows a user to create a new form revision based on any of the form revisions in the list.

Option allows:

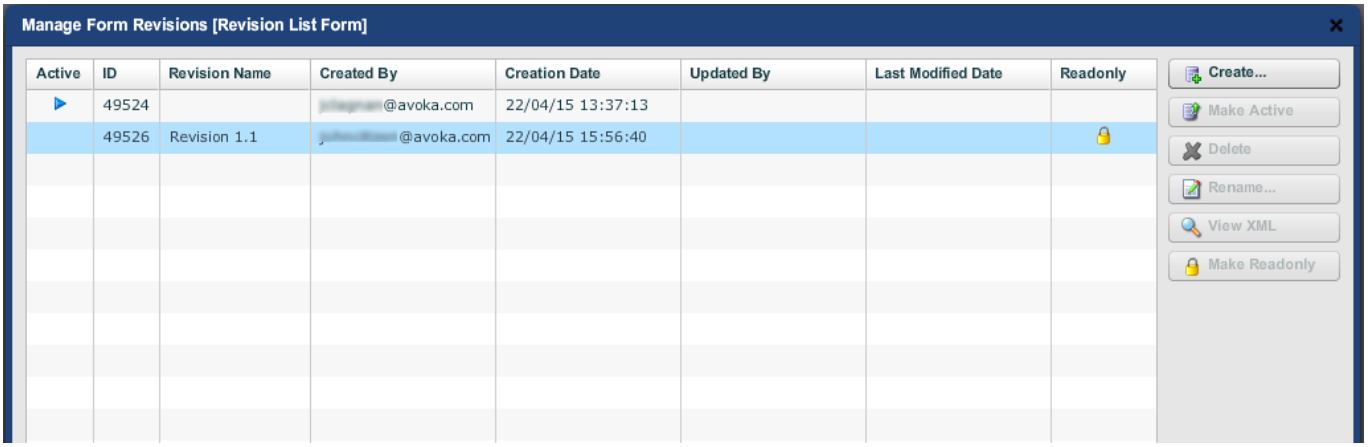
- a user to specify an existing revision ID to copy the newly created revision from. The dropdown list will show all available revision ids for the form.
- Enter a name for the revision. This is a mandatory freetext field that can be used for versioning or informational purposes.
- The newly created revision is:
  - automatically made read only
  - Is NOT made active. The activation of the the new copy is a separate manual step.



Screen 4 - Create Revision Dialog



Screen 5 - Create Revision Dialog with revision name entered. It is copied from the current active version ID 49524



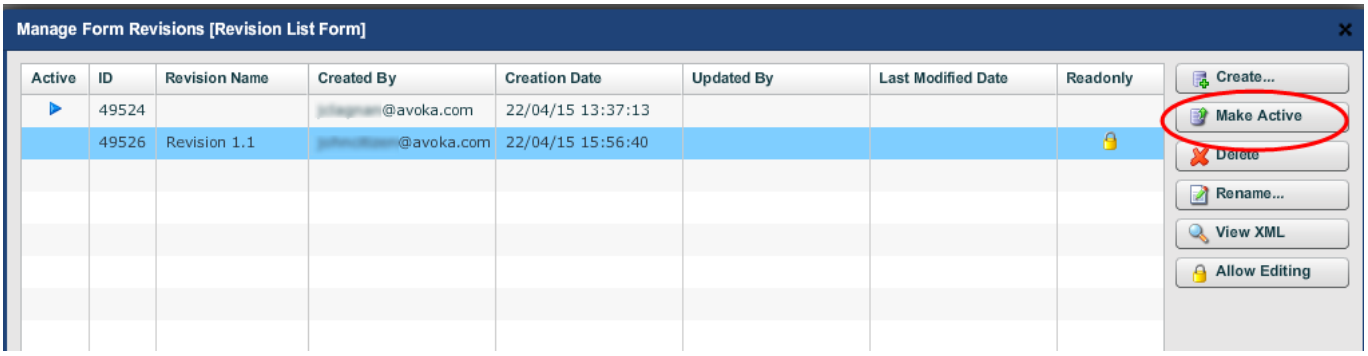
Screen 6 - Revision Dialog with newly created revision name. The new revision is read-only and NOT active.

Make Active

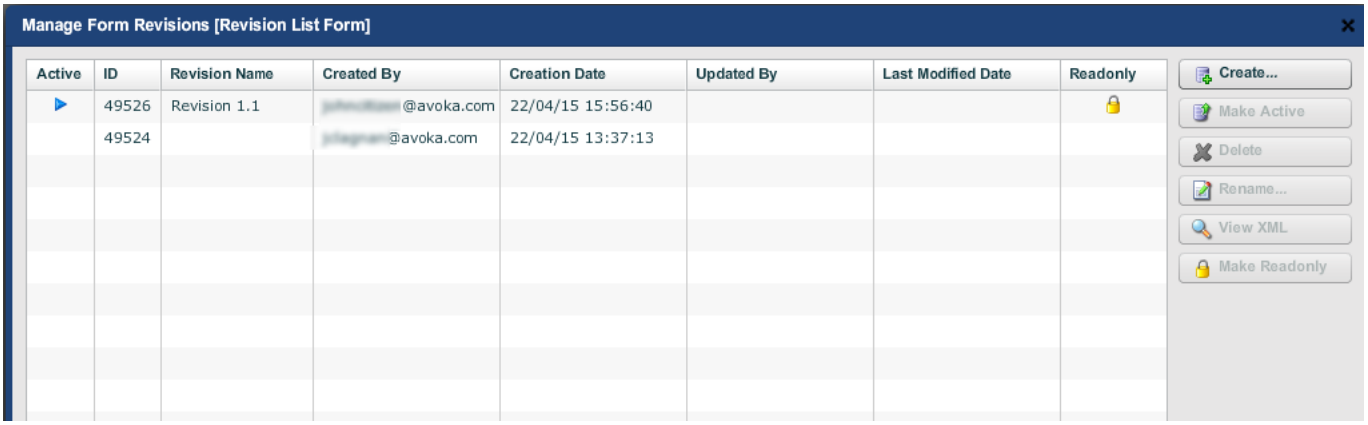
This allows a user to make a revision active. The definition of an active revision is the revision that is shown when the form is opened.

Option Summary:

- Only one revision can be active for any form.
- The active revision is highlighted by the blue indicator in the active column.
- The "Make Active" option is disabled when a revision is already active.



Screen 7 - Revision Dialog showing make active option.

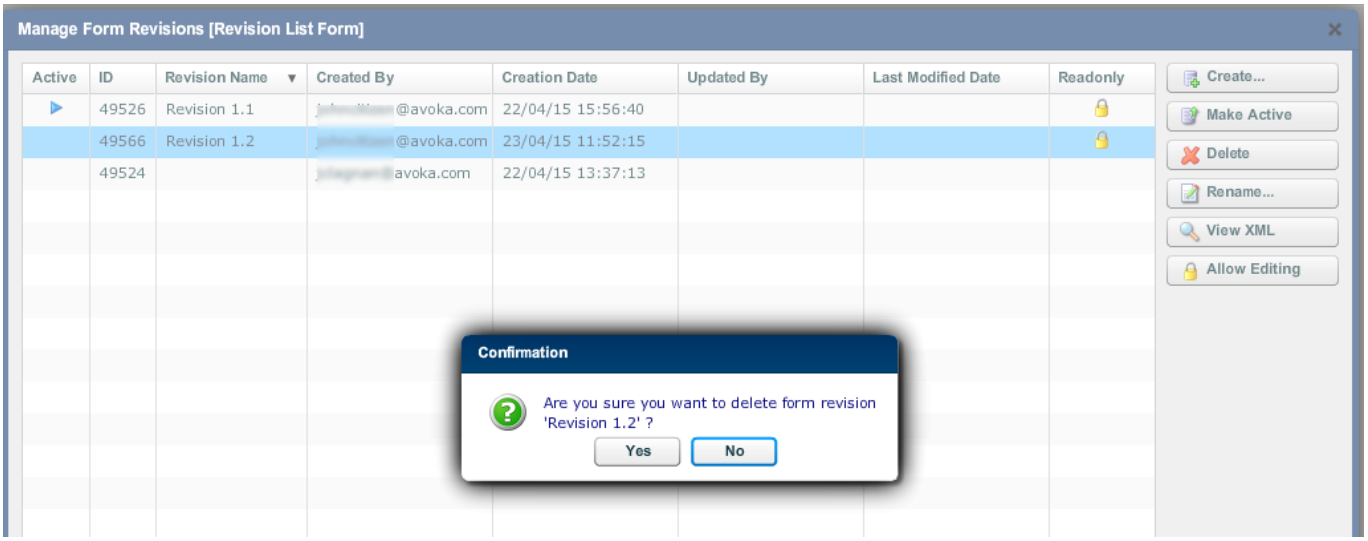


Screen 8 - Revision ID 49526 has now been made active denoted by the blue indicator. The newly activated revision is now sorted to the top of the list.

Delete

This option will delete a revision from the list.

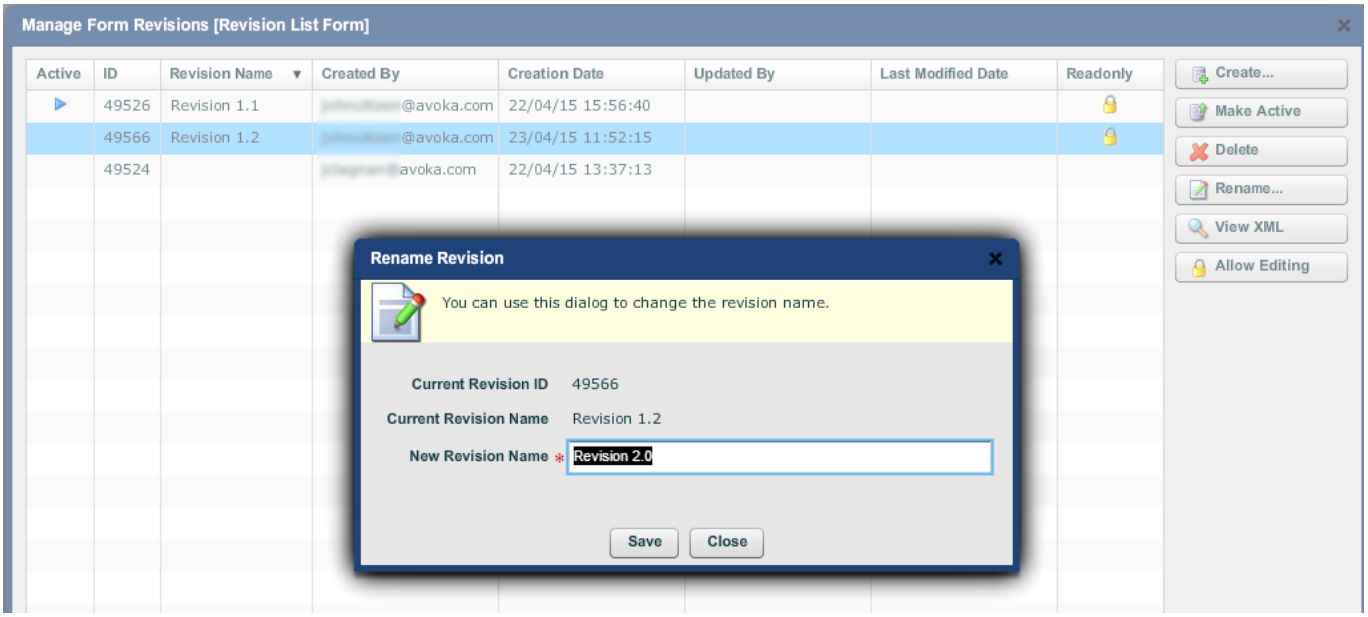
Active revisions can not be deleted. The delete button will be disabled.



Screen 9 - This shows Revision 49566 being deleted. A confirmation is requested before the deletion takes place.

Rename

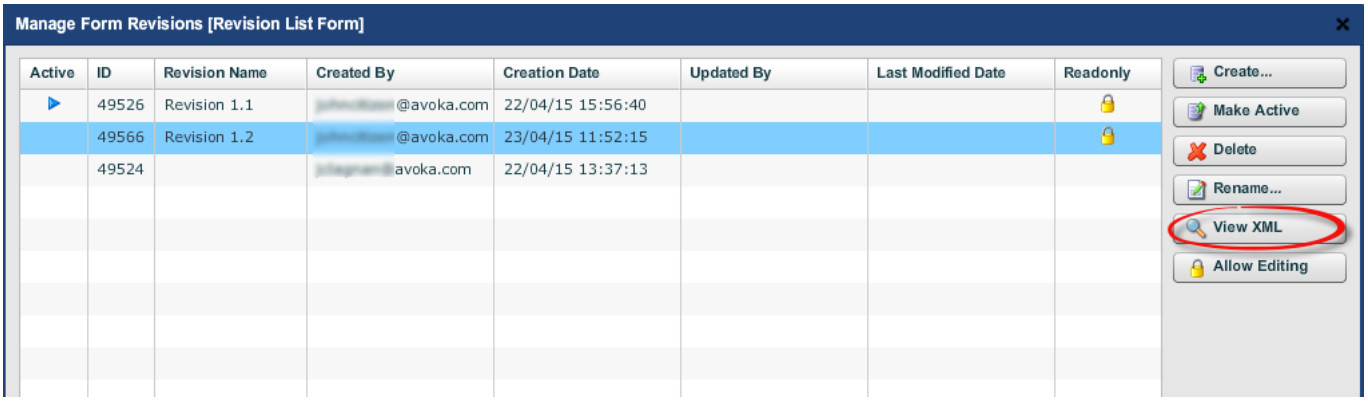
This option allows you to modify the Revision Name text. The revision name is NOT unique and is simply free text.



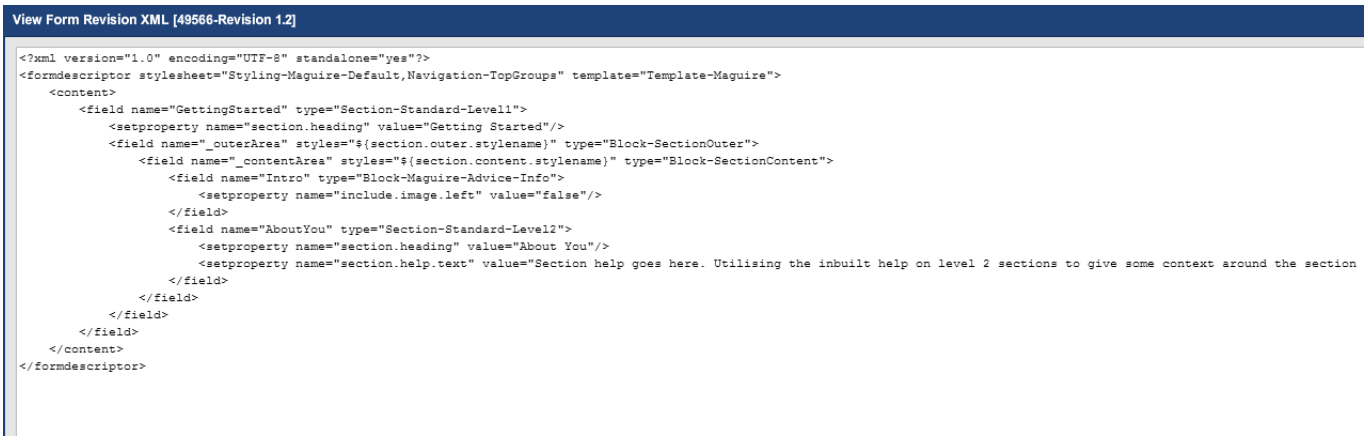
Screen 10 - This shows Revision Name "Revision 1.2" renamed to "Revision 2.0".

View XML

This option allows you to view the XML of the selected revision in the list. The revision does not have to be active. This eliminates the need to make the revision active and then open the form in the form editor to view the xml.



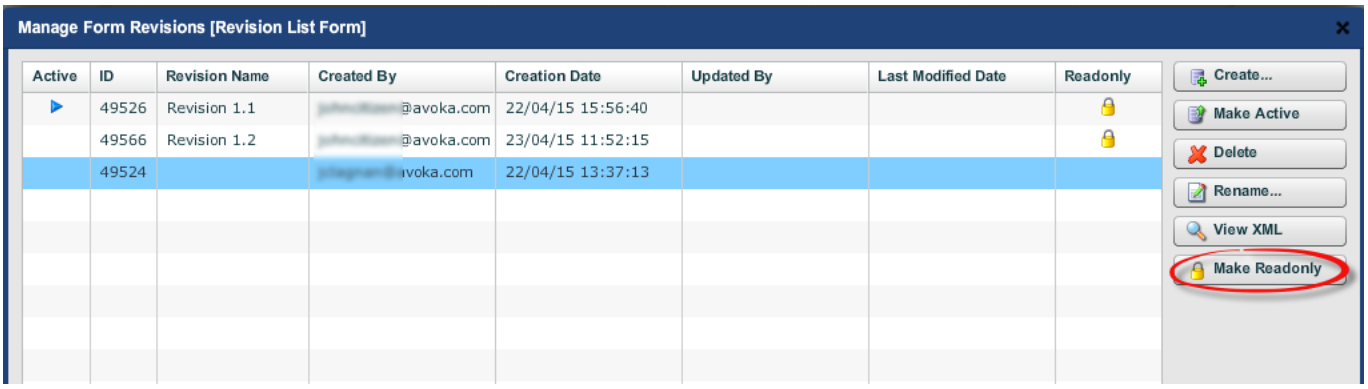
Screen 11 - A non-active revision is selected to view its XML.



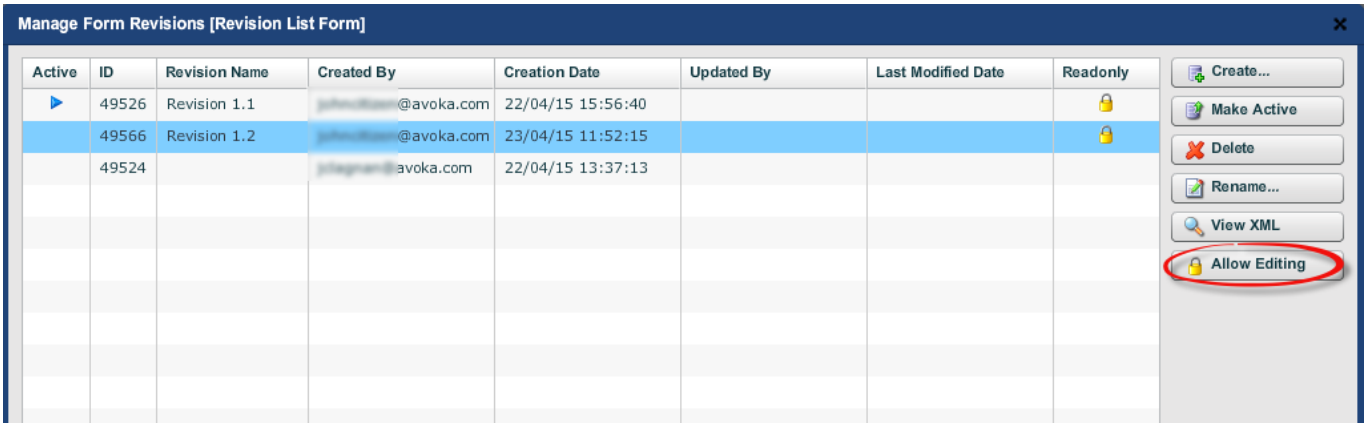
Screen 12 - The XML is shown in a new window with the revision ID and name specified at the top.

Read-Only Option

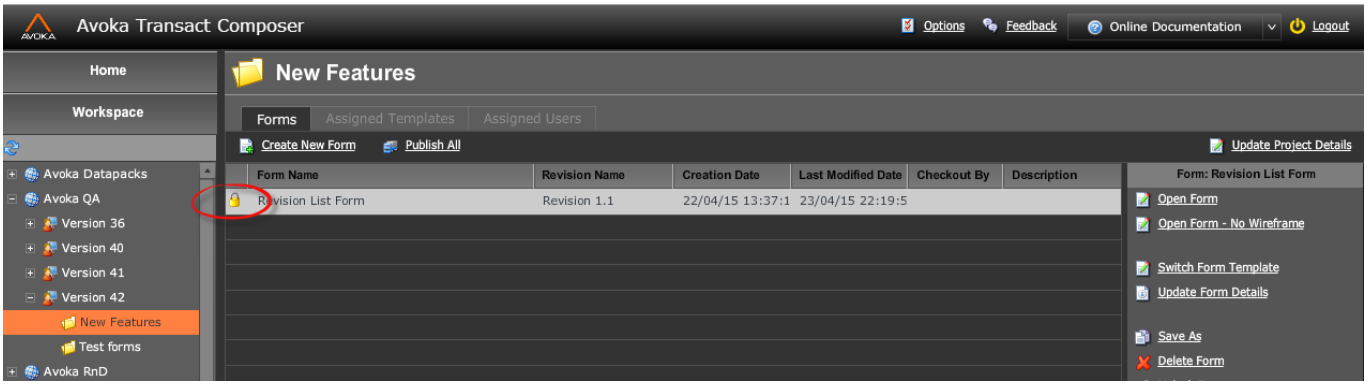
This option allows a user to toggle a revision's setting from read-only to editable. If a revision is set to "Read Only" and is the active revision, when the form is opened it can not be saved. This preserves the integrity of the revision. A read-only version is denoted by a lock in the read-only column.



Screen 13 - The selected revision ID can be made read-only by selecting the option.



Screen 14 - The selected revision ID is already read-only. The option has now change to "Allow Editing".



Screen 15 - The form list screen shows whether the form is read-only by a lock symbol next to the form.



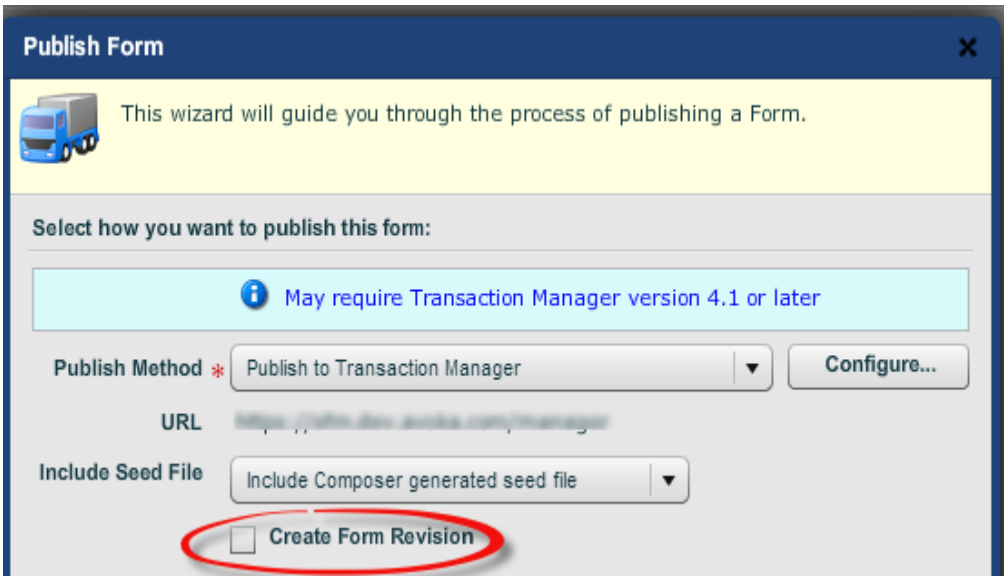
Screen 16 - A Read-only form when opened has the "Save" Button removed, a lock symbol next to the form name and the Form editor header colored red.

Creating a Revision at Form Publish

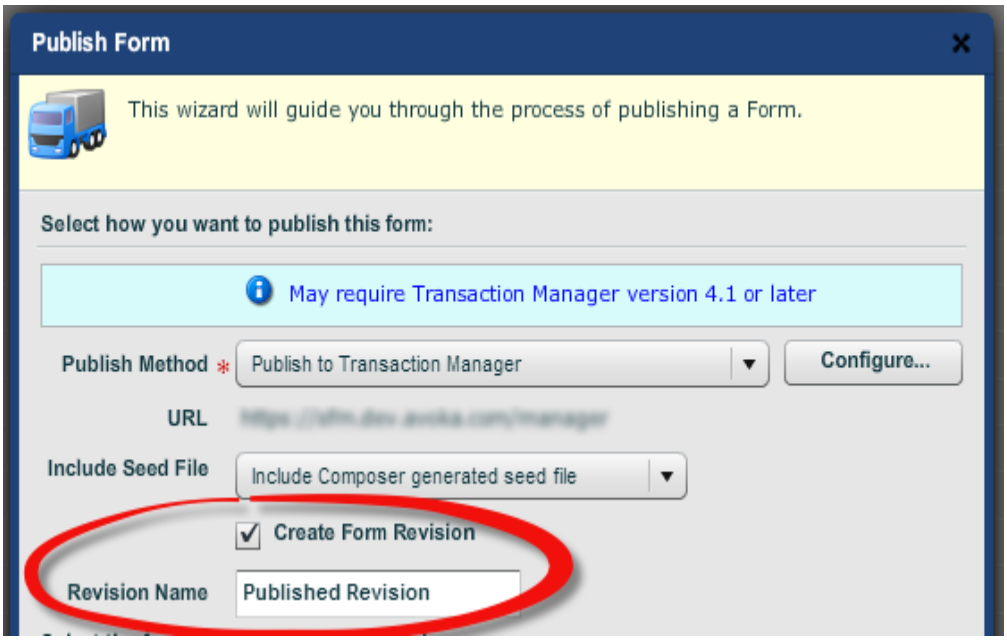
An option has been added to the publish dialog that will create a form revision at the time of publish.

Parameters:

- The option is defaulted to off and needs to be selected by the publisher.
- When selected the revision name field appears. The publisher can change the default value of "Published Revision".



Screen 17-The publish screen showing the option to create a form revision. It is off by default.

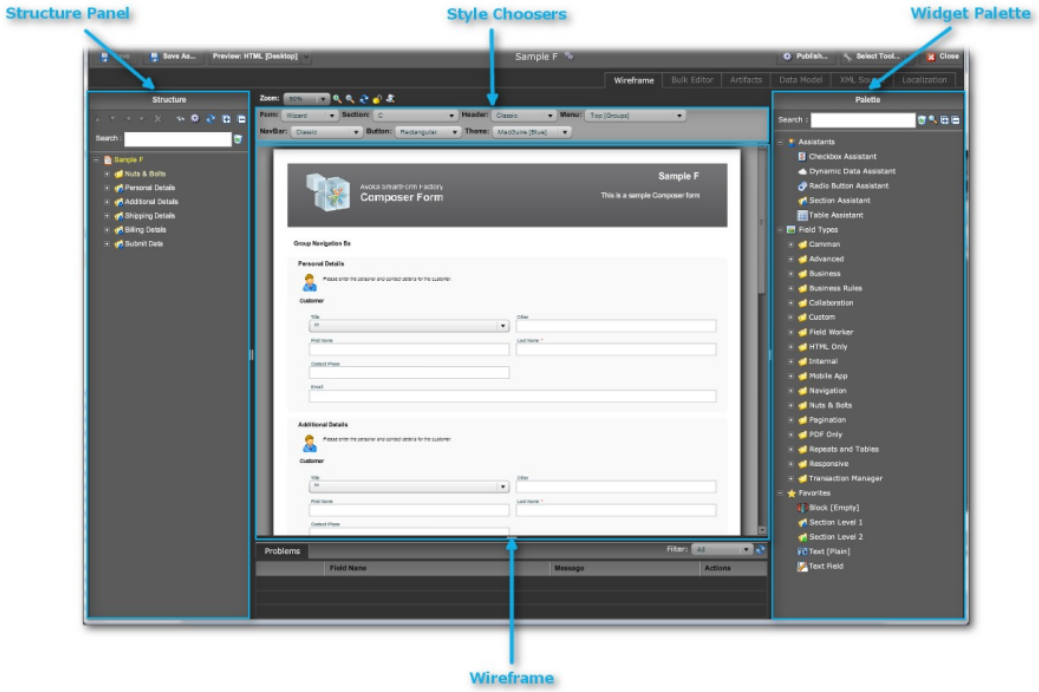


Screen 18 - Showing screen showing the create form version option selected. The user can modify the default text of "Published Revision".

Form Designer

Composer | Form Builder | v4.3 & Higher

The Wireframe View



The Wireframe tab of Form Designer is the heart of Composer. Not only are forms built from elements from the Widget Palette, but the whole logical structure and interactions of the form are built up in the designer. Composer functions as an agile and capable tool, where complex forms are built and deployed, on a range of devices, in a matter of days or weeks.

This user guide emphasizes the automated aspects of form construction in the early pages. Sections below the Advanced Topics section of the manual deal with more complex functionality. These come later in the guide to act as a reference and not to interrupt the narrative.

Structure and Preview

Forms are represented in 2 ways in the designer: the StructurePanel and the Wireframe (in the central panel of the designer). The Structure Panel, a collapsible tree structure, is the true representation of the form. The Wireframe is only a rough guide to how the form will look to the end user filling it in. (See Preview and Publishing on how to accurately view how the form looks.)

Dragging and Dropping on the wireframe.

The Wireframe is a representation of the visible elements. Because each element contains a structure, it is difficult to drop or move elements to the correct place on the element's structure tree. For this reason we recommend that **widgets are dropped ontothe Structure Panel**.

The other tabs ("Bulk Editor", etc.) are more technical, extremely powerful and not of immediate concern. See FormDesignerAdvancedFeatures.

Widget Palette

Overview of Widgets

The "widget" is the icon you drag from this palette onto the form's Structure Panel. (See below.)

The resultant object that appears on the form is called the "field". Other synonymous terms are: "node", "form element", "element" and even "widget". We may deplore the lack of common terminology, but in a technology that is rapidly evolving, regional and corporate difference in vocabulary are inevitable.

This has happened with browser and internet terminology in general: look at the various terms employed by different enterprises and teams for what should be standard topics like "query strings" (also incorrectly called "parameters"), "value pairs" (also called "tokens", "attribute-value pairs, "key-value pairs and so forth).

We will try to be consistent and use "widget" in the Palette and "field" on the form.

Organization of the Palette

The palette has 3 major classifications:

Assistants

Assists designers to build complex structures through wizard-style dialogs.

Field Types

collections of widgets organized into convenient groupings

Favorites

which you, the designer, can populate with your own choice of widgets

The Field Types contain a mixture of simple widgets — like Text Fields — and of compound widgets, made up of a set of widgets organized into some sort of functional structure.

The classification scheme used in Field Types is arbitrary and subject to change. Please spend a few minutes to look through the these types to get a bit of a feeling as to what they contain. The "Search" field at the top of the palette assists you to locate a widget without having to hunt and peck through the Field Types. Suggestions will appear in the palette as you type in your search item.

How to Use Widgets to Create Fields on Forms

You simply drag and drop a widget into the Structure Panel. You can also drop widgets into the Wireframe, but Avoka Technologies strongly recommends that you do not drop widgets onto the Wireframe. The reason for this is: the form's structure actually has many hidden fields and structural elements which can only be seen in the Structure Panel in its Advanced Mode.

Sometimes the new field will appear in the Wireframe; sometimes it completely disappears. And even when it does appear, all may not be well internally.

Widget Types

There are, broadly speaking, 4 kinds of widgets. This section of the manual will only touch on these briefly. All the Field Types in the Palette mix the 4 types freely, so there is little point in covering the widget types in much detail. There is some value in mentioning these. In practice, though, you are will not concern yourself as to the type of widget you are about to drag and drop onto the form.

Simple Widgets

These create simple fields like a single checkbox, text field or a date field.

Blocks

Block are containers of multiple fields. Blocks serve 2 main functions:

To group fields together visually in a colored panel, with a common label and so on To group fields together logically into functional units.

Containers can contain other containers in a hierarchical structure. There is no practical limit to the levels of nesting supported.

For your convenience, there are a number of PredefinedBlocks in the palette. They are invaluable for building complex functions on your forms with little more effort than drag and drop.

Sections

All forms have at least one Section (which is named "Instructions" by default). The content area of the Structure Palette is organized into Sections. Wizard forms display their sections as separate pages. Use the Section widgets to be able to create up to 3 levels of sections.

For More on Widgets

Please see:

FieldTypes

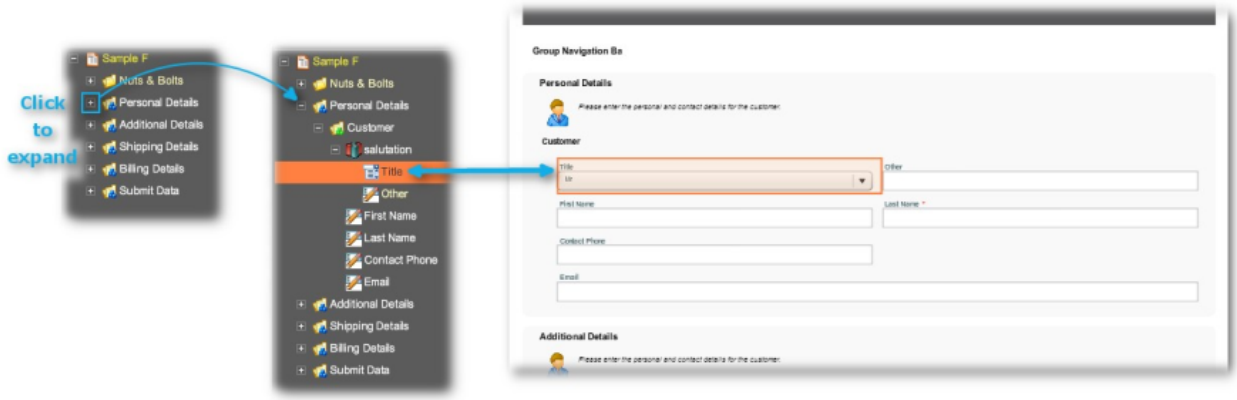
the technical reference to the many types of fields, Assistants

PredefinedBlocks

Structure Panel

Standard Mode

The default mode of the Structure Panel is called the "Standard Mode". This does not expose the full structure of the form, but does expose the visible elements and those invisible elements most relevant for day-to-day work.



Showing the relationship between the Structure Panel and the Wireframe

The Standard Mode is particularly recommended for when you place new widgets onto the structure. This is because, in the Standard Mode, you can see enough of the underlying structure of the form to place the widget correctly — and not into some invisible and inaccessible part of the underlying structure — so that the new resulting field on the form works more or less correctly.

Advanced Mode



Standard Mode vs Advanced

The Advanced Structure exposes all the underlying structure of the form. (You can see this by comparison with the Advanced Mode panel with the XML code in the XMLSourceTab.) Such control is obviously useful to power users of Composer, but this level of control is beyond the scope of this section of the guide.

Nuts & Bolts vs Content

The "Nuts & Bolts" part of the Structure has both read-only and editable items. Read-only items show the settings for the form made in Transaction Manager; in their EditProperties dialog, the Properties tab is blank. Usually the first level items of Nuts & Bolts are read only and their children are editable.
You alter the settings of the editable items through the EditProperties dialog. More Nuts & Bolts items are exposed in the AdvancedMode.

Composer Framework (Advanced Mode Only)

This is a new section of the Composer 4 Structure Panel. This framework exposes the deepest workings of the form, such as the connection with the TransactField App or the HTML generation engine. You should leave this alone except in the rarest or rare circumstances

How to Place a New Field into the Structure

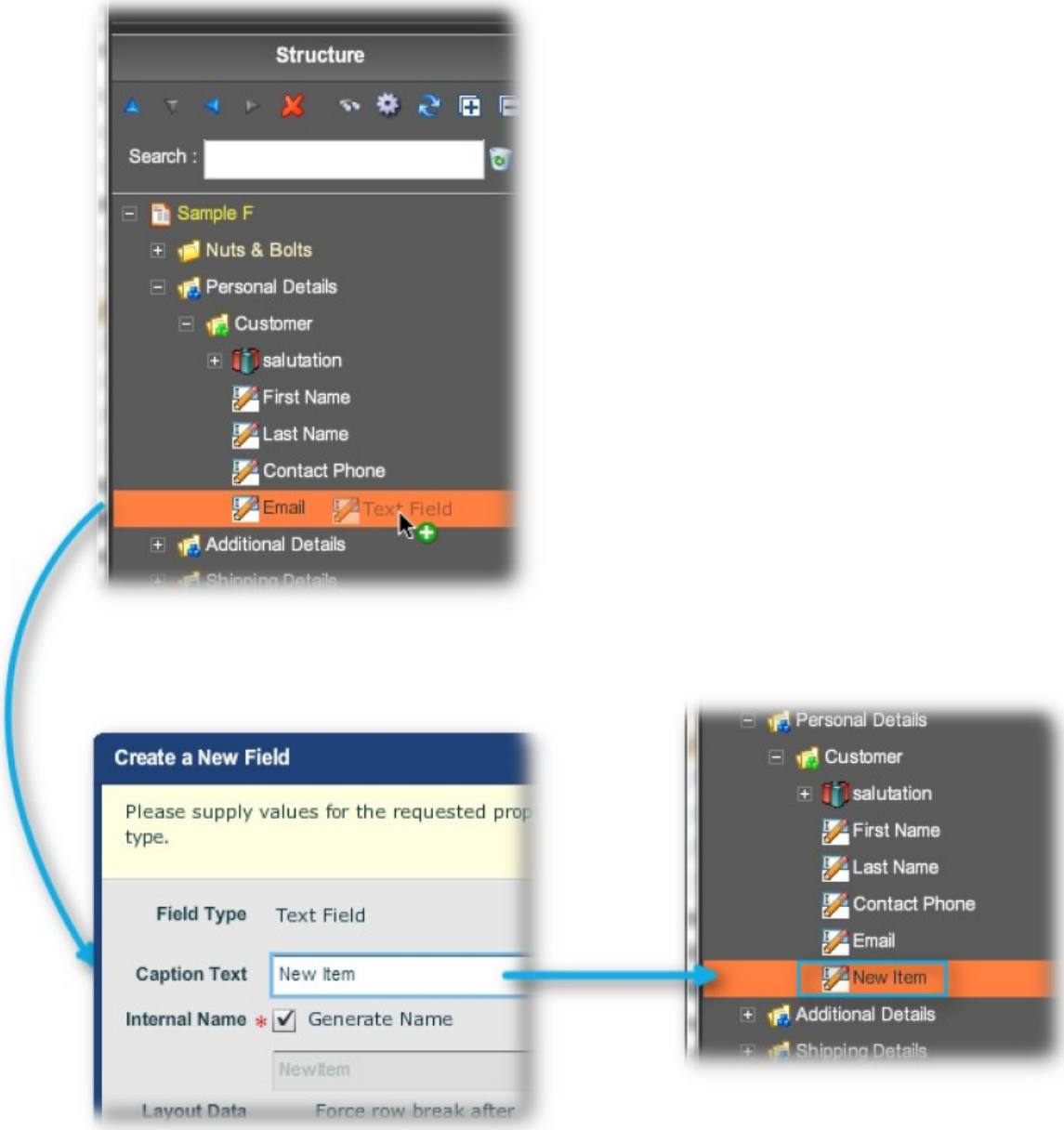
Drag and Drop

The method is to drag a widget over from the Palette (on the right) and drop it onto an object on the Panel (on the left). Either the widget will

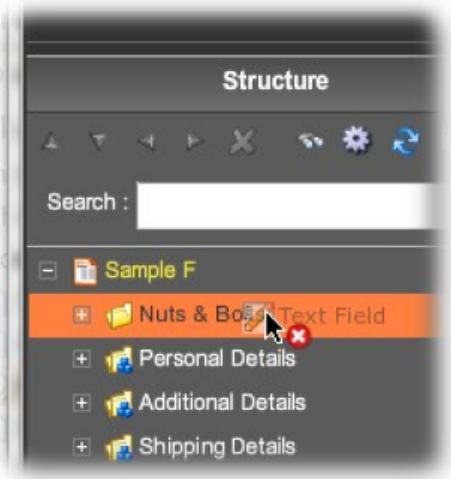
1. drop into the object onto a container like a block or section and it takes up its position as the last item in the container



1. drop below the object onto a simple field (one that does not expand) and it goes below the field at the same level in the structure



1. or refuse to go into the structure when dropped onto an illegal position in the structure.

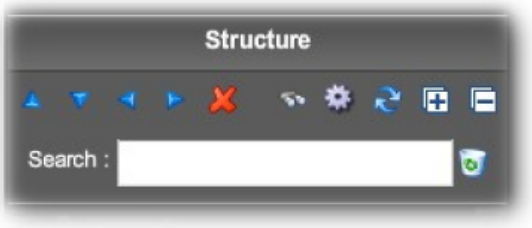


Note: you can force the widget to behave as item 2, by holding down the <ctrl> key as you move the widget into the structure.

Shortcut for Power Users

Select a position in the Structure Panel and, while the position is still highlighted, double click on a widget in the Palette. The new item will go after the still-highlighted field in the structure.

Menu Bar



The menu sits at the top of the Structure Panel. Its functions are (from left to right):
Blue arrows to move a highlighted item in the structure up or down or promote (left) or demote (right) Delete the item
Search for the item in the Search field below (the recycle bin clears the search field) Standard vs Advanced Mode
Refresh the panel Expand all Collapse all

Layout

So far, we have shown how to add fields to the structure, but said nothing on layout: where fields will sit in relation to each other, either vertically or horizontally, what will be the distance between fields and how they position on the form.

This omission is deliberate. Layout is dealt with hereinthis guide and layout is automated in Composer as much as possible.

Why? Wouldn't forms look better if the designer sweats over positioning each element and nudging fields on the web page to get its position perfect? Well, the answer is that forms in a web-browser cannot have all their elements as tightly controlled as with traditional hard-copy forms, as the web page runs the gamut of the ranks of the many web browsers now in common use. Also, these forms will be viewed on different devices: on Android phones and tablets; on iPhones, iPads, iPod Touches, on Windows mobile devices and desktops, on desktop monitors of a range of sizes, via mobile apps, via HTML and PDF.

In such a babble of viewing environments, what was painfully adjusted for one circumstance will look lousy in another. And you cannot dictate which modern browsers or devices you would prefer your customers not to use. Gone are the days of wagging the stern finger and stating "Best viewed in such and such an ancient browser and not the modern one you are using." If users have migrated over to modern browsers so as to be able to engage in social media, do not try to stem the tide with your web forms.

So, these considerations dictate that layout is held in reserve for later discussion. For the present, we encourage you to experiment with widgets in the Palette, and to see what happens when you drop these onto a form.

Assistants

Composer | Form Builder | v4.3 & Higher

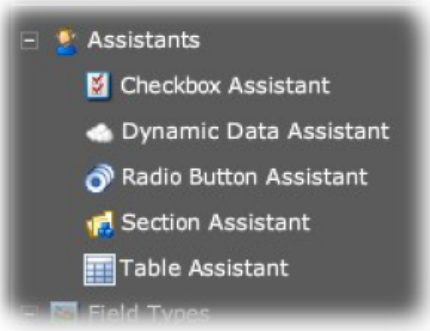
Assistants are located at the top of the Widget Palette and this is the reason they are discussed first. The Field Types in the palette are a mixture of many different types, see Field Types, placing any of them on the form is the same process already covered above. This is also true of Assistants, and they make a good starting point as the resultant fields on the form are so useful, and even rewarding, for the beginner

Assistants are widgets that facilitate the creation of complex structures. Another important set of widgets, that also simplifies complex form structures, is PredefinedBlocks.

We will discuss individual widgets inmoredetaillaterinthis manual. But because Assistants and Predefined Blocks are so powerful, we will now be building examples using these two types. Designing complex forms using individual widgets is more the domain of Composer's power users.

Adding Assistants to a Form

When assistants are first droppedintotheformstructure, they pop up a wizard to add multiple widgets to the form (radio buttons for example). Each of the Assistants features a table to add a component widget to the group, and Composer will both create the group and maintain the overall structure of the group.



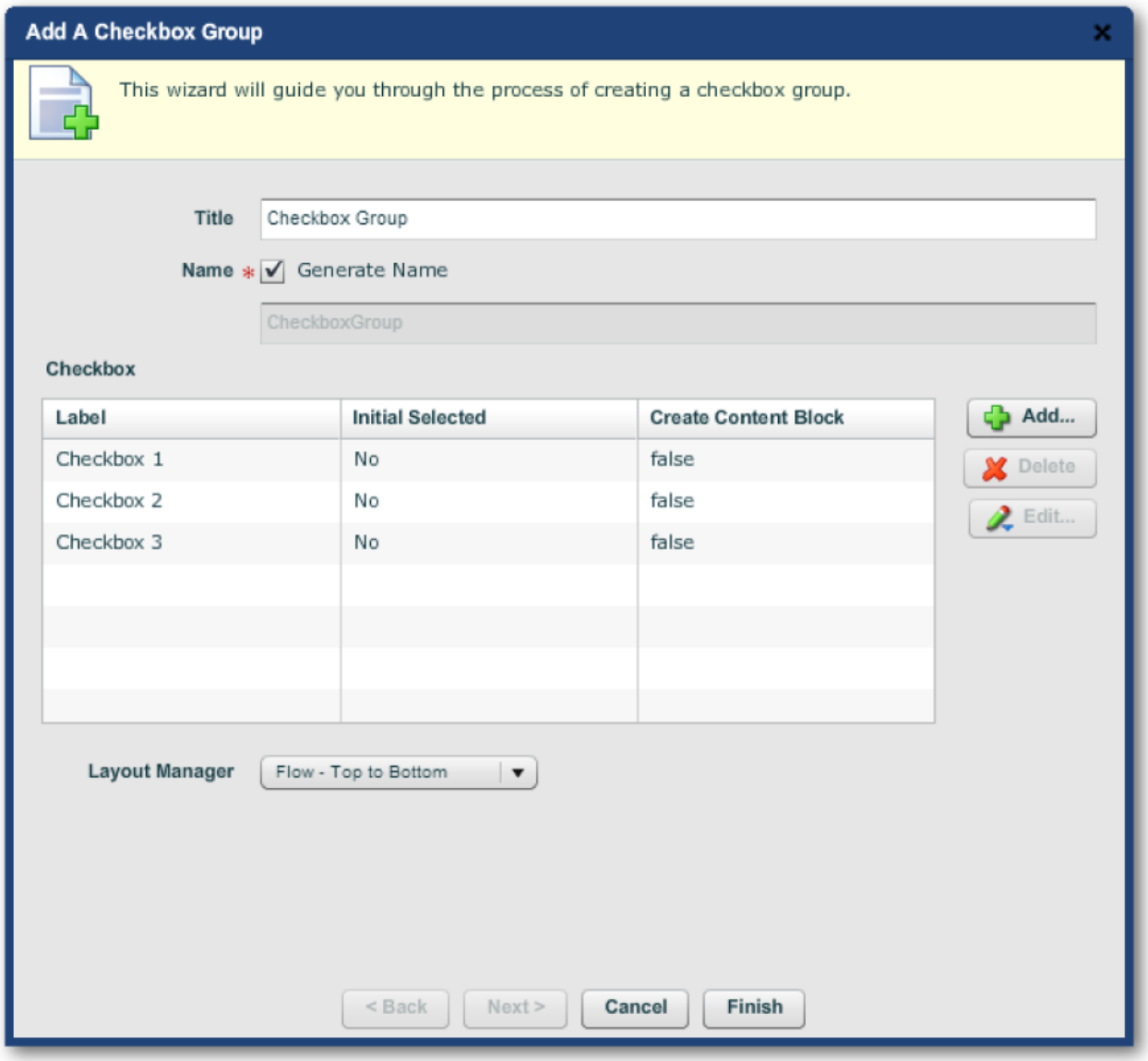
Assistants

Checkboxes

A checkbox is the simplest selector: it can be either checked or unchecked. In a group of checkboxes, more than one can be selected.

Using the Checkbox Assistant

The Checkbox assistant will create, in a single operation, a set of checkboxes and their title. If you're creating multiple checkboxes, it's generally a much quicker and easier way to create them than creating individual checkboxes.





The resultant struture

See below for more on the Content Block.
In Composer, the underlying raw value of the checkbox can be set to any string values (see EditProperties).
By default, the values are checked = "true", and unchecked = "false". However, you can set the checked and unchecked values to whatever you want, such as checked = "Y" and unchecked = "N". This can be useful when checkbox values get exported to legacy systems that do not suppport Boolean values ("true" or "false").
Layout Managers get their own topic below.
See belowforanexplanationoftheContentBlock.

Radio Buttons

Radio buttons are similar to checkboxes, except that only one of them can be selected from a group.
For example, if we add a Radio Button Assistant to the form, the resulting popup wizard dialog looks as follows:



The "Add a Radio Button Group" wizard dialog

The "Bound Value" for a radio button is the value returned by the form when that button is selected. In the example above, the "Bound Value" is the same as the "Label"; you can change this to return a more convenient value, "1" instead of "Button 1" for example.
If you want to reference the value of the button group (i.e. which button was selected), point to the Button Group widget.
If you check "Make Button Group Mandatory", users must click on one of the radio buttons. Otherwise they will be prompted to do so. See Validation.
The "Initially Selected Radio Button" dropdown is for specifying which of the radio buttons is selected by default when the form is first loaded. Buttons need not be selected by default (though many useability experts advise against this). But once any of the buttons has been clicked on, the group cannot be returned to the unselected state. This is a property of HTML.
Layout Managers have their own topic here.
Filling in the wizard creates the following structure in the form:



The structure the "Radio Button Assistant" adds to the form

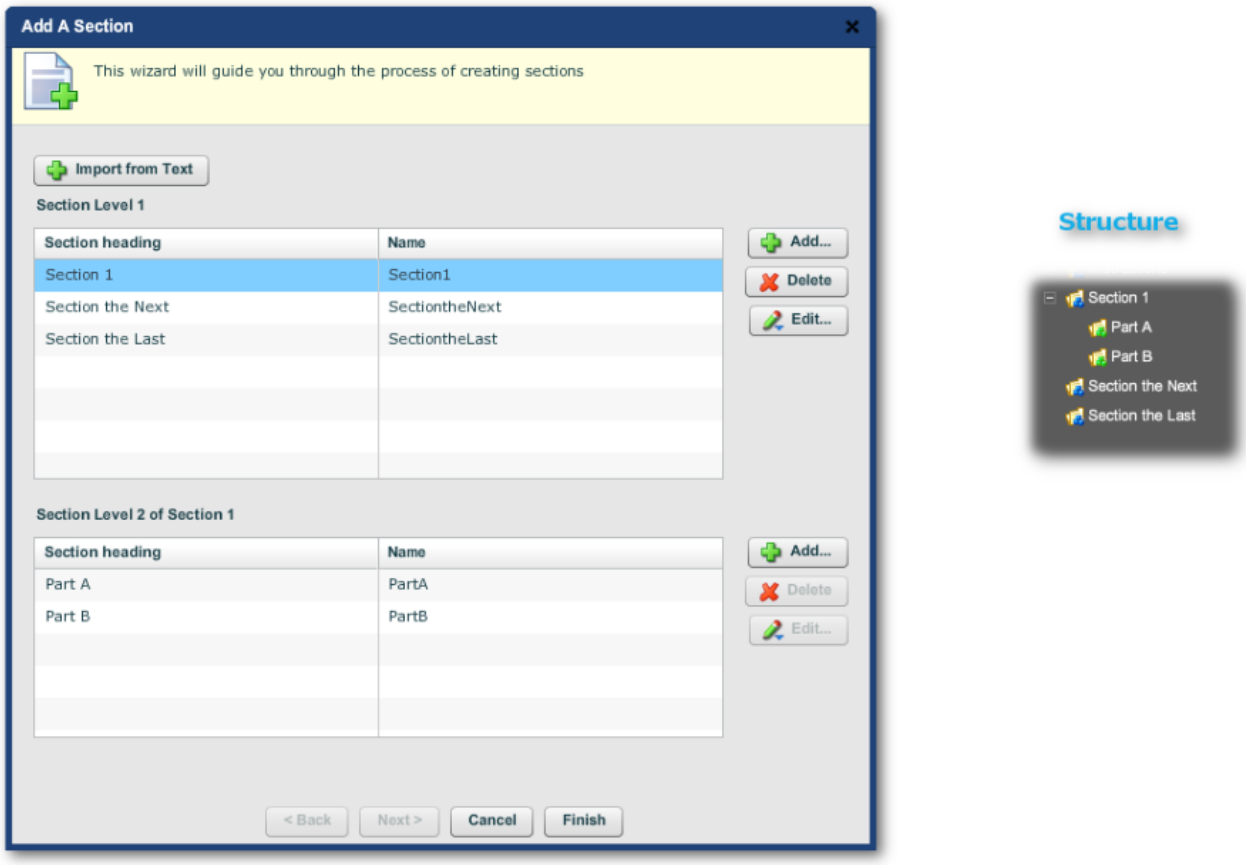
The buttons now render on the form (as seen in Preview) as graphics with a more contemporary look than the standard desktop browser widgets.
Note: the button group now appears in the structure below the buttons. This done to allow the inline validation message to appear correctly on the form.

Dynamic Data

Dynamic Data means that the form's display can change before submission, without the form having to be reloaded. A common use case for dynamic data is to have suggestions appear to assist the user to supply data, an address for example.
Using the assistant is more or less a matter of filling out the wizard, but such a sophisticated concept is best deal with under its owntopic.

Sections

Manually dropping the three levels of Section widgets onto a form is not difficult, but you should avail yourself of the convenience of panning all the sections of a form in the one wizard.

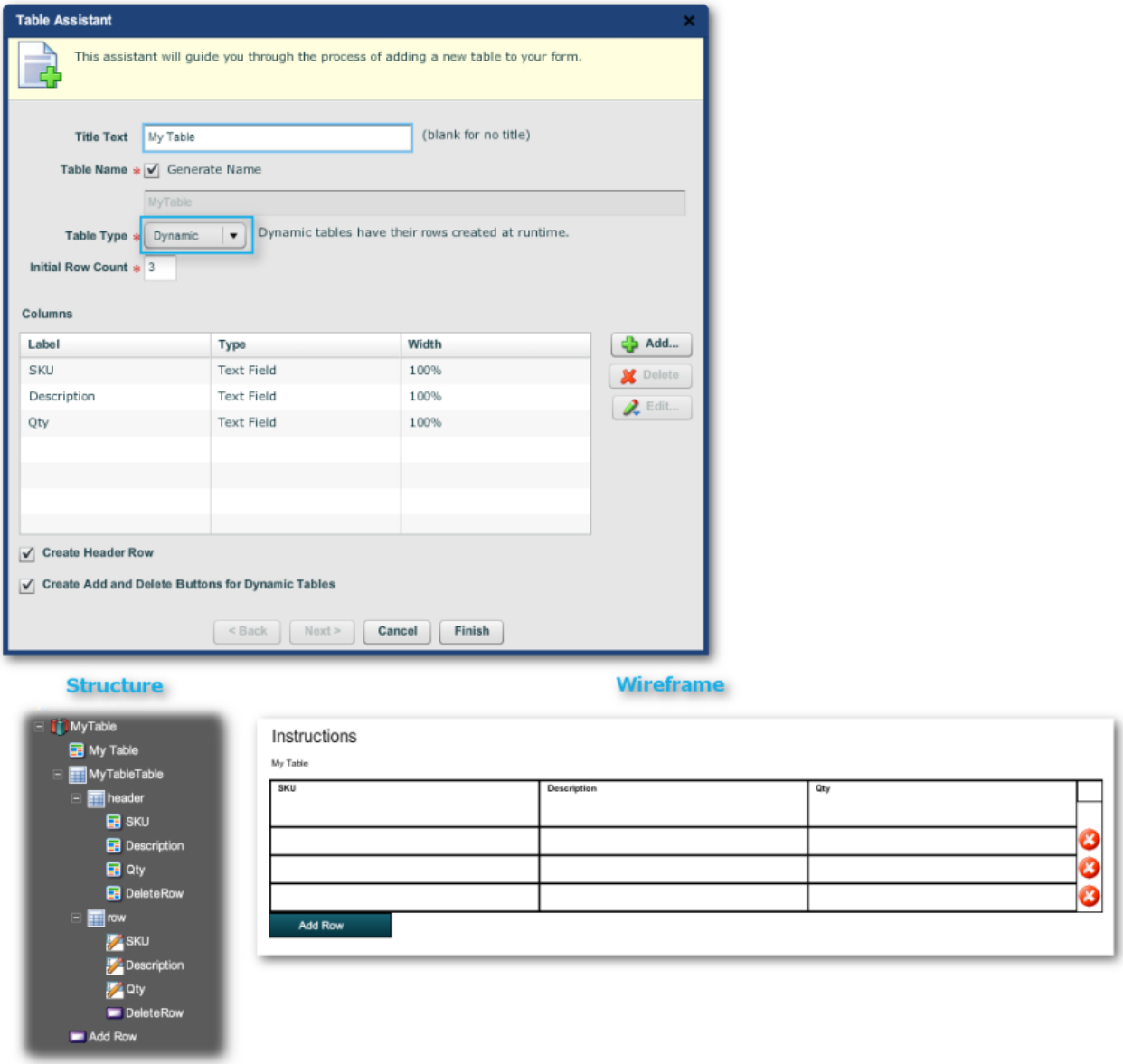


The Section Assistant wizard

The wizard is easy enough to use, but Sections now have greater significance than just bold headings on the form: they now control the user's very navigation through the form, and the mobile slider menu in responsive layout.

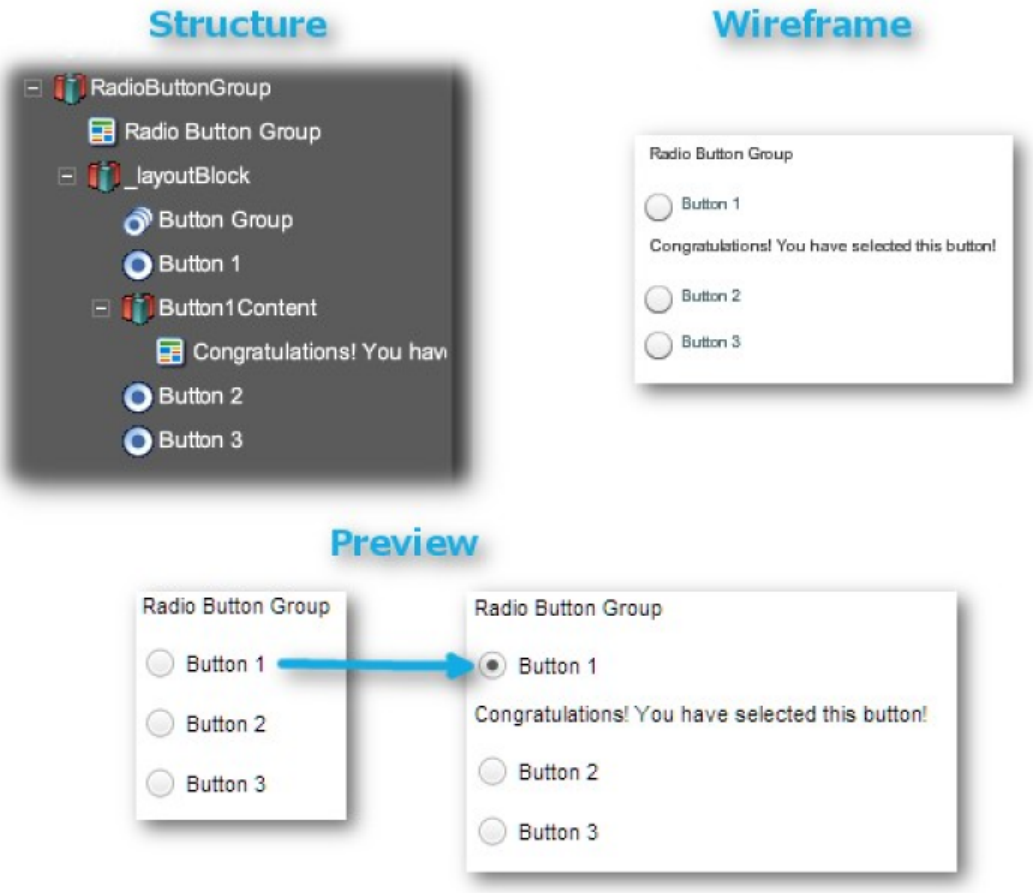
Tables

In forms, tables are often dynamic structures where rows can be added or deleted by the end user; they are not, therefore, simple static HTML tables. In Composer 4, you can use the assistant to create either static or dynamic tables.



Content Blocks

The following applies to the older style of forms.
As you add each element to the group through the "Add..." button in the wizard, the resulting dialog gives you the option of creating a content block for the element. You can drop widgets into this block, the Plain Text object for example. The block is normally invisible, unless that group element is activated, such as on the selection of the checkbox.



You can reverse this behavior by going to the Edit Properties dialog, "Rules -> Visibility Rule -> Edit" and changing the script from
return {Button1};
to
return !{Button1};

You can also manipulate other behaviors by editing widgets in the group, but take care: you do not want to break the structure.

Note: in Maguire template forms, content blocks now appear at the bottom of button groups, **not between buttons**.

Predefined Blocks

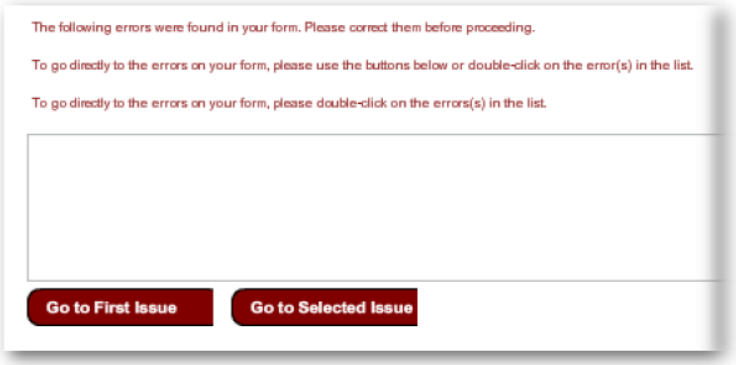
Composer | Form Builder | v4.3 & Higher

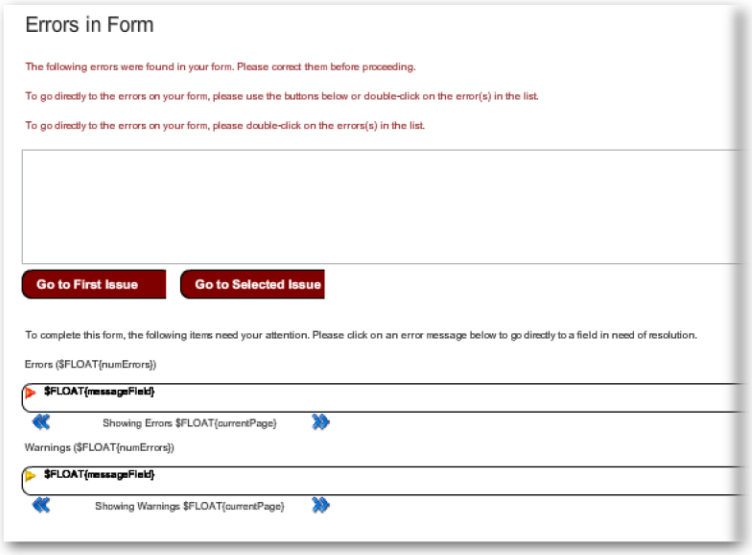
Many of the standard Composer widgets are implemented as prefabricated blocks. You utilize them as you would assistants: drop one into the form's structure and take advantage of the resulting complex structure that adds functionality to your form.

These include various Address blocks, a block to display errors, the standard form header, Sections, the Signature block, and various blocks for use in Transaction Manager.

You cannot modify the predefined blocks in the pallet, only the predefined blocks that you yourself created in the Pallet. But once a predifined block has been added to the form, you can modify that instance and that instance alone.

Here is the current range of predefined blocks generally made available for most organizations in the pallet. You

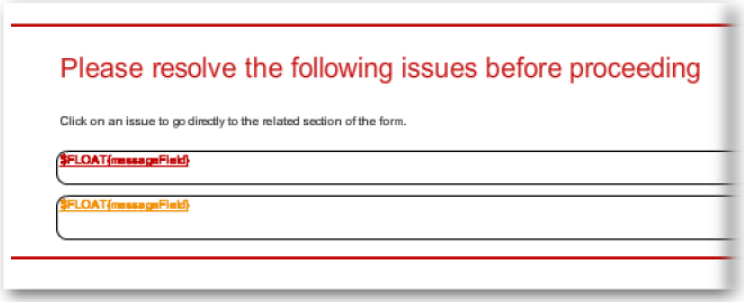




may not see all of these, due to the Composer configuration of your Organization.

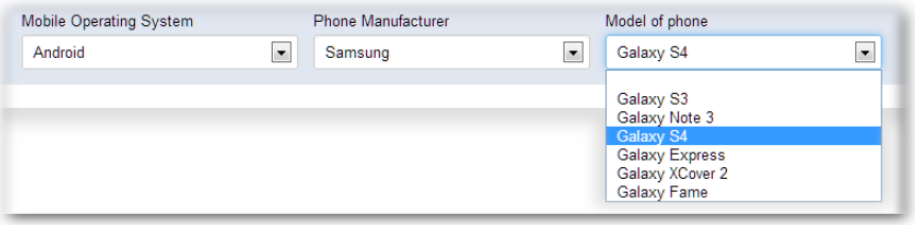
Predefined Block	Description and Wireframe

Common field type	
Fieldset	Used to improve accessibility for forms that have a number of scattered radio buttons, where these do not fall into one convenient block nor have the one controller. Poorly structured collections of radio buttons can confuse screen reading software, making these forms unuseable for end users reliant on readers. See Useability and Accessability
Error List Block	In the event of an error, this block presents a multi-line message to end users (explaining what action they should be taking), a display area to list the errors and two action buttons.
Error Selection Block	Makes a complicated structure, with an Error List Block above an Error and Warning Block.

Advanced field type	
	
Predefined Block	Description and Wireframe

This block does not place visible objects on the form. You must put Cascading Dropdown List widgets on the form which point to the manager.
See Cascading Dropdowns.

Cascading Dropdown Manager



Error and Warning Block

The user can scroll, via left and right navigation buttons, through the warnings and errors.

Form Header

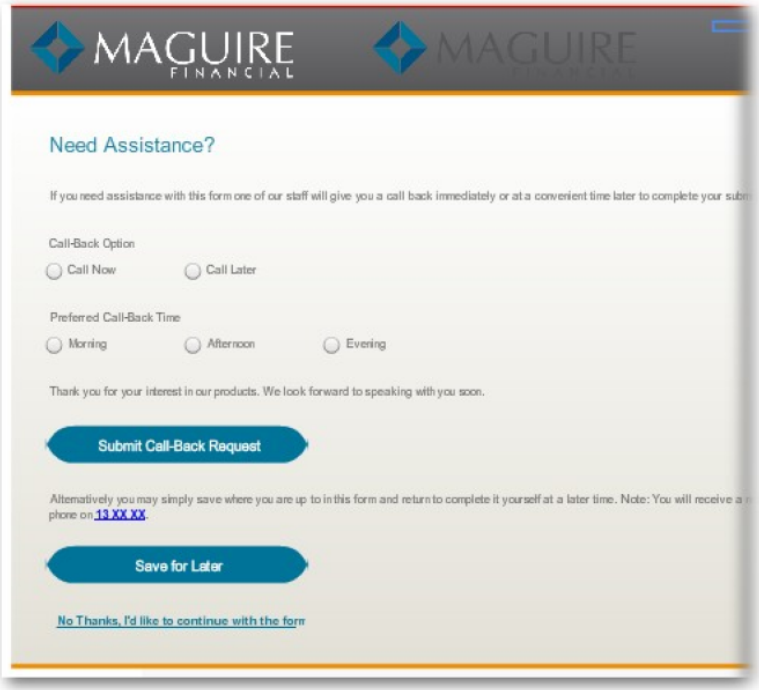
Block containing the elements that make up the header of the form's template: the logo and the text of three levels of headings. (The visibility of of these headings and other elements depends on the template.)The block can be dropped into the structure. Normally there is no need to add an extra heading block to the structure, so this block is for specialized customization only.

In the Wizard, set the "Region" to "North" and the header will appear at the top of all navigation pages.

Predefined Block

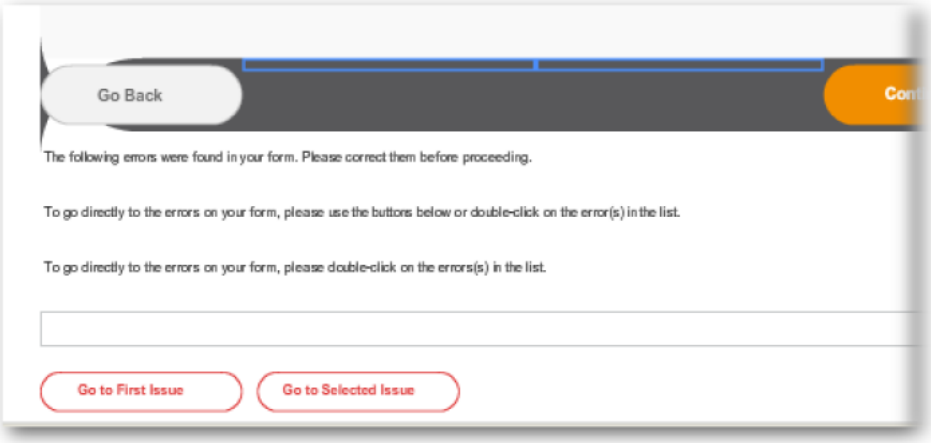
Description and Wireframe





Block with the footer elements of the form, in this case, an Error List Block.

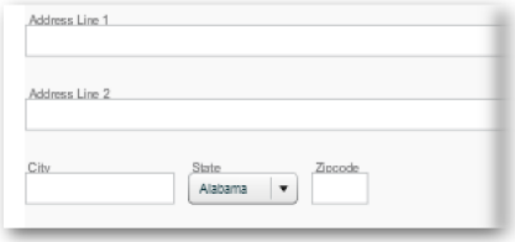
Form Footer



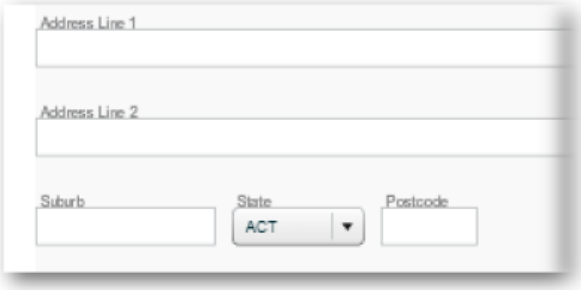
Business field type

Address [Australia]	Produces a block with plain text fields "Address Line1", "Addresses Line 2", and an inline of "Suburb", a dropdown for"State" and 4-digit numeric for "Postcode".

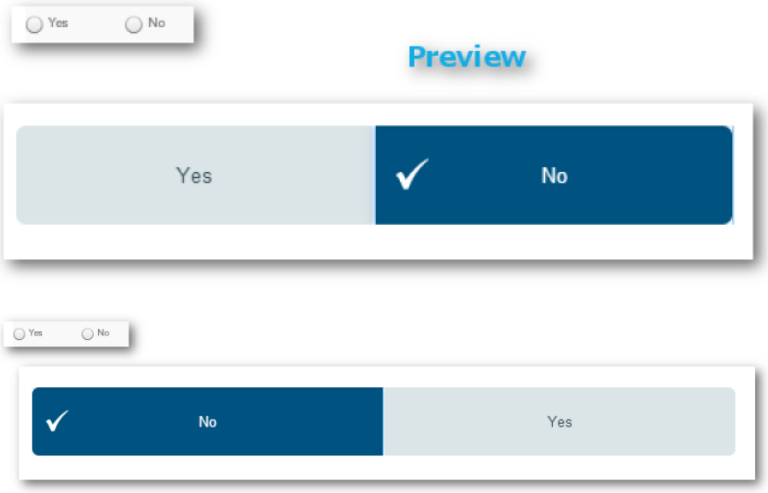




Predefined Block	Description and Wireframe



Address [Billing & Mailing]	Creates an Address [Australia] block and a second block of identical format which becomes writable if the linking checkbox is made inactive.
Address [United States]	Same as the Address block above, except that the "State" dropdown is populated accordingly and the "ZipCode" numeric field accomodates more digits.
Radio Button Assistant Block	This structure is created by dropping this assistant block onto the form. Takes care of the manual configuration tasks of the whole structure, as explained here.



Predefined Block	Description and Wireframe

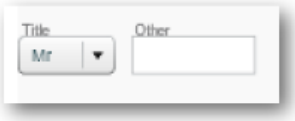


Radio Button List [Yes/No]	A block of two radio buttons, a "Yes" and a "No".
Radio Button [No/Yes]	Button order reversed from the above.

	Invokes a Transaction Manager service to provide the DocuSign signing workflow before delivering the submission to another TM service.					
Signature [DocuSign]						
Title Block	Produces an in-line dropdown ("Mr"	"Mrs"	"Miss"	"Ms"	"Other") and a text field to specify "Other".	



Predefined Block	Description and Wireframe



Business Rulesfield type

Business Rule - Error	These are invisible on the form, but control the behaviour for the form. See the Business Rules section of this manual. These field types cover several widgets or widget groups, or even the whole form in some behaviours. Some aspects of these resemble the "Rules" tab of the "Edit Properties" dialog of individual widgets, but the rules tab applies only to the one widget.
Business Rule - General Purpose	
Business Rule - Warning	
Mandatory Block	
Mandatory Checkbox Block	

Collaboration

Please see Collaboration.

Mobile App (Field Worker) fieldtype

Geolocation Block	Also works on the desktop. When the button is tapped or clicked, it loads two floating fields for latitude and longitude.
Barcode Block	Take a picture of a barcode with a mobile device's internal camera using the "Scan" button on the form. The barcode is decoded, and its value populates the field below the camerabutton.

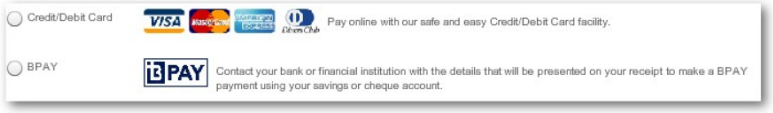
PDF field type

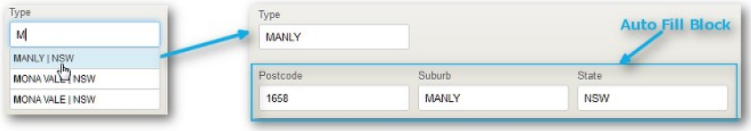
Process Management Field Block	See the section on Field Types.
Smart Attachment Field Block	Adds a table to a PDF form for the user to upload an attachment. HTML forms handle attachments differently, see Submissions and Attachments.

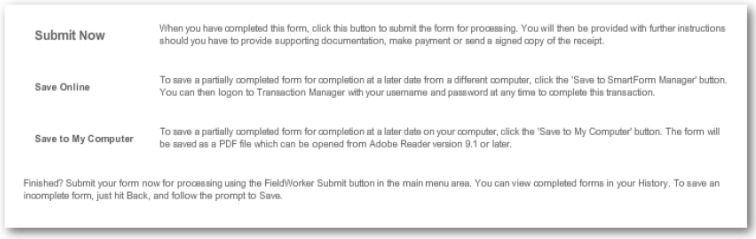
Transaction Manager fieldtype

TM Payment Table	Creates a display table of payments, including GST (Australia's Goods & Services Tax) and Price.





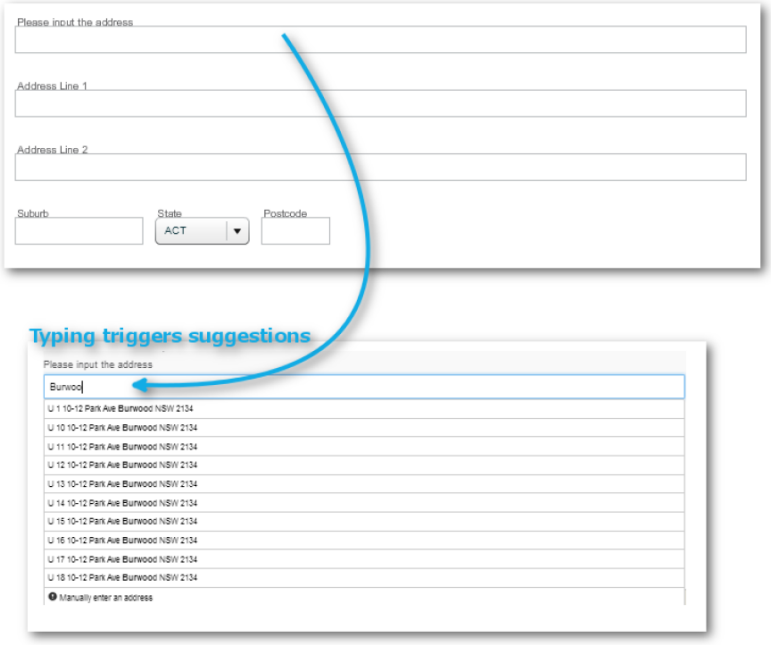




Predefined Block	Description and Wireframe
TM Payment Type Selector	Creates two radio buttons and adds content and logos. (The choices are for Australian conditions,)
TM Reference Data Block	Creates an Auto-suggest field for some predefined list.

TM Submission Block	Creates three buttons and associated content for Submitting, saving to TM natively, or saving in PDF locally. Creates a block of two text displays for the receipt and a receipt bar-code graphic.
TM Submission Receipt	
TM Veda Address Lookup Block [Australia]	Creates a predefined dynamic data block made up of a trigger field and an address block. See the VedaAutoSuggestpredefinedwidgetandservices section.

Predefined Block Description and Wireframe



Tables & Repeats

Composer | Form Builder | v4.3 & Higher

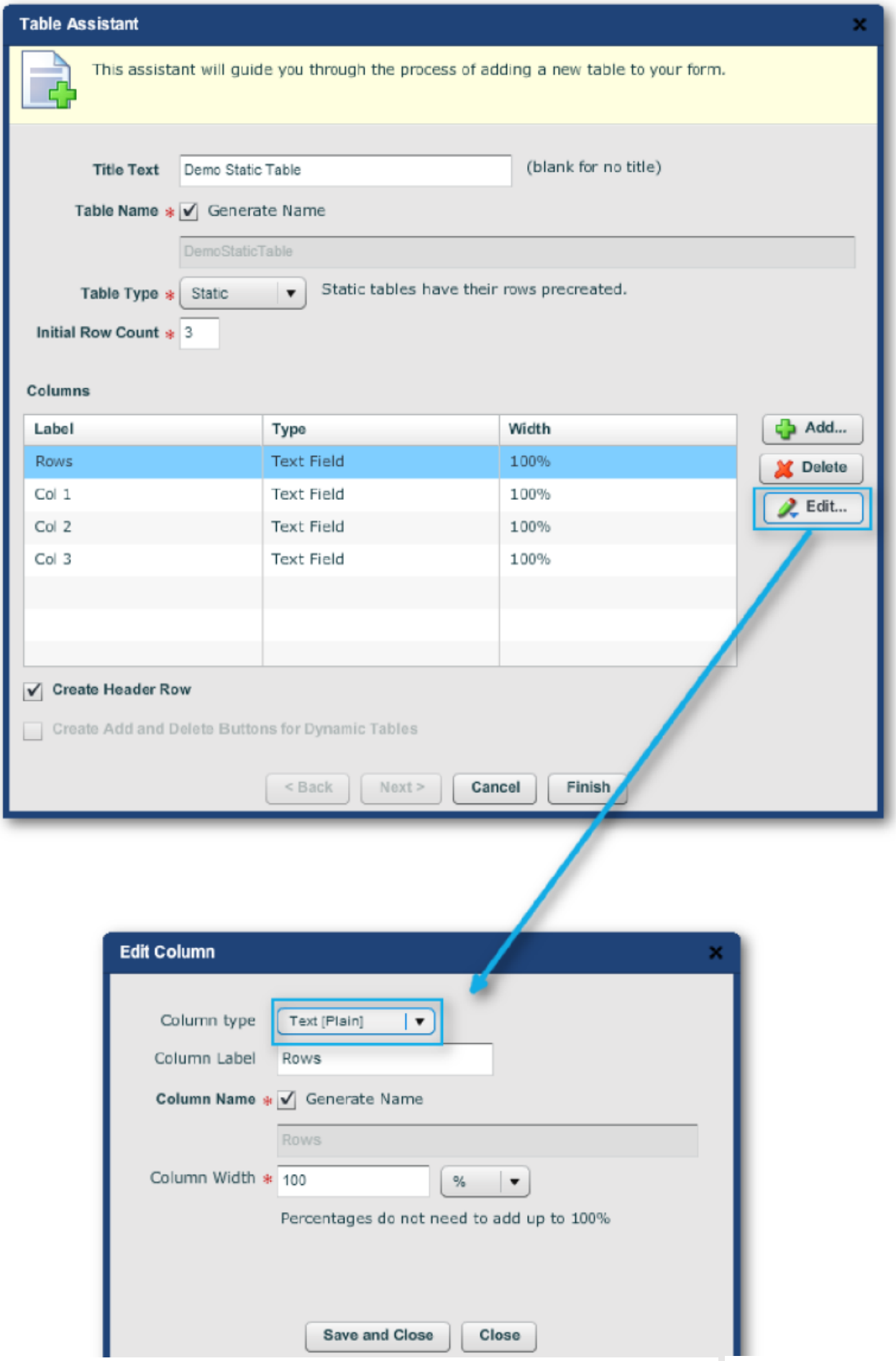
Static vs Dynamic

When you include a table or repeating block into your form, you only design one row of structure, and specify as many rows as you want to display initially.

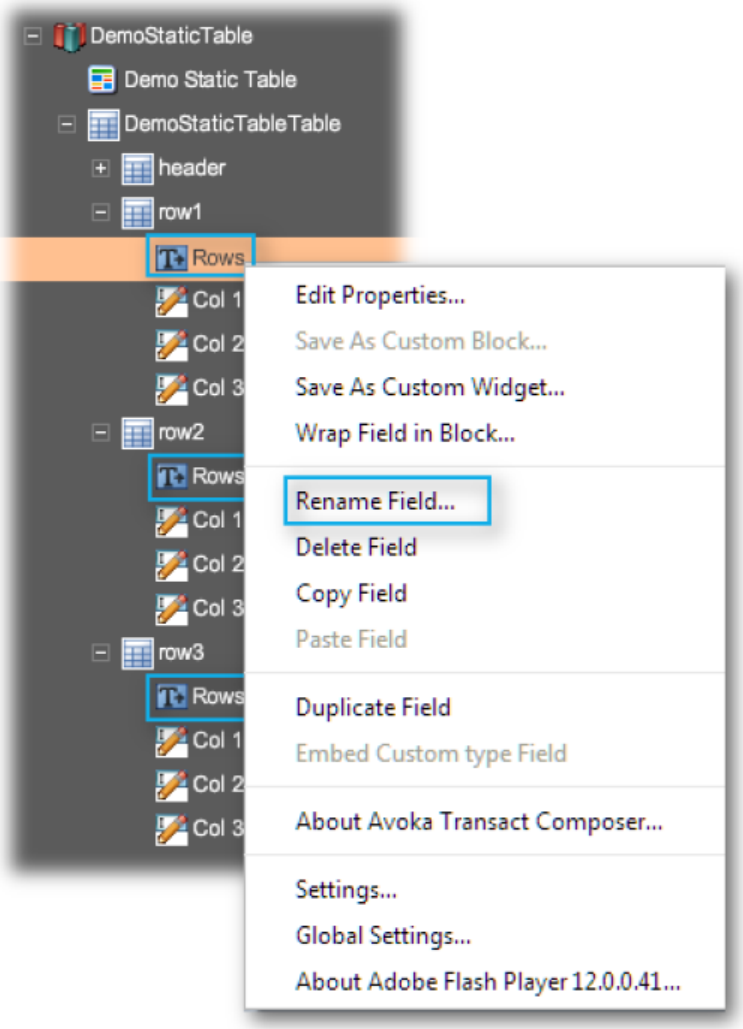
Tables usually have a heading row (or rows). Tables also can be static (i.e. a fixed array of fields for users to fill in) or dynamic, where users can add or delete rows as they go.

A repeat is similar to a table, but instead of being limited to the height of a row, you can make the repeating unit as large as you want, with no restrictions on how the fields lay out — unlike the single line limitation of a table.

Use the Table Assistant to create the table. Creating a static table with headings in each row on the LH side involves a few more steps. Use the Tables Assistant Wizard as follows:


Table Assistant
This assistant will guide you through the process of adding a new table to your form.
Title Text
Demo Static Table
(blank for no title)
Table Name
Generate Name
DemoStaticTable
Table Type
Static
Static tables have their rows precreated.
Initial Row Count
3
Columns
Label
Type
Width
Rows
Text Field
100%
Col 1
Text Field
100%
Col 2
Text Field
100%
Col 3
Text Field
100%
Add...
Delete
Edit...
Create Header Row
Create Add and Delete Buttons for Dynamic Tables
< Back
Next >
Cancel
Finish
Edit Column
Column type
Text [Plain]
Column Label
Rows
Column Name
Generate Name
Rows
Column Width
100
%
Percentages do not need to add up to 100%
Save and Close
Close

The Table Assistant produces this form structure, although the LH captions for each row have to be altered — by right-clicking on each of the Plain Text fields in each of the static rows and selecting "Rename Field...".



After you have used the Table Assistant ti create a dynamic table, you cannot convert the table into a static one.
Note: Tables implicitly require a lot of screen space. In the case of the small screens of mobile devices, you take advantage of Composer's ResponsiveLayout feature which is on by default and requires no further work on your part.

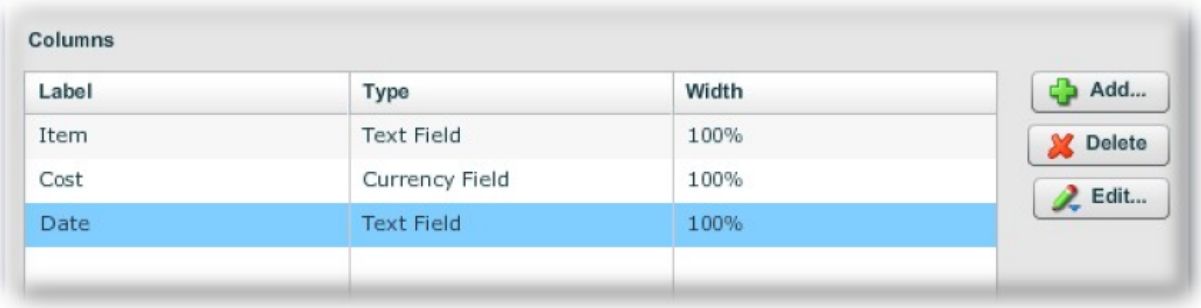
Editing Table Structure

Adding and removing columns from a table requires making several manual changes to the Structure Panel.
You should alter the table element in the structure. This is the container that holds the Header and Row blocks (i.e. "DemoStaticTableTable" in the example immediately above). Double-click on the element and in Edit Properties, go to "Properties -> Layout -> Layout Manager" and alter the "Column Count" or "Initial Row Count" to your requirements.
The Structure Panel makes it easy to change the elements in a row. For example, in the above table created using the Table Assistant, all the row elements are Plain Text fields. Say we want to have the second column to be a currency field and the third into a date field. We just delete these elements from the Row element and drop a Currency Widget and a Date Widget into that Row icon in Structure Panel, name them accordingly (though they need not have the same names as the former elements) and move them into the correct positions

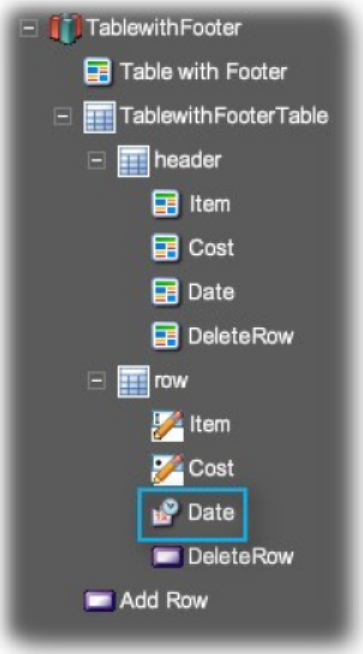
in the structure using the blue arrows at the top of the panel. The result looks as follows:
This flexibility extends across the entire table structure, though be careful to preserve the structural integrity of the table.

Adding a Footer

You add footers to tables through the Structure Panel.
We can use the Table Assistant to create a 3-column table with Plain Text Column, a Currency column and a third column, which we will re-cofigure to hold a Date field.

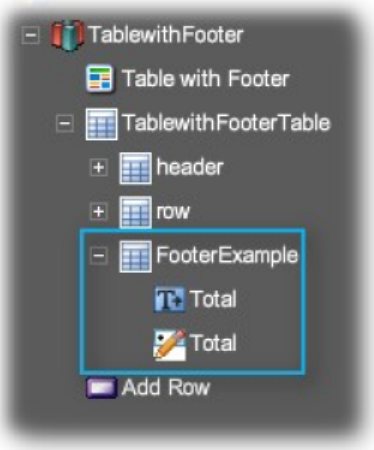


We then manipulate the Structure Panel to delete the text field (by right-clicking on it in the Structure) and drop in a Date widget into the Row container into the correct position (using the <ctrl> key or by using the blue up/ downbuttons).

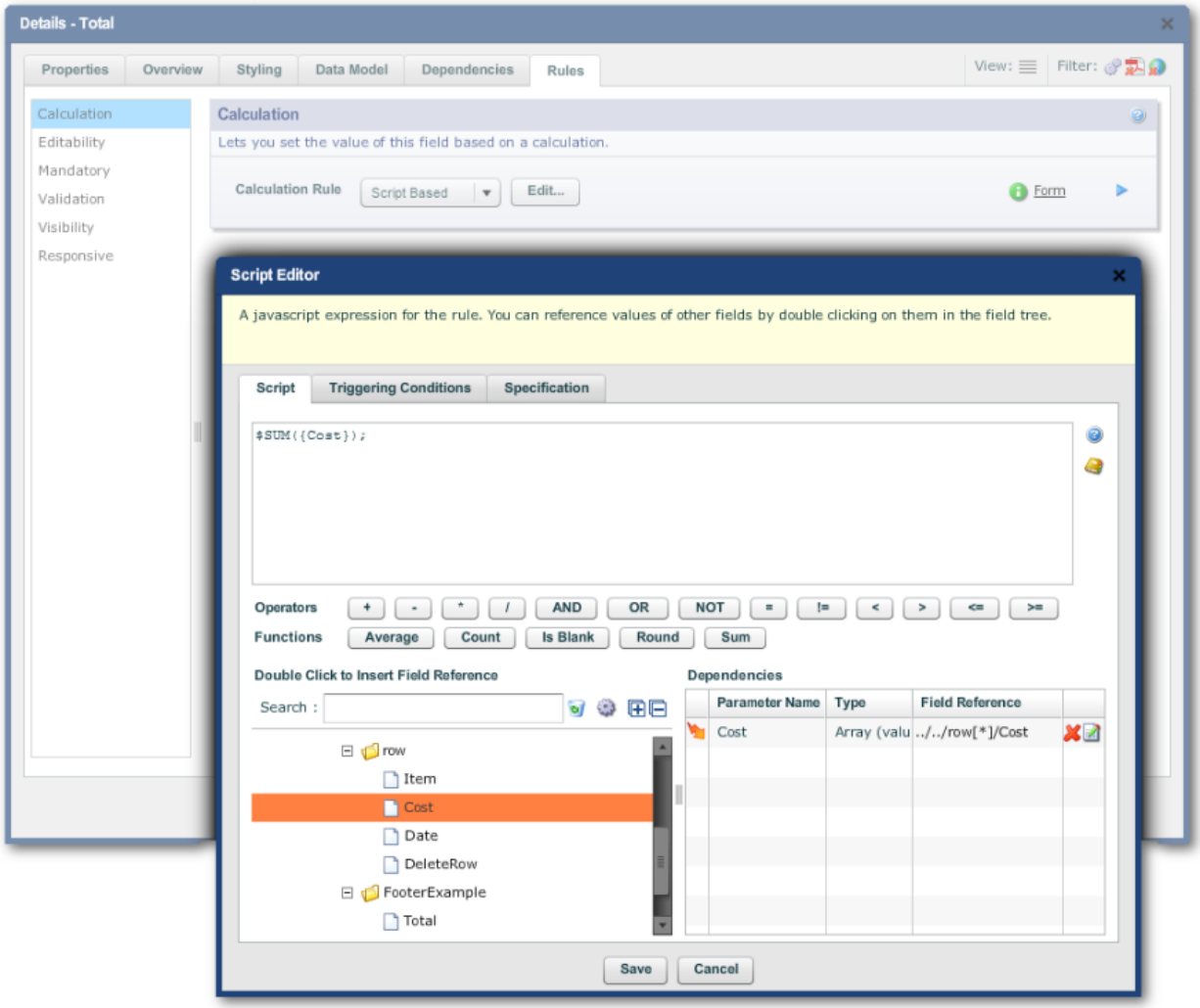


Drop a "Table Footer" widget from the palette onto the table field in the Structure Panel and add appropriate elements from the palette. Here we add only two elements: a Plain text and a Currency. Because the table has four columns per row, we adjust the column widths of these two for the sake of neatness: use "Edit Properties -> Properties -> Layout -> Layout Constraints -> Column Span". The "Second Column Total" has a column span of "1" and the "Total", "3".

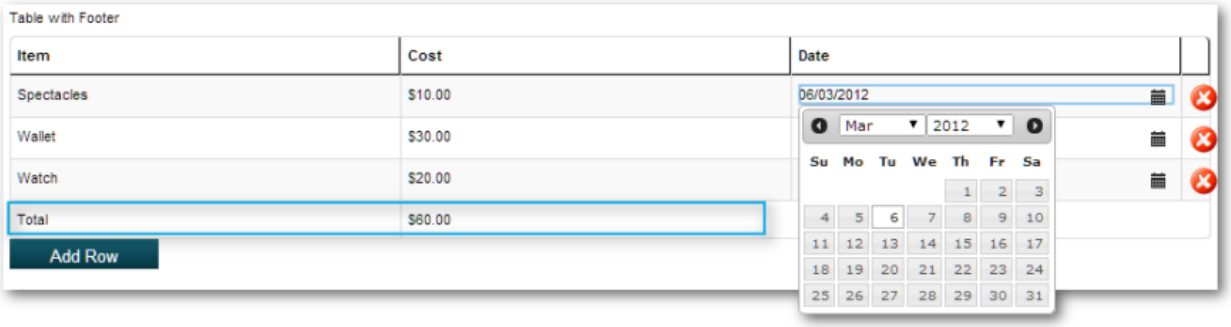
The new structure looks like this:



All that needs doing now, is getting "Second Column Total" to sum the second column. See Scripting and Dependencies on how to do this; it is done simply enough by the Edit Properties dialog for the "Total" currency field. In "Edit Properties -> Rules -> Calculation -> Calculation Rule -> Script Based -> Edit" and fill in the following:



The resulting table as viewed in "Preview HTML", looks like this (with some test values added):



Hint: you should make the table cell total read-only ("Edit Properties -> Rules -> Editability -> Editability Rules -> Never Editiable").

Repeats

There is no Assistant for repeats. This means that the you have to build up the structure in the Structure Panel yourself by dropping in the appropriate widgets and blocks and using the Edit Properties of the fields to script them. See Repeats in AdvancedTopics.

Hint: a good way to start building repeats is to mimic the structure of tables, with blocks for headers, for rows and footers.

Radio Buttons

Composer supports copying and pasting groups of radio buttons from one part of a form into another without the need to alter the scripting of the groups or their component fields..

Templates

Composer | Form Builder | v4.3 & Higher

Overview

We are all now familiar with the concept of templates, be it in word processing, corporate documents, branding or web blogging. In Transact Composer, the template concept goes much further, embodying the look and layout of the form.

- color schemes
- headers
- graphics
- typography and layout managers
- menus and form navigation
- The widgets are made available to form designers underlying scripts
- The data structure of the payload delivered to your business on form submission behavior of the form on mobile devices and in mobile apps ("responsive layout") design elements that accommodate
- accessibility
- privacy
- security
- usability
- hooks for supplementary data passed on to Transaction Manager for analytics:
  - geolocation data
  - form abandonment

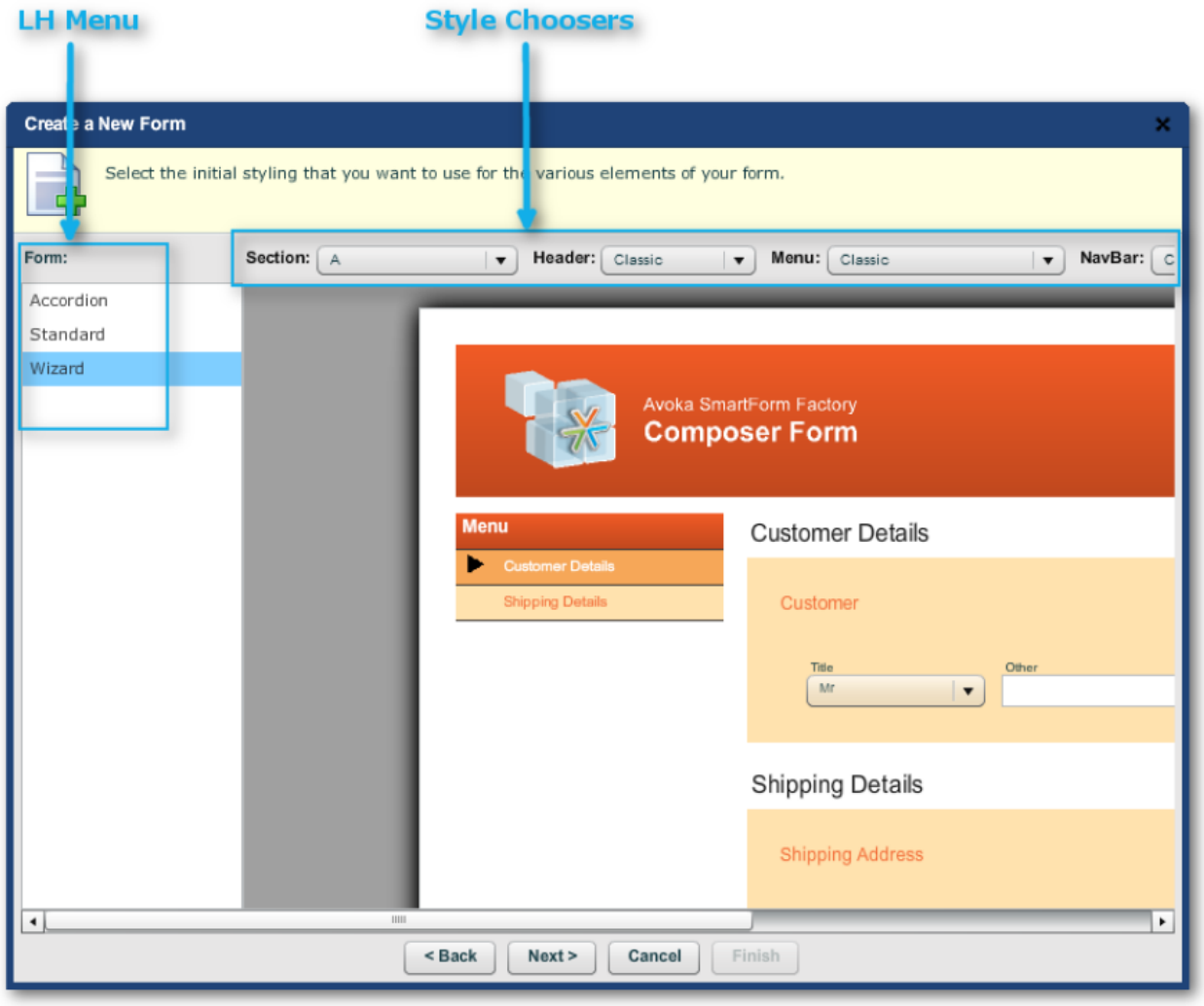
Other chrome in Form Designer, such as the Style Choosers inherited elements in the form's structure panel
This is not a complete list.
If you or your organization are new to Avoka Transact, it would be a good idea to make use of Avoka's support teams to create templates suited to your corporate needs and branding. Composer itself has the tools for maintaining your templates and even for creating new templates (with different scripts and elements), but if your business needs go beyond the standard templates, samples and tutorials (given the name of "cook books"), as is so often the case in real-world business situations, you will save time and money by turning to Avoka's professional services to get at least a set of initial templates to begin producing useable forms.
This guide's Advanced Topics tries to cover, as far as is practicable, the issues involved in template construction and tuning, in scripting, business rules and so forth. The perspective required to digest all this information, though, is best gained by using Composer to produce production forms, gaining real-world experience and seeing Composer add to your bottom line.

Out of the Box

That said, at this stage of your experience with Transact Composer you will either have:
The standard set of templates that come with the release (such as the Maguire templates) or a set of corporate templates conforming to your enterprise's branding and requirements.
So, the assumption is that you have templates to go on with for the time being. You can even begin to design forms that will end up in production, after changing the templates of the form to those suitable for publication on your portal. Composer accommodates this iterative approach. We also assume that you have an Organization and Project set up for you. You can work through the Hierarchy and create your own, providing that you have been assigned the rights to do so, but these Organizations and Projects may lack sufficient resources and templates, as configuration of these is complex for beginners and best avoided at this stage if you can.
So, let us begin at the beginning.

Creating a New Form

At the Project Level, click on "Create a new form..." and fill in the Wizard:
Give the new form a name, and an option description Pick a template from the dropdown list
(See here for a gallery of different templates to give just a taste of the variety of templates.)
The next several steps and their choices are determined by the template
o Could be choices about navigation types or styling
with possible style choosers such as: Sections, Headers, Color Themes, Menus, NavBar, Button, Color.
You can experiment with these choices, via the Wireframe preview under the Style Choosers.



Choosing and previewing templates in wireframes withint the New Form wizard.

Basic Navigation Types

Composer 4 introduces a number ways for users to navigate through forms, and now the one form can have several navigation modes on different mobile devices depending on the responsive layout settings of the form.
But for the moment, let us discuss basic navigation concepts, and leave the complications for later, after we have covered the other concepts needed to understand the rich navigation possibilities now afforded by Composer. The basic unit of navigation is the Section, and the 3 basic navigation types treat them differently.

Standard Navigation

The whole form is on the one web page. Each section has its own heading, usually a colored bar that runs across the width of the page.
The user moves through the page by scrolling down by mouse, scroll wheel, finger swipe, two-finger trackpad gesture, and so forth.
There may be elements that stick to the viewing area, for example down the left side, but these do not change how the user moves through the form.

Accordion Navigation

The form is the one web page, but sections collapse or expand as a convenience to the user. Navigation is essentially the same as Standard.

Wizard Navigation

Here, each section gets a new page. The user moves through each page, usually one at a time via "Next" and "Previous" links.
Actually, the form designer can later incorporate a number of navigation choices into the form:

strictly in sequence

the only way to navigate is through "Next" and "Previous".

LH Lists

useful for showing users where they are now and how many wizard pages they have ahead of them. The items are not selectable

menu selection

where the L:H LIst is selectable. Users can select another page out of order.

restricted menu selection

where users must complete certain sections of the form before being allowed to select some of the menu items. For example, it is common that users be allowed to go back to any already-completed page via the LH Menu. but can only progress to new pages through "Next" and "Previous".
How you implement these are advanced topics.

Combinations of the Above

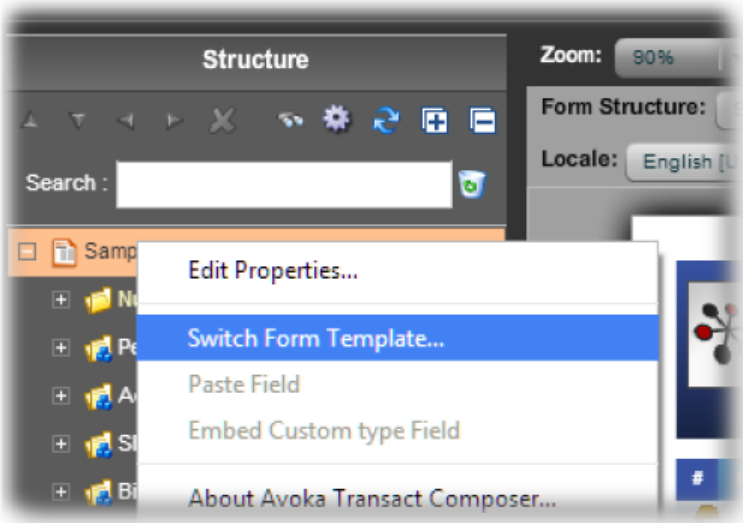
Composer 4 allows different parts of the form to make use of the different Basic Navigation Types. Now, some sections can have Wizard-style behavior while several other sections can be on one web page.

Styling

Instead of a selection of navigation types for forms having a similar design look, the one template may offer forms that have quite different designs or styles.

How to Change the Template

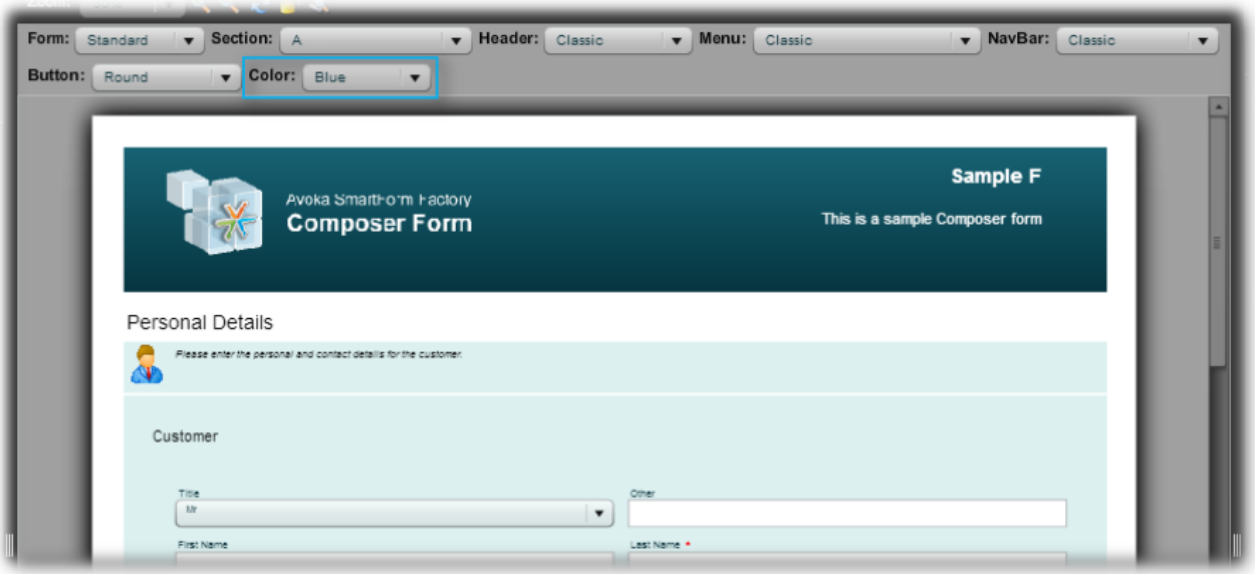
You can easily change the template of a form. Just right-click on the top element of the Structure Panel's tree and select "Switch from Template...." The only coices are templates that have been assigned to the organization (see LibraryAdvancedFeatures).
Be aware that some layout elements of the form may have to be altered as well, as required.



Right-clicking on the top of the Structure Panel tree

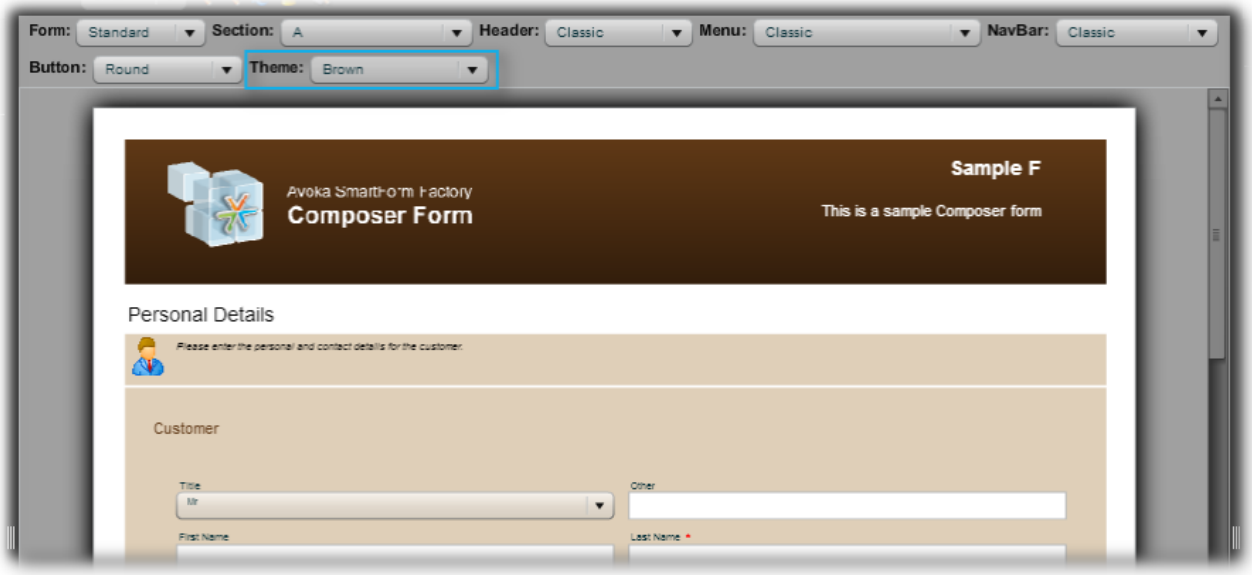
Samples of Templates and Styles

Here, we give a few examples of how templates and style choosers affect the one form's color or theme, its styling and its localization. The style choosers can also be configured (see Templates in Library Advanced Features).
The form below is a simple one with four sections for entering addresses. Some simple choices on the look of sections, of the header block (the colored rectangle with the logo), the wizard form Menu, the NavBar (which holds "Next" and "Previous"), the style of buttons (rounded, square, and the like) and Color or Theme. In these basic templates, "Color" and "Theme" are interchangeable.
Experiment with the various combinations of Style Choosers.



A basic template, and how choice of color or theme changes the appearance.

For example, changing the color or theme alters the color scheme of the form, but does not effect a radical change.



Same form, different color scheme

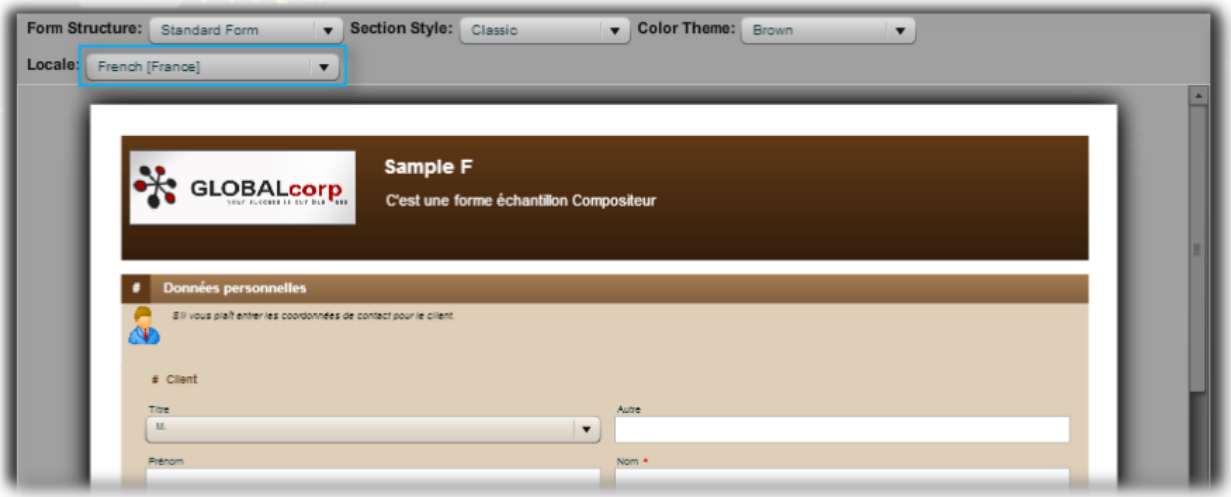
Some templates that go beyond the basics do produce radical changes to the form. Here is the same form with the "Maguire" template. The wireframe view here is only an approximation of how the form will appear to the user. Even basic navigation through the form has quite different features, such as the chevron-style menu at the top of the form, and this is only visible in Preview, or a Mobileslidermenu.



Same form, but with a more elaborate template. Note the Preview gives a more accurate reprentation of the form's appearance.

Some templates support localization. See here on how to configure and tweak this feature for a range of Locales. Once done, changing the Locale for a form in Composer is a simple matter of using the selector. Note that the following examples were automatically generated using Composer's leveraging of Google Translate, and were not checked by native speakers — usually a good idea in a production setting.







Edit Properties

Composer | Form Builder | v4.3 & Higher

The Edit Properties dialog is the control for every element and field in the form's structure. It has many parameters, which in turn are grouped into tabs, sub menus and even sub tabs.

Though the dialog is very powerful, but its settings do not need altering in most cases. That said, Composer's ability to do scripting and to fine tune the data model of the form through Edit Properties — and have these debugged and quickly up and running — is vital for successful form design. The improved layout of the Edit Properties dialog also facilitates making these adjustments to the form.

Composer Version 4's redesigned Edit Properties has a number of new features, including a layout tab for configuring the layout managers of a block, with a small wireframe for viewing the effects of changes on the block. Also, for the first time in Composer, you can view a field's dependencies and triggers.

Invoking the Edit Properties Dialog

Double click on an item in the Structure Panel. The different types of structural elements have different items in their Edit Properties dialog. Broadly, here are the different types of elements:

The form itself (at the top of the Structure Panel tree)

Sections (a special kind of container that controls navigation through the form) Containers and blocks (which can be expanded by clicking on the "+")

Simple Widgets (which do not expand)

Structure

Version 4's Edit Properties dialog has a new layout for better grouping of the many settings of the properties for fields and containers on the form. The dialog now features a row of tabs along its top. Which of these are visible depends on which of the various structural elements was chosen for the "Edit Properties" dialog.

Properties Overview Styling Data Model

Layout Assistant (for blocks and containers only) Dependencies

Rules

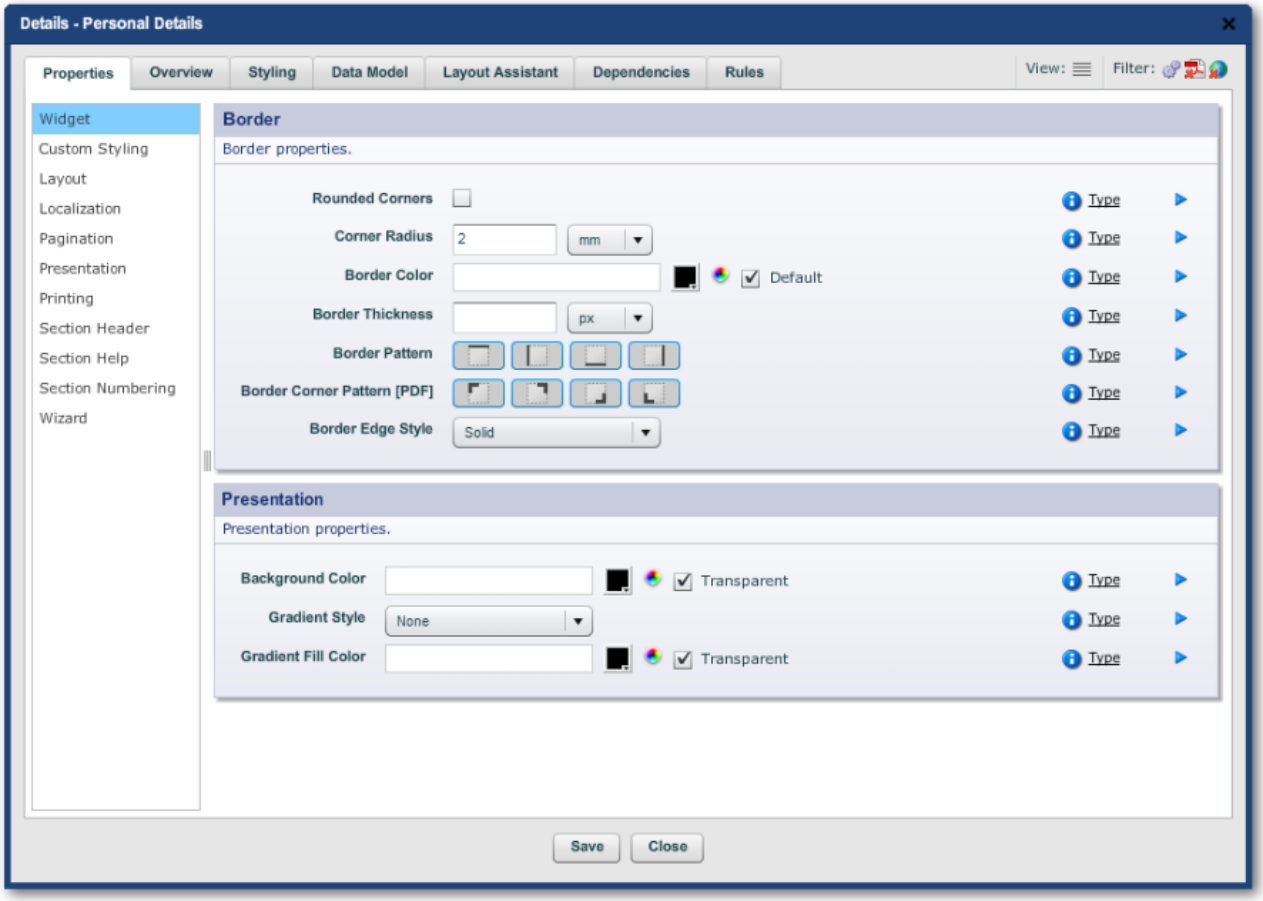
There are a few special tabs that appear for the Edit Properties dialog for the form itself at the top of the Structural Panel tree:

Palette (for the form element only at the top of the Structure Panel) Theme (for the form element only)

Policies (for the form element only) Settings (for the form element only) Localization (for the form element only)

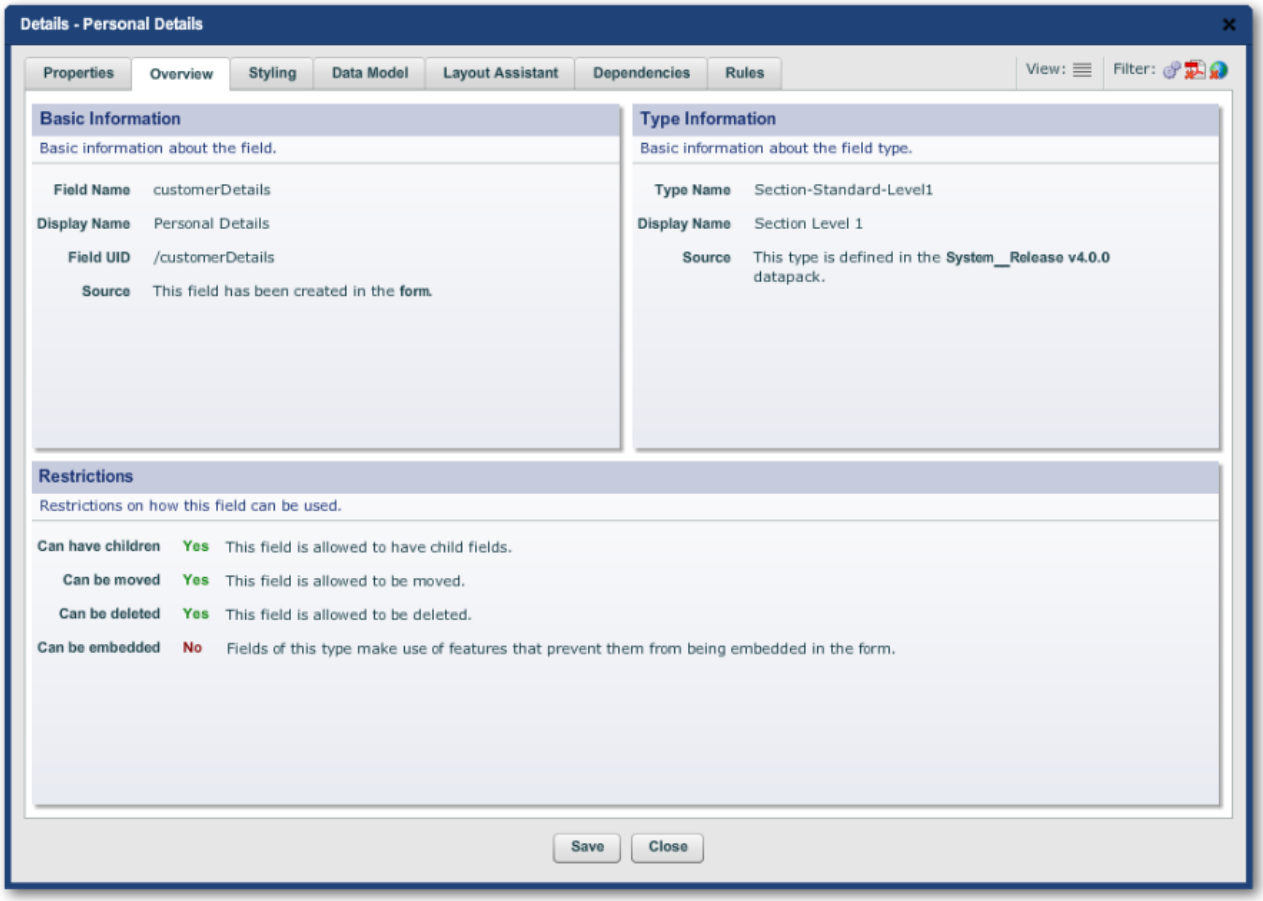
However the properties displayed in these special tabs can be accessed elsewhere in the tree, usually in the Nuts and Bolts. There, the properties appear under the "Properties" tab in the same panels that appear under these special tabs.

"Properties" and "Rules" have Left Hand menus. Each menu item displays a different set of panels to its right. Each of these panels contain their own set of editable properties.



Edit Properties dialog for a tab with a Left Hand menu

"Overview", "Styling", "Data Model" and "Dependencies" have just panels containing properties.



Most of these properties are read only. "Data Model" has editable properties. Two of the panels in the "Styling" tab are editable: "Stylesets" (which gives access to stylesheet preset options) and "Named Styles" (where you can explicitly associate stylesheets with the object).

The "Layout Control" tab, which is visible for blocks and sections only, has its own unique layout, see Field LayoutControl.

See ScriptingandDependencies on how to add scripts to the form — which is done through Edit Properties.

Notation Convention Used

In the rest of this guide, we will use the following notation to show where a property, script or configuration is located in "Edit Properties":

Tab -> LH Menu item -> Panel -> Property -> value

So, in the Overview tab shown above, for example:

Overview -> Basic Information -> Field Name -> customerDetails

The Filter and View Buttons

The "Edit Properties" dialog has a similar dualmodeastheStructurePanel: Standard and Advanced. The cog icon to switch between the two modes is in the top right-hand corner in the "Filter" mini panel. "Advanced" displays more properties, LH Menu items and tabs.

The other two icons toggle on or off the display of PDF-only or HTML-only properties.

The "Hamburger" View icon toggles between the normal Edit Properties view of tabs, menus and panels and the "Property Grid": a combined list of all properties listed in alphabetical order . Use the Property Grid to find a property if you cannot locate it in any of the panels, or to inspect the raw and displayed values of properties.

The Properties Tab

The Properties tab for any structural element usually has a LH Menu. The items displayed, though vary from element to element. The items will be listed and defined in the ReferenceSection near the end of this guide.

These LH Menu items are only groupings of property panels (see The New Dialog topic). The same property panels may also appear elsewhere in the same Edit Properties dialog for the same element, for example the panels in Properties -> Layout will also be found at Layout Assistant -> Properties in the same dialog.

So, the Properties tab's LH Menu usually has many possible entries. The following is intended to be a broad guide so that you can find your way around this tab — and, by implication, the rest of the Edit Properties structure.

Here, we exclude Nuts & Bolts. Indeed, some of the "Nuts & Bolts" items have Property tabs with empty LH Menus

The following table is a rule of thumb only, and we include it here for general guidance. Advanced Mode items are so marked.

Structural Element Type	LH Menu Item	Panels (an indication only)
Form	Form	Form Panel (Holds the settings entered in theoriginalwizardwhentheform was created.)
Section	Widget (Advanced)	Border Presentation (Borders, background colors and gradient)
	Custom Styling (Advanced)	HTML Custom Styling (HTML forms only. Background image, its repetition, custom HTML classes)
	Layout	Layout Manager Layout Constraints Sizing (i.e. the layout fill: "Horizontal" or "None")
	Localizatio n (Advanced)	Localization Properties

	Pagination (Advanced)	Pagination control (PDF only)
	Presentatio n (Advanced)	Presentation (Alignment, Drop Shadow, etc.)
	Printing (Advanced)	Printing (PDF only. Printing properties)
	Section Header	Header
	Section Help	Help
	Section Number	Numbering
	Wizard	Wizard (does this section belong to the form's wizard navigation scheme)
Block	General	General (Labels)
	Widget (Advanced)	Border Presentation (Borders, background colors and gradient)
	Custom	HTML Custom Styling (HTML forms only. Background image, its repetition, custom
Structural Element Type	LH Menu Item	Panels (an indication only)
	Styling (Advanced)	HTML classes)
	Layout	Layout Manager Layout Constraints Sizing (i.e. the layout fill: "Horizontal" or "None")
	Localizatio n (Advanced)	Localization Properties
	Pagination (Advanced)	Pagination control (PDF only)
	Presentatio n (Advanced)	Presentation (Alignment, Drop Shadow, etc.)

	Printing (Advanced)	Printing (PDF only. Printing Properties)
Simple Widget	Caption	Caption Content Caption Presentation
	Field (Advanced)	Field Border Field Presentation (These give the border, background and so forth of the area into which users type their data)
	Widget (Advanced)	Widget Border Widget Presentation (The border, background and so forth for the widget)
	Accessibilit y	Accessibility (Tool tip, custom screen reader text, precedence and so forth)
	Custom Styling (Advanced)	HTML Custom Styling (Custom HTML Classes and whether Boostrap styling applies) PDF Comb Styling
	Data	Data (various properties, many of them Advanced Mode only, including display format, etc.)
	Layout	Layout Constraints Sizing
	Localizatio n (Advanced)	Localization
	Printing (Advanced)	Printing (PDF only)

The Other Tabs

In TheProperties Tab, we gave 4 types of structural elements in the panel. There are actually 5 main types:
The form itself (at the top of the structural panel's tree)
Read onlyelements (mainly found in Nuts & Bolts elements)

Sections Blocks

Simple Widgets

The New Dialog lists the range of other tabs. The current topic briefly deals with two tabs associated with all the elements:

Overview

Styling

Neither has a LH Menu.

Tab	Panels	Editable?	Properties
Overview	Basic Information	No	Field Name Display Name Field UID (the unique identifier of the element) Source (Whether created directly in the form or not)
	Type Information	No	Type Name Display Name Source
	Restrictions	No	Can have children? ("yes" or "no") Can be moved? Can be deleted? Can be embeded?
Styling (Not available in the form element at the top of the tree) (see Stylesheets) **Note**: None of these properties are covered in Properties -> Custom Styling (HTML only)	Stylesets (qv)	No	Lists any stylesets associated with the element
	Typed Styles	No	Lists stylesheets assigned to the element according to its type.
	Named Styles	Yes	Can explicitly add or remove stylesheets to the element and set the order of precedence.

Field Layout Control

This topic covers the mechanics of layout. See here, for a discussion on the aesthetics and design of form layout from. Also see another discussion on UsabilityandAccessibility.

Problems with explicit position layouts

When designers build a form using simple tools, they tend to drag a widget into a specific position on the form — known as absolute or x-y positioning — then re-position and re-size it using their mouse. This works for simple forms only, not when dealing with multiple device types, nor with maintaining existing forms. Explicit positioning:

Cannot ensure that forms flow correctly on different technologies, for example on the desktop, tablets or smart-phones.

Cannot ensure all fields have the same styling and sizing. Cannot ensure that all fields line up nicely.

Cannot ensure proper placement on the high resolution devices. Doing this manually is very time consuming and the results are inconsistent.

Cannot save designers from filling in parameters manually into dialogs. For example, "all standard text fields are 1.5in wide, a caption of 1.2in, and placed 0.7in in from the left margin". The designer then needs to know these magic numbers, and apply them consistently and enter them for every single field on the form.

Makes a big job out of inserting a new field vertically between two existing fields.

Cannot automate changing all the fields on the form to have, for example, all the field captions left or top- aligned, or changing all the fonts or background colors.

Cannot automate changing the width of the form, or its margins and so forth. Instead, every field has to be reposi-tioned.

Layout Managers and Hints

For these reasons, Composer automates the detailed layout of fields. All you do is specify the order of the fields and, optionally, a few extra "hints" and let Composer do the rest.

Hints may be such things as:

Keep this field on the same line as the previous one

Make this field as wide as possible (within the bounds of the space that it has available on the page) Spread these two fields across the page, in the ratio 40% to 60%.

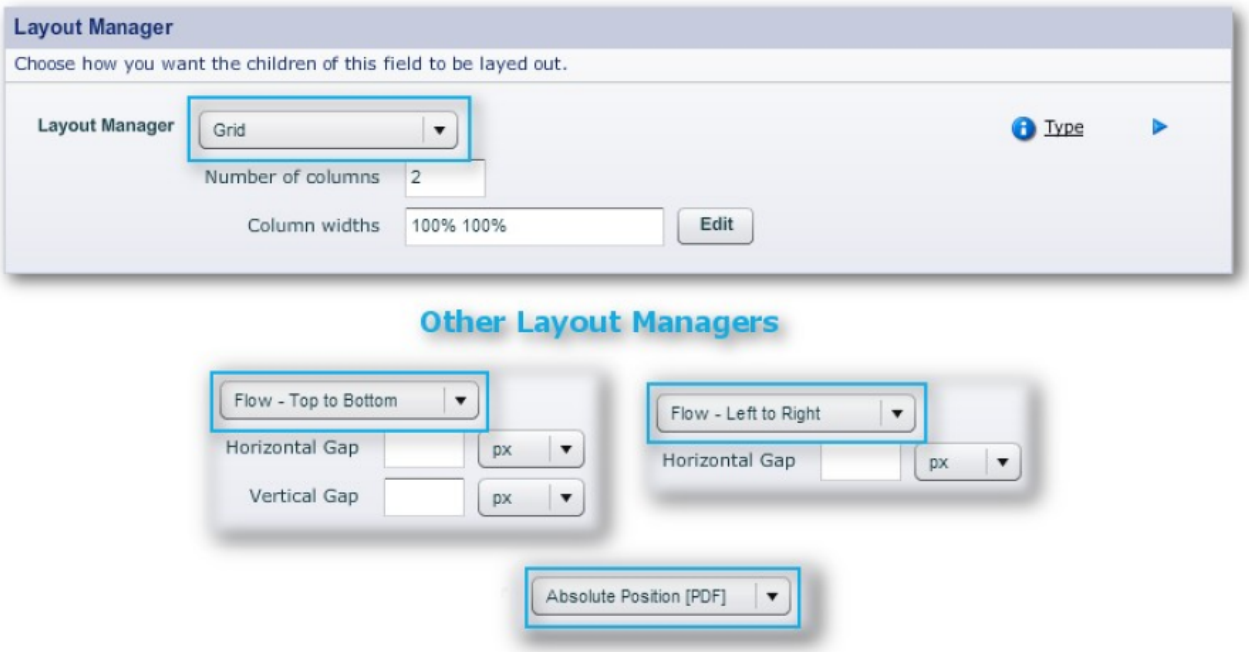
Lay out these 12 fields equally spaced in a 4 column grid

The Layout Panels

The default settings for the Composer layout managers are set in the template for the form. The following documents how to tweak the layout of elements on the form when the default positioning of an element needs it.

You make layout adjustment through 3 panels in Edit Properties. They are:

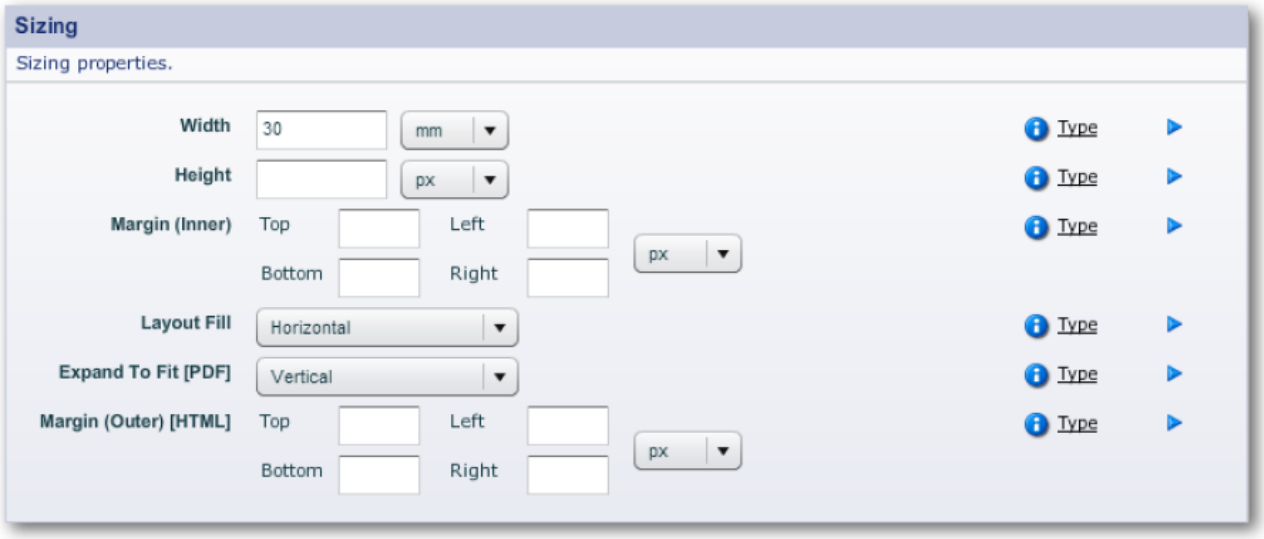
The **Layout Manager** panel



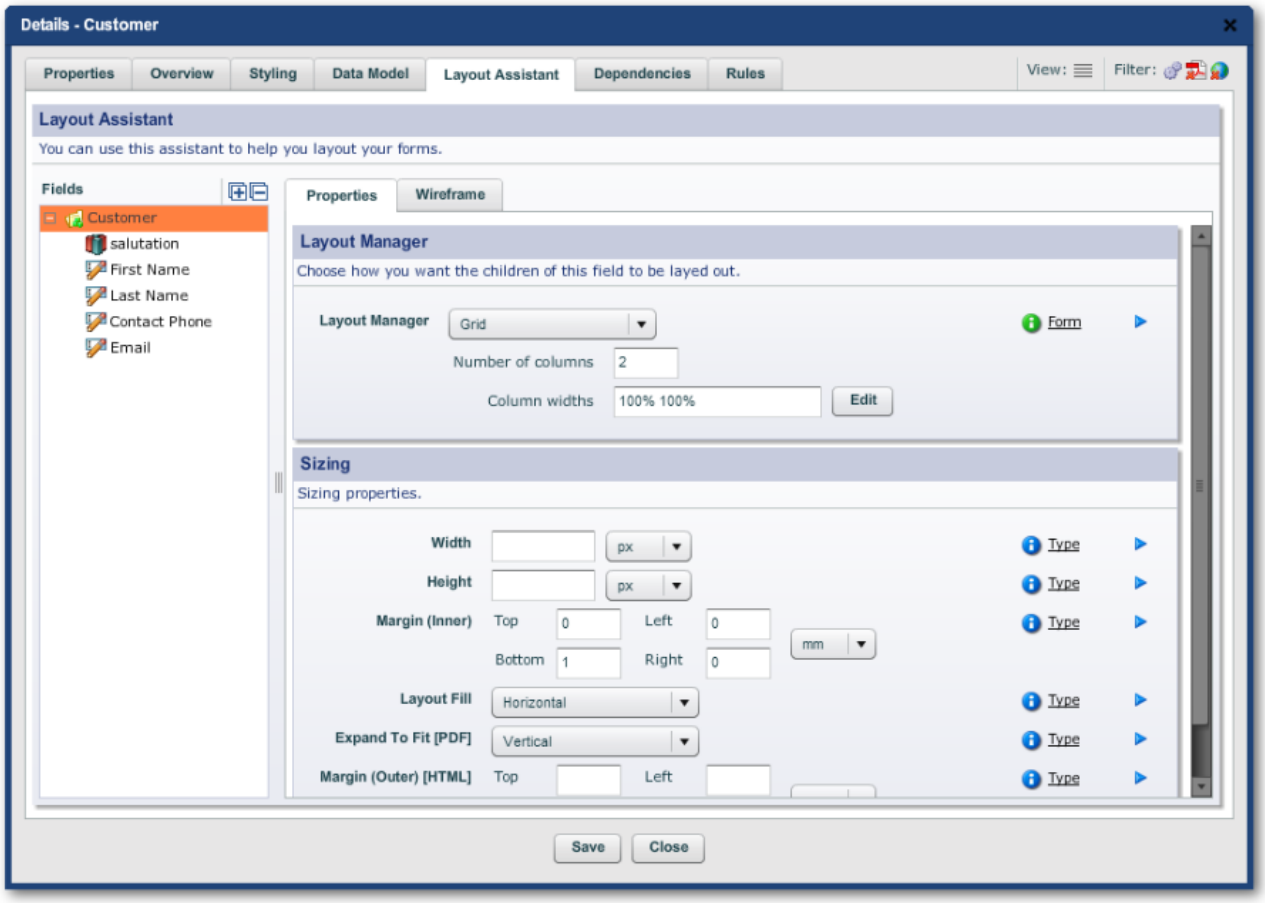
Layout Constraints



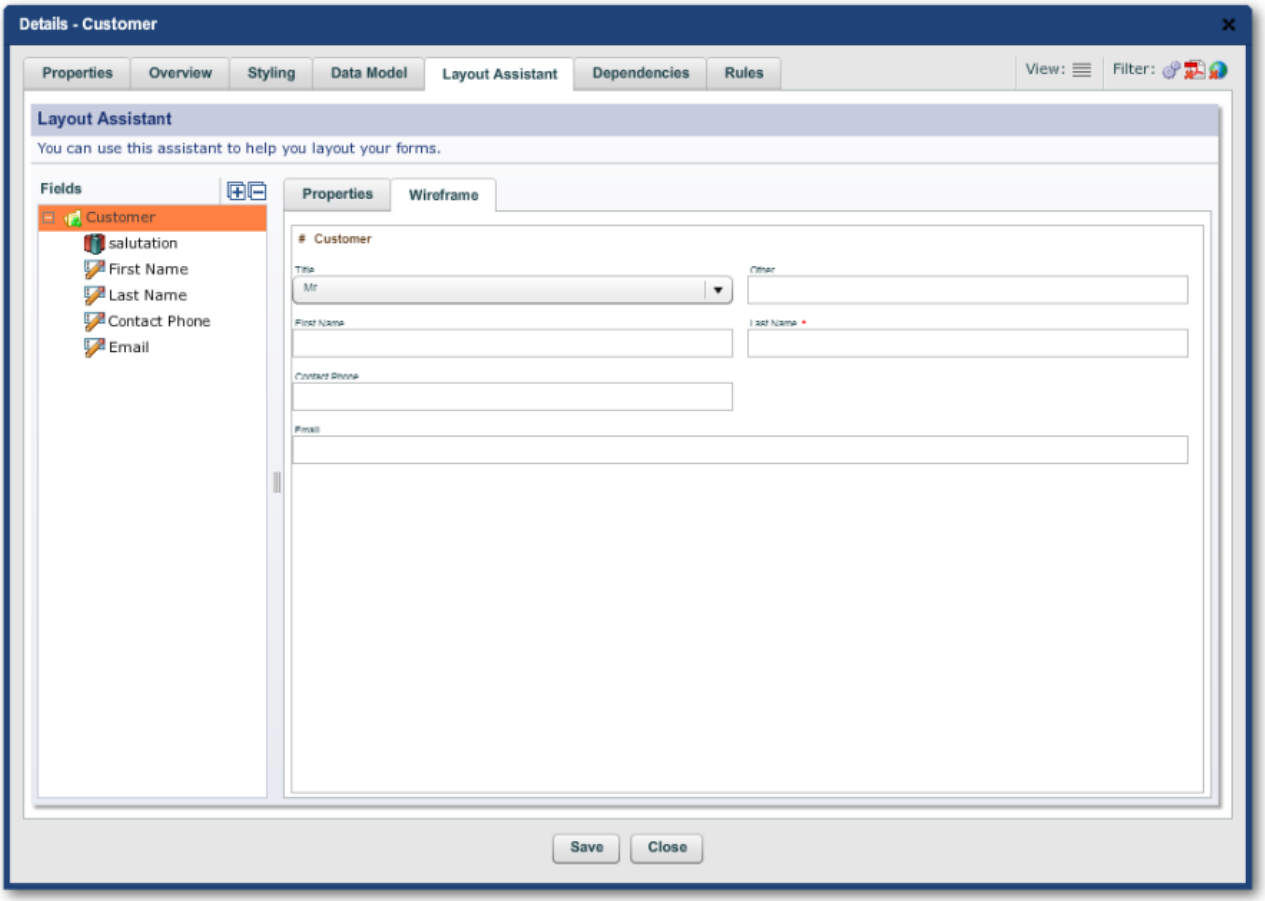
Sizing



You access these panels in "Edit Properties" through "Properties -> Layout" or in the "Layout Assistant" tab. (That tab, though, does not contain the "Sizing" panel.)



Note the mini Structure Panel (marked "Fields") in the LH Menu of the tab. And, there is also a mini wireframe available on the RH side of the tab in which you can test the effect of tweaking the values in the "Layout Manager" and "Sizing" panels.



The Effect of the Different Layout Managers

Flow — Top to Bottom

A lot of forms are very simple - they just contain a series of fields going down the page. This is also the best layout for the narrower screens on mobile devices.

To achieved this type of layout, you can use the Top to Bottom flow. Fields in a Vertical Flow simply go below each other down the page. The Vertical Flow is always the default manager on all sections and blocks.

There are a few options or "layout hints" that you can specify with the Vertical Flow layout manager to achieve higher levels of usability. These include:

Property	Description

Keep on Same Line (Layout Constraints Panel)	If "Keep on Same Line" is checked, this field will be kept on the same line as the previous field. You can set this on as many fields as will fit on the line. This technique is mostly used to group related field together, such as First Name and Family Name, or State and Zip Code.
Layout Fill (Sizing Panel)	If Layout Fill is set to "Horizontal", then this particular field will expand to fit all the available space on the line. This can be used to give your users as much space as possible to fill out longer fields, without having to give your fields explicit widths. If you have multiple fields on the same line with Layout Fill set as "Horizontal", each of them will then share the extra available space equally.

The Top to Bottom flow does not change the width of any of your fields (unless you select Layout Fill). Your field width will be:

The "natural" width of the field as defined in your organization stylesheet; or

The style-set that you associated with this field whenyoudraggeditintotheform. Typical style-sets include widths like "Tiny", "Small", "Medium" and "Large", but this will vary depending on how your organization's stylesheets have been set up. If you want to change the style set after you've created the field, you can change it in "Styling -> Stylesets -> Field Size".

If you select Layout Fill, the layout manager will try to use as much of the line's space as possible and take into account the size of the screen or page, margins, borders and gutter areas, space allocated to menus, and the other fields on the same line.

Note: If you specify too many fields on the same line, this can create layout problems, particularly on smaller screens. The behavior of a line with too many fields will depend on the ResponsiveLayout rules.

Flow — Left to Right

The Top to Bottom Flow is suitable for many simple types of forms; you often need, however, more fine-grained control over how fields spread out across the page. This is where the Left to Right Flow layout manager, comes in.

Horizontal Layout allows you to lay out several fields across the page with very fine-grained control over the size of each field by specifying the percent width of each field.

Note: While we generally talk about percentages, those percentages need not to add up to 100: Composer will adjust accordingly.

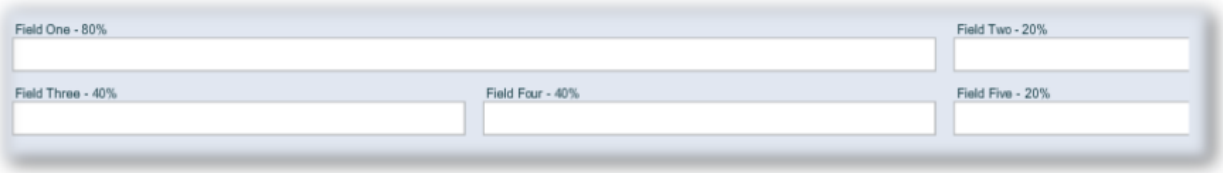
The beauty of using percentages is: should you need to add another field onto the same line, move a field, re- size a field, or switch from portrait to landscape, all the other fields will automatically adjust to accommodate the changes. Form maintenance then becomes easy. The fields will also retain their proportions on the range of mobile devices.

Left to Right Flows are always used in conjunction with Top to Bottom. The Top to Bottom controls each line of fields; and Left to Right within each line.

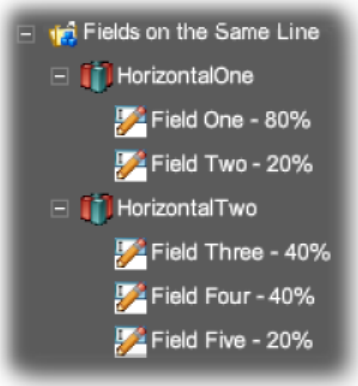
Multiple fields per line

The simplest way to get multiple fields on the same line is to use a Top to Bottom Flow and the "Keep on Same Line" property ("Properties -> Layout -> Layout Constraints"). This, however, gives you little control over the sizing of the fields.

For more control, use a Left to Right flow inside a Top to Bottom to finely control the widths of each of the fields in the single line. Using even multiples of a base size will result in fields on subsequent lines being lined up to each other. This is shown in the screenshot below.



Horizontal Layout



Horizontal Layout in the Hierarchy

Two-column forms

You can create a two column layout by using two blocks laid out within a horizontal flow layout, with fields inside them laid out horizontally.

This is shown in the screen-shot below:



Two Column Layout



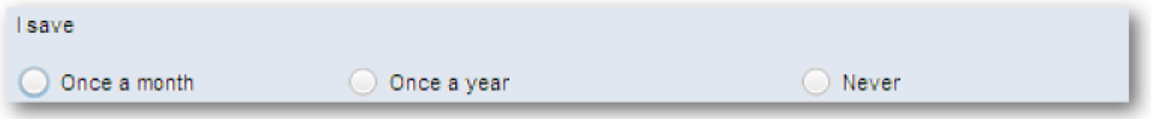
Two Column Layout in the Hierarchy

Note: You can use the Margin property to provide extra spacing between the two columns, or use another smaller block between the two.

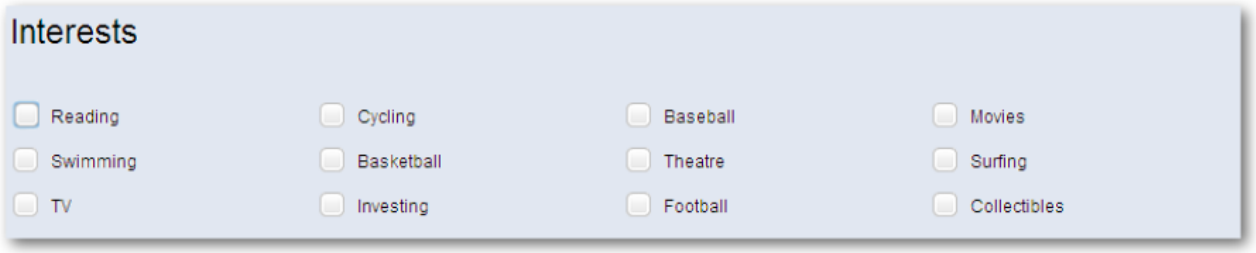
Grid Layout

Often, you have to lay out a large number of fields and, for an elegant layout, all these have to align correctly. The Grid layout manager, which is specified on blocks or sections, can achieve this.

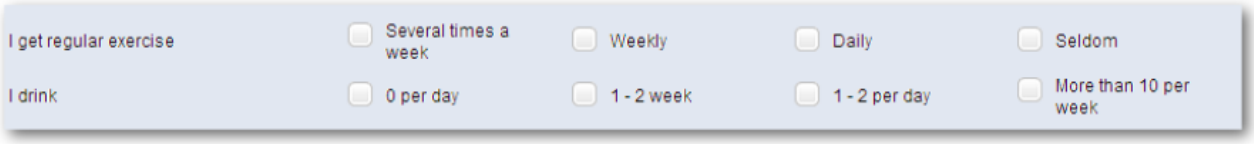
By default, each column in the grid has equal size, but you can also specify the width of each column by defining a percentage of the space available, or a fixed size in mm or inches. For example, in a 3 column grid, you may specify the column widths as "100% 60mm 100%". As in the example, these percentages need not add up to 100.



Grid layout with 3 radio buttons respectively 100%, 60mm and 100%



Grid layout with 4 columnseach one set to 100%

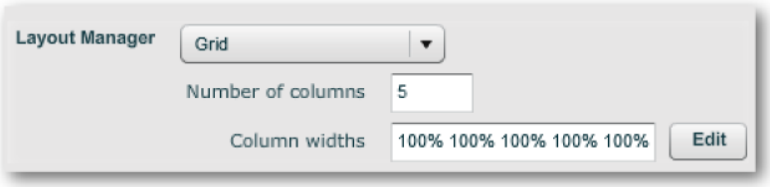


Grid layout havinga text fieldset to 150%and each checkboxset to 100%

Grids can also be used as part of a hybrid layout. For example, a grid could contain one or more blocks which in turn could contain widgets, or grids could be arranged within a vertical or horizontal flow layout.

Each of the objects added inside the grid can specify hints that provide additional control over how they are laid out.

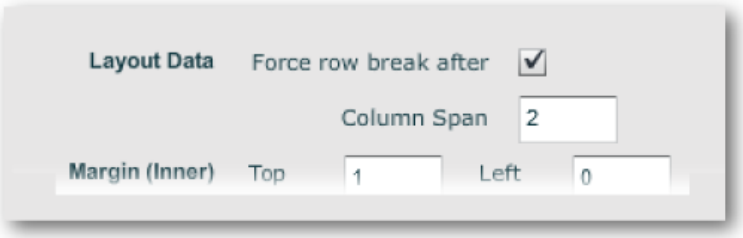
Go to next line Span



Property editor for the parent block

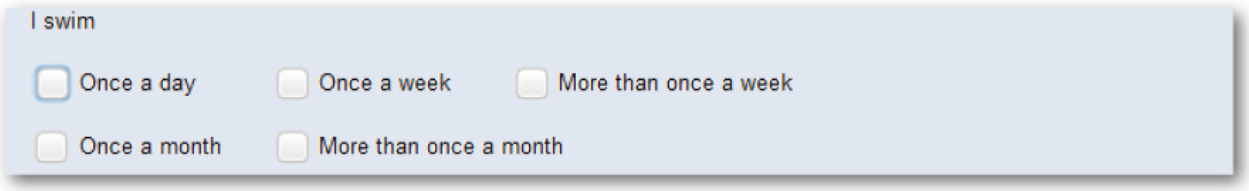
In the example below, I begin by using as I have 5 checkboxes.

Because 'More than once a week' is such as long caption, I select the 'Force a row break after' that checkbox and make it "span 2 columns".



Tweaking the element with "Edit Properties -> Layout -> Layout Data"

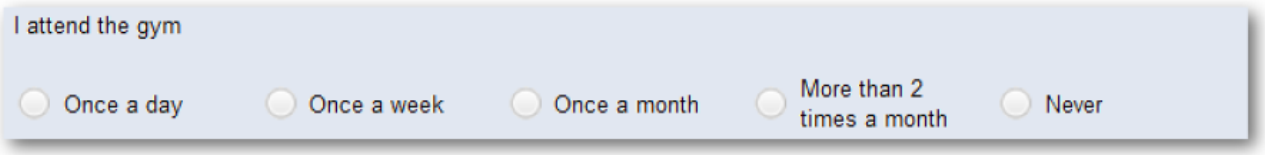
Result:



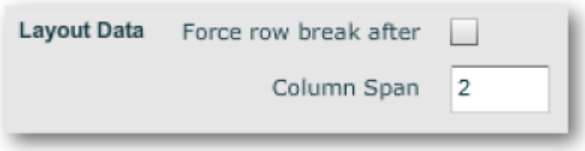
Grid layout where 3rdcolumn spans 2columns and the4th column isforced onto thenext line

Tip: Grid layout is particularly useful for laying out multiple checkboxes and radio buttons. Even if your radio buttons all appear on one line, it's still advisable to place them in a grid layout. Decide on a standard grid size for all radio buttons, and always use that grid size for all radio button groups, regardless of how many radio buttons you have. If you do this, all your radio buttons in your entire form will all line up nicely.

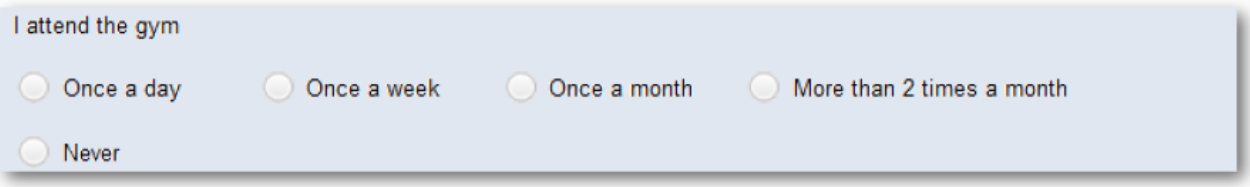
Tip: If you have a radio button with a longer than usual caption, simply set the span to 2 so that it uses up two grid cells.



Grid layout where each of 5 columns set to 100% (Note: the caption on the 4th column)



Property editor for 'More than 2 timesa month'



Grid layout where4th column spans2 columns toprevent caption wrapping

Fillers, Springs and Page Breaks

There are a few special objects in Composer that deal specifically with laying forms out.

Object	Description

Filler	A filler is a simple invisible object that simply takes up space, creating "white-space" in your form. White-space is generally considered a good thing by form designers, since it creates visual separation and grouping between objects, and makes a form easier to read and complete. You can use a filler to separate two other objects either vertically or horizontally. For example, you can put a filler after a heading to create some space between the heading and the first field. Fillers can sometimes be useful, but they are difficult to style - if you want to be able to control the look and feel of your form by changing the stylesheets, you're generally better off using properties on the individual objects that can be more easily styled, such as margins.
Spring	A spring is exactly what it sounds — it's an object that tries to expand as big as it can. For example, if you want a layout with one field on the left of the page, and another field on the right of the page, simply use two fields in a horizontal layout and place a spring between them. Under the covers, a spring is really just a simple invisible object with its width set to 100%. Springs only work with other objects on a single line.
Page Break	Page Breaks only apply to page-oriented formats such as PDF. By default, Composer builds forms that just flow down the page. Fields will automatically "roll over" to the next page when there is not enough room on the current page. Usually, this works fine, but occasionally, you may want to explicitly start on a new page. In order to do this, simply drag a Page Break object into the tree, and your form will immediately skip to the next page.

There are three other properties in Composer that affect page breaks in PDF forms, in the "Properties -> Pagination -> Pagination Control" panel:

Property	Description
Allow Page Breaks	Containers such as blocks and sections allow you to specify whether they should allow page breaks within themselves or not. If you turn "Allow Page Breaks" off, then the entire section will always stay on the same page - if even one field of the section would flow over to the next page, the entire section will be moved to the following page. If you turn "Allow Page Breaks" on, then if one of the fields in the section exceeds the current page, just that field and the ones that follow it will move to the next page. Most stylesheets have Allow Page Breaks turned on for two reasons. Firstly, disallowing page breaks can create large blocks of white space at the end of pages, confusing the user. Secondly, in dynamic forms (forms that grow and shrink), making a very simple change on one page could cause a following section to exceed the current page, and when the entire section is moved to the next page, it will seem to "disappear". Users find this very confusing. You should also be very careful to ensure that any section that disallows page breaks is no bigger than a single page, otherwise Reader will not know how to lay it out.

Force Page Break	This property is associated with a block, and will force this section or block to start on a new page. This is equivalent to inserting an explicit Page Break object into your form.
Block Keep With Policy	Choice between: "None", "Previous" or "Next". In other words, do you want to keep this block on the same PDF with the next block or the previous block, or or you don't care either way.

Field Styling

The form's template sets the style defaults for its fields, and so any globalchangesshouldbemadetothe template itself. That is an advanced topic, not covered here.

Local changes can affect only a single field or whole containers or sections. There are 2 types of these changes:

Changes through stylesheets Manual overrides to stylesheets

Manual overrides are easy to understand but are time consuming and as clumsy as explicit position layouts.

Using Stylesheets

In the WorkspaceHierarchy, Stylesheets live in Libraries. Libraries are associated with Projects or the higher levels of the hierarchy. See LibraryAdvancedFeatures.

You can also create *ad hoc* stylesheets (that is, those not contained in a library) at the Organizationlevel through the Stylesheet tab. See Stylesheets for more.

To do either, you need to have the necessary access levels.

Given that you now have suitably modified stylesheets, you can make use of these through the "Styling" tab of "Edit Properties" of the element. This tab is available for all elements except the form element at the top of the Structure Panel tree.

The tabs has 3 panels:

Stylesets

where designers can choose a style from a set of them as defined in the stylesheet, The most commonest example is "Field Size", which often appears in the wizard when a widget is dropped onto a form's Structure Panel.

Typed Styles

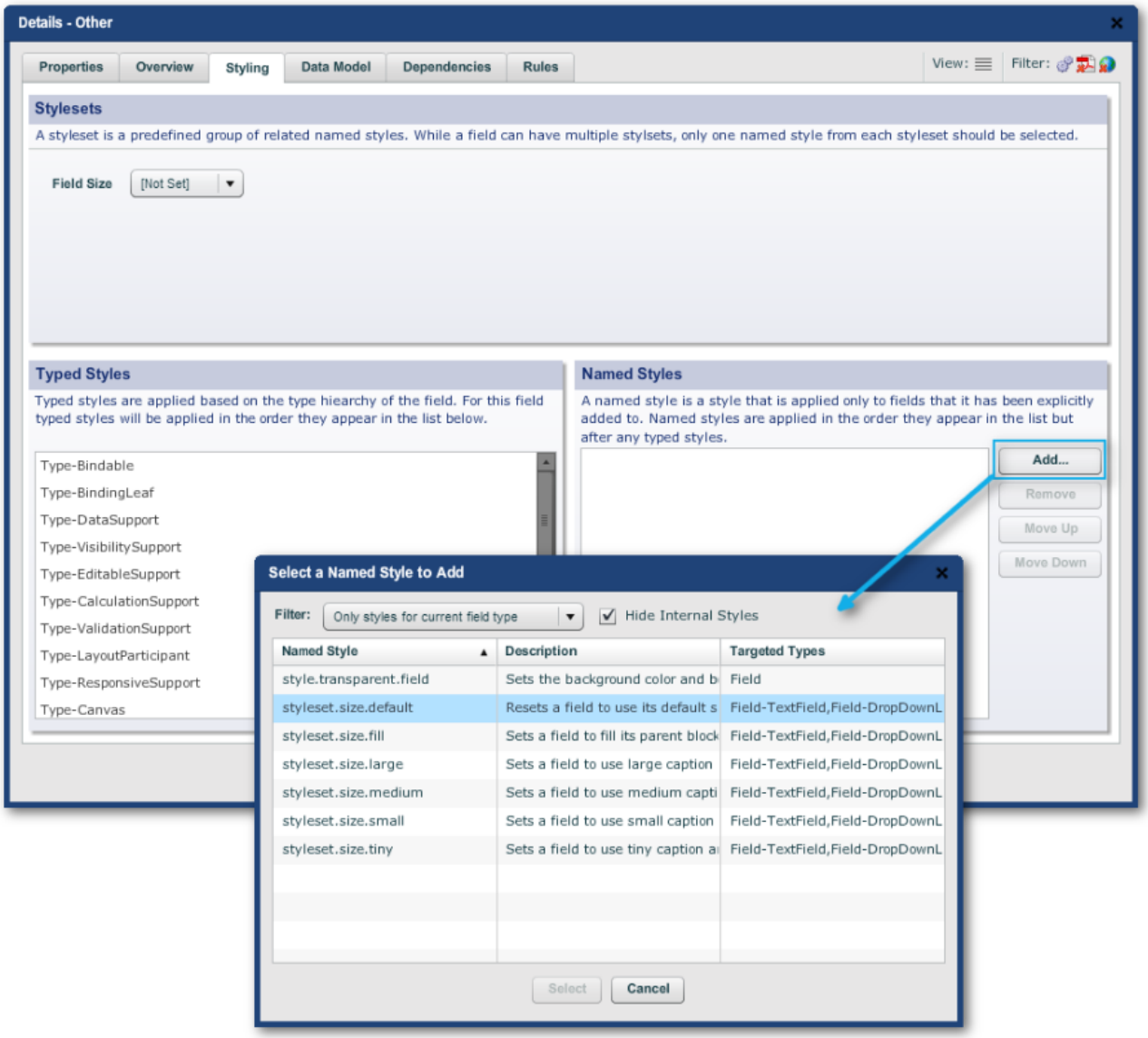
a read-only field that displays the style types associated with the field. These types are defined and assigned in the stylesheet. See Stylesheets.

Named Styles.

These 2 kinds of styles are explained in detail later, For now, suffice it to say that:

typed styles are based on field hierarchy or field function, Their allocation is automatic and based on the field's place in the form's structure

Named styles are allocated manually, and their functions are more local, such as for setting a background color or making a field transparent, and so on.

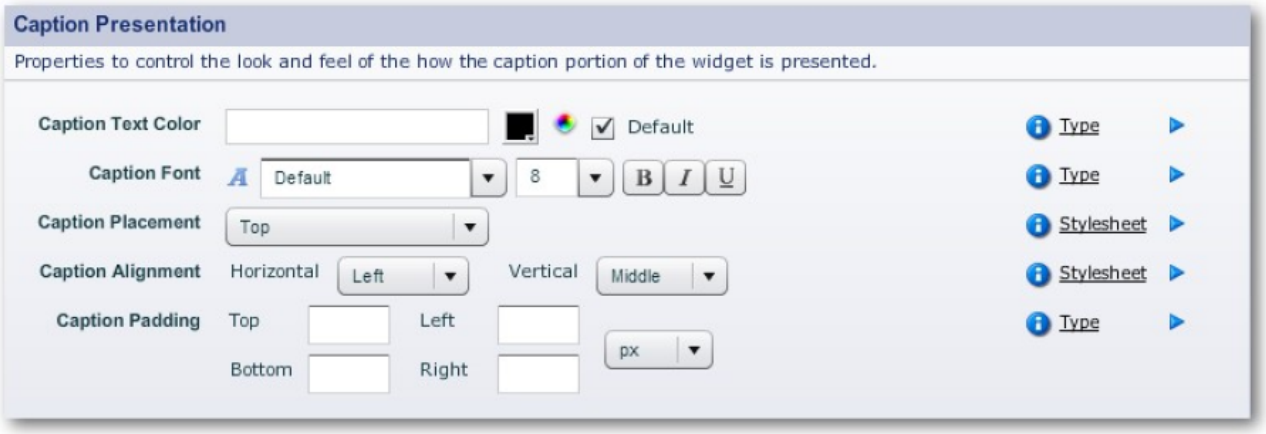


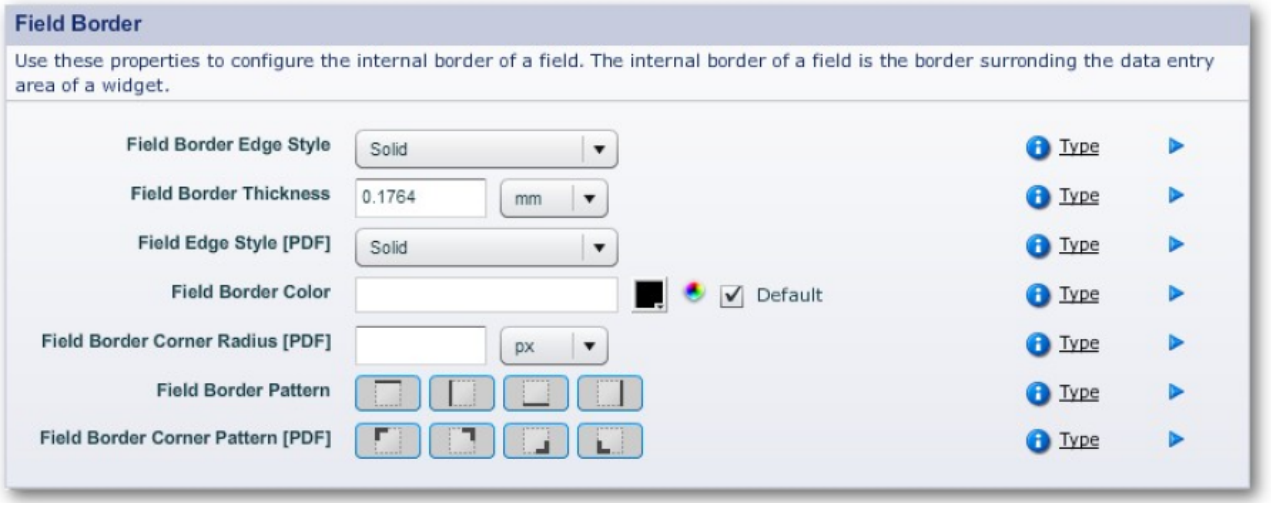
Manual Overrides

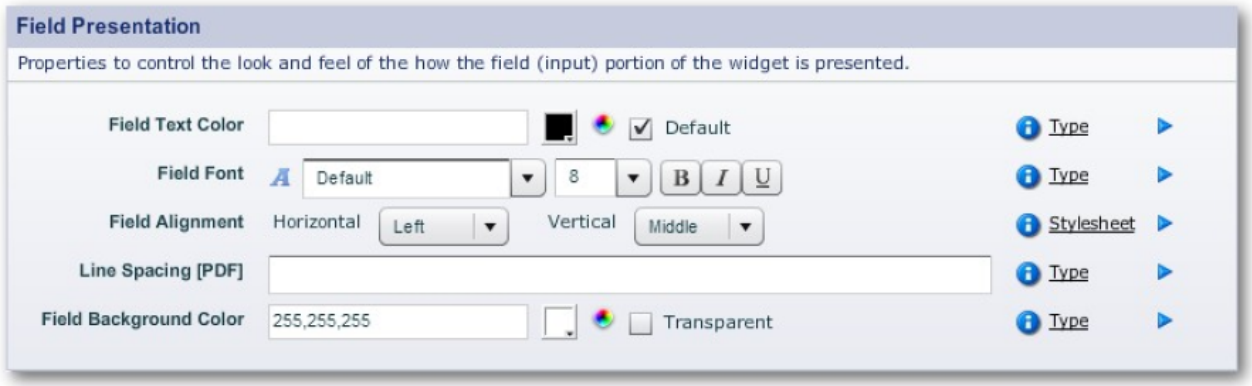
These should be one-off overrides. (If you find that you are continually applying manuals throughout the form, you should be changing the template instead.)

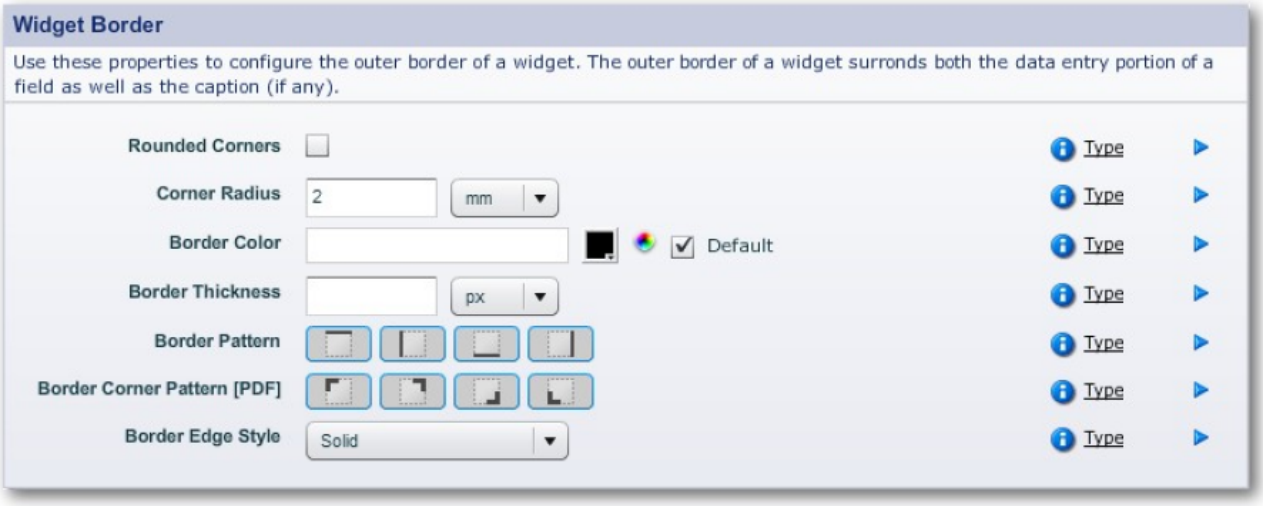
The relevant panels are the "Presentation" and "Border" panels:

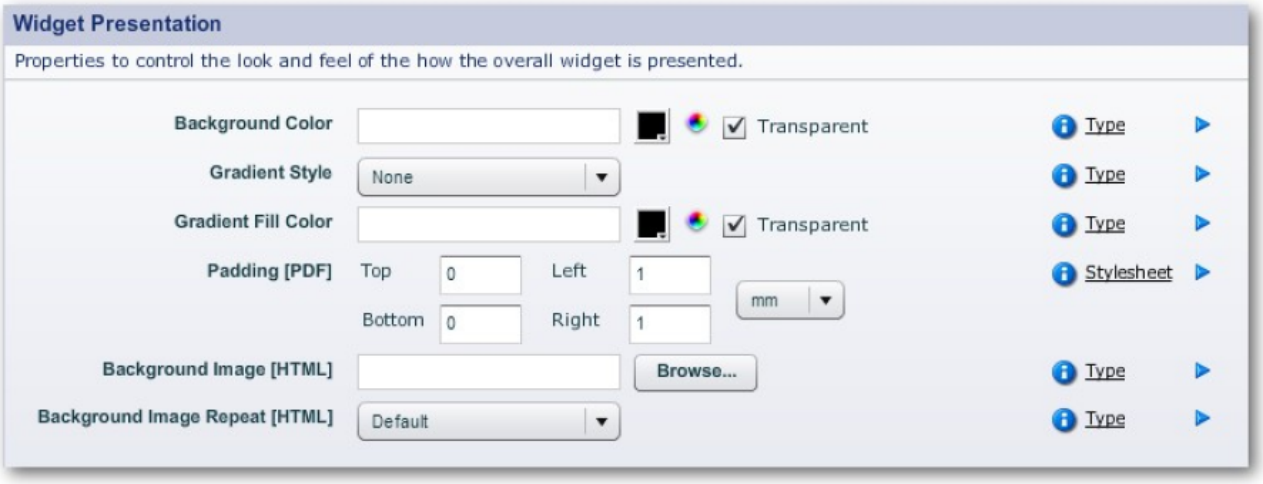
"Properties -> Caption -> Caption Presentation" "Properties -> Field -> Field Border" "Properties -> Field -> Field Presentation" "Properties -> Widget -> Widget Border" "Properties -> Widget -> Widget Presentation"





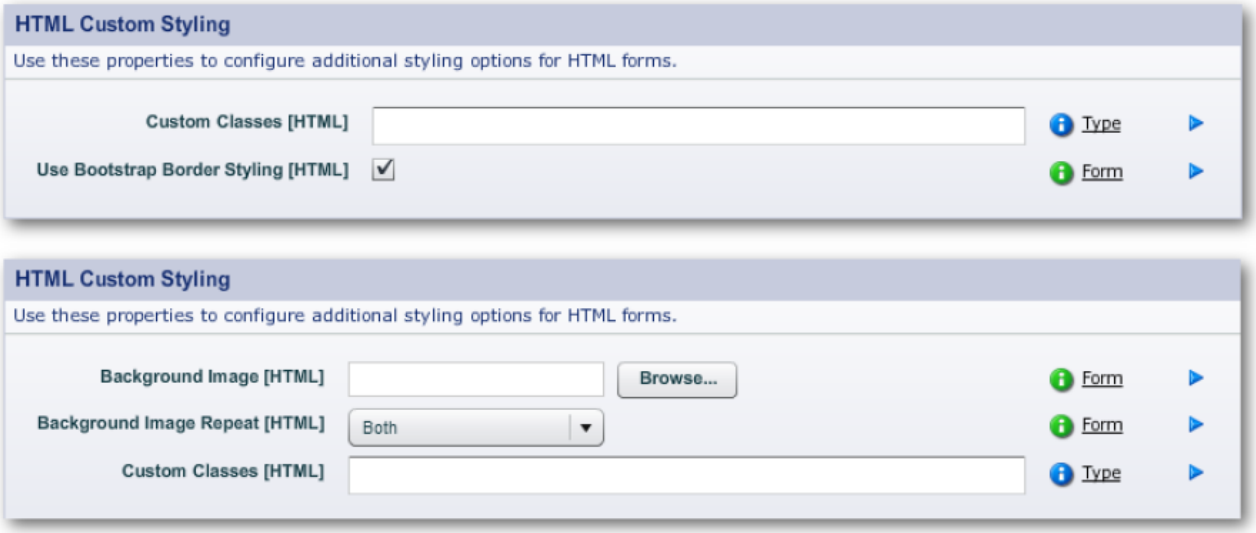



Widget Border
Use these properties to configure the outer border of a widget. The outer border of a widget surronds both the data entry portion of a field as well as the caption (if any).
Rounded Corners
Corner Radius
2
mm
Border Color
Default
Border Thickness
px
Border Pattern
Border Corner Pattern [PDF]
Border Edge Style
Solid
Type


Widget Presentation
Properties to control the look and feel of the how the overall widget is presented.
Background Color
Transparent
Gradient Style
None
Gradient Fill Color
Transparent
Padding [PDF]
Top
0
Left
1
Bottom
0
Right
1
mm
Stylesheet
Background Image [HTML]
Browse...
Background Image Repeat [HTML]
Default
Type

Custom Styling

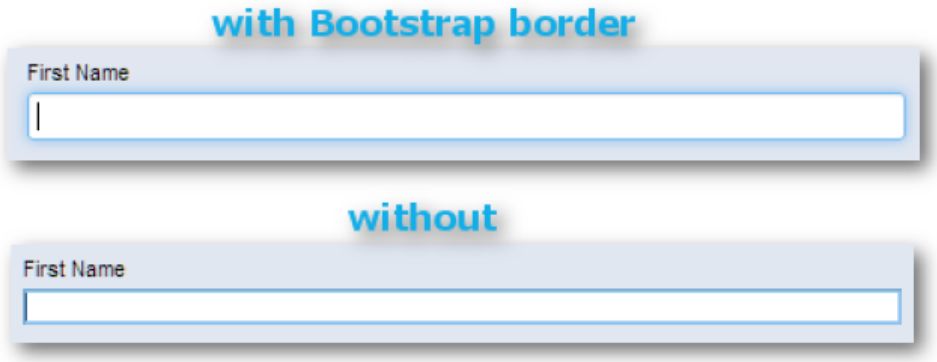
This technique stands apart from the other styling adjustments.There is a panel for HTML-only settings, and another for PDF-only comb-fields.



2 sample "HTML Custom Styling" panels

Custom Classes means CSS Classes from a CSS stylesheet (a quite different entity form a Composer stylesheet) that is not under Composer's environment or control. One scenario is: the Custom Classes belong to a corporate CSS stylesheet. If so, adding these CSS classes via this panel can result in issues at rendering time, and are not an ideal solution.

"Bootstrap border styling" refers to Twitter's Bootstrap, a popular framework for implementing responsive layout behavior. The effect of having this border styling turned on is rather subtle:



A selected text field in preview

The Background Image parameters use a bitmap image file, .png, .jpg and so on, as "wallpaper" inside the object. The "Repeat" dropdown menu can be set to "Default", "Horizontal", "Vertical", "Both" or "None".

Field Data

Overview

A **writable** field's data is dealt with in 2 ways in Composer:

Data properties

which is about the data contents of the field, but considered in isolation from the rest of the form, with the exception of radio buttons that form part of the one group (as they are, after all, just the one data object with several form fields working together)

Data Model

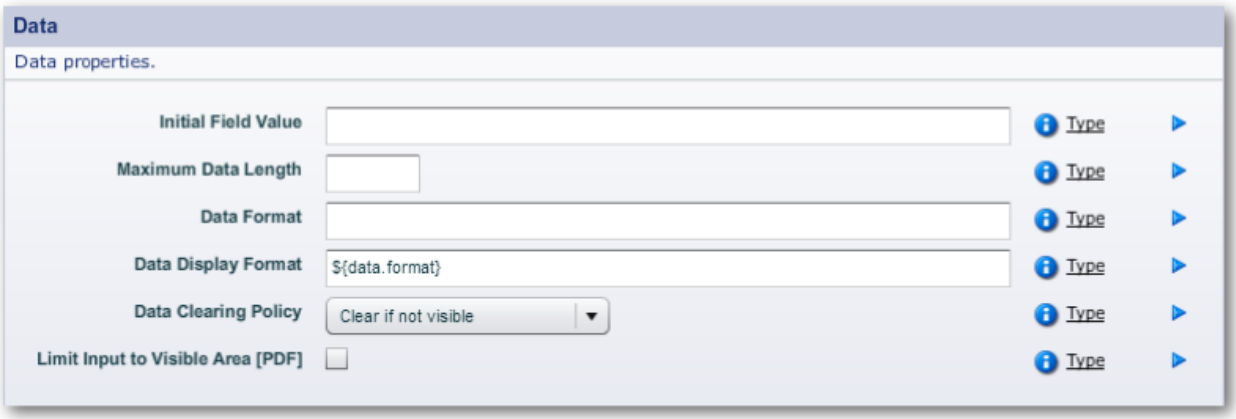
which is how the relationship of the field to the data structures of the form's outputs

Data Properties

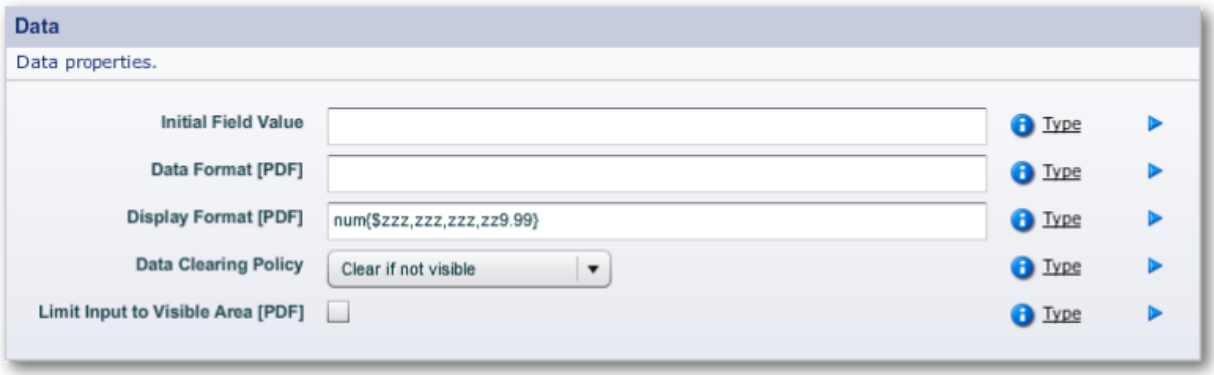
These are contained in "Edit Properties" in "Properties -> Data". There are a number of panels, depending on the type of field:
Data (always present in writable fields)
Dropdown Data (in addition to the above for a dropdown menu field) Decimal Options (when the field is a decimal number)

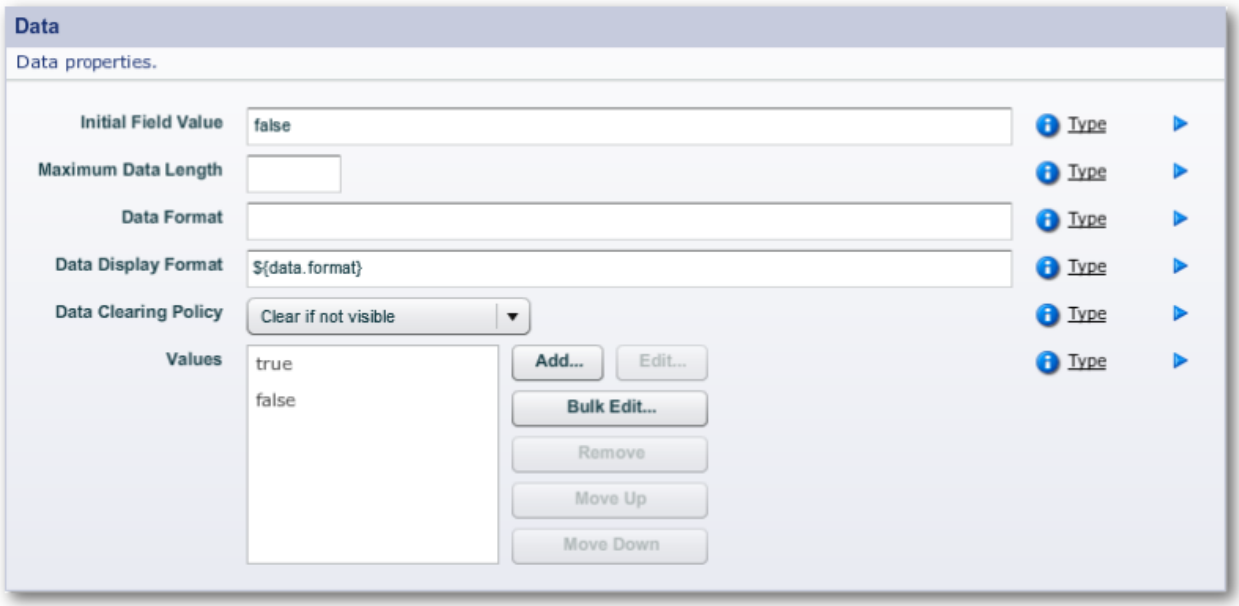
Data Panel

The properties shown in both the Advanced and Standard Modes of the panel vary with the nature of the field. Here are some typical examples:


Data
Data properties.
Initial Field Value
Maximum Data Length
Data Format
Data Display Format
${data.format}
Data Clearing Policy
Clear if not visible
Limit Input to Visible Area [PDF]
Type


Data
Data properties.
Maximum Data Length
Data Format
Data Display Format
${data.format}
Linked Button Group
../buttonGroup
Browse
Clear
Form
Bound Value
$IF{$FIELD{RBTYPE,group},${name},true}
Type


Data
Data properties.
Initial Field Value
Data Format [PDF]
Display Format [PDF]
num{$zzz,zzz,zzz,zz9.99}
Data Clearing Policy
Clear if not visible
Limit Input to Visible Area [PDF]
Type


Data
Data properties.
Initial Field Value
false
Maximum Data Length
Data Format
Data Display Format
${data.format}
Data Clearing Policy
Clear if not visible
Values
true
false
Add...
Edit...
Bulk Edit...
Remove
Move Up
Move Down
Type

Some brief notes on the panel's fields (Advanced Mode):

Initial Field Value

Use this as an *ad hoc* way of repopulating the field. If you need to repopulate a number of fields from a database lookup or as a general configuration of the form, see hereforanexampleofhowthis is done. **Maximum Data Length**

This prevents the user from entering more characters into the field. If you want to do more than merely having the field stop working once the character limit is reached — such as warnings, in-line messages and other helpful cues to asssist users to get to submission — please see Validation.

Data Format

This is the **raw data** sent from the form and as stored in Transaction Manager. This property is often specified for PDF (in XFA data description specification).

The "${data.format}" is explained in ScriptingandDependencies.

Data Display Format

This can be different from the raw format, especially so with dates. See DataTypes.

Data Clearing Policy

An important factor in Privacy.

Values

Appears with fields that have several states, such as Boolean fields (see Field Types). These states can be named and given an order of precedence. The "Blulk Edit" button enables you to import a list of state names from a text file.

Dropdown Data Panel

You use this panel to edit a dropdown menu field's values and displayed values. The displayed values appear in the menu on the form. For example, the form could, say, display "Northern Territories" in the dropdown; but when users select that value, the form passes the value "NT" to the TM server. The "${values}" entry means that all the values equal the displayed values. See ScriptingandDependencies.



Dropdown Data Panel (Advanced Mode)

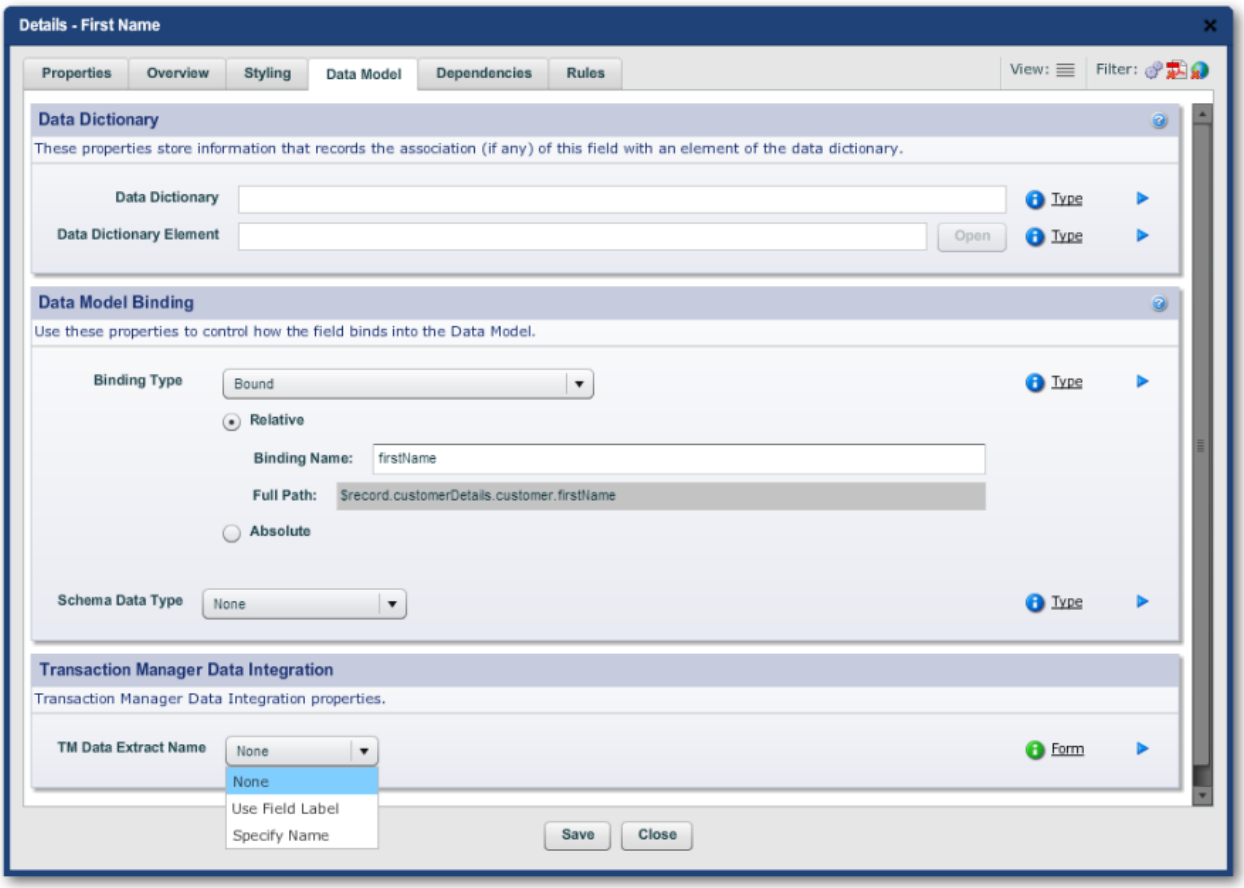
Decimal Options Panel

This panel appears for writable numeric fields and are self-explanatory. In the example below, the field is a currency field and so the positive and negative prefixes have a dollar sign.

The two fields for leading and trailing digits have two properties each for HTML forms and unspecified form types. The leading and trailing values should be the same for both form types.



The Data Model Tab



Scripting

Overview of Scripting

A major advantage of using the Avoka Transact environment — rather than writing your forms from scratch in HTML, Javascript and other scripting languages and writing the server-side process — is that the bulk of the form's implementation is taken care of for you.

But this means that you still have enormous scope to specify how the form behaves and to inject scripts into the form. In this topic, we will discuss the general principals for scripting.

These are the ways to script in Composer:

- Where available there is a **Rule Editor**

This performs simple logic operations. This editor is rather limited.

- A point-and-click **Script Editor** window.

This is for Javascript scripts. The editor is far more capable, but requires a knowledge of Javascript.

- Writing a script yourself in a text editor

The covered in the advanced topic Scripting. These scripts can be complex.

The Composer Scripting Language

In either case, the language used is JavaScript, but with slightly modified function calls, documentedhere.

The Rules Tab

Scripts and Rules go in the Rules Tab. This has LH Menu items and depending, on the nature of the field, can have from none to 7. Here is an example of a Rules tab with a full LH Menu:



Within each panel there are several types of properties:
"Rule" properties "Policies" "Messages" "Actions"
and some checkboxes in, for example, "Mandatory" rules
and "Responsive Rulesets" which are exclusive to the Responsive Rules.

"Rule" Properties

This type of property has a dropdown menu:
Script Based (always available in all "Rules" properties) Rule Based (available in only some "Rules" properties
Always (available in "Editable Rules" and "Mandatory Rules") Never (available in "Editable Rules" and "Mandatory Rules") No Validation (only for "Validation Rules")
Regular Expression Pattern (only for "Validation Rules") for pasting in a regular expression for validation Fixed Length (only for "Validation Rules")
The "...based" items bring up an "Edit..." button, which takes you to the ruleseditor or script editor..

"Policies"

These are always dropdown menus and the choices are restricted to only those that appear in the menus.
The types of policy are:

Editable Policy

i.e. whether the field (or fields) and its children are writable or read-only
Determines what happens when the field is not editable according to the rule in the Edibility panel. Options:

- Disable and gray out text
  - Disable
  - Gray out

Editable Clearing Policy

for more, see Privacy.

Visibility Policy

- When hidden exclude from layout

(i.e. the hidden fields disappear entirely)

- When hidden don't exclude from layout (i.e. leave space for the hidden fields)
- When hidden gray out **Initial Visibility(Policy){}*** seebelow.
- Visible
- Invisible

(i.e. the invisible fields are not excluded from the layout)

- Hidden

(i.e. the invisible fields do not appear and their allocated space in the form does not appear as a blank area)

Validation Policy

(see Validation as there is rather more to this than just the following 2 options)

- Show error dialog
- Not show error dialog

Note: The intent Initial Visibility settings is meant to speed up the initial rendering of the form. If the form is taking a long time to render, visibility rules and visibility policies may be the issues. You can sidestep these by setting the Initial Visibility policy for the relevant fields so that the form will render first up without having to test these rules and policies.

Messages

These are text fields, whose contents are displayed when the rule type — usually "Mandatory" or "Validation" — fails (though you also have the option of displaying the message if the rule passes instead).

Actions

These are exclusive to "Click" LH Menu item. The Click panel only has the "Edit..." button for providing a script. The Rules editor is not available. The script will run only when the field (either a button or a link) is clicked on.

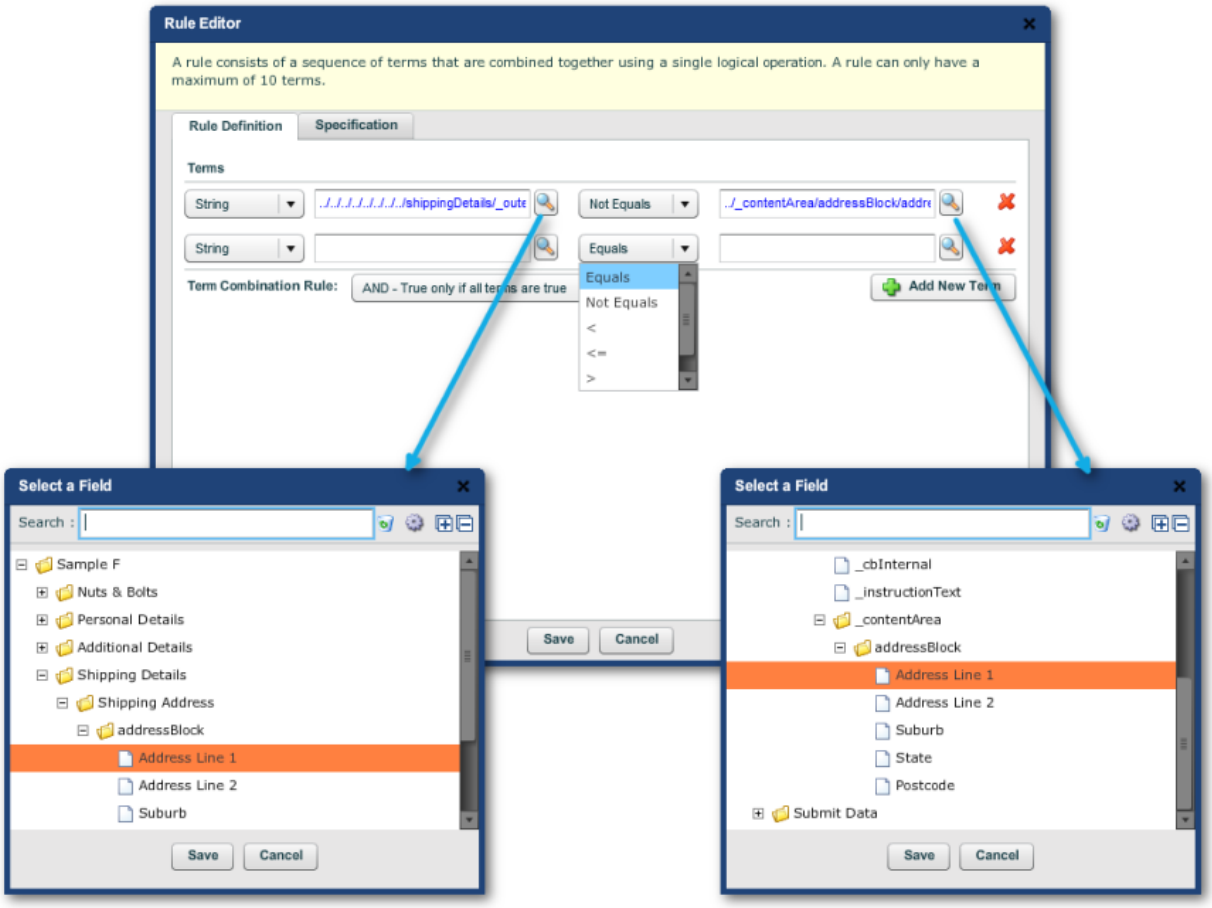
Checkboxes

The checkboxes in the Rules tab panels are for simple either-or options, such as whether the mandatory asterisk will display for mandatory fields.

Responsive Rulesets

These apply only to Responsive Layout settings for the field. See ResponsiveLayout for more on this.

The Rule Editor



You build up a rule with the Rule Editor a "term" (or line in the editor) at a time with a maximum of 10 terms. Each term compares 2 fields on the form, which you select using the two magnifying icons, each of which brings up a "Select a field" dialog containing a mini Structure Panel, as shown.

On the left-most column of each term, there is a dropdown in which you select whether the two fields in the term are strings, numerical or Boolean. Between the two fields is the relationship between the two: "Equals", "Not Equals", "<", "<=", ">" or ">=".

Add more terms as required and then select the relationship between all the terms. You can only nominate one relationship for all the terms. They are:

"AND", meaning that the rule is true only if all the terms are true "OR" where the rule is true only if all the terms are false.

The **Specification tab** is a text area where, if you

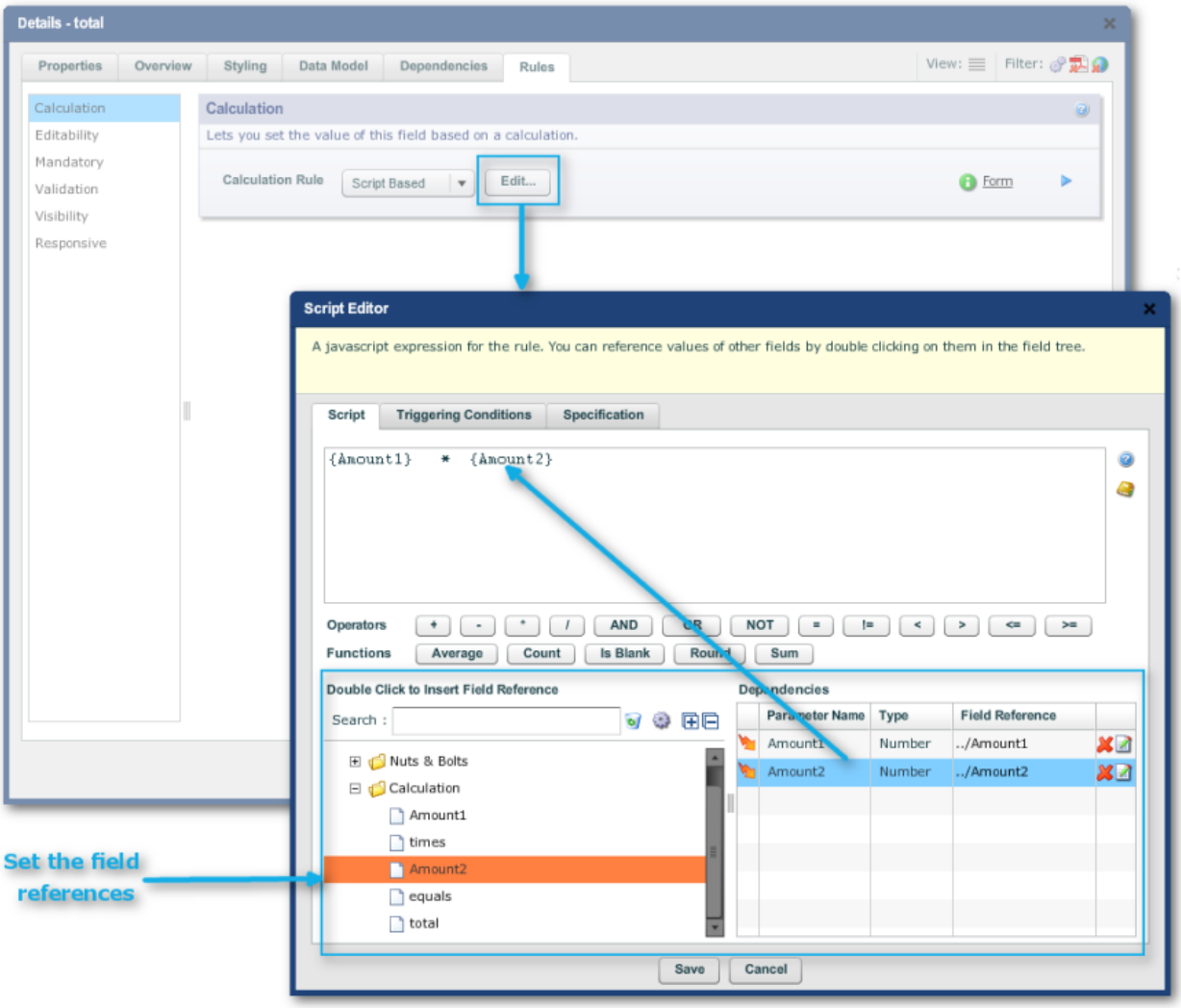
If you know something of Javascript, you can do far more with the Script Editor.

The Script Editor

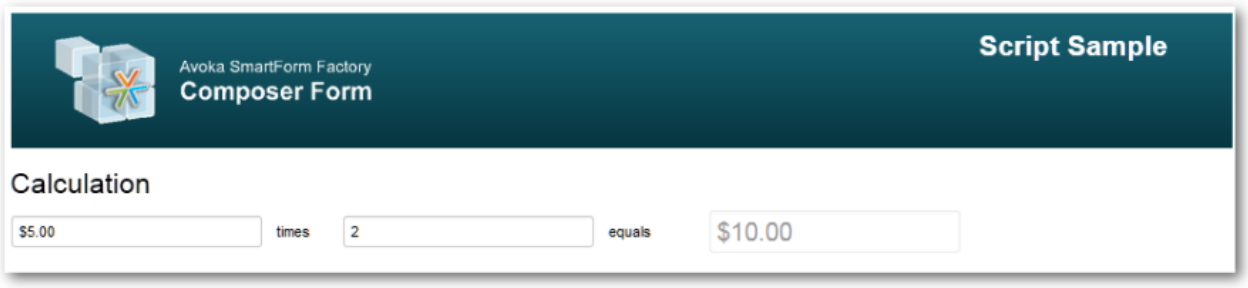
Using this editor requires some knowledge of JavaScript. In this section of the guide, we will only be using very simple sample scripts. The script editor facilitates integration of your JavaScript code with the fields on the form.
The script editor can accomplish much more than the Rule Editor;
Calculations such as summing the values of fields together, or multiplications, averages and so on Write more complex scripts involving "if" statements, loops and so on
In fact, virtually anything the JavaScript is capable of.

Just to be clear, though the scripting language of Composer is JavaScript with some minor differences. One of these allows Composer's JavaScript to use the fields in the form as variables. This makes Composer's scripting capabilities very powerful indeed.
More ground on this will be covered in Scripting in the Advanced Topics. For now, we will be content to just demonstrate how you use the editor to integrate a script into the form.
To make the editor work, you need some working knowledge of JavaScript. We have already given an example of a total in the footer of a table. That example is a sum of repeating elements. The following is a simple multiplication of two non-repeating fields, just to illustrate how scripting works in Composer on a simple level.
Here is a very simple form where we take the first writable field "Amount1" (a dollar value currency field) and multiply it by the second writeable field "Amount2" and display it in "total" a read-only currency field to the right.
We made a few tweaks to "Amount2" and "total" and the "times" and "equals" texts to get the layout as shown below in Edit Properties for each of these fields:
For all, "Properties -> Layout -> Layout Constraints -> Keep on same line (checked)"
For "total", we made it read-only by "Rules -> Editability -> Editablity Rule -> Never Editable"
We could have saved ourselves the trouble of changing each widget by creating a containing block of the 3 widgets and the plain text objects "times" and "equals" and then setting the Layout Constraints.





So, to see the calculation in action, we need to preview the form. Here is what it looks like after inputting a couple of arbitrary values into Amount1 and Amount 2



The JavaScript in the example is only one line, which is the reason the line doesn't end in a semi-colon (";").

The Specification Tab

If want someone else on your team to do the scripting or work the Rules Editor, use the Specification tab to describe the behavior you require. That description will then attach to the field or block.

Dependencies

Overview

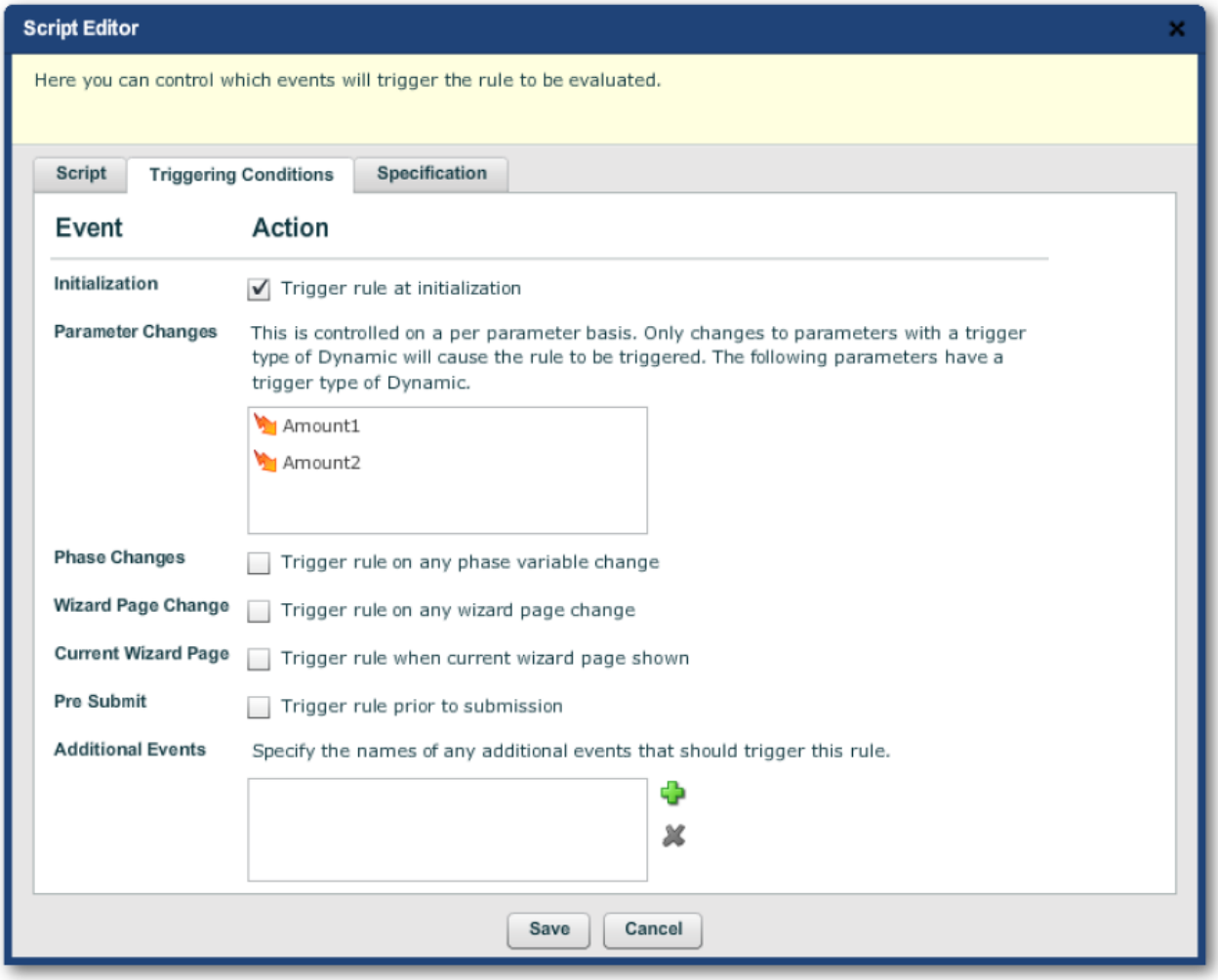
When a user loads a form, fills in its various sections, uploads any attachments and eventually submits it, the form has triggered many events, which Composer groups into "phases", covered in more detail in Business Rules and AdvancedScripting.

Some of these triggers are the dependencies between fields created by scripting. In the above simple example, the "total" field listens to "Amount1" and "Amount2" and when "total" sees that the other 2 now have values, it will calculate and display the value of the calculation. The user does not have to click on a special button to get the total, the form does not have to be reloaded, nor does the user's browser (in this case) have to send anything to the server in this phase. The calculation happens autonomously.

When a form is complex, though, having very many dependencies can create performance issues. Composer has many strategies to tune such forms. One of them is to tune the triggering in the form so that it does not grind to a halt every time the user changes a value in any of the fields. In Composer 4, for the first time, you can inspect triggers and dependencies on the form.

When you paste script into the text area of the Script Editor, remember that you still have to associate the fields on the form, using the same point-and-click method shown above and setting the field references.

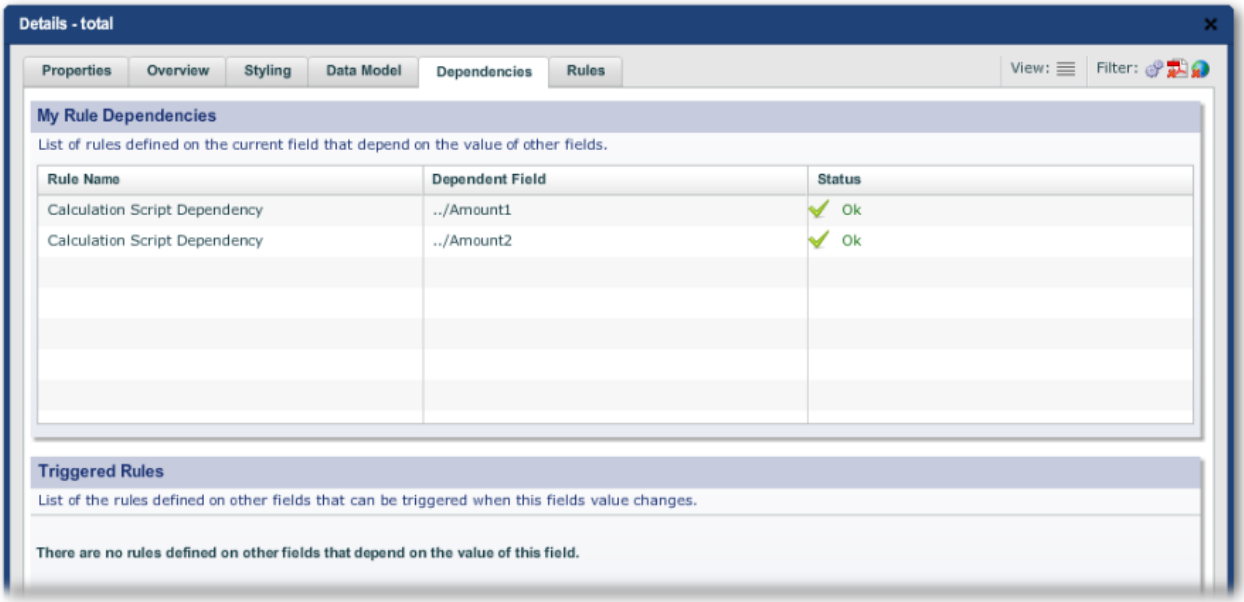
Inspecting Triggers



The triggering conditions of the above simple demo script are listed in the "Triggering Conditions" tab of the Script Editor. We will discuss the advanced settings of the radio buttons in the lower half of the tab later in AdvancedScripting.

Additional events will be covered in the BusinessRules widget discussion.

Inspecting Dependencies



Use "Edit Properties -> Dependencies" to inspect the **dependencies** feeding into the script (in the "My Rule Dependencies: panel) and the **outgoingtriggers** in the "Triggered Rules" panel.

Preview & Publishing

Composer | Form Builder | v4.3 & Higher

Already we have seen, with the simple examples above, that the wireframe's representation of the form does not image all the features of the form as it will appear to the end user, for example here,

So, while you are designing the form, you can do a preflight of it using Composer's Preview function.

And as with the appearance of the form, the Preview function does not fully reflect the behavior of the form in the real world with regards to attachments or in the Mobile App or in other parts of the form's workflow. To see these, you publish the form to Avoka Transaction Manager (or "TM").

Renditions

In Composer you define a form — a data collection experience once, if you will — and have it rendered to look and function differently on different devices and environments.

Composer currently supports two core types of form technology (sometimes called "flavors"):

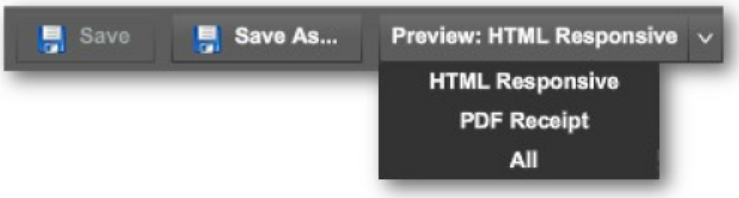
HTML forms using the HTML5 and CSS standards PDF forms using the Adobe XFA standard.

Thus the form often can be rendered in HTML5 and PDF. These days, the general practice is to have users fill out the form in HTML5 and, after submission, get a PDF rendering of the form in the email receipt. The Preview function in Composer has several rendition modes:

HTML Responsive. which uses HTML5 to render how the form will appear in the web browsers of mobile and desktop system.

PDF Receipt, uses the XFA standard and is non-interactive.

The Preview dropdown menu is at the top left of the FormDesigner.



(Some older templates give five choices, but now in Composer 4 all the HTML choices have responsive behavior. This is configurable on the form object in the structure panel, "Edit Properties -> Policies -> HTML Generation -> Responsive -> Form Generation (HTML)" dropdown.)

To preview in HTML5, just select **HTMLResponsive**. In modernbrowsers, the preview will render in a new tab. If the form does not render at all, you may have to change the your browser's opup behavior. For example, in Chrome, you can make the Composer hostname URL an exception to popups.

If you need to resize the preview to simulate mobile devices, just pull the tab out from the tab bar and, if necessary, take the resulting new window out of the full size mode. (Do this with the middle button on the top right of the frame in Windows; or with the green candy button on the top left of the Window in Mac OS X.) See the ResponsiveLayout section of this guide for more on mobile device simulation.

For PDF preview, you must set the browser's preferences to use Adobe Acrobat or its plugin. The reason for this is: all modern browsers have their own PDF renderers and these do not work with XFA.

Publishing to TM

There are good reasons to publish your form to TM while the form is still in pre-production:

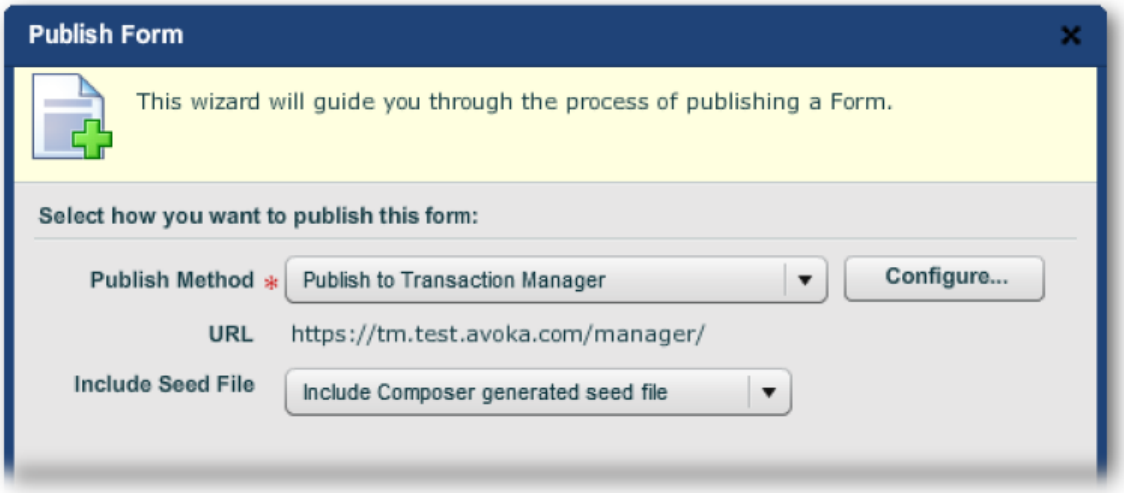
The behavior of the form in preview does not accommodate attachments Preview does not show the look of the form in the context of the portal Preview only simulates how the form will look on a mobile device

And the same applies for how the form will look in the MobileApp

You cannot test the behavior of Camera widgets and other use of the sensors of mobile devices

GPS locating is not available in Preview. GPS is available only in those mobile devices that support it. Without GPS, geolocation functions work only from IP address mapping, which is approximate only.

Publishing to TM is simple, and the wizard is accessed through the "Publish" button in the FormDesigner or at the Forms Level of theUI.



You have the choice of publishing the form directly to TM (via the URL shown below) or of generating a zip file and importing that manually to TM. Publishing directly is the more convenient.

The layout of a tablet form may also change based on the size of the tablet, and whether it's in landscape or portrait mode. This is not controlled directly by the skin, but is controlled within the form itself. Please see ResponsiveLayout.

Please consult the Transaction Manager Administration Guide on how to then publish the form to a web portal (or have a colleague with the required access do it for you). You then will be able to access the test form:

through a nominated portal, or

via the Mobile App, configured to point to a nominated mobile portal (see the Transact Mobile App Guide on how to point your instance of the Mobile App to that portal), or

through a URL created by TM in an approved mobile device web browser (see DeliverytoMobileDevices).

Navigation

Composer | Form Builder | v4.3 & Higher

The navigation methods to guide users through a form are of the utmost importance and deserve much thought when designing new business processes and the web forms that embody these.

In an ideal world, navigation through a form should be as convenient, intuitive and frictionless as possible. Achieving this is goal will always be a balance between the needs of the business and trying to provide the user as enjoyable an experience as possible.

So here, "Navigation" means more in interactive forms than providing "Previous" and "Next" buttons at the bottom of the page. It means more than providing menus and breadcrumbs. It means more than restricting the user's access to the form's pages to a fixed sequence. It means more than allowing the user to roam through the form at random. It means more than tailoring the form's layout to the range of devices that will be used to access it. It means more than providing menus, be they down the left hand side, along the top or, in a touch environment, brought up by swiping to the side.

Whatever the business function of a form, all forms ultimately must aim at having the user complete them through to submission. If the user abandons the form, that means a lost opportunity. And we can now see how many users access the form and how many of them advance to the submission stage and how many abandon the form (see Analytics).

The Chrome of Navigation

" Chrome is the visual design elements that give users information about or commands to operate on the screen's content (as opposed to being part of that content). These design elements are provided by the underlying system — whether it be an operating system, a website, or an application — and surround the user's data."
— JakobNielsen

So, chrome informs users with clues about how to use the form. It is also not part of the form's data — the point of creating the form in the first place — and takes up valuable screen real estate. The smaller the screen, the greater the cost of chrome in terms of space.

But having only minimal chrome, adds friction to the form in that the user will have fewer options to do such useful actions as going back several sections to correct some entry.

The Simplest Chrome: None

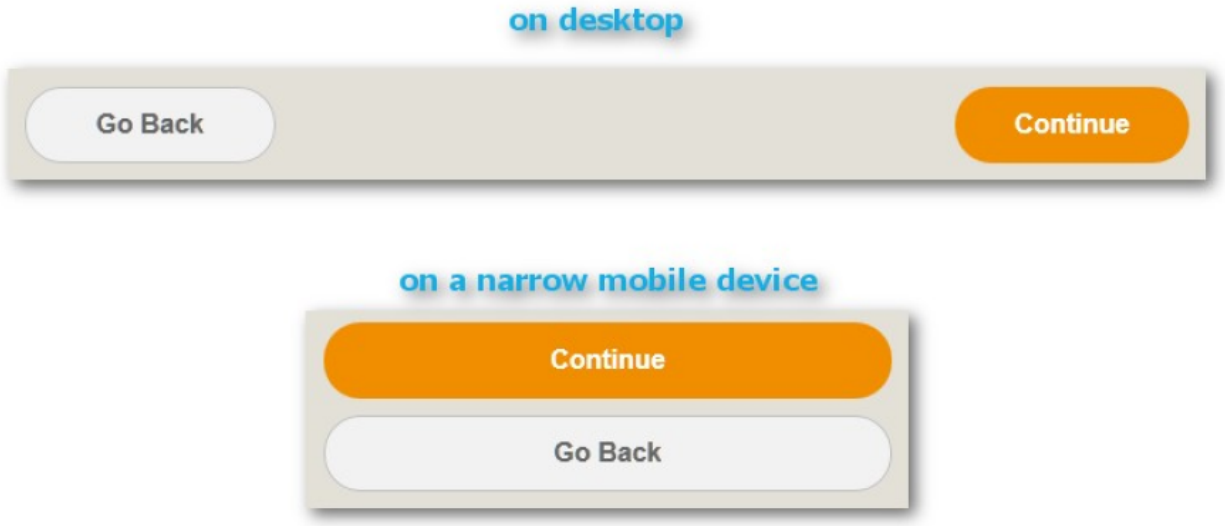
At least by this we mean that the form does not introduce its own chrome. However, the usual chrome that comes with the operating system and browser combination is present, and this chrome will not be perfect, even in such a mature use case as web browsing.

So, this means the designer is limited a single-page form, and the only way to navigate the form is to scroll the form in the browser's window. While that does not sound so bad at first, it is very limiting and, where there is a lot of information to be entered onto the form, leads to vertical clutter. It also leads to users getting confused if they have to make revisions. In order to find where to make the revision, users have to remember how far up or down they have to scroll to find the fields subject for change, or — even worse, use the browser's "Find" function. Thus, even with this, the simplest navigation scheme, there is friction, especially if the form is long and detailed.

And the designer cannot control how users move through the form. Some form elements, Sections for example, can collapse and expand. Other elements on the form can switch from being invisible to visible. But users can still skip to the end of the form, in anticipation of the "Submit" button being there, only to be disappointed when that button does not work because they have failed to fill in all the mandatory fields, or have entered invalid information somewhere up the form. See Validation for more.

Button Navigation

This is slightly more helpful. But users can only step forward and back. This simple chrome uses up the bottom of the page (and may also be duplicated at the top, though many designers resist doing this). This does not amount to much space on the desktop, but in mobile — and especially on narrow smartphones in portrait orientation — that space becomes a large part of the available viewing area. And even more so, when the form is responsive and the two buttons spread out to accommodate the target areas needed for the touch interfaces.



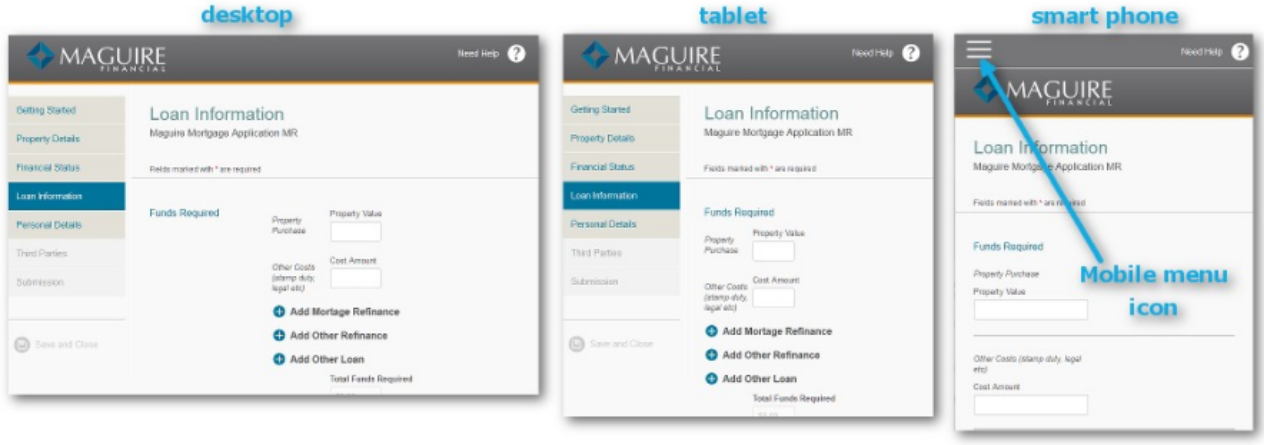
Such a simple concept, though, can suddenly become fraught. The above example has a simple color scheme: orange to move forward and white to go back. But what if users move back and then forget where they have been? Do we indicate whether users have viewed a page through a wider selection of colors? Probably not, because more than two colors have no intrinsic meaning. Users cannot experiment to discover the meanings assigned to colors (such as "already viewed", "yet to be viewed", "not allowed to view", "the last page" and "the first page") except by moving through the pages with the two buttons.

On the desktop, where the viewing area is the largest of any viewing environment. the vertical space occupied by the two buttons is small. But on a narrow mobile device (such as a smartphone in portrait mode), the responsive layout function of the form spreads the buttons out so that their tap area is large enough to be useful and stacks the 2 on top of each other. Both are done to minimize user frustration — a vital consideration in mobile.

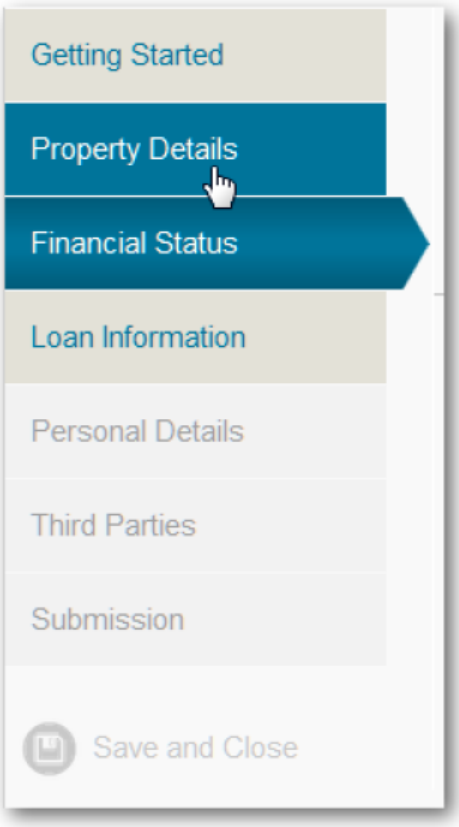
Note: in the following examples, all mobile screen shots are either from Preview or in the mobiledevice's browser, not in the TransactFieldmobileapp.

Left Hand Menus

These are a tried-and-true web design element.



The left hand menu shown above has 3 states (indicated by 3 combinations of text color and background color):
Current item, here "Loan Information" Other items, access allowed
Other items, access not yet allowed
The color coding scheme becomes obvious to users through their interactions with the menu. It does not need to be spelled out.
In terms of navigation and user experience, the left hand menu is a vast improvement on 2 button navigation:
Users get clues as to how much more work they have to do before they get to their goal: being able to submit the form.
They can see what is in store, from the labels of the menu items. They get to see where they have been.
If allowed (see NavigationChoices below), they can skip several pages.
They can move several pages back in the form, without getting confused or forgetting where they were. So what are the down sides, given the menu's superiority over the 2 button scheme?
The space below the left hand menu is dead space. There are a few reasons why this dead space exists, mainly due to past limitations of HTML, and such reasons do not matter much any more. But the convention still exists and some usability gurus advocate it. The dead space is hardly an issue on the desktop. In fact, in the above example, the desktop has another level of deadspace (under the subheading "Funds Required"). The rendering on tablet-sized screens eliminates this second level of whitespace.
The menu buttons on mobile have to be large enough to prevent user frustration. But this makes the left hand menu unsuitable for very narrow devices such as smartphones in portrait orientation. In the above example, the left hand menu disappears entirely, replaced by a typical mobile piece of chrome: the mobile menu icon, familiarly called "the hamburger icon".
Another variation on the left hand menu chrome:



This example has another state added to the 3 already encountered in the previous example: the hover state (as seen in "Property Details" where the user's mouse cursor is hovering over the menu item). Here, "hovering" means that users move their mouse cursor over an object but do not click on their mouse's button to select the object.

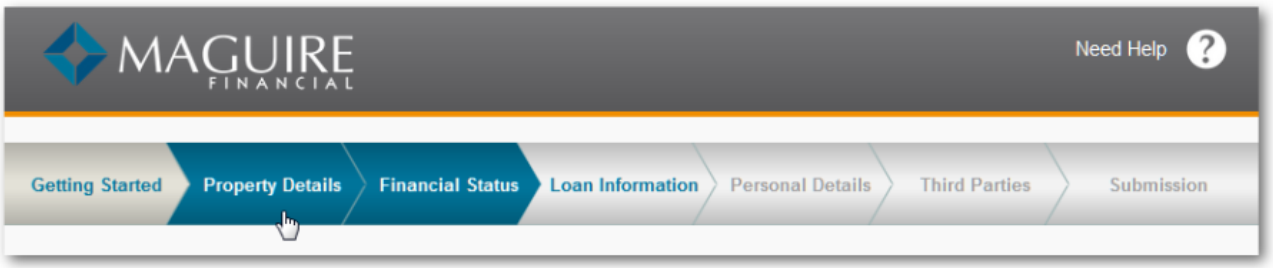
LH Menu Variants (Unrecommended)

The left hand menu can have variants, but most of these have, in the light of experience, led down blind alleys.

Right-justified captions. Notrecommended.

Fly out sub-menus. These have died a quiet death on the web as they demand too much manual dexterity from users.

Top Menus



Top menus are now fashionable and with good reason. They work well on the desktop and with tablets and other wide mobile devices. They also eliminate the dead space of Left Hand menus. This example shows a 5- state top menu: note the differences backgrounds between "Getting Started" (i.e. already visited) and "Loan Information" (available but not yet visited). "Property Details" shows the hover state.

Hover does not exist in a mobile and touch environment. For mobile, hovering is no longer an option.

However, they do not work for narrow mobile devices. And you should keep the captions brief, as the captions begin to wrap for narrower screens. The solution is the same as the above example: the hamburger icon.

2-Level Menus



Avoka Technologies has given much thought to providing second-level menus to their forms. But we have rejected a large number of the existing solutions,

MegaMenus, i.e. big 2-dimensional dropdown menus. These worked well on the desktop, but do not work in mobile due to their dependency on hovering (see TopMenus above).

Flyouts. Again, these are hover-dependent and therefore not an option for mobile.

The example above shows our current practise: "Shipping & Billing" has two subdivisions, shown in the chevrons below the main, topmost menu. Our tests show that these work well, both on the desktop and in mobile.

Mobile Menu Chrome

This is the mobile solution to all the above navigation chrome, aside from the 2-button convention. The Mobile Menu Button is also known among developers as "the hamburger" icon.



Here on an iPhone 5s, the top menu appears in landscape; the Mobile Menu Button in portrait. Tapping on the Mobile Menu Icon causes the menu to slide out from the left. The menu has only 4 states because it lacks hover.

Navigation Choices

Chrome is only part of the story. There are several more degrees of freedom to form navigation.

Unconstrained vs Constrained

Unconstrained here means that users can select any item on the menu at their merest whim. This certainly removes friction but there is a possible down side: the lack of discipline imposed upon users could be a factor in their abandoning the form.

Constrained here means that users can only make a restricted set of menu choices at any one time in filling-in process. To encourage users to understand what is happening and where they stand in their data entry, the subtle coloring of the menu items is tacitly employed — without any tedious explanation. Users who "get it"
will be happy; users who don't will also be happy in that they have not been burdened with the task of ploughing through some ma-rigmarole that is of absolutely no interest to them.

Visible vs Hidden Navigation Choices

Hiding parts of the form is best done to convenience users, not to conceal aspects of the form because they are "unworthy" for whatever arbitrary reason.

So, you may have already noted that the headers of Sections in Composer forms have much that is hidden: the help text, for example. These hidden passages of the form are there to help users who may be struggling and need some assistance. Confident users do not want to see these, as wading through long, unneeded hand- holding text is off-putting and adds for them friction. But not having help for others will add friction for **them**.

Another use case of hiding parts of the form as with accordion wizards, where the sections of the form can expand or collapse (instead of being rendered as new pages). However, do bare in mind that users may not fill in some sections due to inattention — for example not realizing that they should expand some section.

Applying validation will catch this mistake, but you as a designer forfeit the trust of users for letting them down and leading their navigation through the form astray.

Hiding some parts of the form as a result of users choosing some option or another could also be something that eliminates friction for some users and therefore be a good thing.

Validation, Errors and Navigation

There are good reasons for having validation and not allowing users to submit until one or more criteria are met. There are also good reasons why users get fed up with it. Remember: the more users successfully submit a form, the better. In that respect, the web is a popularity contest. The tension between creating friction or gaining the trust of users is not a paradox (a problem to be solved); it is a type of dilemma, where there several options, all correct in themselves, but where you as a designer have to decide what solution to adopt as a judgment call.

Error blocks and validation messages are good to have, but remember the friction they create and that they will win you no popularity.

Abandonment

Abandoned forms are a fact of life in online interactions. We have done our best to give you the tools to achieve reasonable user navigation and hopefully to reduce the percentage of abandoned forms.

To that end, we have introduced to Composer 4 templates capable of detecting where abandonment occurs in the form filling process. This data is then accessed through TM. Note though that there are privacy implications.

Portal Chrome

Avoka Technologies strongly recommends that forms do not appear embedded in the portal page of your enterprise. One common problem is that the portal itself is not responsive and thus the form contained within it will not display properly on a mobile device. Instead, we recommend that:

the form take up the whole browser window (similar in effect to the a href tag's parameter target="_parent" in html)

is not be rendered in a separate window or tab (as seen, for example, with target="_blank")

so, as a corollary, there should not be breadcrumbs on the form, especially those that lead back to the portal's home landi

Sections

Introduction

In previous versions of Composer, navigation was implemented in forms through the wizard settings of the form and the section fields in the Structure Panel. Sections could be organized into three levels. Each section had an elaborate header which included collapsible Help, and various other user aids; sections also had footers. The content was between the two.

So, using Sections to divide the form brings the following advantages:

Sections have chrome associated with them, such as headers, footers, collapsible help and inclusion in the navigation menus.

So, all this associated chrome will be consistent throughout the form. Changes to the chrome can also be (if desired) applied globally

Constrained and Unconstrained menu navigation is controlled through Sections

Section styling is professionally designed, including margins, fonts, and colors, to provide an attractive form Section styling is controlled via stylesheets. Change the stylesheet to subtly or radically change the look of your form.

In Composer 4, the navigation menus (shown above) are more powerful than in older version. To accomodate the new menu functions, there is a break from the older way of setting section levels (as embodied in the Section Assistant shown below) and the new way. Also, now there are many more "Edit Properties" settings for sections that control these new menus. The older section levels and the older style of menus are still present in the product to retain compatibility with legacy designs.

So, we will first discuss the older navigation functions through Wizard settings. Then, we will move on to the new ways of configuring navigation.

The headersandfooters of sections contain many important elements of sections, such as the numbering of sections, the help and more.

Maguire Template

Some of the section navigation behaviors are embodied in the template. This advanced topic is covered in Templates below.

The mobile menu is a feature of MaGuire (see Templates for an example of this template).

Adding a Section

The following applies equally to all templates, including Maguire.

A Section is really a specialized type of block, which uses features of blocks, including having children and hiding sub-blocks within it.

A convenient way of creating sections is to use the Section Assistant and its wizard.

The Section Assistant wizard.

Section Levels

In the older versions of Composer, there are three levels of sections provided in the Palette: Levels 1, 2 and 3. This scheme is retained in Composer 4 for compatibility, but we will discuss below the newer way of organising Sections.

The style of each level can be controlled independently.

These older style of Sections Levels must always be correctly nested within each other — Level 1 at the top, with Level 2 indented one level below, and Level 3 indented inside Level 2. If you nest your sections incorrectly, then your form may look strange. If this happens, simply nest the sections correctly. There is no requirement to use Sections or all the available section levels, although most forms use at least one Section Level 1.

In previous versions of Composer, Level 1 sections are the main wizard pages. This has changed with Composer 4.

Maguire's Use of Sections

Instead of using Section Levels, Composer 4 now uses Section Groups to organize the menu items into a 2- level navigation scheme.

Maguire Template use of Level 2 Sections

We recommend that each Level 1 section in the Maguire template have **at least 1** Level 2 section. This is **not** for inclusion in the navigation menu; the Maguire template makes use of Level 2 sections for optimum layout. The content of each Level 1 section should be distributed among several Level 2 sections. T

Wizards

In older Composer versions, the whole form was set to be a Wizard, a Wizard with a menu, or not a Wizard at all,

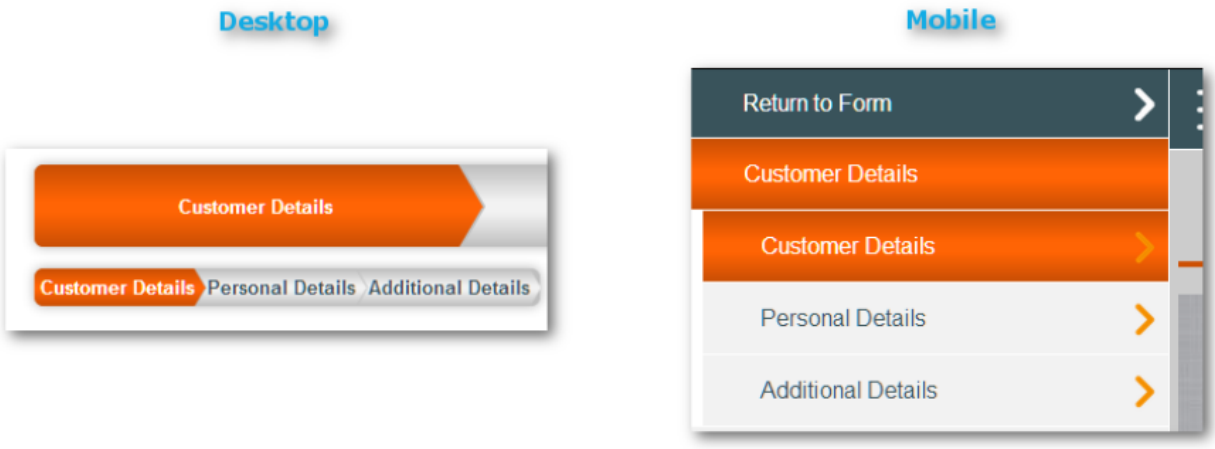
Now, individual sections can set to act as Wizard pages in the Wizard Panel. And, independently, a section can also be set to become a menu item or not.

"Edit Properties -> Properties -> Wizard -> Wizard -> Act as wizard page (checkbox)"

"Edit Properties -> Properties -> Wizard -> Wizard -> Create Wizard Menu Entry (checkbox)" "Properties -> Wizard -> Wizard -> Add to Navigation Menu (checkbox)"

So, if you opt for having a page not act as a wizard page, it will be appended under the previous page; if you still make it create a wizard entry and add to the navigation menu, the section will still appear in the menu, despite not being a separate page.

2-Level Navigation



MaGuire template top 2-level menu

Now, all Sections in the form menu are Section Level 1 (see Adding a Sectiom), irrespective of whether they are top level items or second level items in the navigation menu. A section's menu level is now determined by the property "Properties -> Wizard -> Wizard -> Section Group Name"

Top Level menuitems now belong to their default group (i.e. "${section.heading}")

Second Level Items now are grouped according to their top level menu item's name. With the menu shown above:

"Customer Details" belongs to the section group "${section.heading}"

"Personal Details" & "Additional Details" both are made to belong to section group "Customer Details".

Also, they are both have the setting "Properties -> Wizard -> Wizard -> Add to Navigation Menu (checkbox)" as checked.

Note: Though 2-level navigation is not exclusive to Maguire, the Chevron format of the top menu (shown immediately above) and the slider ,mobile menu are Maguire features.

The Navigation Menu in Composer

Features common to all templates

Properties to configure the appearance of navigation menu states (discussed in LHMenus above)

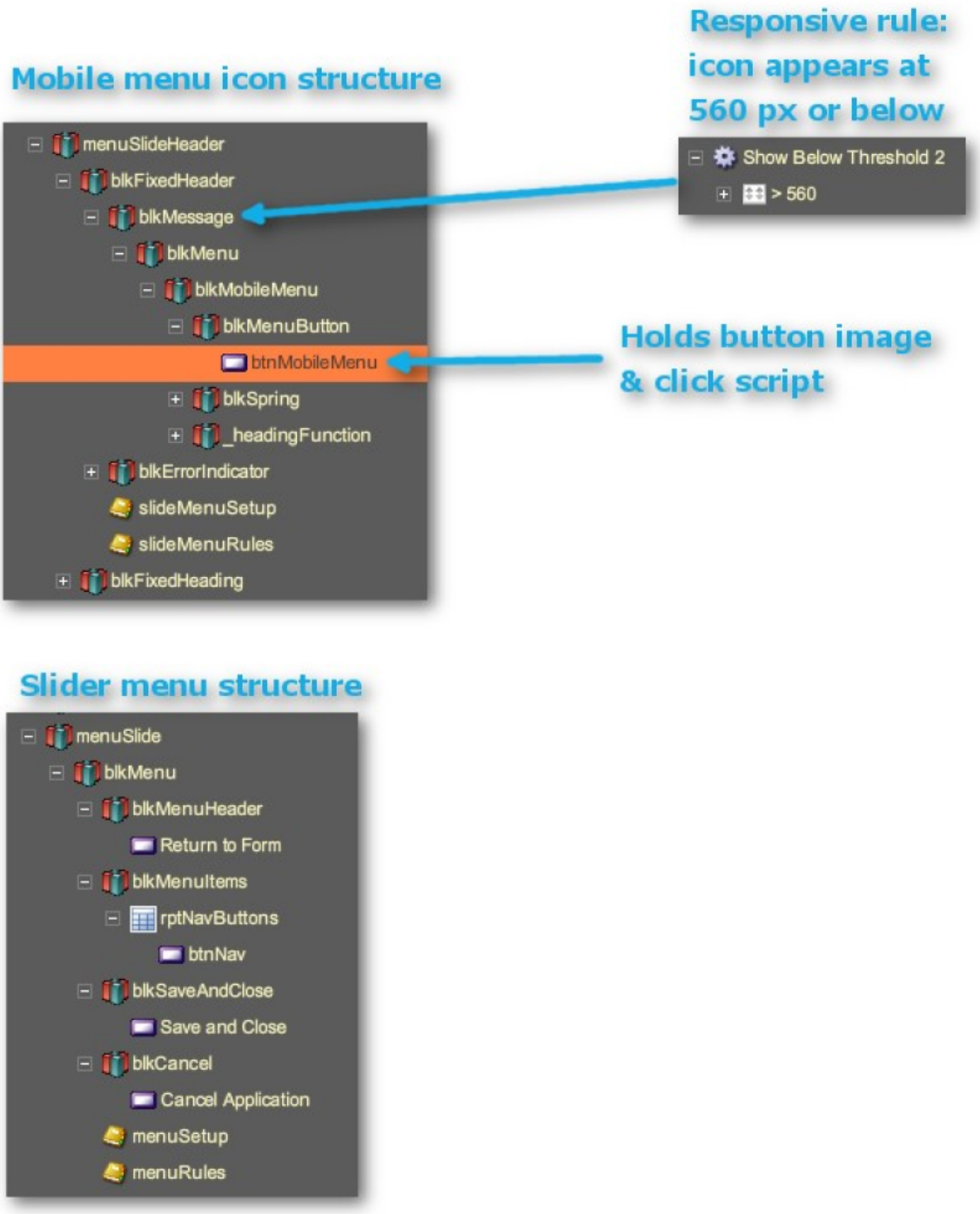
On the form object at the top of the structure panel tree, "Edit Properties -> Theme -> Menu ->" and then these panels:

- "Menu Entry"
- "Menu Entry - Active"
- "Menu Entry - Disabled"
- "Menu Entry - Enabled"
- "Menu Entry - Hover"
- "Menu Entry - Visited"

Maguire Exclusive Features

Mobile menu navigation for narrow mobile devices

Illustration of the structure behind themobilemenuseeninNavigation.



The Maguire navigation menu does not render in wireframe

To view it, you must preview the form, where you can also see the different responsivelayouts of the menu.

Headers and Footers

The Simple Approach

Composer 4, and the older versions of Composer, use the "Edit Properties" of the section objects in the structure panel to configure the sections. This approach is useful for prototyping and proof and concept. It will even work for mobile devices, despite its limitations.

You use 4 panels to control the section objects with the simple approach:

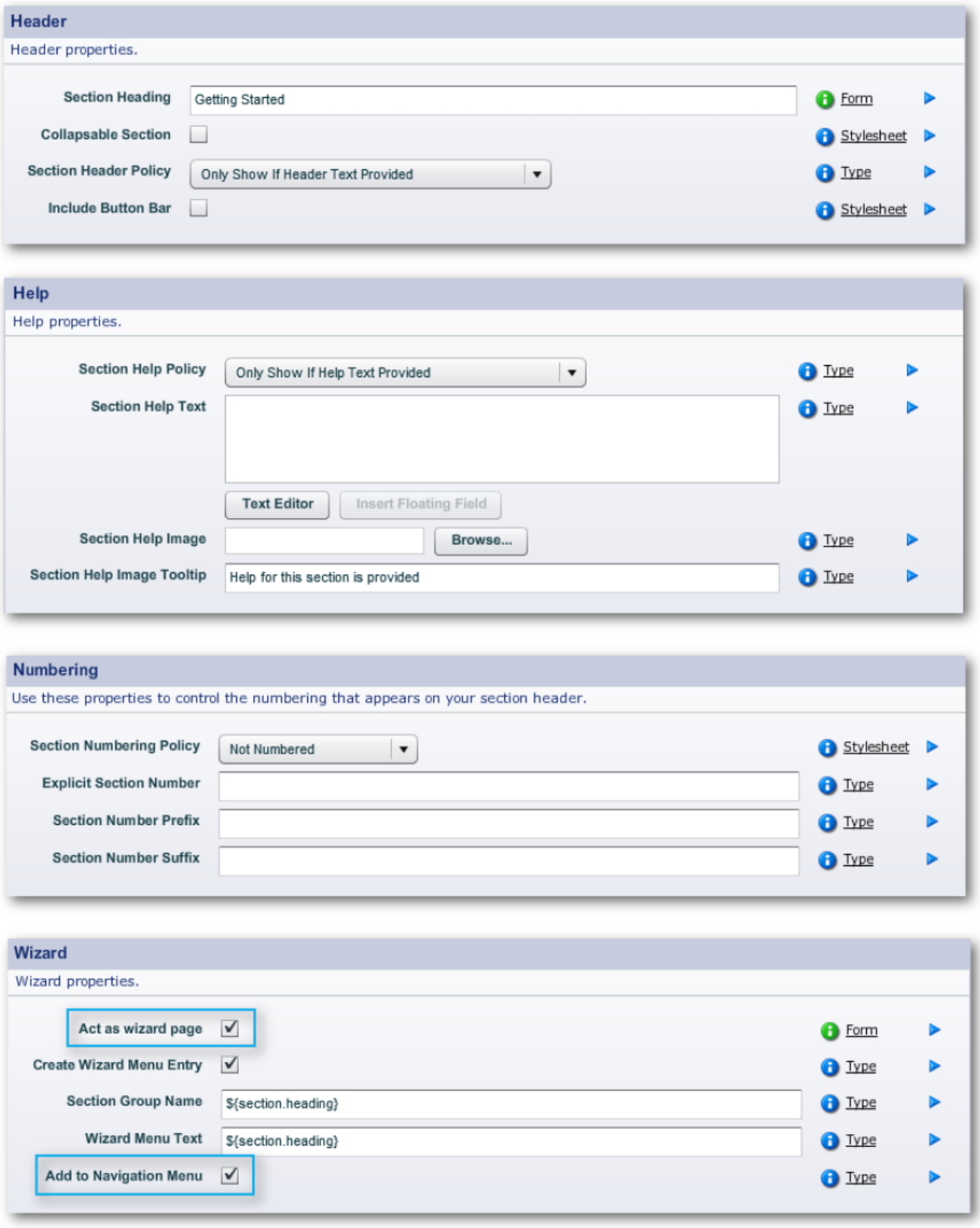
Properties -> Section Header -> Header Properties -> Section Help -> Help

Properties -> Section Numbering -> Numbering

Note: these numbers only appear in form menus controlled by the Wizard panel. They do not appear in Maguire-style navigation menus.

Properties -> Wizard -> Wizard

Note: this panel controls whether the section appears in the (Maguire-style) navigation menu and whether it behaves as a wizard page. In Composer 4, forms can now have some Level 1 sections behaving as wizard pages, while other Level 1 sections may not. In earlier versions of Composer, the whole form either was a wizard or not.



The 2 properties highlighted in the above image of the Wizard Panel support the Maguire-style Navigation. Make sure they are checked when you create a new section in a Maguire-style template.

The Modern Approach

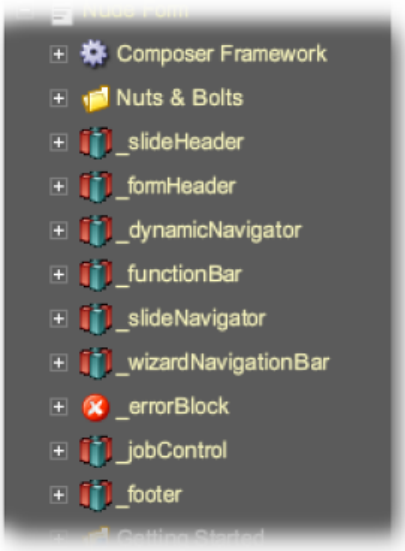
Maguire-style Navigation

The whole approach in Composer 4 and the Maguire style of template is diffewrent from that of the earlier versions of Composer.

The earlier approach was to have the help passages of the form associated with each section of the form. Now, the Maguire style has an overall form header with global help content across the whole form, as part of Composer's responsive layout navigation strategy.



All these features are controlled in the Structure Panel through the following blocks and their contained objects, most of which are visible only in the Advanced Mode of the Structure Panel..



These blocks expand into a large number of contained elements and it is beyond the scope of the this guide to document every aspect of this structure. By all means, explore these blocks to see how the Maguire navigation structure is built up and edited.

Footers

Here, we mention footers only in passing. In the Structure of the form, footer blocks come under the Advanced Mode. There is a global footer block for the form, which is defined in the block's "Edit Properties -> Layout -> Layout Constraints -> Layout Data -> South". Also, each Section has a footer block defined in the Advanced Mode Structure as *Some Section* -> Outer Area -> Footer Area.

In the abovescreengrab, the global footer of the Maguire-style sample is "Transact Footer".

Recolor and Hide on Print

Re-color is a separate facility to the concept of a separate Receipt renditions (see PreviewandPublishing) for printing. In general, we recommend the use of Receipt renditions rather than Re-color on Print.

Most of Avoka's standard Section styles make use of colored backgrounds, with fields having white backgrounds. Studies show that this is best practice in terms of look and usability. (It is also possible to create style-sheets with a more "plain" appearance - please contact Avoka if you would like to explore this option for your organization's forms.)

However; many organizations and individuals still utilize black and white printers, or wish to conserve the amount of ink used for printing forms. Large "swashes" of color do not satisfy this goal, and also do not fax very well. Therefore, most of our sections have the ability to "Re-color on Print", found in "Edit Properties -> Properties -> Printing".

"Re-color on Print" means that when the user clicks the print button in Adobe Reader, the form will "morph" to a more printer-friendly black-on-white look, and then morph back again once printing is complete. This behavior can be controlled via a property on the sections. Note that some widgets (such as Images) will not observe re- color on print, and will print in color if available.

Re-color on print only applies to PDF forms and PDF receipts.

A related property allows particular objects to be hidden when the form is printed, "Edit Properties ->:Properties -> Printing -> Visibility on Print [PDF]". You can explicitly set this property on blocks if you want their contents to be hidden on print.

Layout

Composer | Form Builder | v4.3 & Higher

Just as the concept of form navigation has changed now that users will be interacting with your forms on a variety of devices, where formerly you could have been certain that they would only be using a desktop computer or laptop. Indeed, we now group laptops with desk-bound computers and call them collectively "the desktop". The important distinction is that "the desktop" is now characterized by a large monitor and a pointing device (either a mouse or a trackpad).

Mobile devices have restricted screen space and the user interacts with them through a touch interface. See Devices vs Desktop for more.

As a result, LegacyLayouts now are a thing of the past. What worked in hard copy will probably no longer work on a smartphone.

This early parts of this section of the Composer 4 Guide explores these issues a little. The section ends with ResponsiveLayout.

Also, forms these days will be consumed more in HTML rather than as PDF. User expect this. In fact waiting for a PDF form to load, which often take much longer than the HTML version of a form, is more unwanted friction for the user.

Legacy Layouts

Overview

There are several different factors to consider when laying out a form. We encourage you to think carefully about your goals, rather than trying to lay out a form to match an existing paper form.

Goal	Suggestions
Correct form layout on multiple devices, with differing widths and screen real estate, or switching from portrait to landscape mode.	Keep your layout as simple as possible. Avoid fixed width fields Use vertical layout managers. Use Layout-Fill. Avoid fixed width fields.
Useability, as much as possible, with grouping of related fields together.	Use vertical layout managers, with keep-on-same-line to keep related fields on the same line. For more control, use horizontal layout managers for each line, within an overall vertical layout.
Both PDF and HTML.	In general, design first for HTML, and then preview in PDF and adjust as necessary.

Fitting the form onto a single page (no matter how big the form is).	Composer is not designed to make use of every bit of whitespace. Usability studies show that whitespace in a form greatly improves readablity and usability. Disable all interactive features, including hide/show logic. Expand multi-line text fields, section help, expandable tables and repeats. There is no cost for an electronic forms to have multiple pages. In fact, multiple pages save money by increasing usability. If a form is ultimately going to press, consider having two different form designs: one optimized for interactivity, the other for receipt or print.
Mimicking an existing paper form.	Use horizontal layouts, grid layouts, and fixed width fields as necessary. Use caption left where necessary. Make the desktop HTML a fixed page width. (This is the default for desktop HTML.) Don't try to make your form identical to the paper form. Remember, you want a better user experience for your users, and that means having design changes.

Goal	Suggestions
Replicating an existing paper form as a "pixel perfect" copy.	You cannot create pixel-perfect copies of existing forms with Composer. If you require pixel-perfect forms, you should consider creating a separate receipt form. o **Note:** Transaction Manager can be configured to support a separate form for the receipt and for the interactive form.

Keep it simple

Keep your layout as simple as possible. More complex layouts become extremely challenging. Keep it simple, unless you have very specific requirements.

Let Composer do the work

Composer's Layout Managers do the fields alignment and uniform sizing, based on your organizational template and style-sheets. Better to let Composer do the work, rather than struggling with Composer for the sake of a particular look.

Use Percentages, not Fixed Width layouts

You don't know how big your user's screen will be. Try not to make too many assumptions about widths, and let Composer do the work of layout out your form.

Use Field Styling

You can use Advanced mode in the property editor to explicitly set field widths in pixels, mm, or inches, known as "magic numbers". Sometimes this may be necessary, but this introduces two problems:
Every form designer needs to remember these "magic numbers" to achieve consistency.
If you need to change some aspect of the layout, you need to modify each field independently.
Composer allows you to define standard fixed width field sizes such as Small, Medium and Large in your stylesheets (see FieldStyling). The advantage of these named sizes is that there is nothing to remember, and if you need to change them, you can simply adjust the style-sheet. A good rule of thumb is that these sizes should all be multiples of each other so that combinations of smaller and larger fields all line up.

Checkboxes and Radio Buttons

It can be very useful to use a Grid Layout for groups of checkboxes and radio buttons. Use the same number of grid cells for every set of radio buttons or checkboxes on your form, so that they all line up correctly. Use spans for captions that are very long. We recommend 4 to simplify responsive layout, although this is your preference. Please see the Grid Layout topic on this for details.
Please note that the grid layout is not ideal if you want your form to lay out correctly on narrow screens like smartphones.

Units and usage tips

Sometimes it is necessary to put units or other tips on a form. For example "miles" for a measurement of distance, or "MM/DD/YY" as a tip for the format for a date field. You can use a Plain Text or Rich Text field for this purpose, but often its difficult to get the text to line up correctly with the field itself, particularly with Caption Top fields.
Composer has a special Label Field that always ensures that the label is aligned with the Text Field that can be used for this purpose.

Caption Top or Caption Left

We generally recommend Caption Top, for a number of reasons. This is summarized here: {+}http://+ www.avoka.com/composerblog/2011/10/form-design-tip-caption-top/

Mixing Checkboxes or Radio Buttons with Textfields on a line

If you mix Text Fields with Radio Buttons or Checkboxes on the same line, you often get results where the resulting layout looks bad. This is because Radio Buttons and Checkboxes, by default, have their caption on the same line, while Text Fields (for most Composer stylesheets) have Caption Top.



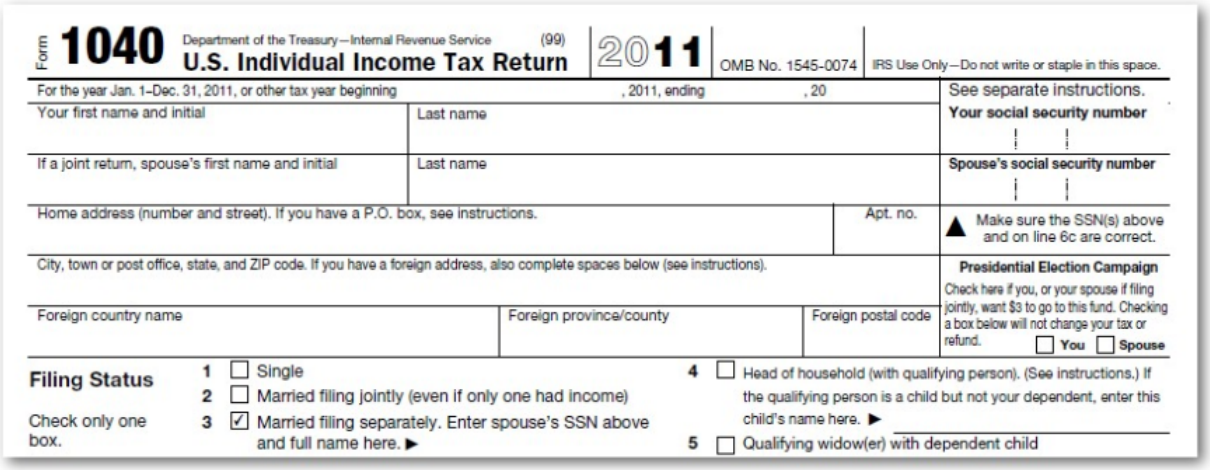
There are a number of simple ways of solving this problem.

Remove the caption by clearing the Label value, or by checking the advanced property Properties -> Caption -> Caption Content -> Never Show Caption. This effectively associates the caption on the Other radio button as the implicit caption for the text field.

Set the "Properties -> Caption -> Caption Presentation -> Caption Placement" on the Text Field to "Left". Place the Other Text Field on the next line.

Boxy Layout

Many paper forms designed for the US Government use a style known as "Boxy". Here is an example of this style:



An example of a Boxy layout

This layout was designed around half a century ago, when it had some advantages for form-designers who created these forms using simple typographic tools, and the printers who printed them. It has serious disadvantages in the modern world, which include:

The layout is often very cramped, and is designed to cram as many fields on the form as possible to reduce printing and handling costs. There is no need for electronic forms to have this restriction, and the value of white space within a form to aid in usability are well documented. Usability studies have shown that this approach creates a large number of problems, including customers having difficulty understanding the form, missing data, and difficulty for those reviewing the form to find the information they need.

Inline help needs to be very concise, and so help is often either terse and difficult to understand, or provided in a separate document. Electronic forms do not suffer from this restriction.

The layout is extremely difficult to maintain. The addition of a single field or changing of even a single word on the form can result in large portions of the form needing to be re-designed and re-positioned.

The fields themselves are difficult to locate on the form, as they don't really stand out from the surrounding text.

Even for handwritten forms, the caption text gets in the way of the area for writing, and often leads to cramped writing which is more difficult to read.

There is no possibility of using hide/show logic or repeating sections in the form, because the entire form is so tightly structured. The 1040 form only allows for four dependents, and has special instructions for applicants with more than four dependents.

Getting fields to line up correctly can be very challenging, particularly in the presence of column-spans and row-spans.

This format is often unsuitable for capturing electronically and feeding into backend systems, because often multiple logical fields are compressed into a single line. For example, in the above example the "city, town or post office, state and zip code" are all combined into a single field. This is easy for a human to read, but quite difficult to split into separate fields to upload into a back-end database.

This style is particularly unsuitable for re-flowing onto different screen sizes, because the fields have very fixed widths and positions, and any attempt to reflow them causes the entire form to loose its visual integrity.

There is generally little consistency within these forms, because they are completely hand created, and often tweaked many times over a long period. The sample above uses text of various different fonts, different techniques for displaying help text, and different styles of heading.

The result of trying to maintain electronic forms in the Boxy style is:

Confused and unhappy users due to low usability

Unsatisfactory forms on tablets or smartphones

Frustrated form developers and high maintenance costs.

It is possible to create Boxy style forms in Composer by creating fields with outer borders, and ensuring that at least one field on each line is set to Horizontal Fill (or other techniques), but, this is a challenge. We generally recommend that it be avoided.

We suggest the following approach:

Use one of Composer's existing stylesheets as the basis for your own organizational stylesheet. This ensures that your forms look and function well on all types of devices including tablets, and on multiple screen sizes, as well as allowing you to make use of advanced features such as hide/show logic, repeating sections, and intuitively laid out forms.

If there is a need to produce a printed or electronic form that is exactly the same as an existing paper form, import

the existing form into LiveCycle Designer as artwork, add overlay fields, and use this form as the receipt in Transaction Manager. You will need to bind the fields in the artwork form to the same XML Schema that is used and produced by Composer, and then all the fields will simply flow across from the interactive form to the receipt. The artwork form is generally quite easy and quick to implement, because the layout already exists, and there is no need to add any interactive features since it will only be used for output. You may need to make certain restrictions in your interactive form such as limiting the number of rows in certain cases.

Layout Issues

Measurements

There are several different ways to specify measurements in Composer. There is also some "tension" between the different ways to specify measurements:

PDF is primarily a physical page-oriented technology, and the most natural way to specify measurements is in physical units, such as inches or mm.

HTML is primarily a screen-oriented technology, and the most natural way to specify measurements is in screen units, such as pixels for desktop devices.

Mobile devices (particularly iOS) prefer to measure in points (pt).

In responsive layout, theunit ofmeasurement isthepx, but this no longer means "pixels" as it did on the desktop.

Composer's Desktop style HTML rendering does attempt to mimic the page-oriented layout of PDF, whereas tablet and mobile rendering largely dispense with the page-orientation.

Composer allows you to enter measurements in any one of these units, and will convert these units on your behalf. Conversion from screen units to page units assumes a particular DPI, and mixing these units may result in slight discrepancies.

Measurements can be changed within a form for field widths and for margins. Most other measurements should be left alone, as they are balanced for use within the stylesheet.

Composer also allows you to specify measurements in percentages or weights if a layout manager is being used — in this case, the actual measurement will be calculated by Composer based on the relative sizes of several different components.

Images height and width are usually specified in pixels, based on the actual pixel size of the image. If you resize an image, best results are often obtained by using integral fractions of the original image size.

Recommendations

In general, you should avoid using explicit widths as far as possible. The main reasons for this are:
Often you don't know what size screen your users are going to be viewing your form on, and you should let Composer take care of sizing your fields to fit the screen. Use a layout manager rather than explicit widths. Widths should be specified in the stylesheet, not in the form. This way, if you need to change anything, you can do it in a single place (the stylesheet) rather than having to do it in every single field.
If you do use explicit widths, we recommend:
Most of Composer's in-built stylesheets use page units in mm. Unless you have a specific reason to do otherwise, it's generally recommended to use mm.
If you are building forms primarily for HTML, and require very explicit control over sizes, specify pixels (px). Note that the meaning of "px" has changed with the advent of high resolution mobile devices. See the relevant passage in Responsive Layout.
If you do need to apply particular sizes (e.g. widths) to your fields, consider using a named style-sheet entry, or a layout manager — this is more maintainable and flexible.

Devices vs Desktop

The parameters that define mobile, touch and desktop are changing rapidly as the computing environment is in a state of flux. Time was when "mobile" was synonymous with "touch". Then Apple introduced "Multi-touch" trackpads to its full range of laptops, with the result that classic touch gestures, such as spreading two fingers to zoom in, moved to the desktop. Next, Windows 8 came along and laptops began to have touch screens for an even more direct experience of manipulating objects with the fingers. It is now commonplace for users to interact with commerce sites with their fingers on tablets.

Though we have already (in Navigation) made the point that mobile devices have different requirements from the desktop, those distinctions are becoming more of a spectrum rather than discrete steps.
For that reason, we would like to show you following video presentation. It lasts about an hour, and is worth your time.
http://vimeo.com/64202295
Video from Luke Wrobelewski http://www.lukew.com/presos/preso.asp?31

Delivery to Mobile Devices

Two Modes of Delivery

Mode 1: The Web Browser

As we have already said, the computing environment is now in a state of flux. In the past, when all web browsing took place on the desktop, there was a period — rightly or wrongly referred to as "browser wars" — when web design was complicated by the different and undocumented ways that the various browser of the time would render the same web page in quite drastically different ways. This came at the worst possible time for the web, when HTML standards were also in transition to the adoption of Cascading Stylesheets (CSS), an attempt to separate style from content.

The end result was the concept of the so-called "modern browser", which means one that supports the latest iterations of the CSS standard.

However a new skirmish has broken out, now on mobile. The current state of play is that there are 3 broad types of operating systems in play:

Apple's iOS operating system, which runs iPhones, iPad and iPod touches,

iOS usually updates annually and, because there is little friction impeding this, most iOS devices rapidly upgrade to the latest version of the operating system

Android, originally developed by Android Inc, which was bought out by Google.

There are a number of manufacturers who produce Android phones, most notably Samsung.

Generally, few Android phones update after purchase to the latest Android release, the reasons for which are many and too complex to discuss here.

Also there are a number of manufacturers in China who have "forked" Android and whose devices run no longer on a Google-sanctioned version of Android.

Microsoft's Windows Phone operating system.

This operates on mobile-devices only. It does not have the market share of the above two. Microsoft's Windows 8.

Microsoft has been pursuing a "Windows everywhere" policy and has, with its latest iteration of Windows, intended that Windows 8 runs on both the desktop and on larger mobile devices such as tablets.

To further complicate this, there are two flavors of Windows 8 on Microsoft's own brand of tablets: the Surface. One runs on Intel central processors; the other (the Surface RT) runs on completely different central processors. The two flavors are not compatible or interchangeable and the Surface RT requires a different version of Windows.

And if that were not enough, every copy of Windows 8 ships with two different browsers: Internet Explorer for the familiar user interface of Windows and Internet Explorer running on theModernuserinterface.

This is only the beginning. The different mobile operating systems also have different browsers, usually a default browser that ships with the phone and others that on some or all of the phone models and manufacturers running that operating system

Operating System	Common versions	Default Browser	Other Browsers
iOS	7, 6	Safari	Firefox, Chrome
Android	4.4 (KitKat)	Chrome	Firefox
	4.1 to 4,3 (Jelly Bean)	"Android Stock Browser"	Firefox, Chrome

Operating System	Common versions Previous versions	Default Browser	Other Browsers
Windows Phone	8, 7	Internet Explorer Mobile	?
Windows 8	8.0, and soon to be released 8.1 (64- and 32- bit versions)	IE and Modern Interface IE	Firefox, Chrome on the standard Windows desktop. IE only on the Modern desktop
Windows 8 RT	8.0 (32-bit)	IE for RT and Modern Interface IE for RT	IE only for the two types of desktop.

It is no surprise that all of these have their quirks and incompatibilities. And, the mobile hardware has also evolved rapidly, so that web pages that failed on many of the older mobiles in the past now run quite comfortably on the newer.

Mode 2: Via Native App

As mobile users know all too well, the other means of delivery is through mobile apps, as these generally offer speed and stability in a given mobile operating system. At least, that is the theory.

Avoka offers the TransactFieldApp, where you can publish your forms to the mobile app without having to redesign the form to run in the app. There are even some widgets that work solely in the app and not the browser.

This mobile app is available for iOS, Android and Windows Mobile Phone.

Which Mode is Superior?

The answer is: it depends. If you need to make use of an app-only function, the choice is obvious. If users are coming to the form from a website, the choice is equally obvious.

Just remember, too, that installing an app (which is usually done through an online store, for instance the Apple Store) is a further source of friction for users.

Responsive Layout

Note: Internet Explorer 8 **doesnot supportresponsive behavior** because IE8 does not support modern CSS frameworks. Responsive Design as a practice only began in 2010.

Introduction

Responsive Web Design is a technique for web pages to render properly both on the desktop and mobile devices. A responsive page can be served to desktop browsers or to mobile device browsers, such as smart phones or tablets, and will render optimally on all of them without the need to detect ("sniff") on what device the form is being rendered, nor the need to serve different versions of the page.

It follows that all forms should now incorporate responsive elements and you should now assume that many users will be accessing all forms from mobile devices.

As we have already said, there are 2 modes of delivery tomobile.

The responsive behavior of a form is controlled by the template associated with the page. There are a number of settings in Nuts & Bolts under "Nuts & Bolts -> Responsive Rules" and these are tuned to the Maguire-style of template.

The Maguire Responsive Behavior depends on 3 thresholds values of the width of the device measured in "px".

This no longer means a number of physical pixels on the device's screen, because many modern devices now have higher resolutions than the monitors of the previous century, yet the expectation is that pages should display the same way as they did before. Also there are devices that have high resolution models, "Retinadisplays" and the like. Again, the expectation is that objects should be rendered the same size on these different device models. Objects cannot be rendered smaller on the higher resolution models because of the touch-based interaction with the user: smaller objects frustrate users as they attempt to accurately hit touch targets.

So the portrait mode width of an iPad (for both Retina and standard resolution models) has been standardized as 768 px and an iPhone as 400 px.

An Operator in a Ruleset determines when the responsive rule set kicks in. The determining factor is the width of the device's browser, measured in "px" (also called "css px" in other documentation), which no longer means a number of physical pixels on the device's screen, because a number of modern devices now have higher resolutions than the monitors of the previous century, yet the expectation is that pages should display the same way as they did before. Also there are devices that have high resolution models, "Retina displays" and the like. Again, the expectation is that objects should be rendered the same size on these different device models. Objects cannot be rendered smaller on the higher resolution models because of the touch-based interaction with the user: smaller objects present difficulties for the fingers of users to accurately hit touch targets.

So, the value of the width of any device in px is the value that the device itself reports back to the engine handling CSS rendering in its browser and in apps running on the device.

So, here, for comparison, are the widths that apply to Apple mobile devices:

Device	Portrait width in px	Landscape width in px
iPhone 4, 4s	320	480
iPhone 5, 5c, 5s, late model iPod Touches	320	568
iPads(both Retina and non- retinadisplays)	768	1024

Composer use of Thresholds

To manage the complex sets of responsive behaviors shown in Navigation above, Composer 4 and the Maguire-style of template have 3 levels of thresholds, set in "Form Options -> Generation Options -> Edit Properties -> Properties -> HTML Generation" in the Responsive panel. The default values are:

Threshold Level	Triggering width (in px)
1	768
2	560
3	400

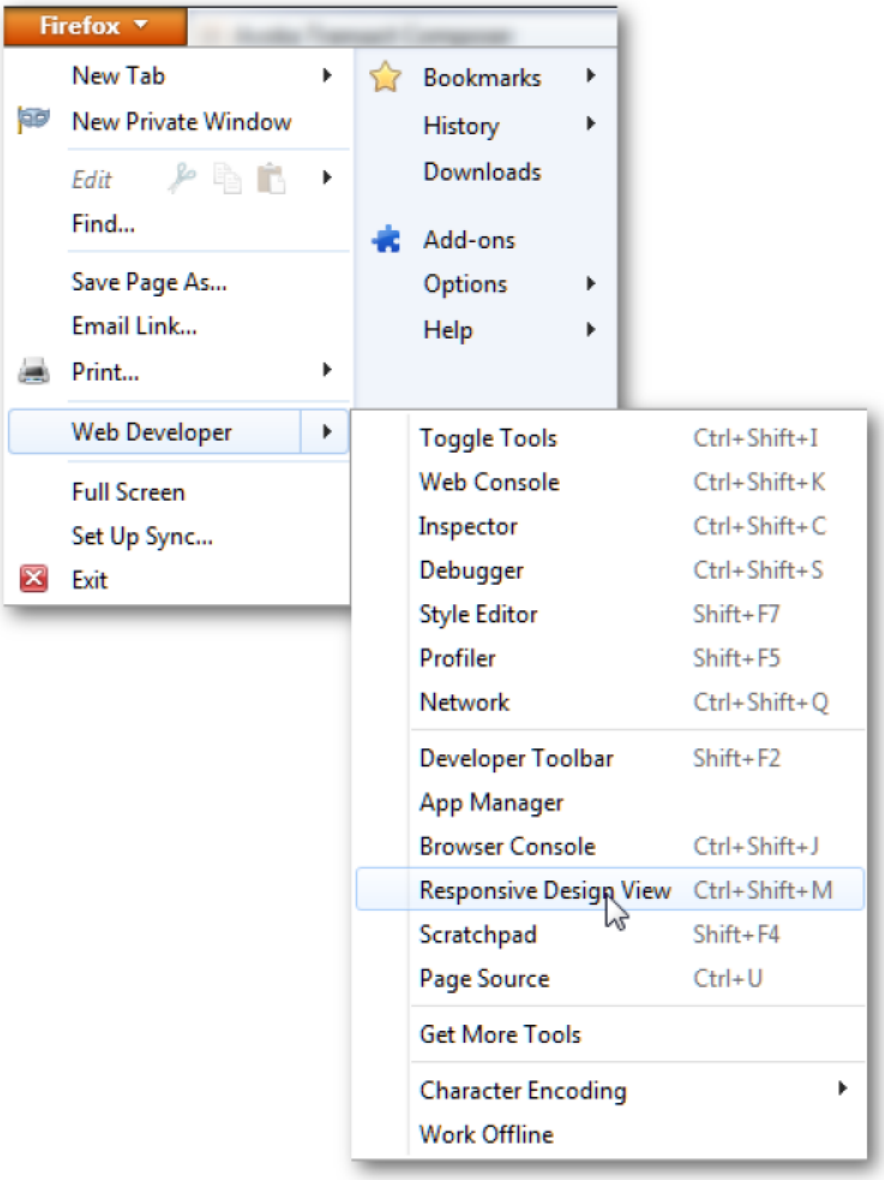
Previewing Responsive Behavior

This explained in Renditions. The default setting for Composer is to have only 2 preview settings, "HTML [Responsive]" and "PDF". If you want, you can separate out the HTML preview modes into Desktop and Responsive through "Form Options -> Generation Options -> Edit Properties -> Properties -> HTML Generation -> Responsive -> Responsive Mode [HTML]". There is really no need to do this in Composer 4; better adjust the width of the browser windowing displaying the HTML Preview and watch the effects as you move through the thresholds.

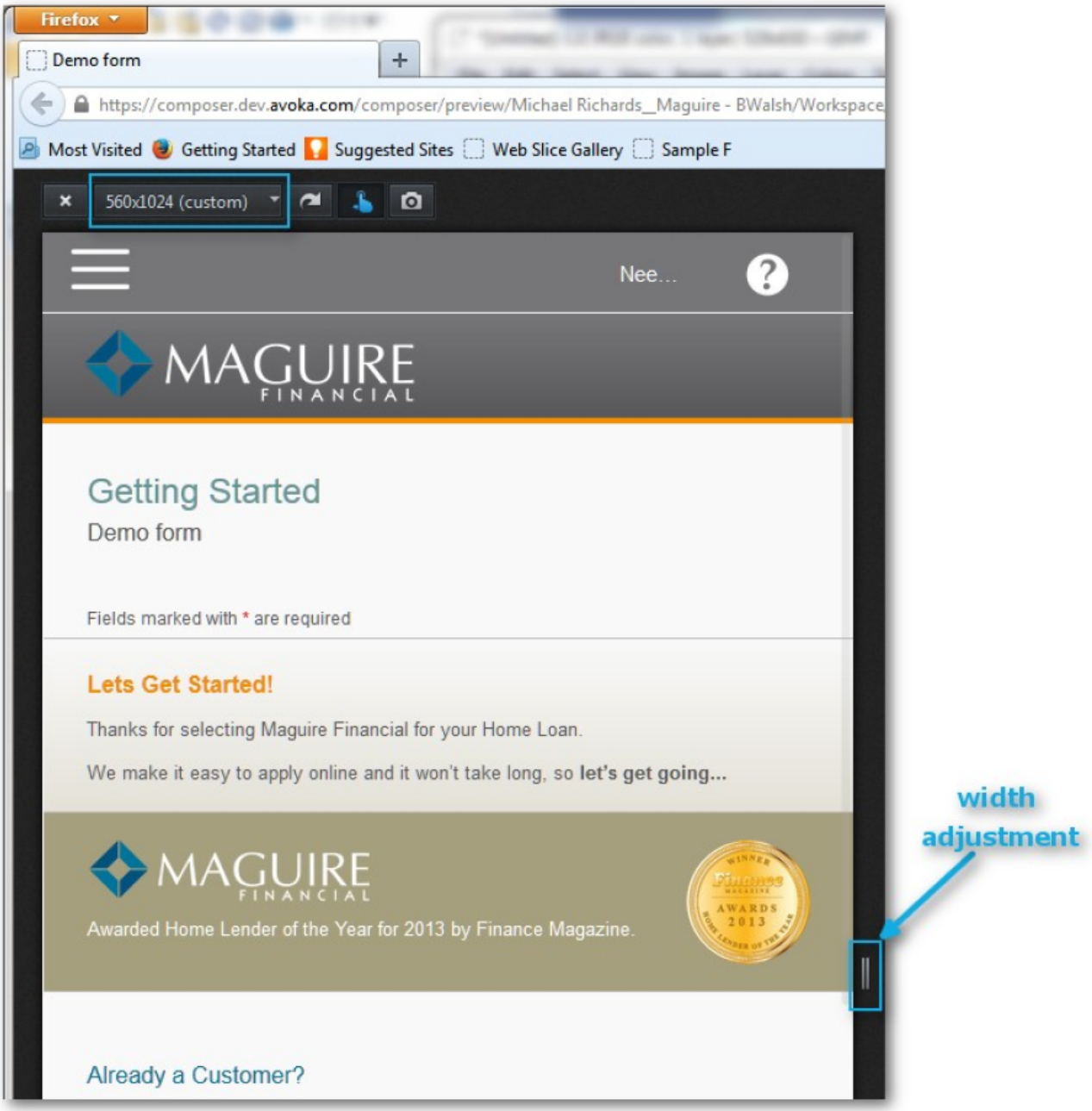
To test responsive behavior, select HTML preview in the dropdown menu, top left of the Form Designer:

The preview will usually show in a new tab on your browser. To test responsive behavior, pull the tab out of its containing browser window, and make the new window re-sizable (done using the window control icons on the top right of the window in Microsoft Windows, or the top left candy-colored icons in OSX). As you narrow the window, you pass through the thresholds and doing so triggers the next stage of responsive layout where the layout of the form changes accordingly. The changes are quite dramatic:

Hint: Firefox has a convenient feature for viewing responsive behaviors. Other browsers do not have such a ready-made facility. You access it in **Windows** from the orange dropdown menu on the top left of the browser window. in **OSX** from "Tools -> Web Developer -> Responsive Design View" in the top menu.



This results in a read-out of the width of the rendered page:



Grid Layout Caveats

Many grid layouts do work in Responsive forms, but not all.

Grids with more than 4 columns may look unsatisfactory above threshold 3. You must preview your form above and below the various threshold widths.

All grid layouts are fine below threshold 3.

Adjust the "Size" settings or the number of columns in the grid for a workable solution.

Actions

In Composer, the Nuts & Bolts Responsive Rules (which, unlike previous versions of Composer, now depend on the threshold levels) are now read only. You cannot alter the Responsive Rules in the Structure Panel.
You can, though, point various elements in your form to whichever of these rules is appropriate for that form element (through the "Edit Properties -> Rules -> Responsive -> Responsive Ruleset [HTML]" dropdown menu).

Wrap

The "wrap" action takes inline fields and, when responsive behavior kicks in, places each of these on separate lines.

Wrap and fill

Inline fields are placed on separate lines and all fields stretch across the width of the display.
Note: tables **must** be set to "wrap and fill" or they will not lay out properly in the narrower thresholds.

Fill

The "fill" action should only be used in Wizards with left-hand menus. For the effect, see the iPad rendering oftheexampleWizard above. Otherwise, the "fill" action only has meaning for responsive behavior for objects that require special scripting.

Hide

Some elements should not be shown on mobile devices as their presence is inappropriate for the layout on that device.

Hide (no media support)

Some elements should not be shown on mobile devices, due to their fixed space requirements or their relevancy.

Show

The opposite of "hide". In other words, the visibility of the form objects allocated to the show responsive rulesets is determined by the device's position among the 3 thresholds.

In Line Validation Methods

See Validation. "In-line validation" means that users receive messages as they proceed filling out the form, rather than receiving a message when they reach the end of the page or form. In-line validation is of the utmost importance on mobile devices, as they reduce user friction if their entry in a field does not pass validation. The last thing they would want is to be told at the end of filling in a multipage form that they have to reverse back through the form, especially in a touch environment where doing so can be difficult and frustrating.

Tool tips

Touch interfaces have a few differences with the desktop mouse-dependent user environment. In touch there is no concept for hovering over an object or text. Instead, users can hold their fingers over some text or object, and other functions of the user interface come into play, such as the "magnifying glass" cursor control for moving through text, or activating the copy and paste controls. Tool tips have no meaning at all in the touch- driven world of mobile. You must therefore not rely on tool tips to guide users through the filling process.

Submissions & Attachments

Composer | Form Builder | v4.3 & Higher

So far, we have discussed how to use the Form Designer, Navigation, and Layout, all of which are meant to lead to submission. When users fail to complete submission of the form, the form is said to be abandoned. This is a most unwanted result and a waste of time for everyone.

Users are often asked to upload files, called attachments, as supporting documentation. Most attachments are generated outside of the form, and can be scans, Word documents, PDFs, JPEGs, PNGs and so forth.

However, there are form elements capable of generating attachments from within the form, for example: scribble pads for users to make their signature

camera widgets for uses to take pictures with a mobile device's in-built camera barcode-reading widgets for use with a device's camera.

Defining attachments (i.e. files uploaded with the form) used to done exclusively in Transaction Manager. Now these can also be defined in Composer.

The Preview function of Composer does not include user interactions when they upload attachments; for that, you will have to publishtoTransactionManager.

File Uploading

Defining Attachments in Composer

There are two attachment widget types. You add all of these to the one section on the form, here called "Attachments". It makes sense to have these in the last section of the form which usually contains the "Submit" button.

TM Attachment Table

Add one only of these, and its rows will automatically be populated with the details of the attachments.

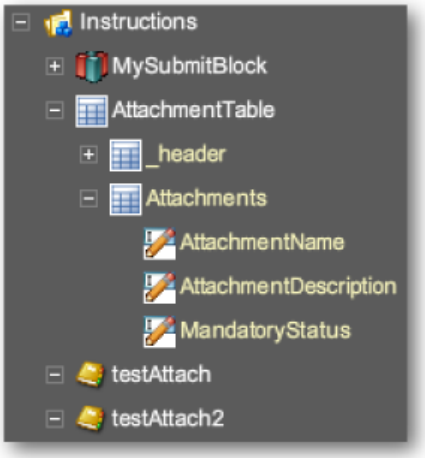
TM Attachment Rule

Add one of these per attachment at the same structural level as the TM Attachment Table. Use the "Create a New Field" wizard, which pops up on adding the rule to the form, to give it an "Internal Name". You can also further specify each attachment's properties in the rule's "Edit Properties" dialog.

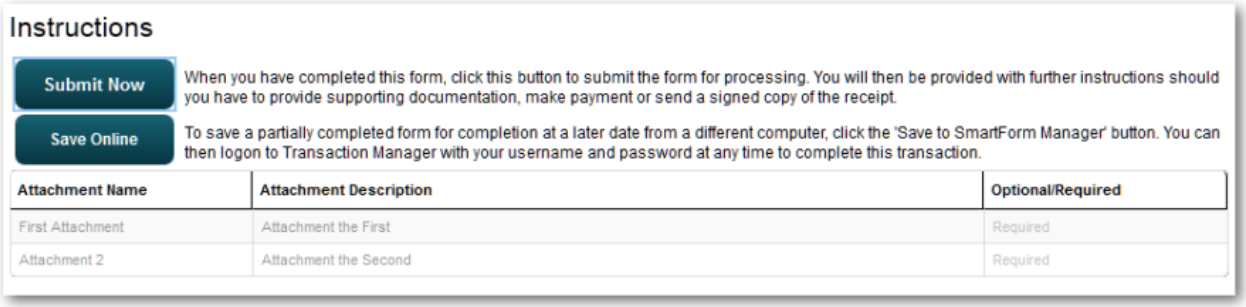
TM Submission Block

You also need submit buttons, so we have added a TM Submssion Block (as "MySubmitBlock") to the section. This block adds several submission buttons to the form, which give users the options to "Submit Now", "Save Online" or "Save to My Computer:"

Here is the resulting structure of adding a "TM Attachment Table" and two "TM Attachment Rule" widgets to a section containing a "TM Submission Block".

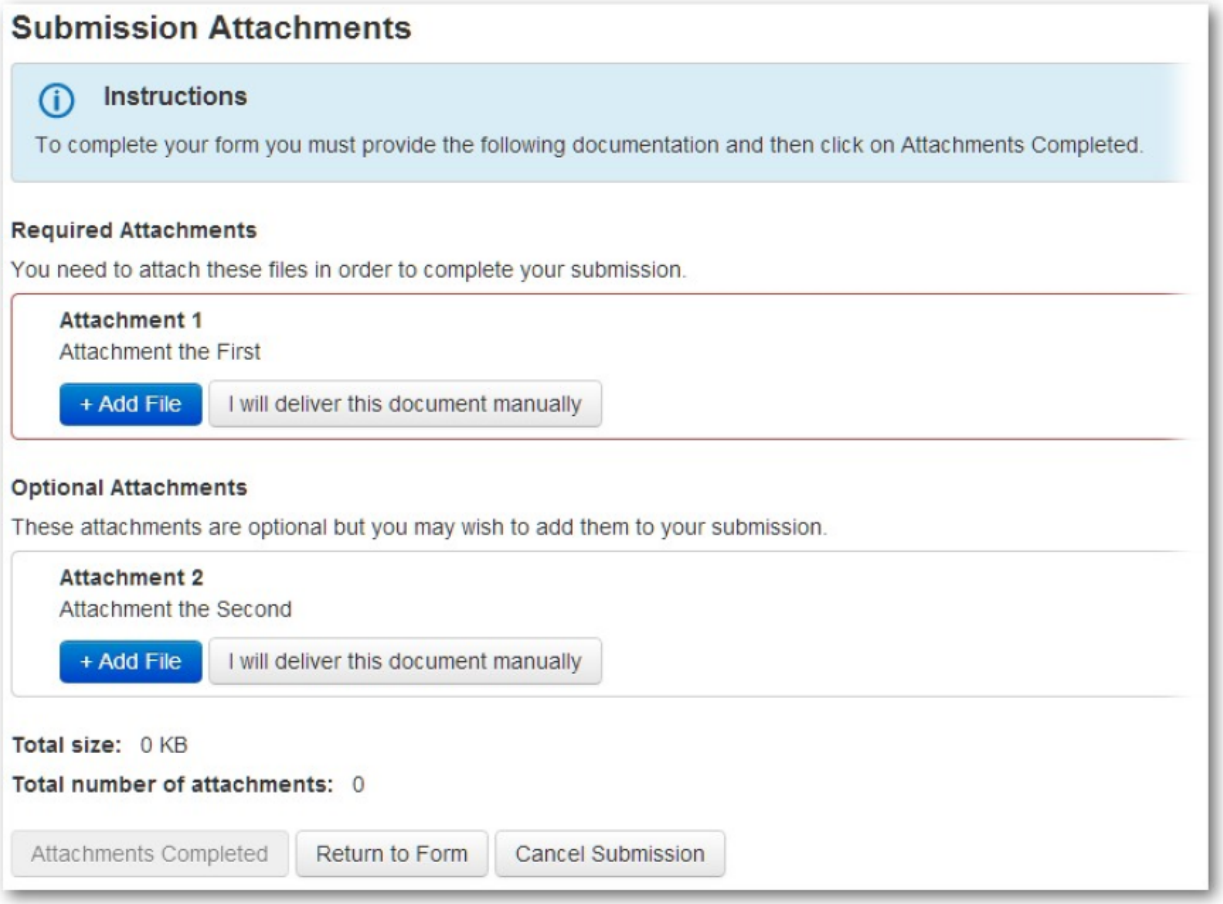


You can then preview the rendition of the form, where the Attachment Table gets populated according the the Attachment Rules:



How this structure renders in preview.

The table informs users that they will be requested to supply these attachment after clicking on the "Submit" button. This behaviour cannot be previewed in Composer: to see this, the form must be published to Transaction Manager. In our example, the form was published without further modification in Composer; all that was done in TM was to assign the form to a portal and to turn off its "Test" mode. The resulting screen after the "Submit" is:



Resuling "Submission Attachments" page.

Delivery of Attachments

The "I will deliver this document manually" buttons appear on the "Submission Attachments" by default for each attachment. To remove them, set in Composer the "Edit Properties -> Data -> Submit Method" for each rule to "Electronic".

Email Acknowledgements

If you require an email to be sent to the user as an acknowledgment, receipt or any other business reason, this must be set-up in Transaction Manager.

There is currently no way to specify emails through Composer. Transaction Manager, though, has a great deal of functionality and control of email messages, Please refer to the Transaction Manager Administration Guide.

Signatures

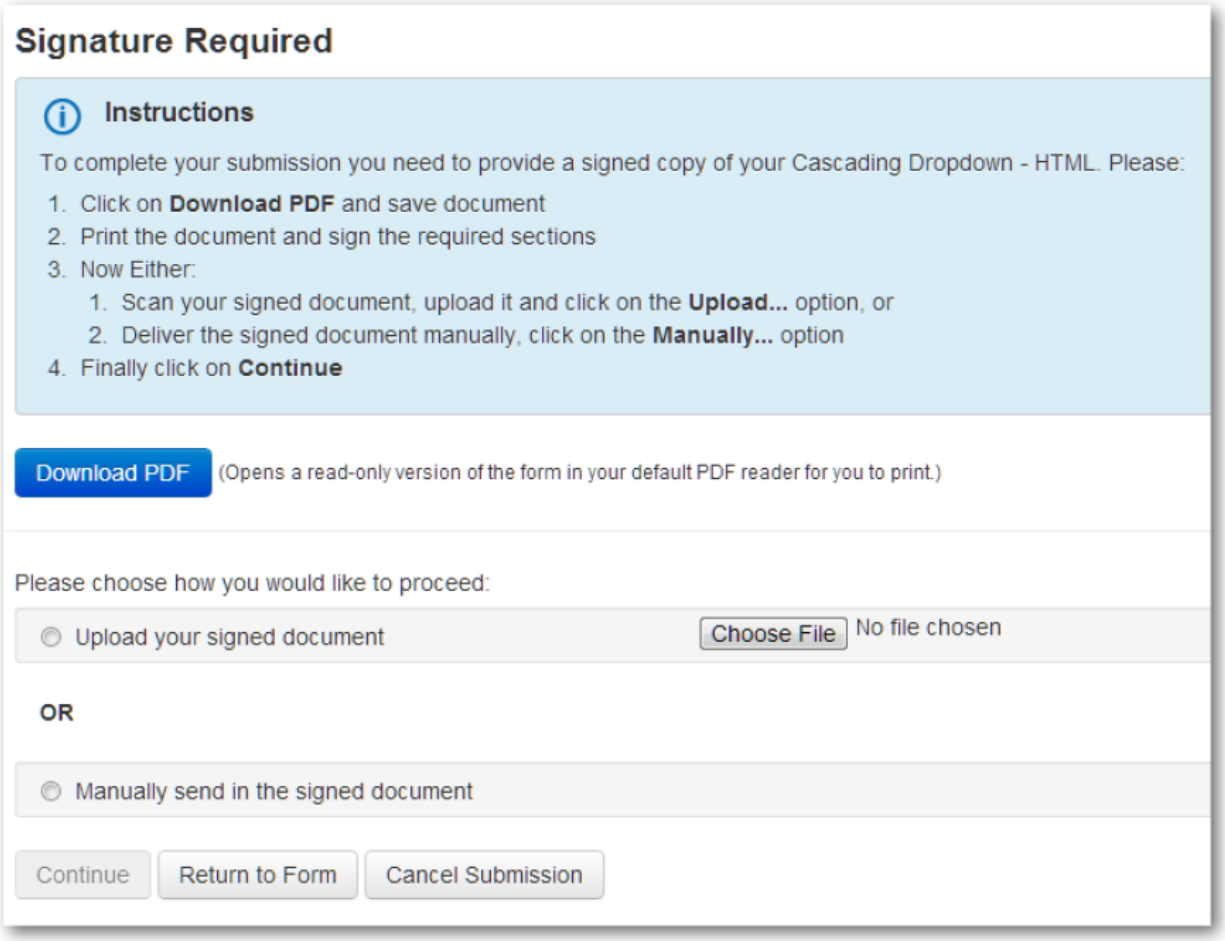
There is a range of signature widgets:

Signature [Wet] and TM Signature [Wet]

Dropping these on to the form has the same effect. Pre-submission, the form presents the user the message, contained in "Edit Properties -> General -> Instruction Set". The default is:

"Signatures are not required at this time. After you have submitted this form, print the receipt, sign it and send it in."

After users have clicked "Submit" they are presented with:



Wet signature form after submission.

DocuSign Signature

Creates a block to handle the DocuSign eSignature process. The users fill in their names and email addresses and DocuSign verifies the users via a two-step process. **Note:** use of the service incurs a service charge.

Signature Pad Widget [HTML]

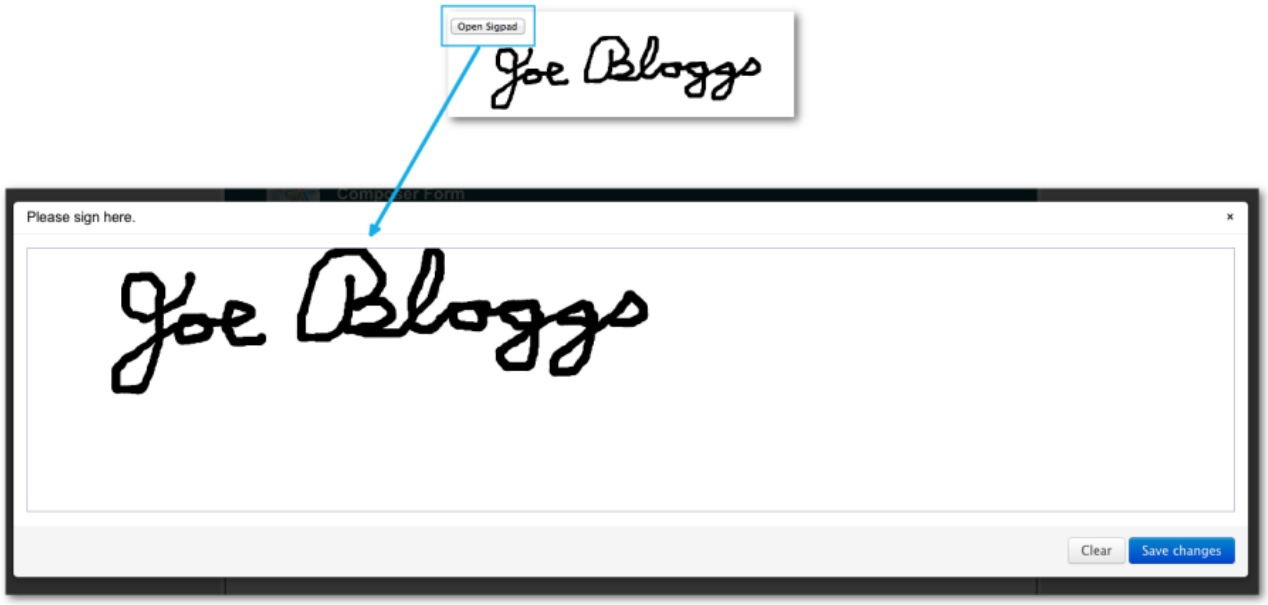
The widget's "Edit Properties -> Properties -> General -> General -> Signature Type" can be set to "Draw It", "Type It", and "Choose at Runtime". "Draw It" gives users a signature canvas to sign with their finger or mouse (see below). "Type It" presents users with a text field.



The signature canvas.

Signature Pad Popup [HTML]

Similar to the signature pad above, except that the pad appears in a separate lightbox superimposed over the form. On "Save", the lightbox vanishes and the signature shows as a thumbnail.



Notes

Use a true Digital Signature if users have a certificate or smartcard available.

Allow the end-user to click a "Sign" button, and specify their username (and the date) if they don't have a digital signature.

In HTML, just use their username and the date.

If in Adobe Reader, and the form is not "reader extended", just use their username and date.

The name and date from a true digital signature will also populate the form's username and date, so in HTML even if users re-use the form, we still have a record of the signer.

When either the true Digital Signature or the Signature field is signed, an invisible checkbox is checked. This will allow standard editability rules to be defined so that the form can be locked down after signing.

Dynamic Data

Composer | Form Builder | v4.3 & Higher

Overview

In Composer, Dynamic Data has two meanings:
Dynamically changing the information displayed on a form without reloading the page

Accessing a third party web service for data for this dynamic display

What is Dynamic Data?

Dynamic Data is an Avoka Transact capability that allows a form to communicate back to the Transaction Manager Server while the form is still being filled in and even display new data on the fly, without the page being reloaded or the form being submitted. This allows an additional level of functionality to be built into forms.
Dynamic Data is a general purpose communication mechanism and can be used for many reasons. Common use cases are:
Sending some data from the form to the server (such as a user id) and retrieving some information to populate fields in the form (such as their address).
Sending some data from the form to the server (such as a user id) and retrieving some information to populate a pick-list in the form (such as a list of the subjects that the user is enrolled in). The pick-list can be in the form of a drop-down list, table, or other visual element.
Sending some data from the form to the server to perform a server-side calculation. Sending some data from the form to the server to perform a server-side validation.
Dynamic Data requires collaboration between the form developer, and a server-side developer who defines a Dynamic Data service in Transaction Manager. Please refer to the Transaction Manager document for more information on this.
Composer provides a number of different facilities for using Dynamic Data:
Widgets that have pre-built Dynamic Data support, such as the "Lookup" Button. An assistantthatbuildslookup-styleDynamicDatacapabilities.

Dynamic Data in Action

We recommended you initiate Dynamic Data invocations by a direct user actions, like a button click, not from implicit actions, such as exiting a field. Direct actions set user expectations that "something time-consuming should now happen, and may even fail." Internet performance and reliability is beyond our control, so using a button sets end user expectations that some action is about to occur.
However, it is also possible to use Dynamic Data to perform "predictive" lookups, such as the list of completion hints seen while typing in a Google search . The AutoSuggest example below shows these predictive lookups using the Veda Geolocation service.

The Dynamic Data Assistant

By dropping the Dyanmic Data Assistant onto the form, and filling in a few details, you can populate your form with a dynamic data fields in a few steps by merely filling in the "Add A Dynamic Data" dialog.

In this example, the Transaction Manager service definition is "ACMEPOST", which in real life would access some webservice from a third party provider. Here, there is only one input field, "Suburb" and the TM service returns multiple sets of 3 fields each.

The "Result Block Type" selector toward the bottom of the dialog gives 3 options:

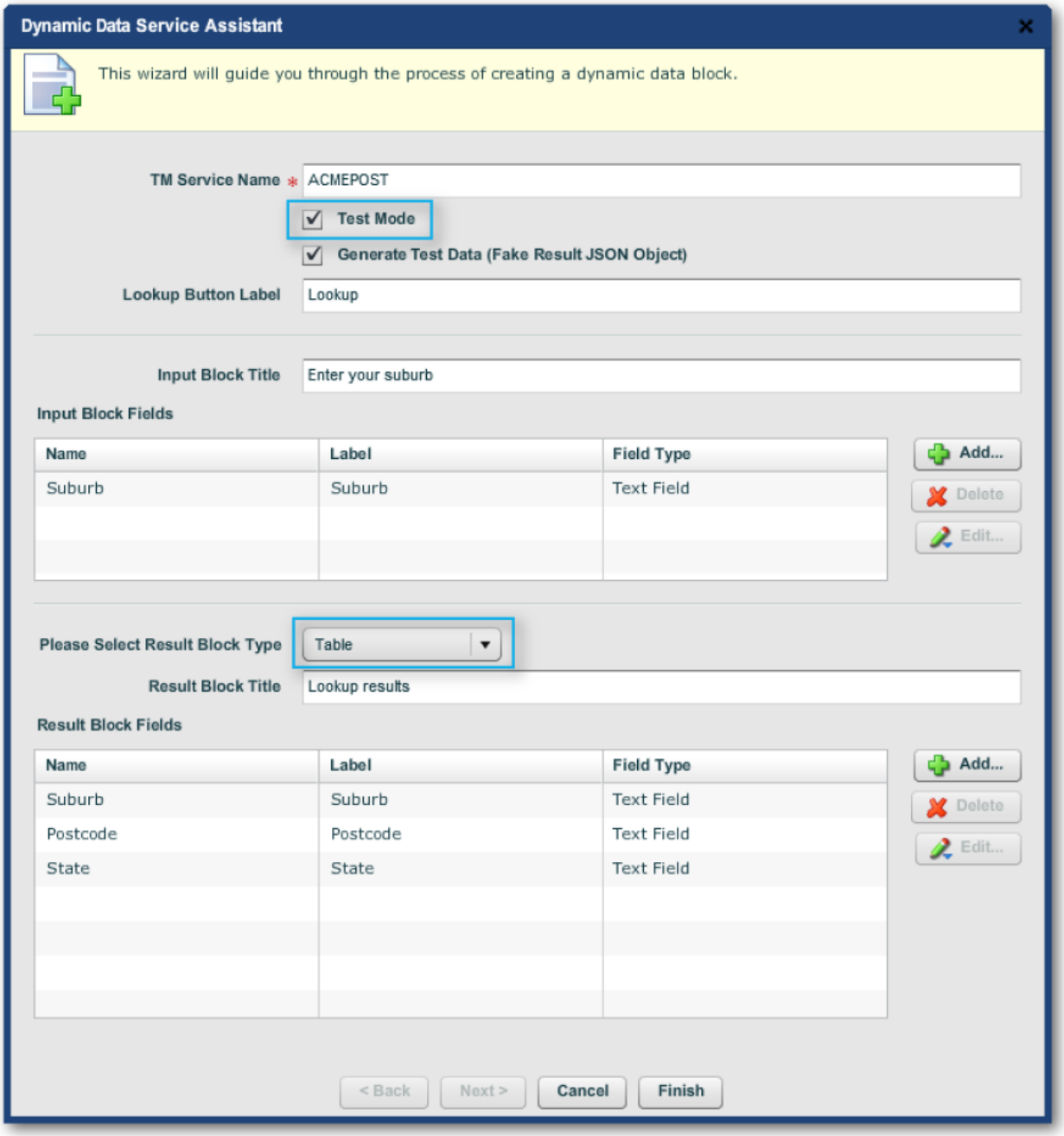
Standard Block

which will display only one set of returned data

Repeatable Block

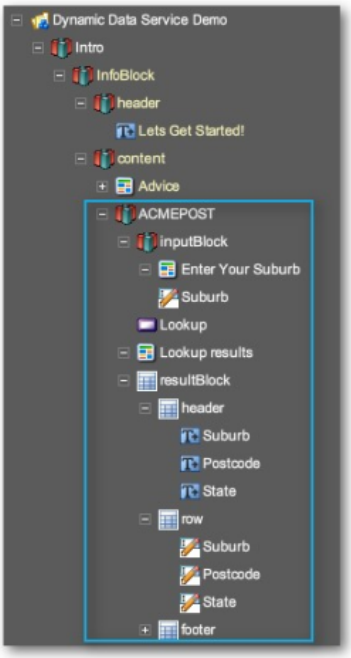
which will display multiple sets of data in the more flexible format than a table structure

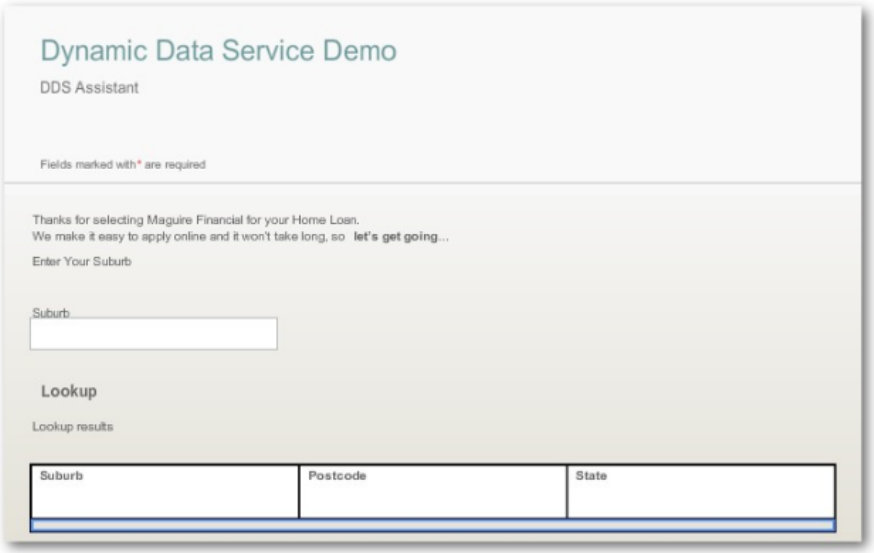
as selected in this example. Each set of returned data will populate a row of the table.
We have also activated the "Test Mode" checkbox. The assistant will then provide a set of fake data automatically. You can also provide your own fake data, as explained below. Of course, for the form to work from real data, the service must already be defined in TM.



The Dynamic Data Assistant wizard dialog.

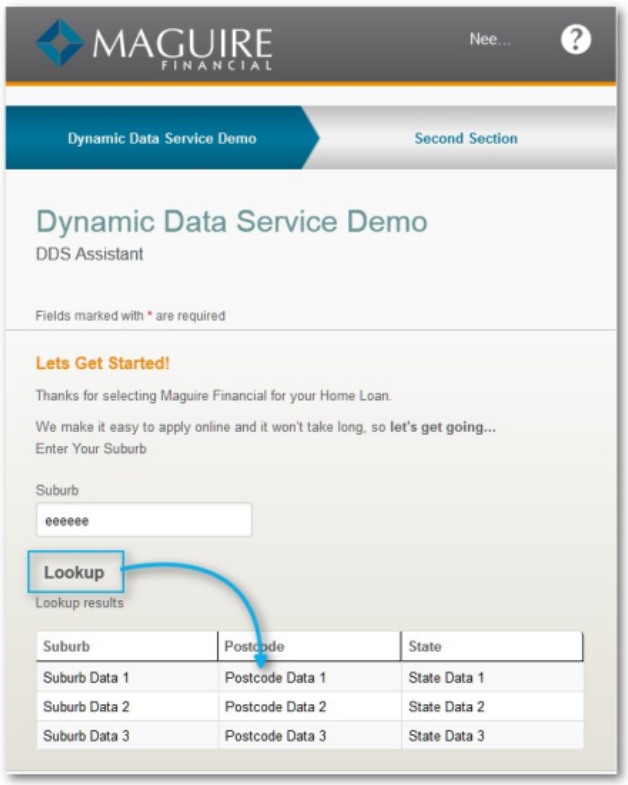
This dialog results in this wireframe and structure:

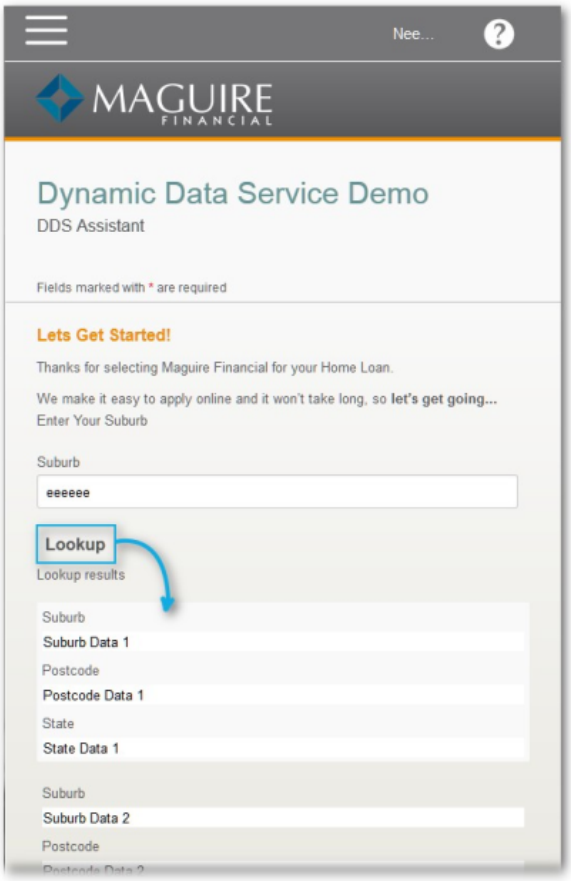




Dynamic Data Assistant creates the following strutcure and wireframe.

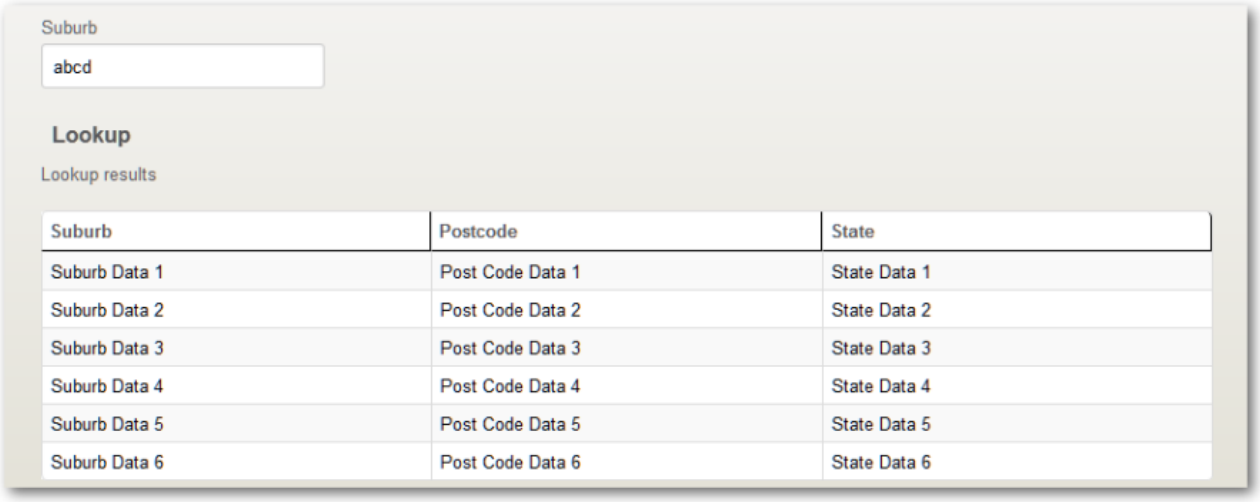
Previewing the form in HTML and clicking on the "Lookup" button results in the following display of the default fake data generated by the assistant.





Preview of the form with fake data generated by the Dynamic Data Assistant, desktop vs narrow mobile

The fake data belongs to the structure's button, here called "Lookup". Going to the button's "Edit Properties -> Properties -> Data -> Dynamic Data Testing" exposes a text field called "Test Data (Fake Result JSON Object)". If you wish, paste your own JSON data here. If the "Result Block Type" is either "Repeatable Block" or "Table", all the JSON fake data will display; if not, only the first set of the fake data will display.



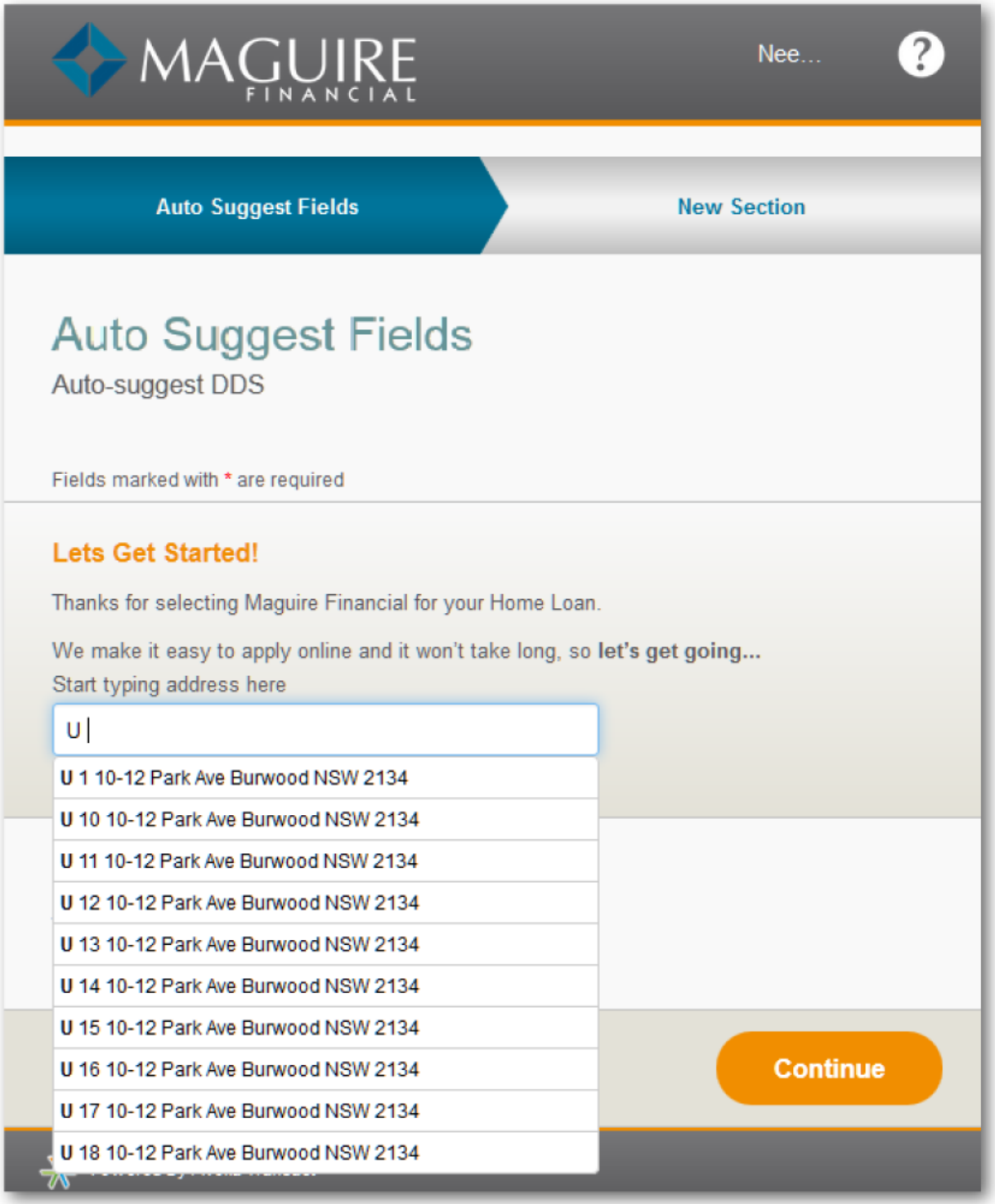
When the service is returning an unmanageable number of hits, consider using pagination.

AutoSuggest

Overview

An auto-suggest field is a text field (and its associated control widget) where the rest of contents of the field are suggested to the user in a popup below the field, and the user selects the correct information. The suggestions are obtained from a dynamic data service (DDS). A common use case for auto-suggest is entering a household address with suggestions coming from a geodata DDS.

Here is how the field looks and behaves:



For the autoSuggest widget to work, the widget configuration has to correspond to the Groovy scripts of the

service in the "Service Definitions" page of Avoka Transaction Manager. In other words, unlike most widgets in Composer, the necessary service creation and scripting must already exist in Transaction Manager before the widget can be added to the form.

The following discusses:

Building an autoSuggest form element from scratch, with a service we have defined as ACME POST in Transaction Manager

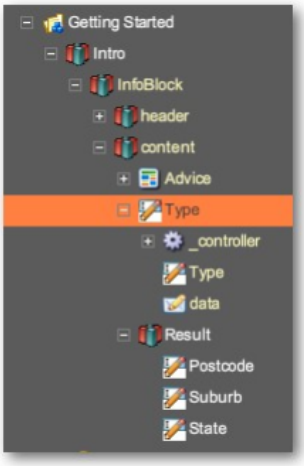
Using predefined widgets and scripts that ship with both Composer and Transaction Manager for the VEDA GeoCoder webservice

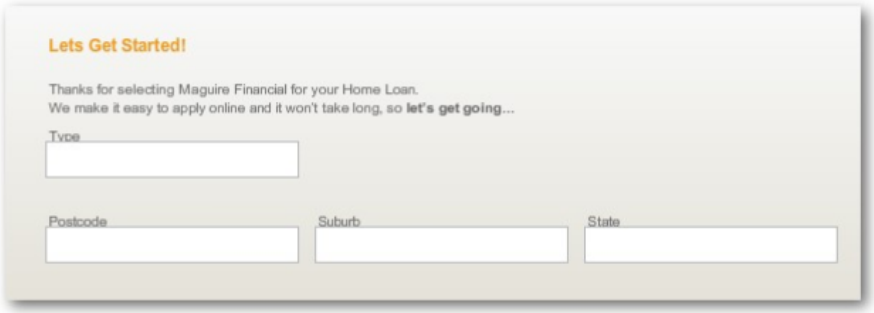
For further detail on the configuration parameters, refer to thenotesbelow.

Reference Data Lookup Block

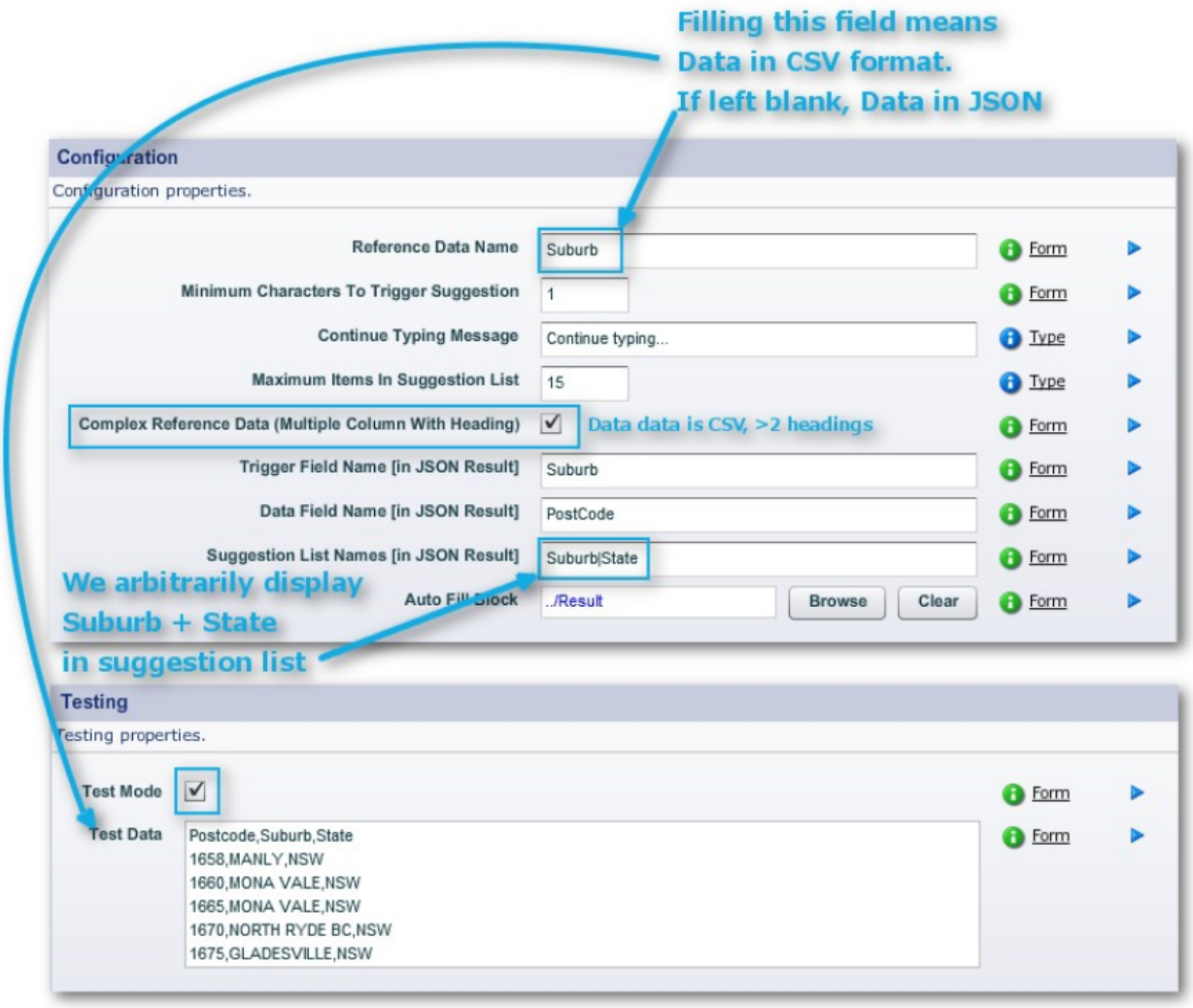
CSV Data With Headings

Another widget, the TM Reference Data Lookup Block, also provides lookup data. The aim of this widget is to give a convenient way to provide an auto suggest function with no programming. The reference data can be an Avoka Transact dynamic data service defined in TM (see the Transaction Manager Administration Guide), or from pre-populated data (see hereforanexampleofhowyouintroduceprepopulateddataontoaform using a Data Field widget). In this example, we use test data.



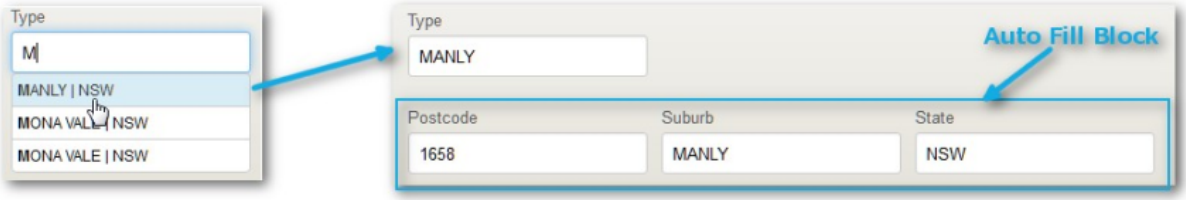


The only change we made to the structure built from the block was to build the "Result" block (3 in-line text fields named according to the table headings of the CVS test data) and point to it in the "Configuration" panel. Note the "Complex Reference Data" checkbox: it is activated because the data is a table with more than 2 column headings.



Panels in "Properties -> Data" for the "Type" node (a TM Reference Data Lookup block)

Which results in the following behavior:



CSV Data Without Headings

You can have only two columns of data, that is only one pair of data points per row separated by a single comma. Make sure the Complex Reference Data checkbox is unchecked.

So, as there are now no headings, the convention is that the left data point or column is refered to as "display". The right data point or column is called "data". The default parameters of the Configuration Panel is to have the left column, "display", as the "Trigger Field Name"; the right is the "Data Field Name".

Notes on Auto Suggestion _controller Data Configuration Parameters

The following tables describes some of the "TM Auto Suggestion Controller" parameters and gives the default abstracted values provided off-the-shelf in the "TM Veda Address Lookup Block [Australia] -> _controller". If you want to change these values, you should do this in the TM Veda Address Lookup Block [Australia], not in its children.

Note: you still nominate to use test data in the "TM Veda Address Lookup [Australia] -> _controller -> Data -> Test Mode" checkbox.

AutoSuggest _controller -> Properties -> Data -> various panels	Parameter Description	VEDA Controller Abstracted Value
Dynamic Data Service Name	Points to the DDS	$../{dds.service.name
Minimum Characters to trigger auto suggest	The number of keystrokes before the auto-suggest popup appears	$../{dds.min.chars}
Continue Typing Message	A message displayed in grey in the trigger field while the user has still not made a selection from the suggestions presented	$../{continue.message}
Not in List Message	Displayed at the bottom of the suggestions if the typed input in the trigger field is not an exact match to one of the suggestions presented. The user clicks on the message to leave the trigger field. What then happens is determined by the "Clear Trigger Field When Not Matched" checkbox	$../{notInList.message}
DDS Fail Message	Displayed if the service times out. This is usually left blank.	[blank]

Clear Trigger Field When Not Matched checkbox	If the user leaves the trigger field, and the entry does not match any of the suggestions, the field becomes empty. Otherwise, the un-matching entry will remain. The "Not in List Message" will be displayed.	[checked]
Second DDS Service Name	Enables another layer of dependent data to be accessed from a DDS service and receive more auto suggestions in a secondary field. It follows that there are secondary input fields, and that test data can also be pasted into another Test Data field.	$../{dds.-detail.name}
Timeout	specifies number of milliseconds before the DDS access fails. This value is usually left as "0".	0
Delay of Trig-gering DDS (in mil-liseconds)	sets the delay before the user input triggers a query on the DDS. If this is set too short, there will be usability issues.	200

"Hide Standard Address Block" in VEDA Block Widget

This is a special checkbox in "TM Veda Address Lookup Block [Australia] -> Properties -> Data -> Veda Configuration": the "Hide Standard Address Block". Its behavior is as follows:

When **checked:**

- Initally only the trigger field is visible and the address block invisible,
- If the user selects one of the suggestions, the filled-in trigger field only is visible
- If the user selects "My address is not listed", the address block becomes visible and the focus moves to "Address Line 1".

When **not checked**:

- Initially, both the trigger field and the address block are visible
- Initially only the trigger field is writable
- If the user selects "My address is not listed", the address block becomes writable and the focus moves to "Address Line 1".

In **both cases**:

- The trigger field will be cleared, regardless of the setting "_controller -> Clear Trigger Field When Not Matched".

Sequence Numbers

Sequence numbers are enabled by these lines at the end of the service definition Groovy script in Transaction Manager:

```
// See if there is a queryString holding a sequence number def value = request.getParameter("seq")
 if (value == null || "".equals(value)) {
// The request parameter 'seq' was not present in the query stringOR
// The request parameter 'seq' was present in the query string but has no value return result
 } else {
// add the sequence number back on theresult
 result += new JSONSerializer().toJSON("{'seq' : '" + value + "'}") return result
 }
```

If there have been a number of requests made on the DDS, the responses may not be returned in the same order as the requests. If sequence numbers are being emitted in response, the value of seq is incremented with each response, allowing TM to tell if responses came back out of sequence. TM will then only act on the most recent response, that with highest seq number.

Internet Explorer 8 Issue

End users on IE8 can receive multiple alert dialogs. The workaround for this is setting the "TM Auto Suggestion Controller -> Edit Properties -> Rules -> Click -> Failure Notification" and choose "Do Nothing".

Pagination

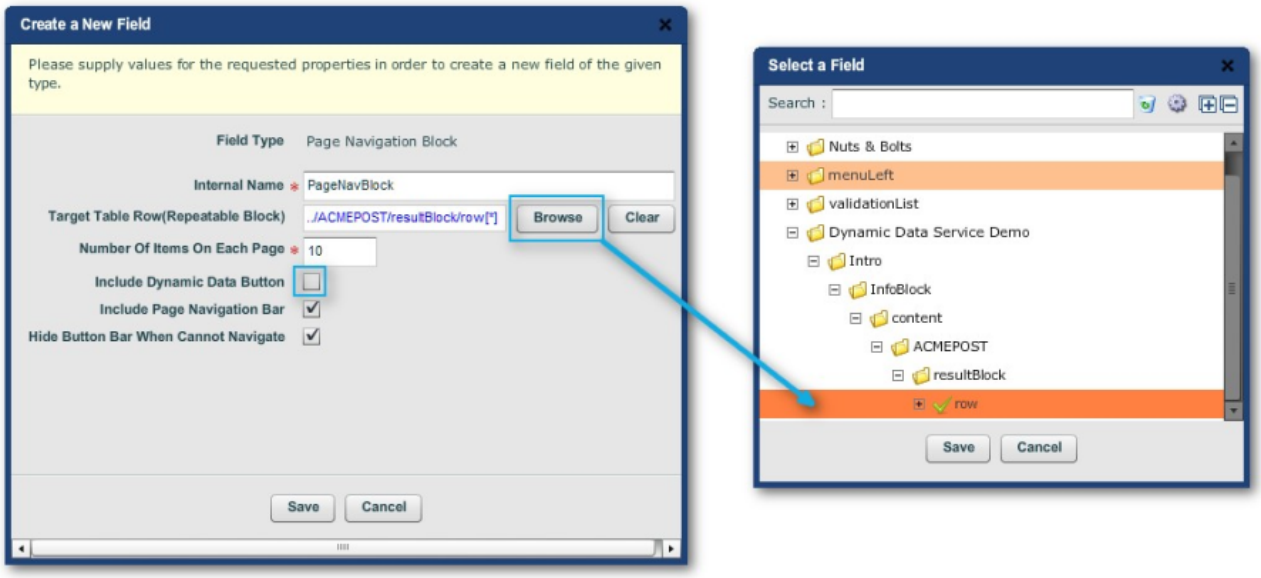
Composer now has a "Pagination Block" widget (currently specifically for Dynamic Data using a "Lookup" button). It is very simple to add to a form using a Dynamic Data Service with a table in the result bock:

just drop in a "Pagination Block" onto the form

point the lookup button of the input block of the dynamic data service to the page controller of the Pagination block

point the page controller of the Pagination block to the lookup button of the input block We illustrate this by adding pagination to the ACMEPOSTexample.

Add from the Palette, a "Field Types -> Pagination -> Page Pagination Block" element. Fill in the resulting dialog:

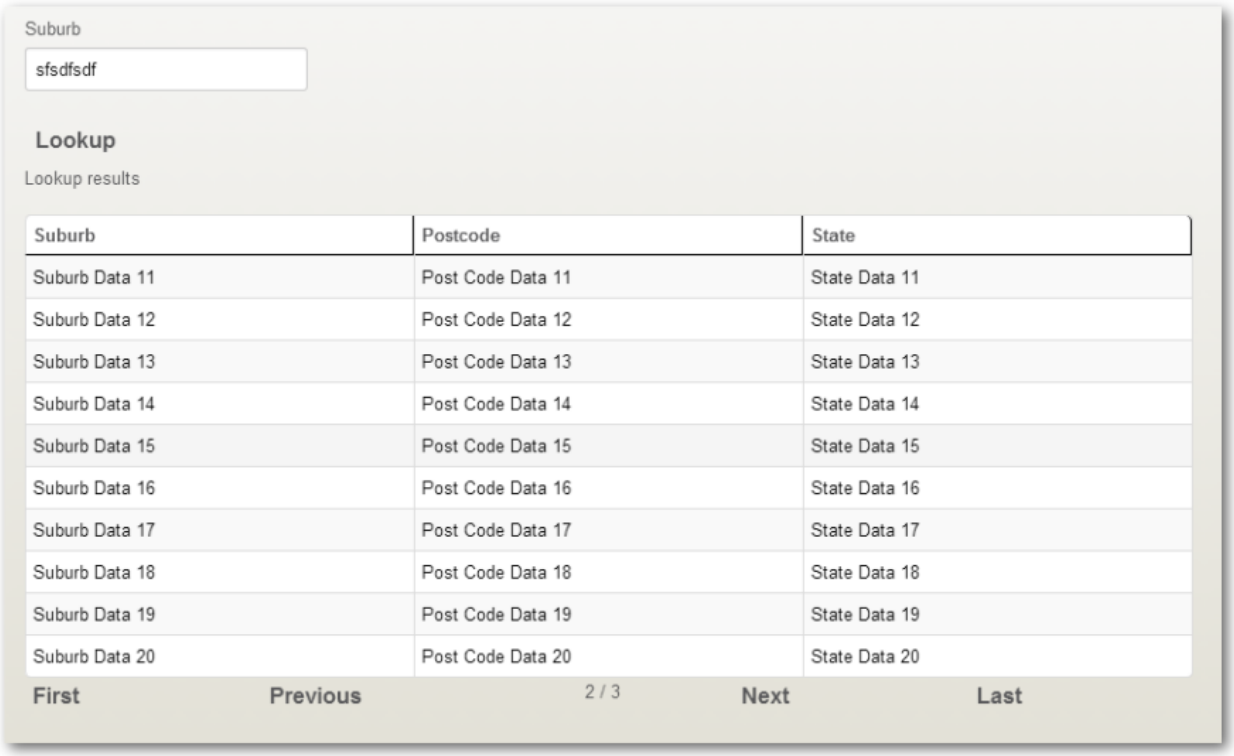


Adding a Page Navigation Block

Note: the "Include Dynamic Data Button" in the Page Navigation Block is not activated in this example: there is already a button in the ACMEPOST Dynamic Data block.

And. *vice versa*, you point the Dynamic Data Service back to the page navigation structure, via the ACMEPOST Lookup button's "Edit Properties -> Properties -> Data -> Data -> Page Navigation Controller".

There must be enough rows of fake data in the result table to activate the pagination functionality. If so, the pre-view looks like this:



2nd page of the page navigation output (using fake data).

Please refer to the Responsive Layout section about viewing the result table on mobile devices. You should give some thought on how you want the table to appear [intheresponsivelayout thresholds](), where each table cell gets its own line instead of being laid out in rows.

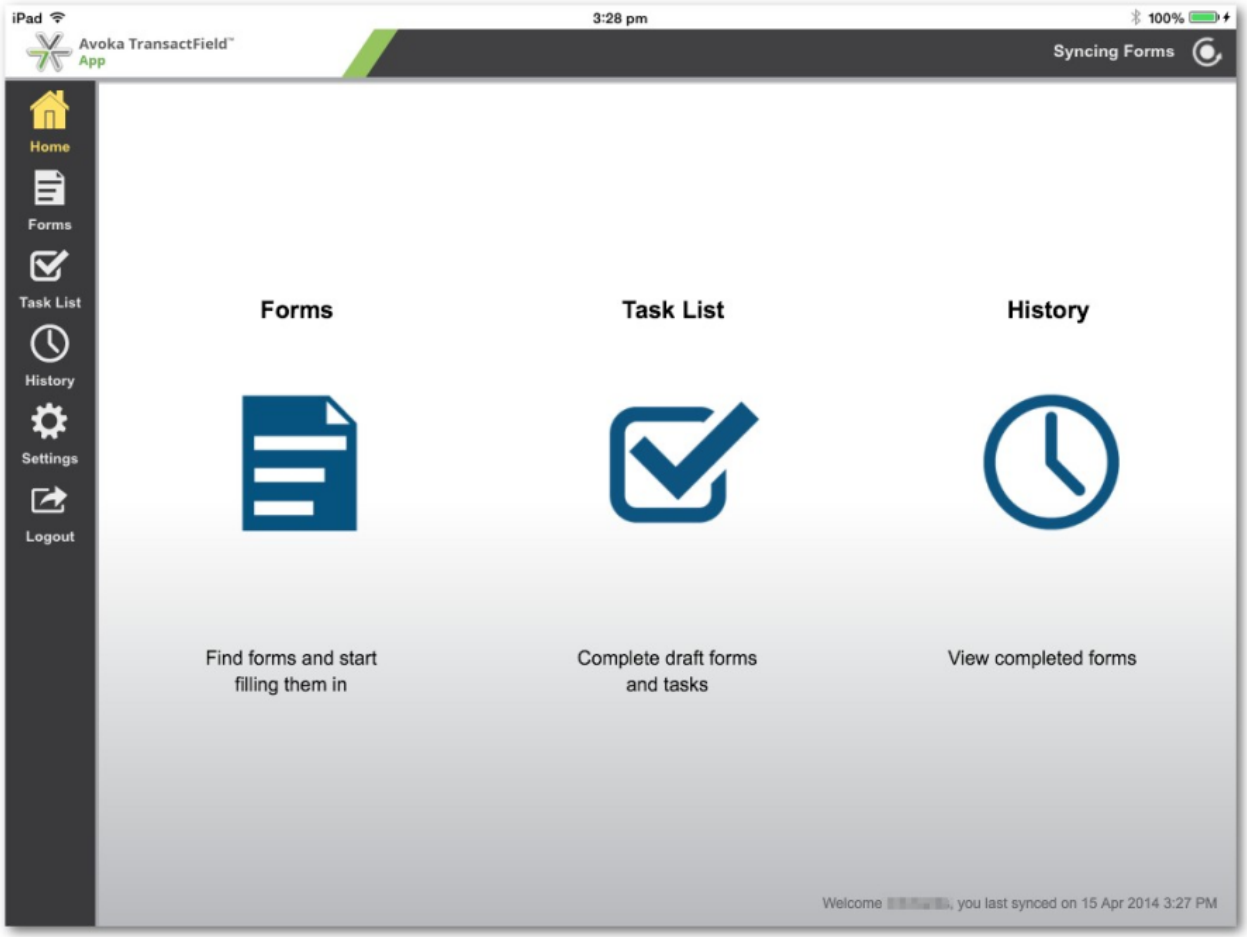
TransactField Mobile App

Composer | Form Builder | v4.3 & Higher

Avoka Technologies offer a couple of ways for users to access your forms on mobile devices:
Through the device's web browser, with Responsive Layout technology
Via a native TransactField App.

About the App

The Avoka TransactField App runs on iPads, iPhones, Android tablets and phones, and Microsoft Windows devices.



The Avoka TransactField App's main user interface

It delivers to these devices intuitive mobile data-capture applications that are built using the Avoka Transact platform. The TarnsactField App client affords the following capabilities to mobile workers:
Allows field workers to take their forms with them out in the field
Gives field workers a task list of forms to complete or approve and a timeframe in which to do these. Synchronizes forms automatically to the device so that they can be filled out online or offline
Holds submitted forms temporarily on the device, completing the submissions when reconnected Enables forms to make use of device capabilities, including camera, GPS, and sign-on-glass Enforces security, including password protection and encrypted data on the device and over the wire Provides full control over which forms go to which field workers through permissions and roles Synchronizes to all apps when a new form or an updated form is deployed.
Please refer to the Transact Mobile App Guide for more on these functions. There is very little to do in Composer to utilize these: they are taken care of by TM and by Composer's scripting.

How to Publish a Form to the Mobile App

Use the following steps:

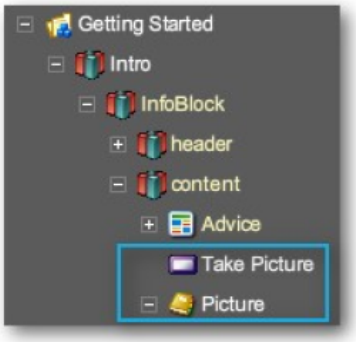
1. Save the form and publish it toTM
2. In TM, save or import the form
3. In TM at "Forms -> Form -> Details", uncheck "Test Mode
4. In TM at "Forms -> Form -> Portal Access", make sure "Mobile App" is one of the "Assigned Portals".

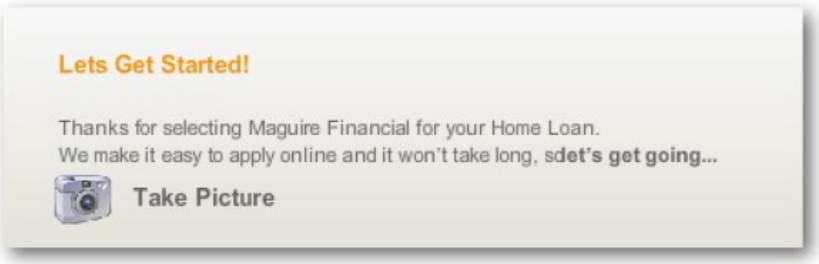
Now, when you open the mobile app on a device (making sure that you are connected to the correct instance of TM) you should — after the synching process is completed — see the form by tapping on the "Forms" icon and be able to select it in the forms list.

Mobile Camera Support

Mobile devices came with in-built cameras, and so users want to be able to integrate this picture-taking capability into forms. Users tap on a form's "Take Picture" button widget and generate an image attachment. This gets uploaded when the form is submitted.
Though the attachment generated by the button is like other attachments, the Button [Take Picture] widget is unique in that:
It only works in the TransactField App
It is not supported on the desktop or in a mobile device's web browsers

The Standard "Take Picture" Button



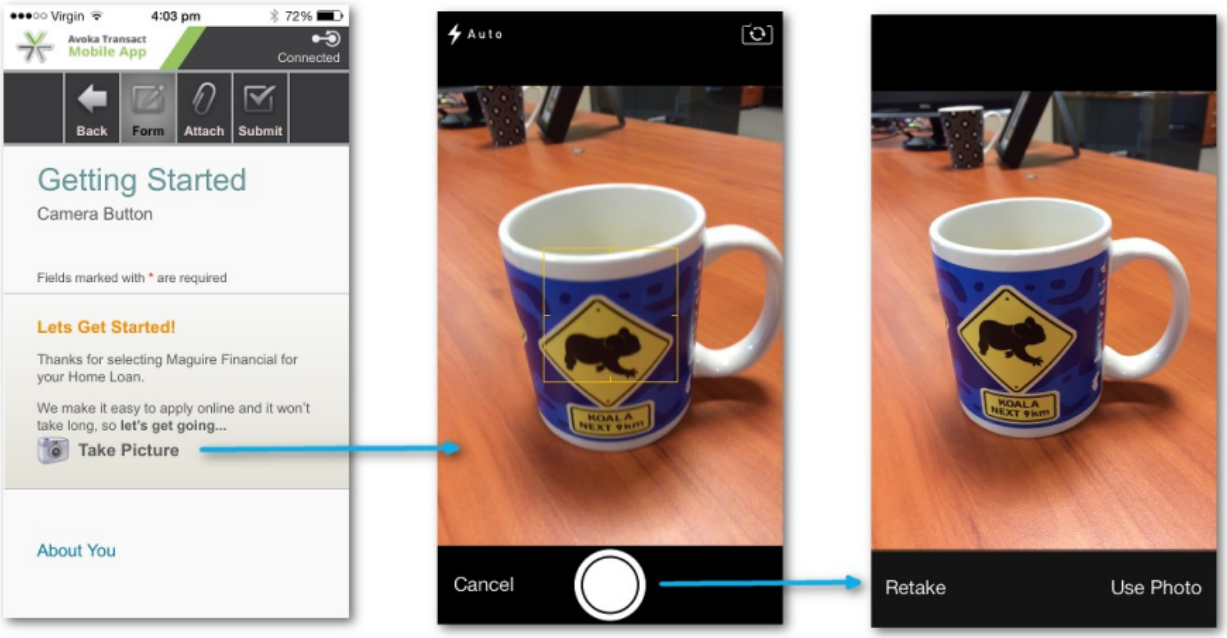


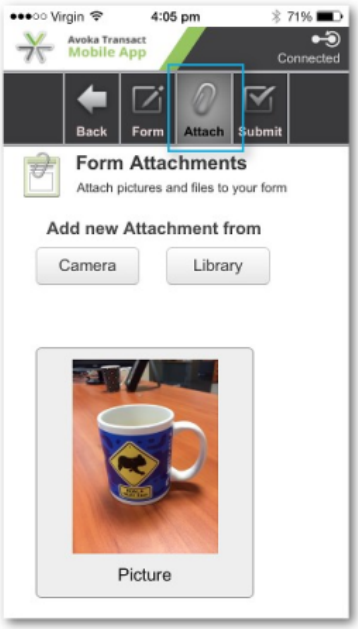
Very simple form with a camera button

The "Button [Take Picture]" widget works only in the Mobile App. Tapping on it takes mobile device users to the in-built camera app. The user gets to take the picture and choose either to retake or use the photo. It is not saved to the device's camera roll. For the sake of appearance, we have set "Take Picture -> Edit Properties -> Styling -> Style Sets -> Button Type -> Push button".

The form also needs a TM Attachment Rule widget to activate the Attach tab in the Mobile App; we have named this element "Picture" in the structure. Without it, no attachment file will be uploaded to Transaction Manager.

The resulting form behaves as follows in the Mobile App (on a smartphone):





The form in the Mobile App, and the effect of the "Take Picture" buitton

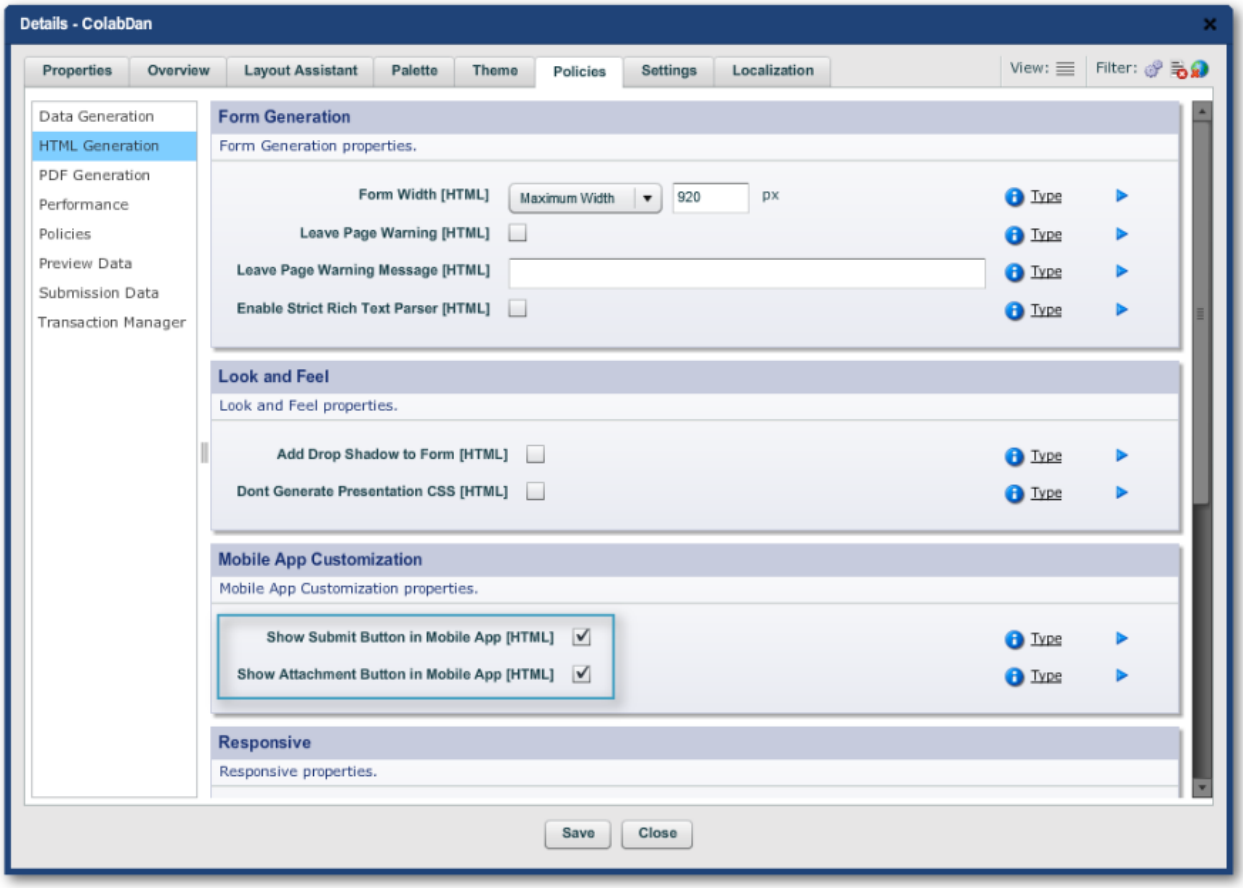
For simple forms like this example, and the user has taken only one photo, the user can go straight to the "Submit" tab and submit.

If the users have opted to "Use Photo" more than once, they have to go to the "Attach" tab and choose which image gets attached.

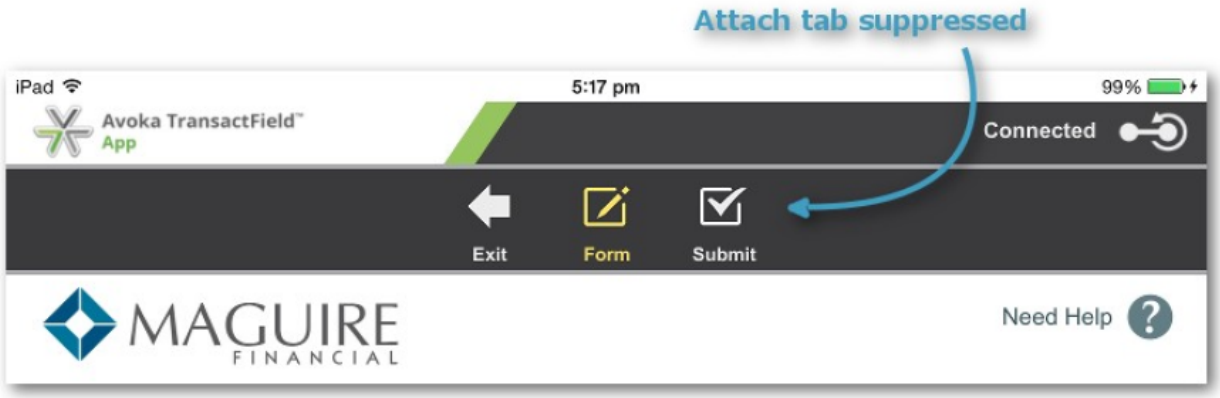
Suppressing the Attach and Submit Tabs

There are use cases where you will want to have attachments controlled on the form in the TransactField App, rather than in the app's tabs.

You can suppress either tab (or both) in Composer's Structure Tree: "<Form> -> Edit Properties -> Policies tab -> HTML Generation -> Mobile App Customization panel".

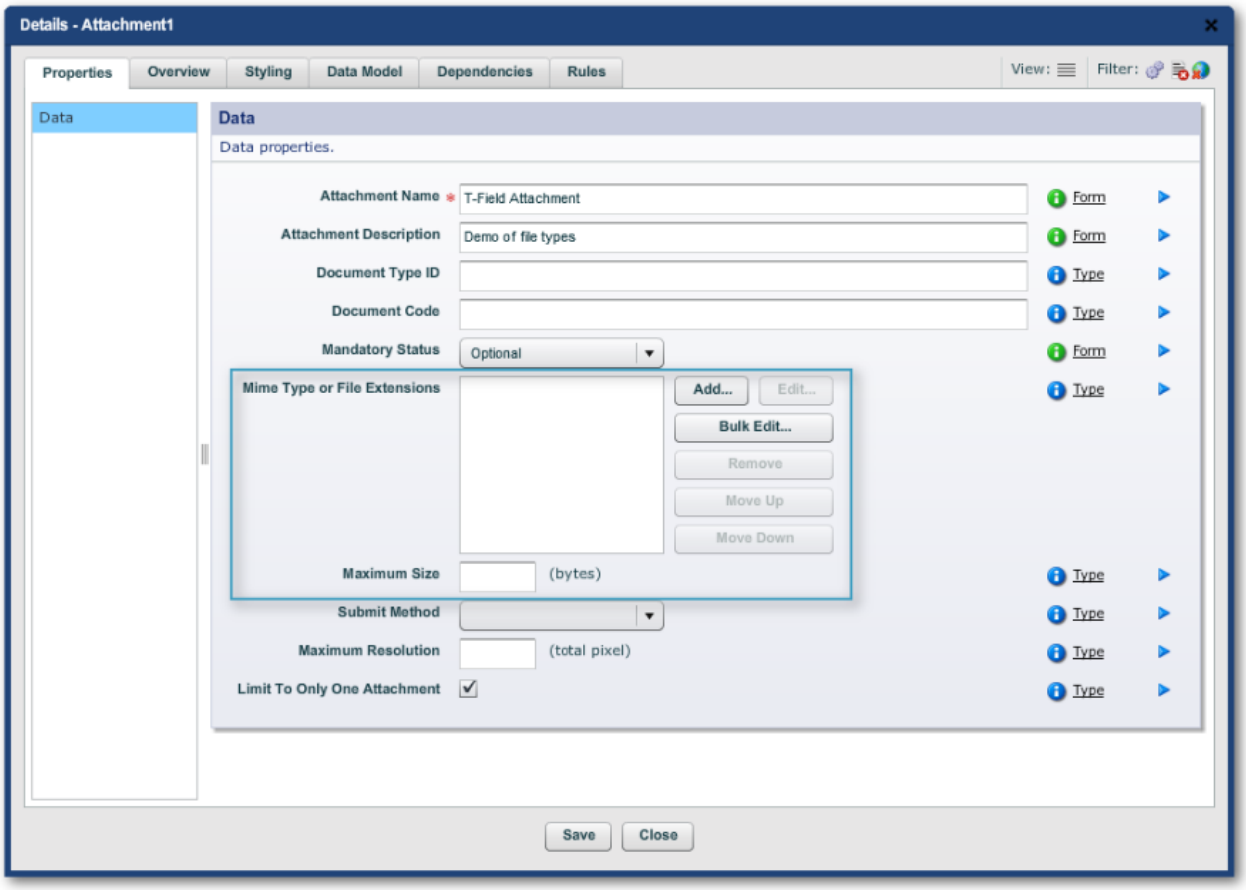


Here is an example of the Attach tab being hidden in the TransactField Mobile app:



Restricting Attachment File Types in the App

You now can control the file formats that the TransactField App can accept for an attachment or attachment. Place a TM Attchament Rule widget on the form and fill out the dialog:



Saving an attachment of the wrong type in the App results in a popup:



Multi-channel

Composer | Form Builder | v4.3 & Higher

Overview

The Avoka Transact gives you the tools to publish all your forms on desktop and on mobile devices, and yet be useable on all these different form factors. This is more than a convenience — it has become a cost of doing business.

It follows that users will want to move freely between their devices, without needing to complete a complex form on one device in the one session. Users now expect that not only can they resume a form later, but that they can do so on a different device or platform.

The Avoka TransactWeb Multi-Channel Tracking Technology enables end-users to start, pause, continue and complete their applications across channels – from their mobile device to their desktop, the call center, and the branches.

Here is a video of a presentation Avoka made at a finance industry forum. It illustrates the device agnosticism users will appreciate.

{+}http://youtu.be/nNY8MdPw1Qc+

Authenticated Users

With the Maguire Template

The Maguire template already has this technology embedded as a "Save and Close" button. On the desktop it is at the form header. On narrow mobile devices, it becomes an option in the slidermenu.

You can configure TM to prompt users to opt if they want to receive an email (to an address they nominate) when they "Save and Close". They are immediately sent an email giving a link to the portal and their unfinished form. Users click on the link, log into the portal (if required) and continues to fill in the form, which is prepopulated with their data.

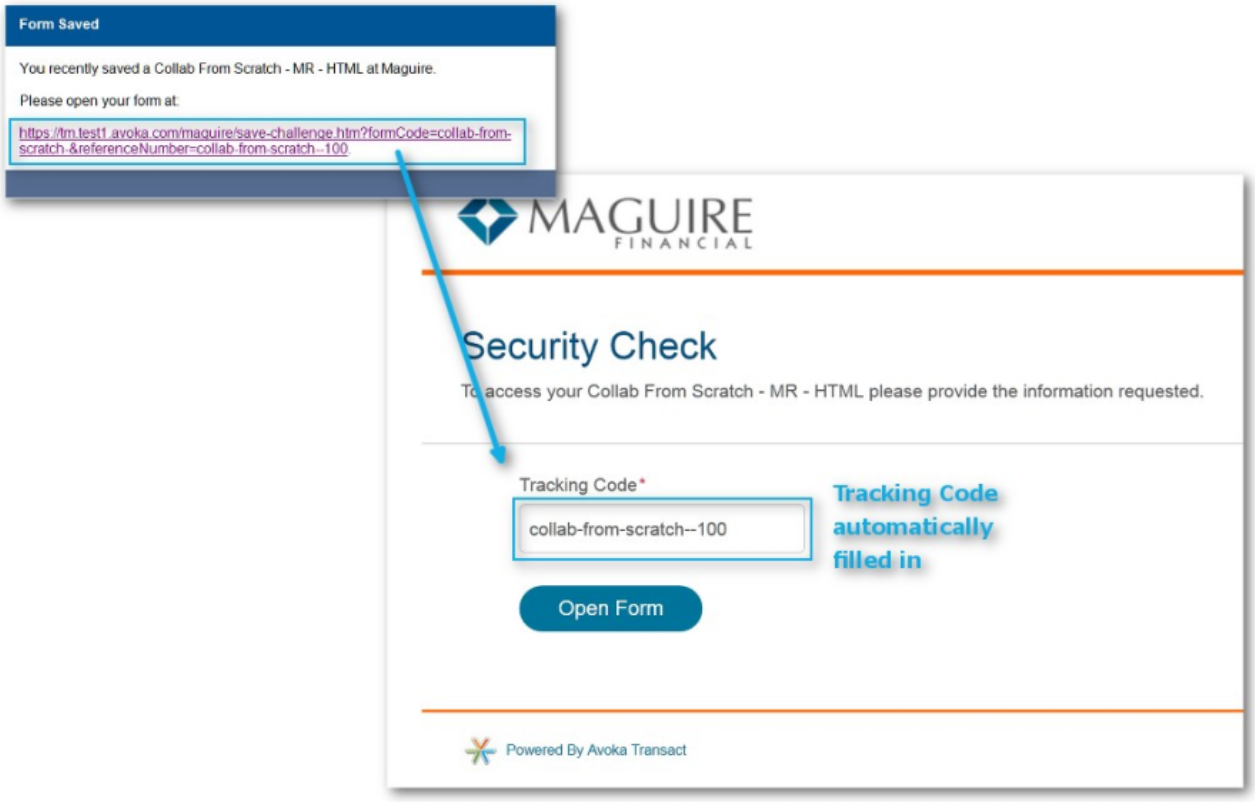
With Other Templates

To get enable multi-channel, drop the TM Submission Block onto the form. You can reword the text fields in the block to suit. You can even remove the "Save to my Computer" and text, as saving the form as a PDF is not useful in the context of multi-channel delivery.

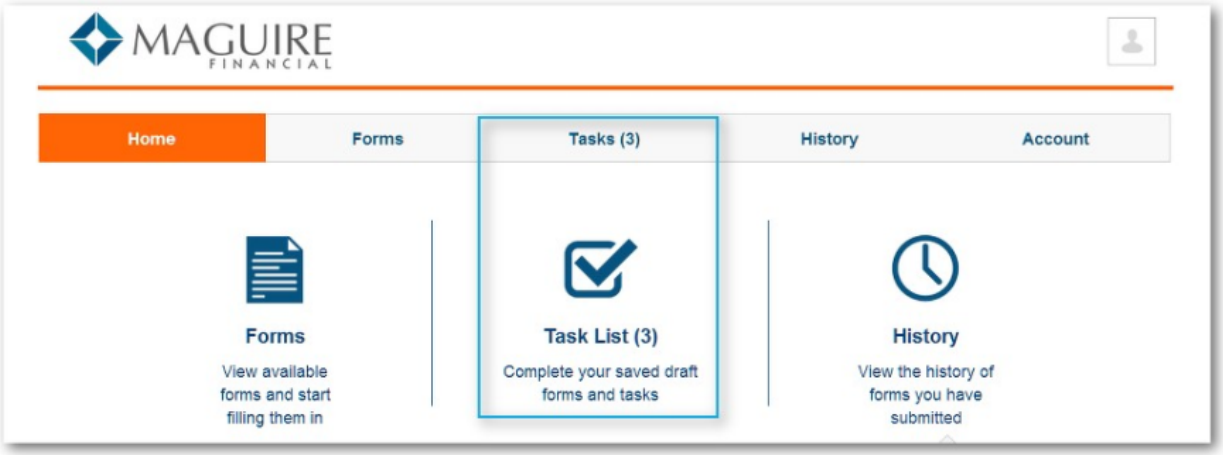
The Task List

So, users can resume their forms either by

Clicking on the link in the email. This takes users directly to the form, after first being challenged to supply ID and Password, and not to the portal's authenticated landing page.



Going to the portal's authenticated landing page and accessing the prepopulated form through the Task List.



TransactField App

These Save buttons in both the Maguire template and in the TM Submission Block, have special scripts so that they will work in the TransactField App. Putting the Save and Close buttons on the form will alter the App's Submit tab to accommodate the multi-channel save; the action will no longer be activated by the button on the form.

Of course, the button will appear on the form when the form is rendered on the device's browser (Safari in iOS and Chrome in both Android or iOS). On narrow devices, such as smartphones in portrait orientation, the Save and Close button will be in the Maguire slider menu.

Anonymous Users

Actually, Avoka Transact supports anonymous users, both for multi-channel and for collaboration.

The reasons you would want to do this are:

To reduce user friction

Some users, while happy to provide some personal details in order to request some service or good, are reluctant to create yet another transient user identity that will probably get used only the once

It is a necessary condition for Form Sharing, where several people have to fill in the one form, such as a pair of partners or another party is required to provide some documentation or statement before the form in question can be submitted.

Support for anonymous users is a configuration performed in TM. Please see FormSharing.

Form Sharing

Composer | Form Builder | v4.3 & Higher

Overview

Just as a form can be shared between the devices of a single user, see the Multi-channel topic above, so too can a form be shared between a number of different parties before it gets submitted.

The kinds of business cases this function supports are:

Sharing the form between applicants and their lawyers Between applicants and their partners

Between various parties filling in the same form where they are not in physical contact.

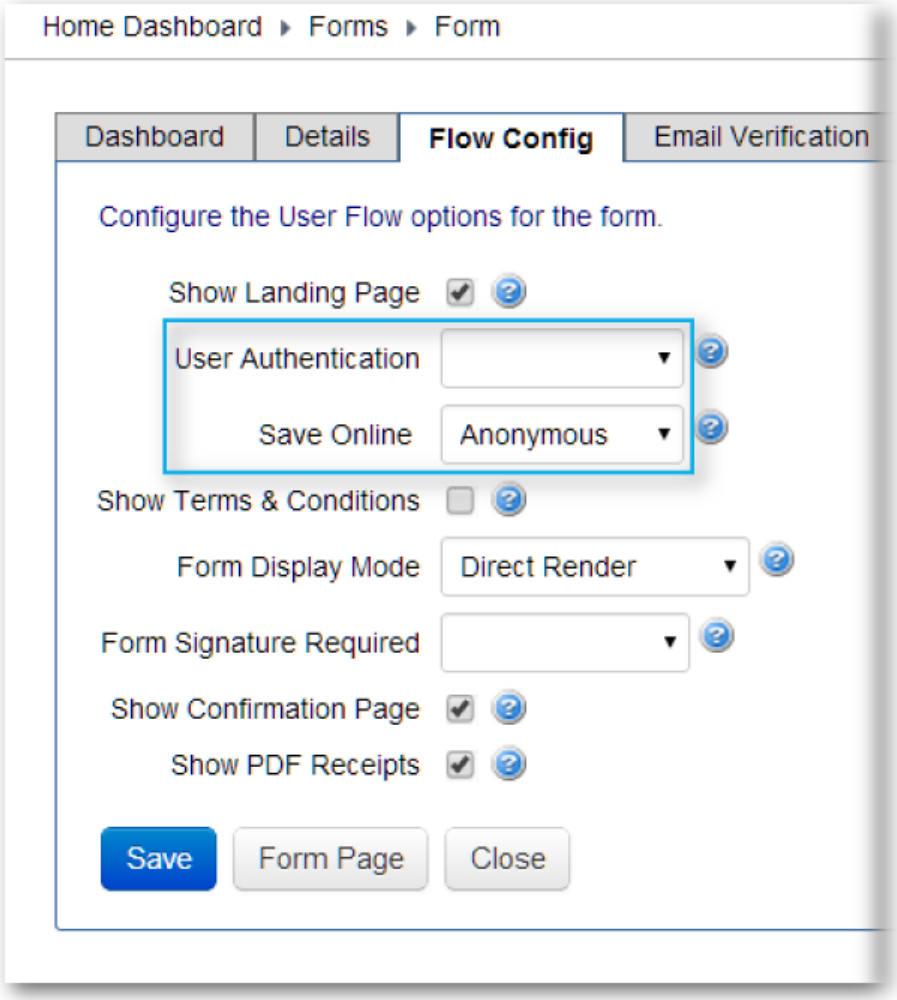
The sharing involves giving the name, email of the nominated other party, and a text area for notes to the other party; then the user, instead of submitting the form (as would be done of the form were complete), clicks on a dedicated button with some appropriate caption like "Activate Sharing". TM generates an email containing a link. The other party clicks on the link and is taken to the form (prefilled with the other user's data) which can then be edited, files attached and so forth.

The following notes on enabling form sharing assume that you are using the standard form sharing widgets that ship with Composer. We do not discuss configuring the form's behavior for, say, edibility or visibility of its fields. We also are using the default submit button, which — without any extra scripting — allows any of the parties to submit the form. After submission, the form sharing ends.

Minimum Prerequisites for Form Sharing

User Authentication Settings

The form's Authentication settings are made in Transaction Manager.

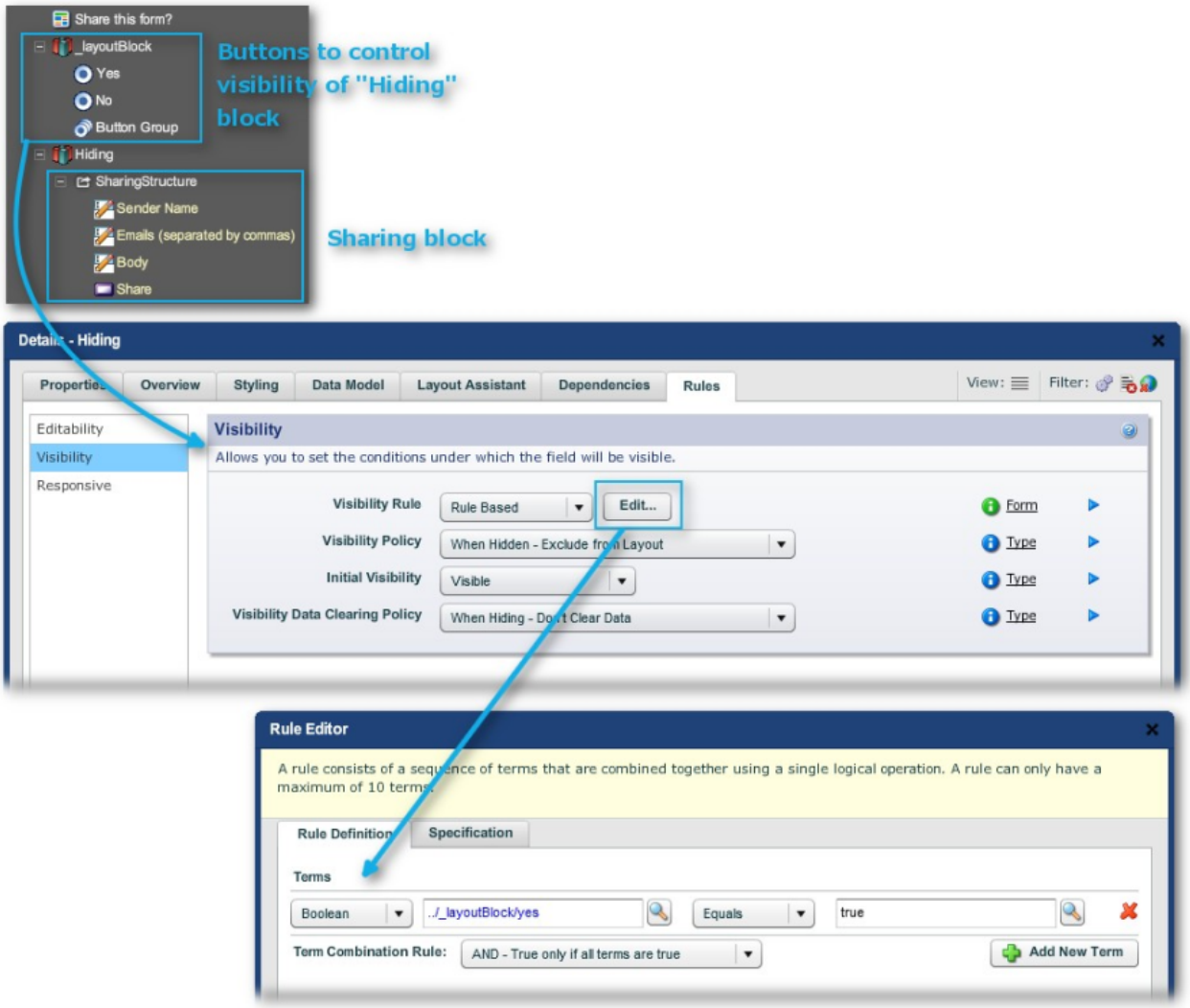


User Authentication has to be Anonymous (or unspecified at the very least). If not, the prefabricated TM Sharing block has scripts that make it invisible to authenticated users.

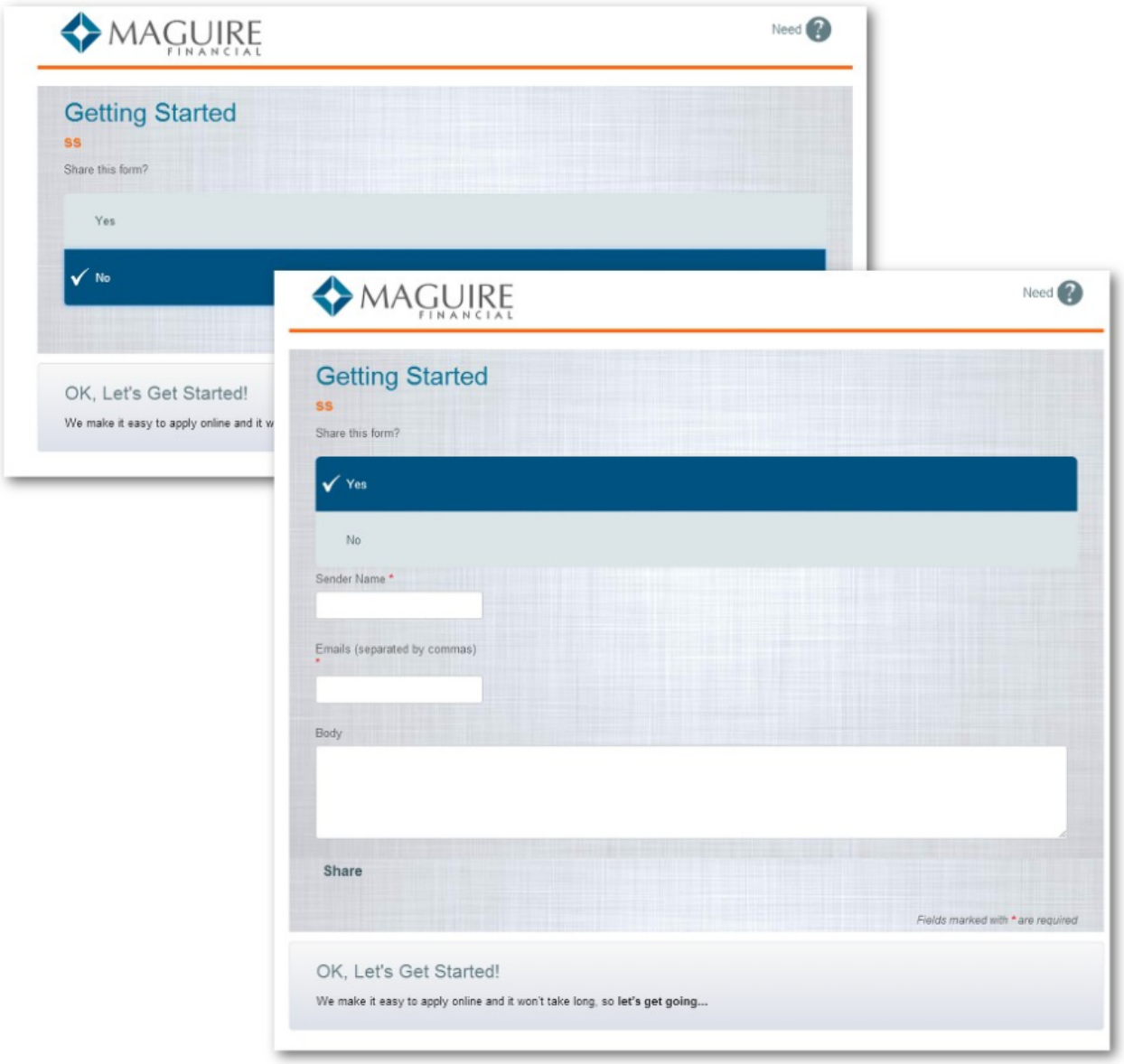
Form saving must also be anonymous so that the other parties can also save the form when they activate the share button.

There must be mandatory fields to capture the contact details of the original form user, otherwise there will be no connection to that person.

Simple Case using Prefabricated Blocks



Which results in the following behavior.

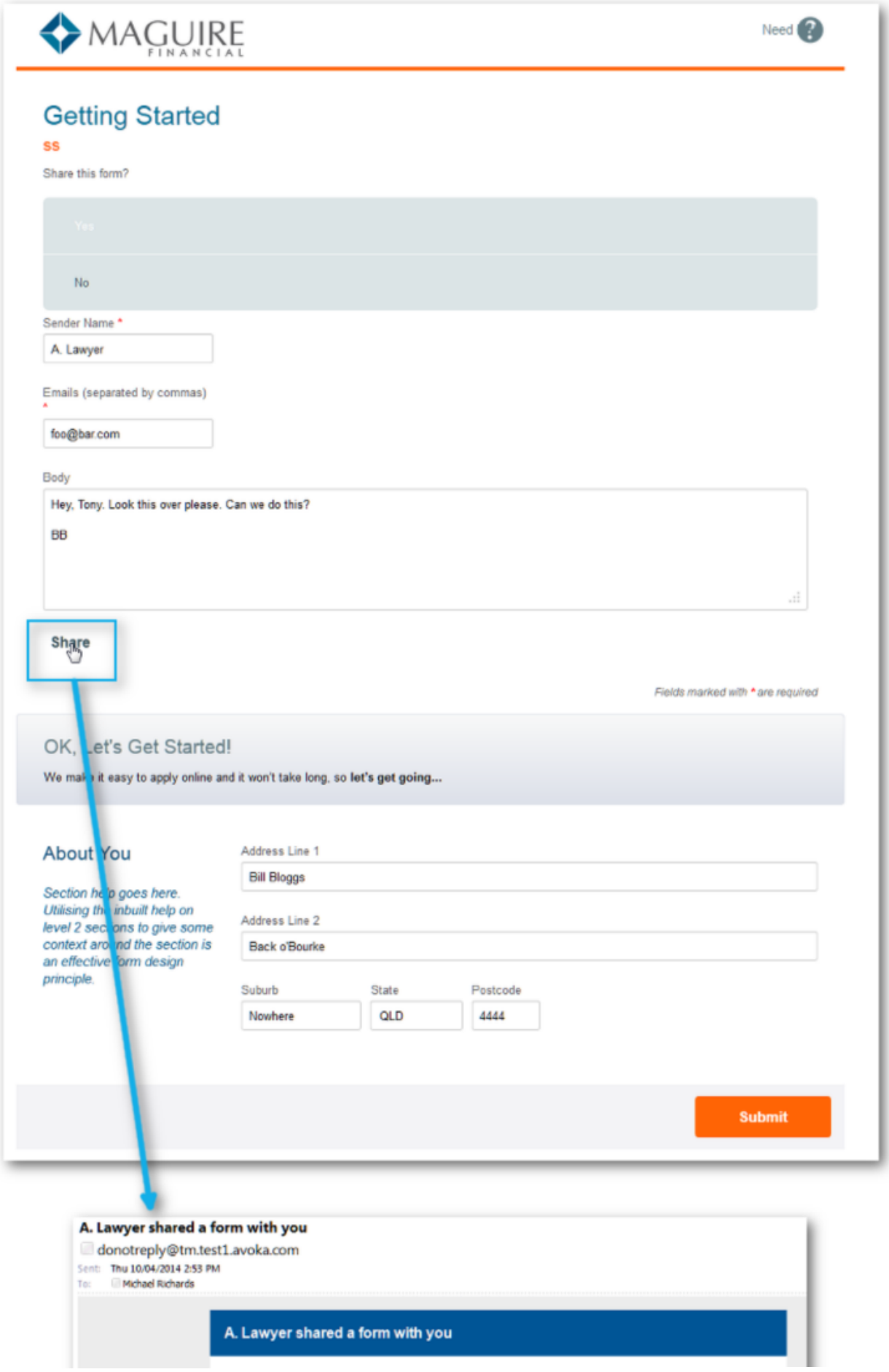


The first party opens the form anonymously (using, in TM terminology, the "form friendly URL"). The user is given 3 choices:

Open New Form Open Saved Form Login and Open Form

For this example, we are going to select the first option.

© 2015 Avoka Technologies Pty Ltd 142


MAGUIRE
FINANCIAL
Need
Getting Started
SS
Share this form?
Yes
No
Sender Name *
A. Lawyer
Emails (separated by commas)
*
foo@bar.com
Body
Hey, Tony. Look this over please. Can we do this?
BB
Share
Fields marked with * are required
OK, Let's Get Started!
We make it easy to apply online and it won't take long, so let's get going...
About You
Section help goes here. Utilising the inbuilt help on level 2 sections to give some context around the section is an effective form design principle.
Address Line 1
Bill Bloggs
Address Line 2
Back o'Bourke
Suburb
Nowhere
State
QLD
Postcode
4444
Submit
A. Lawyer shared a form with you
donotreply@tm.test1.avoka.com
Sent: Thu 10/04/2014 2:53 PM
To: Michael Richards
A. Lawyer shared a form with you
Please open the form at:

The second party can then open the form and share it back or submit it.

In TM, you can view the history of the form (when opened, when shared and when submitted) and the contents of the form (in XML).

Note: when you are testing form sharing (using a form served by Transaction Manager, not using Composer's Preview), all of the users cannot be logged into any of the portals. To test form sharing, you may have to initiate a new browser session, one in which you have not logged onto a portal. You can get round this be starting another browser altogether; you can already be logged into the portal in Chrome and test form sharing in a fresh session of Firefox.

Why Hide the Sharing Block?

This prefabricated block is mandatory. The form will not submit if the block is visible, unless the user enters some random data into the fields displaying in-line validation messages.

Sharing in Collaborations

This is possible to do. However, the user authentication of the first step needs to be anonymous. Also the link back to this user is through email alone and the service definition object will need to be modified. See Collaboration on how this feature works in general. Currently the documentation does not include what to do to be able to share the form before the first submission.

Composer Form Parts

Composer | Form Builder | v4.3 & Higher

Form Parts projects introduce a special type of form known as a part-form. Within a part-form, one or more parts can be created. Inside the part, a form developer can create any content, including entire section level 1's. The parts can then be exported from within the part-form to the organization.

Overview

In projects where there are many forms and/or larger and more complex forms, two problems can arise:

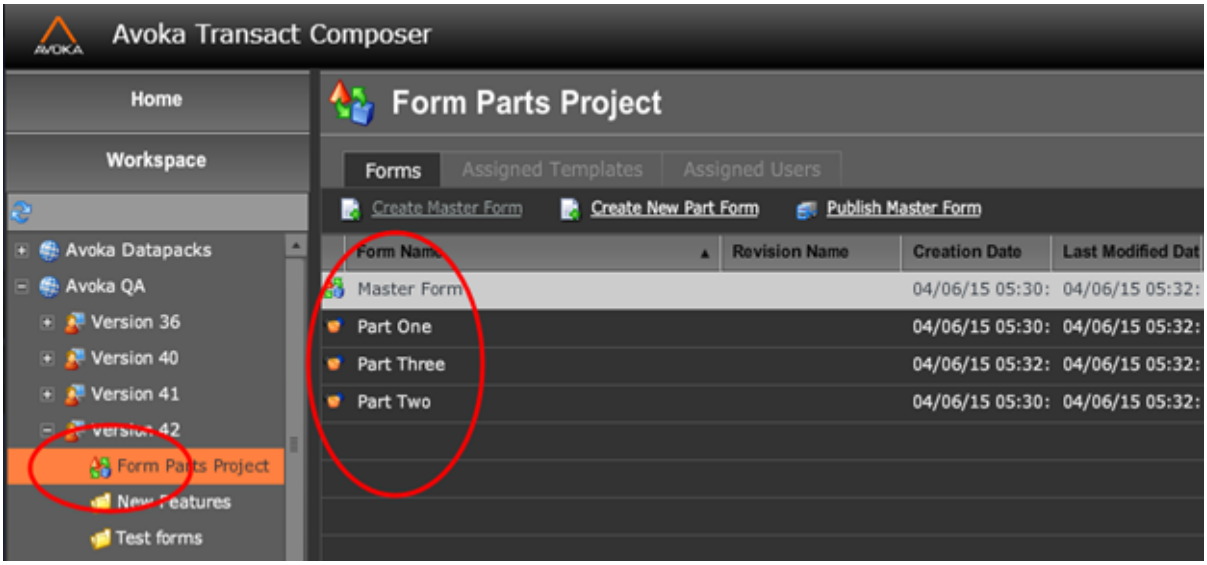
- Composer does not allow two users to work on the same form at the same time.
- Often forms contain large re-usable sections, such as a "Personal Details" section. Blocks can be used to create these sections, however blocks are generally designed for smaller entities, such as an address block. Composer isn't optimized for creating larger blocks, especially those containing sections. The block editor is an advanced tool which is intended for power users, and isn't suitable for editing these types of larger blocks. Instead, it would be preferable to use the regular Composer editing environment for creating these larger blocks.

Form Parts solves both of these problems.

Form Parts projects introduce a special type of form known as a part-form. Within a part-form, one or more parts can be created. Inside the part, a form developer can create any content, including entire section level 1's. The parts can then be exported from within the part-form to the organization.

Another special type of form unique to Form Parts Projects is known as a master form. It is used to include any of the exported parts that have been created in the part-forms. A master form can override properties from the parts if necessary. Business rules can be created either within each part, or in the master form spanning multiple parts.

These new form types are shown below:

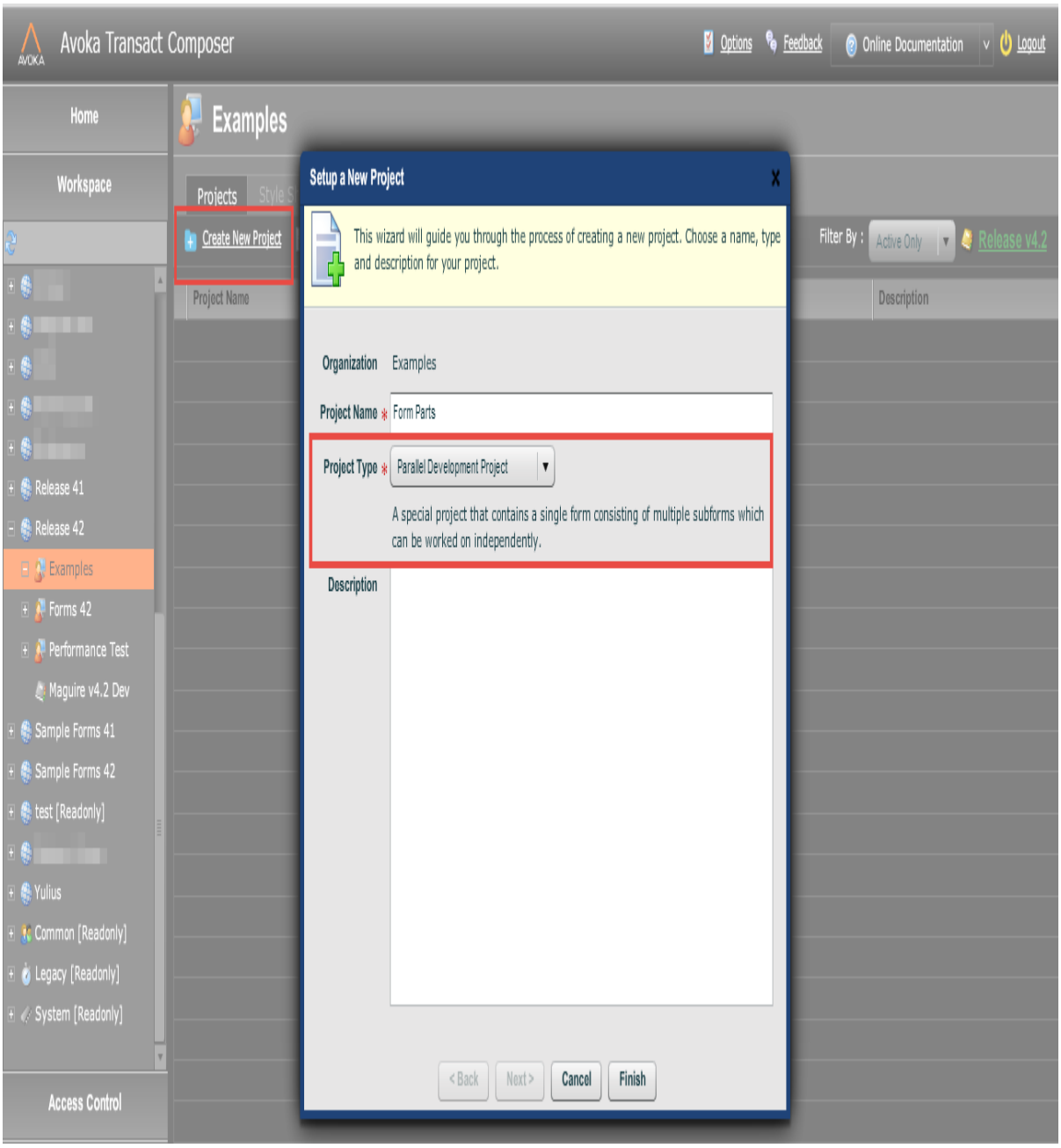


This allows multiple developers to work on different parts of a single form in a controlled and convenient way. It also allows re-usable parts to be created that can be used in multiple master forms.

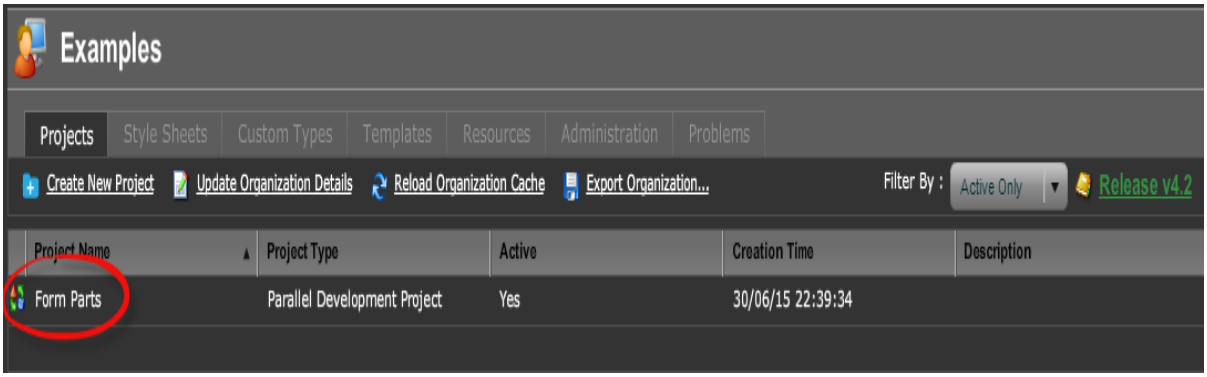
Setting Up a Form Parts Project

Form Parts projects (also known as Parallel Development projects) behave differently to normal Composer projects. For this reason they are distinguished from normal projects with different icons.

Creation of a Form Parts Project is done at the Organization Level by selecting "Create New Project". In this dialog the Project Type dropdown "Parallel Development Project" must be selected.



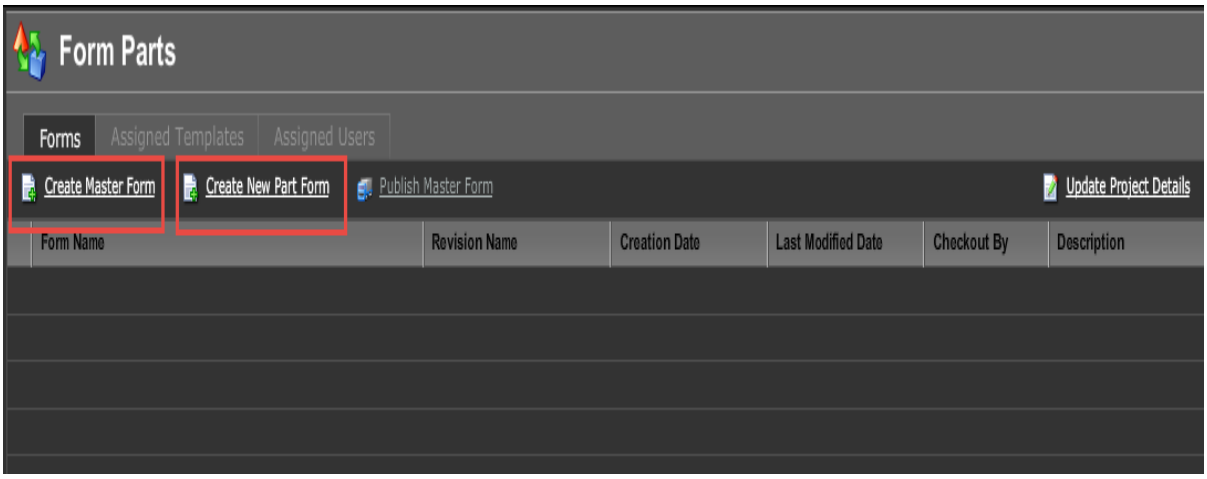
This creates a Form Parts project



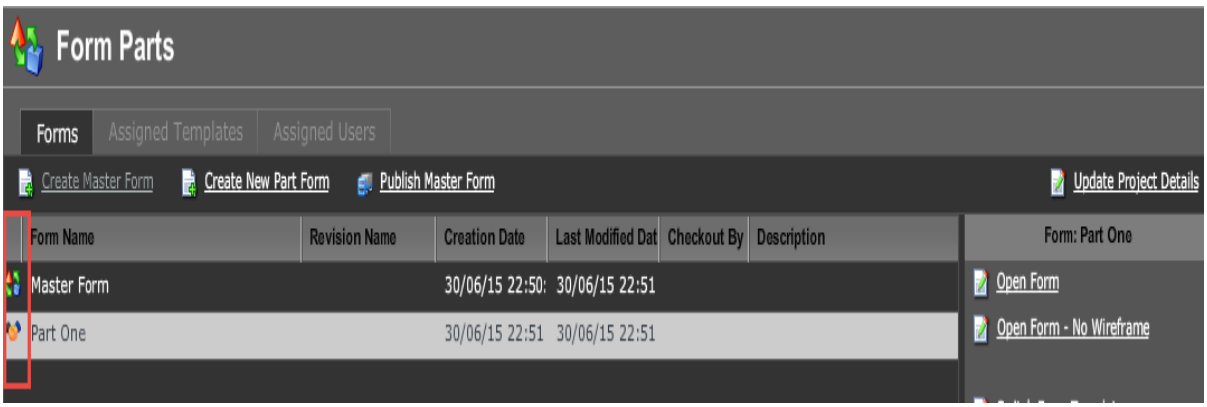
Creating Master Forms and Part-Forms

At the project level you can create the part-forms and master forms by selecting the options.This process is much the same as creating an ordinary Composer form.

Only one master form can exist in a form part project. Multiple part forms are permitted.

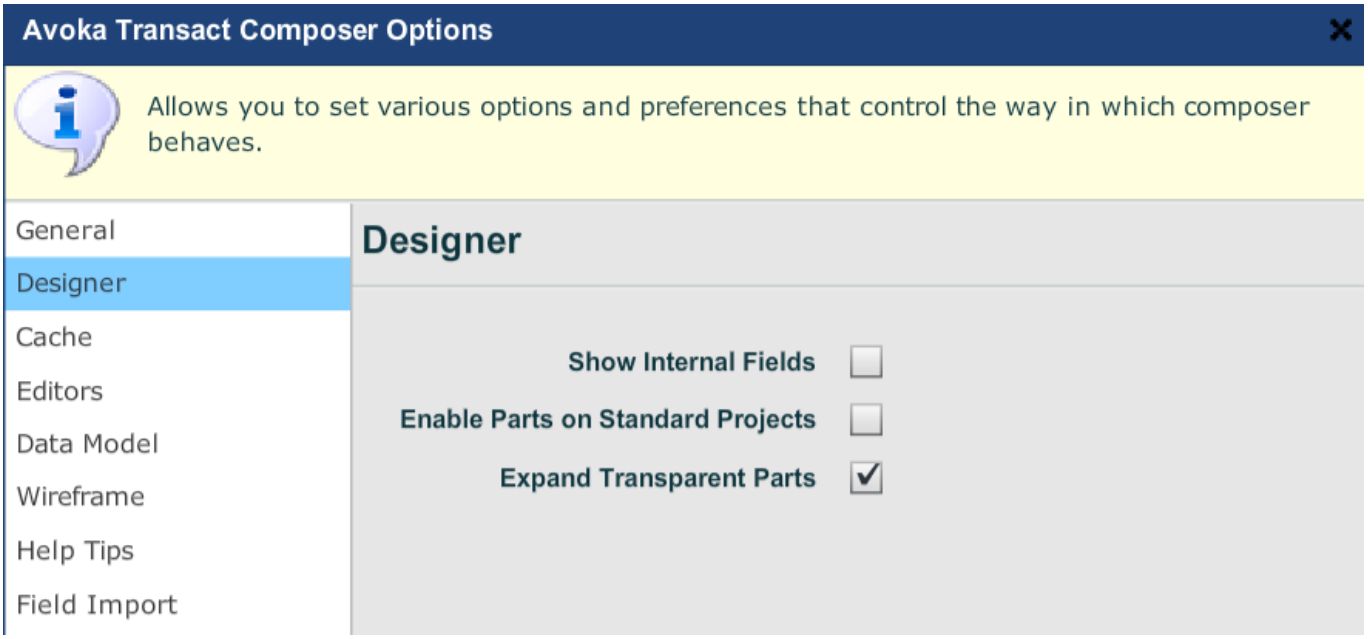


The different form types are distinguished by their icons. The Create Master Form option is now disabled in the screenshot below.



There is nothing special about a Parallel Development project, or the master form or the part forms. These are really just regular project/form, which are tagged in a special way to give them different icons in the Composer user interface. They also enable some special toolbar icons in the project menu, such as Create Master Form and Create New Part Form. You can convert freely between a Parallel Development project and the Standard type of project. You can promote a regular form to a Master form using the right hand icon menu at the project level (but you can only have one master form per Parallel Development project).

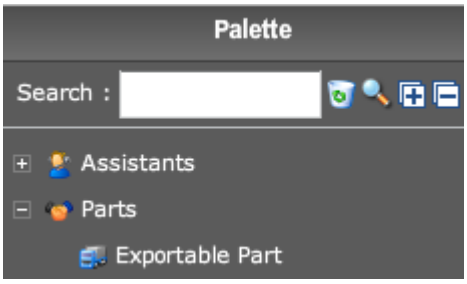
If you wish to use the special parts menu options in a regular project, then you may do so by checking "Enable Parts on Standard Projects" in the Composer Options dialog.



Generally the master form and the parts will all be in a single project. However, a master form in project B can also use parts that are defined in project A. This approach might be useful for the case where multiple forms utilize the same set of common sections, and these sections are defined in a separate common project.

Creating Parts within the Part Forms

The part forms act as containers for the parts. In order to create a part, drag a Exportable Part from the Parts folder in the palette into the top level of your form.

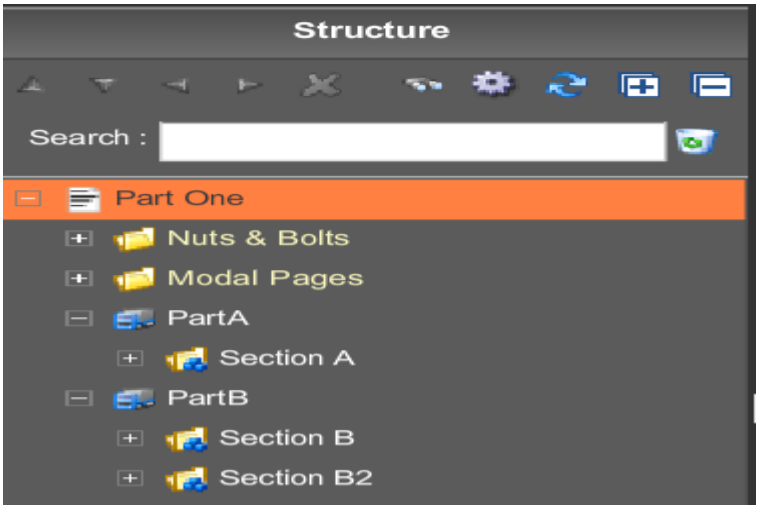


You can have as many parts as you want within each part form.

Parts can be either transparent or opaque. This will be covered later.

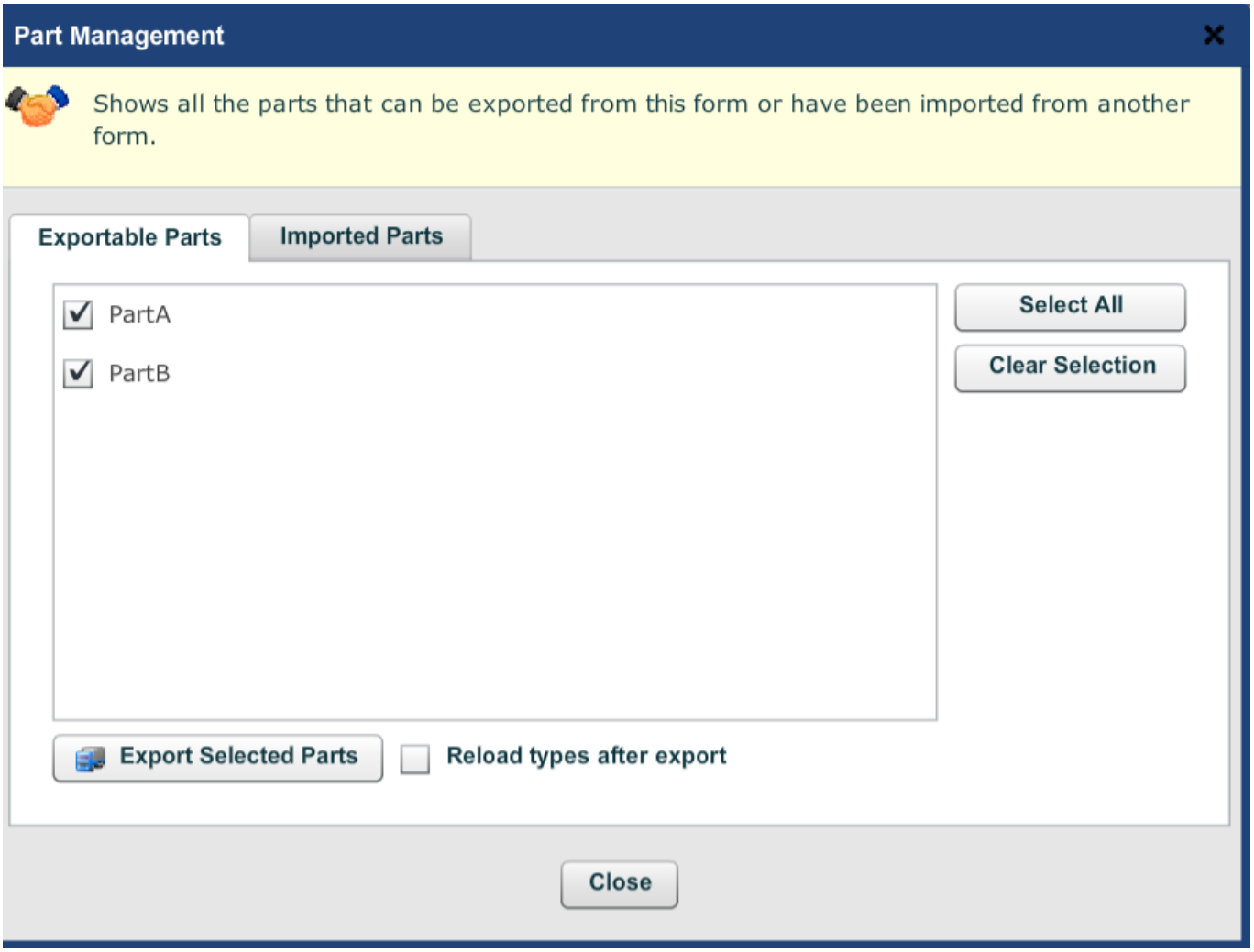
You can then add any content you wish into your Part. In most cases, you will drag a Section Level 1.

The example below shows a Part One form that contains two parts, Part A and Part B. Usually a part form will only contain one part, and a each part will contain one section, but this example illustrates that a part form can contain multiple parts, and a part can contain multiple sections if desired. The sections can contain arbitrary content, including fields, blocks and business rules. However, if your form contains multiple parts, you must not create any dependencies or business rules between the parts. Each of the parts can be developed, and tested in isolation, as it it was a normal form.



In general, it is recommended that each part form contain only one part, and that each part contains only one section. It's also usually a good idea to name parts and part forms (and even the sections they contain) using a consistent naming scheme, so that the parts and sections can easily be located.

In order to use these parts within a master form, they must first be exported from the part form. In order to export one or more parts, click on the Parts... icon which is visible in the toolbar of all part forms. This will bring up the Part Management dialog shown below.



Your parts are not available to be used in a master form until they have been exported. Click Export Selected Parts in order to export the selected parts. It is not usually necessary to select "Reload types after export" - use this checkbox if you want to verify that the parts have been exported by viewing them in the component palette.

After exporting, the selected parts will appear on the component palette on the right under the Parts node, with a sub-node which is the name of the project, as shown below:



If you use parts extensively, this can result in many projects and parts under the Parts node, which can become confusing. This can also result in slower time to load the types for the organization. If you use a large numberof parts, you should either use a project and part form naming convention that allows you to easily identify the various parts, or you should create multiple organizations to group the various parts and the forms that use them.

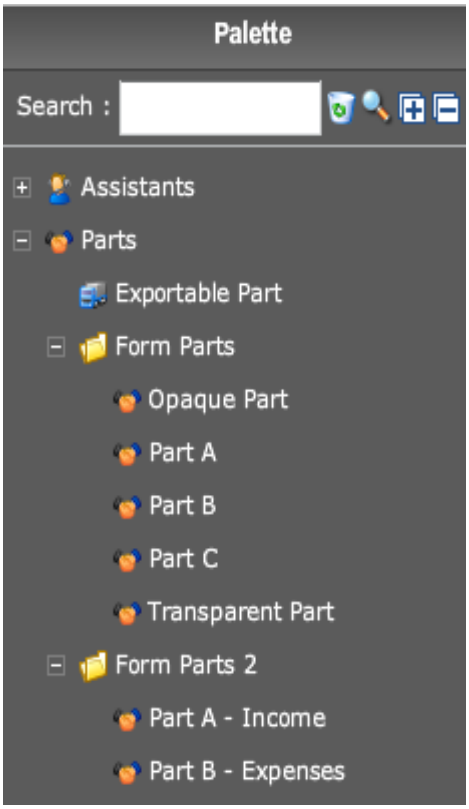
An exported part is actually just a special type of block, similar to a block created using "Save as Custom Block" or the block editor. It is really just the way that the block is created, edited and exported that is different.

Saving the form does not automatically export the parts. And exporting the parts does not automatically save the form. This is important so that you can control when and how you update your exported parts. Each of these operations must be performed separately. This is useful because it allows you to finely control when you make your part available to others. For example, you wouldn't want to export your part if it has a bug, or if you're not finished developing it.

Be careful of renaming a part - the part will be exported under the new name, but the original part will be retained - this may cause confusion. You can manage the parts as in the Custom Types tab at the organization level.

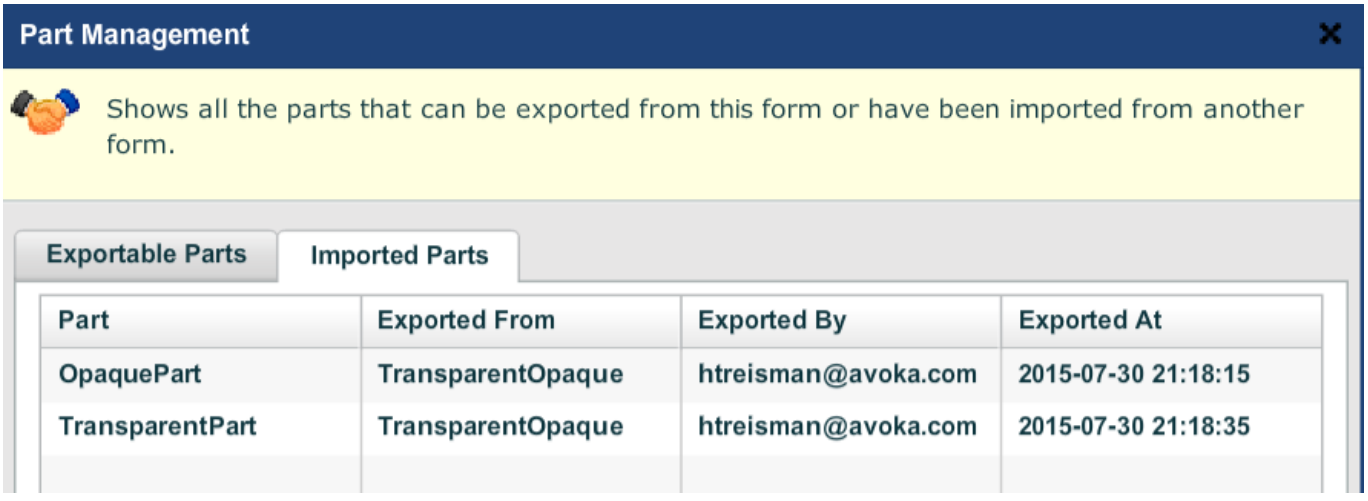
Combining the Parts to Create a Master Form

In the master form, simply drag and drop the various parts that you require from the Parts folder in the palette into your master form. Because parts are generally section level 1's, you should drag the parts into the top level of the form. For parts that contain other types of content, drag them into the appropriate locations.



Parts are grouped by the name of the Project in which they are defined. You can drag as many parts as you want into your master form.

Once you have imported parts into the master form, you may view a summary of what parts have been included by clicking on the Parts,, button and selecting the Imported Parts tab. This tells you not only what parts are being used, but also which part form they are defined in, and when/who exported them.



Important: it will be important later on when we discuss cross-part business rules that all parts have the same name everywhere. The easiest way to ensure this is to accept the default name for each part when you add it to the master form.

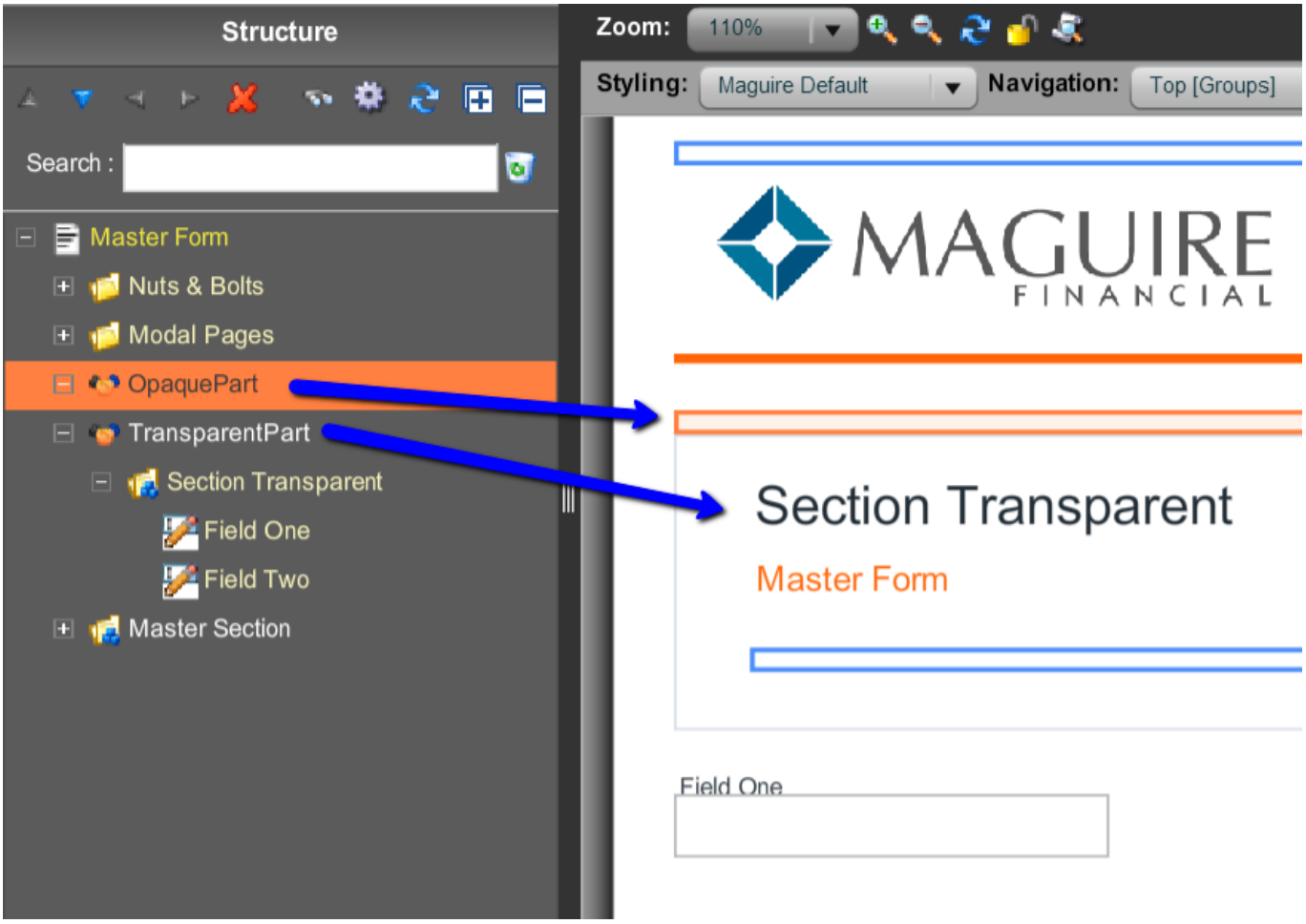
Opaque and Transparent Parts

A transparent part is the default type of part. When imported into a master form, it looks just like a normal section of the form.

An opaque part is treated as a "black box", and is not actually loaded into the Composer designer. An opaque part's contents will not appear within the structure tree or in the wireframe. However, the contents of an opaque part will appear in the form itself, when previewed or published.

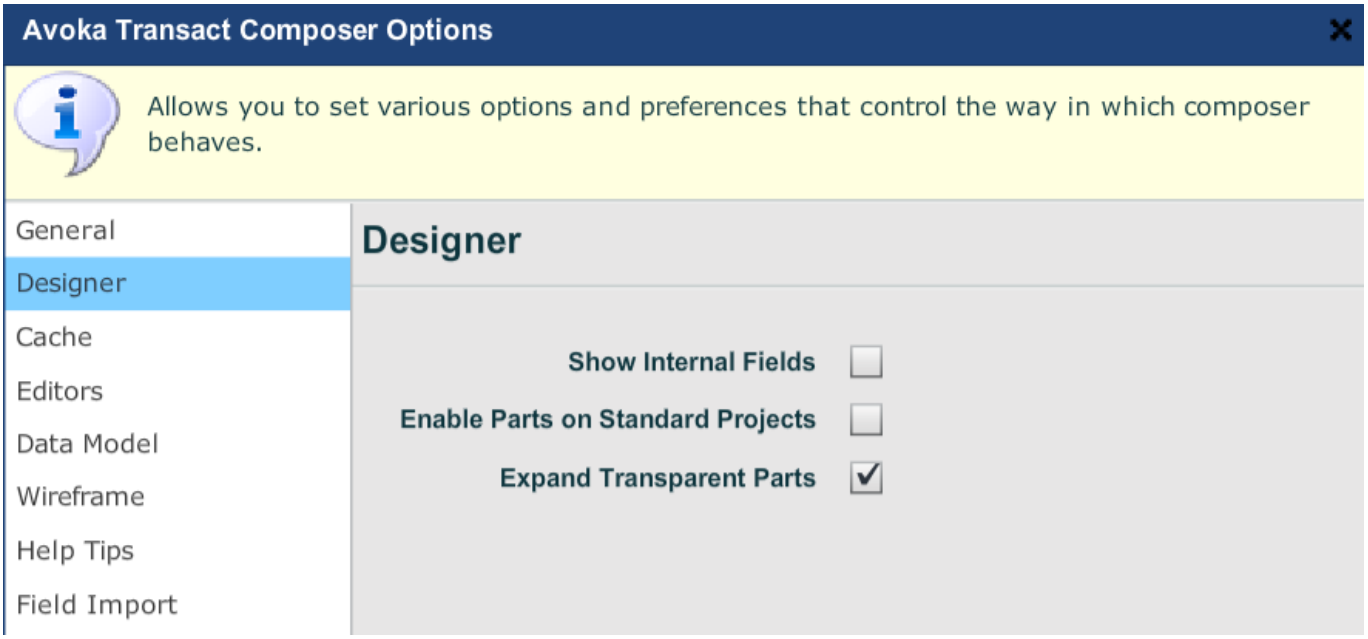
There is no real difference internally between a transparent and opaque part - it is only the Composer editor that treats them differently from a visibility perspective. You can modify whether a part is transparent or opaque in its property dialog.

The difference between an opaque and transparent part when imported into a master form is shown in the picture below.



You can mark your parts as opaque in order to indicate your intention that your part should be used "as-is", and not modified. An opaque part will also mean that the form using the opaque part will load quicker.

By default, transparent parts will not have their contents displayed in the structure tree. This serves as a reminder that the contents of the part are not intended to be modified. In order to see the contents of transparent parts in the structure tree, you need to select "Expand Transparent Parts" in the Composer Options dialog.

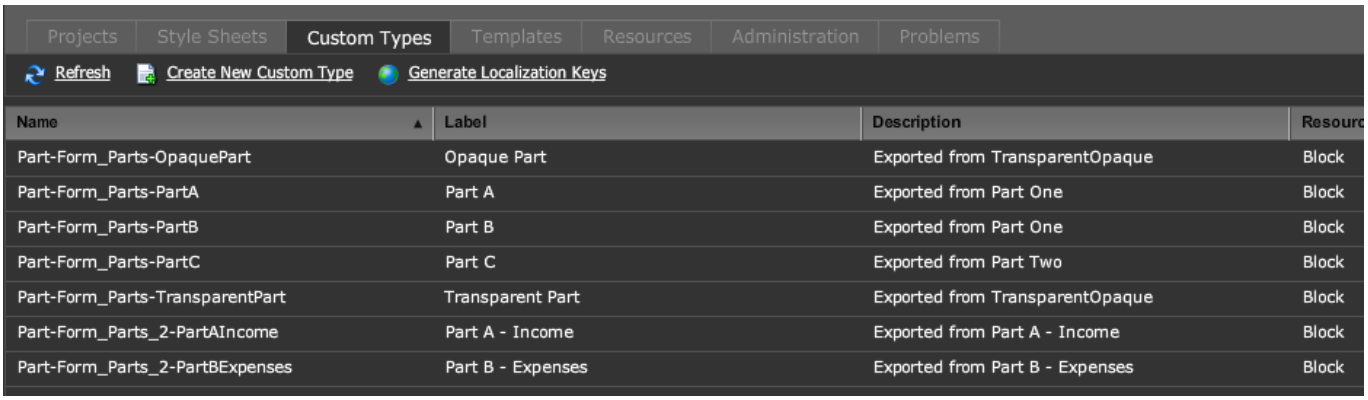


Even if the "Expand Transparent Parts" option is unchecked, the fields in a Transparent part are always available in the rules editor, and so may be used within rules.

Parts and Versioning

Part Forms (the forms that contain the parts) are like any other type of form, and benefit from the usual Composer form versioning capabilities. However, the parts themselves, which are exported from these part forms, and which exist within the component palette, only have a single version - this being the latest version that was exported.

Under the covers, parts are exported as regular Composer blocks, and can be viewed within the Custom Types tab at the Organization level.



Name	Label	Description	Resourc
Part-Form_Parts-OpaquePart	Opaque Part	Exported from TransparentOpaque	Block
Part-Form_Parts-PartA	Part A	Exported from Part One	Block
Part-Form_Parts-PartB	Part B	Exported from Part One	Block
Part-Form_Parts-PartC	Part C	Exported from Part Two	Block
Part-Form_Parts-TransparentPart	Transparent Part	Exported from TransparentOpaque	Block
Part-Form_Parts_2-PartAIncome	Part A - Income	Exported from Part A - Income	Block
Part-Form_Parts_2-PartBExpenses	Part B - Expenses	Exported from Part B - Expenses	Block

If you wish to revert to a previous version of a part, you must open the version of the part form that contains it, and re-export.

Cross-part rules

When business rules exist entirely within a part, then they are very easy to set up - simply define the rules within each part as normal, and the rules will be imported along with the part into the master form. However, when it is necessary to define rules at the master form level that span multiple parts, then things can get a little more complicated.

There are several different approaches to dealing with cross-part rules. The approach you choose will depend on the sophistication of the form and the business rules, as well as the sophistication of your development team. Three different approaches are discussed below. There are probably other approaches that can be used too, but these are the recommended approaches.

All of these approaches require the development teams to use discipline in the way that they use the parts capabilities. It is possible for a single project to mix and match the various techniques listed below - but this can become confusing, and it recommended that the form developers on a project consciously select a particular approach and stick to it.

Simple - All Cross-part Rules in the Master Form

All the cross-part rules are defined in the master form, as if it were just a single large form. Rules can be set up that include fields in any of the parts, and/or any of the fields that are explicitly placed in the master form. This has the advantage that it is very simple to use and understand.

However, it suffers from some disadvantages:

- All the responsibilities and dependencies are on the master form developer. If there are a large number of cross-part rules, this limits the ability for multiple developers to be able to work independently on these rules.
- When any of the parts are changed, the master form developer needs to fix everything in the master form. In some cases, it might be preferable for each part developer to manage their own cross-part rules.

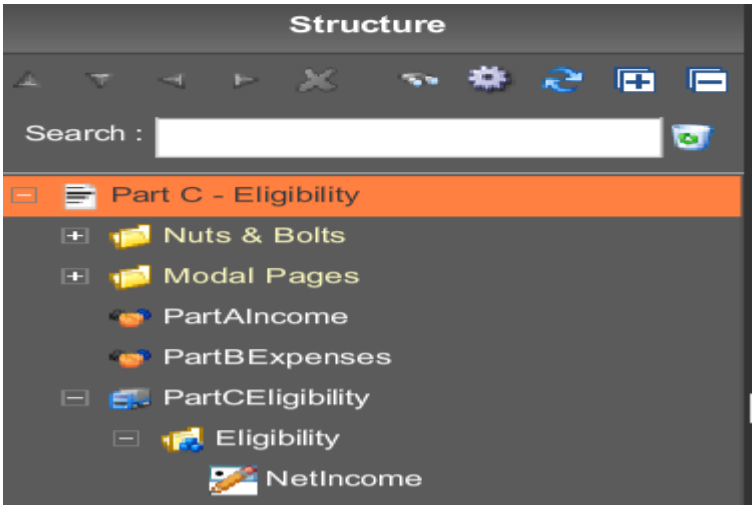
Normal - Each Part References Other Parts it Requires.

This is a slightly more sophisticated approach, and encourages each part developer to define and maintain the rules in their own parts independently.

Let's consider the following example:

- Part A - Income Section. This section has a numeric field named TotalIncome.
- Part B - Expenses Section. This section has a numeric field named TotalExpenses.
- Part C - Eligibility Section. This section contains a calculation for a field NetIncome = TotalIncome-TotalExpenses, which is in turn used to determine whether a customer is eligible for a loan.

The developers of Parts A and B develop their parts as usual. The developer of Part C, because there is a dependency on fields outside of his part, will add these parts into the part form. The Structure tree for Part C is shown below:



Note that:

- The above diagram is the structure for Part C, NOT for the master form. The master form will include parts A, B and C, but will not directly contain any business rules.
- Part C is defined and modified within this part form. But parts A and B are defined elsewhere, and simply imported so that they can be referenced.

The developer of Part C will include the calculation rule into Part C. When Part C is exported, it will contain a business rule that references Parts A and B.

When Parts A, B and C (and any other additional parts) are included in the master form, the business rule that exists in part C will reference fields in other parts. However, since these parts also exist within the master form (as imported parts), the references will be correctly resolved, and the business rule will work correctly. (This is why it was stated earlier that each part should always be given the same name wherever it it used.)

Tips:

- Within the Part C form, only add cross-part business rules to Part C (not Parts A or B). Because we are only exporting Part C, any business rules you add to parts A or B will never be exported.
- When editing a part form, use Transparent parts, but uncheck "Expand Transparent Parts" in the Composer Options screen. This will display the imported parts without any contents, as shown above. This will serve as a reminder to not add any business rules to these parts. The business rule editors will still allow you to create rules in Part C that include field from Parts A and B.
- You can preview and test your form as if it were a complete form.
- You can set the wireframe to only display Part C, using the Wireframe Options button. That way your form will load faster, and also you won't be tempted to accidentally add rules to parts that aren't defined in this form.
- If one of the other editors makes a change to part A or B, you may need to adjust rule business rules. However, you can usually detect this using the Problems pane.
- When working in a single form, Composer will automatically refactor your business rules if you move or rename a dependent field. However, this will not be done when the dependent field is modified in a different part. It is therefore advisable to either limit the number of changes made, or to communicate well between developers when these changes are made. The Problems pane will notify you of any errors.
- Generally when using this approach, the master form just contains a series of parts, and has no content or business rules of its own. It is possible to mix the approaches, but this is not recommended, as it can get very confusing.

This approach does have a number of disadvantages in certain situations:

- Part C has a dependency on Parts A and B. It will fail if used in a form that does not have Parts A and B, and cannot be used independently of them.
- This approach may not work very well when a particular form has business rules that reference a large number of different parts - you may be better off using the Simple approach in this case.

Despite these minor disadvantages, this is the approach that is generally recommended.

Advanced - Model/View/Controller

This is an advanced approach to cross-part rules, using a well-known pattern from software engineering, the Model/View/Controller pattern. In this approach, rather than defining business rules directly between the fields of one part and those of another, an intermediate data model is constructed, which serves as the communication mechanism between the parts. In summary:

- Model = a special "data model" part containing only data fields. (A data field is a special invisible field available in the palette.)
- View = the regular parts containing sections.
- Controller = the cross-part business rules (these can be defined in a number of places, as described below).

This approach is a little more complicated than the preceding ones, and requires some discipline and rigor in the development process. However, it has the advantages of:

- An explicit definition of what data is shared between the parts.
- A software "contract" about how this data is shared and used.
- There are no direct dependencies between any part and any other part - there is only a dependency between each part and the shared data model. This means that each of the part forms only need to include the model part, and nothing else.

When creating the model, there are some interesting choices about what data elements to create in the model, and where to place the rules (or controller). For example, if we had a simple calculation of "C = A + B", where A, B and C are each in a different part, then we could have any of the following:

- A and B in the model. C exists only as a real field, and contains the calculation.
- A, B and C in the model. The business rule exists on data element C within the model. The real field C has a calculation equaling the data field C.
- Other choices are possible.

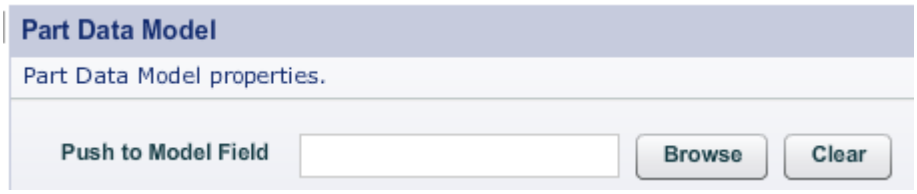
One simple approach is to only include in the model those fields that are needed for business rules in other parts, and only to include business rules in the parts. However, your project team may choose a different approach.

Once the model has been created, it is imported into each of the part forms, and then two steps need to be performed:

1. Create any business rules that are dependent on data fields in the model.
2. "Push" the values of any fields that need to update the values in the model. (It is up to the developers to ensure that each field in the model is only updated by a single real field.)

Note that while it is theoretically possible for the model to "pull" the values it needs from the various parts, this breaks one of the fundamental principles of model/view/controller, in that the model has no dependencies on any of the views. Views can reference the model, but the model should not reference the views.

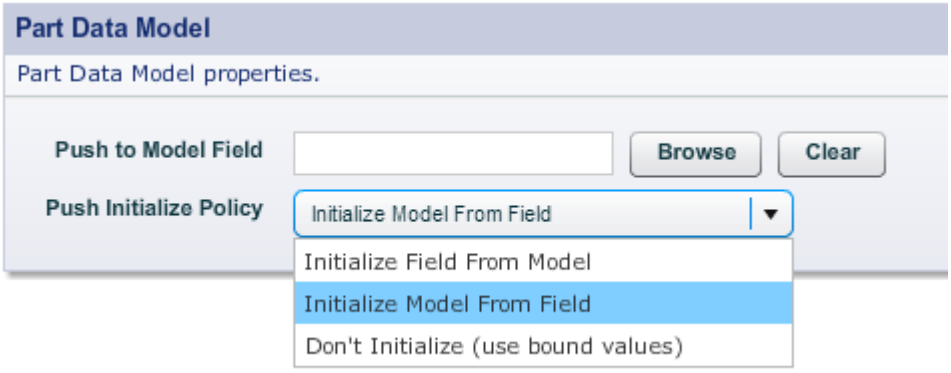
In order to do this, you can use the "Push to Model Field" property of the Part Data Model section on the Properties/Data tab of the properties dialog.



Another interesting decision occurs because you now have two fields (a data field in the model, and a real field in the view), both of which represent the same physical data element. The question becomes: How do we bind these two fields to the XML, and initialize each of the values? Remember that when a Composer form loads, each field will first have its value initialized from the XML data it is bound to, and then any business rules will be fired. The choices are as follows:

1. Bind the real field to initialize it. Initialize the model field from the real field.
2. Bind the model field to initialize it. Initialize the real field from the model field.
3. Bind both fields. No additional initialization is necessary, because they are both bound, and will both be set from the binding values in the XML. This can be a little dangerous, because you are responsible for ensuring that the XML contains the same value in both locations, which in turn ensures that both fields are initialized to the same value. If the XML is inconsistent, you will get strange behavior, with rules not appearing to reflect the data shown. Alternately, you could selected either "Initialize Field from Model" or "Initialize Model from Field", and even though both are bound, as soon as the form loads, the second field will be set to the value of the first one.

You can control which of these approaches using the "Push Initialize Policy" property, as shown below.



The decision outlined above actually occurs any time that you have two different fields that represent the same piece of data, not just in this approach to cross-part rules. The same reasoning, and the same solutions, can be used in these cases also.

When using this approach, it is recommended that you make each of the view parts opaque, and the model part transparent.

Usability & Accessibility

Composer | Form Builder | v4.3 & Higher

General Statement

The W3C defines Accessibility as "Web accessibility means that people with disabilities can use the Web. More specifically, Web accessibility means that people with disabilities can perceive, understand, navigate, and interact with the Web... ". For more information, see http://www.w3.org/WAI/intro/accessibility.php/

Accessibility is challenging. Users with vision impairment, who depend on Screen Reader applications that read aloud the contents of a web page, have to be accommodated. Forms also must cater for partial visual impairment (such as color blindness or poor acuity), cognitive impairment, motor and auditory disabilities.

Composer does a number of things to assist compliance:

- Composer generates many of the HTML tags that are required for correct accessibility. For example, all images are generated with an ALT tag so that blind users can have the image read out to them.
- Composer test forms are verified by a number of accessibility test tools to ensure that the HTML Composer generates passes these tools' criteria.
- Composer's use of stylesheets means that if your first form is accessible, then all subsequent forms are much more likely to be accessible.

However, while Composer tries to do as much as possible, it cannot absolutely verify that your forms are accessible. For example, while Composer will generate an ALT tag for your images, it cannot verify that the tag's text is actually a good description of the image.

Composer will apply consistent colors and styles, but it cannot verify the the contrast of your foreground text against your background color meets the criteria W3C criteria. For more information, see the full WCAG guidelines at http://www.w3.org/TR/WCAG/

To summarize, the accessibility of your forms is ultimately your responsibility.

Our latest portals are largely WCAG compliant, but be aware that individual forms may not be. Some customizations can defeat WCAG compliance.

There is a misconception that the use of JavaScript in a web page makes the page inaccessible. This is completely incorrect. This comes from a perception many years ago, that certain operations in JavaScript confuse screen readers. However, modern screen readers handle JavaScript well, and in fact some JavaScript can make your forms easier for all users, including the disabled. The appropriate use of JavaScript is more of a positive for the disabled users — for example, hiding a section that isn't relevant means that the unsighted user will not have to tab through many irrelevant fields.

Temenos will aim to achieve 100% WCAG v2 AA compliance in TotalValidator, and 100% compliance in SiteMorse. While we do pass automated tests, we also understand that there is a difference between passing tests and being truly accessible. Any additional or specific Accessibility requirements are not necessarily in scope but will be dealt with on a case by case basis.

FAQs

Question:

Are we confident to state that they will be WCAG v2 AA compliant for forms created in Composer?

Short Answer:

No. We will never be able to certify that. But we can make it as easy as possible to achieve AA.

Long answer:

Passing automated tests doesn't guarantee compliance (although it's a good start). Automated tests tell you when you've done something wrong, but they don't tell you when you've failed to do something right. We are in the process of doing further human expert tests on sample forms to improve our compliance.
Ask two different "experts" to certify a form, you can very often get two different answers – there is a level of subjectivity in interpreting WCAG. It also depends on the browser, and the screen reader.

Composer is just a tool. It helps you to build forms that work well, but it also allows you to do what you want. You could easily do things in your form, either intentionally or unintentionally, because of the way you build it, or the JavaScript you hand-craft, that breaks accessibility. We can NEVER certify that your forms will be accessible just because our tool starts from a good place. For example, you may put light grey text on a dark grey background and fail contrast requirements. Or name an image inappropriately. It's a bit like asking Microsoft to certify that all documents you produce will be spelled correctly just because Word has a spell checker.

Ultimately, meeting the needs of people with disabilities is about trying to do the right thing. Organizations will be attacked if they appear not to care. Being concerned, but making some mistakes, and learning from them is defensible. Picking the right tool is a good start. Doing some accessibility testing as part of your development process would further support your good intentions.
Ultimately, the only way you can be 100% sure that the forms you build are accessible is to get an accessibility expert to test them.

Tab Order

Tab Order is the order in which your cursor will navigate from one field to another when you press the Tab key. This is particularly important for users who touch-type, who want to avoid taking their hands off the keyboard and onto the mouse in order to move to the next field; and also for visually or physically impaired users, who favor the keyboard over the mouse.

In Composer, you don't have to set the tab order - Composer sets the tab order for you. There is nothing for you to do, and nothing for you to maintain. If you insert a new field, it will simply be added in the correct tab order relative to the fields around it.

If you need to explicitly control the order of tabbing, you can achieve some control by wrapping your fields in blocks. Tabbing will always occur from field to field within a block before moving to fields in subsequent blocks.

Privacy

Composer | Form Builder | v4.3 & Higher

General Overview

Privacy of information has, for us, 3 major concerns:

1. That only appropriate information is temporarily retained and then submitted
2. That the submission is sent securely to the server
3. That submitted information is retained securely

Only the first of these is the concern of form designers using Composer.

Privacy Concerns with Hidden Fields

Many organizations have strict data privacy policies. A common policy is that the organization should never collect data that the end-user is not aware of. This can be an issue in the following example:

The end user indicates that they would like to include insurance for a spouse by clicking the "Insure my spouse" checkbox.

The form displays an optional section, where the spouse's details are collected.

The end-user later decides not to insure their spouse, and unchecks the "Insure my spouse" checkbox. The Spouse section is hidden.

The end user is now submitting private information about their spouse, that is not required, but has been captured in fields that are no longer visible. This can be viewed as a privacy breach.

Here are two ways of solving this problem:

Clear Hidden Data on Submit

The standard Composer Submit button has a property named "Clear Hidden Data on Submit". This is checked by default. If this property is selected, the form will automatically clear the values of any fields that are hidden when the form is submitted.

Note: The Submit button also has a property named "Clear Non-editable Data on Submit". This is rarely turned on, but is available if required.

Overriding

Sometimes it is necessary to preserve the data in an invisible field even though the user can't see it. For example, there may be a calculated field that is used purely for internal purposes, or there may be fields that are used to communicate information between the form the Transaction Manager.

You may override the usual Clear Hidden Data on Submit behavior on the Submit button by selecting the Preserve If Not Visible option in the Data Clearing Policy property on the Data tab.

Note: There is a special Composer field named "Data Field". This is just a regular Text Field, but is invisible by default, and has Data Clearing Policy set to Preserve. It is intended to store (and submit) internal data rather than user-visible data.

Clear Data when Hidden

Another option it to clear the data as soon as the field is made invisible. This is controlled by an advanced policy property in the Rules area. In most cases this is set to false. The same policy also exists for Read-only fields, although the need for this is rare.

There are pros and cons to this approach:

Pro: It is quite obvious to the end-user that their hidden data is cleared, because if they re-show the data, it will be cleared.

Con: If the user accidentally hides a field or section they have already entered information into, that information will be lost, to their annoyance.

Note: If a field is hidden implicitly (for example, because it is on a wizard page that is currently hidden), the data will not be cleared.

When to Clear Data

The decision whether to clear data from fields on their first hiding or on submission of the form, is a policy decision for each organization.

In both cases, privacy is maintained.

Clear-on-submit generally provides a better user experience.

Clear-on-hide provides a more obvious demonstration to the end-user that their private data will be removed.

Composer's defaults focus on user experience, and so hidden fields clear on submit by default. You may change these defaults in your organization or in a particular form if desired. Please contact Avoka for details on how to set this up in your style-sheets.

Validation

Composer | Form Builder | v4.3 & Higher

Overview

A validation rule verifies the data entered into a form. The simplest and most common type of validations are single field validations, which validate the data in a single field. More complex validations use the values of several different fields.

Note: Validation rules fire only once the user has entered data into the field. You may want to combine a Validation Rule with a Mandatory Rule to ensure that the user enters valid data.

Composer provides several types of standard validation types:

Fixed Length

The fixed length rule ensures that the length of the data (ie the number of characters typed) is exactly the value specified. This is usually useful for a customer identification number or similar that is a fixed length.

Regular Expression

A regular expression is a sophisticated, industry-standard way of specifying a validation pattern. An explanation of regular expressions is beyond the scope of this document, but there are numerous books and web sites that you can refer to. You can also use search engines to find standard regular expressions for common types of data. An example of some regular expressions are:

^[a-zA-Z0-9_-\.]@[a-zA-Z0-9_-\.]\.[a-zA-Z]{2,4}$ - email address

^http\://[a-zA-Z0-9-\.]+\.[a-zA-Z]{2,3}(/\S*)?$ - URL A good resource for regular expressions is: {+}http://regexlib.-com/+.

Online regular expression editor and evaluator can be found at: {+}http://gskinner.com/RegExr/+ and {+}http://+www.debuggex.com/.

The download from following site can also be helpful: {+}http://www.weitz.de/regex-coach/+.

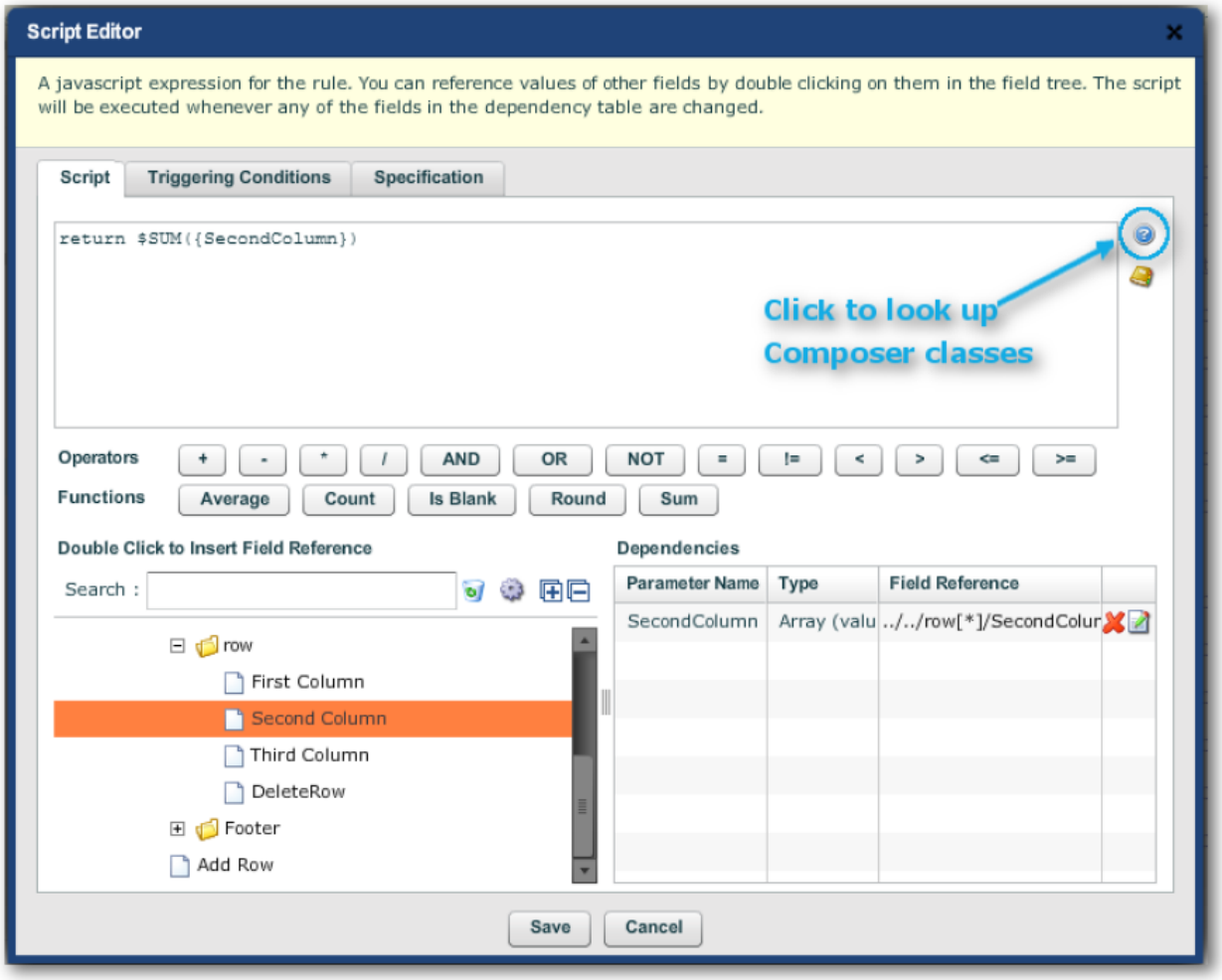
Validation Script

You can write a JavaScript function of arbitrary complexity to evaluate the value entered into any field. In order to write a validation script, you will need to have some minimal JavaScript knowledge. If you return false, the validation will fail.

Here is a sample validation script:

```
var myvalue = sfc.convertToString(sfc.getRawValue(me)); //1 if (myvalue.length < 3) return false; //2
if (myvalue.match("[0-9]")) return false; //3 return true; //4
```

Line 1: This is the usual way to obtain the value of the current field. The "sfc" functions are standard functions that are provided within the Composer libraries. For a full listing, click the "?" icon in the Script editor.



Icon in Script Editor to access Composer classes listing

Line 2: This line uses the JavaScript built-in length operator to test whether the field contains at least 2 characters. Returning false causes the validation to fail

Line 3: This line uses a regular expression to test whether there are any numeric characters in the field. If there are any numeric values, the script returns false.

Line 4: If none of the preceding lines return a false, then return true. This will mean that the validation rule passes.

Validation and Hidden fields

Validation rules are always fired, even if the field is invisible, or is hidden because of some other business rule. This is to enable "hidden" validation rules that aren't represented as a widget in the form.

If you have a field that may become hidden, and you want to ensure that the validation rule on the field doesn't fire when the field is hidden, one way to ensure this is to set the "When Hiding - Clear Data" policy. Validation rules don't fire on empty fields.

Note: This is different to the behavior of mandatory fields. Mandatory fields become non-mandatory when hidden.

Validation Messages

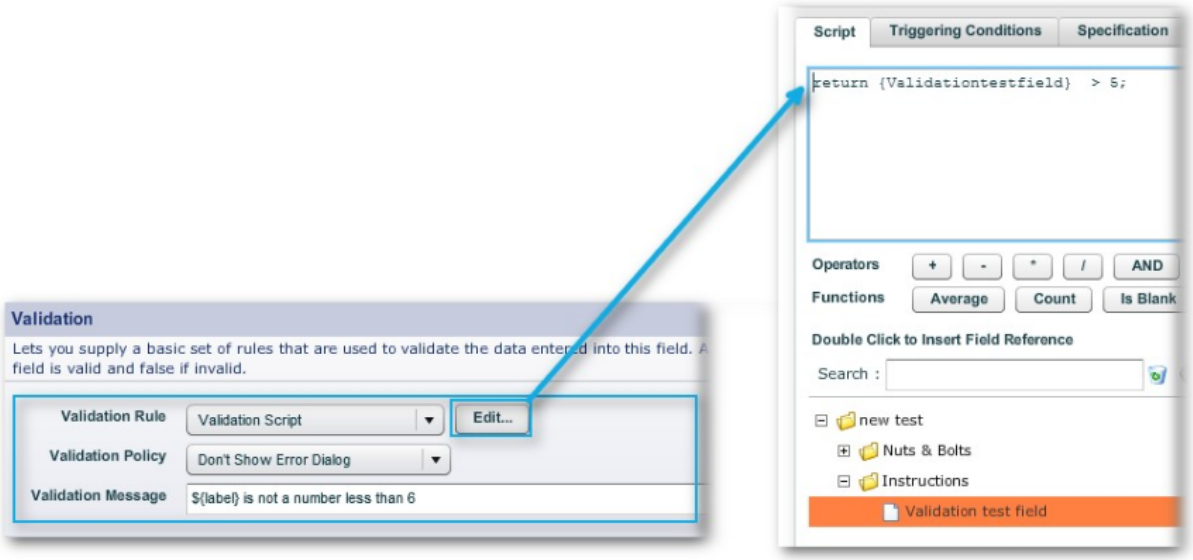
There are 3 types of validation messages:

In the ErrorBlock of the form. Error blocks do not display on mobile devices.

As a dialog, which displays on the failure of the element's validation. Users have to "OK" to continue and so these are rather annoying and intrusive. Inline validation which is the most convenient of the 3.

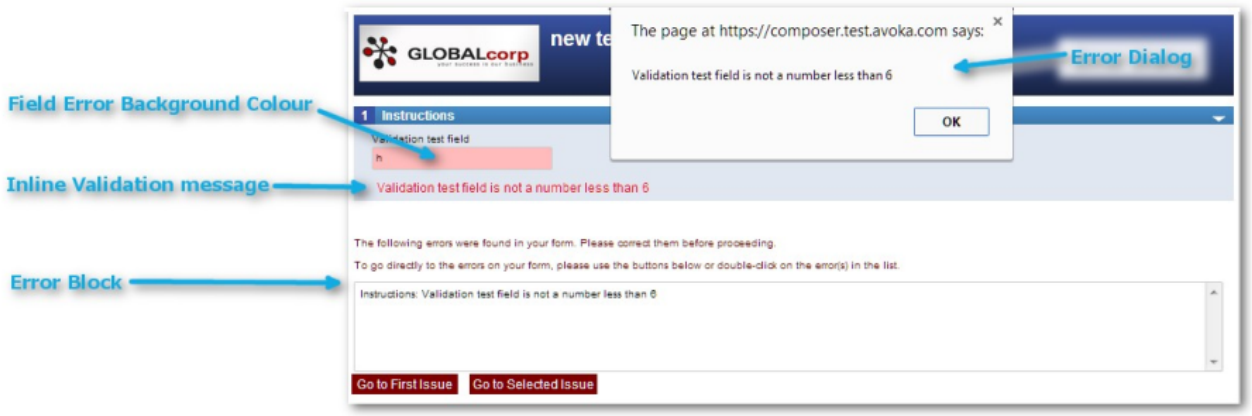
All 3 are turned on by default.

Here is a test field set up to have a trivial validation test. The Error Dialog is left on. The Inline Validation is left on the default setting for the form.



Validation settings for a text field

In the Desktop Preview, the form behaves as follows:



Resultant Validation messages on the Desktop

The Error Dialog is intrusive, requiring users to click on the "OK" button. You can turn it off, but only for each widget individually, by "Edit Properties -> Rules -> Validation -> Validation -> Validation Policy -> Don't Show Error Dialog".

The Error Block does not appear on mobile devices.

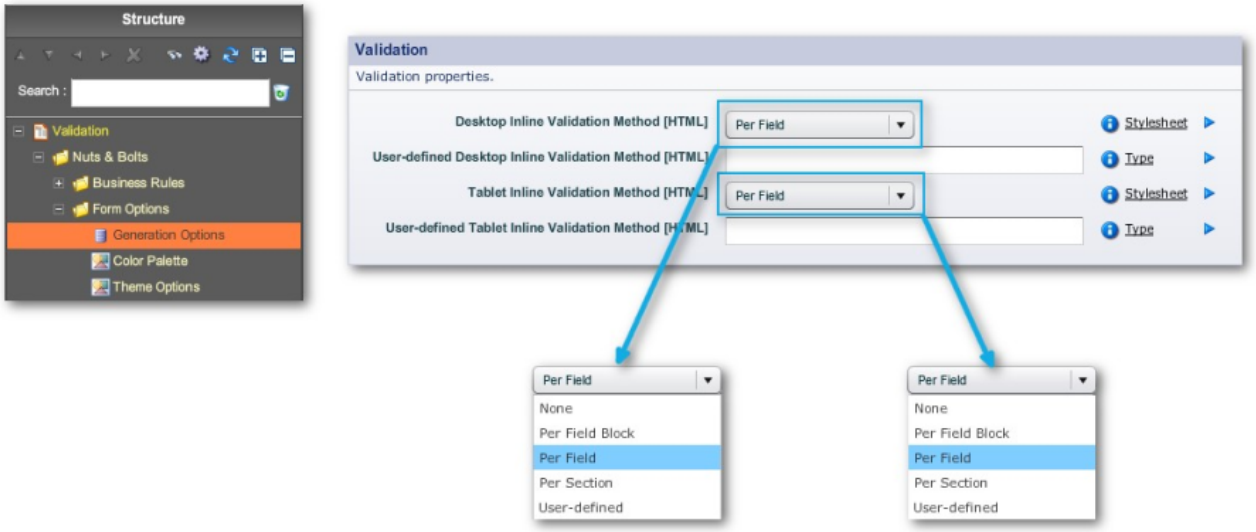
In Line Validation

Inline Validation is really another instance of Dynamic Data, that is: if a field's validation test fails, and inline validation is turned on, the message to the user is displayed without the form page reloading.

The following examples have the Error Dialog turned off for each widget. The Error Block remains on, though will only be visible for the desktop previews.

Inline Validation Settings

These are found in "Nuts & Bolts -> Form Options -> Generation Options -> Edit Properties -> Properties -> HTML Generation -> Validation", as shown below:



Some of the important form-wide HTML Inline Validation options

Inline Validation Example

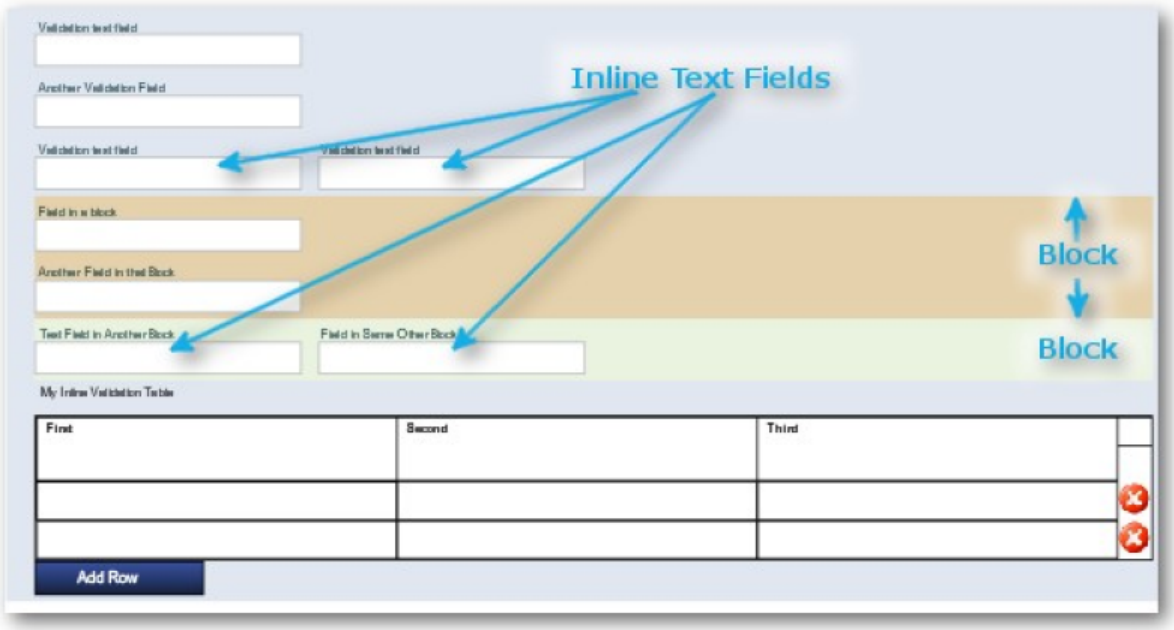
The form used here, though very simple, embodies the various inline validation behaviors for the desktop and on mobile devices. Its salient features are:

2 blocks in the first section (delineated by background colors)

Text fields on separate lines or inline (as inline validation behaves differently for these)

A simple table (to show how inline validation methods behave on the desktop and in the various modes of mobile devices).

The Responsive Rules are the defaults, so that on an iPad, Landscape is below Threshold 2; portrait, below Threshold 3.



Wireframe of form used to demonstrate Inline Validation

Desktop Inline Validation Method [HTML]

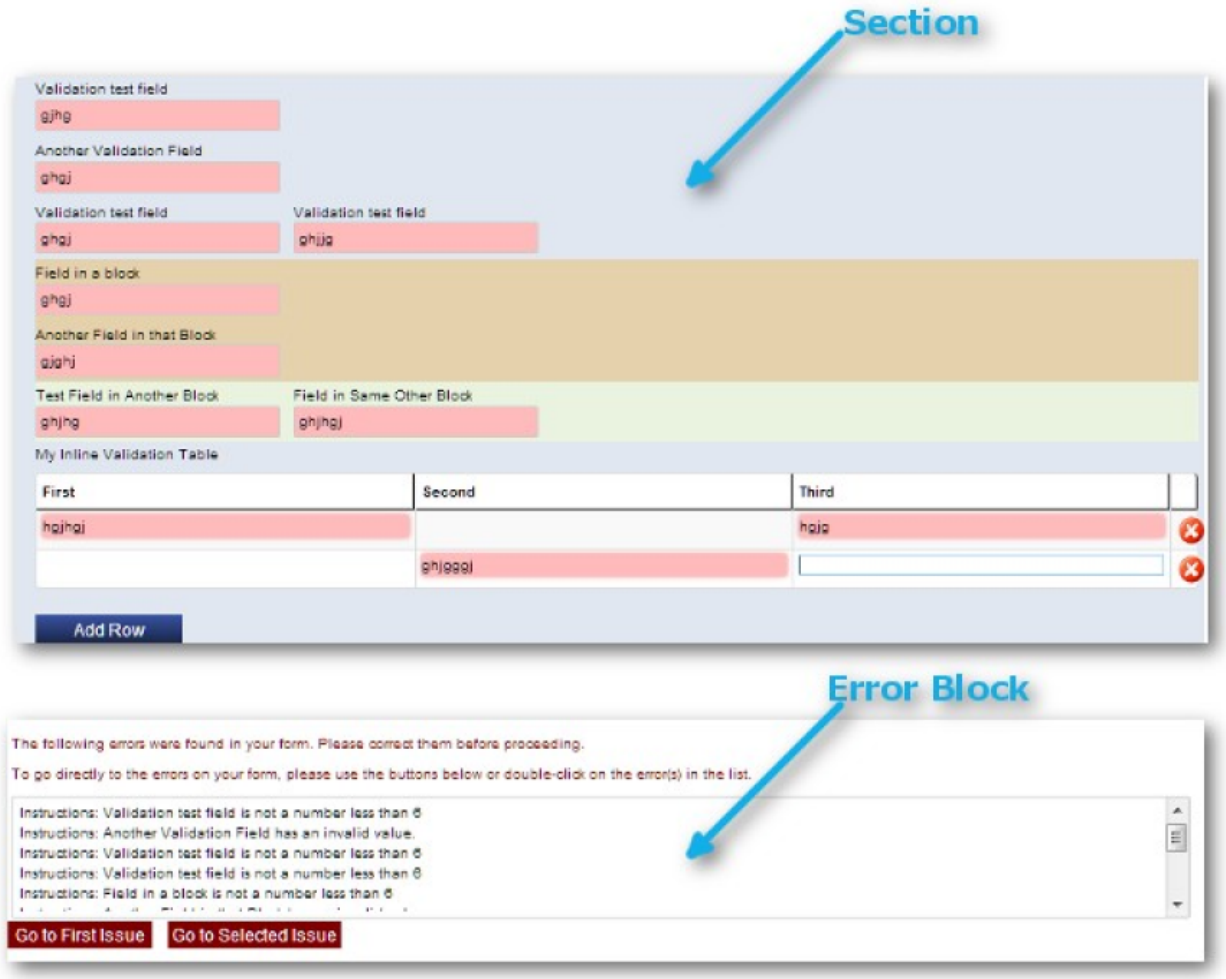
In all cases, we have turned off the Error Dialog for convenience for each element at "Edit Properties -> Rules -> Validation -> Validation -> Validation Rule -> Don't Show Error Message".

All the fields with data entry are set to fail validation in this demo. The text of the Inline validation methods are set in each widget at "Edit Properties -> Rules -> Validation -> Validation Message".

Here are the Preview results for each desktop setting.

None, Desktop

The only messages the user gets is the Error Dialog when turned on at the widget-level, or the clickable Error Block list.



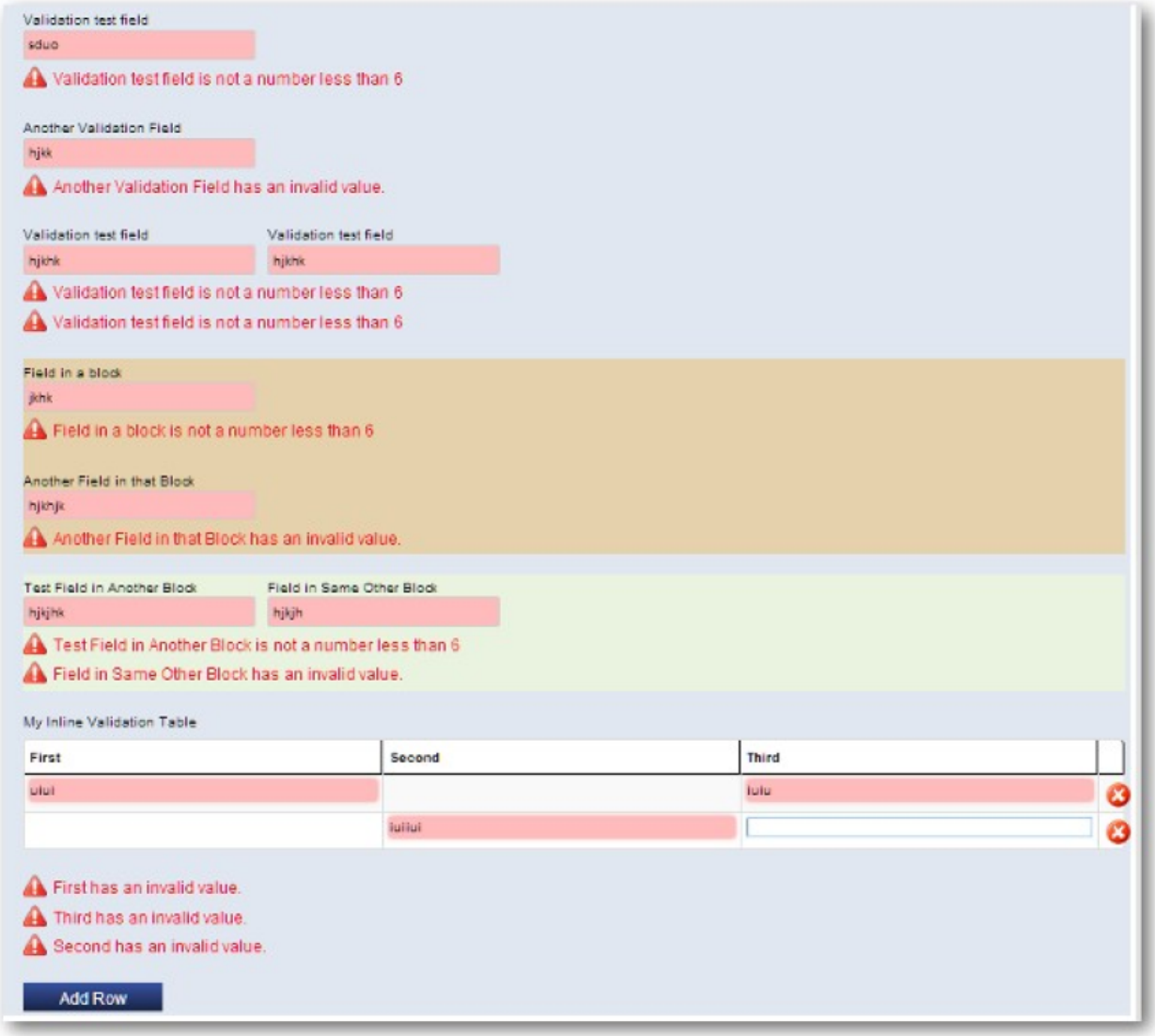
Inline Validation: Desktop, None

Per Field Block, Desktop

Inline Messages are under each field or line of fields

The same behaviour for fields within blocks (see the two blocks marked out with background colours in the example below)

Inline messages for elements in tables get listed in a group below the table but above the standard "Add" button.

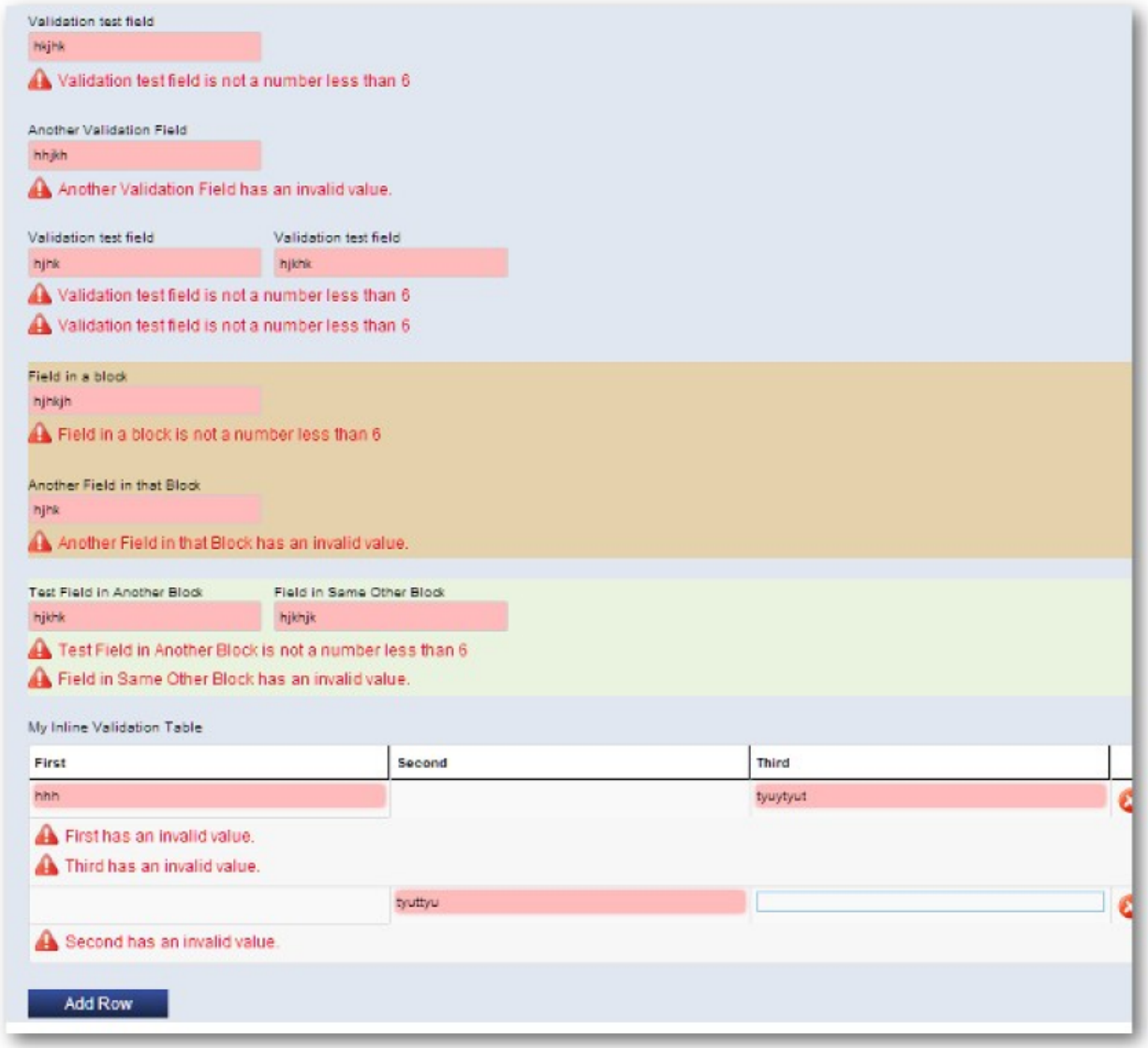


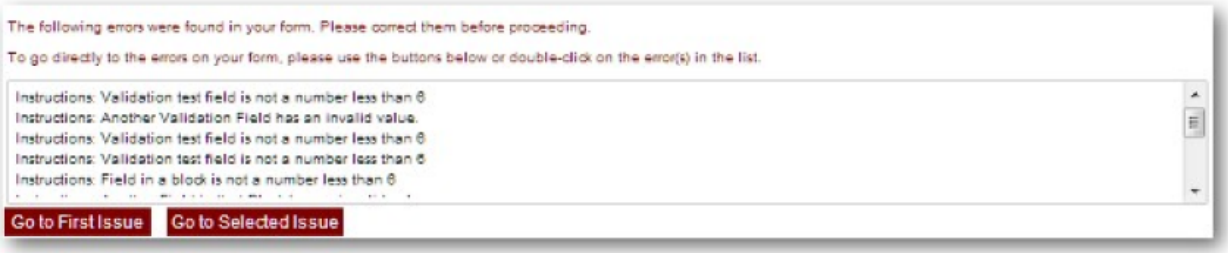


Inline Validation: Desktop, Per Field Block

Per Field, Desktop

The same beaviour as the "Per Field" setting, except for
Tables, where the messages for each row come in a group under that row.



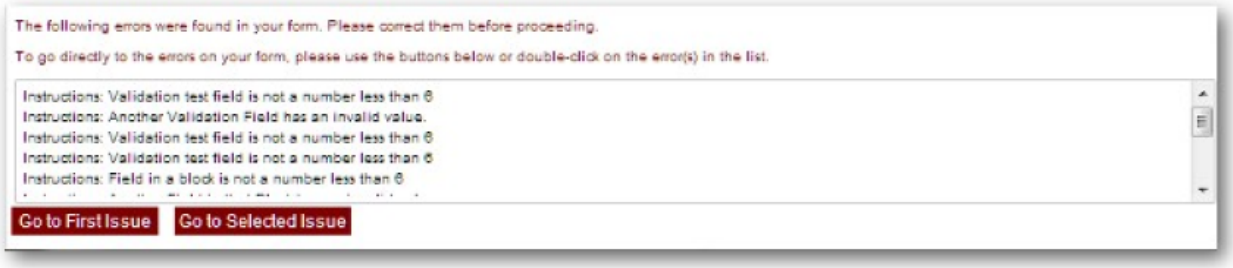


Inline Validation: Desktop, Per Field

Per Section, Desktop

All the Inline Messages in a section come in a group at the end of each section.

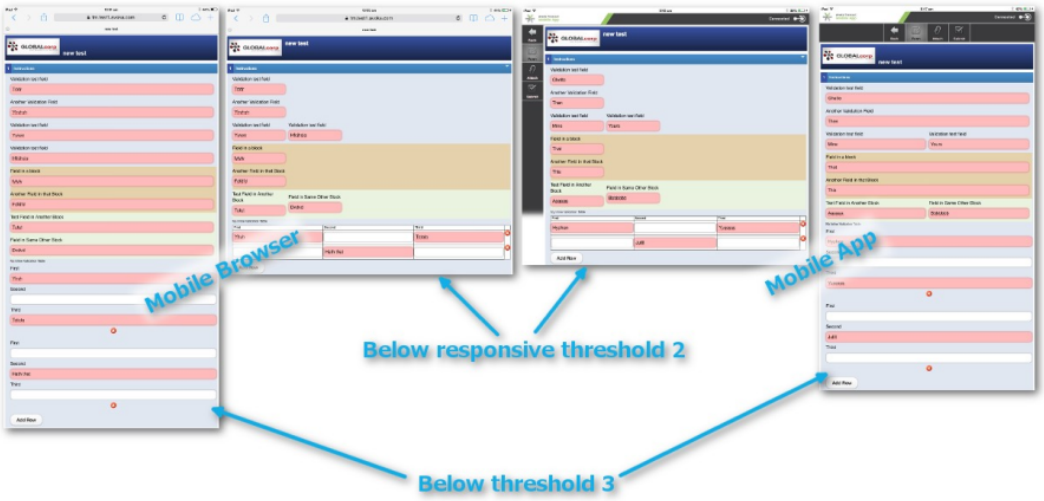




Inline Validation: Desktop, Per Section

Mobile Inline Validation Method [HTML]

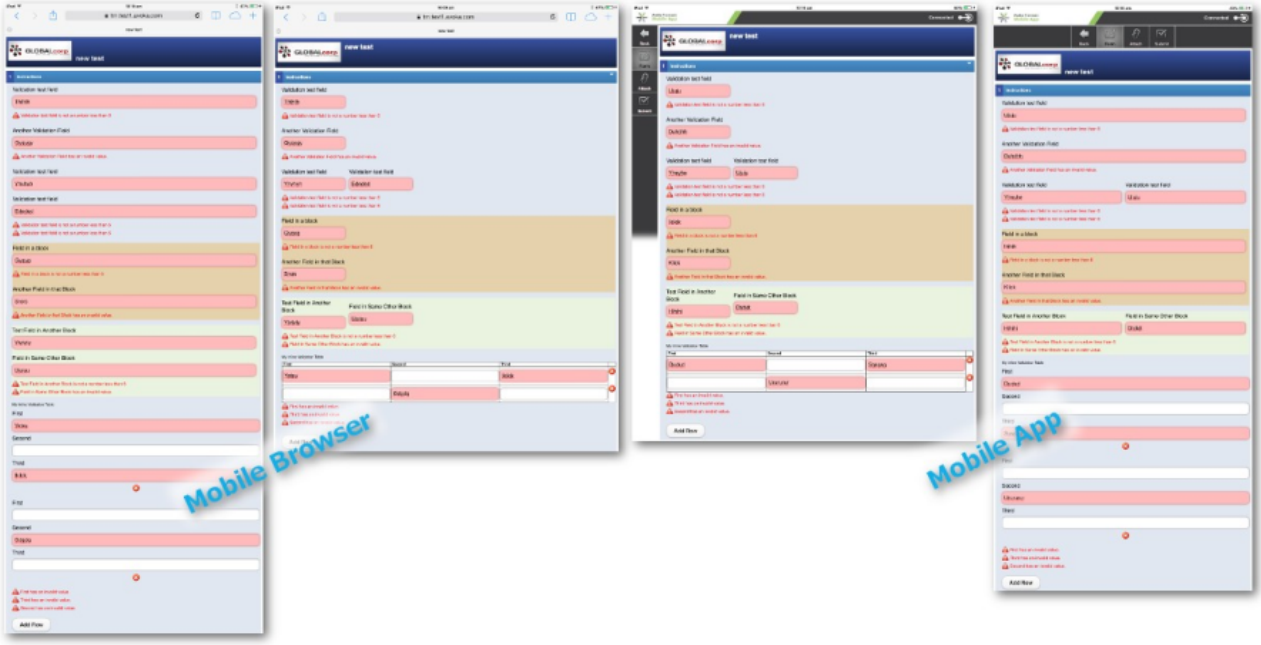
The mobile settings are independent of the desktop settings, and apply equally to forms accessed through viewing the form either in a mobile browser or in [thetheMobileApp](#).



None, Mobile

Mobile: Inline Validation method = None

Per Field Block, Mobile



Mobile: Inline Validation method = per FieldBlock



Per Field, Mobile

Mobile: Inline Validation = per Field

Per Section, Mobile



Mobile: Inline Validation method = perSection

Transact Composer User Guide and ReferenceAdvanced Topics

Analytics

Composer | Form Builder | v4.3 & Higher

Overview

Transaction Manager, the backend of the Transact platforms forms, will obviously log information on form submissions:

Data on the original form request:

o The user's IP address

o the agent string of the browser, from which holds information on the Operating System (Windows, OS X, iOS, Android)

The version number of the system

the device (for example, Windows desktop, iPad, and so on)

the browser brand (i.e. Chrome, Firefox, Safari, Internet Explorer) and version number

- The referring page (i.e. the page containing the link the user activated to reach the form) and portal
- The time stamp of the original request
- Data and keys on the session, cookies and other technical information Whether the form was saved or abandoned, including
- When the form was submitted (if ever)

The form's rendering time (vital information for optimizing formperformance)

TM stores the data on each form request and also consolidates these data and renders them in various graphs and dashboards in TM.

Composer's Role in Analytics

Background Saving

Time was when the data entered onto a form was not saved back to the server until the end user submitted the form. Now, Composer has tools to:

Configure background saving

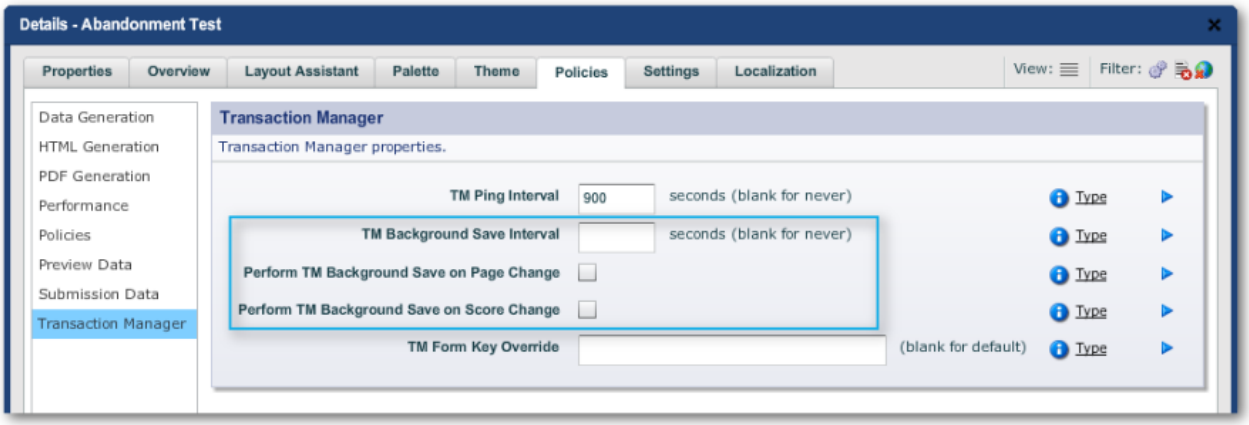
where data is sent dynamically back to Transaction Manager before the form is submitted.

The data can be either the values entered onto the form or an arbitrary numerical score where you ascribe various numerical values to some or all of the fields on the form

Background saving can be set to take place at regular time intervals, or when the user navigates to another wizard-style page, or when the numerical score changes value

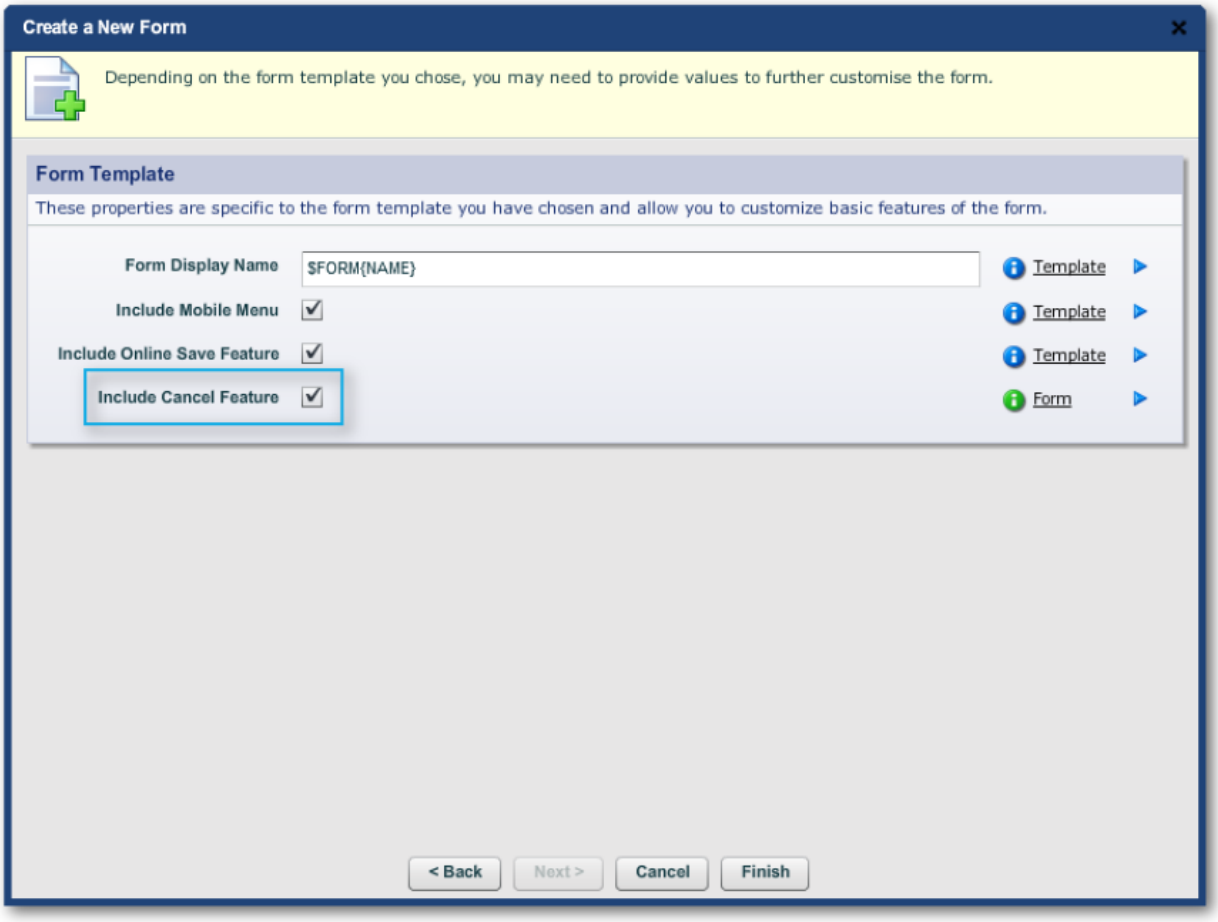
You may not want, for privacy reasons, to retain the actual data entered on the form by an end user who has not yet submitted this data, but still want to have data on where the user reached in the form before abandoning it so that you can modify the form so more users successfully get to the submit stage of entry. This substantially increases the value of the form to your business. Without such a feedback mechanism, you will not be able to make as many meaningful improvements to the form.

You activate background saving through "Structure -> <Top item of the structure tree> Edit Properties -> Policies tab -> Transaction Manager menu item -> Transaction Manager panel:



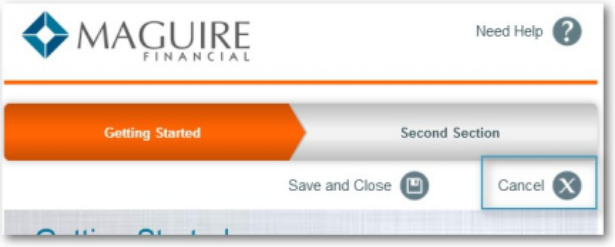
The Cancel Button in Maguire Forms

The Maguire template forms have a Cancel facility. This must either be activated when the new form is created, the Maguire template chosen in the "Create a New Form" Wizard and the "Include Cancel Feature" checked in the last page of the wizard dialog.

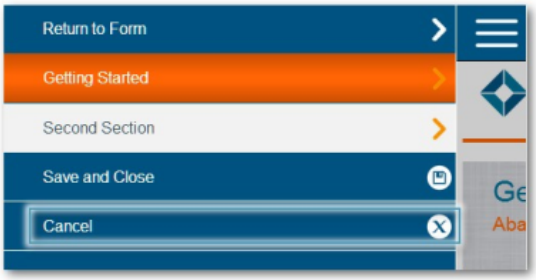


The resulting feature is a "Cancel" button. Currently this is **not** visible in the Wireframe, only in the Preview (here in both desktop mode and in triggered mobile responsive view):

Desktop Mode



Mobile Mode

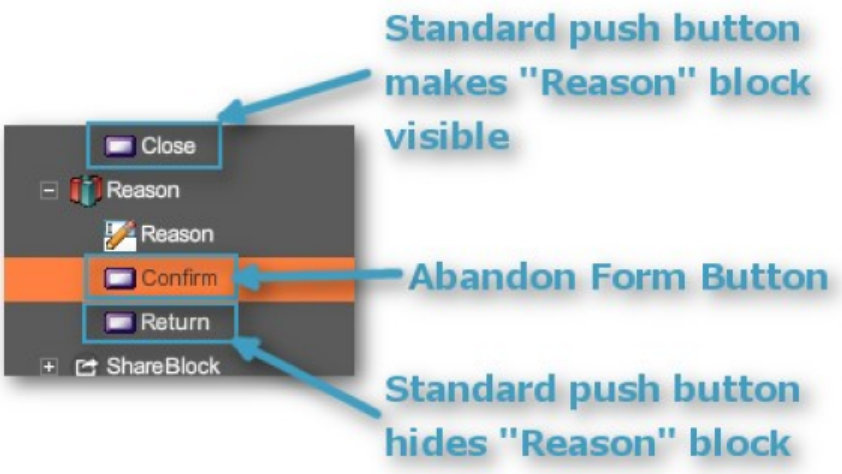


Clicking on "Cancel" results in the form's data being saved to TM and the form's data and analytics being displayed in TM under "Home Dashboard -> Abandoned Transactions".

The Abandon Widget in non-Maguire Forms

The same function is available in non-Maguire templated forms. Place the "Palette -> Field Types -> Abandon Form Button" on your form. Probably the best places are in the Header and Footers (which are exposed in the Advanced Mode of the Structure Palette as the "Form Header" and "Form Footer" blocks below the Nuts & Bolts block.

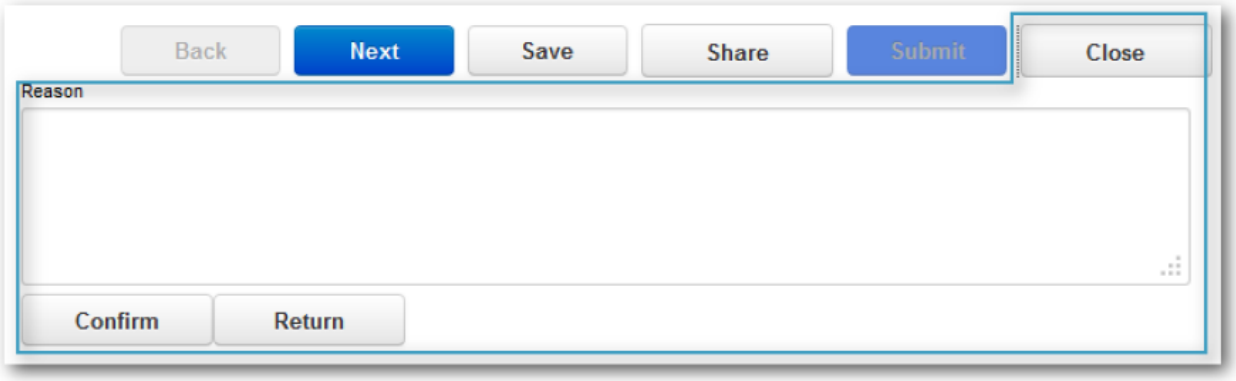
You can even add a "Reason" text block for users to give the reasons for abandoning the form. We recommend the following technique to make the text block hidden and then become visible when users click on a "Close" button.



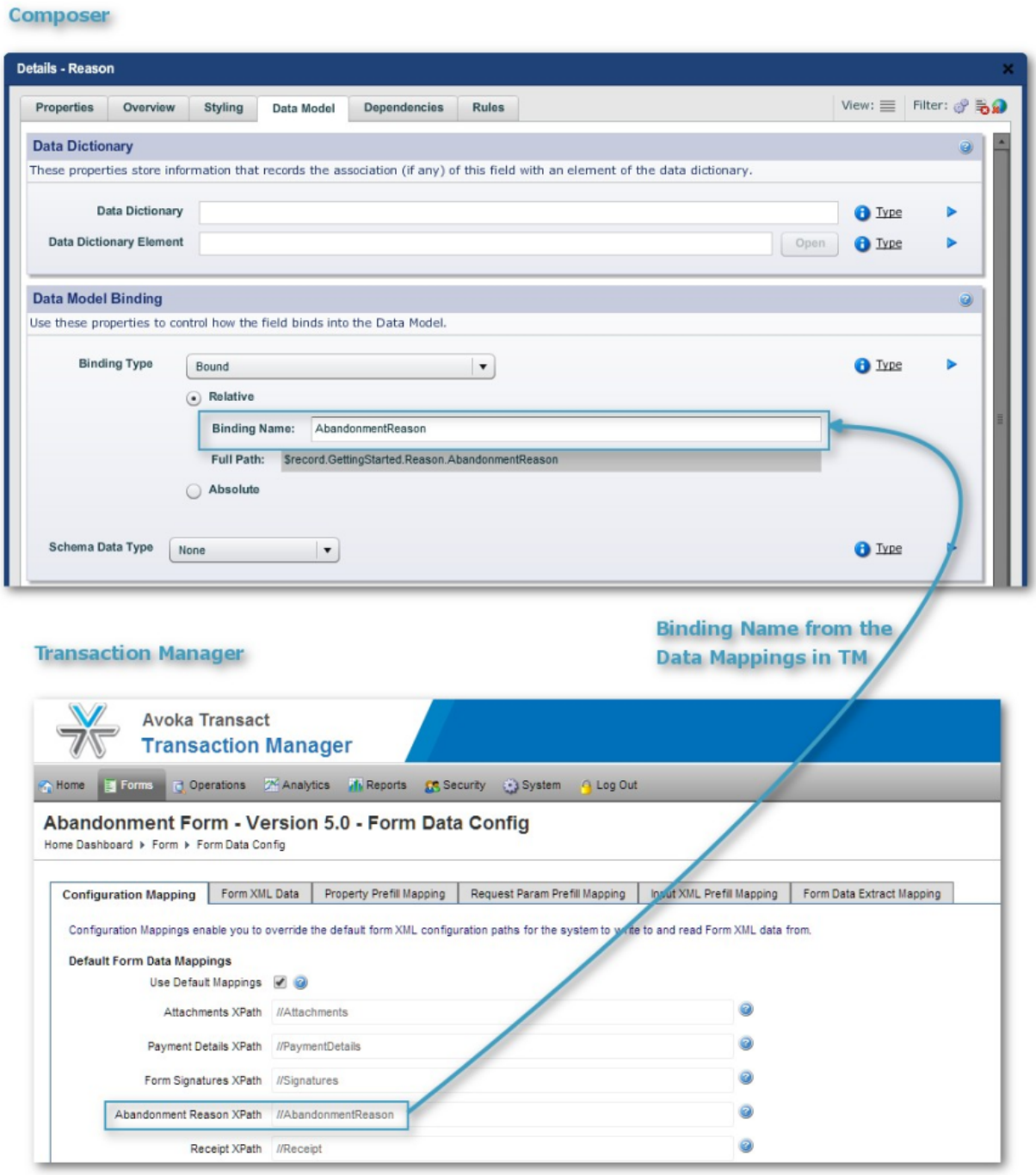
Use Click Action scripts to control the show/hide behavior of the "Close" and "Return" standard buttons. The most basic Click Action scripts to accomplish this (with no refinements whatsoever) are:

For the "Close" button: sfc.updateVisibility({Reason}, true, "exclude"); And the "Return": sfc.updateVisibility({Reason}, false, "exclude");

On the form, the fields look as follows (when placed into the "Form Footer" block"):



By careful configuration of the "Reason" text area, you can make the contents of this field be listed in the TM "Home -> Operations -> Abandoned Transactions" page in the Abandon Reason column. Use the Data Model Binding panel ("Edit Properties -> Data Model tab -> Data Model Binding panel").

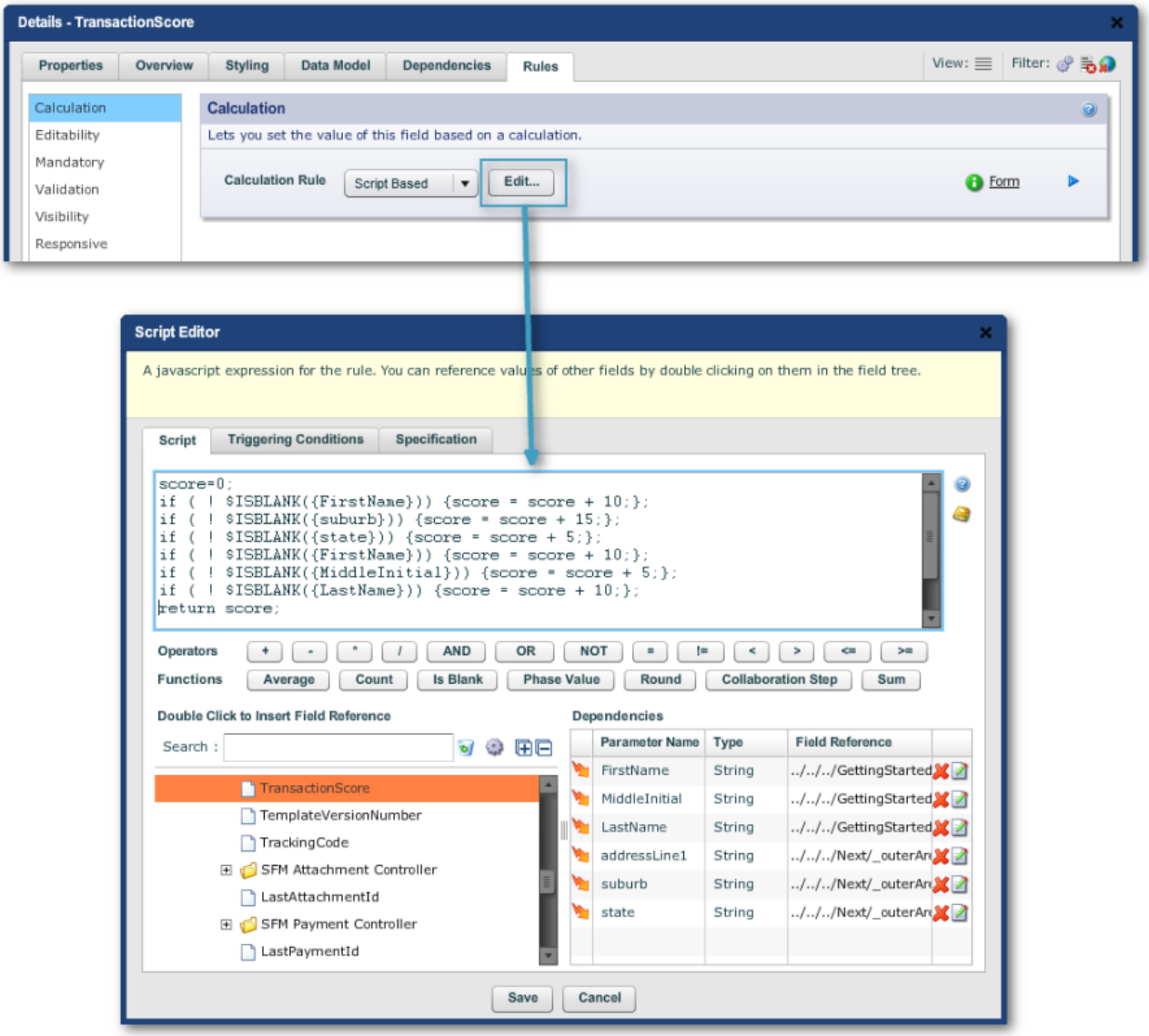


The Binding Name used is one of the Default Form Data Mappings in TM. The double slash "//" just means a wild-card: in other words, the field mapped to AbandonmentReason can be anywhere in the form's structure. If there are more than one such field, TM will use the first it encounters in the tree. You should be setting the mapping unique for best results.

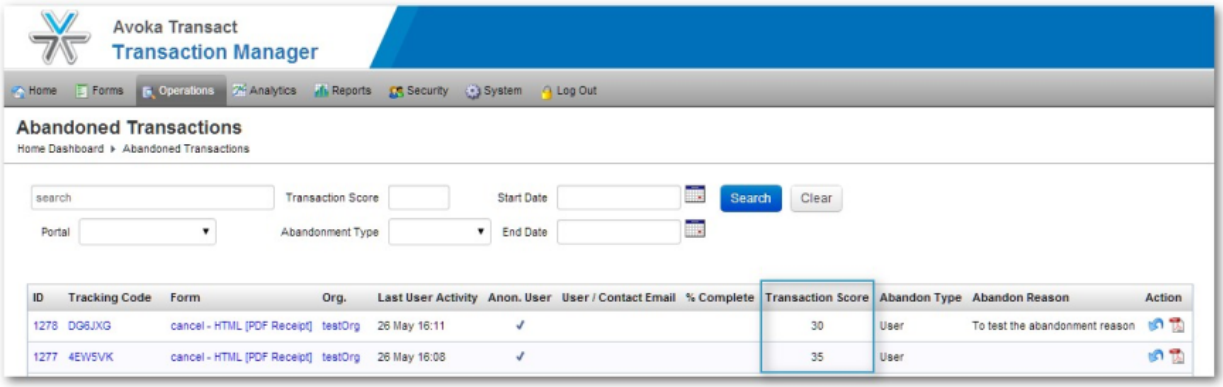
Scoring

We have seen how you can retain data from an abandoned form. However, due to privacy concerns, may not actually want to retain this data, and yet still get some metrics on the abandonded form so as to identify usability issues which could have contributed to its being abandoned in the first place. One way is to score the form to see how far the user managed to get to before abandoning or canceling. Scoring could also be useful in your analysis, even when you retain the incomplete data.

All you need do to set up scoring in Composer is add some simple scripting to "Structure Panel -> <form name> -> Nuts & Bolts -> Transaction Manager Support -> TransactionScore -> Edit Properties -> Rules tab -> Calculation panel":



The above script is a very simple and unsophisticated example. Whatever the script used. it must return an integer, which will, after you have exported the form, then be displayed in Transaction Manager under "Operations -> Abandoned Transactions":


Avoka Transact
Transaction Manager
Home
Forms
Operations
Analytics
Reports
Security
System
Log Out
Abandoned Transactions
Home Dashboard ▸ Abandoned Transactions
search
Transaction Score
Start Date
Search
Clear
Portal
Abandonment Type
End Date
ID
Tracking Code
Form
Org.
Last User Activity
Anon. User
User / Contact Email
% Complete
Transaction Score
Abandon Type
Abandon Reason
Action
1278
DG6JXG
cancel - HTML [PDF Receipt]
testOrg
26 May 16:11
✓
30
User
To test the abandonment reason
1277
4EW5VK
cancel - HTML [PDF Receipt]
testOrg
26 May 16:08
✓
35
User

Field Types

Composer | Form Builder | v4.3 & Higher

There are a few field types (or "widgets") that require further documentation in this manual. They are:
Rich Test Fields Cascading Dropdowns Prepoplated Dropdowns

Rich Text Fields

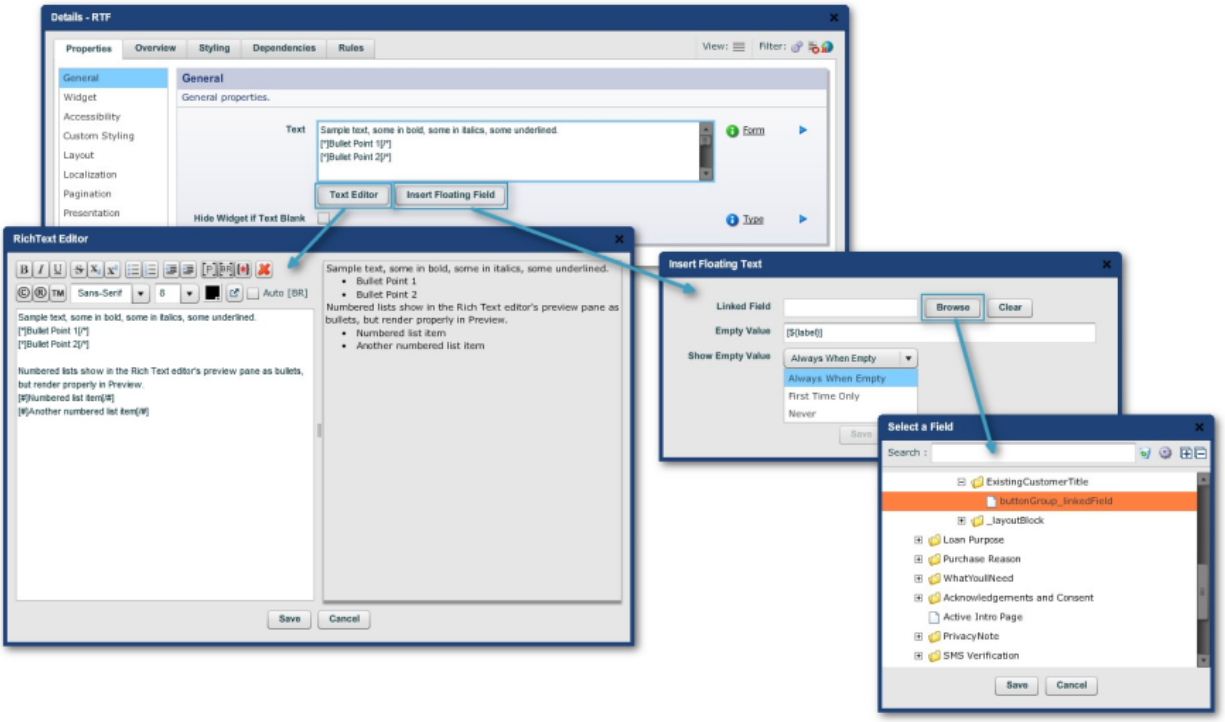
A Rich Text is a read-only widget that has styling, such as bolding, italics, fonts, and colors, embeded into the text. Rich Text is often used for explanatory or legal text in forms; many other fields also support Rich Text, such as the Labels for all fields and Section Help Text.

You can embed the values of other fields within a Rich Text Field. **This replaces thelegacy widget "FloatingField"** which no longer ships in the Palette.

You control the format of the text by embedding "tags" or "markup" into the text. We will list these below. Tags must be properly opened and closed as a pair. If the pair is incorrect, the text will not display correctly. Rich Text may render slightly differently in the wireframe, PDF and HTML. This is because the underlying implementations on each platform are not the same. Composer attempts to make the text appear as close as possible on all platforms — if you discover discrepancies, please notify us with a sample.

Rich Text Editor

The Rich Text Editor ("Text [Rich}" widget, "Edit Properties -> Properties tab -> General -> General panel -> provides simple ways of embedding rich text tags into your text.



Consider the famous quotation:

Ask not what your country can do for you; ask what you can do for yourcountry.

Instead, use a Country Dropdown List at the top of the form asking for which country. And then depending on the country choosem, the quotation reads:

Ask not what America can do for you, ask what you can do forAmerica.

or

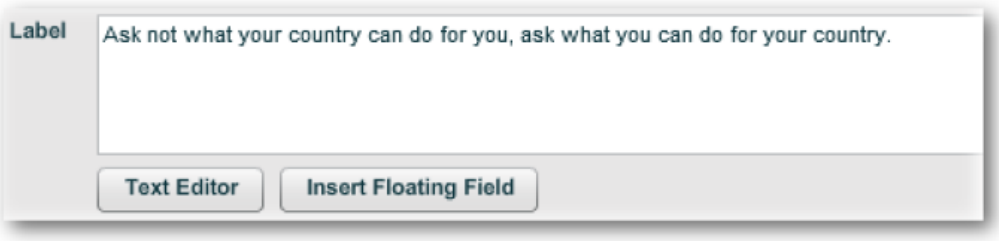
Ask not what Canada can do for you, ask what you can do forCanada.

And so on.

This is quite straight forward to do.

Create the Country Dropdown (or any other field whose value you want to appear in the text.) This field can appear anywhere in the form.

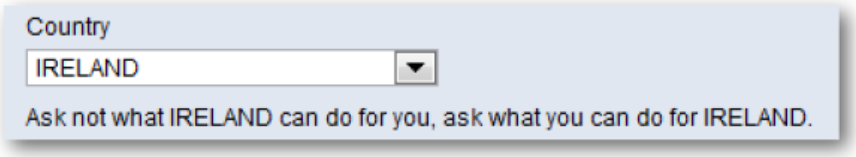
Create a Rich Text field as shown below:



Click on the "Insert Floating Field" button, and point at the Country Dropdown created earlier.
You may also want to update the other two values in the dialog - these control what is displayed in the Floating Field if the Country Dropdown has a blank value.



Copy and paste the floating field reference into the two locations, so that it looks like this:
Ask not what$FLOAT{Country_linkedField} can do foryou, ask what you can do for $FLOAT{Country_linkedField}.
Test your form. You should get something that looks like this:



Note that Composer has created a hidden Floating field underneath the Rich Text field. Do not delete this field, as it temporarily stores the value to be injected into the Rich Text, and is linked to the real value. You can modify the formatted value of this field if desired.The Rich Text toolbar icons can be used in two ways:
To apply a tag to an existing piece of text, select the text, and then click the icon.
If you click the icon when no text is selected, the matching tags will be placed into your text, with the cursor positioned for you to start typing.
The Auto [BR] checkbox, if selected, will automatically add a "[BR/]" tag into your text every time you press the Enter key.
Tip: It is strongly recommended not to specify custom fonts, as these may not render correctly on different systems.

Tag Reference

There are two types of tags, those that affect individual characters within the text, and those that turn whole paragraphs into list items (either bullets or numbered items).

Character tags

Tag	Formatting	Sample	Result
[i]..[/i]	Italics	Hello [i]World[/i]	Hello World
[b]..[/b]	Bold	Hello [b]World[/b]	Hello **World**
[u]..[/u]	Underline	Hello [u]World[/u]	Hello World
[x]..[/x]	Strike Thru	Hello [x]World[/x]	Hello ~~World~~
[sub][/sub]	Subscript	Hello [sub]World [/sub]	Hello world
[sup][/sup]	Superscript	Hello [sup]World [/sup]	Hello world
[colour:#00FF99] [/colour]	Color	[colour:#ff0000]Red [/ colour] and [color:rgb(0,0,255)]blue [/ color]	Red and blue
[font:Arial][/font]	Font type	My font is [font:Myriad Pro]Myriad Pro [/font]	Fonts available depend on the fonts available in your destination system, and vary between PDF and HTML. See separate topic on Fonts.
[fontsize:9pt] [/fontsize]	Font size	My font is so [font-size:16pt]big[/ font-size]	My font is so big.
[link:][/link]	Link	[link:[http:// www.avoka.com]Avoka[/ link]	Avoka (Links work in HTML and more recent releases of Adobe Reader.)
[copy] [reg] [trade]	Symbol	Copyright [copy] 2010 Avoka Technologies.	Copyright © 2010 Avoka Technologies.

Tag	Formatting	Sample	Result

${options:policy.mandatory .marker}	Mandatory marker. This is used to insert the mandatory marker (usually a red asterisk, but controlled via your stylesheet) into your help text.	All fields marked with ${options:policy.mandatory .marker} are mandatory.	All fields marked with * are mandatory.

Bullets, Lists and ParagraphTags

Bullets and Lists affect an entire paragraph of text. The bullet character or numbering style is controlled via the properties and the stylesheet for your form. Extra levels of indent are controlled via a leading dot.

Example	Sample
[]USA[/] **[.]CA[/]** **[..]San Francisco[/] [.]NY[/]** **[..]New York[/] []Australia[/] [.]NSW[/]** **[..]Sydney[/]**	USA • CA San Francisco • NY New York Australia • NSW Sydney
[#]USA[/#] [.#]CA[/#] [..#]San Francisco[/#] [.#]NY[/#] [..#]New York[/#] [#]Australia[/#] [.#]NSW[/#] [..#]Sydney[/#]	1. USA 1. CA 1. San Francisco 2. NY 1. New York 2. Australia 1. NSW 1. Sydney
[#reset]	Use this tag to reset the numbering list back to 1. The reset take will take effect in the next rich text field.

[p]Use the paragraph tags to wrap around multiple paragraphs of text. Each paragraph will be spaced from the each other.[/p] [p]This is another paragraph[/p]	Use the paragraph tags to wrap around multiple paragraphs of text. Each paragraph will be spaced from the each other. This is another paragraph
[.p]A paragraph. Notice the spacing[/p] [..p]One level of indent[/p] [...p]Two levels of indent[/p]	A paragraph. Notice the spacing One level of indent Two levels of indent
You can break [br/] text onto [br/]multiple lines without any spacing	You can break text onto multiple lines without any spacing

Auto-numbering is controlled by Sequences in Nuts & Bolts. This is an advanced topic, and usually configured for you in your style-sheet.

Paragraphs are optional, but are useful for longer blocks of rich text.

Radio Buttons

Radio buttons in Composer required a lot of documentation. No longer: Composer 4 now handles them gracefully.

Creating Radio Button Groups Manually

The Widget Palette now contains a "Smart" family of radio button elements:

Smart Radio Button

Smart Radio Button Group

To create a functioning set of radio buttons, you need the Group widget and a smart button widget for each of the buttons to be selected. For each smart button you place on the form, in the wizard dialog, use the Browse button to point to the Linked Button Group. (If you neglect to do this in the wizard dialog, you can point to the group in "Edit Properties -> Properties tab -> Data -> Data panel".)

If you want to pick up the selected value of the button group in, say, an inserted floating field in a Rich Text Field, always point to the button group.

Now you can take a whole button set, place them into a simple block and copy that block to another part of the form. This now works without any fuss.

Mandatory Radio Buttons

It doesn't really make sense to make an individual Radio Button mandatory. As you can see above, you can make the Button Group mandatory - this means that the user will have to select one of the buttons in the group. See MandatoryBlock.

Calculating or Using the value of a Radio Button Group

If you need to calculate the value of a Radio Button Group, you need to place the calculation on the Button Group, rather than individual buttons. Please refer to ScriptingandDependencies.

If you need to use the value of a Radio Button in another rule, you can either treat individual buttons similarly to a checkbox (and use Is-Selected in the parameter) or you can refer to the value of the Radio Button Group.

Dropdown Menus

Use the "DropDown List" widget from the Palette.

A Dropdown List has two sets of values in the "Data" tab of the advanced properties dialog of the widget:

Display Values

This is a list of values that are displayed in the user interface.

Values

This is a list of underlying raw values the field will pass on to TM when a user selects a display value.

Display Values has a default value of ${values}. This means that by default, the set of Display Values is identical to the Values. This simplifies using Dropdown Lists, because you only need one set of lists.

However, sometimes it's useful to have a different display value versus the underlying value. For example, you may want to display countries in the Dropdown list using their full names such as Australia, United Kingdom and France, but store their underlying values using the two letter country abbreviations such as AU, UK, and FR. In this case, you can simply modify the values of Display Values.

Note:

The number of Display Values and Values must be identical.

You must delete the ${values} formula if you're going to add explicit Display Values.

If you use Composer's translation feature, only the Display Values will be translated. This means that any code that uses the Values will continue to work without modification.

Internally, the Values and Display Values properties are stored using a pipe character "| " as a separator. This means that you cannot use pipe characters in your Values or Display Values.

To access the Display Value in a calculate event, use the following script:

sfc.getFormattedValue(noderef)

In PDF, a Dropdown list can have a blank value even if "blank" is not one of the allowed values. (When a Dropdown list is first displayed, it's value is always blank unless initialized.) In HTML, a Dropdown List is only allowed to have a blank value if blank is one of the values in the Values list. In order to achieve compatibility between the two implementations of Dropdown List, Composer provides a "Automatically Add

Blank Value [HTML]" property which is checked by default. This has the added benefit of making the concept of a Mandatory Dropdown List meaningful.

Composer also provides a number of prepopulated dropdown lists. See PredefinedBlocks. These include: Dropdown List [Countries]: A list of countries from around the world.

Dropdown List [Genders]: Contains 'Male' and 'Female'.

Dropdown List [States+Codes,United States]: A list of states found in the United States is displayed in the dropdown list, e.g. Alabama. This is what the form filler sees. However a corresponding two letter country code e.g. AL is stored in the underlying data.

Dropdown List [States,Australia]: A list of Australian states.

Dropdown List [States,United States]: A list of states found in the United States. In this dropdown list, the displayed value and 'stored' value are identical.

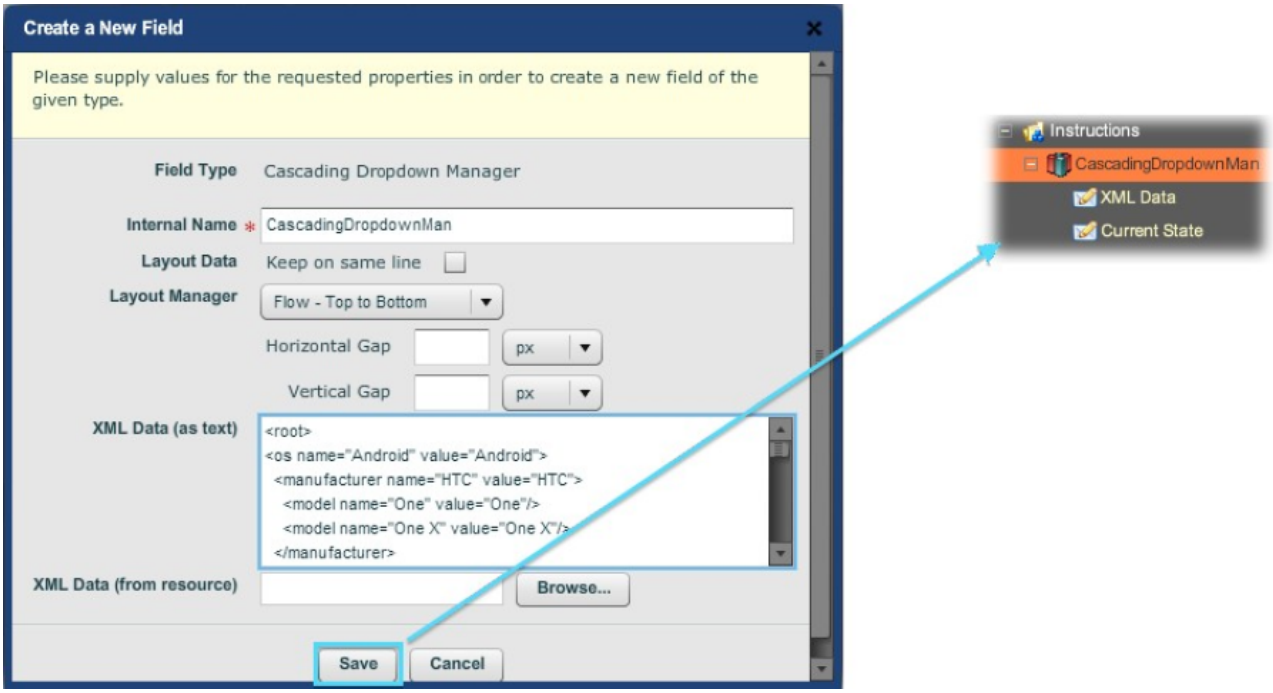
Dropdown List [Yes/No]: Contains 'Yes' and 'No' values.

Cascading Dropdowns

The first step is to define the behaviour of the set of cascading dropdowns with XML. Here is an example of three orders of cascading dropdowns: os -> manufacturer -> model.

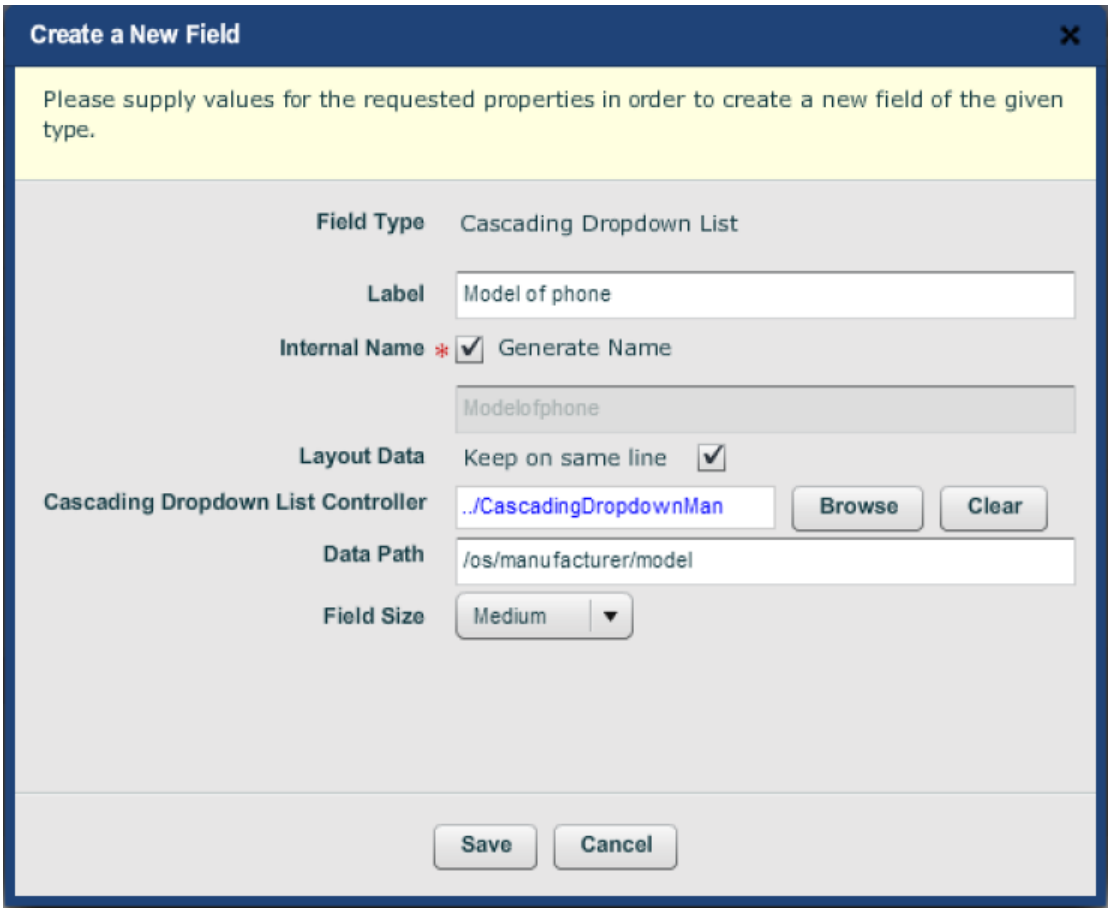
```
<root>
<os name="Android" value="Android">
<manufacturer name="HTC" value="HTC">
<model name="One" value="One"/>
<model name="One X" value="One X"/>
</manufacturer>
<manufacturer name="Samsung" value="Samsung">
<model name="Galaxy S3" value="Galaxy S3"/>
<model name="Galaxy Note 3" value="Galaxy 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>
</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>
```

Drop a Cascading Dropdown Manager widget onto a section (here, the default "Instructions") in the Structure pane. Paste the XML definition into the XML Data test field.


Create a New Field
Please supply values for the requested properties in order to create a new field of the given type.
Field Type
Cascading Dropdown Manager
Internal Name *
CascadingDropdownMan
Layout Data
Keep on same line
Layout Manager
Flow - Top to Bottom
Horizontal Gap
px
Vertical Gap
px
XML Data (as text)
<root>
<os name="Android" value="Android">
<manufacturer name="HTC" value="HTC">
<model name="One" value="One"/>
<model name="One X" value="One X"/>
</manufacturer>
XML Data (from resource)
Browse...
Save
Cancel
Instructions
CascadingDropdownMan
XML Data
Current State

The Cascading Dropdown Manager predefined block dialog, with the XML pasted into the data textbox. "Save" creates the structure shown.

Drop three "Cascading Dropdown List" widgets into the structure, Here we have put the three widgets onto the section level of the form. Fill in their dialogs, giving the name (as it will appear on the form), point to the "Cascading Dropdown Manager Controller" and give the data path from the root as defined by the XML. Here is the dialog for the last of these lists:



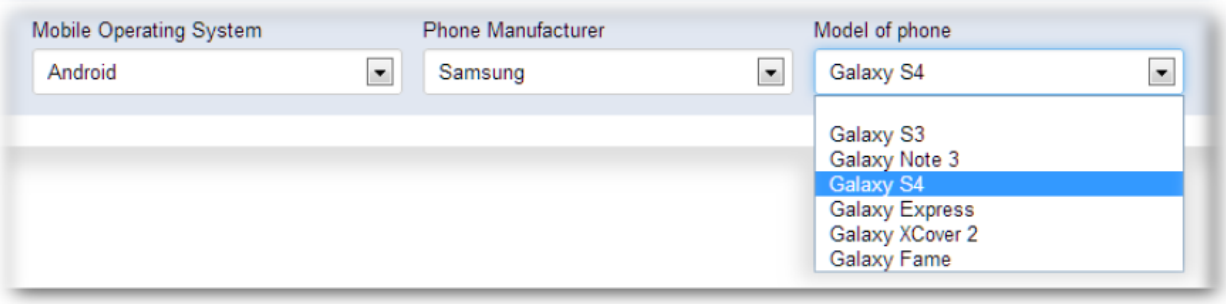
The dialog for creating the "model" level cascading dropdown list.

For convenience we have kept the lower two list widgets on the same line as the "os" top order. The final structure looks as follows:



The final structure on the form.

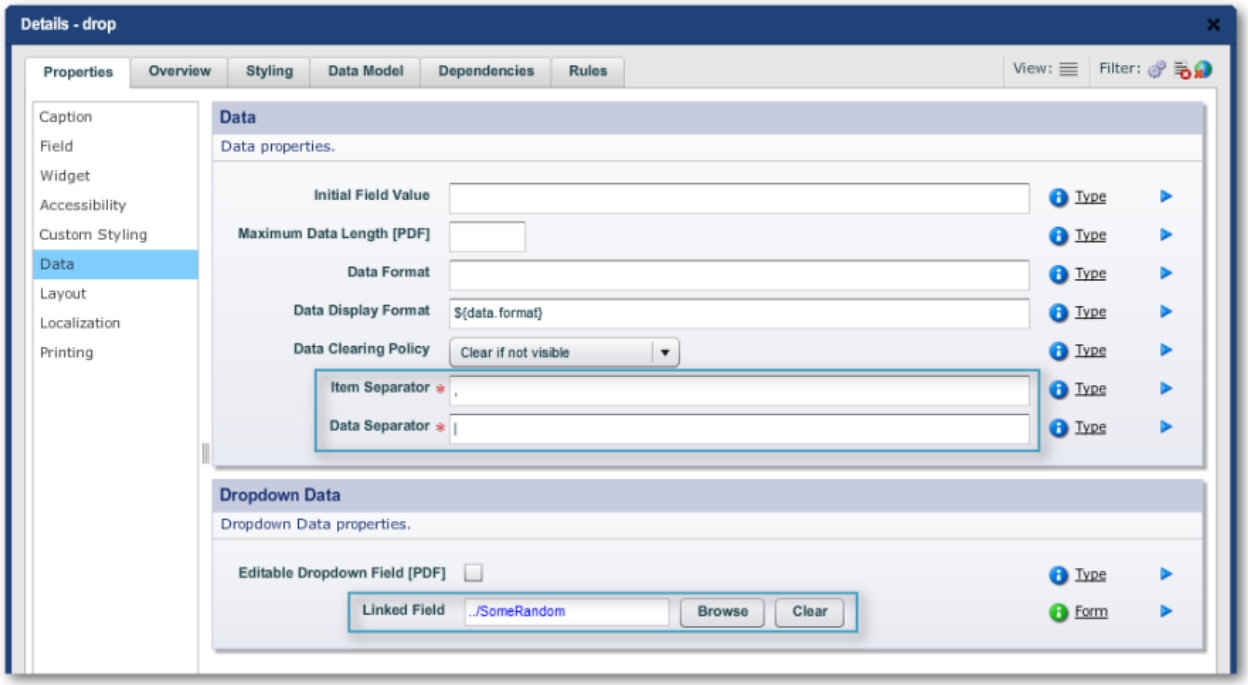
And here is the preview rendition:



Preview of the cascading dropdowns.

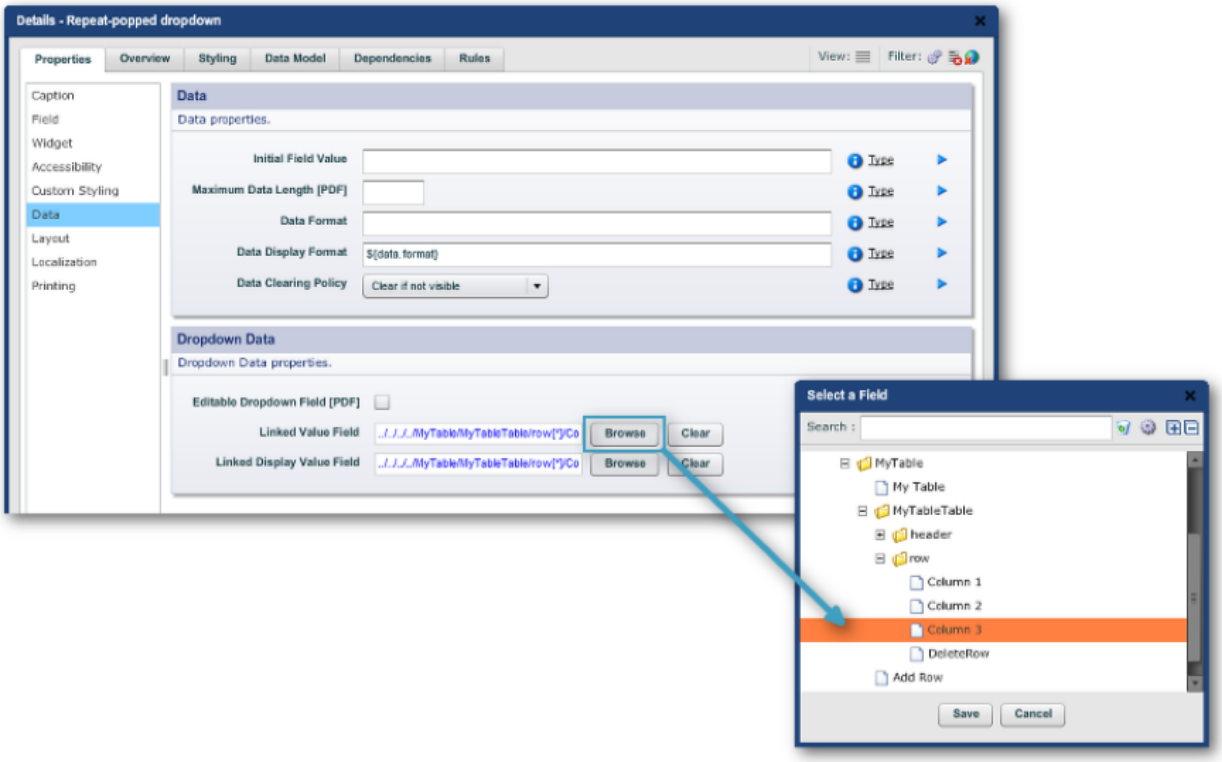
Prefilled Dropdown Lists

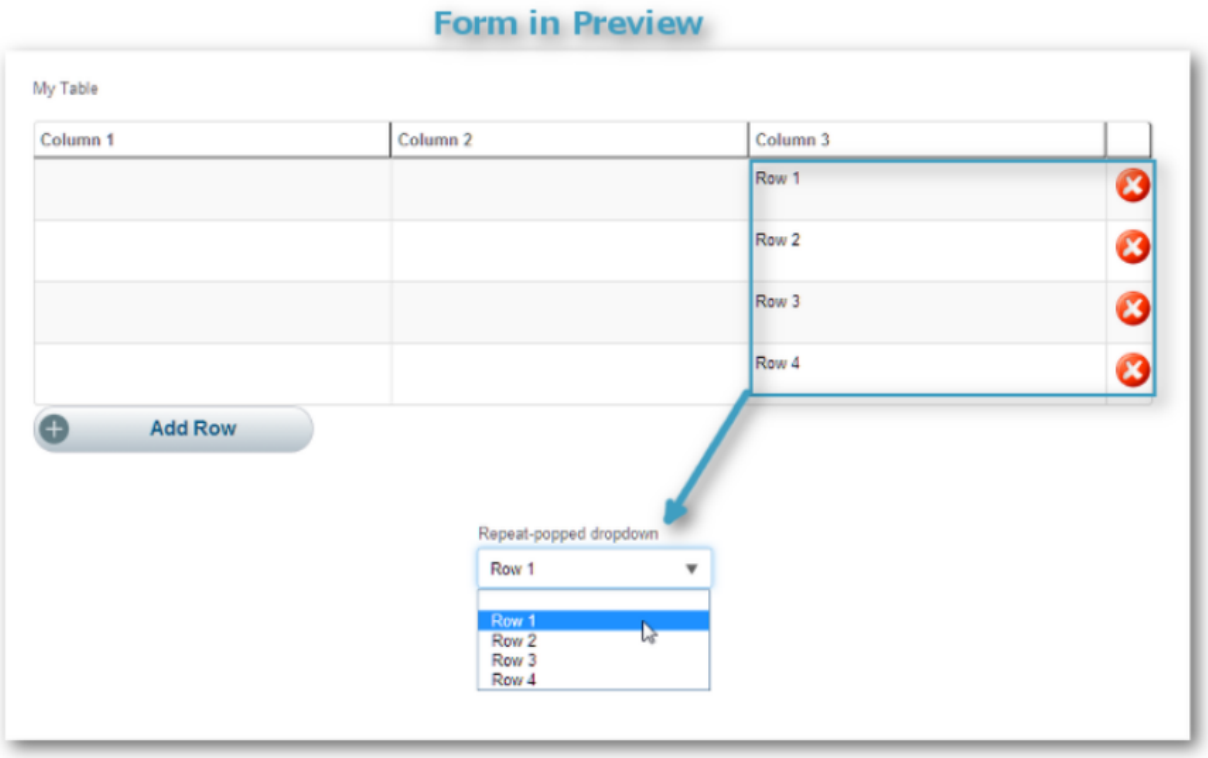
The Populating Dropdown List widget allows a Dropdown Field to be linked (via the Linked Field property) to a data field (created by dropping "Widget Palette -> Internal -> DataField" onto the form's structure. TM is then configured (in TM, "Home Dashboard -> Form -> Form Data Config link -> Property Prefilled Mapping tab -> Edit button") to prepopulate the data field with values. The default is for TM to give the data as a string using the default separators in the Data Field's Edit Properties:



Prefilled Dropdown Lists from Repeatable Areas

The Prepopulating Dropdown List from Repeat widget creates a dropdown menu where the items are taken from a nominated column of a table or a repeat.





Repeats

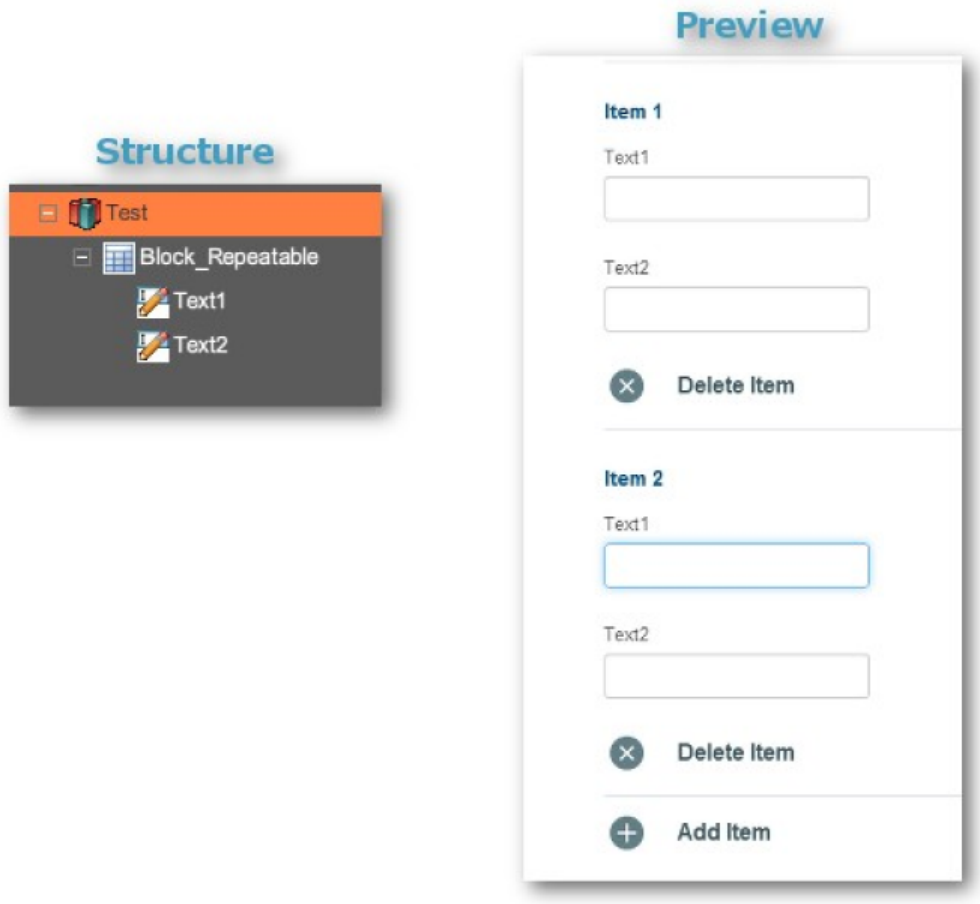
Composer | Form Builder | v4.3 & Higher

Overview

Repeats in earlier versions of Composer used to be complex. You could not copy a a whole repeat structure block to another location on the form without generating error messages and time was you had to use many work-arounds to get the new structure work. All these had to be documented for many pages. No longer, with Composer Version 4.

Repeats, then, are a more flexible repeating structure than Tables. Each "Row" in a repeat can be a layout block of fields. That means the row is not confined to a rigid grid-like structure, as with tables.

Otherwise, repeats are easy to construct. The simplest way is to use the predefined block "Repeating Block Template". Into it, insert a "Block [Repeatable]" and populate it with the fields you want repeated. The template gives you Add and Delete buttons for users to dynamically add or delete blocks. You can also use Edit Properties on the block template to change layout and the wording of the buttons.



We encourage you to experiment with the widgets in Repeats and Tables and discover how powerful is this approach.

Advanced Scripting

Composer | Form Builder | v4.3 & Higher

JavaScript in Composer

Composer uses JavaScript as its scripting language. The only differences are some Composer-related libraries which contain Composer classes to, for example, reference fields in a Composer form or for other Composer integrations.

This section of the Guide focuses on form input scripting. Elsewhere we will cover the XML code used to tweak the format the form's output for some special purpose (such as to accommodate some integration point with the Enterprise's backend).

Syntax

The syntax for Composer's scripts is that of JavaScript. JavaScript is maintained by the Mozilla Foundation. Here are the canonical references:
{+}https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide+{+}https://developer.mozilla.org/en-US/-docs/Web/JavaScript/Reference+
Other authoritative references are:
{+}http://www.w3schools.com/js/js_syntax.asp+{+}http://www.w3schools.com/jsref/default.asp+{+}http://www.-tutorialspoint.com/javascript/javascript_syntax.htm+{+}http://javascript.info+{+}http://www.-java.com/en/download/faq/java_javascript.xml+

"Me" Instead of "This"

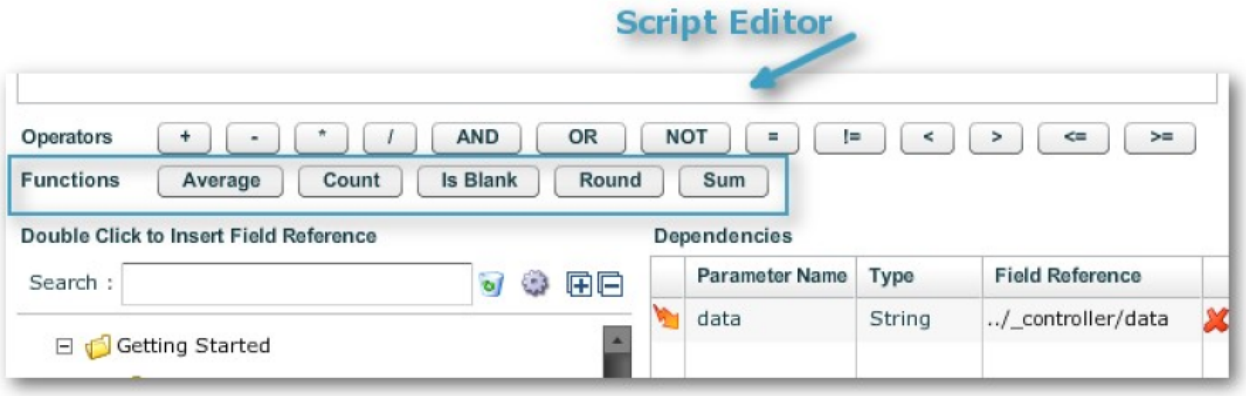
In JavaScript, the "this" object generally refers to the current object receiving the event. Never use the "this" object in Composer scripts.
All Composer JavaScript (for rules, click action scripts, etc.) are embedded in a function, rather than being coded directly under an event. (The reasons for this are beyond this document's scope.) The commonly used "this" object, which refers to the current object, is not available in functions. Instead, Composer provides a "me" object. The me object refers to the object where the business rule is located.
If you have some existing JavaScript code that works correctly in either PDF or HTML, you can generally simply replace every occurrence of "this" with "me" for the same code to work in Composer.

Other Departures from Standard JavaScript

If you want to reuse a legacy script in Composer and want it to work correctly across platforms, you must modify it to use the technology-independent Composer Framework JavaScript library instead of the native libraries. For example, use sfc.getRawValue(me) rather than me.rawValue. See the Avoka Composer Framework link on Composer's Welcomepage.
Here, in the context of Transact Composer, "across platforms" specificity means that the sfc calls work for both PDF and HTML.

Classes and Functions

Many of the Composer functions, which are defined in sfcFunctions.js library, have been designed for your convenience — to accomplish a lot with little code. Many of these are embodied in the function buttons in the ScriptEditor.



The function buttons insert the corresponding function code into the Script tab's text window. That code ("$SUM(...)" for example) gets expanded further at runtime (to "sfcFunction.sum(...)" for example

We will also use the terminology "sfc classes" interchangeably with "Composer functions". We will mention here a few of these sfc classes created for you convenience:

sfc.copyValues (srcBlockNodeRef, targetBlockNodeRef);

though note that this does not work recursively, i.e. to blocks within blocks.

Composer also uses conveniently has a number of scripting conventions, such as parameters, which reduce the amounts of code needed to refer to form elements and their values. For more, see the Composer Framework.

Syntax Checking

There are a number of JavaScript tools on the web for checking syntax:

{+}http://www.javascriptlint.com/online_lint.php+

A simple webpage, where you paste your JavaScript fragment into a textarea field and you get back a report of any simple syntax errors. Note that this tool is somewhat limited in scope, but this makes it very useful for quick checks.

Most modern browsers have JavaScript tools {+}https://developers.google.com/chrome-developer-tools/-docs/javascript-debugging+ documentation for Chrome's inbuilt checker {+}https://developer.mozilla.org/en-US/docs/Tools/Debugger+{+}https://developer-.mozilla.org/en/docs/Debugging_JavaScript+ These document Firefox's inbuilt Web Console tool {+}https://developer.apple.com/library/mac/documentation/AppleApplications/Conceptual/+Safari_Developer_Guide/Debugger/Debugger.html

The documentation for Apple's Safari on OS X

Composer Framework

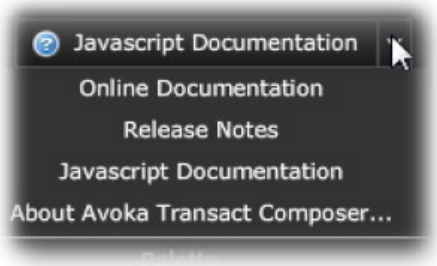
The JavaScript Libraries

Composer has several JavaScript libraries. These are exposed in the Advanced Mode of the Structure Panel ("<Form> -> Composer Framework ->")
formBridge.js (included in PDF forms) htmlFormBridge.js
sfmSupport.js which supports
o sfc.js
This defines the methods in the Composer public API. json.js
for JSON support.

The Public API

Documentation

You can access the API documentation at anytime in Composer, through the Online Documentation dropdown at the top right-hand corner of the Composer User Interface.



Select the "Javascript Documentation" option.
This opens a webpage in a new tab in your browser, listing the methods in the public API. (Avoka does have a set of internal APIs, but these are subject to frequent change. We strongly recommend not using these.)

Calls

The sfc calls are designed to work both for HTML and PDF. This is the reason sfc framework originally came into being.
Most of the sfc methods are static: you just call the method, specify the parameters and the method returns some value. Some typical examples:
sfc.getDaysDifference(startDate, endDate) sfc.convertToString(value) sfc.formatDate(dateObject, dateFormat) sfc.-getRawValue(noderef)
Others are dynamic, such as the sfcInstanceManager which operates on a delegate (i.e. an event that is being listened to). You also can make use of Composer's BusinessRules widgets to listen to events.
However, Composer provides "parameters" which are convenient constructs to refer to form objects with a min-imum of coding.

Using JavaScript's Built-in Functions

You can use the standard JavaScript functions, though you should make preferentially use of any sfc method instead of the standard JavaScript method as a general rule. But there are far more JavaScript functions for you to draw on.

There are many references for these:

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objectshttp://www.w3schools.-com/jsref/jsref_obj_global.asphttp://www.tutorialspoint.com/javascript/javascript_builtin_functions.htm

For example:

```
var n = 12345;
var x = n.toExponential(); // x will be 1.2345e+4
```

Parameters

In the Script Editor

The above examples of sfc static methods calls have a number of parameters, such as "startDate", "endDate" and so forth. Composer' Script Editor can conveniently refer to fields on the form for these calls.

Refereing to form elements becomes a simple matter of point-and-shoot to elements in the Structure Panel on the bottom-left of the Script Editor. The field is referenced in the JavaScript in the scripting text field by its field name in curly brackets. The Dependencies panel in the dialog gives the Parameter Name (the name within the curly brackets), the data type (about which, more later) and the Field Reference (which is the field's position in the structure, relative to the field for which the script is being created).

So, we could use an sfc call on such a parameter:

```
sfc.convertToString({Amount1})
```

You should not merely type the parameter name into some curly brackets in the script: the editor works best if you insert your cursor into the point of the JavaScript and double-click on the parameter listed in "Dependencies".

A useful byproduct of using these Script Editor parameters is that Composer is able to keep track of form elements when they are moved within the form, so that you do not have to re-point to these elements in their new locations.

Edit Parameter Data

Clicking on the Edit icon in the right-hand column of the Dependencies panel (see immediately above) brings up the Edit Parameter Data dialog where you can:

edit the parameter name (i.e. the name that goes into the curly brackets Choose the parametertype via a drop-down:

o string

- Boolean

o number

- Is Selected

o node
o raw value o UID Trigger Type o Static

- Dynamic

We will discuss the Trigger Type further in OptimizingPerformance.

Parameter Types

Parameter Type	Explanation
RawValue	Always returns a String representing the "raw" value of an object. Does no conversion, so it returns dates, booleans and other values as the underlying string that the value is stored as. Uninitialized values may return a null.
String	Always returns a string. Nulls will be converted to empty strings. (This gives you one less thing to "worry" about in your calculations - you don't have to test for both null and empty string.)
Number	Always returns a number. It will do its best to convert any parameter to a number. If it cannot convert it, it will return zero.
Boolean	Takes anything that resembles a true value, and return the boolean value "true". This includes "1", "T", "true", "Y", "yes", "on", and is case insensitive. Otherwise it returns the boolean value "false". Blank or Null is false.
IsSelected	Only used for checkboxes and radio buttons. Returns Boolean true if the check-box/radio-button is selected.
Node	Returns the internal node object that represents an object. If you point at anything in a repeat from outside the repeat, this returns null (because it's really multiple nodes). If you refer to any object inside a repeat from within the same repeat, it returns the node. Nodes can be passed to internal SFC functions to perform operations on the object itself.
UID	Provides a reference to a node. Similar to a node, but can be used to reference an object in a particular row within a repeat.

Node Reference

There are three ways to refer to a node (or "form element", the terminology we have been using consistently so far):
The Script Editor parameter
the relative path (as seen in the "Field Reference" column value in the Dependencies panel) the Full Path (or "Absolute Path" from the form's root, usually by default "$record")
The field's UID (Unique Identifier)

Obtaining the Fullpath

If you do not avail yourself of the convenience of this point-and-shoot construct, you will have to refer to the form field using the Data Model tab by double-clicking on the field in either the left- or right-hand panels and copying the "Fullpath" from the Data Model Binding panel in the properties dialog.
For example
$record.GettingStarted.FirstName.data

Obtaining the UID

You could get this from "Edit Properties -> Overview tab -> Basic Information panel -> field UID"
For example:
/GettingStarted/_outerArea/_contentArea/FirstName/data
Or you can use the sfc call: sfc.convertToUid(noderef)

Data Types

Overview

Data Typing is used throughout Information Technology. In Composer, there are two sets of data types:
Composer's own typing, as embodied in parameterdatatypes

- which are used by Composer internally. Some of these are exclusive to Composer:
  - Node
  - UID
  - Is Selected JavaScript'styping
- not to be confused with the Composer types, though some do resemble Composer's

The Composer types, as you would expect, were designed to handle data for both HTML and PDF forms.
JavaScript is looser in its data typing, which means that you usually do not explicitly specify some data's type. Circumstances, however, may require you to type data in order to get desired results.

JavaScript Types

JavaScript has less types than Composer. For example, there is no such thing as a Date type. All JavaScript has are:

String

which is a sequence of characters, usually delimmited by straight quote marks for example: "To be or not to be"

Number

and there is no distinction made between inegers or decimals for example: 2, 3.142, 123e5 (scientific notation for 12300000)

Boolean

that is: true or false **JavaScript arrays JavaScriptObjects**
for example: JSON objects

Undefined

A variable with no value

Null

An empty variable
Note: undefined is not the same type as null. This is a subtle distinction, but it does have relevance to Composer scripting, as in the a scripting example given in VisibilityinCollaboration.

Raw Data

This is a Composer internal data type. All data is held internally in Composer as a string of characters, be it a number, date, string and so on. This can be tricky: for example there is a difference between the integer 2, the floating point value 2 and the string "2". JavaScript handles the calculation of adding an integer to a floating point gracefully, but can mishandle adding an integer to a string.
Dates are a potential source of problems. Please refer to the Dates topic.

Conversions

Because of JavaScript's loose typing, the need to convert form values hardly ever arises for Script Editor parameters. Composer looks after this for you.
However, if you are writing your own field references manually, you may have to convert the field's value from its native raw data value. The Public API has a number of methods to do this:
sfc.convertToBoolean(value) sfc.convertToFloat(value) sfc.convertToInteger(value) sfc.convertToString(value)
Converts a number value (or Boolean?) into a Composer string data type. sfc.formatDate(dateObject, dateFormat)
see Dates sfc.getFormattedValue(noderef)
returns the value of the node according to its Composer data type sfc.getRawValue(noderef)
the opposite of the formatted value: retaurns the raw value, irrespective of the Composer data type.

Explicit Conversions

You can convert String values using the public API's functions (listedabove). For example:
var myvalue = sfc.convertToNumber(sfc.getRawValue(me));
This code will convert the raw value of the current field ("me") to a Number.

Implicit Conversion

JavaScript is a loosely typed language, and will attempt to convert data for you. The rules are a little complicated, which is why we suggest that you always try to convert to the appropriate type before you use them.

One simple example is a way to convert any data type into a string - simply concatenate an empty string to it. For example:

var var1 = 18; // var1 is a number

var var2 = var1+""; var2 is a string, because we concatenated "" to it.

Comparisons

Be careful when making comparisons between different data types. JavaScript will attempt to convert them for you, but the results may not always be the expected.

In JavaScript, the "this" object generally refers to the current object receiving the event. Never use the "this" object in Composerscripts.

All Composer JavaScripts are embedded in a function (see Advanced Scripting), rather than being coded directly under an event. (The reasons for this are out-of-scope.) The commonly used "this" object to refer to the current object, is not available in functions. Instead, Composer provides a "me" object. The "me" object refers to the object where the Composer script is located, i.e. the form element whose Edit Properties dialog holds the script.

If you have some existing JavaScript code that works correctly in either PDF or HTML, you can generally simply replace every occurrence of "this" with "me" for the same code to work in Composer.

Note: If you want the script to work correctly across different technologies, you must also rewrite it to use Composer's technology independent JavaScript library rather than the native libraries. for example, use sfc.getRawValue (me) rather than me.rawValue. See the Avoka Composer Framework.

Dates

Date Formats and Patterns

There are a number of different ways in which data is stored, displayed, edited and validated in Composer forms. These are mostly useful for dates and numeric values, which can be represented in different ways. When you are using Composer, you can specify these different formats by using data patterns, such as DD/ MM/YYYY or $#,###.##.

The different formats are:

Format	Description
Edit Format	This is the format that is used when the user types into the field. For example, 3/27/2012, or 49.99. It is possible to have multiple alternate edit formats - for example, allowing either 2 or 4 digit years.
Display Format	This is the format that is displayed when the user exits the field. It can be more readable than the edit format, and clarifies the interpretation. For example, 27th March 2012 or $49.99

Raw Format	This is the value that is actually stored within the form at runtime. It's the value you get from calling the getRawValue(). This is standardized to make programming easier • it doesn't matter what your formats are used for the form users, you always get a consistent value in any code you write. For dates, this is ISO format, such as 2012- 03-27.
XML Format	This is the format that is used to actually store the data in the XML file. Composer will use this format for pre-population data, and will convert to this format on submission.
Validation Pattern	PDF forms allow a separate validation pattern that is used to ensure that the data is valid. In Composer, the Validation Pattern is always identical to the Edit Pattern.

Usually, you don't need to be aware of these patterns because they are specified in your organizational stylesheets. However, you may need to occasionally override these patterns for particular fields.

In most cases, users will enter dates using pop-up date editors, and so date edit format patterns are not so important. However, some users, especially "power" users and visually impaired users may prefer to enter dates using the keyboard.

Recommendation for Dates

For dates, we recommend that the most useful Data Format is:
North America: date{M/D/YYYY}| date{M/D/YY}
Elsewhere: date{D/M/YYYY}| date{D/M/YY}
These format will:
accept single digit days and months with or without a leading zero (flexible)
accept 2 digit or 4 digit years (flexible)
for editing, will always display days and years without leading zeros (simpler)
for editing, will always display years including the century (no ambiguity) Working with Dates
Composer uses the Javalibrary moment.js (freely distributable under the terms of the MIT license) to handle dates.
The date clases in the Avoka Composer Framework are:
sfc.formatDate(dateObject, dateFormat)
where "dateObject" is is the raw value of the form element, sfc.getRawValue(someFormElement)
sfc.getDaysDifference(startDate, endDate) Where the startDate and endDate are raw values.
sfc.getTodaysDate()
Returns today's date as YYYY-MM-DD
This is not the generation date of the form (which you access in scripts with the property formula $GENERATION{DATE} instead)
This call returns only the calendar date. To get the time of day and timezone as well, use this native JavaScript call instead, where "x" will be your date object.
var x = new Date();
sfc.isLeapYear(year)
Where "year" is a 4-digit value. Returns true or false.
sfc.isPastDate(date)
Where "date" is the raw value. Returns true or false
sfc.isToday(date)
Where "date" is the raw value.
sfc.parseDate(dateString, dateFormat)
Parses a date according to the given date format. Both parameters are strings. Returns a raw value.

Advanced Date Output Formats

The sfc methods for date handling depend on the Moment.js library. This JavaScript library can output dates in various formats, as specified by this table:

Input	Output
M, MM	Month Number (1 - 12)
MMM, MMMM	Month Name
Q	Quarter (1 - 4) – sets the month to the first month in that quarter
D, DD	Day of month

Do	Ordinal day of month (from 2.6.0)
DDD, DDDD	Day of year
d, dd, ddd, dddd	Day of week (NOTE: these formats only make sense when combined with "ww")
e	Day of week (locale) (NOTE: these formats only make sense when combined with "ww")
E	Day of week (ISO) (NOTE: this format only make sense when combined with "WW")

Input	Output
w, ww	Week of the year (NOTE: combine this format with "gg" or "gggg" instead of "YY" or "YYYY")
W, WW	Week of the year (NOTE: combine this format with "GG" or "GGGG" instead of "YY" or "YYYY")
YY	2 digit year (see below)
YYYY	4 digit year
gg	2 digit week year (if greater than 68 will return 1900's, otherwise 2000's)
gggg	4 digit week year
GG	2 digit week year (ISO) (if greater than 68 will return 1900's, otherwise 2000's)
GGGG	4 digit week year (ISO)
a, A	AM/PM
H, HH	24 hour time
h, hh	12 hour time (use in conjunction with a or A)
m, mm	Minutes
s, ss	Seconds
S	Deciseconds (1/10th of a second)

SS	Centiseconds (1/100th of a second)
SSS	Milliseconds (1/1000th of a second)
Z, ZZ	Timezone offset as +07:00 or +0700
X	Unix timestamp
LT, L, LL, LLL, LLLL	Locale dependent date and time representation

You parse an input date with sfc.parseDate (dateString, dateFormat), using the formats in the Input column of the table immediately below. The output of this call gives you a properly formatted raw date format.

You can use these to build the full input format specification, using other characters as punctuation. For example: YYYY-MM-DD or YYYY/MM/DD or DD/MM/YYYY or MM/DD/YYYY. Internally, Composer's raw date format is YYYY-MM-DD, or for more precision to include time of day, timezones along with the date, it uses the ISO DateFormat.

You also can control how a date object displays using sfc.formatDate(dateObject, dateFormat), using the formats displayed in the right-hand "Output" column. As with inputs, you can concatenate these together using characters as punctuation. For example: "MMMM Do, YYYY, HH:mm a, timezone Z" formats as "May 19th, 2014, 13:27 pm, timezone +10:00", where the Z specifies the timezone as the timezone offset from GMT.

To return the precise time and timezone, use

var x= new Date;

and get the formatted time with our sample format with the call

sfc.formatDate(x, "MMMM Do, YYYY, HH:mm a, timezone Z")

Regular Expressions

Overview

A regular expression is a sophisticated, industry-standard way of specifying a validation pattern. An explanation of regular expressions is beyond the scope of this document, but there are numerous books and web sites that you can refer to. You can also use search engines to find standard regular expressions for common types of data. An example of some regular expression patterns are:

^[a-zA-Z0-9_-\.]@[a-zA-Z0-9_-\.]\.[a-zA-Z]{2,4}$ the pattern for email addresses

^http\://[a-zA-Z0-9-\.]+\.[a-zA-Z]{2,3}(/\S*)?$

the pattern for HTTP:// URLs

RegEx References

https://developer.mozilla.org/en/docs/Web/JavaScript/Guide/Regular_Expressionshttps://developer-.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/RegExphttp://www.regular-expressions.info/javascript.html
http://regex101.comhttp://regexpal.com
A good resource for regular expressions is: http://regexlib.com/.

Online regular expression editor and evaluator can be found at: http://gskinner.com/RegExr/ and http://www.debuggex.com/.

How to Use Regular Expressions

There are two ways to use patterns in JavaScript:
var patt = new RegExp(pattern,modifiers); or more simply:
var patt = /pattern/modifiers;
The modifiers (also called "flags" in the documentation) are:

g

global match

i

ignore case

m

multiline; treat beginning and end characters (^ and $) as working over multiple lines (i.e., match the beginning or end of each line (delimited by \n or \r), not only the very beginning or end of the whole input string)
y
sticky; matches only from the index indicated by the lastIndex property of this regular expression in the target string (and does not attempt to match from any later indexes). This allows the match-only-at-start capabilities of the character "^" to effectively be used at any location in a string by changing the value of the lastIndex property.
In other words, in most cases where you are not using a modifier, just use the pattern as follows:
/^[a-zA-Z0-9_-\.]@[a-zA-Z0-9_-\.]\.[a-zA-Z]{2,4}$/
in your JavaScript code.

Using Property Formulae

Overview

The following scripting constructs, the property formulae, only apply while the form is being generated. When generation is complete, the fom is then delivered to the user, either as HTML the browser or as XFA to Acrobat reader (or some other reader) or to a pdf browser internal module or plugin.
These property formulae allow you to query the build environment, to inspect a form field, to find a field in the form hierarchy, perform some string manipulations, and other useful functions.
They do not trigger anything while the form is running and only affect the time it takes to serve a form.

$BUILD

Call	Result
$BUILD(ENVIRONMENT)	Returns the values of the called arguments: ENVIRONMENT VERSION REVISION
$BUILD(VERSION)	
$BUILD(REVISION)	

$CHAR

Call	Result
$CHAR{COMMA}	Returns a comma character ','
$CHAR{DOT}	Returns a dot character '.'
$CHAR{SLASH}	Returns a slash character '/'

$CHAR is a convenience to specify characters, in a text field for example, that would normally are parsed in JavaScript.

$DATA

Call	Result
$DATA(ROOT)	Returns the root element of the form: E.g. "/AvokaSmartForm"

$ENCODE

Call	Result
$ENCODE{SCRIPT,value}	Encodes the value so it can placed in a javascript string literal
$ENCODE{RICHTEXT,value}	Replaces richtext markup with its form specific implementation

$ENCODE{URL,value}	URL encodes the supplied value
$ENCODE{XML,value}	Encodes the value with XML markup entities
$ENCODE{STRIP,value}	Removes richtext markup
$ENCODE{TRIM,value}	Removes cr, lf characters, leading and trailing spaces
$ENCODE{UPPER,value}	Converts the value to uppercase
$ENCODE{LOWER,value}	Converts the value to lowercase
$ENCODE{FNNAME,value}	Converts the value to a valid javascript function name by replacing invalid chars with _

Note: $ENCODE is unique among the property formulae, in that you can specify multiple values, pipe- delimited. For example: "$ENCODE{TRIM| UPPER,value}", will first trim the string (i.e. remove any leading or trailing space characters) offered as the "value" and then convert the characters to upper case. The order of these pipe-separated arguments is important, as changing the order of them my affect the output. Note that these are executed from left to right.

$FIELD

Call	Result
$FIELD{TYPE}	returns the type of the current field

Call	Result if any

$FIELD{TYPE,relativeUid}	returns the type of the field at the relative uid
$FIELD{TYPE,relativeUid,type Name}	returns 'true' if the type of the field at the relative uid is an instance of the supplied type name
$FIELD{VALUE}	returns the value of the current property of the current field
$FIELD{VALUE,typeName}	returns the value of the current property as defined in the specified type
$FIELD{RBTYPE}	For radio buttons only, indicates whether the buttons value is controlled by the group or by the button returns 'group' for group controlled buttons (case 1&2) returns 'button' for button controlled buttons (case 3)
$FIELD{RBTYPE,buttonType}	For radio buttons only returns true if the button type (as deternmined by $FIELD{RBTYPE}) matches the supplied type.
$FIELD {INSTANCEMANAGER}	returns the relative uid of the closest parent instance manager or blank if not found
$FIELD{NESTLEVEL}	returns the number of repeat instances this field is nested inside of
$FIELD{INDEX}	returns the index of the field as a child of its parent
$FIELD{REVERSEINDEX}	returns the reverse index of the field wrt its parent
$FIELD{ISFIRST}	returns true if this field is the first child of its parent
$FIELD{ISLAST}	returns true if this field is the last child of its parent
$FIELD{CHILDCOUNT}	returns the number of child fields
$FIELD{HASCHILDREN}	returns true if the field has at least 1 child
$FIELD {RESOURCE,resName}	returns the contents of the given resource applying formula substitutions with the context of the current field.

$FIND

Call	Result
$FIND{propertyname}	Searches from the current context up the hierachy for a property with the given name

Call	Result
	and returns the value of the first one found. If a property isn't found then a blank string is returned.
	E.g. $FIND{section.name} returns the section name from within any child field of the section
$FIND{path,propertyname}	Sames as $FIND{propertyname} but the upwards search begins from the field found at the relative path provided. If no field can be found at the supplied path then a blank string is returned.
	e.g. $FIND{../../someField,label}

$FLOAT

TBA

$FORM

Call	Result
$FORM(NAME)	Returns the form's values of the called argument
$FORM(DESCRIPTION)	
$FORM(RELEASE)	
$FORM(TEMPLATE)	

$GENERATION

Call	Result

$GENERATION(TECHNOLOGY)

$GENERATION(FLAVOUR)

$GENERATION(FLAVOR)

$GENERATION(TARGET)

$GENERATION(VARIANT)

$GENERATION(UUID)

$GENERATION(TIMESTAMP)

$GENERATION(DATE)

$GENERATION(DEBUG)

$GENERATION(DEBUGSCRIPT
)

$IF

TBA

$INFO

Call	Result
$INFO(USER)	Returns the form's info values of the called arguments
$INFO(ACCOUNT)	
$INFO(CLIENT)	
$INFO(ORGANISATION)	
$INFO(ORGANIZATION)	

Call	Result
$INFO(PROJECT)	
$INFO(SERVER)	

$INFO(ENVIRONMENT)

$LIST

Call	Returns
$LIST{CONTAINS,list,value,separator}	Returns true if the list contains the value
$LIST{INDEXOF,list,value,separator}	Returns the 0 based index of the value within the list
$LIST{ITEMAT,list,index,separator}	Returns the item at 0 based index position within the list
$LIST{SIZE,list,separator}	Returns the size of the list

$LOCALE

Call	Result
$LOCALE(LANGUAGE)	Returns the form's locale values of the called arguments
$INFO(COUNTRY)	
$INFO(CODE)	

$LOOKUP

TBA

$MATH

Call	Result
$MATH{PX,value}	Converts the value to pixels and strips the units
$MATH{IN,value}	Converts the value to inches and strips the units
$MATH{MM,value}	Converts the value to millimetres and strips the units
$MATH{MAX,-value,...}	Returns the largest value - if the first value has units then the result is returned in those units

$MATH{MIN,-value,...}	Returns the smallest value - if the first value has units then the result is returned in those units
$MATH{SUM,-value,...}	Returns the sum of the supplied values - if the first value has units then the result is returned in those units
$MATH{PRODUCT,-value,...}	Returns the product of the supplied values (multiplies them together) - if the first value has units then the result is returned in those units
$MATH{HALF,-value}	Returns half of the supplied value - if the value has units then the result is returned in those units
$MATH{TWICE,-value}	Returns twice of the supplied value - if the value has units then

Call	Result
	the result is returned in those units
$MATH{MARGIN,value}	Converts a composer margin to a technology supported margin

$RULE

Call	Result
$RULE{TRIGGER,rulename)	Returns the trigger width in pixels for the given rule

$SEQUENCE

Call	Result
$SEQUENCE{Sequence argument, Sequence argument1, ..., RESET)	Resets the named sequences.

$STRING

Call	Result
$STRING{CONTAINS,value,pattern}	Returns true if the value contains the pattern as a substring

$STRING{INDEXOF,value,pattern}	Returns the 0 based index of the pattern within the value or -1 if not found.

$TEST

Call	Result
$TEST{BLANK,value}	Returns true if the supplied value is blank
$TEST{NOTBLANK,value}	Returns true if the supplied value is not blank
$TEST{EQUALS,value1,value2}	Returns true if value1 and value2 are the same
$TEST{NOTEQUALS,value1,value2}	Returns true if the value1 and value2 are not the same
$TEST{NOT,value}	Returns true if value is the literal false otherwise returns false
$TEST{AND,value1,value2}	Returns true if value1 and value2 are both the literal true
$TEST{OR,value1,value2}	Returns true if value1 or value2 are the literal true
$TEST{LESSTHAN,value1,value2}	Returns true if value1 as a number is less than value 2 as a number
$TEST{GREATERTHAN,value1,value2}	Returns true if value1 as a number is greater than value 2 as a number

Optimizing Performance

Composer | Form Builder | v4.3 & Higher

Overview

Forms in Composer are smart: they are dynamic and user actions in filling out the forms trigger events and change the form on-the-fly as users enter data. For example, a user can click on a checkbox and a number of fields become visible as a result. Or users fill in numerical values in a table column and a total is displayed in a field at the foot of the table.

Many forms in the real world elicit a great amount of information from the end users, with the consequence that a number of events ripple through the form as users complete entry into individual fields.

Why are there so many events on these smart forms? They are there to:

ensure that, as a default, all the data and all the dependencies on the form operate with complete integrity all of the time

ensure that all validation rules and scripts have occurred

ensure that any changes made at any time during the filling out process to any fields in the form affect all objects on the form

This design approach means that calculations, scripts and dependencies produce predictable and consistent results and that the data presented to end users is accurate and up-to-date. These take place without the form's designer doing any further work. But, as is often the case with a "one size fits all" approach, there are trade offs in large, complex forms:

an increase in generation time for very large forms an increase in the rendering time of the form

increased lag for the user to move from one field to another in such forms

increased lag when some action takes place, such as navigating through the form, submitting, saving it.

Form Generation

A full discussion of this topic is beyond the scope of this guide. The form's HTML is not statically stored in Transaction Manager and just downloaded to the end user. Instead, a number of processes take place:

the form's dependencies are inspected, resulting in a hierarchy of triggering and firing the form is prepopulated with data while being served

server-side processes to track the form from its being served to its outcome (submission, saving, abandonment, anonymous sharing or progression through a collaboration)

and even all the web server processes to deliver all the assets of the form to the user's device

Form Rendering

This is the time it takes for the device to render the form in either the browser or in the Mobile App. The form is served to either as HTML5 code interspersed with JavaScript. The JavaScript code is not merely the scripts you have placed into some of the form's fields through the Script Editor: there is also much code to integrate the form and your scripts together, as well as the functioning of the other form widgets. All these scripts and HTML have to load and initiate on the device. Most mobile devices are slower than desktop devices — although they are getting faster.

Be aware that the whole form downloads to the device, not just the visible or editable parts of the form. The same is true for all the prepolutation data, all of the form's autosuggest data (as distinct from such data provided by an external dynamic data service) and even mobile device responsive elements normally not visible on the desktop. All of these affect the rendering times, but the good news is that the increase in processing speeds of devices, and the huge performance increases in modern browsers, rendering times are no longer the problem they once were.
In the end, you must always take the user experience into account, and this may lead you to simplify your forms as a general rule.

Speeding up Generation and Rendering

It follows that these two, generation and rendering, depend mostly on the complexity of the form itself. A couple of settings in Composer have a minor influence on form generation (as we will see below). The form rendering times for the most part on the amount of HTML and JavaScript code to be processed. Optimization of rendering really means simplification of the form — just as with any other web page.

Optimization Strategies

Composer now has tools to improve the performance of forms while they are in use. Most forms of medium to low complexity require no optimization at all and run fast enough for users to have no issues. This may not be the case with very complex forms with a great deal of dependencies.
There are a number of possible strategies for tuning complex forms to run faster:
Tuning the dependencies
Tuning the triggers that make scripts fire Tuning the events that make scripts fire
Tuning the observers watching form objects and events Use of the Business Rules widgets on the form

Tuning Dependencies

Inspecting Dependencies

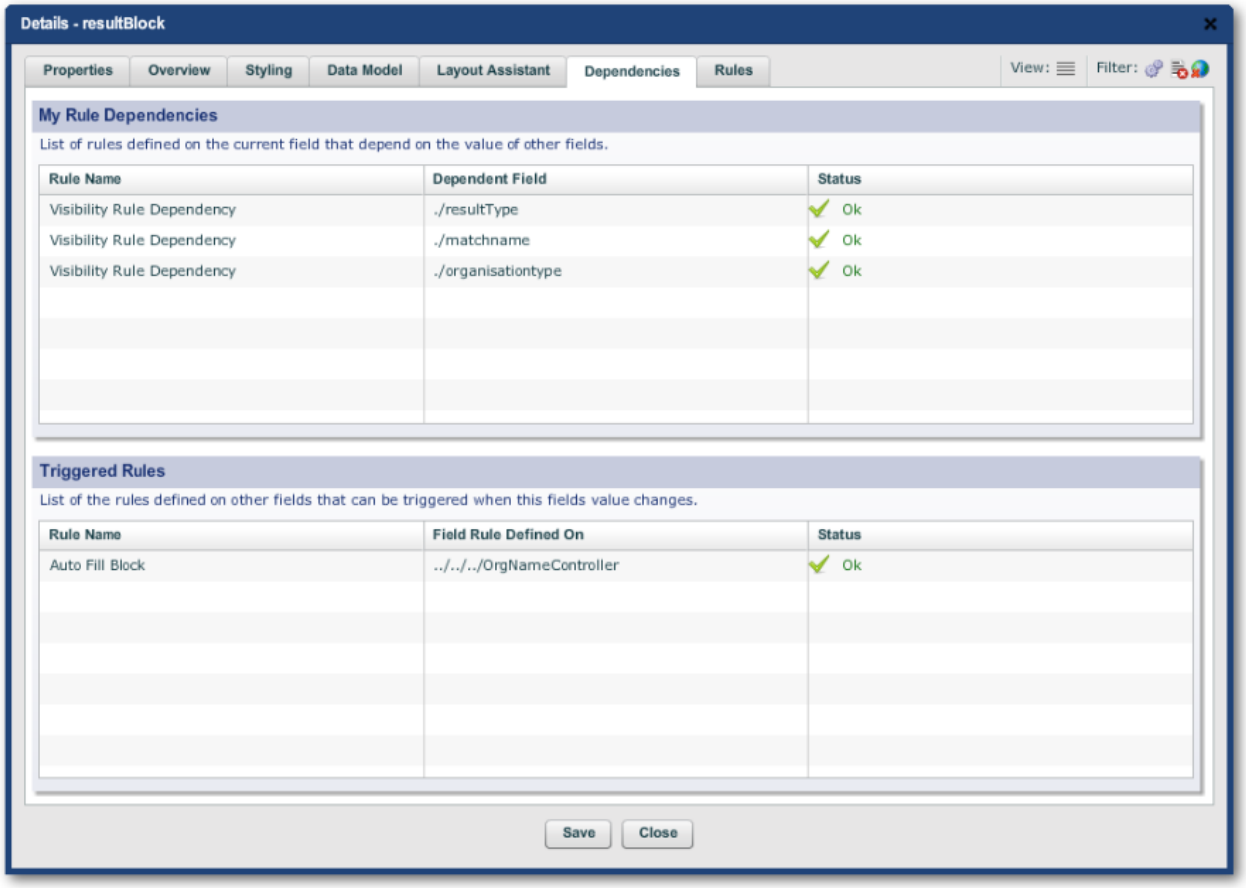
The Edit Properties dialog now allows you to inspect a form element's dependencies through "Edit Properties -> Dependencies tab". The tab has 2 panels:

My Rule Dependencies

which lists the form elements on which depend this element's the Rules or Scripts. In other words, this panel shows the rules, scripts that fire and affect this element's value upon the change of any of the elements listed.

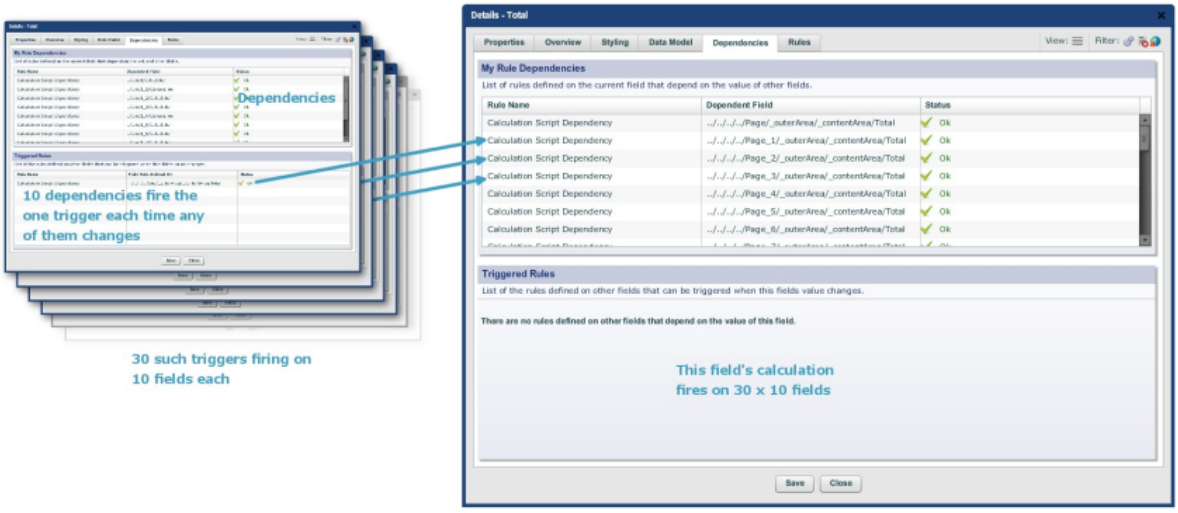
Triggered Rules

which lists the rules or scripts which fire upon this element's change.



Here is an extreme case, where we have deliberately created so many dependencies on the form that moving from field to field takes more than 10 seconds each time. The form has 30 wizard-style pages. Each page has a static tables of 10 rows, 3 columns each. Each row has a calculation, in the 3rd column, of column 1 multiplied by column 2. At the bottom of each page is a subtotal of all 10 calculation fields. The last page of the form is a total of all 30 sub-totals.

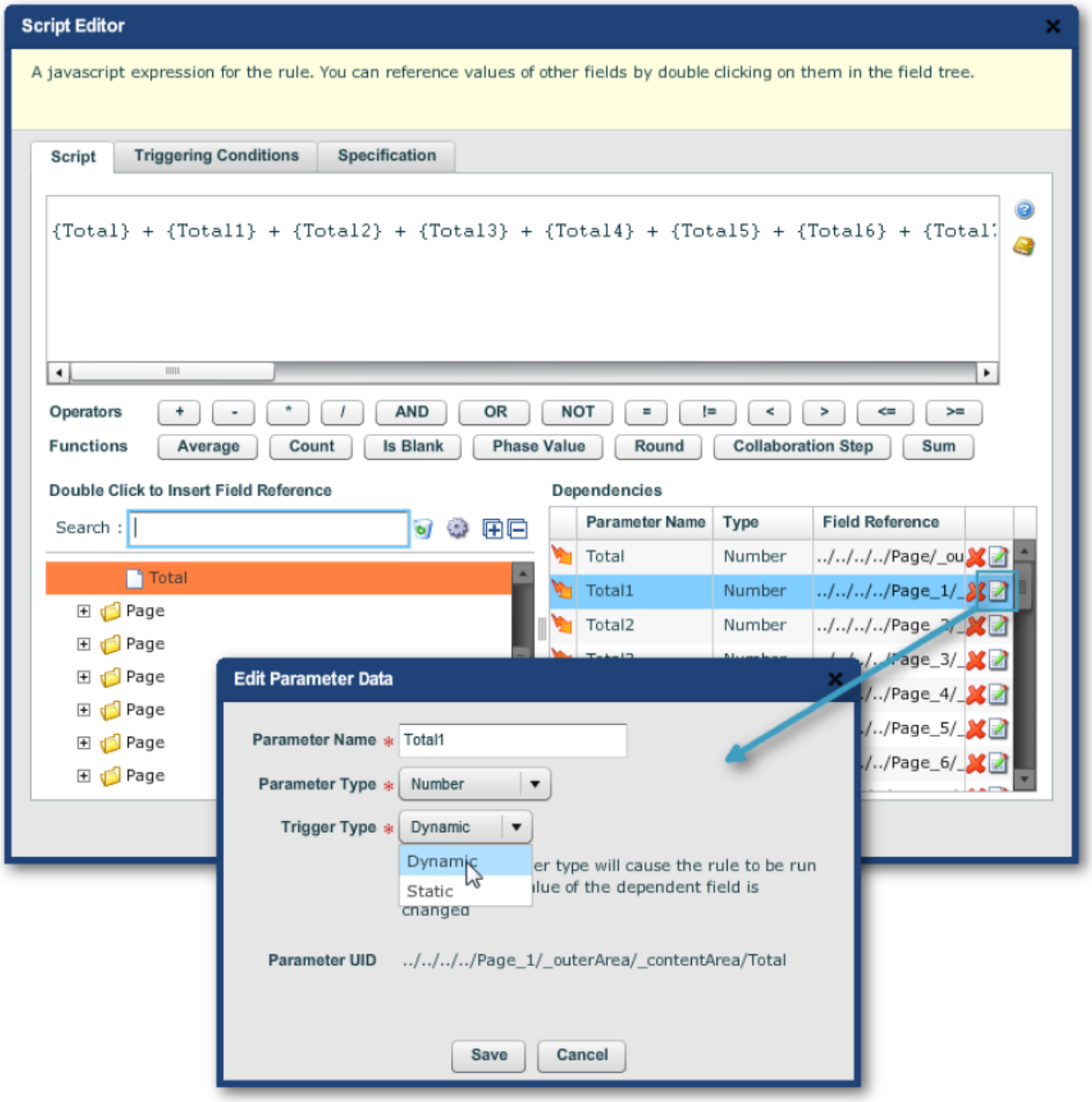
This means that the total on the last page is triggered by any changes in some 91 times 30 fields, i.e. the field is triggered by changes to any one of 2,730 fields. Let us focus on just the subtotals triggering the total:


Dependencies
10 dependencies fire the one trigger each time any of them changes
30 such triggers firing on 10 fields each
Details - Total
Properties
Overview
Styling
Data Model
Dependencies
Rules
My Rule Dependencies
List of rules defined on the current field that depend on the value of other fields.
Rule Name
Dependent Field
Status
Calculation Script Dependency
Ok
Triggered Rules
List of the rules defined on other fields that can be triggered when this fields value changes.
There are no rules defined on other fields that depend on the value of this field.
This field's calculation fires on 30 x 10 fields
Save
Close

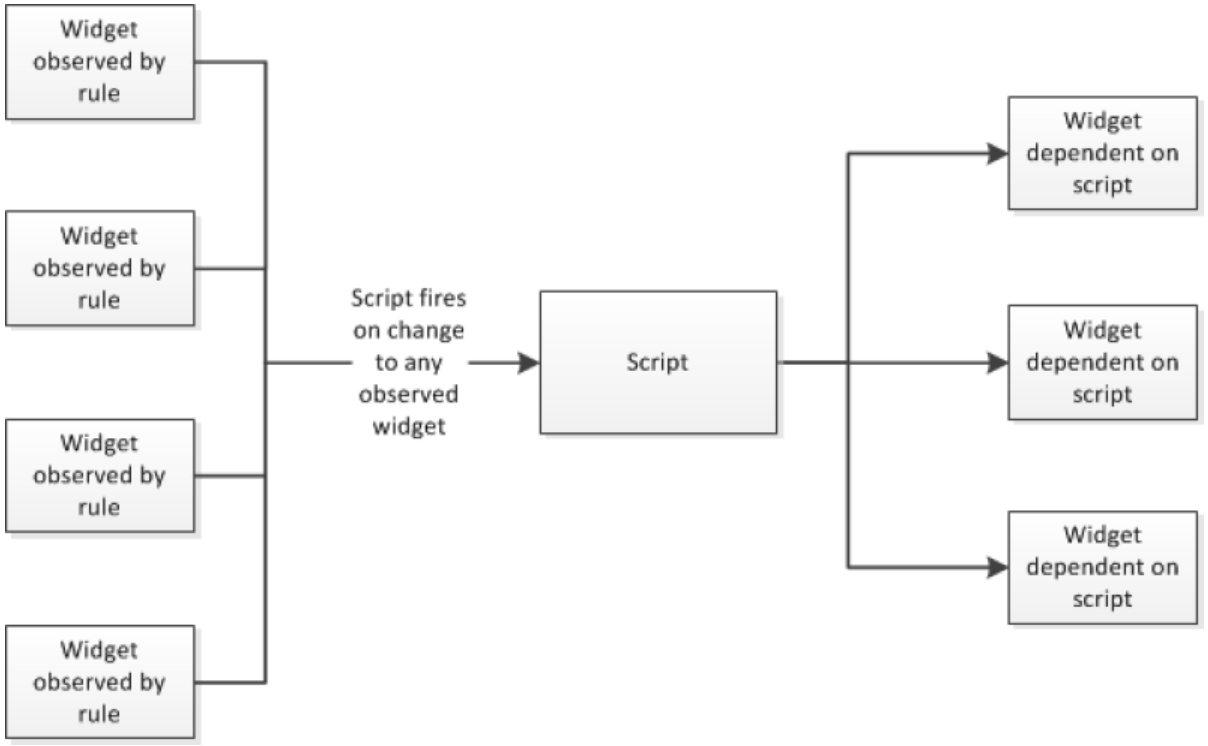
Dynamic vs Static Dependencies

When you write a script in the script editor of the Edit Properties dialog, you reference a form object using a Parameter. This creates a dependency, which duly appears in the My Rule Dependencies panel of the Dependencies tab.

The default setting for these is to have a Dynamic trigger type, as indicated by the presence of the lightning arrow in the left-hand column of the Dependencies table of the Script Editor:

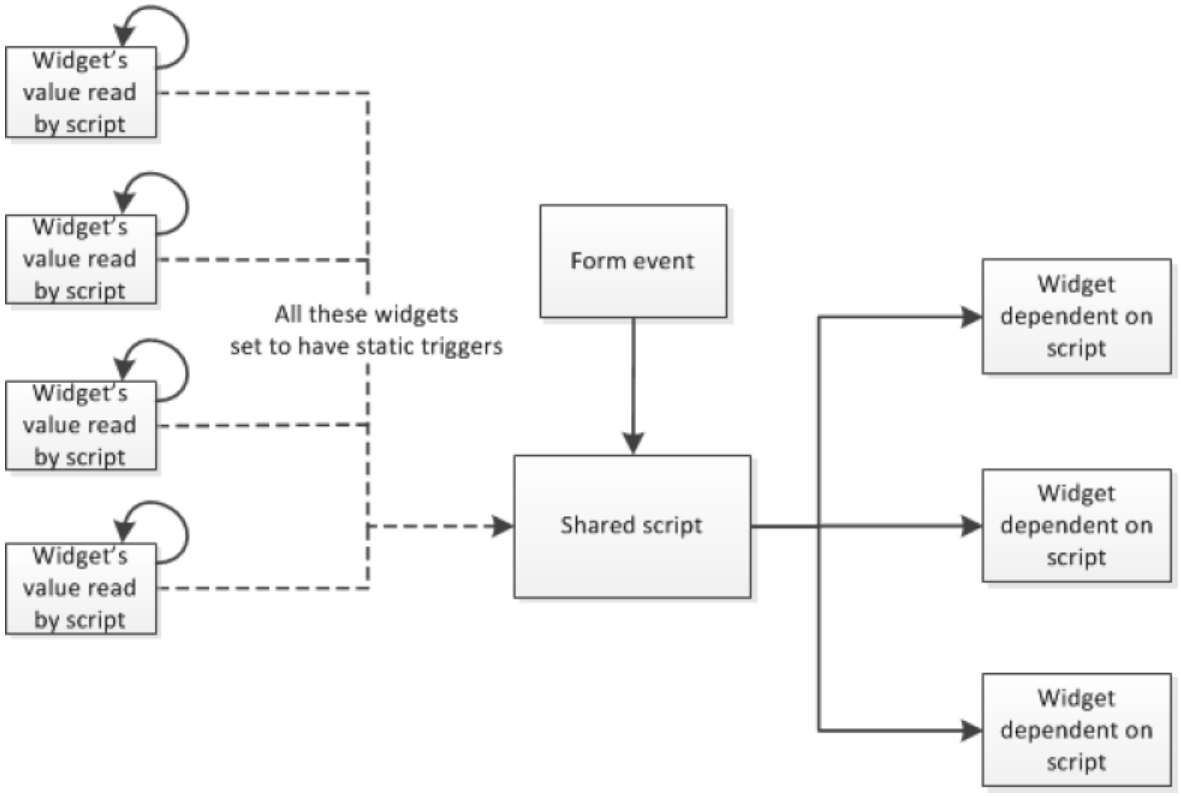


A dynamic trigger type means that the script will fire whenever the field referenced changes. That means the field is always watched and the trigger will always fire on any change to the field.

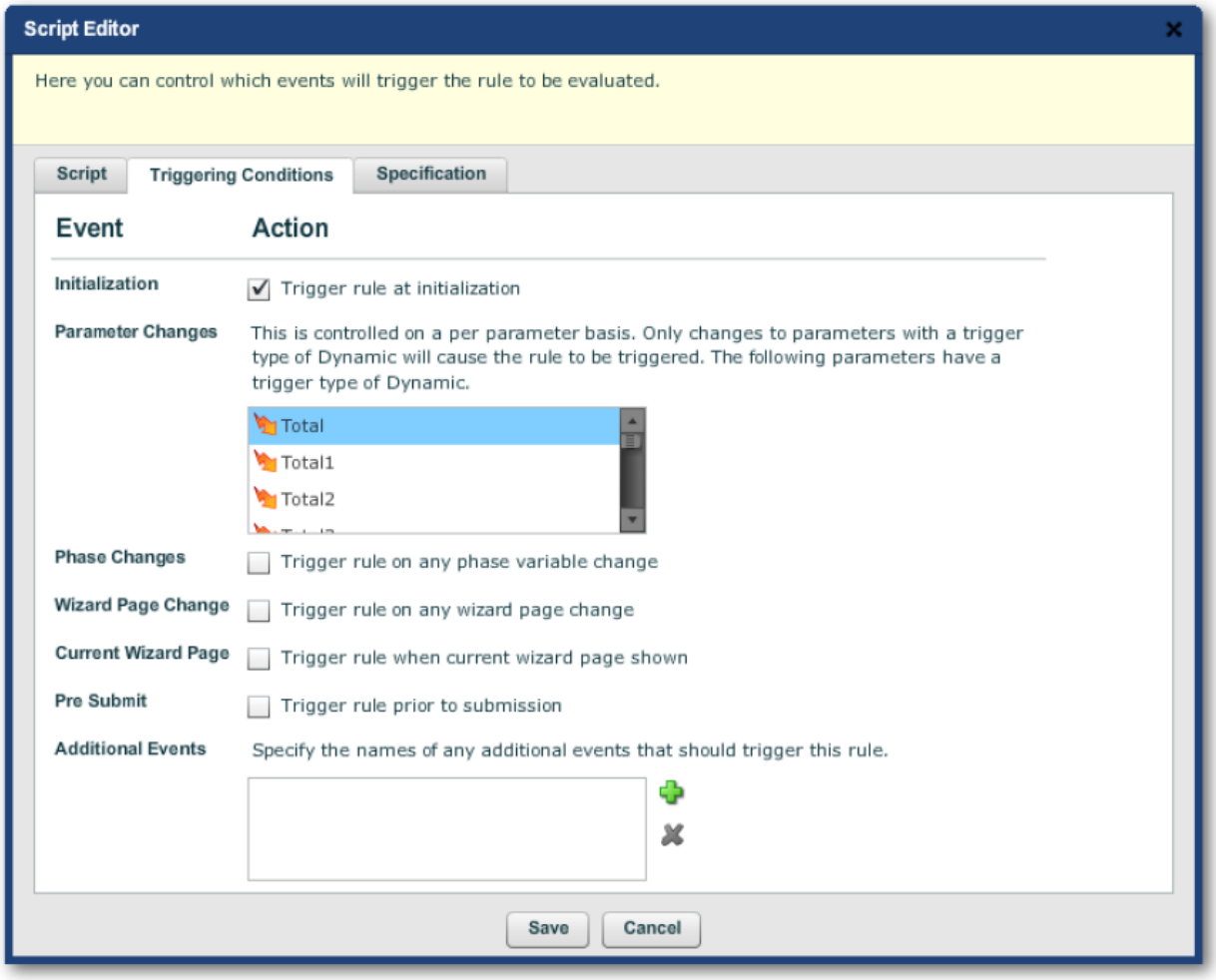


Changing the trigger type to Static means that the widget will no longer be observed. So, if we change all the dependencies in the above example to the Static trigger type, the rule will not fire. That we do want the rule to fire sometimes goes without saying, because we want the total to be the correct value. Which brings us to the Trigger Conditions tab in the Script Editor.

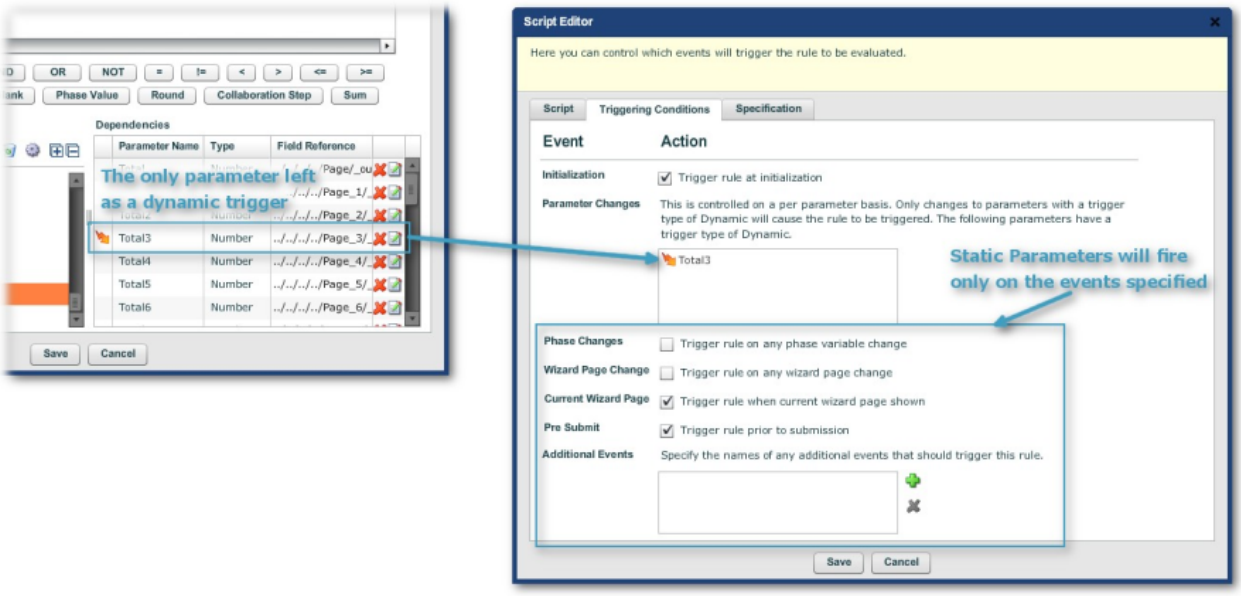
Trigger Conditions



Here, the widgets are only read when particular form events occur. We can specify the events in the Triggering Conditions tab of the Script Editor:



But the problem here is that all the parameters listed in the Parameter Changes list have dynamic triggers. That means the settings below the list will are not reduce the firing events by very much: the rule will fire on the Parameter Changes listed. So, as a demo, we have changed all the parameters but one to static, and set them to fire only when the user moves to the wizard page with the field containing the script or just prior to submission. Of course, in practice, you would most likely, in this case, turn all the trigger types to static to give you complete control over the script firing by eliminating the dynamic triggers.



And, the reverse also applies: you can then set the triggers from the form element with the script to the widgets dependent on the script (the widgets on the right hand side of the diagram with the form event) to also be static through the Script Editor of each of these widgets.

You can see that we can now, by going through the triggering conditions throughout the form, reduce the triggers from the original 2,730 fields to a much smaller number and substantially improve the performance on this form.

You might want to leave a parameter or two still with dynamic triggers: these could be a checkbox or button, whose action is to fire the script. In earlier releases of Composer, there were some work arounds to control the firing of scripts. These included having hidden a checkboxes (to act as a Boolean to make the script fire) or using blocks as a parameter reference (in the absence of static triggering). These are now deprecated.

Trigger at Initialization

You can turn off this option and the form will render in the browser. This would only make rendering perceptively faster if the script is compute intensive. It is not worth doing for the majority of scripts. But note that the setting is not global: it applies only to this script. Also, make sure that you are not creating issues that by not running this script.

Phase Changes

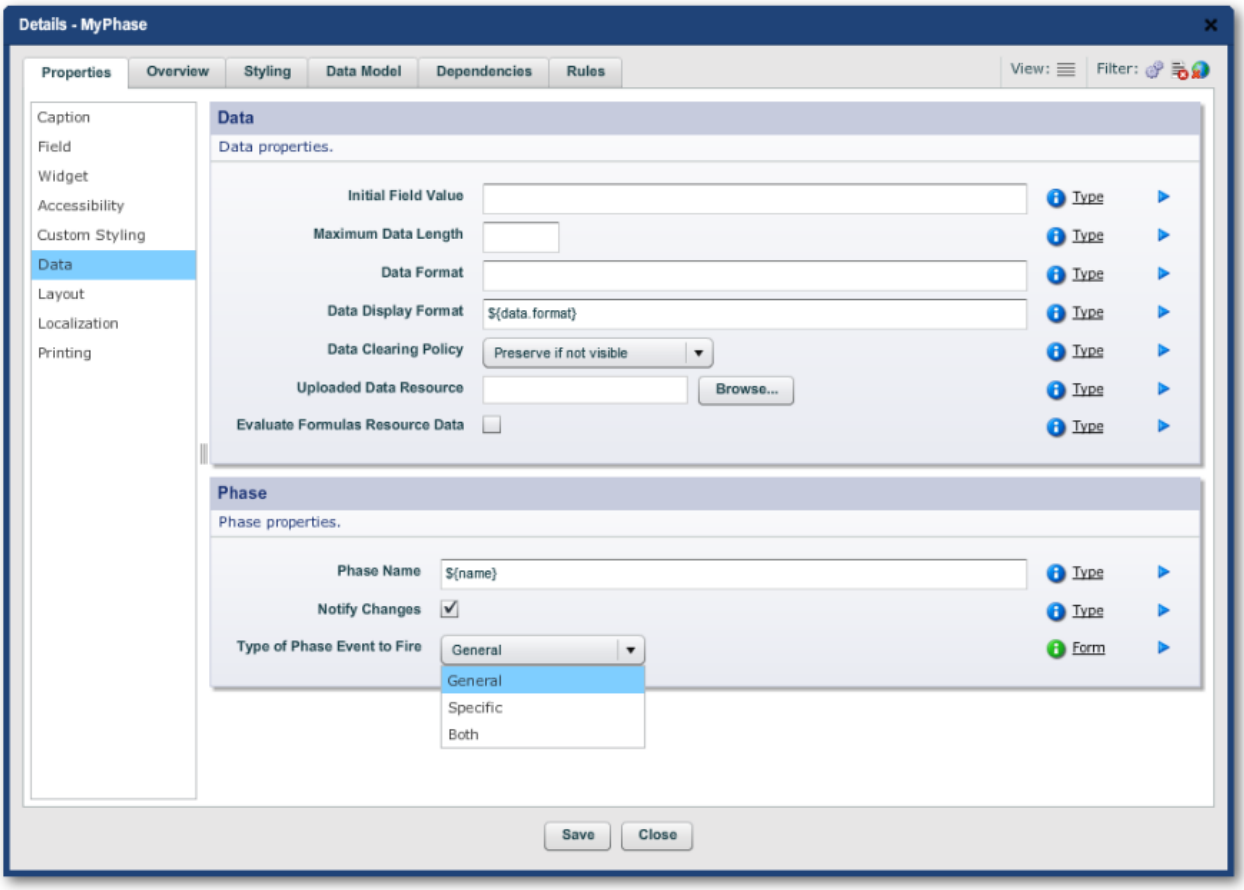
The events listed in the Triggering Conditions tab are self-explanatory. Phase Changes may not be.

A phase is a non-localized variable that holds some value and which can be observed for whatever purpose. It is non-localized because you need not reference it through either a relative or absolute path.

Here follow some brief notes on creating a phase.

Pick up a "Phase" Widget from the palette (use Search to find it) and drop it onto "Nuts & Bolts -> Phases" in the Structure Panel. In the resulting wizard, it is usual to check the "Exclude from Layout Manager" as the widget normally does not appear on the form.

The Phase is configured in the Data tab.



Put any scripts required in the Rules tab, as per usual. In other scripts, you can reference the phase by pointing to the structure widget as a Parameter or by src.getPhaseValue("phase name") as before.

Make sure you check the Phase Change checkbox in the Triggering Conditions tab of the scripts you want to fire on phase changes.

Additional Events

You can also type in the names of other events. The standard events are listed here in this guide.

Caveats

As you can see, these tools are powerful and require no programming as such.
However, as we have mentioned above, after optimization, you must test the form to see if the scripts yield the correct answers now that changes to field values no longer ripple through the entire form automatically. There are also other undesired effects that you must guard against, such as preventing validation rules from firing and therefore letting invalid data through to submission.

Business Rules

Composer | Form Builder | v4.3 & Higher

The Rules Properties section of this manual describes how you add scripting and logic to individual widgets. Though some widgets, like blocks, can pass on their rules to their children, this methodology does not allow rules to be assigned to several widgets, especially those not related or which do not share inheritances. What if the widgets are scattered throughout the form and cannot easily be wrapped by some block in the structure panel without including other widgets that you do not want to similarly inherit properties? This problem cannot be solved through "Edit Properties" individually for all the widgets concerned.

The "Business Rule" and "Mandatory" field types solve this problem by being central containers for rules and scripts. Other widgets can now share, for example, a script, by pointing to a business rule widget in the their "Edit Properties" dialog.

There are three types of Business Rule field types:

GeneralPurpose

These contain the logic that control visibility, editability, data clearing, event firing, triggers and event listening.

Error

These generate messages and also prevent submission of the form

Warnings

These also generate messages but do not prevent form submission.

1. Mandatory CheckboxBlock

Allows you to enforce the activation of at least one checkbox in the block: leaving all of the boxes unchecked will prevent submission of the form.

1. MandatoryBlock

You use this to enforce that **at leastone** of the contained form elements are given values.

These rules widgets are intended for advanced users only, though many of their features are no harder to implement than editing some of the properties in the "Rules" tab for each widget. They are a powerful tool for optimizing response times of complex forms

General Purpose

Configuration Settings

The General Purpose Business Rule widget's Rule tab holds the following:

Visibility Rule

Note the difference between **AlwaysVisible**, **NeverVisible** and **Script** or **Rule Based**.
That is: the value of an invisible field being observed by the Business Rule could be affecting the outcome of the rule because the field has not been cleared.
See also ClearingPolicies.

Visibility Policy

Initial Visibility
Visibility Data Clearing Policy
Business Rule
Either "None" or "Script Based"

Additional Event LIsteners

See here

Business Rule Trigger

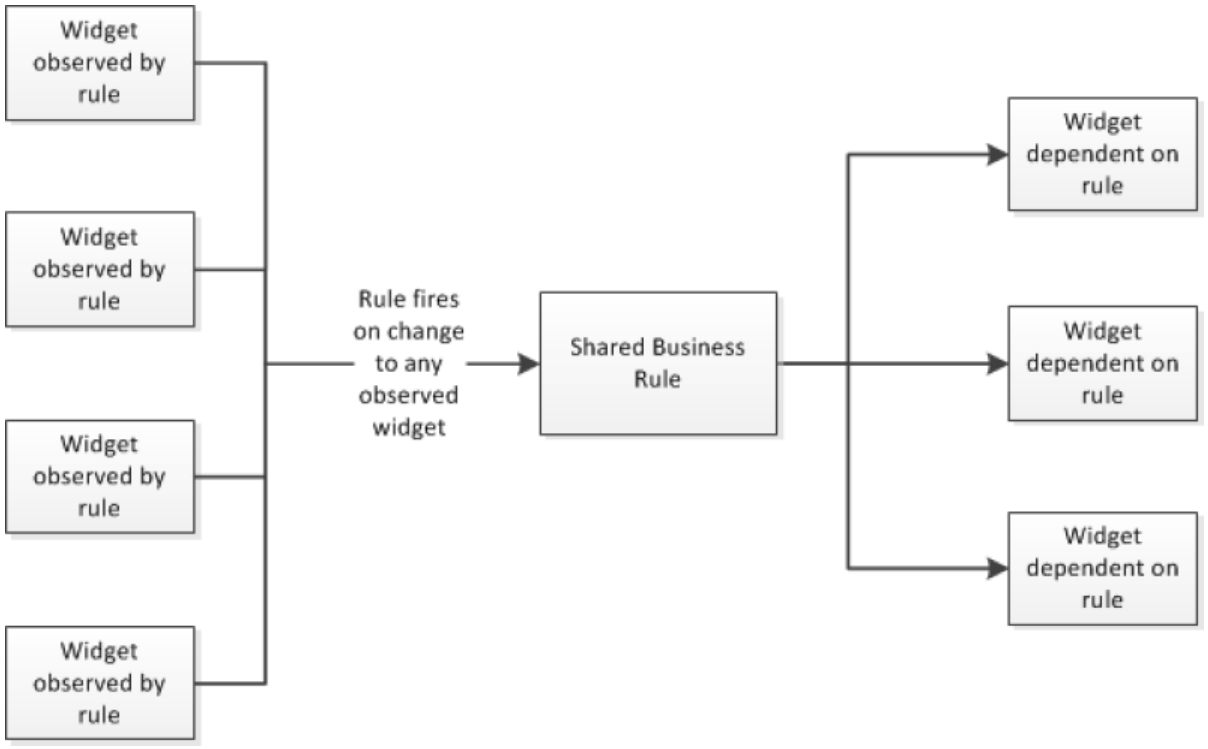
Run Always | Run only when visible.

The Relationship Between Nominated Widgets and the Rule

There are two broad ways for a General Purpose Business Rule widget to fire:

The default

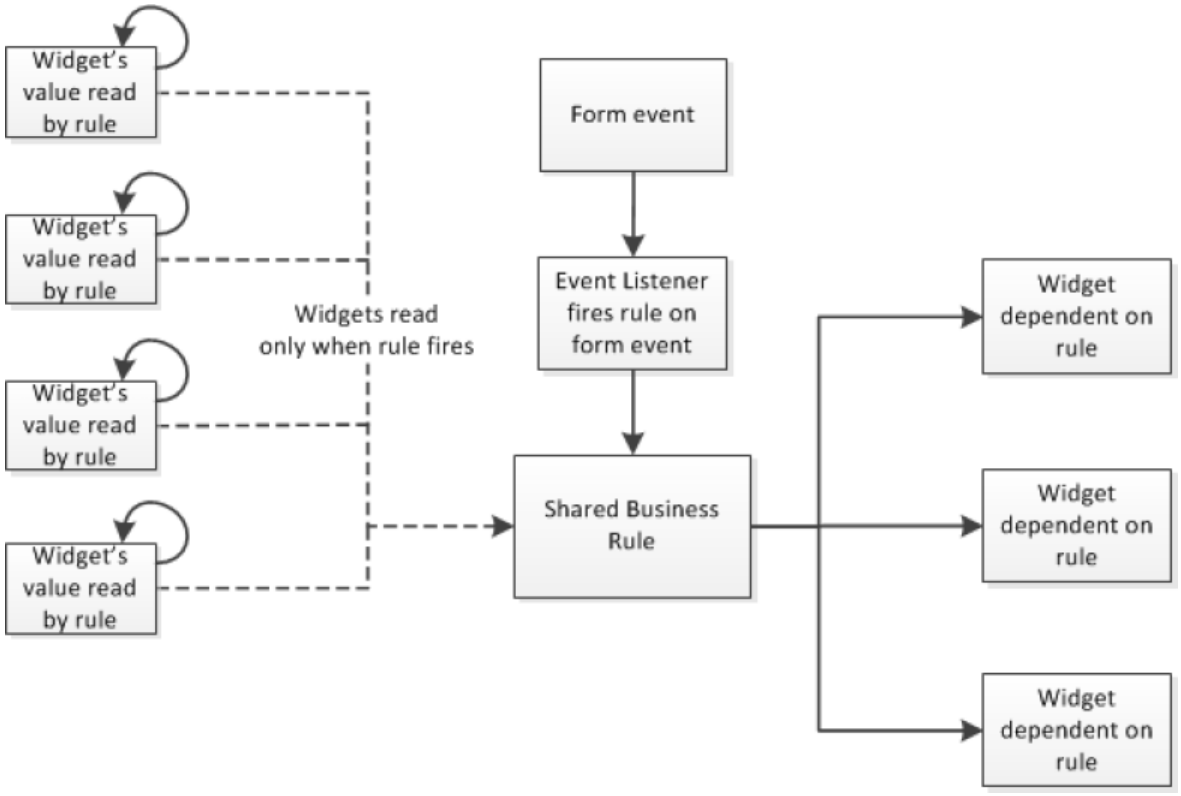
where the rule fires every time one of the nominated widgets changes. In other words, the rule's listener is forever observing these widgets for change and will fire every time one of these changes. This can slow the form down if the scale of all these observed widgets is large enough.



Rule fires on change of any observed widget; as a result, the rule could fire many times and make the form laggy

1. Additional Event Listeners

Here, the widgets linked to the rule can asynchronously change, but the rule will only fire when a form-level event is detected by the listener. The form events are listed, comma separated in the "Additional Event Listeners" field in the "Data" tab of the Business Rule widget.



Rule fires less and only on a nominated form event or events; results in a more responsive form.

The above second case results in the Rule widget firing less frequently than for the first case: every change to any of the observed widgets no longer results in the rule firing every time.

Standard Composer Events

Event	Description
init	The first event to which a widget can listen when the form starts. This event sets the initial data-based state.
postInit	Runs immediately after init. Used to run code when everything is in the initialized state.
change	The only event that does not call everything listening to it. It will look at the source node and look up the list of fields with a dependency on that field.

phase	Will fire when any phase value changes. evt.evtData is the name of the changed phase. Its new value is fetched with sfc.getPhaseValue("<phaseName>"), where <phaseName> is the name of the affected phase.
evtValidationView	This event is fired once all current validation updates have been completed and it is ready to update the error lists, etc.
preSubmit	this event fires immediately before the form is submitted (although there is an issue that preSubmit is not currently fired when using a form in Mobile App but this should be resolved in a future release).

Error

Error Business Rules will, on error, prevent the user from submitting the form.

The rule has more settings than the General Purpose business rule properties. The visibility rules in particular can cause confusion if not properly understood.

Focus Field Target

The rule fires when the user focus becomes the nominated field and whenever the value of the field changes. The rule can be set to "Run Only When Visible" (which is the default setting) or "Run Always". The nominated field will display the warning background colour while the rule is in error.

Clearing Policies

See Privacy on the privacy considerations that dictate data clearing policies. Note the distinction between "ClearHiddenDataonSubmit" and "ClearDataWhenHidden".

The distinction may seem trivial if we consider only the form's data at the point of user submission. However, when one of form's Business Error Rules is dependent on a field that happens to be hidden, and that field is causing the rule to be in error, that hidden field would prevent the user from submitting the form, even though all the visible data is suitable for submission.

So, this is the reason for the distinction between "Visibility Data Clearing Policy" and "Editable Clearing Policy" in the rule's properties.

Displaying Errors

The default display of form errors is the standard footer of the typical template. The same structure is also available with the Error List Predefined Block. The errors are listed in a clickable list; double-clicking on a list item takes users to the form element needing recitification. You also can write the explanatory text displayed as well as the error messages themselves. You therefore have the tools to assist users though the process and, therefore, minimise the rate of form completion failures.

The Form can display both errors and warnings through the ErrorandWarningPredefinedBlock. Users can scroll through wither the errors or the warnings using next and previous links.

The ErrorSelectionBlock displays both of these blocks, one on top of the other.

The Edit Properties -> Rules tab of the Error Business Rule can also be set to displaying an error dialog: a popup with the text of the Warning Message of the Rules tab. The same also can be set for warnings.

The use of these various blocks gives you some flexibility in how you want problems with the form data to be explained to users and what to do to remedy these issues.

Warning

These rules behave the same as the Error Business Rules, except that warnings do not prevent the user from submitting the form. We have already discussed the display of warnings above.

Mandatory Checkbox Block

This mandatory block enforces the rule that its containing checkboxes must have at least one member selected before the form can successfully be submitted.

Mandatory Block

This mandatory block enforces the rule that its containing members and groups must have at least one member selected before the form can successfully be submitted.

There are, however, a subtle differences between using this widget or using the button assistant and checking the"Mandatory"box in the creation wizard for the assistant:

The Mandatory Block by default leaves all its contained buttons or checkboxes unchecked or without value. If end users try to submit the form with all these boxes unselected or unfilled, the error message for the block is displayed and all the contained elements get red outlines.

Checking the "Mandatory" box of the button assistant wizard means that at least one of the buttons (the first by default) is selected at form initialization and there is no way to go to an all-unselected condition.

Several checkbox or buttons blocks can be contained in the mandatory block, all of them unchecked at initialization, and yet only one button of one group must be selected before submission and all the other buttons in the other groups can remain unchecked.

Library Advanced Features

Composer | Form Builder | v4.3 & Higher

Overview

As we have already seen above in TheHierarchyandEnvironment, an Account has Libraries and Organizations under it. Also, there are also special libraries called "Releases" and "Service Packs" which we will call here "Compatibility Settings"; but these are assigned to Organizations. Also, the Organization itself has its own library elements attached to itself. Here are the different Libraries and their locations:

Library Type		Location or Association
Compatibility Settings	Releases	Live above the Workplace structure. Each Organization is associated with a particular release.
	Service Packs	Live above the Workplace structure. Each Organization may be associated with particular service packs. A Service Pack is also relevant to a particular release. Any Library that is neither a Release or Service Pack is termed an "ordinary" Library.
Ordinary Libraries	Native Libraries and Style Packs	Style Packs are Libraries packaged by Avoka Technologies. Otherwise, they are ordinary Libraries. "Native" here means that the Libraries live in the Account, and are therefore already shared with the
Library Type		Location or Association
		Account
	Shared Libraries and Shared Style Packs	These are ordinary Libraries that live in other Accounts and which have been explicitly shared with this Account.
The Organization's Internal Library		Lives inside the Organization, not — unlike the other Libraries in the Account — as a separate Library entity under the Account. The elements in the Organization's Internal Library affect all Projects in the Organization regardless of other settings.

Composer Account Administration

If you wish to carry out work on Libraries, you should consult the "Transact Composer Account Administration Guide". This is a handy reference on where to go in the Composer user interface in order to carry out such important Library-related tasks as:

Allocating users at the Organization and Project levels Creating and importing libraries

Inspecting which libraries are available to Organizations Assigning Libraries from other Accounts to an Organization

Maintaining Libraries and their contents (style sheets, templates, custom types and property sets) Ongoing maintenance

Maintaining compatibility through revision maintenance of Organizations

The Administration Guide gives a number of tasks to be completed in order to make Libraries and particular Templates available to Project users. The rest of this topic below discusses the theory on how the stack of inheritance works and what its effects are. The administration tasks and the theory work together and both need to be understood.

How Libraries Relate to Organizations and Projects

There are 2 properties in the behavior of libraries that need to be understood here:
Visibility and

Inheritance

Visibility is the easier to understand. Inheritance then is a consequence of visibility.

Inheritance is extremely important. It determines what templates are available at the form level and the style of each template at the Project level. Please refer to Inheritance Within Forms. especially the diagram showing how the inheritence of library components then affect the behavior of the form's elements.

Visibility

Ordinary Libraries at the Account Level

Ordinary Libraries — those not part of the Compatibility libraries — are visible at the Account Level. The rules for visibility for ordinary Libraries operate at the Account level and are:

Libraries that are live in an Account are visible to all Organizations in the Account. This includes Libraries imported into an Account via a package. ("Workspace -> <Account> -> Import Library or Organization link") The Libraries of an Account can be explicitly shared (or not shared) with other Accounts. (Go to "Workspace -> <Account> -> <Library> -> Administration tab -> Library Share sub tab".)

Visibility means that the components of the Library, **withthe exception oftemplates**, are shared by all Organizations and their children in the Account. Here, we do not include Administration (the configuration settings for visibility for users and Organizations) and Problems (a log of error messages for the Library).

Compatibility Libraries

Compatibility Libraries contain Releases and Service Packs; these are visible at the Organization Level. They are allocated at the Organization level in the wizard when a new Organization is created ("Workspace -> Create New Organization link"). Compatibility can be changed later (via "Workspace -> Organization -> Change Release action button"), meaning that the Organization will point to a different set of compatibility Libraries.
So, the consequence is that different Organizations in the same Account can be compatible with different Composer versions and service packs.

The Organization's Internal Library

In terms of its type of contents, the internal Library is pretty much an Ordinary Library. But in terms of its effect, it is the most powerful of all the Libraries because it lies above the Search Path settings of the Organization.

Search Path Inclusion at Organization Level

When an Account is visible to an Account, either by virtue of belonging to the Account or by being shared, the contents of the Library still doesn't affect the Organization until it has been included in the Organization's Share Path. You specify the share path in "Workspace -> <Organization> -> Update Organization Details action button -> second screen of the resultant wizard".
Note: the choice of Libraries for search path inclusion is only between Libraries native to, or shared with, the Account.

Templates

Template availability operates at the Project level. Only Templates from visible, search path-included Libraries can marked "Available" to a Project.

Visibility Summary

Note: Visibility has to cascade down from the Account Level. Thus, for a Template to be allowed for a project, its Library has to be shared at the Account Level and put into the Search Path at the Organization Level.

Level	Object	Location	Method
Account	Ordinary Library (Templates, Stylesheets, Custom Types, Resources)	Within Account	Always Visible
		In another Account	Shared with other accounts: "Workspace -> <Account> -> <Library> -> Administration tab -> Library Share sub tab"

Organization	Compatibility Libraries (Releases and Service Packs)	Assigned to Organization through "Release Settings"	In wizard when creating account "Workspace -> <Account> -> Create New Organization link" or when updating Organization "Workspace -> <Account> -> <Organization -> Change Release action button"
Level	Object	Location	Method
	Ordinary Library (Templates, Stylesheets, Custom Types, Resources)	Anywhere in the environment. Must be shared with the Account and included in the Search Path	In wizard when creating account "Workspace -> <Account> -> Create New Organization link" and 2nd screen of the resulting wizard or when updating Organization "Workspace -> <Account> -> <Organization -> Update Organization Details action button", 2nd

			screen of the resulting wizard.
	Internal Library	Is automatically allocated to every Project in the Organization. Affects all Projects in the Organization and its settings overide all other settings.	Examine and populate this library through "Workspace -> <Account> -> <Organization> -> the tabs Stylesheets to Resources"
Project	Templates	Anywhere in the environment. Containing Library must be shared with the Organization	Allow or Block the template from being used in new forms in the project. "Workspace -> <Account> -> <Organization> -> <Project> -> Block or Allow this template to be used in new forms in this Project action button".

Inheritance

The Cascade

Stylesheets and Custom Objects now live in the Libraries, not in Templates. All Organizations now inherit all the Stylesheets, Custom Objects and Resources contained in the Compatibility Libraries.

This means that all users have access to all the objects in a release and that all the release's widgets will appear in the Form Editor's palette. (In fact, if you have the access, you can open a release and you will find that the widgets in the "Custom Types" tab, though you must check "Include non-editable types" to see them.)

So, there is a cascade of inheritance at the Project Level. which is where the Library visibility settings matter. Forms within the project inherit the Library visibility of their Project. The form inherits all the Stylesheets, Custom Objects and Resource, plus all of the allowed Templates, of all the visible Libraries.

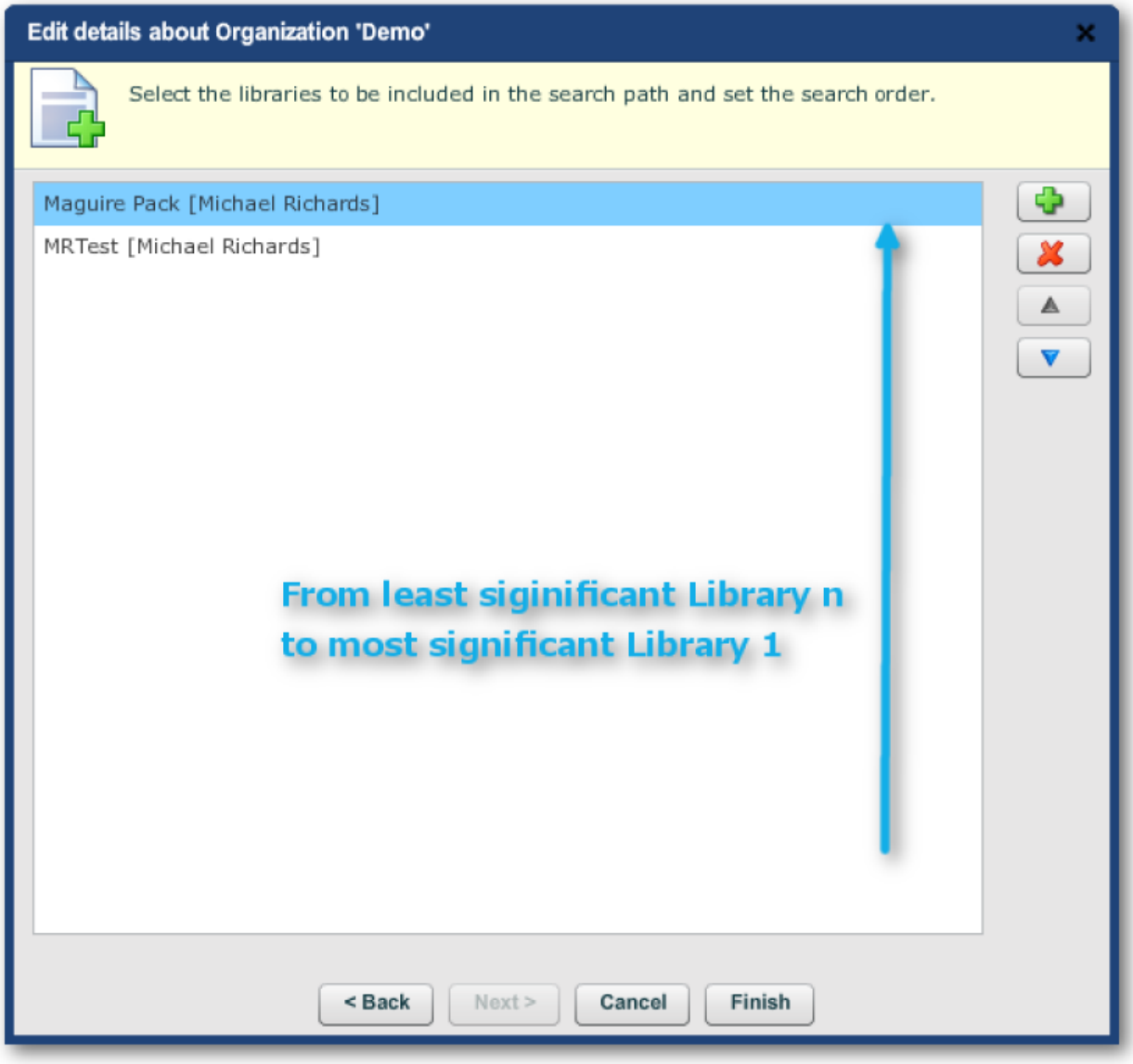
Search Path Order

So, what happens when there is a conflict, for example stylesheets in different libraries with the same name? This is resolved as follows, moving through the stack of libraries from the least significant to number 1 in the Search Order and then to the Internal Library:

Compatibility Library -> Search Order Library n -> ..... -> Search Order Library 1 -> Organization's Internal Library

The Internal Library, therefore, lies at the top of the stack.

You set the Search Order in "Workspace -> <Organization> -> Update Organization Details action button -> second screen of the resultant wizard":



Do note that where n > 2, the search order for the other lesser significance libraries may still matter because the element at the top of the stack may not specify every parameter and those that are not specified will be populated according to the order of values encountered in the stack.

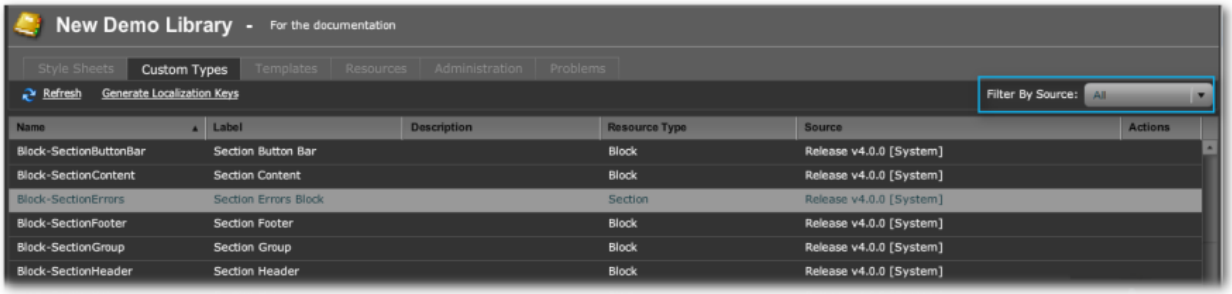
Custom Types

Location: Workspace -> <Organization> -> Custom Types tab or
Location: Workspace -> <Library> -> Custom Types tab
Custom Types refers to all the members of the Widget Palette of the Form Designer. There are 2 types of Custom Types:
Non-editable CustomTypes
the standard members of the Widget Palette.
Editable Custom Types which are your saved types.

Localization Keys

used for the Internationalization of forms. We will discuss this elsewhereinthis guide.

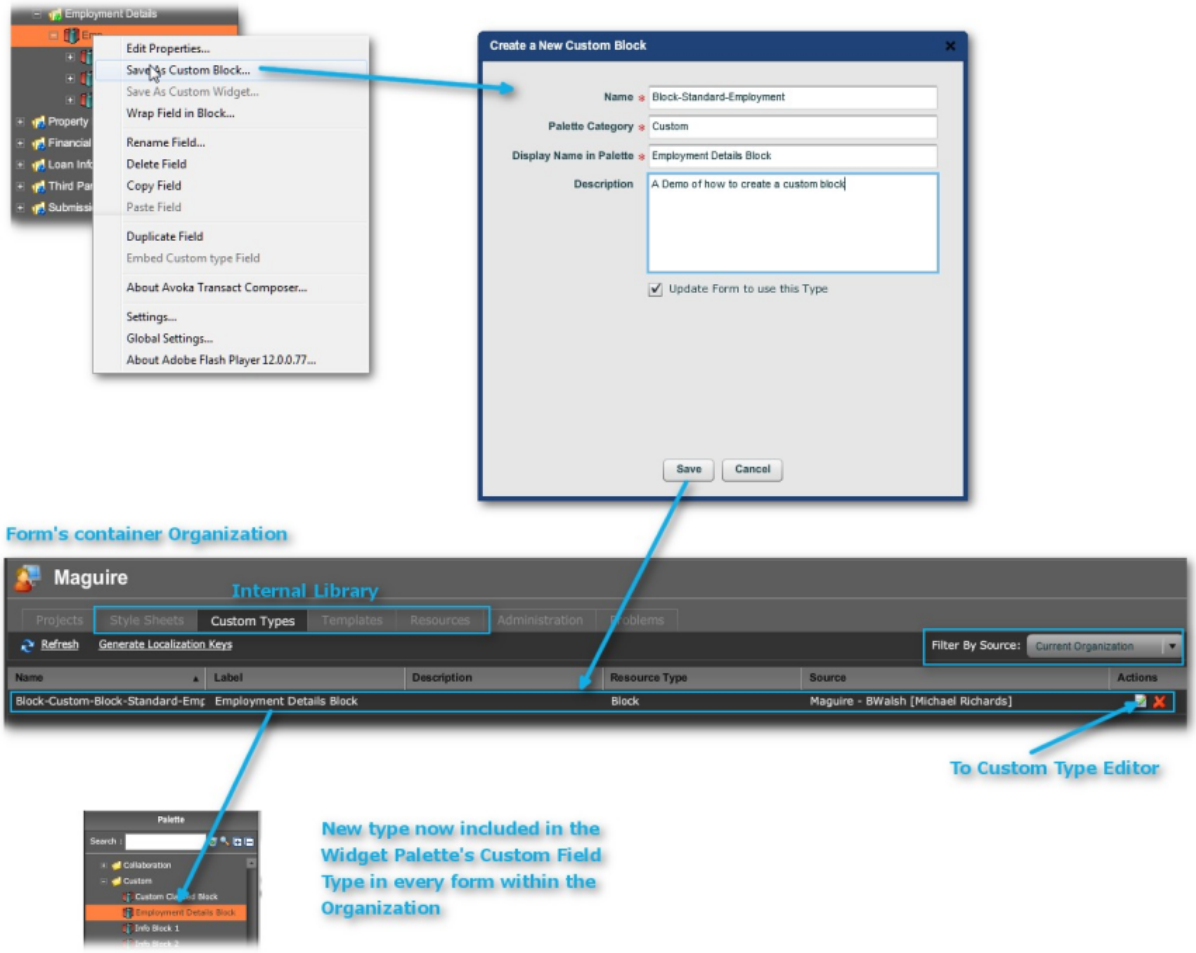
Non-editable



To list the non-editable Custom Types visible for a Library (either Ordinary or internal to an Organization), go to the Custom Types tab of the Library or Organization. Set the "Filter By Source" to "All".
There are no Action items.
All the non-editable Custom Types have been inheritedfromthestack, so the bulk of them come from the Release and Service Pack.

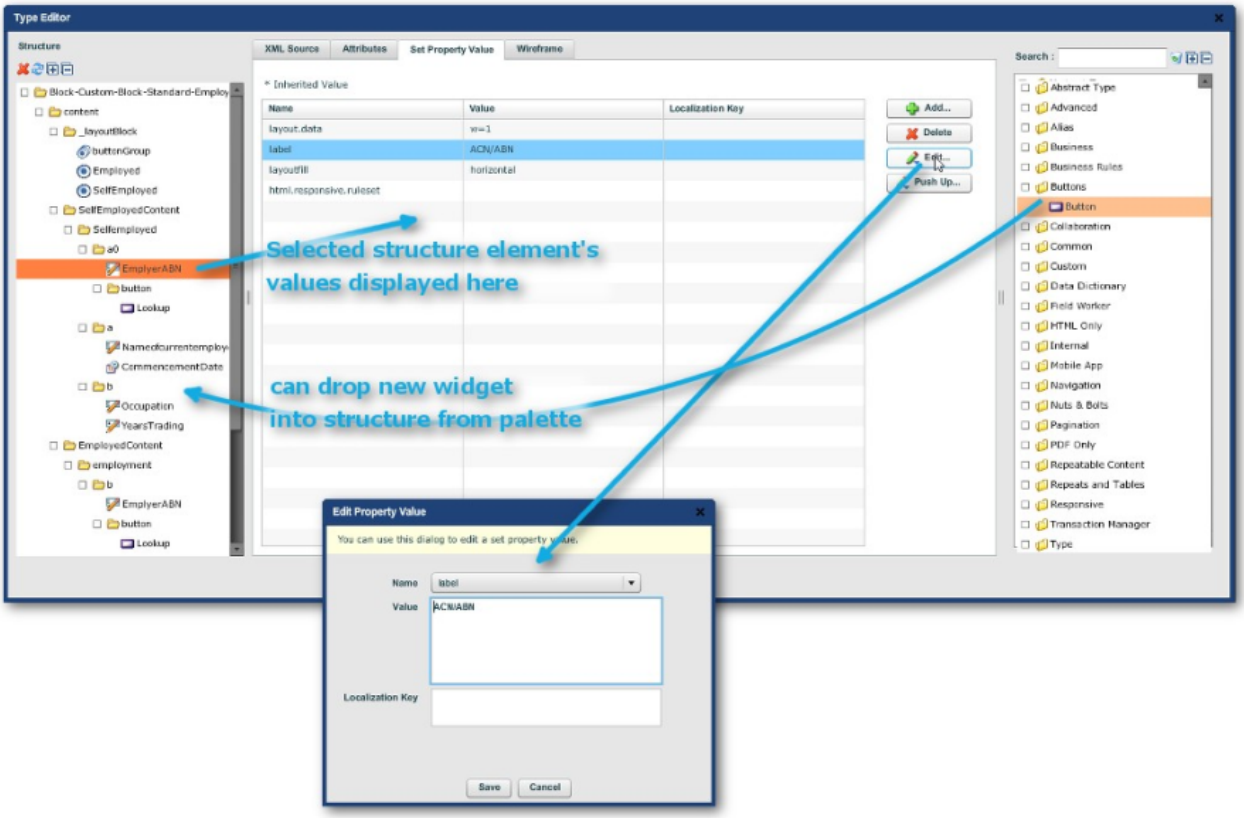
Editable

There is only one way to create a Custom Type: by using "Save As Custom Type/Block" from a form's Structure Panel. The newly-saved type will then appear in the Internal Library of the form's containing Organization. It is not possible, using the User Interface of the Form Designer, to create a Custom Type in one of the Account's Ordinary Libraries.

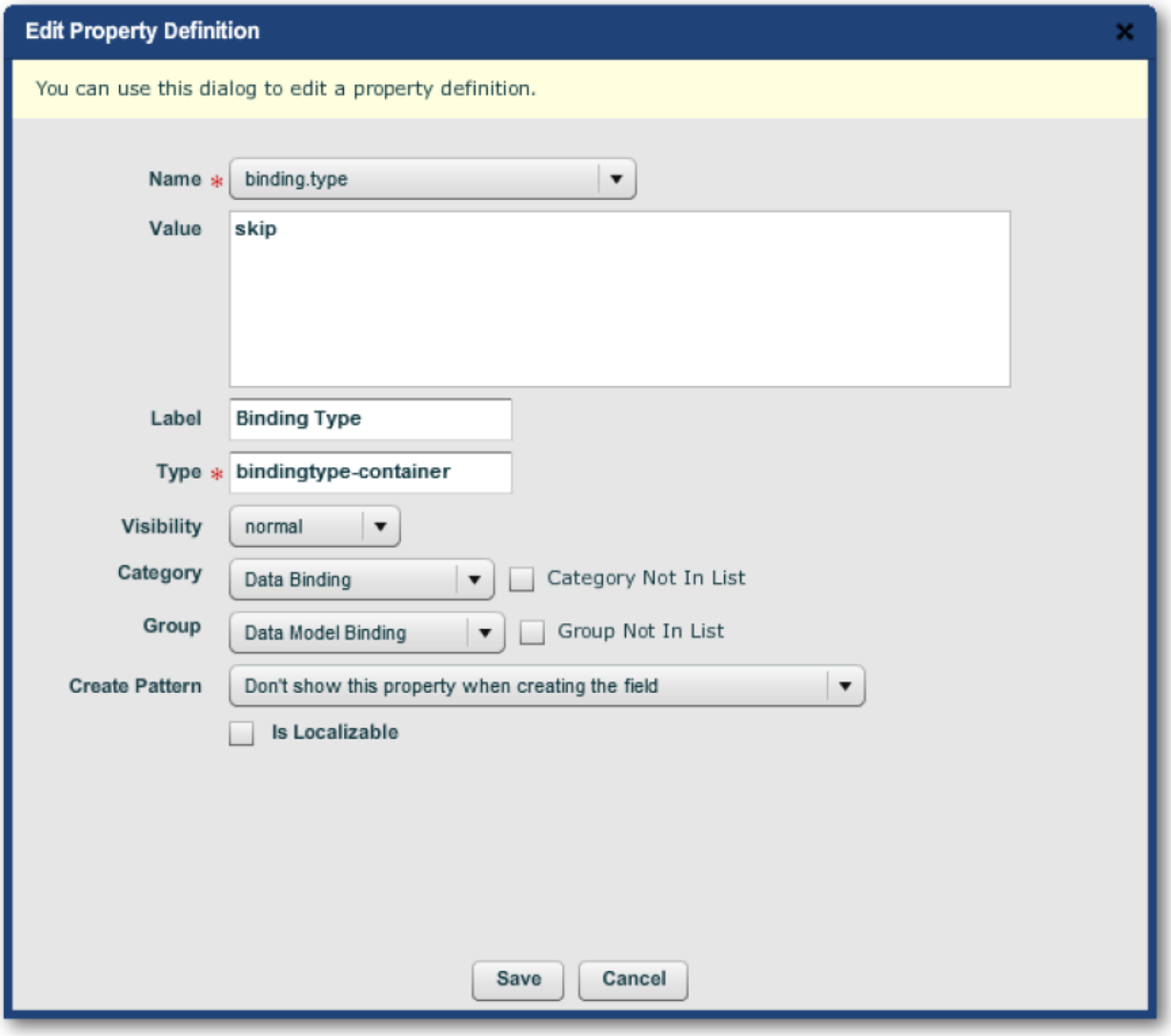


Because of the position of the Organization's Internal Library in the inheritance stack, your Custom Types will be available to all Projects in the Organization. However, you are not able to export these to other Organizations in the Account or to other Accounts. You can, though, export the whole Organization as a package and import it into another Account and the Custom Types in the imported Organization's Internal Library will also come across.

You can open the editor on editable types:



In the Set Property Values tab, you can edit the values of all the structure elements contained in the Custom Type. Inherited values from the stack are marked with an asterisk ("*").



The **Create Pattern** dropdown menu in the Edit Property Definition dialog defines whether:

The property appears or does not appear in the wizard when the custom Type is dropped onto the structure panel of forms

and if it does appear in the wizard. whether the property is marked mandatory or not.] Here are examples of the other tabs in the editor:



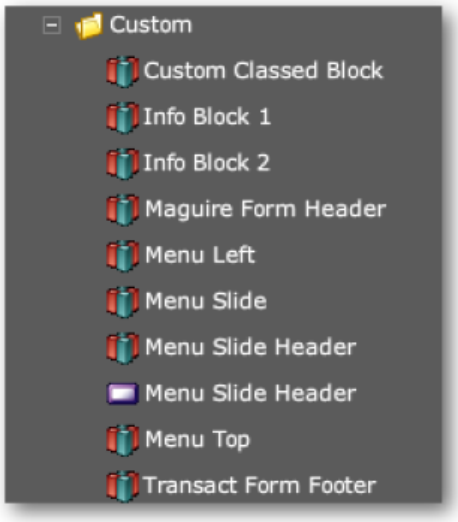
Note that the XML Source updates on-the-fly as you add or subtract new objects into the structure or edit

attributes and property definitions.

Style Packs

Custom Types play a major role in Style Packs. The Custom Field Type, for Organizations where a Style Pack is part of the stack, will have a rich selection of prebuilt widgets.

The Maguire Style Pack is a good example of this. It has a number of form structures not covered by the usual Nuts & Bolts settings.



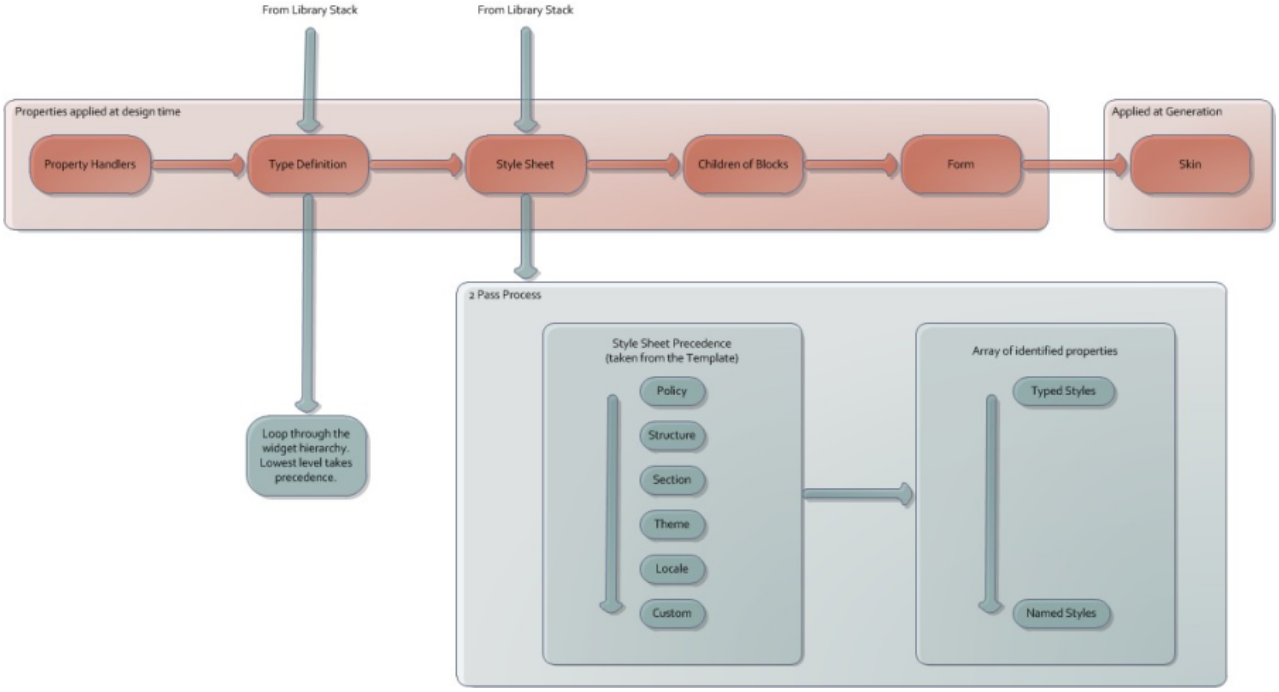
Besides these custom widgets in the Form Designer's Palette, such a style pack also has its own set of Stylesheets, Templates and Resources to supplement the Release. Typically, a release has a very large number of Stylesheets because the set has to cover practically every time of form element, while a Style Pack's set is very much smaller and far less general in scope.

Inheritance Within Forms

Overview

As above, forms inherit their Library elements (Stylesheets, Custom Types, Templates, Resources) through the stack. So, the library elements available to the form has already been established. This includes stylesheets that you have modified to change color-schemes and other components of the general look of your forms.

But then, inheritance now occurs through the form's structure:
The settings of the various Style Choosers
Settings of parent form elements and their children (blocks and their contents, for example) The use of Named Styles (or even manual over-rides



Widget Hierarchy

As we have already seen in CustomTypes, some widgets are blocks of widgets and and children blocks. The properties of the simplest elements of these compounds are the ones to take effect. See the next topic immediately below.

Inheritance in blocks

Inheritance in blocks is similar to inheritance for individual widgets, but it is complicated by the fact that blocks contain content. When you drop a block onto a form, its content is also inherited, and can also be overridden, just like regular inherited objects. So you can simply double click on any of the child objects contained in a block, and modify their properties.

However, there are some important differences between inherited children and regular children. For example:

You cannot delete an inherited child (although you can change its visiblity to "Never visible", which achieves a similar result)

You cannot re-order inherited children

You cannot insert another object between two inherited children (although you can append an object at the end of the inherited children)

In order to differentiate between inherited children and regular children in the tree, inherited children are shown in yellow. This helps you to identify why certain operations (such as right click->Delete) are disabled for some objects.

If you make a change to a property of an inherited child object, only that particular property change is stored in your form (or overridden); all other property changes are obtained from the shared block. So if the definition of the block is changed, your form will automatically inherit all of the changes, except the specific properties that you've overridden. This makes maintenance of your form library very simple - you can change blocks in one place, and have the changes automatically reflected in all your forms.

Starter Content

Sometimes when you drag certain types of objects such as sections, tables or even complete forms, you will see a checkbox called "Include Starter Content". Starter Content appears similar to inherited content, but there are some important differences.

Starter Content is not really part of the block itself - it's just additional children that are optionally added to make your life easier and quicker. This is partly to save you time, and partly to show what type of content is appropriate for this type of object. There is no difference between the starter content, and content that you may have manually added yourself.

Therefore, you can modify, delete or move starter content just like you can any other object - none of the restrictions of inherited content apply.

Resources

The Resources tab in a Library is where you store uploaded files. In theory you can upload any kind of file. But, the User Interface of Composer really only allows you to utilize only graphics files in a form.

Other file types commonly found in library resources are JavaScript (.js) and Cascading Stylesheets (.css). Please refer to LibraryAdvancedFeatures on how to utilize these in forms.

Other file types stored in the Resources tab are XML (for data definitions), XFA files, .zip files, backups and others. These are currently beyond the scope of this guide.

Custom CSS

Composer uses an abstracted stylesheet mechanism in order to allow styling of both HTML and PDF forms. For HTML forms, you can implement additional styling by directly specifying the CSS in the form definition to achieve effects not possible in Composer alone.

Note: CSS is rarely developed in isolation. It is usually specifically designed to add styling to a set of web pages that are structured in a particular way. This has two implications:

1. 1. 1. In order to use CSS effectively, you must understand the way that Composer generates HTML forms. The easiest way to do this is generally to open a Composer generated form in a browser that enables you to explore the HTML and CSS, such as Firebug (in Firefox), in Google Chrome or Web Inspector in Safari.

1. 1. 1. It is unlikely that importing an existing CSS file from, for example, your corporate web site, will have much value to be used directly in Composer forms (except in the most trivial of cases). It is more likely that you will extract certain elements from your existing CSS, restructure or refactor them, and then import the modified CSS into Composer.

Note: If you are going to be directly implementing CSS, you must not only understand CSS in general, but also be careful to test your CSS on multiple browsers and devices. The advantage of sticking with regular Composer styles is that we have done all the testing to ensure that changes work correctly in all browsers.

Custom JavaScript Libraries

Sometimes you may need to add custom JavaScript libraries to your project. This is an advanced topic, and we generally recommended that you add custom libraries with the help of Avoka professional services.

You can do this as follows:

Write your JavaScript functions using a text editor, and save as a .js file, such as mycode.js.

Drag the JavaScript Library widget from the palette to the Nuts and Bolts/JavaScript Libraries node. Specify a name.

Click the browse button to upload and select your mycode.js file to be associated with this library.

You may now invoke functions in mycode.js using the following syntax:

library_name.function(a, b);

If you want two different versions of the library for PDF and HTML:

Create two versions of the .js file. Ensure that each library has the same functions, with the same signatures, but with appropriate implementations.

Create a JavaScript Library object for each .js file, but check the checkbox to specify it's used for only HTML or only PDF respectively.

Override the Library Name property (which defaults to the Field Name by default) so that both Libraries have the same Library Name.

This way you will be able to use the same syntax to call functions, and it will automatically have the right version of the library in either the HTML or PDF forms.

Templates

Overview

We have already seen ademonstrationofthepoweroftemplatesinComposer. Composer 4.0 has an editor for creating new templates which now gives you control over some of the template properties.

which Style Choosers will be visible

what options each of these choosers has custom choosers

which form types will be available Custom form types

Layout of form headers and footers

However, the bulk of the work of creating a template involves editing

stylesheets

Using the Template Editor in conjunction with editing the XML source of the template uploading the appropriate resourcefiles to be referenced in the XML source

There are about 120 stylesheets and over 300 Custom Types in the latest release, not to mention those in future service packs. Such a workload is the reason we recommend that Avoka Professional Services be engaged to create templates for you, if what you require is an extensive departure from the templates that ship with the product.

But, it is also true that no one would in reality contemplate a complete re-invention of the wheel: i.e. an overhaul of every stylesheet, property and parameter of the shipping libraries.

That being so, here follows some notes on the creation of template based on moderate intervention into the stylesheets

Stylesheet Modification

Location of the Stylesheet Modifications

The location of these revised stylesheets depends on where you want these modifications to sit in theLibrary stack.

Will these modifications be made available to A few Projects in an Organization

Or to all projects within one Organization.

To several Projects in more than one Organization, all in the same Account

Or to all Projects in several Organizations, still all in the same Account Or across several Accounts

Ultimately the choice is to have your new stylesheets either in an internal Library for one Organization or to be in a separate library in the Account and make sure that any form you intend to use the template has the necessary visibility of the library elements.

Scope of Stylesheet Modifications

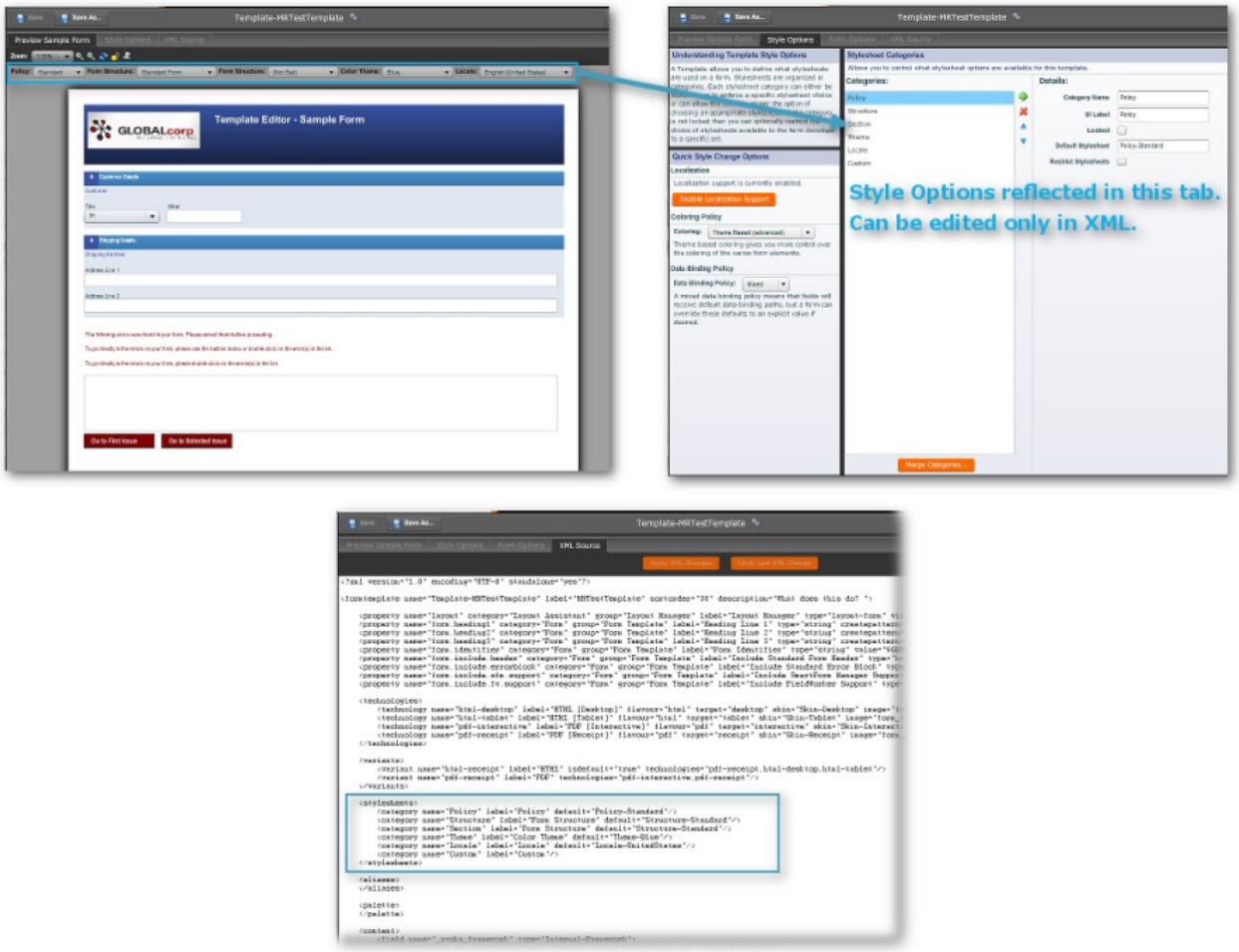
Just be aware that changing stylesheets will also affect the standard-issue templates. If you want the templates shipped with a release to still look and work the same as when they were installed, you should reserve, or create afresh, an Organization to hold the unedited stylesheets that accompanied these standard templates.

Editing the Elected Stylesheets

We have already described how to do this. If you want your new stylesheet to replace the standard issue stylesheet in the stack, you must give it the same name as the original. After you have saved the new version, you should be able to see it in the "Stylesheet" tab of the Library or Organization you have chosen to contain the modified version. Note the values in the "Source" column for stylesheets with the same name to check that you new version exists. Use the template editor to view if the change is expressing itself in the template and that it is having the intended effect.

Template Editor

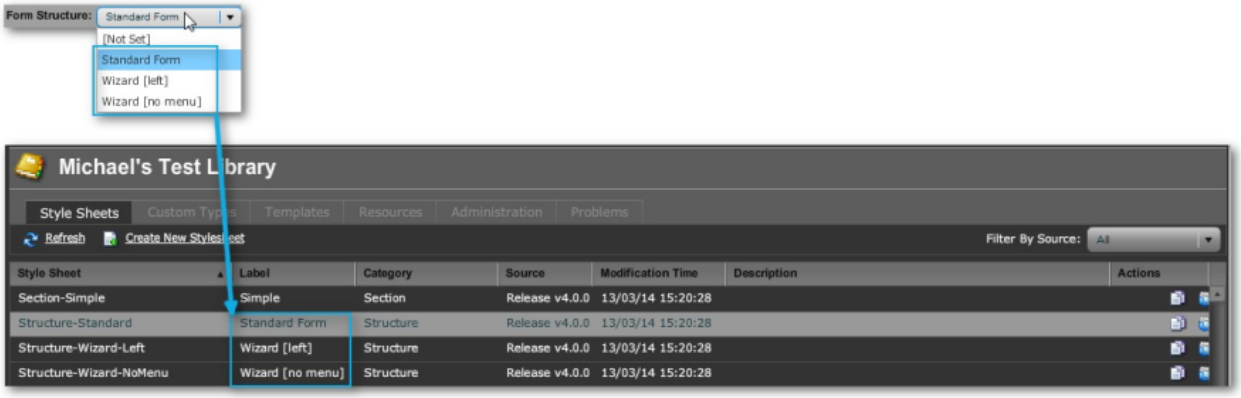
You access the Template editor through "Workspace -> <Account> -> <Organization> or <Library> -> Templates tab -> Edit Template action button". The editor itself has 4 tabs, 3 of which we show immediately below:



Style Options tab

The Style Categories panel of this tab allows for configuration of the Style Chooser dropdowns at the top of the Form Designer's wireframe.

The Categories are, as expected the stylesheetcategories and the dropdowns are populated by each stylesheet in the given category.



For the Details panel of a Stylesheet Category, the settings are:

Detail	Explanation
Category Name	Cannot be edited. Derived from the Stylesheet category chosen for a Stylesheet Chooser dropdown
UI Label	The caption appearing next to the stylesheet chooser. For example, we here have "Form Structure" for the Structure category.
Locked (checkbox)	If unchecked, the category's chooser appears in the form wireframes. If checked, the category gets no visible chooser in the wireframe.
Default Stylesheet	The name of the default stylesheet in the chooser menu. Here "Structure-Standard", giving "Standard Form" as the default in the dropdown.
Restrict Stylesheets (checkbox)	If checked the chooser is populated by only a few of the available stylesheets in the category. A textarea appears, in which you list the stylesheets used in the restricted menu..

Editing the XML Source of the Template

Currently, though the set of stylesheets inherited to the template from the stack determine the layout and look of the template's content, the actual content of the form is dictated by the form's XML Source. This you can view and edit in the XML Source tab.

Editing this is a job only for the technically competent in this area, otherwise you could do a lot of damage. However, for those who feel up to attempting this, we have included the TemplateSchema at the back of this guide.

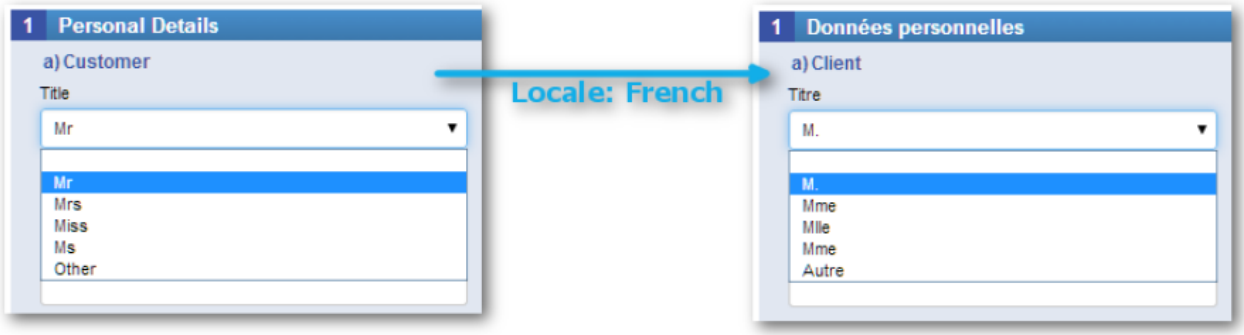
Localization

Overview

Composer's Localization feature allows you to provide your forms in multiple languages. Earlier in this guide, you would have seen amodestdemooflocalizationinaction.

For localization to be successful, you must have a native-speaker to edit the localization resources. For a rough draft you can use Composer's auto-translation feature (which uses the Google Translate engine), but if you have ever used Translate to render a foreign language web page, you would know that the results are less than convincing.

But once you have obtained a decent translation of a form, you will have no need to lay out the form anew for new supported languages (or "locales"). Even dropdown menus, and other widgets involving choices, will reconfigure to reflect the language change:

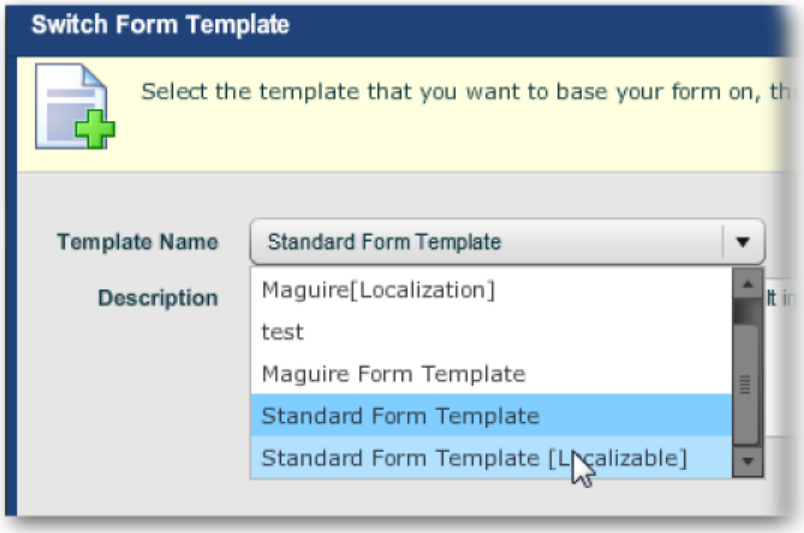


Activating Localization

The template used for a form must support localization. So, either you switch the form's template to one that does support localization, or you activate localization support in the template.

Switching Template

Done in "Workspace -> <Account> -> <Organization> -> <Project> Forms tab -> <Form> -> Switch Form Template button". Usually templates with localization activated indicate this in their title in square brackets.



You then can move to the next step: SettingTarget Locales

Activating a Template

You can also activate Localization in an existing template. Go to "Workspace -> <Account> -> <Library> or <Organization> -> <Templates> tab -> <template> -> Edit Template action button" which brings up the Templateeditor.

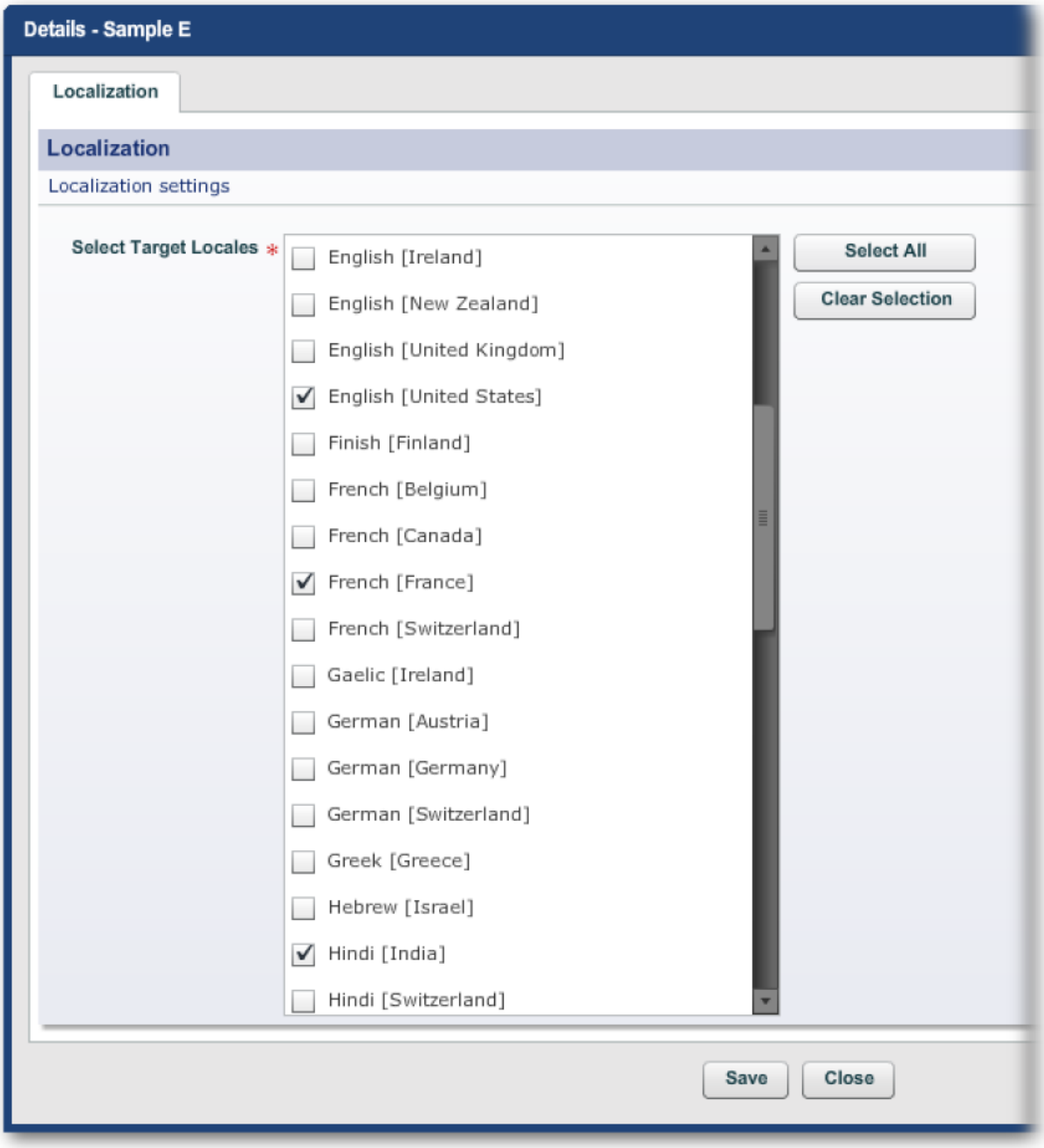
In the editor, "Style Options tab -> Quick Style Changes Options panel -> Enable Localization Support button".

Setting Target Locales

You have to configure Localization for each form.

Open the form in the Form Designer and go to the Localization tab ("Workspace -> <Account> -> <Organization> -> <Project> -> <Form> -> Localization tab").

Click on the "Set Target Locales" link. This brings up a dialog for selecting locales.

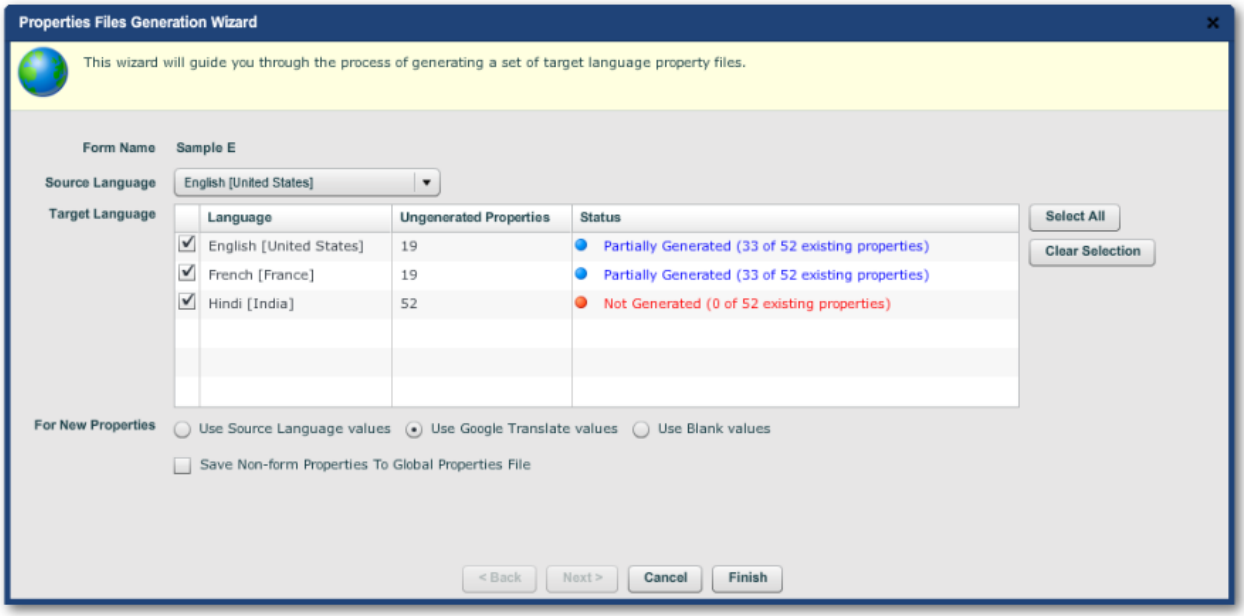


Generating Keys

You then have to generate keys for the form's elements. This is simply done by clicking on Generate Keys link in the form's Localization tab and completing the brief wizard. You can then move on the next step.

Generating the Properties File

Click on the Generate Properties File link in the form's Localization tab.



The "For New Properties" radio buttons and checkbox:

Use Source Language Values

create the new files, but leave the values the same as the original locale and let the (human) translator change the values to the new locale

Use Google Translate Values

and use these values as a rough draft for the translator

Use Blank Values

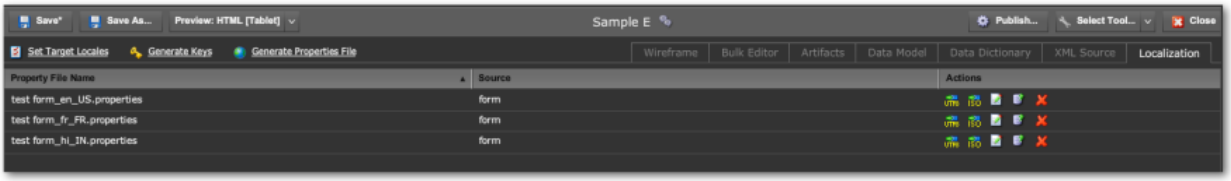
let the translator fill in the blank values. This means the translator will have to have access to the Properties file of the original locale of the form.

Save Non-form Properties to Global Properties File

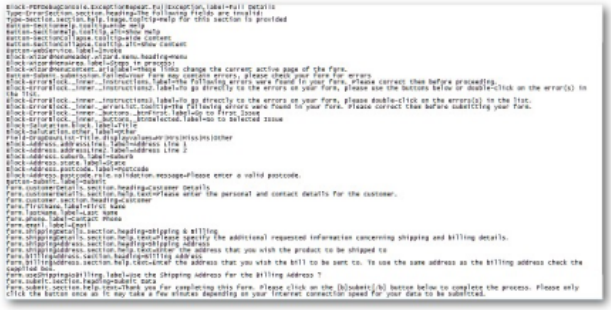
This means that a global properties file is created for each locale at the Organization's internal library resources level. It lists only the template's common properties (such as "Type- Section.section.help.image.tooltip" or "Button-SectionCollapse.tooltip" and so on).

Form properties are the content of the individual form, such as "form.customer.section.heading" or "form.firstName.label". Form properties are objects that begin with "form.".

These property files for form properties will appear in the Localization tab of the form:



Their contents look as follows:







The translators then can edit the translation of the properties. Be careful, though, that the diacritics and glyphs of any script (here Devanagari for the Hindi locale) are handled properly by the text editor used by the translators.

Deployment Options

The Localizations of the form can be deployed to different instances of the parent form in Transaction Manager, or for that matter to different portals.

Stylesheets

Composer | Form Builder | v4.3 & Higher

Overview

Stylesheets are the building blocks to any template. This also means that there are a great number of these and they have many categories. You are even free to add to the list of categories (which you do in the "Save Stylesheet as..." dilaog shown below).
Stylesheets can be simple where they specify a few properties for a single element type on a form, for example a "rounded button".
They also can be compounded, built out of a set of simple stylesheets.
In Composer, all styles. whether simple or compound, are defined by a stylesheet. These are sometimes called "styles".

Typed Styles

A typed style is a style that applies to all objects of a particular type. For example, by default, all Buttons in your form have the same background color because there is a Typed Style for Button that defines the foreground and background colors for all Buttons. The same applies to all object types.
Note: Composer uses a fully object-oriented type hierarchy. Some styles apply to "base" objects in the type hierarchy. For example, a Text Field is a sub-class of Field. Properties can be styled at either the base-class level (in which case it applies to all objects of that base-type) or at the sub-class level (in which case it applies only to objects of that sub-class). This is the reason for the chains of names separate by
dots. Even if you do not grasp this concept, you will still find styles useful.
Normally, every object you may use in your forms will already have a typed style associated with it, which was created by the person who designed your organizational stylesheet.
There are many typed styles needed for any template, be it designed by Avoka Technologies for your organization or for one of the standard libraries that ship with the product. In either case, a full set of typed styles becomes available to an Organization after a library is assigned to the organization. These are read- only.

Named Styles

In most cases you want all fields of a particular type to always look the same on all forms. (And that is why we set up typed styles for each field type.) But occasionally, you will want to override some of the styling for a particular type of field. For example, there are two ways to represent a Button on a form — either as a traditional push-button, or as a flat mobile-friendly style of a text-only button. Though they are just two slightly different styles of buttons they have different foreground and background colors, fonts and so on.

You select these named styles for a form object by "Edit Properties -> Styling -> Style sets". The available stylesets and their choices (for example, "Button Sizes", "Button Shapes" and so forth) are determined by the Named Styles associated with the Typed Style of the object.

Tip: It's generally a much better idea to define and use a named style, rather than explicitly setting "magic values" that you have to remember, such as sizes, colors and other properties. Not only does this mean you don't need to remember the values, but also, if you ever want to change the values, you can do so in one place.

Another example of using named styles is sizing of fields. Rather than explicitly setting widths in inches or mm, you can create named styles such as "large", "medium", and "small".

Stylesheet Categories

Stylesheets can now be categorized (either for convenience or for template choices, as we will see). You can use the already-existing categories or create your own.

Under the covers, you can define any number of stylesheet categories, and apply them in any order. Each category can define any number of styles sheets, each of which will appear in the drop down lists. Avoka has defined a standard set of stylesheet categories that work for many customers. you can define your own using the "Save Stylesheet as...." dialog shownbelow,

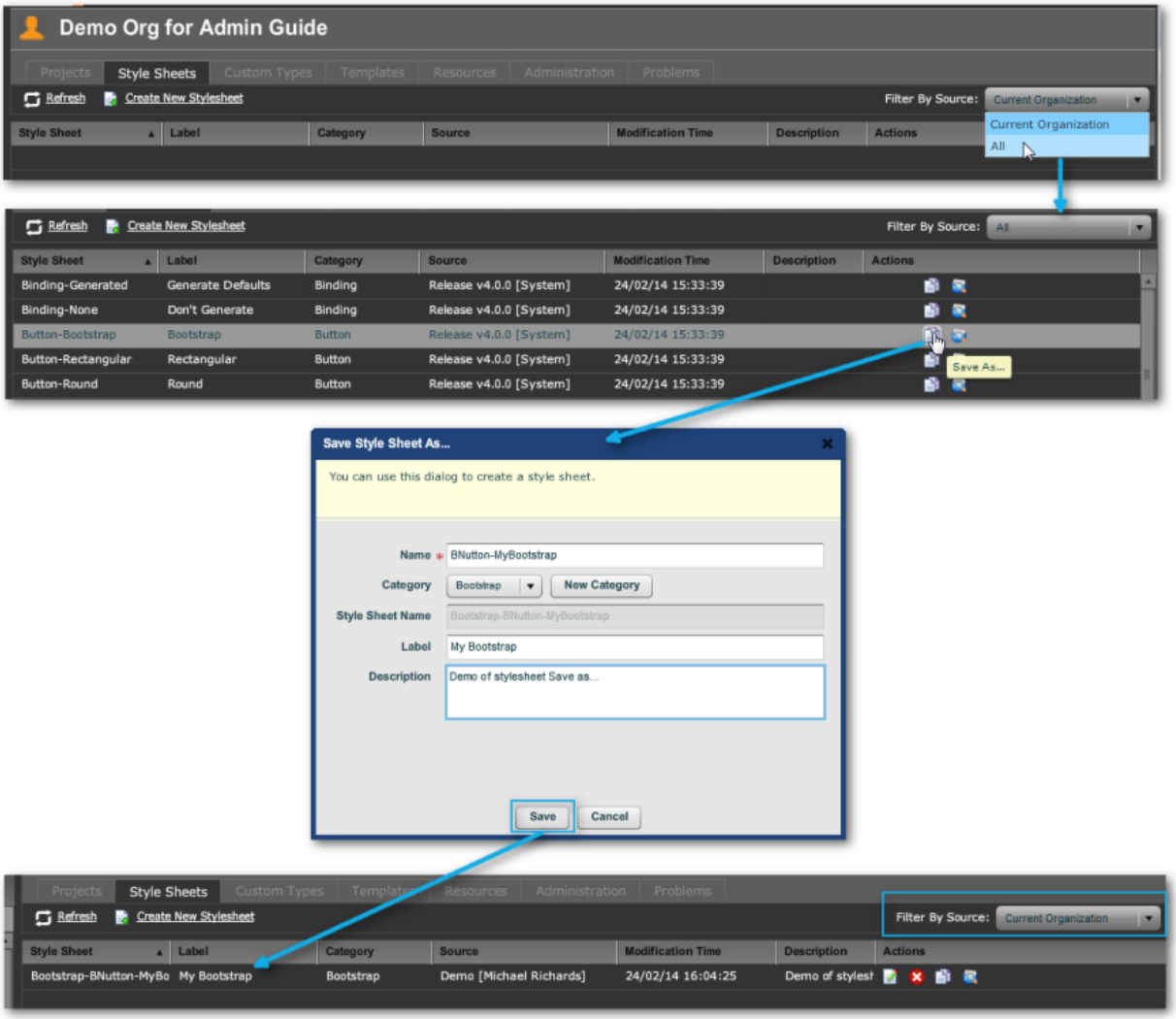
The standard stylesheet categories that Avoka has defined go from the most general (overall form structure) to the most specific (colours and locale). However, it possible to define other stylesheet categorization schemes for organizations based on their needs. For example, it is possible to define an organization's template with no stylesheet categories at all – all forms would look exactly the same. Or perhaps with a choice of only two stylesheets in a single category. Or on the other extreme, you could potentially define a different stylesheet category for each different type of component.

The standard stylesheets provided by Avoka in the default templates are more to show the capabilities of the system – most organizations will only define a very small number of stylesheet combinations that represent the styles of forms in their organizations. On the other hand, some organizations, particularly those that "white label" their forms, may define a very large number of different stylesheets for each of the sub-organizations that they represent.

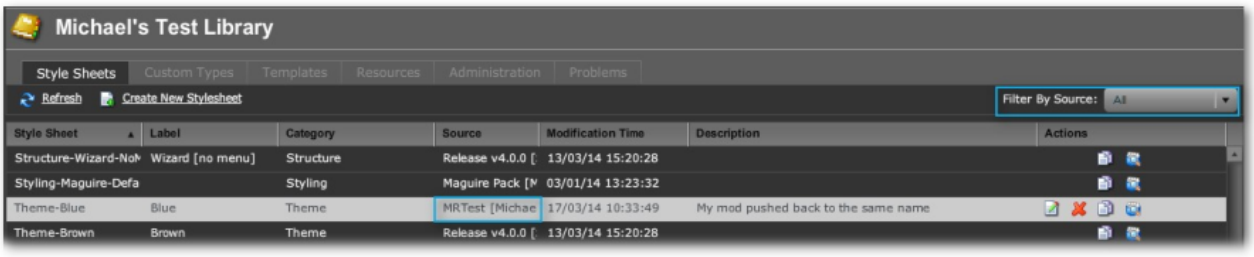
You then can, if you want, associate these categories with the StyleChoosersofthetemplate.

Editing Stylesheets

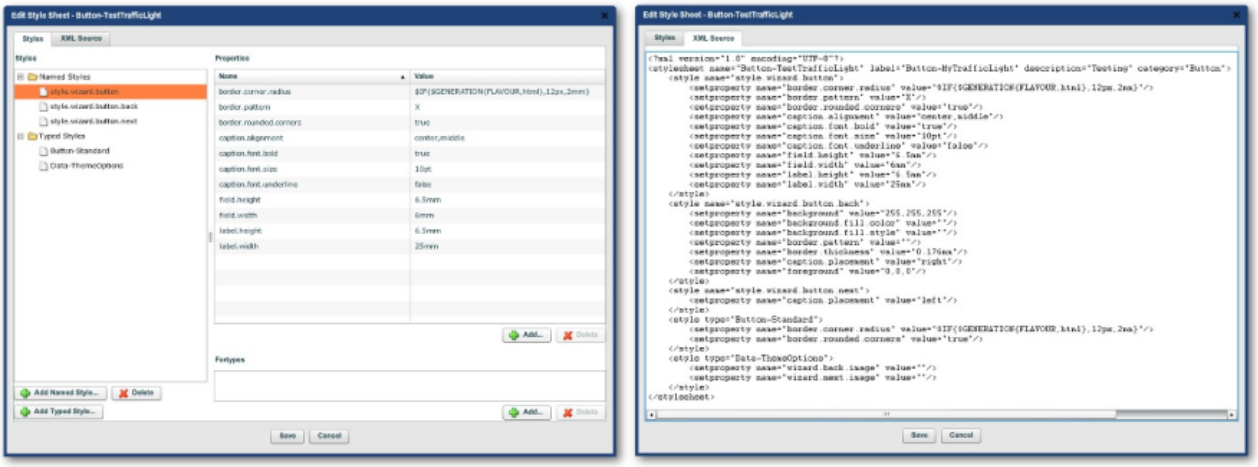
Location: Worksheet -> <Organization> -> Stylesheets tab or
Location: Worksheet -> <Library> Stylesheets tab
By default, an Organization can access the Account's stylesheets. These are all read-only. You can perform a "Save as..." on each of these so in that you can edit the new versions and also assign them to nominated libraries.



This is the way to populate both Libraries and Organizations with bespoke stylesheets.
So, you can replace a standard-issue stylesheet (from the release, say) with your own in the library by saving the stylesheet with the same name. For example, if I edit "Theme-Blue" and give it the name "Blue" and keep the category "Theme", my modifications are saved as Theme-Blue (but **only** in this particular library) and will affect all form elements that use the "Theme-Blue" stylesheet from this library in the stack. The entry in the Stylesheet tab for the Library shows the source of the style:



Once you have saved a stylesheet, you can edit its properties (double-click on item in the list or use the edit Action):



All of the values of the Properties of the Named Styles and Typed Styles are clickable and editable. You can also add and delete styles and properties and even edit the XML (though do not attempt to do so unless you know what you are doing). The Edit Stylesheet function is therefore very powerful.

Customizing Stylesheets

The ability to customize stylesheets is one of the most powerful features in Composer, for several reasons:
You can make global changes to your forms in a single place, and this will be automatically applied to all the forms that use that stylesheet.
You can control the scope of the changes by being careful about which library's stylesheets you change. For example, changes to the internal library of the Organization will affect the whole Organization. You can alter the scope of changes to other libraries by altering the Search Path and the Search Order for Organizations.
Even so, the scope Stylesheet changes seem to naturally affect Organizations. Trying to have Projects in the one Organization pick up different stylesheet versions is complicated and difficult.
Stylesheets hide (or encapsulate) a lot of the advanced properties associated with individual widgets. These advanced properties don't even appear in the Composer by default (you can turn on Advanced Properties in the Property Dialog), which means that you don't have to worry about them. You just get on with the job of designing your form.
The stylesheet embodies your organizational style guide, ensuring all your forms conform to the style guide. Just use the standard stylesheet, and let Composer do the rest. If you organizational style changes, simply change the stylesheet, regenerate all your forms, and you're done.
Tip: While stylesheets and their close relative, Templates, are very important, they are also moderately complex. We strongly suggest that you contact Avoka if you require either a custom stylesheet, or a modification to one of the standard stylesheets. We can either develop the stylesheet for you, or provide some customized training to enable you to make your own stylesheet changes.

Form Designer Advanced Features

Composer | Form Builder | v4.3 & Higher

Overview

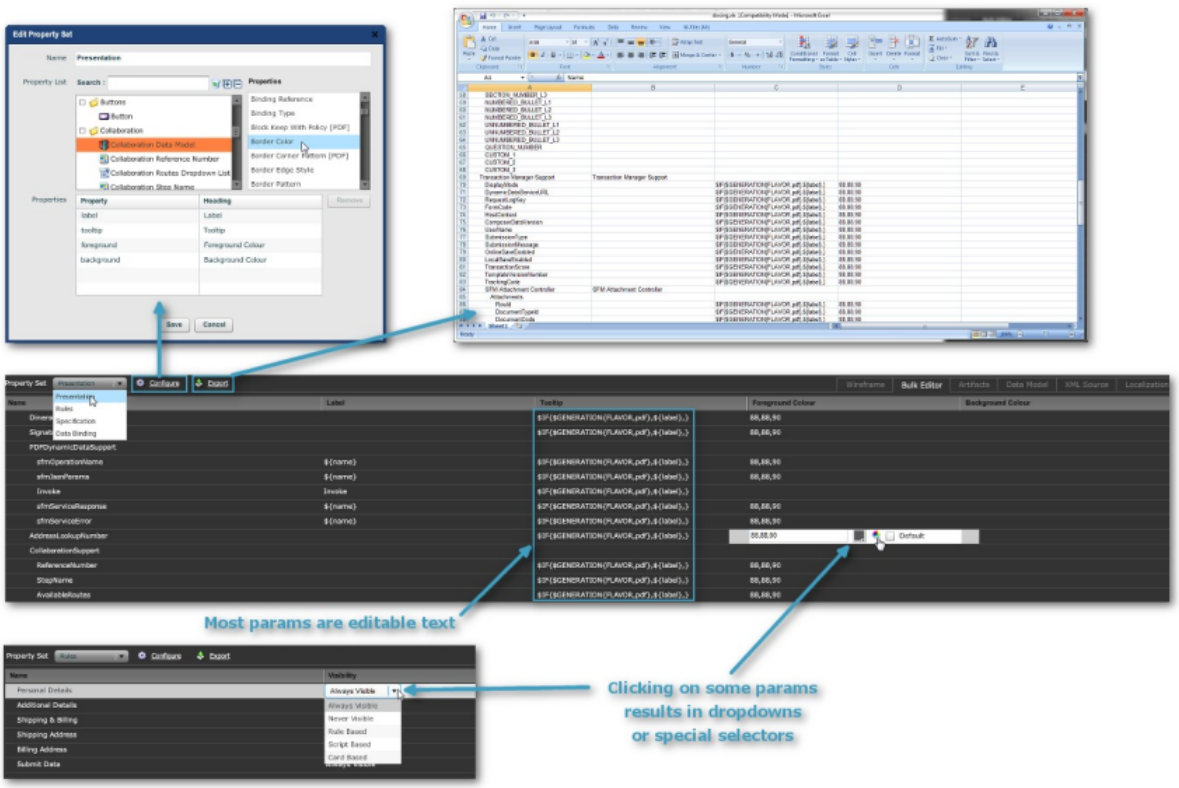
Besides the Wireframe tab of the Form Designer, there are:

The Bulk Editor tab The Artifacts tab The Data Model tab The XML Source tab The Localization tab

These do not provide further functionality than that already given in the Edit Properties dialogs and panels, but instead organize the parameter and settings of the form into tables. Some of these allow parameter settings to be changed and can be more convenient than going in and out of numerous Edit Properties dialogs for the fields and nodes on the form. The tabs provide an alternative and more comprehensive view of the parameter settings of the form than the atomic and fragmented glimpses afforded by going through the properties of a large number of individual form widgets.

There are at present some limitations: you cannot now alter the settings for the same type of parameter on a number of form elements. But, the ability to be able to survey the setting on, for example, all the Composer scripts within the form is a vary valuable facility.

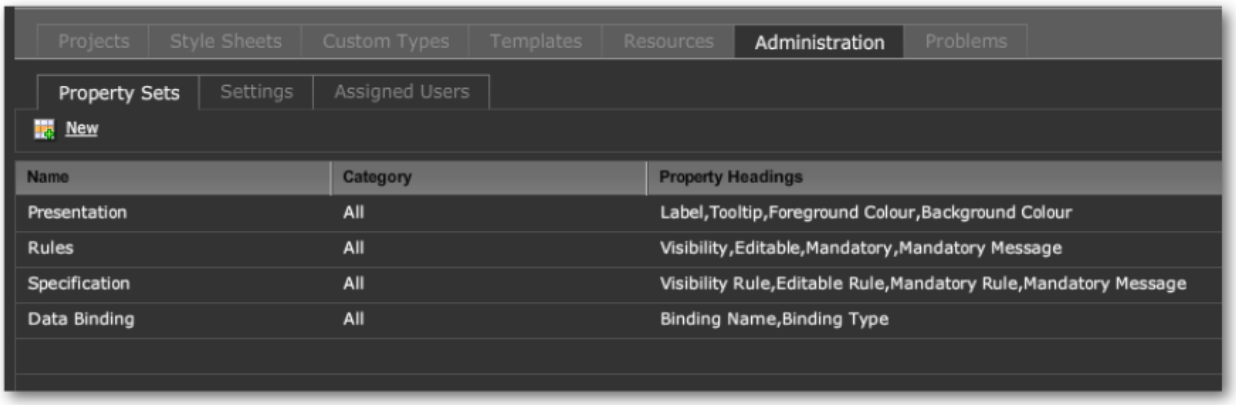
Bulk Editor



In the Rules Property Set listing, you can select "Script Based" under Visibility, Edibility and so on, and then use the "Edit" button that appears in the row to bring up the "Script Editor", a very convenient way to access this dialog.

Property Sets

The form's parameters are arranged in Property Sets, which belong to the Organization. You edit, and create new, property sets in "Workspace -> <Organization> -> Administration tab -> Property Set tab".



This the only place in the UI that you can create a new set. However, you cannot at present configure the new set. You must configure it in the Bulk Editor of one of the forms in the Organization, using the Edit Property Set dialog from the "Configure" link at the top of the Bulk Editor page. Here you select a property from the Property List (which just like Widget Palette in the Form Editor) and then double-click on the Property you want in the Property Set. The property then appears in Property list in the lower half of the Edit Properties Set dialog. You can change the wording of the heading of the resulting Property Set column in the Bulk Editor

Artifacts Tab

This lists, and allows you to download, files associated with the form. Here is a partial list of the files:

<form name>.xls

A spreadsheet of all the form elements and their properties, including the Nuts & Botls

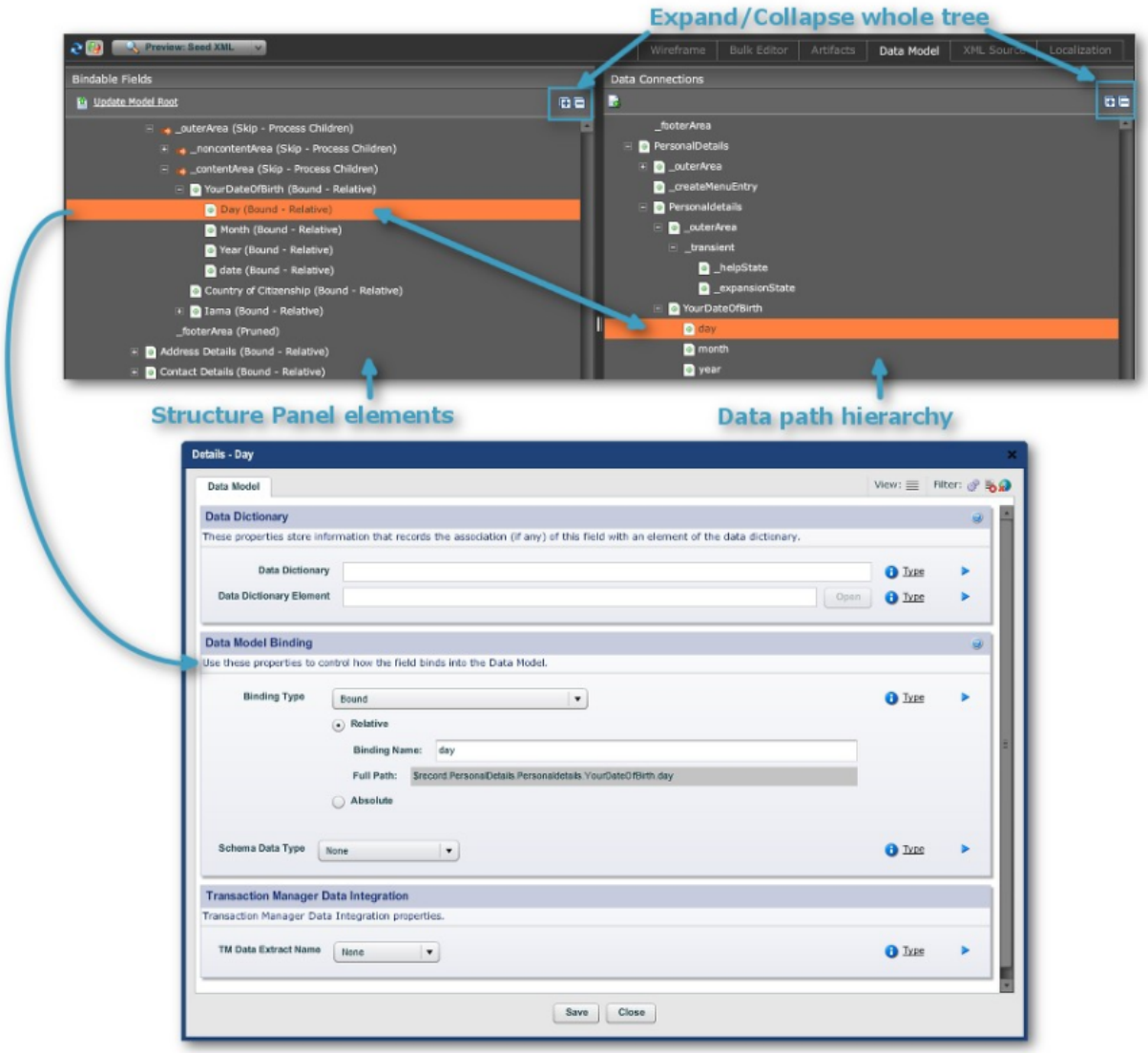
<form name>.xml

The form definition in XML.

<form name>_info.properties

A text file giving listing the creation date, time and time zone, and the previewtechnology.

Data Model Tab



The left-hand panel gives the various bindings (bound, unbound, relative, absolute) of the fields in the structure; the right shows the field's position in the data hierarchy. Select a field in one of these two panels and it will be selected in the other panel.

Double-clicking on a field in the left-hand panel will also bring up a dialog with the Data-related panels to be found in the "Edit Properties" dialog, though conveniently grouped into a "Data Model tab" created for this part of the User Interface.

The hierarchy depicted in the right-hand panel is the data hierarchy. In Composer, the data hierarchy is for the most part a reflection of the form's structure. This means that the data structure of the field bindings within a block is, for better or for worse, similar to the way the fields relate according to layout and block groupings. It is beyond the scope of this document to show how to change these relationships without altering the layout structure of the form itself.

XML Source Tab

This tab is an editable display of the XML definition file of the form. You can (provided you the rights to do so) save you changes, or you can rollback the last edits you have made to the code. Only do this if you are sure of what you doing; otherwise you can do serious damage to the form.

The XML frquently has Character Data ("CDATA") blocks, which are code that is passed straight through the XML parser without it reading the block (which it would not understand in any case). CDATA here is used, in the main, to pass JavaScript fragments.

Localization Tab

This is explained elsewhere in theguide.

Transact Composer User Guide and ReferenceAccess & Security

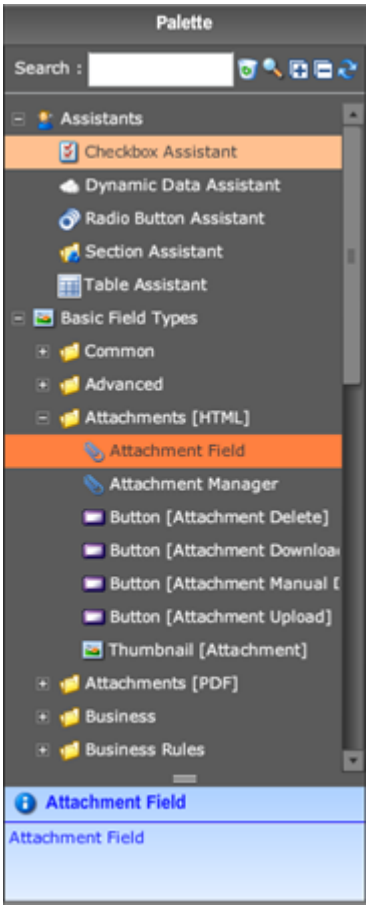
Attachment Fields Overview

Composer | Form Builder | v4.3 & Higher

Prior to Avoka Transact Version 4.1, capturing attachments was only facilitated by the Self-Service Portal once data entry was complete and the 'Submit' button had been clicked in the form. With this approach attachment capture is an Independent step in the transaction process, completely separate to the form, and this sometimes makes the transaction a little disjointed. While this flow is still supported in 4.1, an alternative has been introduced to enable form designers to include file capture inside the form at the time that it is relevant. This is facilitated by a new Composer widget called the Attachment Field.

Finding the Attachment Field in the Palette

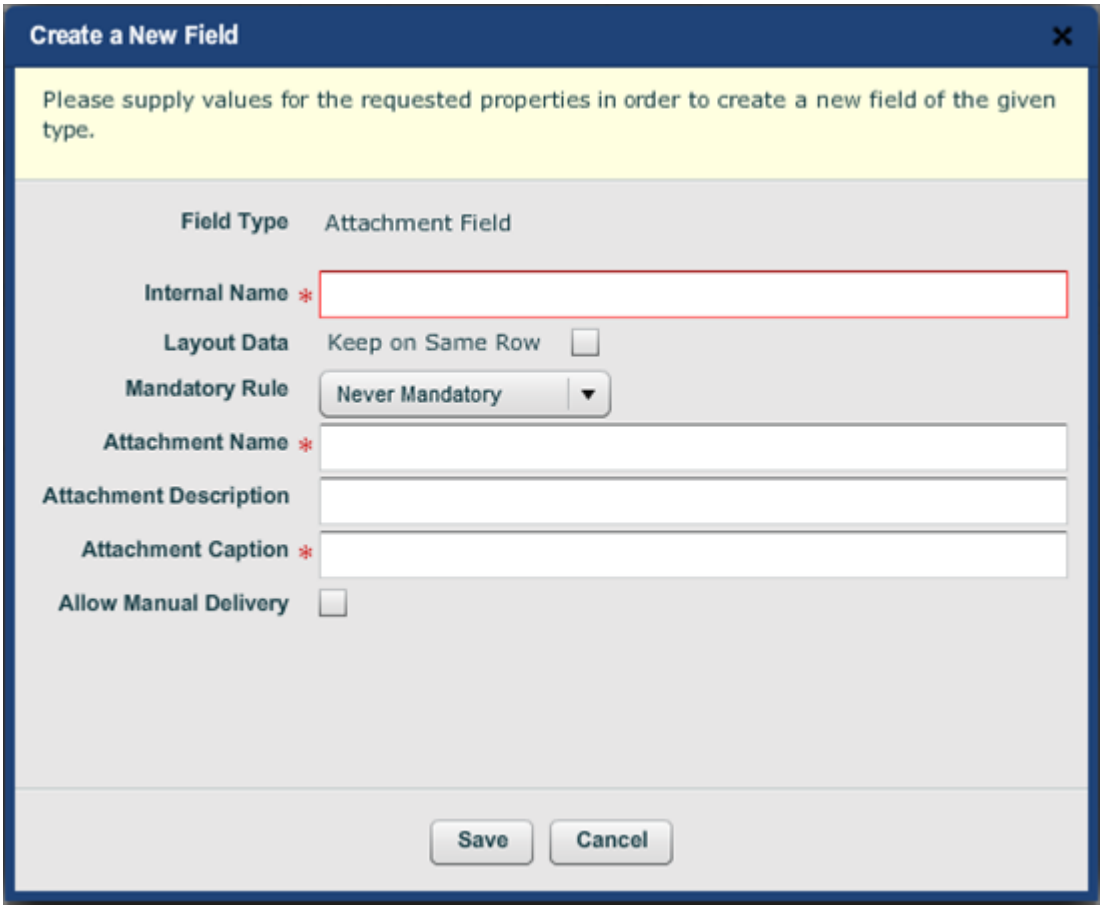
You can find the "Attachment Field" under the "Attachments" category.



Configuring the Attachment Field

Initial Config Options

After you drop the field in a form you will get the following initial options, three of these are mandatory (with the red asterisk).



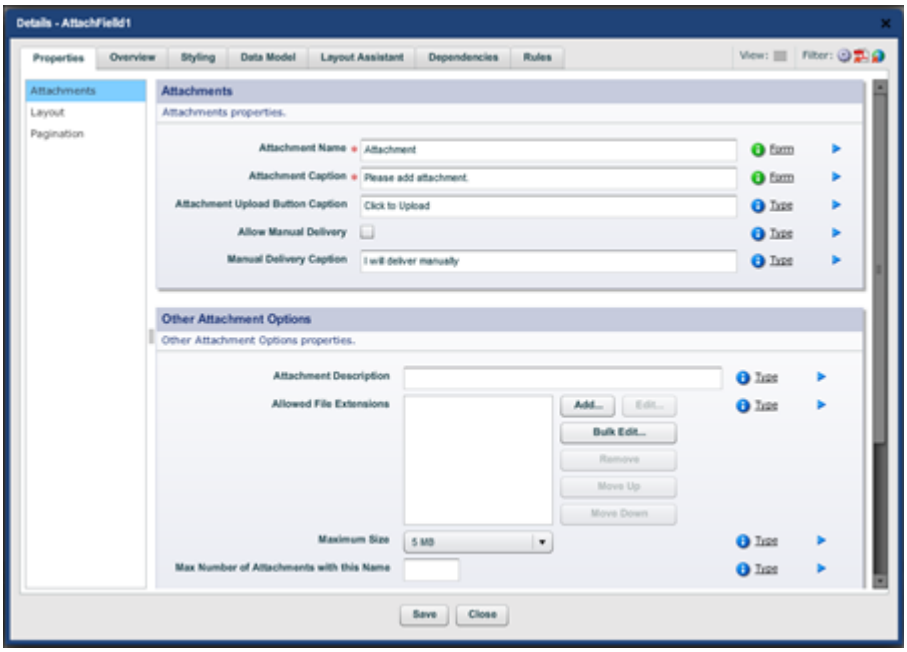
Basic Mode Config Options

Within the field properties tab there is an "Attachments" category which contains the following basic mode options:

Name	Description	Image
Attachment Name	This is the attachment name that will be sent to TM.	
Attachment Caption	This is the caption text that will display at the top of the field.	
Attachment Upload Button Caption	This is the caption text that will display in the upload button.	
Allow Manual Delivery	If set to true, a link button will appear below the upload button which allows the user to delivery the attachment manually.	See images for "Manual Delivery Caption"
Manual Delivery Caption	This is the caption text that will show in the manual delivery link button.	
Attachment Description	This is the attachment description that will be sent to TM.	
Allowed File Extensions	This is where you can set the allowed file extensions. If set, an error message will return if the user uploads a file with an extension not in the list.	

^ Will only allow .png files.

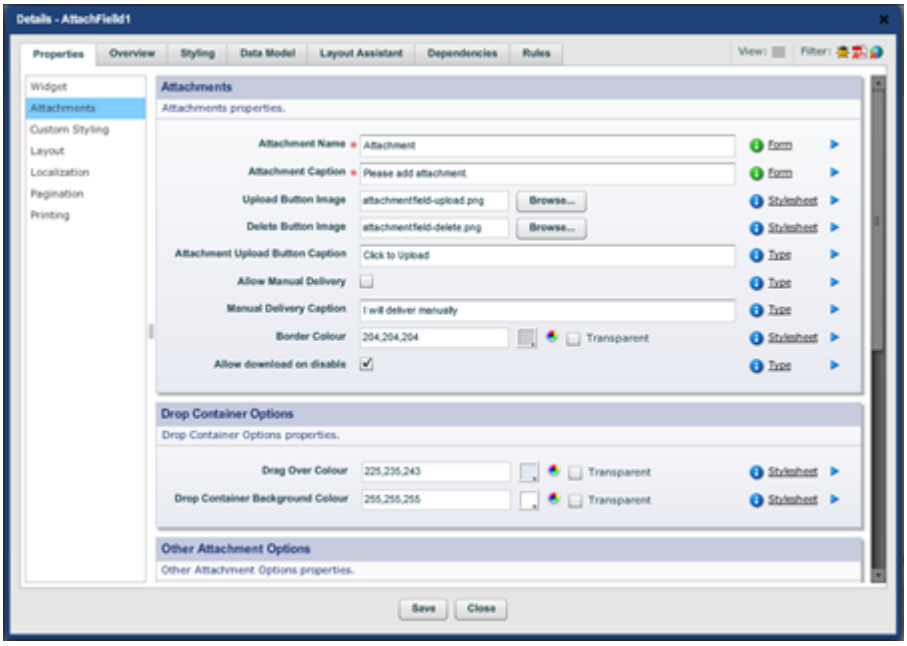
Maximum Size	This property allows you to set the maximum file size allowed. Ranging from 0.5MB to 25MB. An error message will return if the user uploads a file with a size beyond the limit set.
The Max Number of Attachments with this Name and use Default Progress Indicator	These properties should be left as default for the 4.1 release.



Advanced Mode Config Options

Name	Description	Image
Upload Button Image	Allows the user to change the upload image which is shown in the initial field state	
Delete Button Image	Allows the user to change the delete image which is shown in the completed field state	
Border Colour	Allows the user to change the border colour for the attachment container, this can be done in a style sheet using the fields type style.	
Allow download on disable	Set to true by default, this allows the attachment download button to still be active when the field is in read-only mode.	
Drag Over Colour	Allows the user to change the colour used when a file is dragged over the container , this can be done in a style sheet using the fields type style.	
Drop Container Background Col-	Allows the user to change the Drop Container background Colour , this can be done in a style sheet using the fields type style.	

our



Testing in Preview Mode

Only zip files are allowed in preview mode.

The User Experience

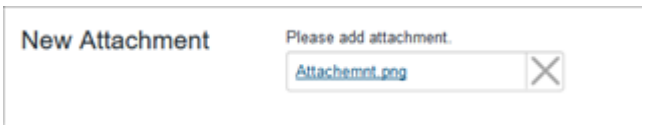
Interacting with the Attachment Field

To capture an attachment, add the Attachment Field to the form in the location that you wish to capture the attachments. Attachment fields support visibility, editability and mandatory rules like standard fields.

In its empty state, the attachment field shows the 'Click to Upload' button which launches the standard browser file dialog. When an attachment that complies is provided the upload button disappears and the file name is populated into the field as a clickable link that will trigger a download if clicked.

This link will be active, even if the field itself is disabled. The form can be cleared again by clicking the delete ('X') button.



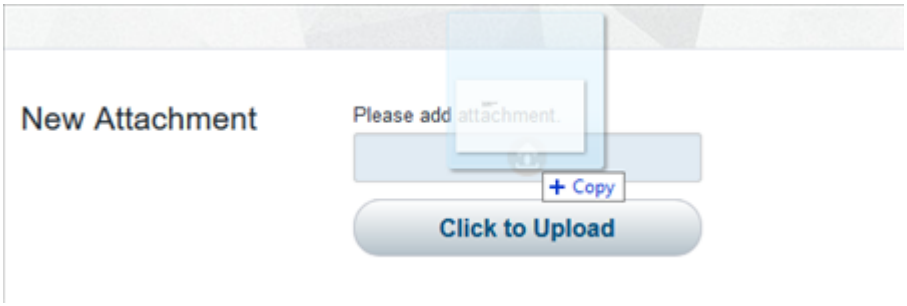


Form designers can configure the attachment field to support the manual delivery option where users indicate they will provide the attachment by conventional means.

> **NOTE**: As of version 4.1, the attachment field is not yet supported in TransactField although it is intended that this capability will be included in a later release.

Drag and drop support

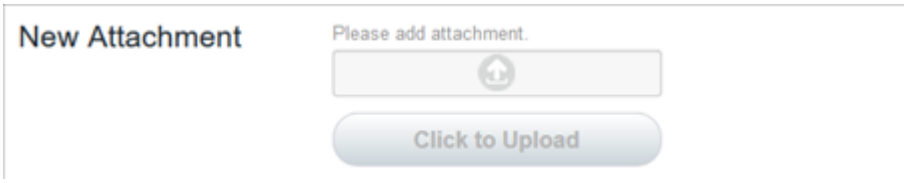
Drag and drop is supported in most browsers (IE8 and IE9 are not supported)



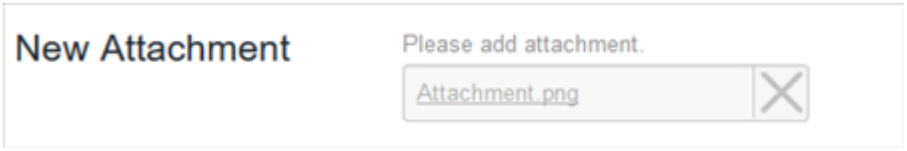
Read-Only Behavior

In read-only mode when the property "Allow download on disable" is set to true (it is true by default) the delete button is hidden and the download attachment button is clickable.

"Allow download on disable" set to true



"Allow download on disable" set to false



Exposing Attachments in Review & Approval Style Collaboration Jobs

Collaboration Jobs runs over a number of steps. The step name is passed by Transaction Manager to the form. Editability rules can be applied to attachment fields based upon the step name.

The initial applicant filling out a blank form usually wants full write access to an attachment field (upload / download / delete). However a reviewer checking the form may be required to view the attachment but not edit it. By configuring the editability rule to only disallow edits in subsequent steps, this scenario can be easily accommodated.

Your Attachment Field must have "Allow download on disable" enabled (default).

Support Notes

In TransactField, the attachment field displays as in receipt mode. In the future, the attachment field will include the capability to upload attachments within TransactField.

Obtain Form Details

Composer | Form Builder | v4.3 & Higher

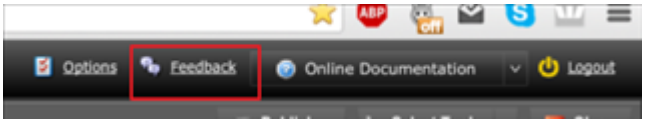
Once your system is in place, there will come a time when you would like to update forms, or request some mentoring or support on a form design. Throughout the life cycle of composer environment, you may rearrange folders or projects, take copies and backups of forms or change form names, it is really important to ensure that you are aware of which form is the current production form, and subsequently, communicate this to your developers or support team when requesting assistance.

If you need to restore or just clean up your environment, see restore the environment.

This short guide will show you how to get the details of the form you wish to request support or development on.

To obtain your form details for developers or support purposes:

1. Before you begin, ensure you have identified for yourself the current production form in the Composer environment. If you have followed the instructions in the linked article above, then you may have written in the description of the form "Current PROD form".
2. Open the required form in Composer
3. Click **Feedback** in the top right of the Composer environment.



4. Check your form details.



The idea behind this screen is to enable you to collect information about the form, the organisation and the composer environment you are using, to make it clear and simple for your development or support teams to identify which form you need assistance on.

5. Click **Copy Details to Clipboard** and then paste the clipboard into an email, or into your support ticket, To communicate these details. The sample of the details is shown below.

```
-- Form Details --
Account: Test Forms
Organization: Test Forms
Project: April 2014
Form: TestForm
Template Name: Template-Maguire

-- Build Details --
Version: 4.3.2
Build Date: 3 June 2016
Build Number: 47396
Build Type: cloud

-- Search Path Details --
Release Path: Release v4.0.0
Service Pack path: v4.0 Service Pack 2
Custom Path: Free Trial Forms__Updated Maguire Pack 4 Current

-- Browser Details --
Chrome/51.0.2704.103
```

6. Click **Close**.

Restore the Environment

Composer | Form Builder | v4.3 & Higher

we actively encourage all customers to put controls around development and promotion of forms to a live environment, through the use of multiple development, testing and staging (as well as the production) servers. Having separate environments means you can adequately test your work, and promote to live without affecting other services. Unfortunately, this doesn't always happen, and below is an example of a situation where we have assisted in solving a confusing and non structured UAT and Production environment mess.

Imagine this situation, you've just purchased our Journey platform and have built some forms. You didn't worry about purchasing a non production environment just now, because you don't have anything live, so you want to just go live with the whole system and your new forms.

We recommend purchasing your test and production systems together, so that a clean go-live can occur, however in this scenario, we were not able to take advantage of a test server.

Later, you decide it would be good practice to get a UAT machine and start developing forms and services on that. More developers come in, publishing forms to UAT from composer, creating copies of forms, publishing more things to UAT. Before you know it, your forms list looks like this:

- "Credit Card Application"
- "Credit Card Application (form)"
- "Credit Card Application 2"
- "Credit Card Application test"
- "Home Loan Application"
- "Home Loan Application (UAT)"
- "HL Application"

So, which one is the UAT proper version that you should export to production when you are ready? What if you have 100s of forms and it looks like this?

Don't let it get to this!

Firstly, don't let it get to this stage. We can do this by:

1. In Composer, Implementing a naming or versioning standard
   - I.E – all revisions of forms that are the current working copy should be denoted using "Current Version" for the Revision Name
   - Use the versioning tool within Composer and create your own processes to identify forms
2. Set up your composer security correctly
   - Developer
   - Account Administrator – able to unlock forms and add users/change passwords.
3. Setting up the composer development environment

- Have one folder per developer to be used as a sandbox area
- Create a "Backup" folder – This folder will contain a list of forms, denoted by the production form name, with the date appended that it was moved to the backups folder
- Production – production forms only
- Process for transferring forms:
- If changes are required, copy the current production form to the relevant sandbox project
- Make form changes
- Test changes
- Backup the production form to Backup project using the "Save As" functionality within Composer
- Then update the "Production" project with the newly updated form

4. Publish to Manager.
5. Ensuring that no one can publish directly to the Production TM environment from the Composer environment.
6. In Manager, Implementing a Form Code standard. For example, for a credit card application you may use the syntax – <BRAND CODE><FORM CODE> which could look like MAGCCA (where the brand is Maguire, and the form code for credit card application is CCA). Each form should follow this standard and this standard will depend on the makeup of your business. You may have geographical jurisdictions instead of brands so your code may be <GEOGRAPHY><FORM CODE> for example, AUSCCA (Australia), or UKCCA (United Kingdom).
7. Ensuring any debug or sandbox forms in UAT have the “Test Mode” flag set.
8. Lock your permissions so only those people who need to publish forms to UAT or Prod have the access to (Avoka have a set of recommendations on this).

To correct this issue:

1. You will need to set up some type of matrix using a spreadsheet in order to take note of the form name and code, and the corresponding MD5 value, and which server this exists on.
2. Remove connectivity between Composer and the Production Environment to ensure no forms can be publish to production directly from Composer.
3. Match the Form Archive MD5 value of Prod forms to UAT forms (these forms are evidence that forms have been correctly exported from UAT, and imported to production
4. Go to the Transaction Manager Administration Console
5. Go to Forms, Forms, and find the form in question
6. Click on “Properties” for the form version in the form dashboard
7. Choose the “Form Archive Info” tab
8. Take a note of the “Form Archive MD5” Value
9. Match this value to the suspected corresponding form in production

10. Where forms are in UAT and also appear to be in Prod, but the Forms Archive MD5 does not match, this is evidence that the form has been published directly to production.
11. An option in this case is to export the form from production, and import it into UAT. Then you can publish from Composer to that form in UAT
12. Once satisfied that the forms in prod are correctly reflected in UAT and Composer, forms in UAT can be made “inactive”. Likewise superfluous forms in Prod can be made inactive
13. After a period of time, forms in prod that are inactive can be deleted.

If these steps do not work for your particular problem, contact our support and ask for assistance.

Composer Account Administration

Composer | Form Builder | v4.3 & Higher

The Composer Administration Guide steps an administrator through features of Composer that manage and maintain their organizational hierarchy as well administration of users in their Accounts.

Aim of this Guide

Avoka Transact Composer has a set of administration tools, which allow for fine-grained control over Composer's configuration. There are also a number of maintenance tasks that you regularly should perform, so that your users can get the most out of Composer.

Notation

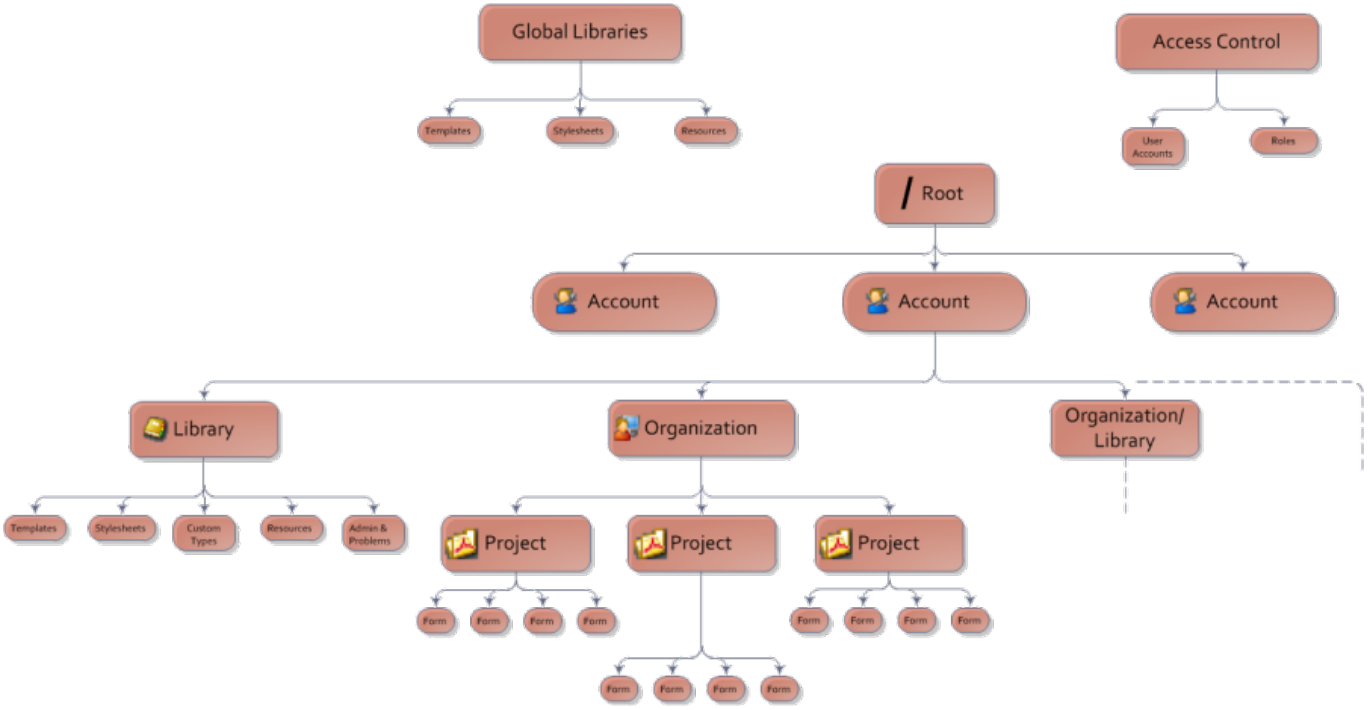
Throughout this administration guide, we use the notation "->" to indicate a location in the interface to carry out a task.; for example: "Access Control -> Users -> New System User"

Angle brackets ("< >") indicate the general name, for example "<Account>" for "Foobar Account" or "My Account" or whatever.

The Workspace

Composer | Form Builder | v4.3 & Higher

Workspace Hierarchy



The Root level is the top of the Composer environment; this is at the server level. You access Composer through a recommended browser and assigned URL If you have opted to host Composer yourself and not in the Avoka community cloud some of the issues of hosting your own environment may not be covered in this document.

The Composer workspace is organized into a number of Accounts. Above these lies the Avoka-administred Root level. You will have been assigned to one or more of these Accounts, which will be the areas you will be administering. The Account level usually corresponds to the Enterprise level in the real world. An Account can have many Organizations, usually corresponding to the department level in the enterprise.

Floating above the Root are the Global Libraries and Access Control. Global Libraries are, as it were, dynamic super libraries, in the sense that changes to these will ripple through the entire system of account-level libraries. More on libraries will follow.

We have placed, somewhat arbitrarily, Access Control outside of the Workspace, mainly because it lies outside of the Workspace structure (though not entirely, as we will see). Also, although you can create roles and assign users, all your access control settings are not held under the account level like, say, libraries.

In most usual account configurations, the end users are normally granted rights to set up Projects and create forms under them. The Project level is often used by designers as their personal space. However, you decide whether you will adhere to this scheme.

Creating New Organizations

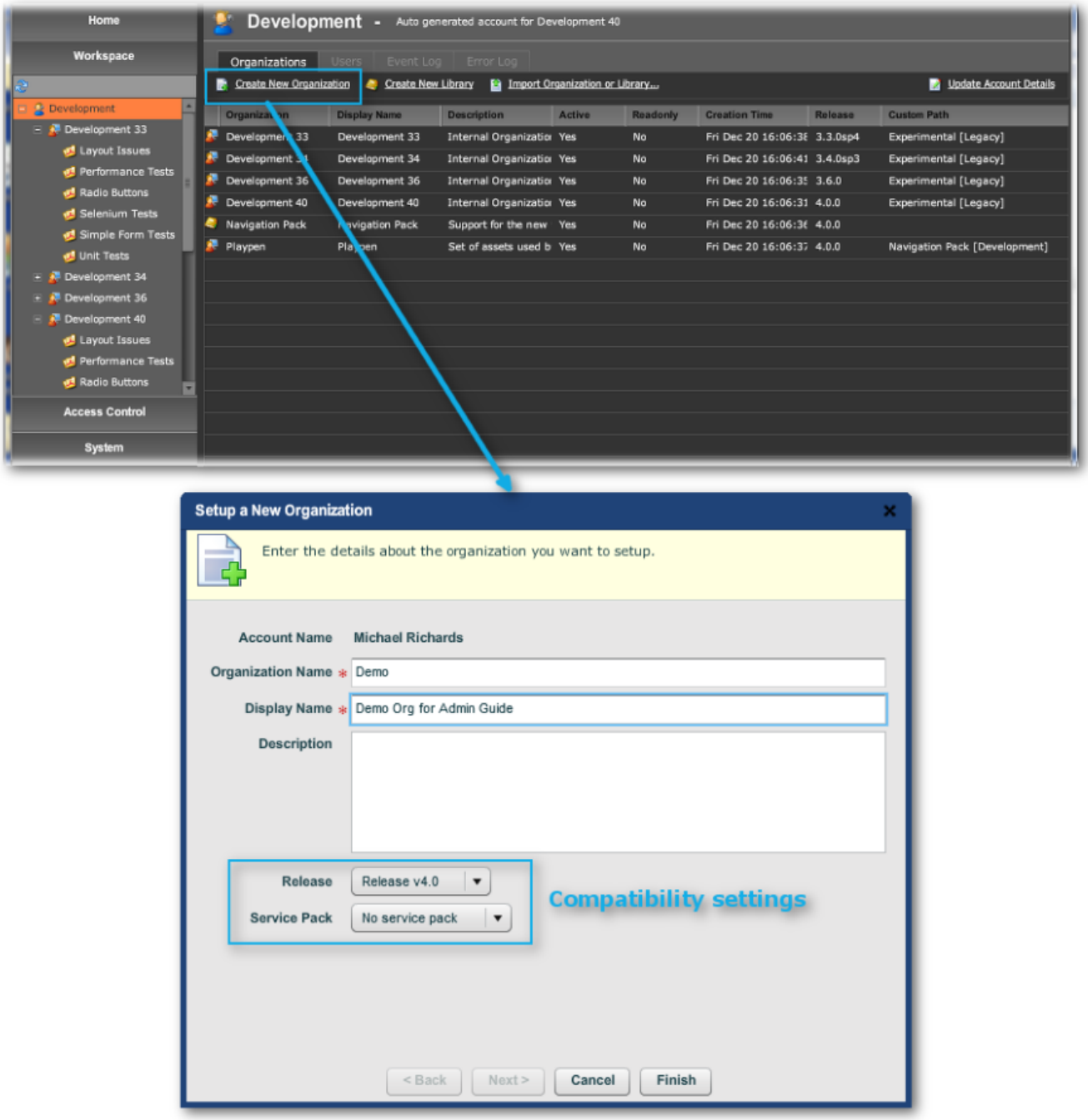
Composer | Form Builder | v4.3 & Higher

Creating New Organizations

Location: Workspace -> <Account>

Required Permissions: ORGANIZATION_ADMIN, ORGANIZATION_EXPORT, ORGANIZATION_IMPORT

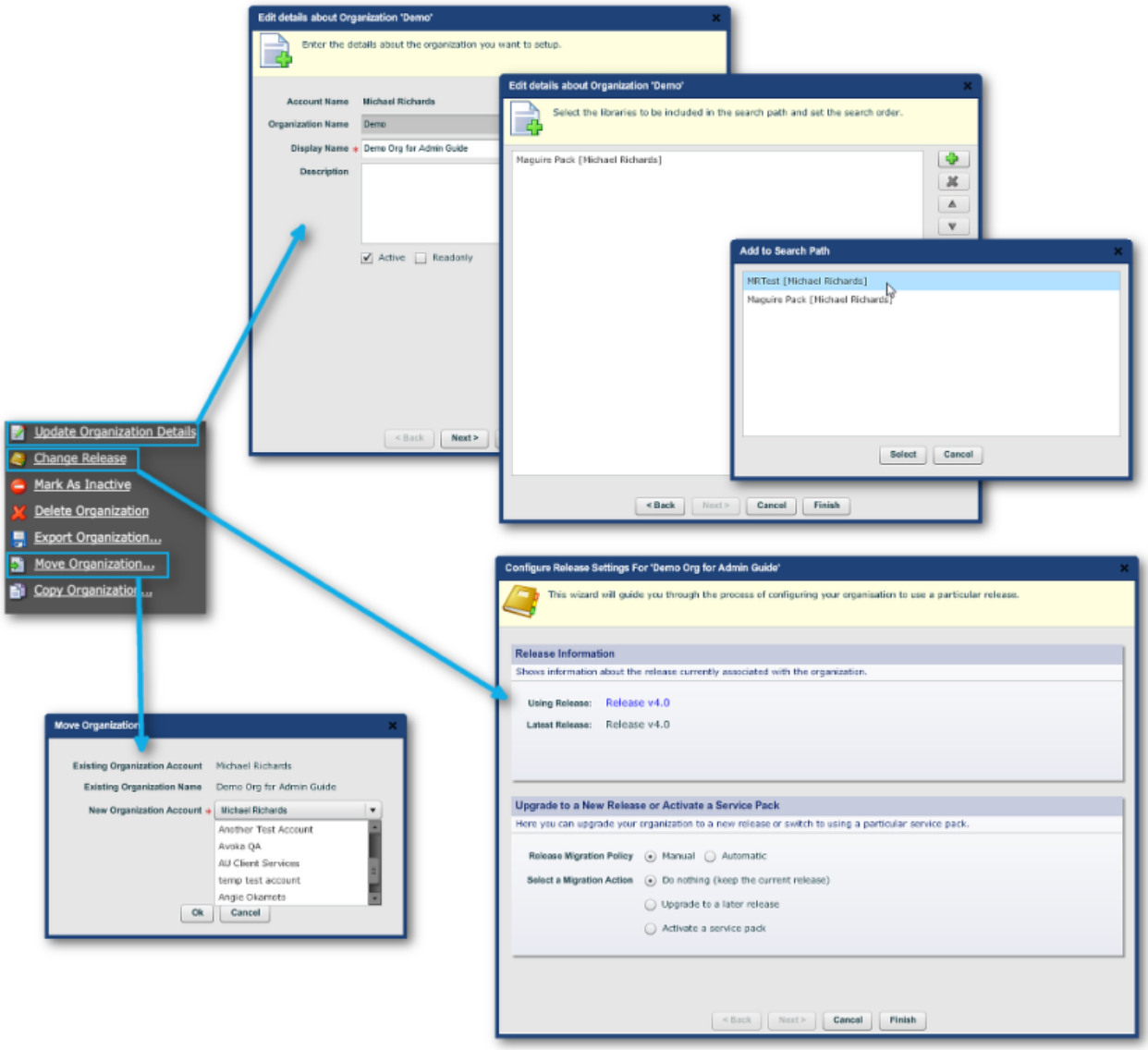
Roles: System Administrator, Avoka Engineer, Account Administrator, Avoka Partner



You can also create Organizations at the Location: System -> Organization Management.

We strongly recommend that you set the release compatibility to the latest version and service pack; however, you have the option of going with earlier versions.

The Organization needs to be configured, carried out at the Account Level of the UI:



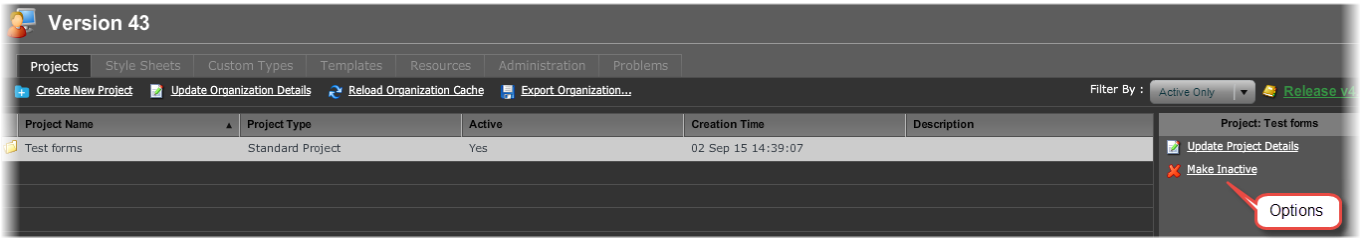
Creating New Projects

Composer | Form Builder | v4.3 & Higher

Location: Workspace -> <Account>
Required Permissions:PROJECT_ADMIN
Roles: System Administrator, Account Administrator, Avoka Engineer

You create new Projects at the Location "Workspace -> Account -> Organization" of the UI and at that level only.

The Project Options from the Organization level look like this:



You also have some editing actions:

- Update the Project's details
- Make the Project inactive (which is tantamount to deleting the Project)

The Project level of the UI looks like this:



There are 2 further tabs at this Project level. These are inspectors where you can view the allocation of templates and users to the Project,

- Assigned Templates: you can filter templates by Current Organization or for all templates in the environment.
- Assigned Users: you can block individual users so that they will not be assigned to new forms

Creating & Importing Libraries

Composer | Form Builder | v4.3 & Higher

An account can have many Libraries. Each of these is a collection of assets, i.e. templates and so forth.

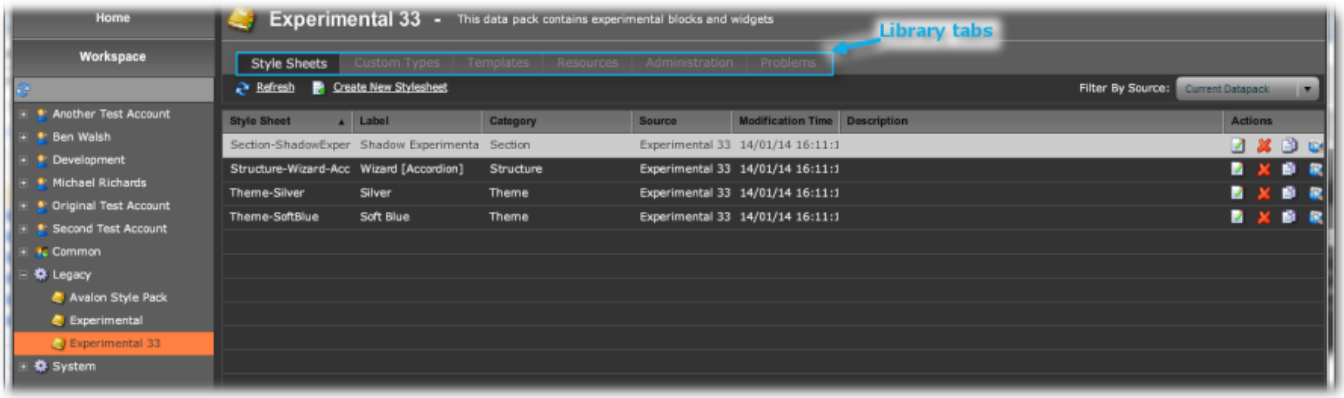
A Library or Libraries can be assigned to Organizations.

So, it becomes possible that different Organizations (or even Projects) in the one Account can have different designs to the their forms through the use of different Libraries.

This flexibility was not easily configurable in earlier versions of Composer.

The various releases and libraries in Composer form a stack. We strongly recommend that you read "Library Advanced Features" in the Composer Guide.

Library Level of the UI

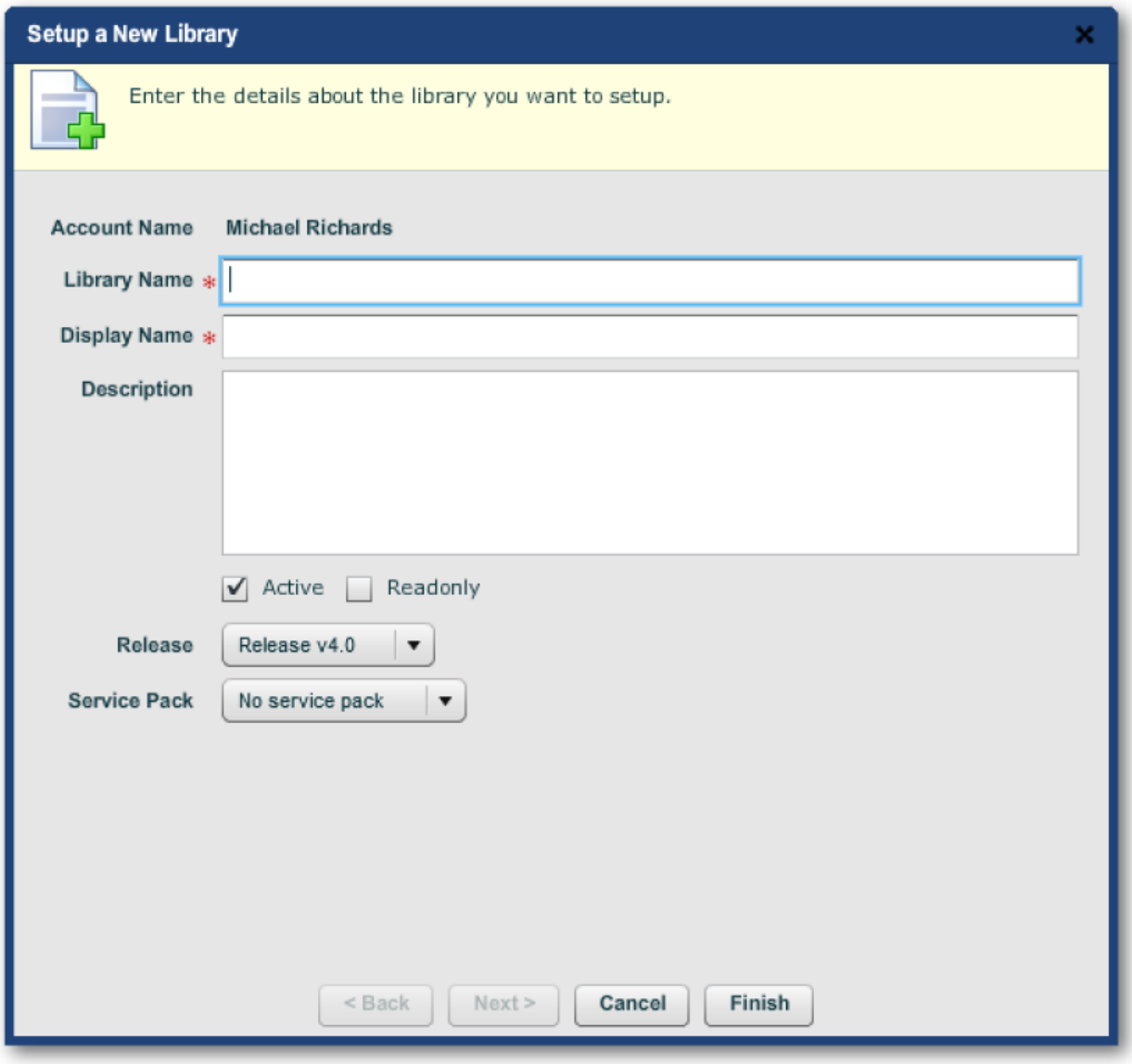


The Library level has a number of powerful features accessed through the Library tabs and the

whole structure of library elements are accessible to users given the necessary rights (see Allocating Users).

Creating a New Library

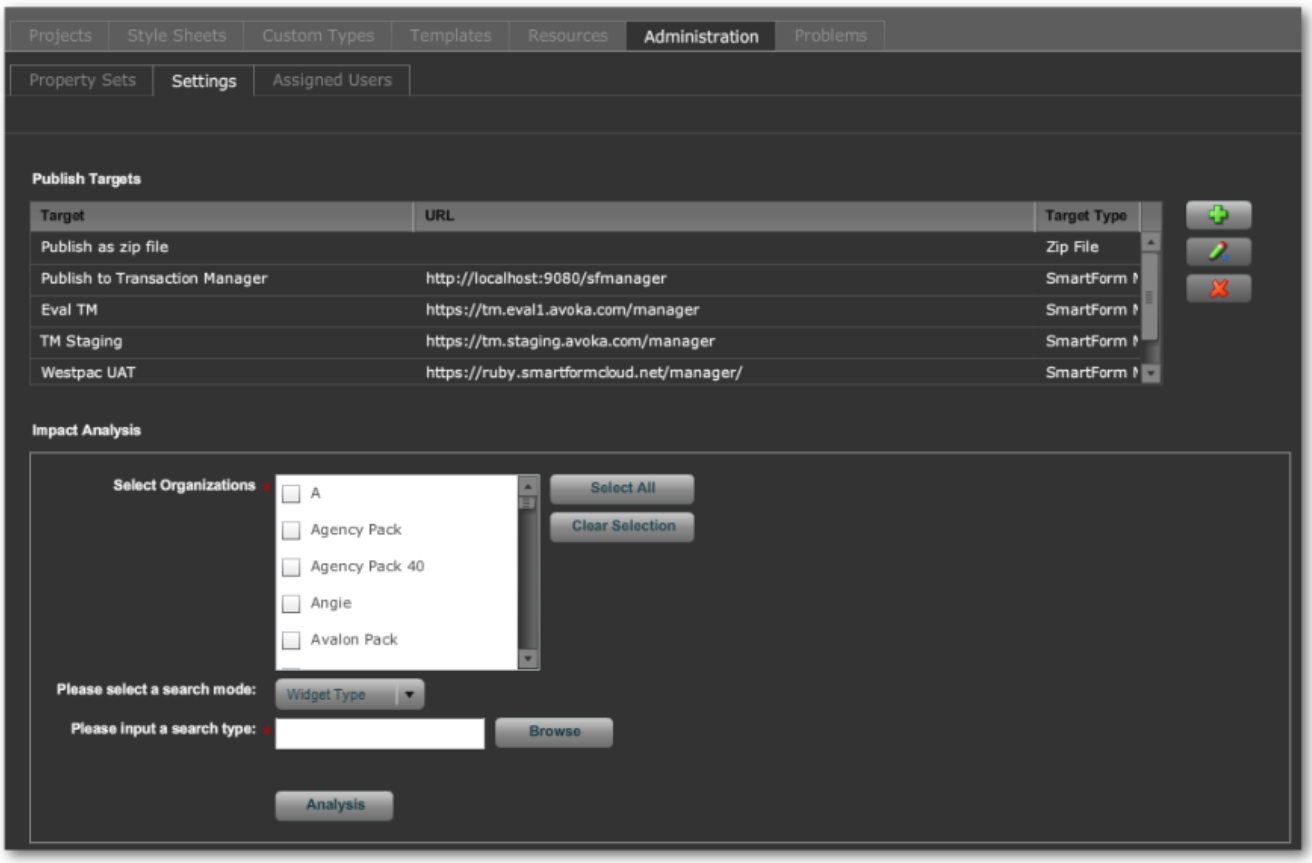
Location: Workspace -> <Account> -> Create New Library link
Required Permissions:ORGANIZATION_ADMIN, LIBRARY_EXPORT, LIBRARY_IMPORT
Roles: System Administrator, Account Administrator, Avoka Engineer, Avoka Partner



Export Options

Composer | Form Builder | v4.3 & Higher

Location: System -> Data Management
Roles: System Administrator
Location: Workspace -> <Organization> -> Administration tab -> Settings tab
Required Permissions:DATAPACK_EXPORT
Roles: Avoka Engineer, Avoka Partner



You can configure how your users export forms to Transaction Manager by using the "Publish Targets" parameters. You can add or delete targets and edit them.

The content, on clicking on the "Export" button and select one of the two options configured above, is either downloaded to your browser as a .zip file or sent to the Transaction Manager instance at the https URL.

The "Impact Analysis" produces a report on which Projects in the selected organizations (and you can select multiple organizations) for the following search modes:

- Widget Types
- Templates
- Style Sheets

Then, you must select to search for particular items in these modes: i.e. for a particular widget or widgets, for a particular template or a particular style sheet

This results in a report dialog that gives the project and forms that use these items.

Inspecting an Organization's Libraries

Location: Workspace -> <Account> -> <Organization>
Required Permissions:ORGANIZATION_ADMIN
Roles: System Administrator, Account Administration, Avoka Engineer, Avoka Partner

At the Organization level of the UI, the Organization's Projects are selectable in the default tab and perform actions on them.

But from the point of view of administrating the Organization, the tabs other than Projects are useful for inspecting the library elements assigned to the Organization, using:

- Style Sheets tab
- Custom Types tab
- Template tab
- Resources tab

Each of these tabs have, top right, a dropdown filter selector to view these by the Current Organization or by All Organizations (in the current environment).
Use the Administration tab to configure and allocate Property Sets.

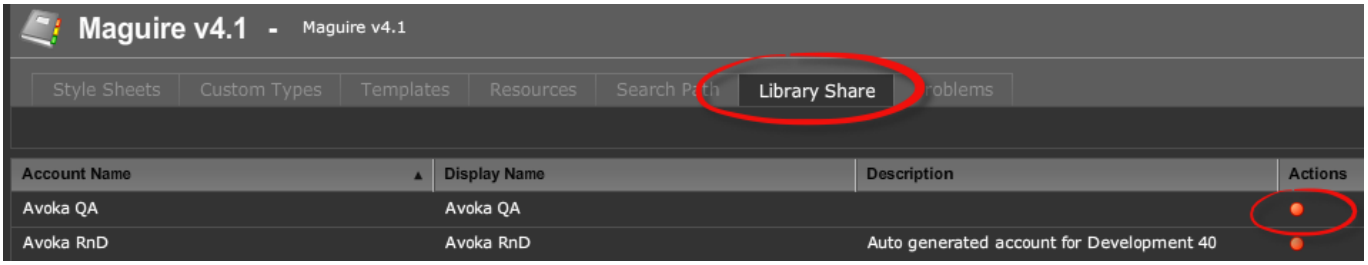
Maintaining Libraries

Composer | Form Builder | v4.3 & Higher

Sharing Libraries with Accounts

Location: Workspace -> <Account> -> <Library> -> Administration tab -> Library Share tab

Required Permissions:LIBRARY_SHARE
Roles: Avoka Engineer, Avoka Partner



The library share tab lists all the Accounts in the Environment. The Action button can be used to associate the library to Account or alternatively block the Library.

Note that the library is not visible in the Account's organizations until the library is shared with the account.

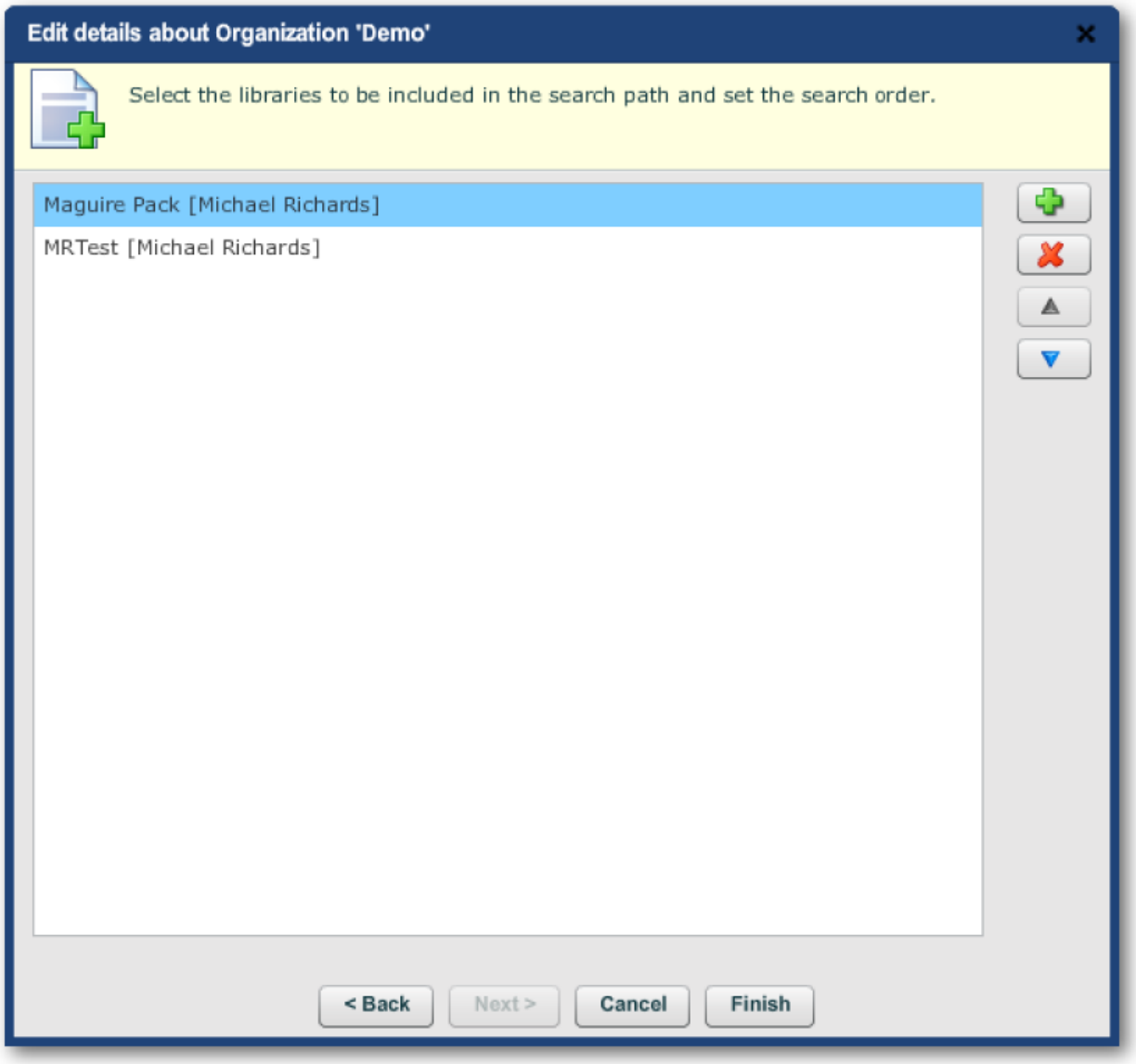
Assigning Libraries to organizations from the Workspace

To assign the library to an organization, either:

- go to Workspace -> <Account> -> <Organization> and click on the Action "Update Organization Details" or
- at the Workspace -> <Account> -> <Library> click on "Update Library Details".

Required Permissions:ORGANIZATION_ADMIN, LIBRARY_SHARE
Roles: Account Administrator, Avoka Engineer, Avoka Partner

In both cases, go through the first stage of the Wizard, and click on "Next", where you can assign Libraries to your Organization or Library. The dialog where you do this looks like this:



Libraries also can come with Service Packs (see Change Release) and are then termed "Data Packs". But despite the different name, they are libraries and can be configured the same ways.

Assigning and Maintaining Stylesheets

Overview

Stylesheets are the building blocks to any template.

This also means that there are a great number of these and they have many categories.

You are even free to add to the list of categories (which you do in the "Save Style Sheet as..." dilaog shown below).

Generally, you will be assigning Stylesheets at Workspace -> <Account> -> <Library>. Freshly-created Organizations do not have their own stylesheets, but can access the libraries assigned to them.

Editing Stylesheets

Location: Worksheet -> <Organization> -> Style Sheets tab

or

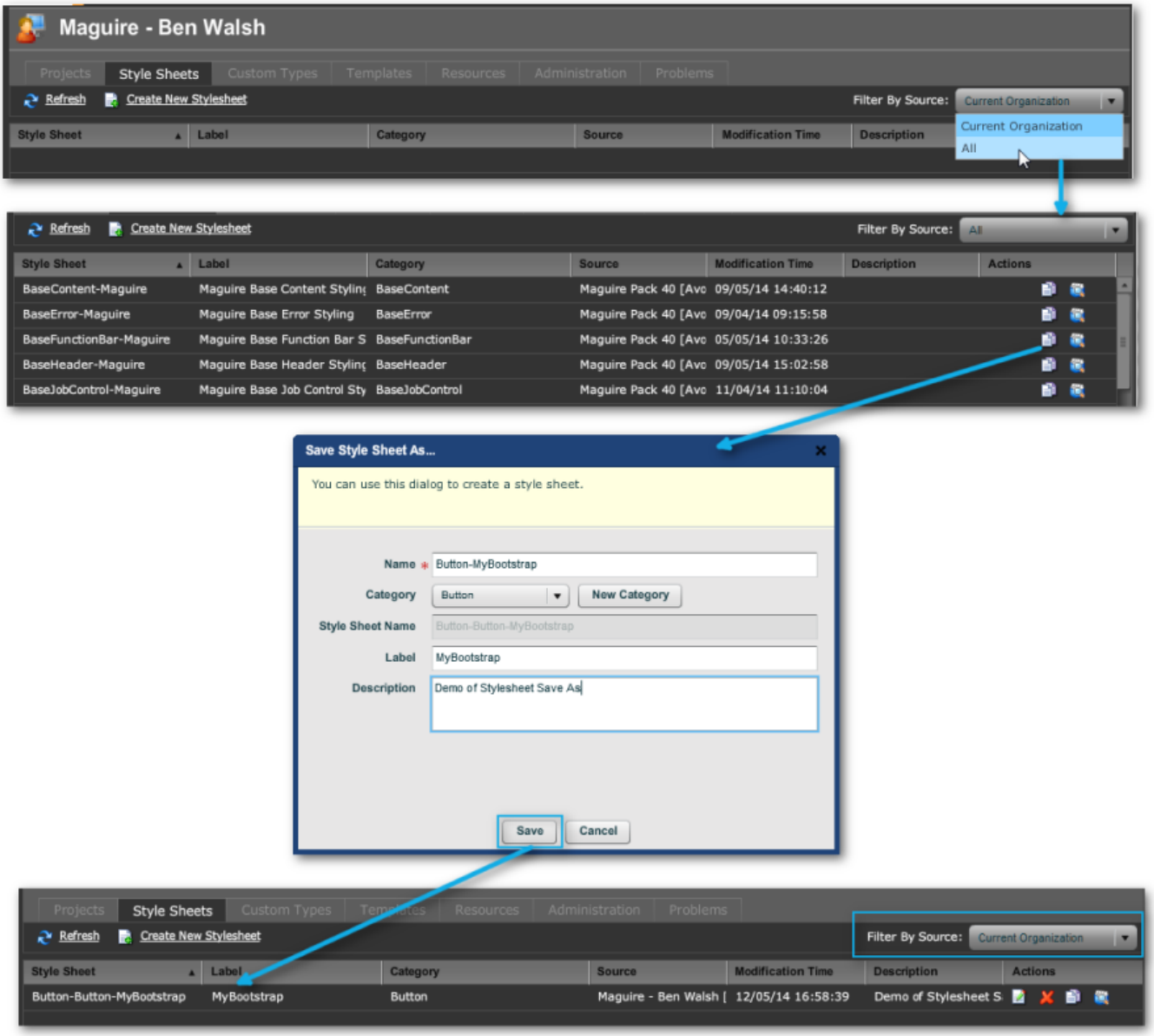
Location: Worksheet -> <Library> Style Sheets tab

Required Permissions: STYLESHEET_MGMT

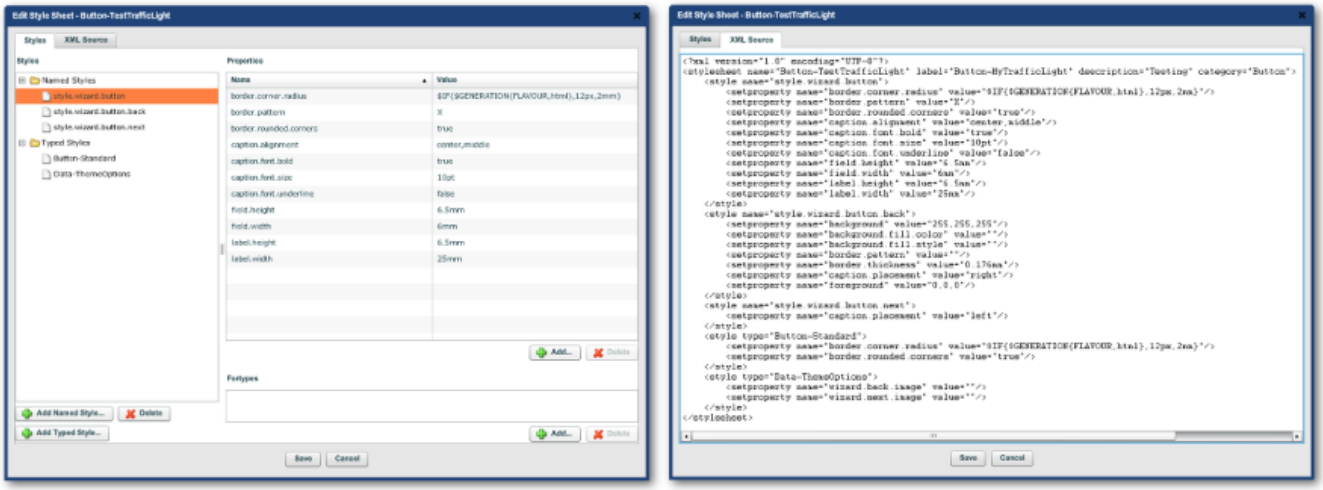
Roles: System Administrator, Account Management, Avoka Engineer, Avoka Partner, Developer

By default, an Organization can access the Account's stylesheets. These are all read-only.

You can perform a "Save as..." on each of these so in that you can edit the new versions and also assign them to nominated libraries.



This is the way to populate both Libraries and Organizations with bespoke stylesheets.
Once you have saved a stylesheet, you can edit its properties (double-click on item in the list or use the edit Action):



All of the values of the Properties of the Named Styles and Typed Styles are clickable and editable. You can also add and delete styles and properties and even edit the XML (though do not attempt to do so unless you know what you are doing). The Edit Style Sheet function is very powerful.

Maintaining Custom Types

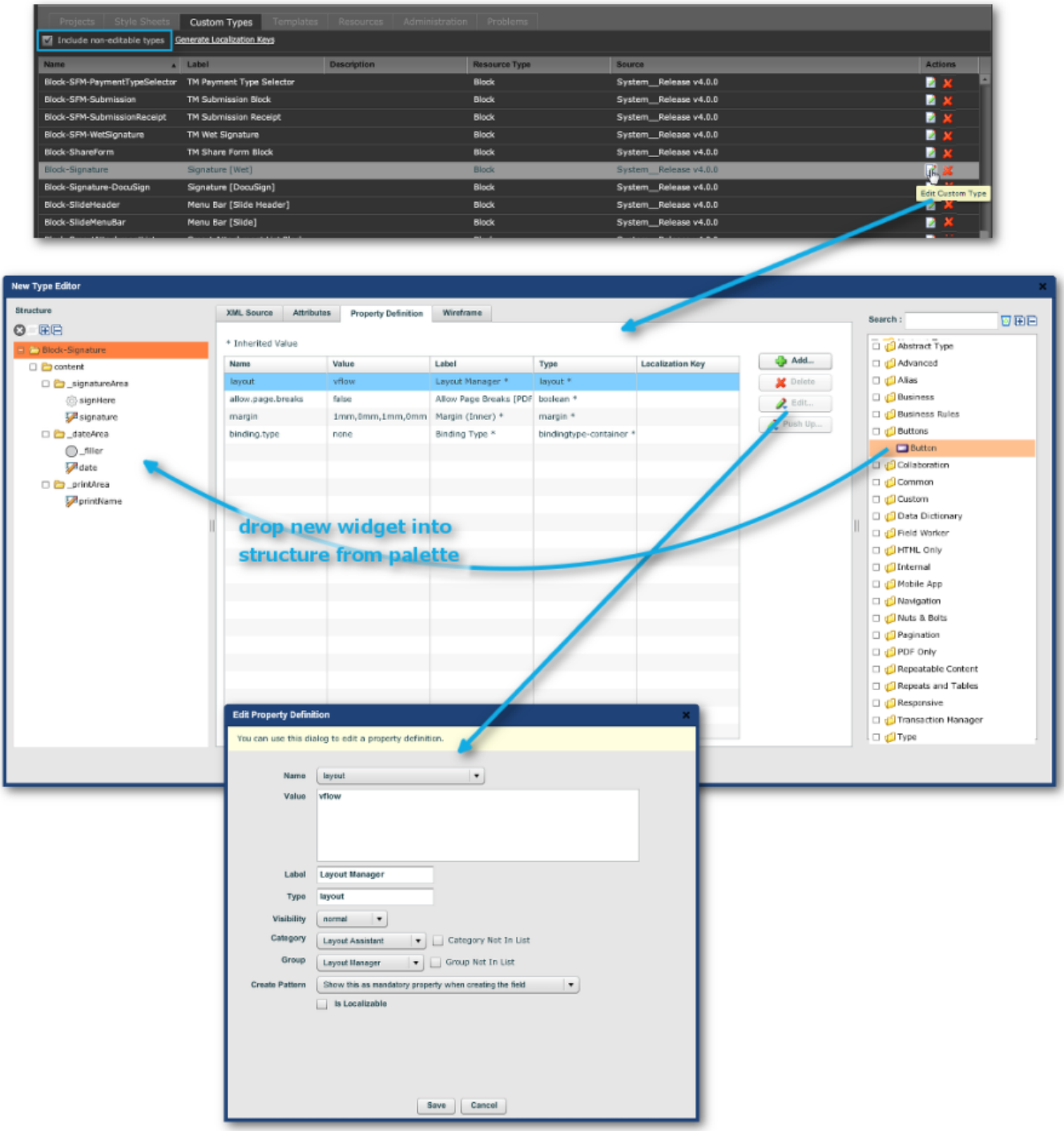
Custom Types

Location: Workspace -> <Organization> -> Custom Types tab
or
Location: Workspace -> <Library> -> Custom Types tab
Required Permissions:MODULE_MGMT
Roles: Account Administrator, Avoka Engineer, Avoka Partner, Developer.

Custom Types, in the context of Library administration, refers to the members of the Widget Palette of the Form Designer (see the Transact Composer 4.0 Guide). There are 2 types of Custom Types:

- **Non-editable** Custom Typesthe standard members of the Widget Palette. You cannot access the type editor for these types.
- **Editable** Custom Typeswhich are your saved types. These will only be saved to the internal library of the organization with an additional type not relevant here:
- **Localization Keys** used for the Internationalization of forms. This is not really an administration task, and so is covered in the Composer Online Documentation.

When you are allowed to edit a custom type, you will be able to access the "Type Editor,



The **Create Pattern** dropdown menu in the Edit Property Definition dialog defines whether:

- The property appears or does not appear in the wizard when the custom Type is dropped onto the structure panel of forms
- and if it does appear in the wizard. whether the property is marked mandatory or not.

Here are the other tabs in the editor:



Note that the XML Source updates on-the-fly

For most Custom Types, you will only be able to view the XML source in a read-only window.

Maintaining Templates

Location: Workspace -> <Account> -> <Organization> | <Library> -> Templates tab

Required Permisions:TEMPLATE_MGMT

Roles: System Administrator, Avoka Engineer, Avoka Partner

You need to create a new template in the Template tab in order to edit it, just as with stylesheets.

The wizard to do so has you base the new template on an already existing one.

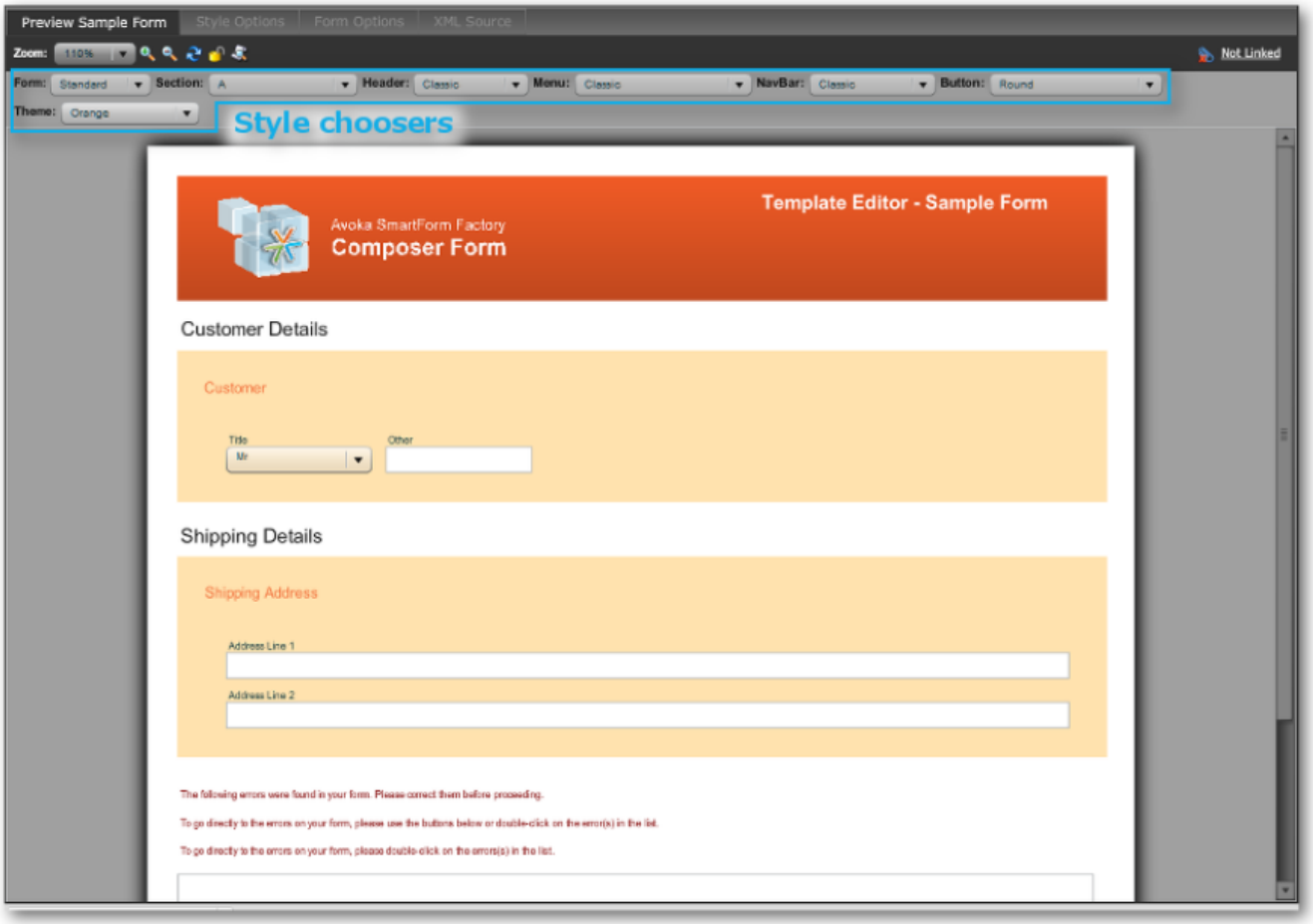
Your choice of base template is restricted to the templates already assigned to the Organization. With the new template saved, you can then invoke the "Edit" action. The template designer has 4 tabs:

1.Preview Sample Form

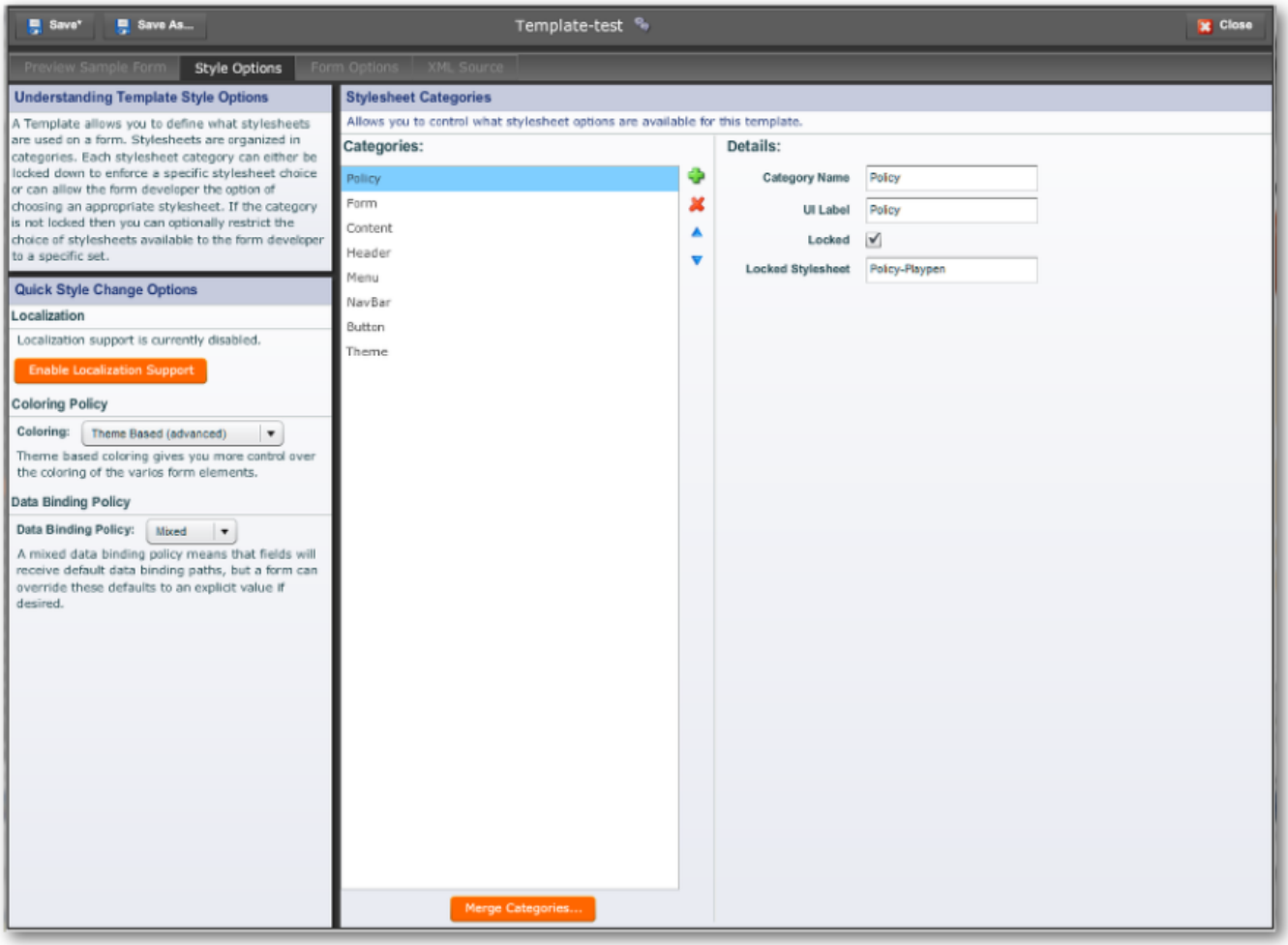
2.Style Options

3.XML Source

Preview Sample Form



Style Options



This part of the system is where you can attach or delete whole stylesheet categories (see Custom Types) to the template.

The Quick Style Change Options panel allows three simple settings:

- Turning Localization on or off (see Custom Types below).
- Setting the Coloring Policy to

o Palette Based (simple)where there is a "Color" dropdown in the Style Choosers marked "Color" and the end user chooses "Blue", "Green", "Gray", and so forth.

o Theme Based (advanced)where there is now a "Theme" dropdown and the changes to the appearance of the template are more extensive

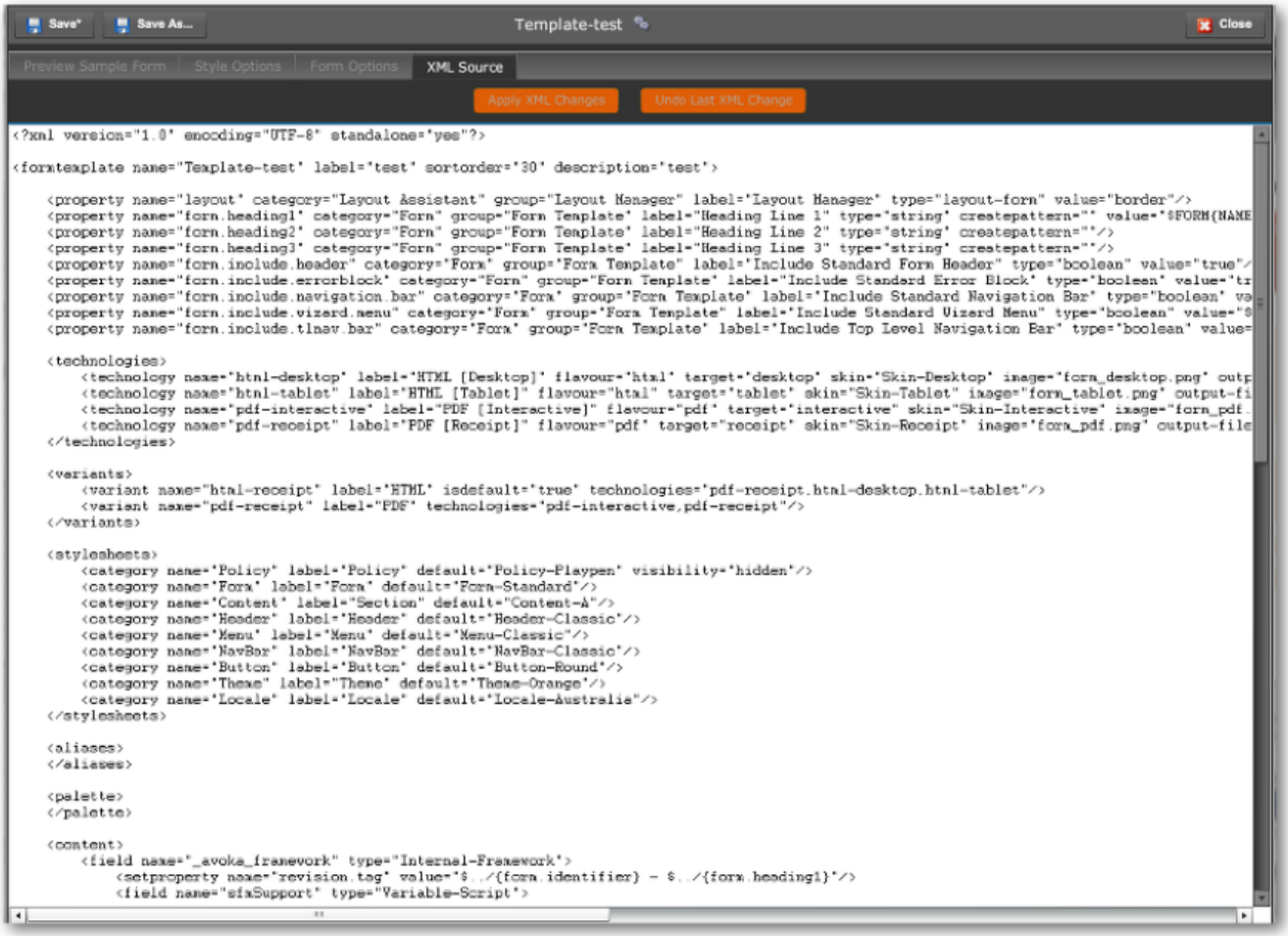
o Custom (do it yourself)where policy is defined in the XML. Only for the experienced.

The Data Binding Policy has 2 settings:

- Mixed where designers have the option of altering the data binding of elements in the Structure Panel of Form Designer.
- Explicit where designers cannot alter data binding for any element.

The Stylesheet Categories panel is currently read-only,

XML Source

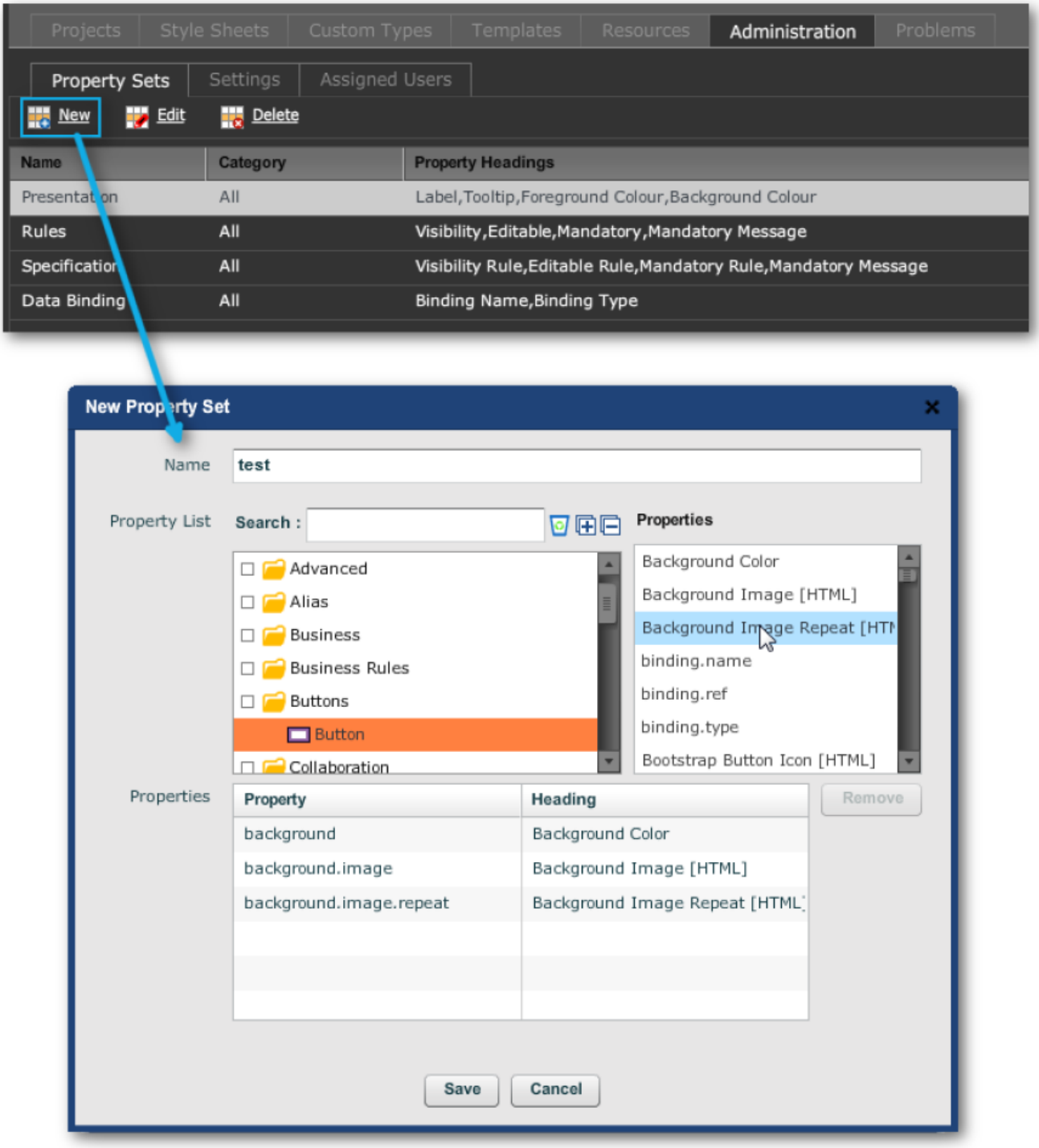


Assigning Libraries

Maintaining Property Sets

Composer | Form Builder | v4.3 & Higher

Location: Workspace -> <Account> -> <Organization> -> Administration tab -> Property Sets tab
Required Permissions:ORGANIZATION_ADMIN
Roles: System Adminisatration, Account Administrator, Avoka Engineer, Avoka Partner



Property sets allow you to group the Properties of various Custom Types together into convenient sets. You can see some of the default sets in the above screen grab: Presentation, Rules, Data Binding and so forth.

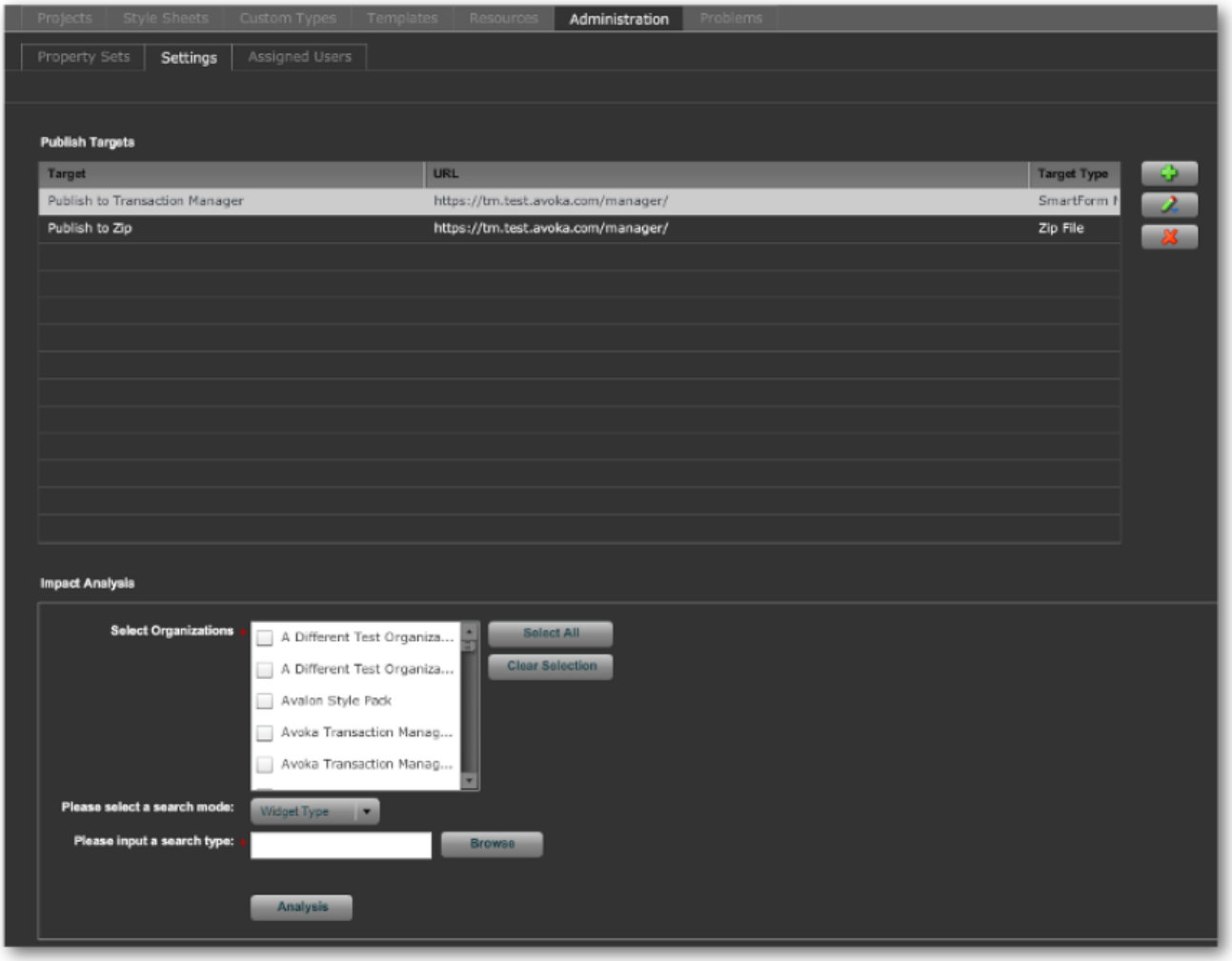
Publishing & Impact Analysis

Composer | Form Builder | v4.3 & Higher

Location: Workspace -> <Account> -> <Organization> -> Administration tab -> Settings tab
Required Permissions:ORGANIZATION_ADMIN
Roles: System Administration, Account Administrator, Avoka Engineer, Avoka Partner

This panel allows 2 functions

1. Set the publish targets
2. Run an impact analysis that will report on specified search criteria in the organization



Publish Targets

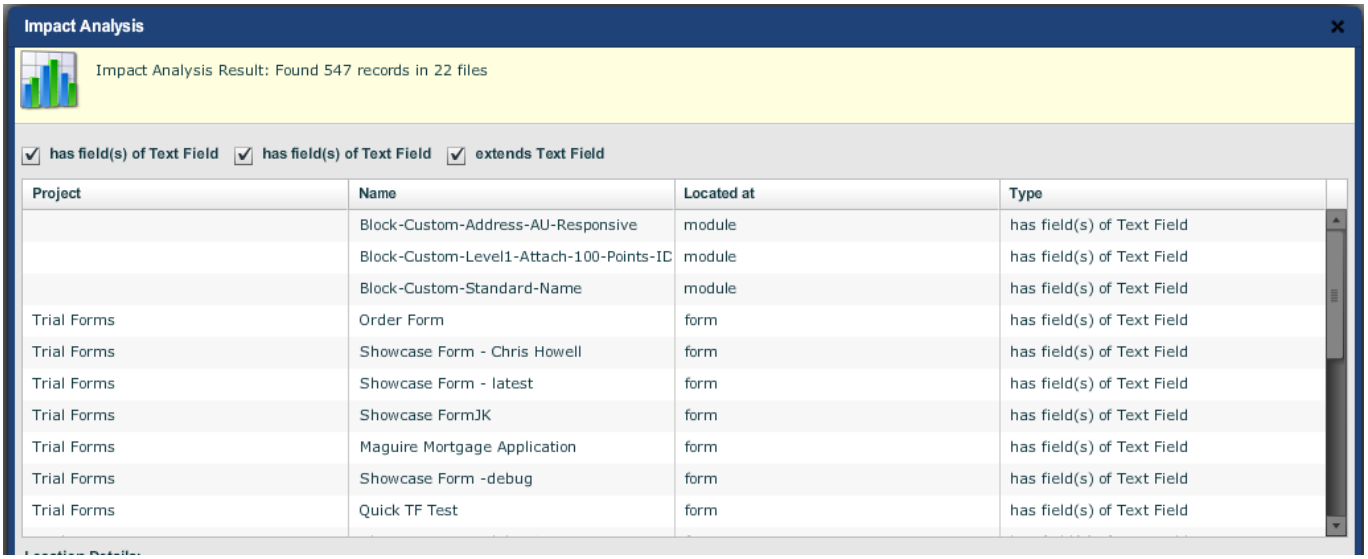
The **Publish Targets** panel allows you to set the default options for the Organization. These options will be presented to end-users whenever they publish forms to Transaction Manager. Here you can define targets - combinations of different instances of TM as defined by their URLs and of target types (i.e. either zip files for uploading to TM or Composer Form packages posted online to the TM nominated instance).

Impact Analysis

The Impact Analysis panel produces a report on the impact of various elements on selected Organizations. You can nominate which search mode: whether on **Widget Type**, **Template** or **Stylesheet**.

The resultant read-only report lists the hits by **Project** and **(Form) Name**.

The example below shows a search on Widget Type "Text Field". The result shows all the organizational assets that contain a "Text field".


Impact Analysis
Impact Analysis Result: Found 547 records in 22 files
has field(s) of Text Field
has field(s) of Text Field
extends Text Field
Project
Name
Located at
Type
Block-Custom-Address-AU-Responsive
module
has field(s) of Text Field
Block-Custom-Level1-Attach-100-Points-IC
module
has field(s) of Text Field
Block-Custom-Standard-Name
module
has field(s) of Text Field
Trial Forms
Order Form
form
has field(s) of Text Field
Trial Forms
Showcase Form - Chris Howell
form
has field(s) of Text Field
Trial Forms
Showcase Form - latest
form
has field(s) of Text Field
Trial Forms
Showcase FormJK
form
has field(s) of Text Field
Trial Forms
Maguire Mortgage Application
form
has field(s) of Text Field
Trial Forms
Showcase Form -debug
form
has field(s) of Text Field
Trial Forms
Quick TF Test
form
has field(s) of Text Field

Creating Users

Composer | Form Builder | v4.3 & Higher

Composer supports two tiers of user: Client and System.

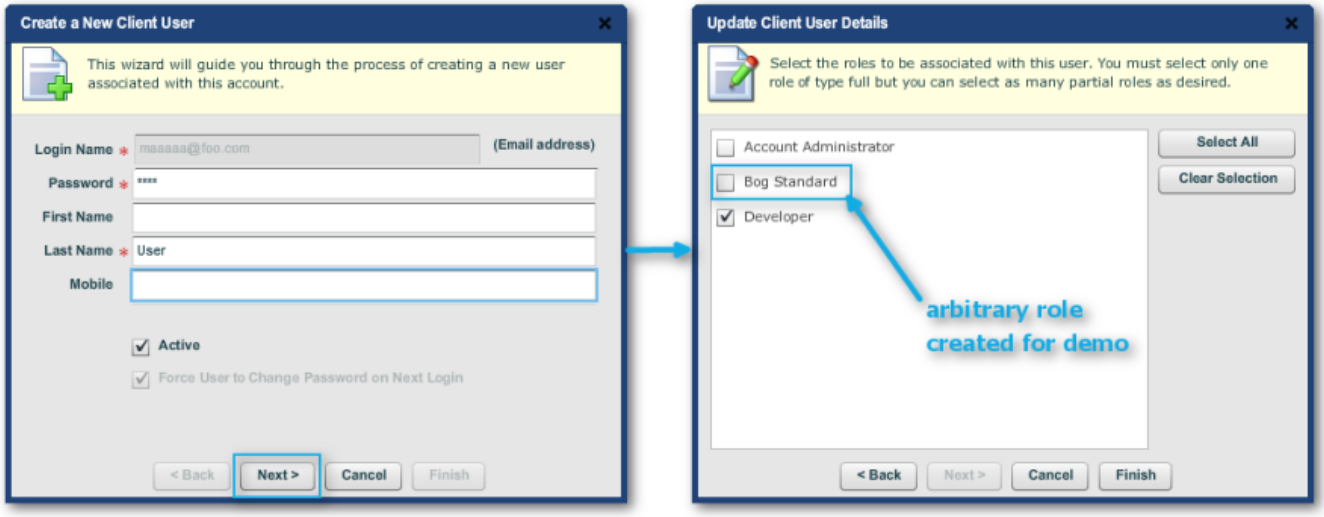
- Client Tier who can only see read-only Accounts designated "Public Accounts" and the one Account to which they have been allocated.
- System Tier who can access one or more non-public accounts and who in general have more powerful roles than the Client Tier.

Client Tier Users

Location: Workspace -> <account> -> Users tab
Required Permissions:USER_ADMIN
Roles: System Administrator, Avoka Engineer, Account Administrator,

Client tier users can be created only at this location.

Use the "Users (tab) -> Create New User" link (see here) to access the "Create a New Client User" wizard:



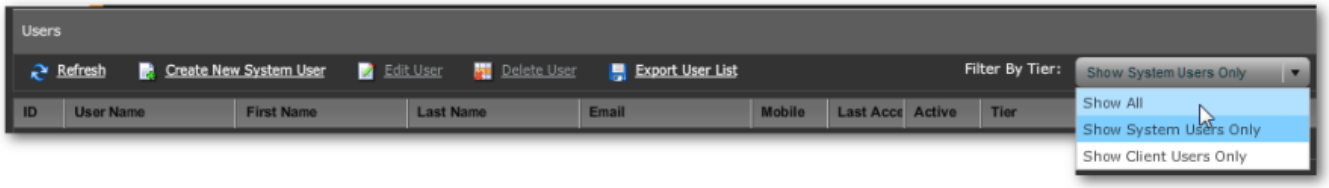
When you progress to the next part of the wizard, choose the role or roles for the user you are creating. Please refer to Creating Roles on how to define roles.

Here, we have created an arbitrary role for the sake of the demo.

The non-System user you create here at the Account level also appears in System -> Access Control and can then be assigned to this, or some other account, through System -> Account Management.

System Tier Users

Location: System -> Access Control -> Users
Required Permissions:SYSTEM_USER_ADMIN
Roles: System Administrator

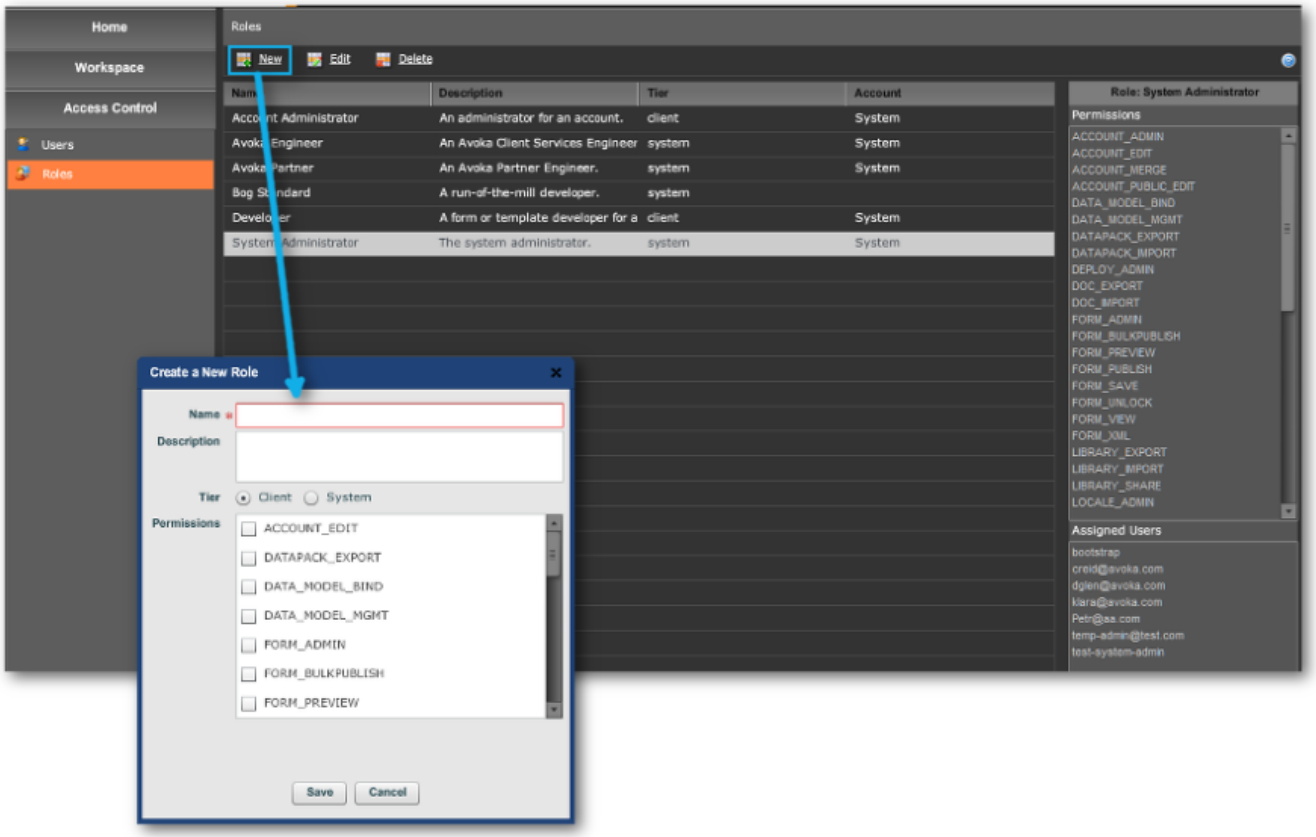


The "Create New System User" link invokes the same wizard as in Client Tier Users, and you also get to assign a role to the user.

Creating Roles

Composer | Form Builder | v4.3 & Higher

Location: Access Control -> Roles
Required Permissions:SYSTEM_ROLE_ADMIN
Roles: System Administrator



There are two tiers of user: Client and System.

- Client Tier who can only see those read-only Accounts designated "Public Accounts" and the one Account to which they have been allocated.
- System Tier who may also see other non-public accounts and who in general have more powerful roles than the Client Tier.

You also decide the permissions to allocate to each role. You actually have quite a lot of control as there are many permission items to choose from. Here are the default System roles that ship with the product:

Permission	Description
ACCOUNT_ADMIN	Allows access to the account and organization management panes, creation/deletion/editing of account info.

ACCOUNT_EDIT	Allows editing of account meta data (display name,etc).
ACCOUNT_ MERGE	Allows merging of accounts.
ACCOUNT_ PUBLIC_EDIT	Allows writing to public organizations and accounts.
DATA_ DICTIONARY_ BIND	Allows binding of form fields to the data dictionary.
DATA_ DICTIONARY_ MGMT	Allows creation and maintenance of the organization level data dictionary.
DATA_ DICTIONARY_ STATUS	Allows status changes of data dictionary elements.
DATA_MODEL_ BIND	Allows binding of form fields to the data model.
DATA_MODEL_ MGMT	Allows creation/deletion and editing of data connections, uploading and deleting of seed files.
DATAPACK_ EXPORT	Allows exporting system datapacks.
DATAPACK_ IMPORT	Allows importing system datapacks.
DEPLOY_ADMIN	Allows creation/deletion/editing of deployment properties.
DOC_EXPORT	Allows exporting documentation datapacks.
DOC_IMPORT	Allows importing documentation datapacks.
FORM_ADMIN	Allows viewing form statistics, generating form spec information.
FORM_ BULKPUBLISH	Allows bulk publishing of forms.
FORM_PREVIEW	Allows previewing of forms, access to artifacts pane.
FORM_PUBLISH	Allows publishing of forms.

FORM_SAVE	Allows editing and saving of forms, creation and management of form revisions.
FORM_UNLOCK	Allows unlocking of forms locked by other users, allows opening (but not saving) of forms locked by other users.
FORM_VIEW	Allows viewing and opening of forms (but not saving/preview).
FORM_XML	Allows viewing and saving of form xml.
LIBRARY_EXPORT	Allows exporting of account libraries.
LIBRARY_IMPORT	Allows importing client libraries.
LIBRARY_SHARE	Allows sharing of a library (or patch) with another account.
LOCALE_ADMIN	Allows generation and uploading of localized property files, generation of localization keys
LOCALE_ AUTOTRANSLATE	Allows use of auto translation tools.
MODULE_MGMT	Allows creation/deletion and editing of custom widgets/blocks, allows creating custom widgets and blocks from within a form.
NEWS_ADMIN	Allows creation/deletion/editing of welcome screen news.
ORGANIZATION_ ADMIN	Allows editing organization meta data and search paths, upgrading to a release/servicepack, deleting, renaming, creating, moving and copying organizations, managing organization property sets.
ORGANIZATION_ EXPORT	Allows exporting user organizations.
ORGANIZATION_ IMPORT	Allows importing user organizations.
PATCH_EXPORT	Allows exporting patches.
PATCH_IMPORT	Allows importing patches.
PROJECT_ADMIN	Allows creation and deletion of projects, restricting access of users and templates to projects.

RELEASE_ADMIN	Allows access to the release management pane, editing release meta data, toggling release active state, bulk organisation release upgrades.
RELEASE_ EXPORT	Allows exporting releases.
RELEASE_ IMPORT	Allows importing releases.
RESOURCE_ MGMT	Allows uploading/deleting resources.
ROLE_ADMIN	Allows creation/deletion of roles for an account.
SERVICEPACK_ EXPORT	Allows exporting servicepacks.
SERVICEPACK_ IMPORT	Allows importing servicepacks.
STYLESHEET_ MGMT	Allows creation/deletion and editing of stylesheets, allows push to stylesheet from within a form.
SYSTEM_FORM_ DEBUG	Allows use of debug versions of system script libraries.
SYSTEM_FORM_ PREVIEW	Allows system wide previewing of forms.
SYSTEM_FORM_ PUBLISH	Allows system wide publishing and bulk publishing of forms.
SYSTEM_FORM_ VIEW	Allows system wide viewing of forms.
SYSTEM_LOG_ VIEW	Allows access to system level log files.
SYSTEM_ROLE_ ADMIN	Allows creation/deletion of system level roles.
SYSTEM_USER_ ADMIN	Allows creation/deletion of system level users and creation of user list.
TEMPLATE_ MGMT	Allows creation/deletion of templates.

TIPS_ADMIN	Allows creation/deletion/editing of quick help tips.
USER_ADMIN	Allows creation/deletion of users for an account.

And here is the list of permissions for these predefined roles.:

Role	Permissions

System Administrator	ACCOUNT_ADMIN ACCOUNT_EDIT ACCOUNT_MERGE ACCOUNT_PUBLIC_EDIT DATA_MODEL_BIND DATA_MODEL_MGMT DATAPACK_EXPORT DATAPACK_IMPORT DEPLOY_ADMIN DOC_EXPORT DOC_IMPORT FORM_ADMIN FORM_BULKPUBLISH FORM_PREVIEW FORM_PUBLISH FORM_SAVE FORM_UNLOCK FORM_VIEW FORM_XML LIBRARY_EXPORT LIBRARY_IMPORT LIBRARY_SHARE LOCALE_ADMIN LOCALE_AUTOTRANSLATE MODULE_MGMT NEWS_ADMIN ORGANIZATION_ADMIN ORGANIZATION_EXPORT ORGANIZATION_IMPORT PATCH_EXPORT PATCH_IMPORT PROJECT_ADMIN RELEASE_ADMIN RELEASE_EXPORT RELEASE_IMPORT RESOURCE_MGMT SERVICEPACK_EXPORT SERVICEPACK_IMPORT STYLESHEET_MGMT SYSTEM_FORM_DEBUG SYSTEM_FORM_PREVIEW SYSTEM_FORM_PUBLISH SYSTEM_FORM_VIEW SYSTEM_LOG_VIEW SYSTEM_ROLE_ADMIN

TIPS_ADMIN
USER_ADMIN

Account Administrator	DATA_MODEL_BIND DATA_MODEL_MGMT FORM_ADMIN FORM_BULKPUBLISH FORM_PREVIEW FORM_PUBLISH FORM_SAVE FORM_UNLOCK FORM_VIEW FORM_XML LIBRARY_EXPORT LIBRARY_IMPORT LOCALE_ADMIN LOCALE_AUTOTRANSLATE MODULE_MGMT ORGANIZATION_ADMIN ORGANIZATION_EXPORT ORGANIZATION_IMPORT PROJECT_ADMIN RESOURCE_MGMT STYLESHEET_MGMT USER_ADMIN

Avoka Engineer	DATA_MODEL_BIND DATA_MODEL_MGMT DATAPACK_EXPORT FORM_ADMIN FORM_BULKPUBLISH FORM_PREVIEW FORM_PUBLISH FORM_SAVE FORM_UNLOCK FORM_VIEW FORM_XML LIBRARY_EXPORT LIBRARY_IMPORT LIBRARY_SHARE LOCALE_ADMIN LOCALE_AUTOTRANSLATE MODULE_MGMT ORGANIZATION_ADMIN ORGANIZATION_EXPORT ORGANIZATION_IMPORT PATCH_EXPORT PROJECT_ADMIN RELEASE_EXPORT RESOURCE_MGMT SERVICEPACK_EXPORT STYLESHEET_MGMT SYSTEM_FORM_DEBUG TEMPLATE_MGMT USER_ADMIN

Avoka Partner	DATA_MODEL_BIND DATA_MODEL_MGMT DATAPACK_EXPORT FORM_ADMIN FORM_BULKPUBLISH FORM_PREVIEW FORM_PUBLISH FORM_SAVE FORM_UNLOCK FORM_VIEW FORM_XML LIBRARY_EXPORT LIBRARY_IMPORT LIBRARY_SHARE LOCALE_ADMIN LOCALE_AUTOTRANSLATE MODULE_MGMT ORGANIZATION_ADMIN ORGANIZATION_EXPORT ORGANIZATION_IMPORT PATCH_EXPORT
Developer	DATA_MODEL_BIND DATA_MODEL_MGMT FORM_PREVIEW FORM_PUBLISH FORM_SAVE FORM_VIEW FORM_XML LOCALE_ADMIN LOCALE_AUTOTRANSLATE MODULE_MGMT RESOURCE_MGMT STYLESHEET_MGMT

Allocating Users

Composer | Form Builder | v4.3 & Higher

At the Account Level

All client tier users are created at the Account level and are therefore assigned to that Account.

All system tier uses have access to all Accounts.

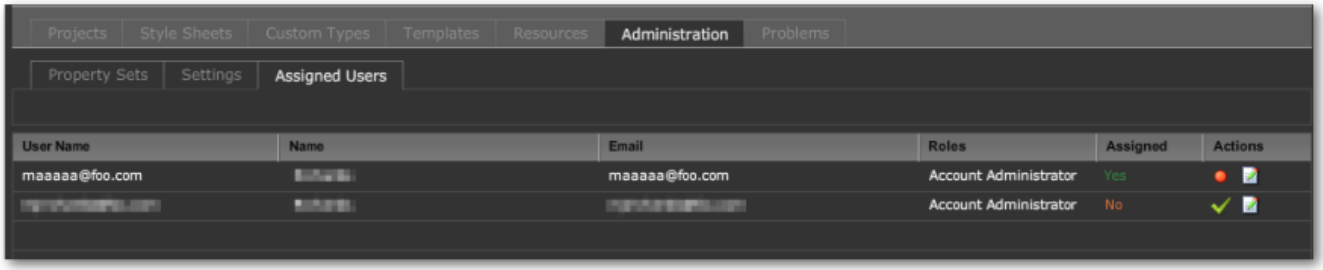
At the Organization Level

Location: Workspace -> <Account> -> <Organization> -> Assigned Users

Required Permissions:ORGANIZATION_ADMIN

Roles: System AdministratorAccount Administrator, Avoka Engineer, Avoka Partner

This location lists all the Account-level client tier users.



The left Action buttons can be either red (to block that user from any new Projects in the Organization) or a green check mark (to allow the user instead).

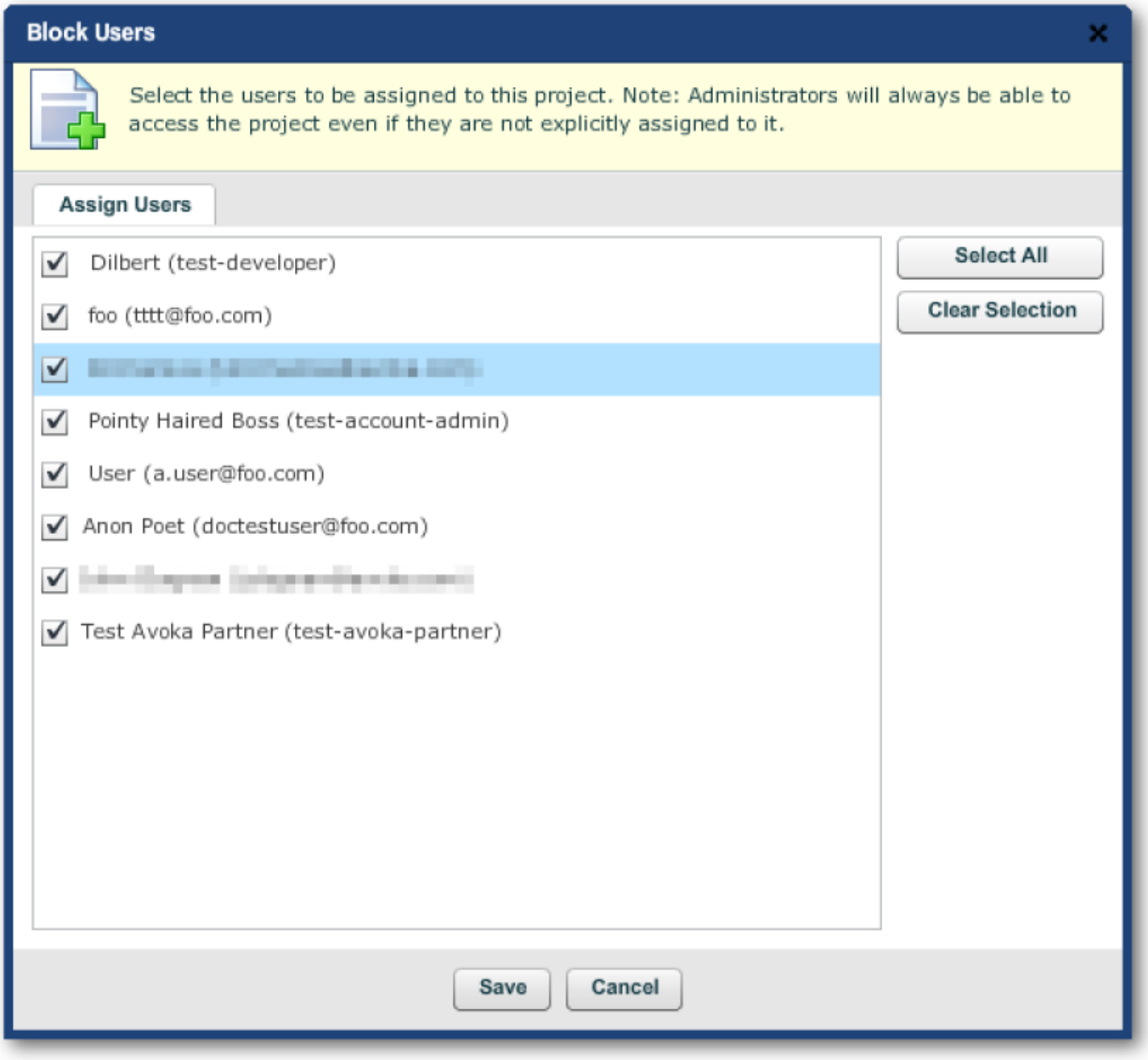
So, at this level, you do not assign users — you either block or unblock them.

At the Project Level

Location: Workspace -> <Account> -> <Organization> -> Project -> Assigned Users tab -> Assign Users link

Required Permissions: PROJECT_ADMIN

Roles: System Administrator, Account Administrator, Avoka Engineer, Avoka Partner

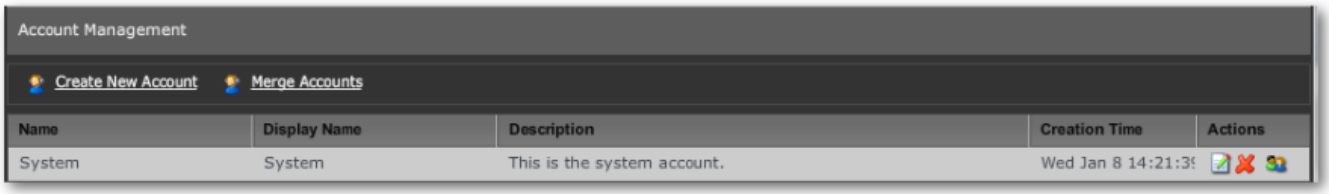


Or, you can use the Workspace -> <Account> -> <Organization> -> Project -> Assigned Users tab, which looks the same as that at the Organization level above.

Creating New Accounts

Composer | Form Builder | v4.3 & Higher

Location: System -> Account Management
Required Permissions:ACCOUNT_PUBLIC_EDIT, ACCOUNT_ADMIN, ACCOUNT_EDIT, ACCOUNT_MERGE
Roles: System Administrator



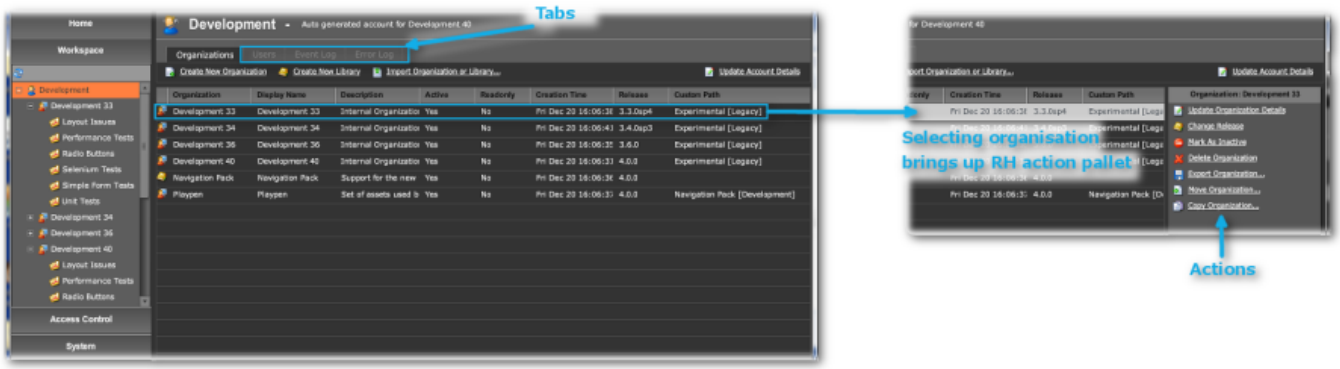
This is the only place in Composer where you can create an Account. You supply the Name, the Display Name and Description for the account.

Use the Actions to edit these properties, to delete the account and assign **System Users** to the account.

Again, this is the only place in Composer where these can be done.

Once an Account has been created, its Account level of the Workspace looks like this:

Location: Workspace -> <Account>



Deployment Properties Setting

Composer | Form Builder | v4.3 & Higher

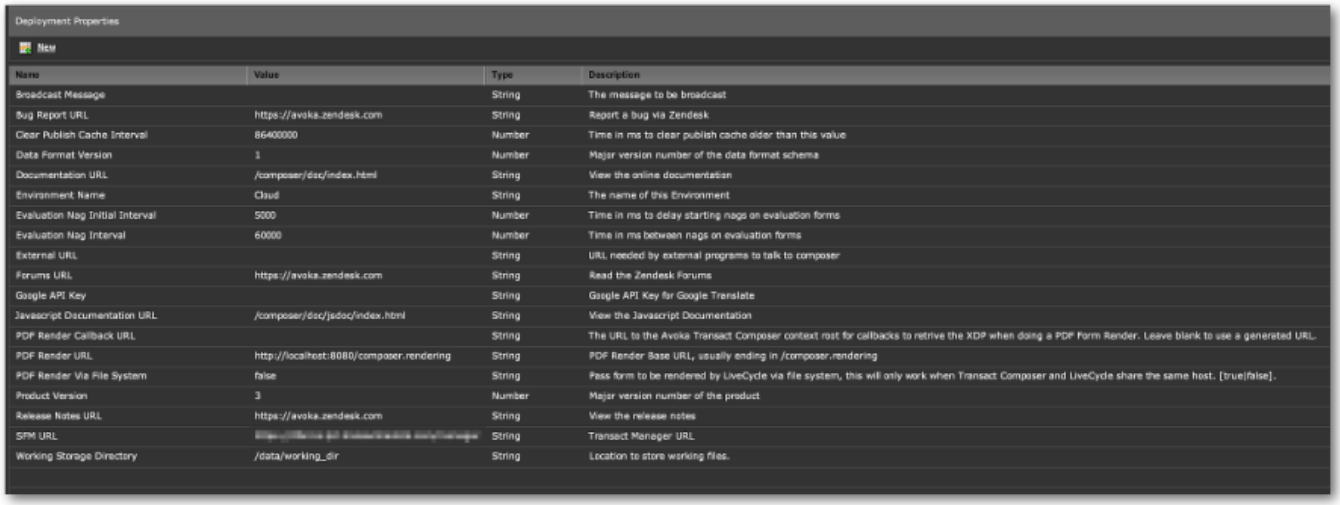
Composer comes with a set of deployments properties that you can use as they are or edit them as required.

Location: **System -> Deployment Properties**

Required Permissions: DEPLOY_ADMIN

Roles:System Administrator

The System -> Deployment page lists the parameters of the Composer deployment. The parameters are editable. Some of them are shown below:



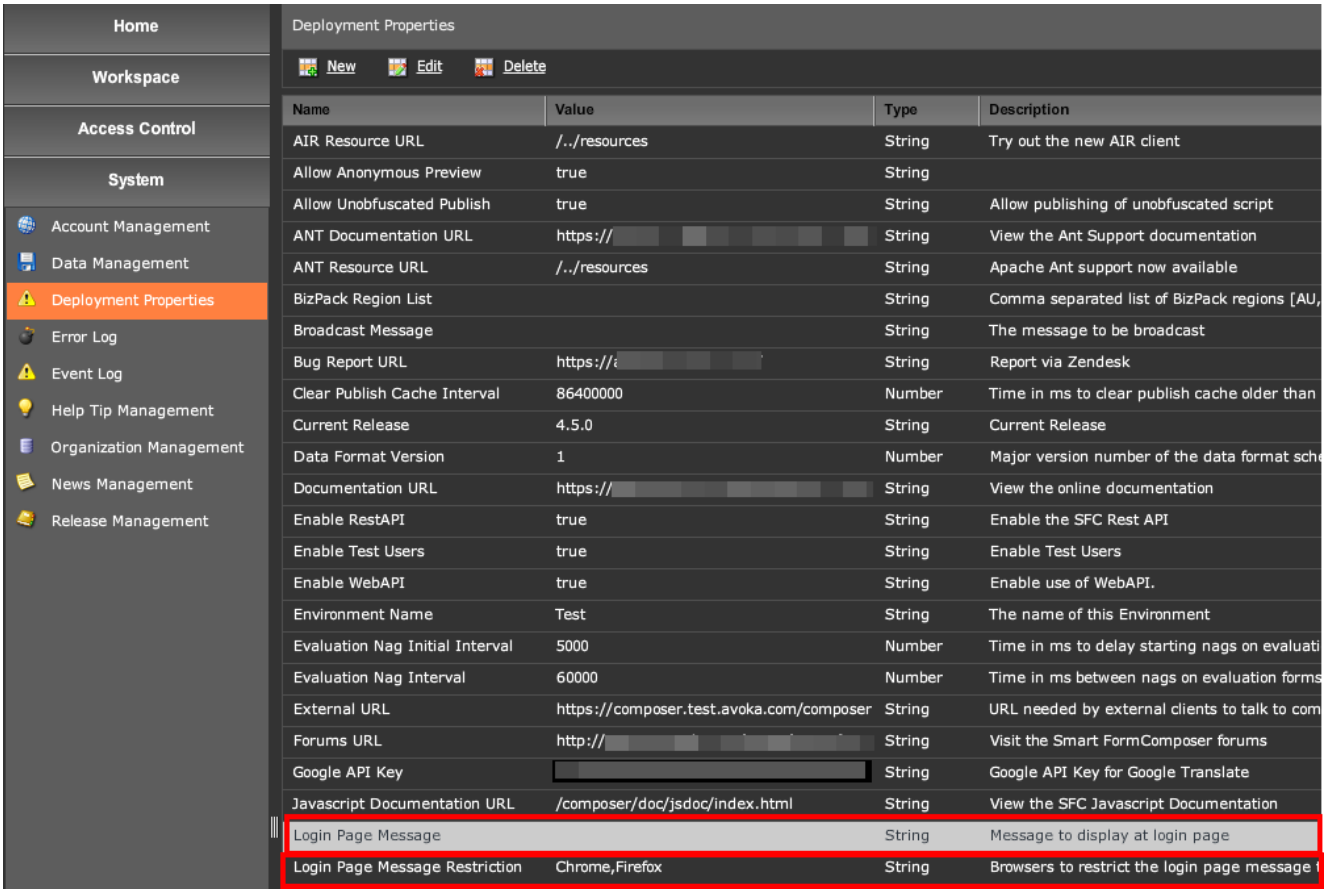
Here are a few examples how to make use of these properties.

Display Important Information on the Composer Login Page

When you need to display some important information to Composer users, the best places to do so is on the user log in screen.

To configure a login screen message:

1. Edit the **Login Page Message** deployment property with your message text and formatting.
2. Update the **Login Page Message Restriction** with a list of comma separated browsers this message will be displayed on. Leave blank for all browsers or use the following list removing the browsers that are not wanted. Edge, IE, Chrome, Safari, Firefox



3. Check that the message appears on the login page as expected,

You have to have the above role and permission assigned to you to be able to see and edit this property.

Maintaining Help, News, and Tips

Composer | Form Builder | v4.3 & Higher

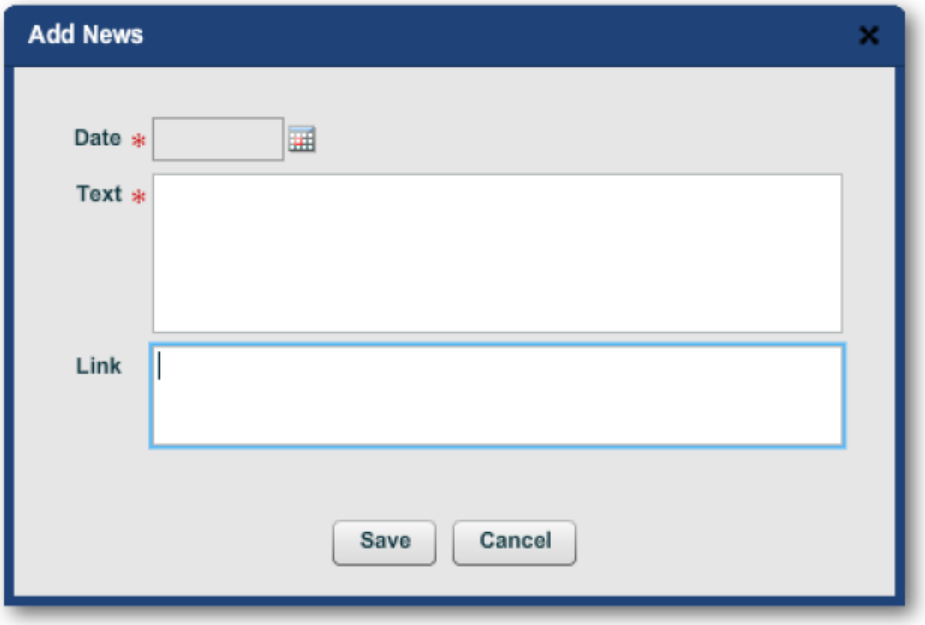
Location: System -> News Management

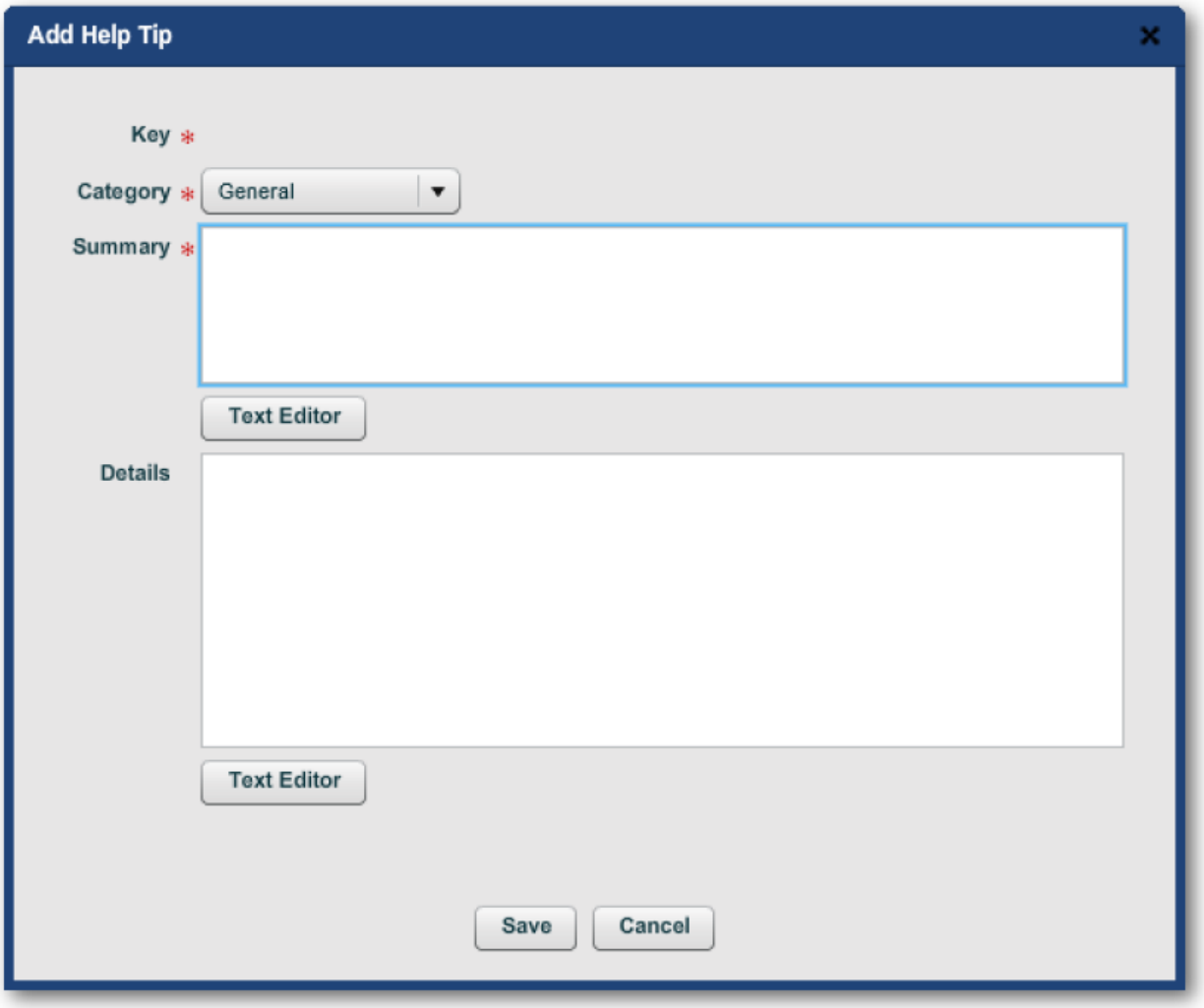
OR

Location: System -> Help Tip Management

Required Permissions: NEWS_ADMIN, TIPS_ADMIN

Roles: System Administrator





Categories:

- General
- Data Dictionary

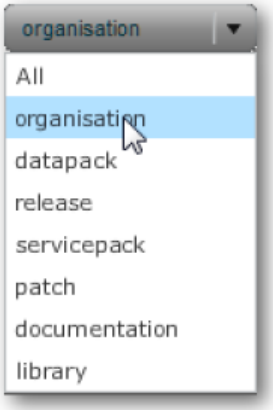
Text editors.
Import data: in XML file
Export as XML file downloaded to browser.

Organization Management

Composer | Form Builder | v4.3 & Higher

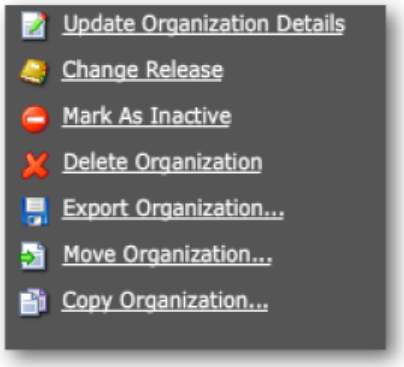
Location: System -> Organization Management
Roles: System Administration

Use this to administer Organizations, various Libraries, and everything else at the same level of the Workspace (see The Workspace). You can filter the list on the following types:



- **Organizations**
- **Data Packs** in other words, special customization packs
- **Releases**
- **Service Packs**
- **Patches**
- **Documentation** in other words, the online HTML help for an Account in the form of a flat .zip file
- **Libraries**

The Actions available on each item are:



Organizations can be exported as a .zip file, which is useful for importing Organizations to different environments.

The Copy function works within the one environment.

There is also a link (top left of the Organization Management page) to Import a previously-exported Organization.

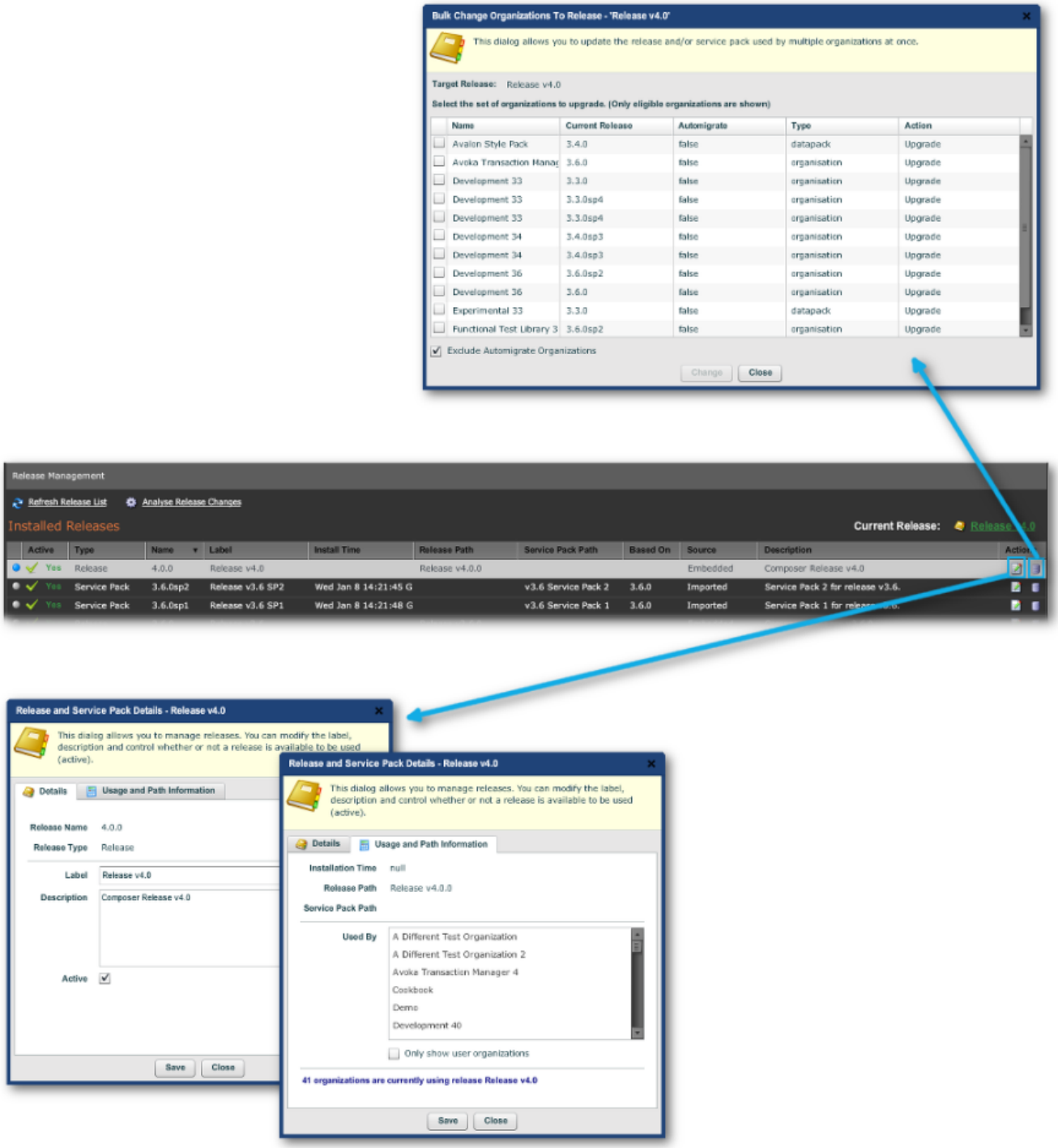
Release Management

Composer | Form Builder | v4.3 & Higher

Location: System -> Release Management
Required Permissions: RELEASE_ADMIN
Roles: System Administrator

When you create Organizations, you set their release compatibility with previous releases and whether the Organization migrates to new releases automatically or manually.

You can also alter the release settings for several Organizations in bulk.



Viewing the Logs

Composer | Form Builder | v4.3 & Higher

Location: System -> Error Log
and
Location: System -> Event Log
Required Permissions:[SYSTEM_LOG_VIEW| Creating Users#topic_Creating_Roles2SYSTEM_LOG_VIEW]
Roles: Avoka Administrator

There are 2 log sets:

- Error
- Event

The logs are listed in a table with these headings:

- ID
- Time
- Account
- Organization
- User Name
- Message

Double-clicking on each log entry brings up a dialog with these values and some others, namely for Errors:

- Context
- Solution
- Stack Trace

and for Events:

- Event Type

Both log listings can also be filtered on:

- Start and End times
- Account
- Organization
- User
- Message

There is no option in Composer for clearing these logs.

Managed Datapacks

Composer | Form Builder | v4.3 & Higher

The Managed Datapack framework has been created to tie the version of shared datapacks to a composer release automatically. Datapacks such as Maguire reside in a shared datapack account and are not allocated directly to a client. Their allocation is controlled by a Managed datapack that contains the rules for which Maguire version should be used for each Composer release. According to the rules, the correct Maguire Pack version is added to the Organization's path automatically.

By centralizing the datapacks, deployment of new versions is simple because there is only the shared Datapack Account to update, rather than all the accounts on the environment.

The Managed Datapacks are subscribed at the account level. This allows organizations to add the managed datapack to their path if they wish to tie in the Composer version release with the compatible datapack release automatically.

Background and strategic fit

Many organizations have adopted the Maguire styling and BizPacks resulting in the need to incorporate them into the composer framework to ensure backward compatibility is addressed and that testing ties in with Composer releases.

Features of using Managed Datapacks

#	Title	Description
1	Ability to tie datapacks to Composer releases	We require a way of linking composer releases with the correct version of Maguire or any other datapack like Bizpacks
2	Auto upgrade of datapacks	Organizations using Managed Datapacks need to be automatically upgraded to the compatible datapack release when the composer version is upgraded.
3	Allow for migration code to be run when updating datapacks.	When upgrading to a new version of a datapack such as Maguire, forms in the old version may require changes to make them compatible. This needs to be run automatically on upgrade.

User interaction and design

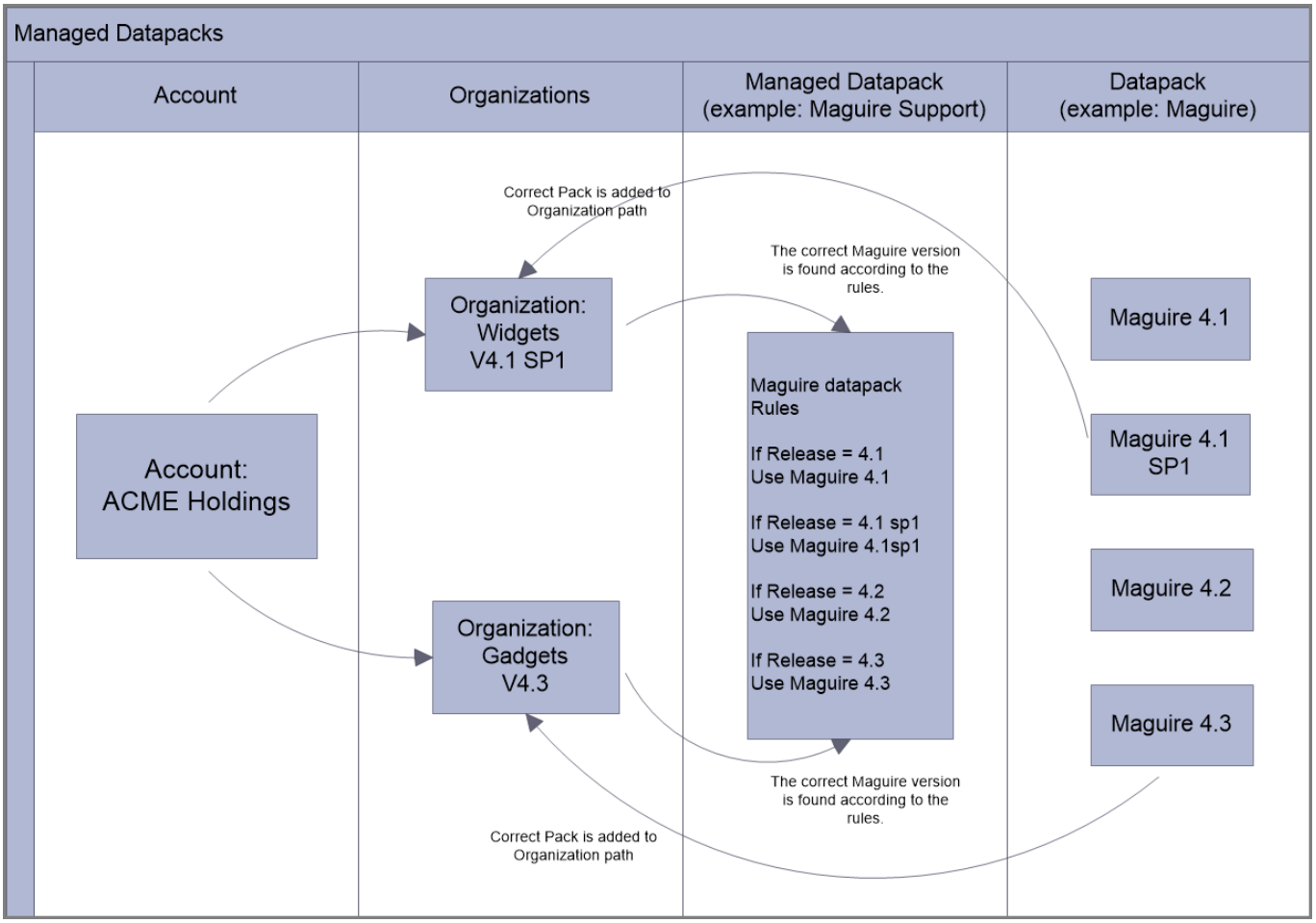
The following diagram illustrates where the managed datapacks reside and how they relate to accounts.

In this framework, the Maguire datapacks are not directly assigned to an organization. They still reside in a shared datapack account (Avoka Datapacks) but their allocation is controlled by the the Managed datapack that contains the rules for which Maguire version should be used for each Composer release. According to the rules, the correct Maguire Pack version is added to the organizational path automatically.

Versioning of Maguire Packs will be aligned with Composer releases.

The Managed Datapacks are subscribed at the account level. This allows organizations to add the managed datapack to their path if they wish to tie in the Composer version release with the compatible Maguire release automatically.

Note that it will still be possible to bypass this process and allocate the Maguire Packs directly **however this will need to be managed manually by the client and is NOT recommended in most cases.**



- Managed Datapacks.
  - These have no content just an organization.xml file
  - The descriptor in this file will contain release/service pack based rules for determining the actual datapack(s) to add the path.
  - Accounts will subscribe to the managed packs making them avaialble to organizations
  - Organizations can then add them to their path via the edit organization function.
- New Maguire releases will force a Composer service pack (or point) release even if there are no composer changes.

Managed Support for BizPacks

Composer | Form Builder | v4.3 & Higher

Bizpacks use the same Managed Datapack framework as Maguire. However there are some differences.

There are 4 Bizpack libraries that are tied to regions. Universal, AU, EU, US.

They reside in a shared datapack account called Avoka Bizpacks.

Initial Allocation of Bizpack Support

For a new environment Bizpack Support allocation is determined by region. A deployment property in Composer => "BizPack Region List" defines the Bizpacks that will be automatically allocated to all accounts in the Composer instance.

All Accounts automatically subscribe to the universal Bizpack library. The regions are determined by the deployment property. The screen grab below is from a Composer environment that only had AU set up in the deployment property. On initial allocation the account "Client Orgs" has Universal BizPack and AU BizPack Support (Not EU or US). It is of course possible to allocate support to the other regions by selecting them in this dialog.

Importing the Bundled Packs

The process described above is initiated by an Import function in the Release management Screen. By selecting this option "Import Bundled Packs" the process of setting up all the Bizpacks in their correct account and allocating the Bizpacks to accounts, based on the deployment property is initiated.

Gaining Access to the BizPack Libraries

We have discussed how the Bizpack support packs are allocated to the Accounts. This only makes the packs available for use to all the organizations under the account. If the organization's release does not have an associated BizPack then nothing will be allocated until the org is upgraded.

When an existing organization is upgraded, the rules in the Bizpack Support Packs available to the account are checked. If the target release for the organization matches one of the rules then the correct Bizpack Support pack will be added to the organization's path.

The BizPack Support packs can also be added manually to the search path by selecting the Update Organization Details dialog and selecting the Plus Button. Only libraries allowed at the account level will be shown for selection. I the following screens only AU and Universal Bizpacks are shown because the Account does not have US and EU selected.

Account only has Bizpack Universal and AU Support Selected

Organization can only allocate these Universal and AU Bizpacks until account setting allows EU and US

It is possible for Organizations to be subscribed to all 4 Bizpacks.

If the Bizpacks are assigned to the Organization and the release version is Bizpack Compatible the palette will have a new folder called Bizpacks. This will contain all the widgets available in the assigned Bizpacks.

Removing Access to the BizPack Datapacks

Using the same dialogs as in the previous section, you can also remove access to BizPacks for an organization or account.

Deprecated Widgets

Some existing Composer widgets have been moved to the Bizpacks.

Existing forms using these widgets may be impacted when they upgrade to 4.2 SP1.

All the widgets that have been moved to Bizpacks are still available in a deprecated folder in Composer. The forms will still generate but will return warning messages.

The form developer can either

1) Ignore the warnings - not recommended because support for these widgets may be removed.

2) Delete the widgets from the form

3) Assign the appropriate BizPack to the organization.

The Composer Palette

Core widgets that have been moved to the Bizpacks will appear in a Deprecated Folder in the composer palette. This only occurs if the organization's release version is Bizpack Compatible and the Bizpack containing the deprecated widgets is not assigned to the Organization.

The screen grab below shows widgets specific to the US region that have been moved to the US region Bizpack. The organization has not subscribed to the US Bizpack so these widgets have been moved to the deprecated folder. If the US Bizpack was assigned to the Organization the widgets would not appear in the deprecated folder.

Support still exists for these widgets however a warning will be shown at form generation to inform the user that the widget they are using has been deprecated and they should use the correct Bizpack.

Managed Support for Maguire

Composer | Form Builder | v4.3 & Higher

A Managed datapack called "Maguire Support" has been created to simplify the distribution of the Maguire datapack and ensure that the correct release of Maguire is being used on the correct version of Composer.

Details of how Managed Datapacks work is below but essentially all that is required for users of Maguire is:

1) The Client Account is set up to subscribe to Maguire Support.

2) Organizations add the Maguire Support Pack to their Path.This happens by default in Accounts that are subscribed to Maguire Support.

NOTES:

No other Maguire datapacks should be present. Custom Client packs can be created to extend Maguire.

The Maguire support pack uses the organization's composer version to ensure that the compatible version of Maguire is being used.

Composer Ant Support v4.5.0

Composer | Form Builder | v4.3 & Higher

Requirements

The minimum requirements for using Ant with Composer are as follows. Later versions of all dependencies are supported.

Dependency	Minimum version
Java	Java 7
Ant	Ant 1.8
Composer	Composer 4.5.0 server

Setting up your Ant build file

To set up your Ant build file you will need a copy of the Composer Ant JAR file *sfc-ant-4.5.0.jar* which you can get from https://transact.composer.avoka.com/resources. You will also need versions of Java and Ant that meet the minimum requirements.

Before you can use any of the Ant tasks provided in the Composer Ant JAR file, you will need to make them available by adding the following *taskdef* tag to your build file, specifying the appropriate JAR file path for your system.

Make sure you set loaderref="sfc" in the *taskdef* tag otherwise most tasks will not work.

```
<property name="sfcant.path" value="<insert path to jar file>/sfc-ant-4.5.0.jar"/>
<taskdef resource="com/avoka/sfc/ant/antlib.xml" classpath="${sfcant.path}" load-
erref="sfc"/>
```

Here is an example of a complete Ant build file that exports an organization (using *sfc-export-org*, see below). It assumes that you have placed the Composer Ant JAR file in a lib directory at the root of your project.

Sample build file

```
<?xml version="1.0" encoding="UTF-8"?>

<project name="sample" default="export.samples" basedir=".">
	<property name="lib.dir" value="${basedir}/lib"/>
	<property name="target.dir" value="${basedir}/target"/>
	<property name="sfcant.path" value="${lib.dir}/sfc-ant-4.5.0.jar"/>
	<property name="user" value="<insert your username here>"/>
	<property name="password" value="<insert your password here>"/>

	<!-- Make the sfc-ant tasks available -->
```

```
	<taskdef resource="com/avoka/sfc/ant/antlib.xml" classpath="${sfcant.path}" load-
erref="sfc"/>

	<!-- Set up the connection details for the composer server you want to use -->
	<sfc-server id="uscloud" url="https://transact.composer.avoka.com/composer" user-
r="${user}" password="${password}"/>

	<target name="export.samples">
		<mkdir dir="${target.dir}"/>
		<sfc-export-org account="Common" organization="Samples" server="uscloud" toFi
e="${target.dir}/Samples.zip"/>
	</target>
</project>
```

Composer Ant Tasks

The Composer Ant JAR file provides the following Ant tasks.

- sfc-export-account
- sfc-export-org
- sfc-form-spec
- sfc-import
- sfc-publish
- sfc-server

---

sfc-export-account

The *sfc-export-account* task exports all organizations in an account to a ZIP file on the local file system.

Task Parameters

Attribute	Description	Required	Default
account	The name of an account in the Composer workspace. This account will be exported.	Yes	
server	The Composer server to use, specified as the id of an *sfc-server* task defined elsewhere in the build file.	Yes	
toFile	The path of the ZIP file to be created.	Yes	
overwrite	Whether or not to overwrite the destination ZIP file if it exists already. Valid values are *true* and *false*.	No	true

Example

```
<sfc-server id="uscloud" url="https://transact.composer.avoka.com/composer" user-
r="user@avoka.com" password="********"/>
<sfc-export-account account="Sample Account" server="uscloud" toFile="${basedir}/sample-
account.zip"/>
```

---

sfc-export-org

The *sfc-export-org* task exports a single organization or library in an account to a ZIP file on the local file system.

Task Parameters

Attribute	Description	Required	Default
account	The name of an account in the Composer workspace.	Yes	
organization	The name of an organization in the account. This organization will be exported.	Yes	
server	The Composer server to use, specified as the id of an *sfc-server* task defined elsewhere in the build file.	Yes	
toFile	The path of the ZIP file to be created.	Yes	
overwrite	Whether or not to overwrite the destination ZIP file if it exists already. Valid values are *true* and *false.*	No	true

Example

```
<sfc-server id="uscloud" url="https://transact.composer.avoka.com/composer" user-
r="user@avoka.com" password="********"/>
<sfc-export-org account="Sample Account" organization="Sample Organization" server-
r="uscloud" toFile="${basedir}/sample-org.zip"/>
```

---

sfc-form-spec

The *sfc-form-spec* task generates a form specification (form spec) XML file for one or all forms in a project.

Task Parameters

Attribute	Description	Required	Default

account	The name of an account in the Composer workspace.	Yes	
organization	The name of an organization in the account.	Yes	
project	The name of a project in the organization. Form specs will be generated for one or all forms in this project.	Yes	
form	The name of the form for which a form spec will be generated. Use "*" to generate form specs for all forms in the project.	No	"*"
adjustLocalizationKeys	Whether or not to replace #s in localization keys with their resolved values. Valid values are *true* and *false*.	No	false
preserveNullValues	Whether or not properties with no value defined are kept in the form spec. Valid values are *true* and *false*.	No	false
server	The Composer server to use, specified as the id of a *sfc-server* task defined elsewhere in the build file.	Yes	
toFile	The path of the file to be created. This will be an XML file in the case of a single form or a ZIP file if generating form specs for all forms in a project.	Yes	
overwrite	Whether or not to overwrite the destination ZIP file if it exists already. Valid values are *true* and *false.*	No	true

Examples

Generate a form spec for a single form

```
<sfc-server id="uscloud" url="https://transact.composer.avoka.com/composer" user-
r="user@avoka.com" password="********"/>
<sfc-form-spec account="Sample Account" organization="Sample Organization" project="My
Project" form="Test Form" server="uscloud" toFile="${basedir}/test-form-spec.xml"/>
```

Generate form specs for all forms in a project

```
<sfc-server id="uscloud" url="https://transact.composer.avoka.com/composer" user-
r="user@avoka.com" password="********"/>
<sfc-form-spec account="Sample Account" organization="Sample Organization" project="My
Project" form="*" server="uscloud" toFile="${basedir}/my-project-specs.zip"/>
```

sfc-import

The *sfc-import* task imports an organization ZIP file from the local file system to an account on a Composer server.

Task Parameters

Attribute	Description	Required	Default
account	The name of an account in the Composer workspace. The organization in the ZIP file will be imported into this account. NOTE: This will override the account name from the ZIP file. Leave this attribute blank to use the account name from the ZIP file.	No	
organization	The name of an organization in the account. This organization will be imported. NOTE: This will override the organization name from the ZIP file. Leave this attribute blank to use the organization name from the ZIP file.	No	
server	The Composer server to use, specified as the id of a *sfc-server* task defined elsewhere in the build file.	Yes	
file	The path to the ZIP file to be imported.	Yes	

Example

```
<sfc-server id="uscloud" url="https://transact.composer.avoka.com/composer" user-
r="user@avoka.com" password="********"/>
<sfc-import account="Sample Account" organization="Sample Organization" server="uscloud"
file="${basedir}/sample-org.zip"/>
```

---

sfc-publish

The *sfc-publish* task creates a publish ZIP or FAR file of a Composer form, suitable for importing into Avoka Transact Manager.

Task Parameters

Attribute	Description	Required	Default
account	The name of an account in the Composer workspace.	Yes	
organization	The name of an organization in the account.	Yes	

project	The name of a project in the organization.	Yes	
form	The name of a form in the project. This form will be published.	Yes	
variants	A comma separated list of variants to include in the published ZIP file. Use "*" for all variants or "!" to just use the default variant as defined in the template.	No	"!"
format	The type of file to create. Valid values are *zip* and *far*. Note: *far* is only supported when a single variant is specified.	No	zip
server	The Composer server to use, specified as the id of an *sfc-server* task defined elsewhere in the build file.	Yes	
toFile	The path to the ZIP/FAR file to be created.	Yes	
overwrite	Whether or not to overwrite the destination ZIP/FAR file if it exists already. Valid values are *true* and *false*.	No	true

Example

```
<sfc-server id="uscloud" url="https://transact.composer.avoka.com/composer" user-
r="user@avoka.com" password="********"/>
<sfc-publish format="zip" account="Sample Account" organization="Sample Organization" pro-
ject="Samples" form="My Sample" server="uscloud" toFile="${basedir}/my-sample.zip"/>
```

---

sfc-server

The *sfc-server* task is used to define connection information for a Composer server. Other Composer Ant tasks have a server attribute that needs to match the id of an *sfc-server* task defined in the build file.

Task Parameters

Attribute	Description	Required	Default
id	The id assigned to this server definition. This is an arbitrary identifier used to identify the Composer server in other Ant tasks and must be unique within the build file.	Yes	
url	The URL of the Composer server; for example, https://transact.composer.avoka.com/composer.	Yes	
user	A Composer user name, used to authenticate the Ant build process when accessing the Composer server referenced by the url attribute.	Yes	
password	The password for the Composer user identified by the user attribute.	Yes	

Example

```
<sfc-server id="uscloud" url="https://transact.composer.avoka.com/composer" user-
r="user@avoka.com" password="********"/>
```

Composer On-premise Upgrade Guide (for major release)

Composer | Form Builder | v4.3 & Higher

On-premise Composer installations will require regular upgrades to take advantage of new features and defect fixes.

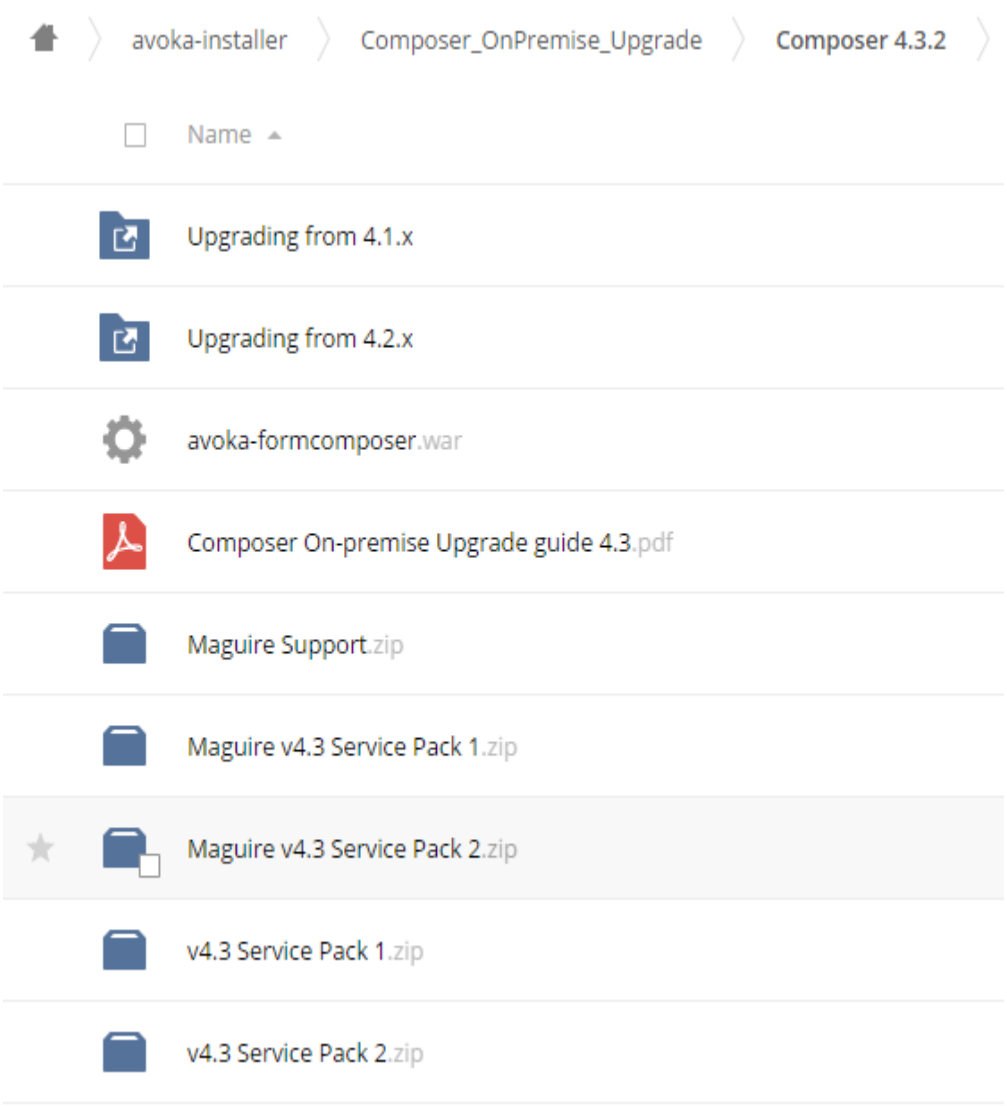
A special build is required for on-premise upgrades. This is available to customers at => On Premise Composer Upgrade Directory

Each available release is listed under this folder and includes all the items required for a successful upgrade. This will include:

- The release "WAR" file.
- Maguire Support Pack
- Maguire datapack
- Service Pack if not a major release.

If the client does not have the latest version of composer on their instance they will need to additional actions to the ones below. These are described later in this document.

The screenshot below shows the artifacts in the files.avoka.com upgrade folder for a 4.3.2 upgrade. The 2 sub-folders "Upgrading from 4.1.x" and "Upgrading from 4.2.x" are only required by clients who are NOT on 4.3 and will be required later in the upgrade.



Upgrade Steps

Composer upgrades require the deployment of a new WAR file in the JBoss application server. The WAR file can be downloaded from On Premise Composer Upgrade Directory, you can contact the Avoka support team to obtain access. The upgrade will automatically upgrade the database if required, it is therefore important to create a full backup of the database before a new WAR file is deployed.

For updating composer war file:

1. Backup composer database
2. Backup old the avoka-formcomposer.war from [JBOSS_HOME]\server\default\deploy\,
3. Shutdown JBoss for Composer
4. Copy new avoka-formcomposer.war to [JBOSS_HOME]\server\default\deploy\, replacing the old avoka-form-composer.war
5. Delete [JBOSS_HOME]\server\default\tmp folder

6. Start JBoss for Composer
7. The database upgrade process will be kicked off once the JBoss application server has been started.
8. You will now be able to access Composer as per usual

Follow up Actions

After a successful upgrade the following actions are required:

1) Updating the Service Pack(s) (Not required for major releases)

Service packs are named "v4.x Service Pack x". These need to be uploaded to Composer by a site administrator in the System Account. For major release (eg. 4.4, 5.0 etc) this is not required.



2) Adding Maguire Packs

If there is a corresponding Maguire release, there will be Maguire Pack Libraries that need to be uploaded to the Avoka Datapack Account

1) Maguire Support - This is the rule pack for Maguire. It will use the Organization's version to select the correct Maguire Pack. When uploading it can replace the existing pack.

2) Maguire vn.n - This is the Maguire Pack itself and needs to be added to the Avoka Datapack. There may be multiple to add.

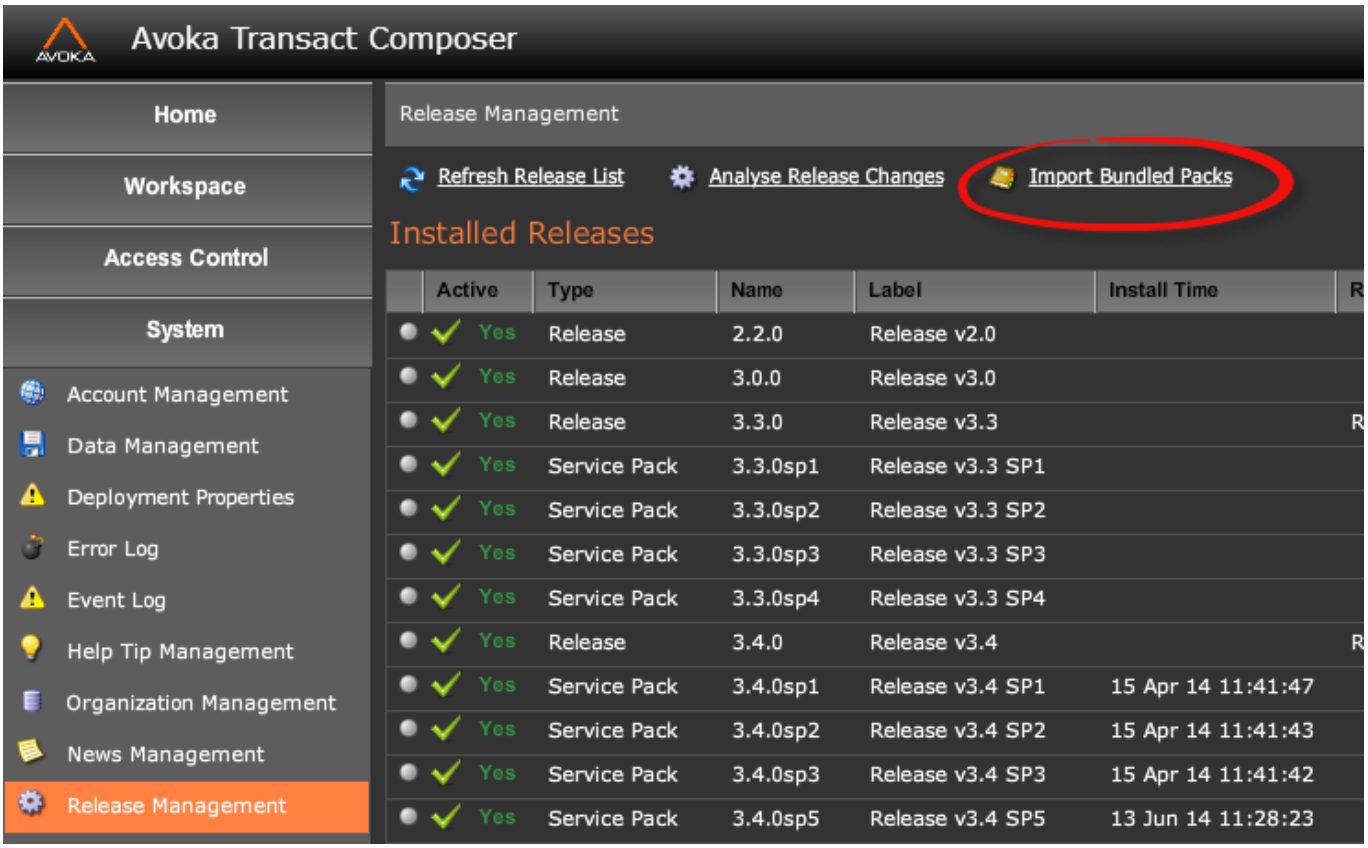


3) Activating Bizpacks

Bizpacks are bundled into the WAR file however they are not activated automatically. To do this the administrator needs to:

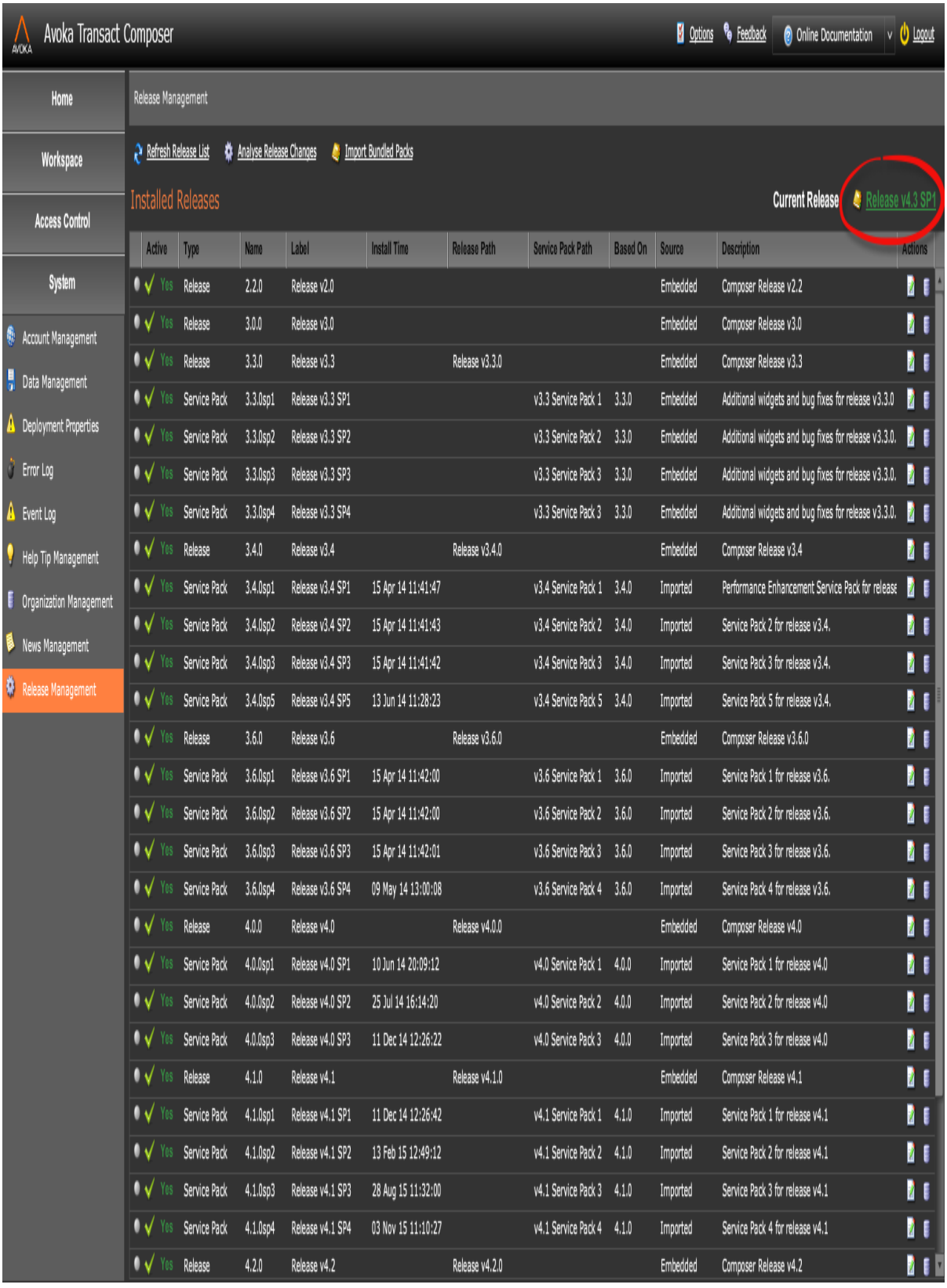
1. Go to: System=>Release Management
2. Select the "Import Bundled Packs" button.

This will add the Bizpacks to the Avoka Bizpacks account automatically.



4) Setting the current release

1. Go to: System=>Release Management
2. Select the "Current Release" link on the top right hand corner.
3. The dialog will contain a dropdown with the available releases. Select the latest release and hit "Ok".



5) Clients not on the latest release

For clients who are currently not on the latest release. That is 4.1.x or 4.2.x the additional steps are described below

Upgrading From Older Versions

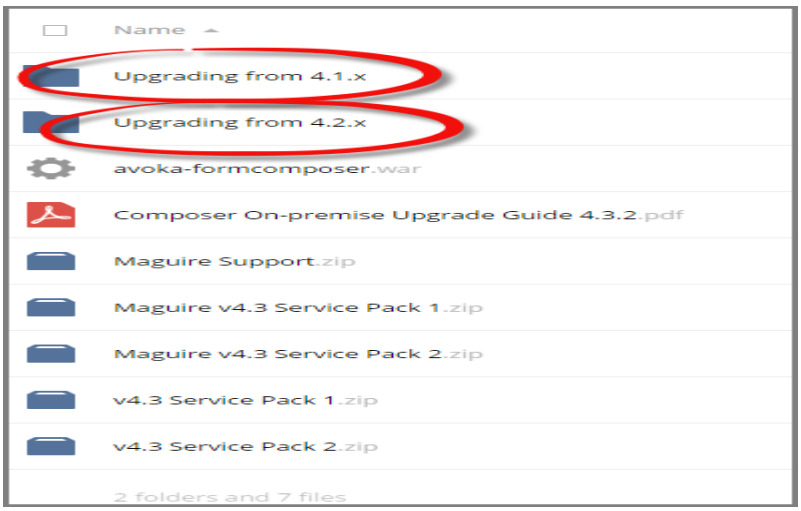
Rollback

To rollback the upgrade please follow the following steps:

1. Shutdown JBoss for Composer
2. Restore the database
3. Copy the backed up avoka-formcomposer.war to [JBOSS_HOME]\server\default\deploy\
4. Delete [JBOSS_HOME]\server\default\tmp folder
5. Start JBoss for Composer
6. You will now be able to access Composer as per usual
7. Contact the Avoka support team and provide them with the relevant JBoss log files for further analysis

Upgrading from an older version

The following two folders are required by clients upgrading from version 4.1 and version 4.2.



These contain the datapacks that need to be loaded to ensure the environment is fully up to date.

Note that if you are upgrading from 4.1.x to 4.3.x you will need to add the datapacks from both folders.

Steps to upgrade

1. Open the folder required. Note that not all of the following may be present.
2. If there are Bizpacks in the folder you need to add them to the Avoka Bizpacks Account in Composer.
3. If there are Maguire packs they need to be added to the Avoka Datapacks Account in Composer.
4. If there are Service Packs or Release Packs these need to be added to the System Account in Composer.

Composer On-premise Upgrade Guide (service pack release only)

Composer | Form Builder | v4.3 & Higher

1) Updating the Service Pack(s)

Service packs are named "v4.x Service Pack x". These need to be uploaded to Composer by a site administrator in the System Account. For major releases (eg. 4.4, 5.0 etc) this is not required.



2) Adding Maguire Packs

If there is a corresponding Maguire release, there will be Maguire Pack Libraries that need to be uploaded to the Avoka Datapack Account

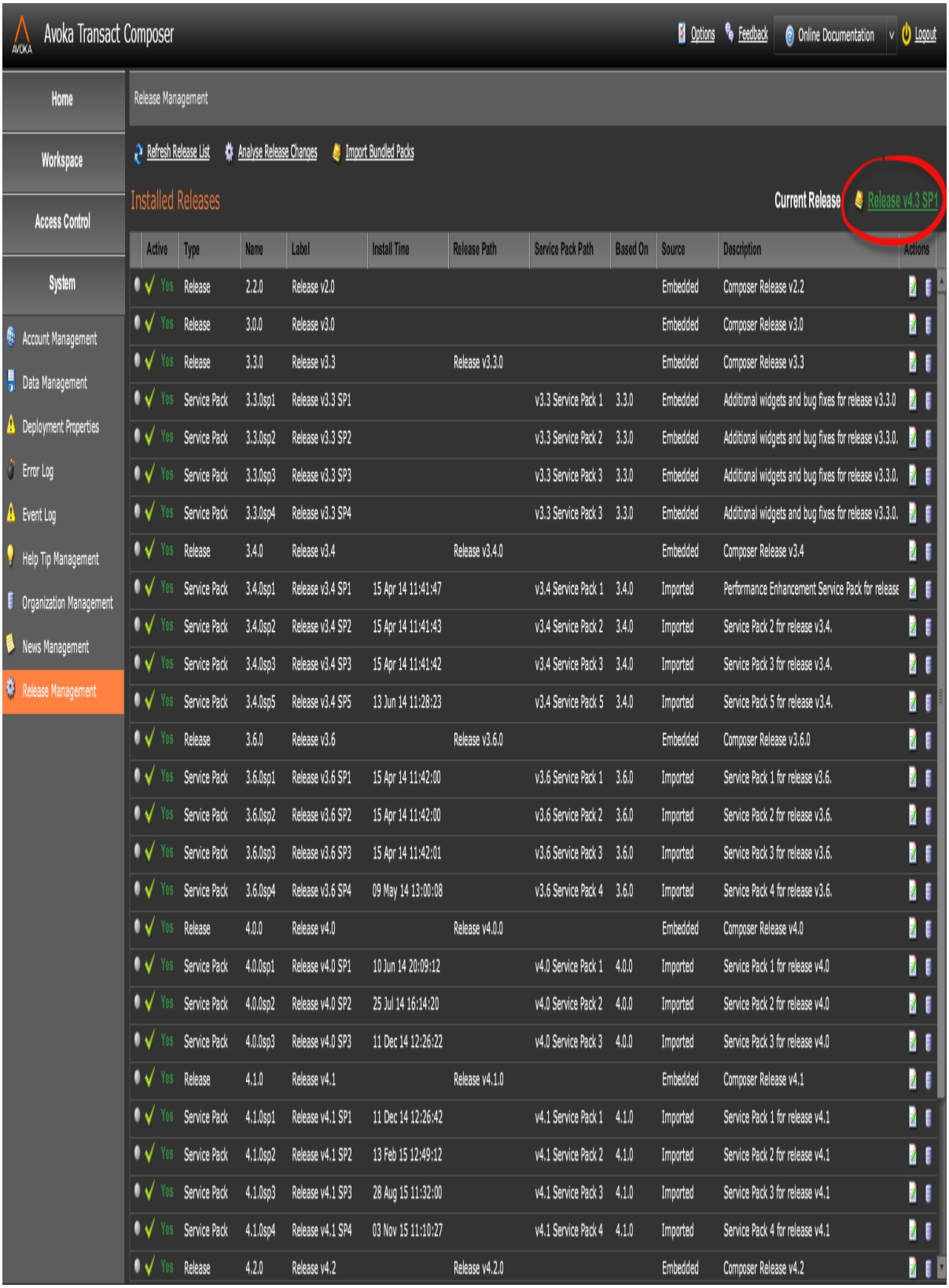
1) Maguire Support - This is the rule pack for Maguire. It will use the Organization's version to select the correct Maguire Pack. When uploading it can replace the existing pack.

2) Maguire vn.n - This is the Maguire Pack itself and needs to be added to the Avoka Datapack. There may be multiple to add.



3) Setting the current release

1. Go to: System=>Release Management
2. Select the "Current Release" link on the top right hand corner.
3. The dialog will contain a dropdown with the available releases. Select the latest release and hit "Ok".



Active	Type	Name	Label	Install Time	Release Path	Service Pack Path	Based On	Source	Description	Actions
Yes	Release	2.2.0	Release v2.0					Embedded	Composer Release v2.2	
Yes	Release	3.0.0	Release v3.0					Embedded	Composer Release v3.0	
Yes	Release	3.3.0	Release v3.3		Release v3.3.0			Embedded	Composer Release v3.3	
Yes	Service Pack	3.3.0sp1	Release v3.3 SP1			v3.3 Service Pack 1	3.3.0	Embedded	Additional widgets and bug fixes for release v3.3.0	
Yes	Service Pack	3.3.0sp2	Release v3.3 SP2			v3.3 Service Pack 2	3.3.0	Embedded	Additional widgets and bug fixes for release v3.3.0.	
Yes	Service Pack	3.3.0sp3	Release v3.3 SP3			v3.3 Service Pack 3	3.3.0	Embedded	Additional widgets and bug fixes for release v3.3.0.	
Yes	Service Pack	3.3.0sp4	Release v3.3 SP4			v3.3 Service Pack 3	3.3.0	Embedded	Additional widgets and bug fixes for release v3.3.0.	
Yes	Release	3.4.0	Release v3.4		Release v3.4.0			Embedded	Composer Release v3.4	
Yes	Service Pack	3.4.0sp1	Release v3.4 SP1	15 Apr 14 11:41:47		v3.4 Service Pack 1	3.4.0	Imported	Performance Enhancement Service Pack for release	
Yes	Service Pack	3.4.0sp2	Release v3.4 SP2	15 Apr 14 11:41:43		v3.4 Service Pack 2	3.4.0	Imported	Service Pack 2 for release v3.4.	
Yes	Service Pack	3.4.0sp3	Release v3.4 SP3	15 Apr 14 11:41:42		v3.4 Service Pack 3	3.4.0	Imported	Service Pack 3 for release v3.4.	
Yes	Service Pack	3.4.0sp5	Release v3.4 SP5	13 Jun 14 11:28:23		v3.4 Service Pack 5	3.4.0	Imported	Service Pack 5 for release v3.4.	
Yes	Release	3.6.0	Release v3.6		Release v3.6.0			Embedded	Composer Release v3.6.0	
Yes	Service Pack	3.6.0sp1	Release v3.6 SP1	15 Apr 14 11:42:00		v3.6 Service Pack 1	3.6.0	Imported	Service Pack 1 for release v3.6.	
Yes	Service Pack	3.6.0sp2	Release v3.6 SP2	15 Apr 14 11:42:00		v3.6 Service Pack 2	3.6.0	Imported	Service Pack 2 for release v3.6.	
Yes	Service Pack	3.6.0sp3	Release v3.6 SP3	15 Apr 14 11:42:01		v3.6 Service Pack 3	3.6.0	Imported	Service Pack 3 for release v3.6.	
Yes	Service Pack	3.6.0sp4	Release v3.6 SP4	09 May 14 13:00:08		v3.6 Service Pack 4	3.6.0	Imported	Service Pack 4 for release v3.6.	
Yes	Release	4.0.0	Release v4.0		Release v4.0.0			Embedded	Composer Release v4.0	
Yes	Service Pack	4.0.0sp1	Release v4.0 SP1	10 Jun 14 20:09:12		v4.0 Service Pack 1	4.0.0	Imported	Service Pack 1 for release v4.0	
Yes	Service Pack	4.0.0sp2	Release v4.0 SP2	25 Jul 14 16:14:20		v4.0 Service Pack 2	4.0.0	Imported	Service Pack 2 for release v4.0	
Yes	Service Pack	4.0.0sp3	Release v4.0 SP3	11 Dec 14 12:26:22		v4.0 Service Pack 3	4.0.0	Imported	Service Pack 3 for release v4.0	
Yes	Release	4.1.0	Release v4.1		Release v4.1.0			Embedded	Composer Release v4.1	
Yes	Service Pack	4.1.0sp1	Release v4.1 SP1	11 Dec 14 12:26:42		v4.1 Service Pack 1	4.1.0	Imported	Service Pack 1 for release v4.1	
Yes	Service Pack	4.1.0sp2	Release v4.1 SP2	13 Feb 15 12:49:12		v4.1 Service Pack 2	4.1.0	Imported	Service Pack 2 for release v4.1	
Yes	Service Pack	4.1.0sp3	Release v4.1 SP3	28 Aug 15 11:32:00		v4.1 Service Pack 3	4.1.0	Imported	Service Pack 3 for release v4.1	
Yes	Service Pack	4.1.0sp4	Release v4.1 SP4	03 Nov 15 11:10:27		v4.1 Service Pack 4	4.1.0	Imported	Service Pack 4 for release v4.1	
Yes	Release	4.2.0	Release v4.2		Release v4.2.0			Embedded	Composer Release v4.2	

Composer Attachment Definitions

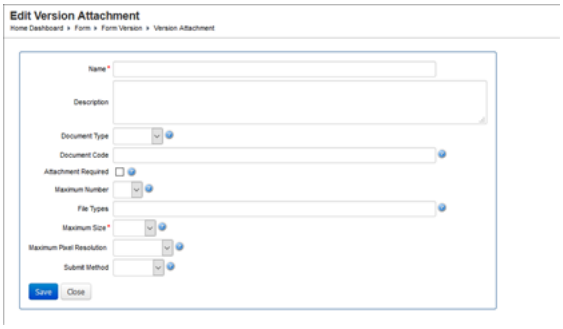
Composer | Form Builder | v4.3 & Higher

Attachments can be defined either in Transaction Manager, or in the form itself via Composer. Both approaches allow Attachment specific restrictions to be specified.

The following sample displays attachments defined in Transaction Manager and Composer.

Transaction Manager

Attachments can be defined against a Form Version.



Attachment defined in Transaction Manager

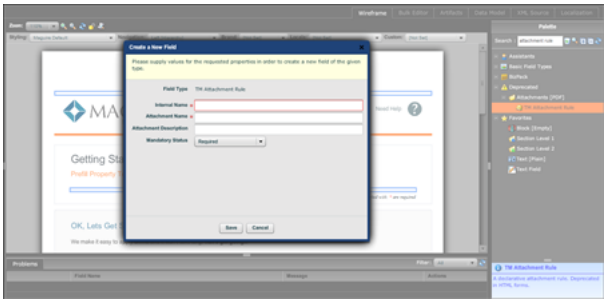
Pros

- Attachments can be modified without needing to redeploy the Form

Composer Form

TM Attachment Rule

Attachments can be added and specified by adding a TM Attachment Rule widget to a form.



Attachment Rule in Composer

Pros

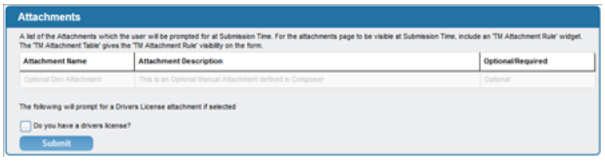
- Composer Attachments can have inclusion rules defined (So they can be dependent on other form elements/conditions)

Cons

- Attachment modifications need form redeployment

TM Attachment Table

The TM Attachment Table will display all of the Transaction Manager Attachments (Including TM Attachment Rules defined within Composer) currently applicable to the form. Any TM Attachment Rules with inclusion rules defined need to be true to be visible in this list.



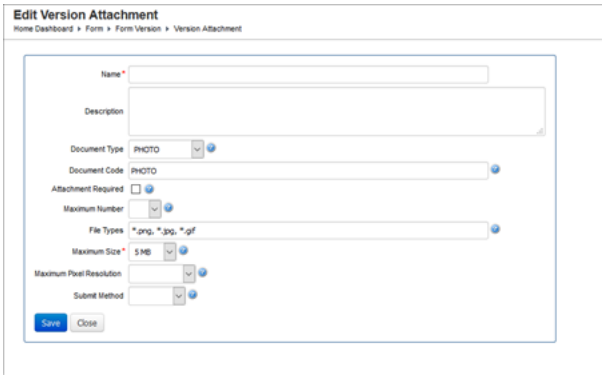
TM Attachment Table rendered in a form

Composer Attachment Restrictions

Composer | Form Builder | v4.3 & Higher

Attachments configured in Transaction Manager or Composer can be defined with certain restrictions to improve data integrity or constrain the submitted document. The following restrictions can be configured:

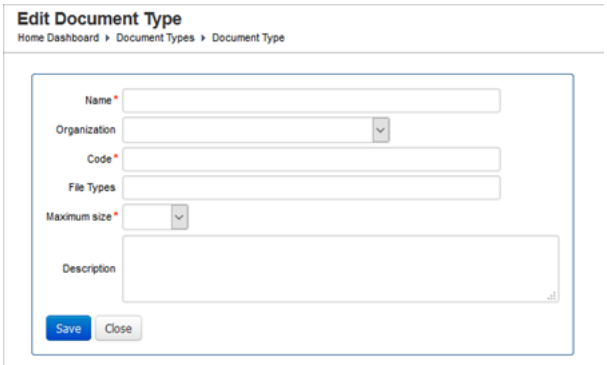
- Mandatory/Optional
- Document Type - Predefined Rules that specify File Type, Size, Description
- Document Code - Code associated with attachment
- Mime Type/File Ext - Comma separated list of file extensions
- File Size - Maximum File Size
- Pixel Resolution - Maximum Image Pixel Resolution
- Submit Method - Electronic (Upload a document) / Manual (Upload a document)



Example of an Attachment configured in Transaction Manager using a Predefined Photo Document Type which reuses its configured File Type, Size and Pixel Resolution Requirements

Transaction Manager - Document Types

Generic Document Types to be used across multiple Attachments can be predefined in Transaction Manager. This allows file type, size, description, code to be preconfigured. Document types can also be specific to an Organisation.



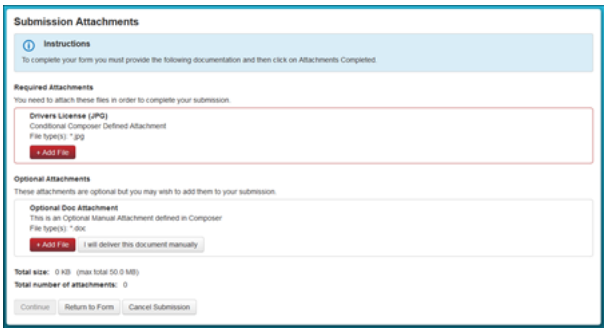
Document Type Editor in Transaction Manager

Composer Attachment Time

Composer | Form Builder | v4.3 & Higher

Attachments can be added at different times in the overall flow.

Attachments defined within Transaction Manager or as TM Attachment Rules in Composer will prompt the user after submission. The user will also be informed of any applicable File Type or Size restrictions applicable to each attachment.



Form Prompting User after Submission for attachments

The following sample displays TM Attachments defined in Transaction Manager and Composer. The user is prompted to provide the attachments after form submission.

Sample form: https://tm.demo.avoka.com/govassist/servlet/SmartForm.html?formCode=attachments6

Composer Attachments in Repeats

Composer | Form Builder | v4.3 & Higher

Attachments in repeats or table-rows can be a little problematic, because Attachments are (by definition) global to the submission, whereas attachments in repeats are (again, by definition) repeating. The problem is that every attachment needs to have a unique name, and by default all the attachments in a repeat will have the same name.

There is a fairly easy way to work around this:

1. In the hierarchy tree, click the toolbar **gear** to show hidden fields.
2. Locate the appropriate attachment.
3. Expand it, and find the sub-field named Attachment Name. Open the properties dialog for Attachment Name.
4. You will notice that the Initial Value of this field is set to **$../{attachment.name}** - this will evaluate to the **Attachment Name** property at the Attachment Rule level.
5. What we need to do is use a Calculation Rule to make the value of this sub-field a unique value.

There are several ways to do this, but probably the easiest way is to use a calculation rule based on concatenating a string with the current repeating row index. You will need to drop a **Repeatable Block Index** field into your repeat in order to use this in a calculation.

Transact for Salesforce App

Journey Platform | All Personas | 5.1 & Higher

Transact for Salesforce App was created in response to recurring feedback we received from our customer base. Feedback around the Customer Experiences produced by the Journey platform has always been very positive and we've been fortunate enough to assist many businesses transform the way they interact with their customers online. However, several clients were concerned about the staff experience and didn't want their sales and customer service staff to have to login to a separate system in order to initiate these online transactions and track their progress to completion. Many of these customer facing staff spend their day inside a **CRM**[1] environment, becoming very productive in this environment, and for a growing number of our clients this CRM is Salesforce.com.

Transact for Salesforce App is about improving the staff experience, specifically the staff who spend the bulk of their time working in Salesforce and dealing directly with the customer.

We asked ourselves some questions:

- What if Sales staff could send out an Application Form to one of their contacts, directly from the Contact record, and have it pre-filled with information from Salesforce?
- How could we make it easy for Customer Service staff to assist customers complete their transactions without having to leave the Salesforce environment?
- Could we record completed form-based transactions into the CRM such that they get added to the historical timeline of the customer relationship?

Transact for Salesforce App is designed to answer the following questions:

1. Provide a Salesforce embedded UI that allows users to view and manage transactions that reside on the Journey platform without having to access the platform directly, removing the need for them to leave the Salesforce environment.
2. Provide a generic integration channel between Manager and force.com to avoid custom integration work for each use case.
3. Provide a simple framework for integration that enables Salesforce consulting partners and Salesforce administrators with the ability to more easily configure and control the receipt of data from and release of data to the Journey platform.

You can easily install and configure the application, which supports:

- Salesforce - all editions of Salesforce are supported. All Avoka APEX code is written for API version 33.
- The Manager versions.

You can perform the following application administration and managementactivities:

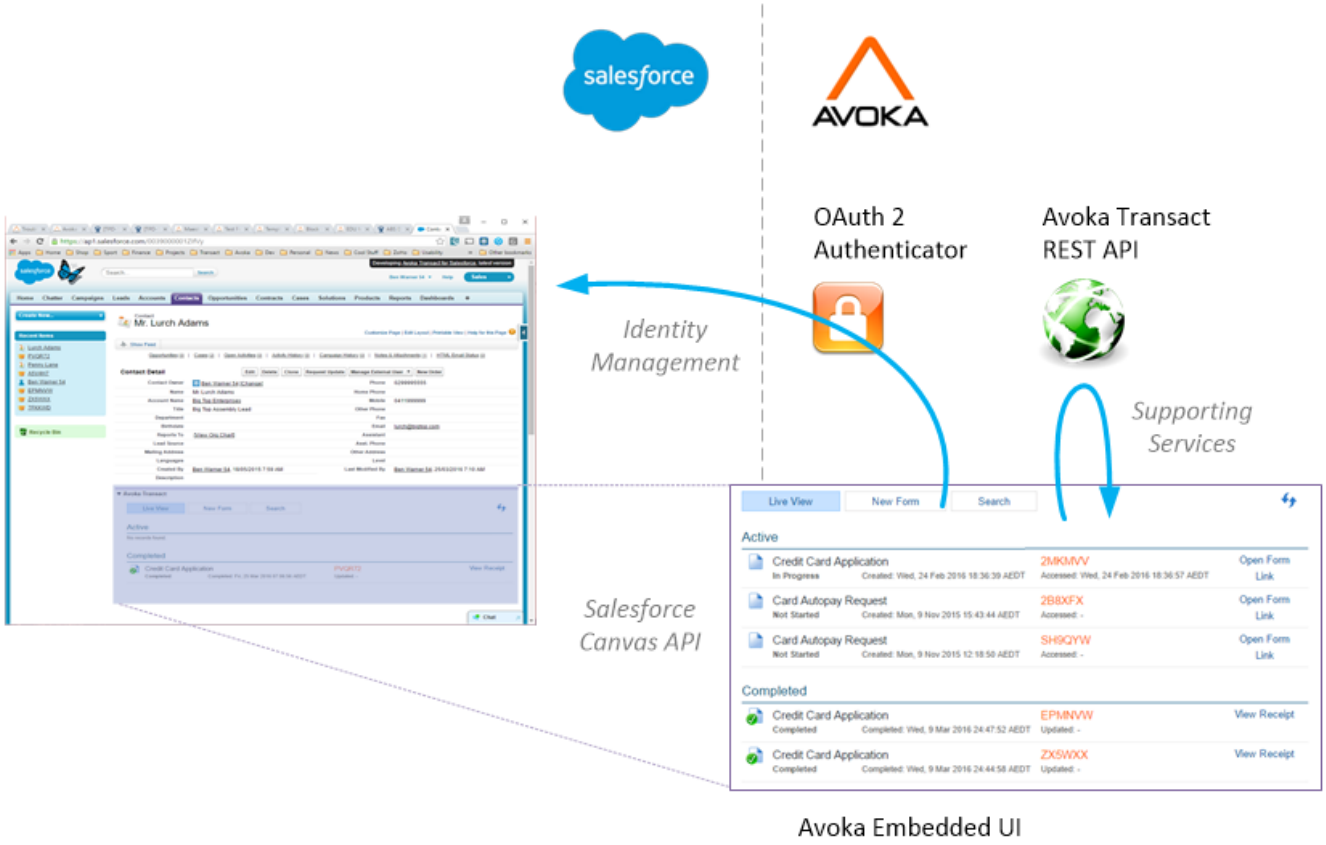
---

[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.

- Configure Transact for Salesforce App
- Enable or restrict access to forms
- Access security tokens
- Manage user licenses

User Interface Design

The user interface is facilitated by a Canvas application that is hosted on the Manager server and rendered into the Salesforce UI using the Canvas framework. A set of REST APIs exposed on the Journey platform support the UI for accessing, creating and updating transactions.



Styling

Styling of the Canvas UI is designed to blend in with the Salesforce Desktop UI by utilizing complementary colors, fonts and button styles. Where appropriate, subtle changes have been applied to provide a more modern look and feel, yet still maintaining a natural integration into the visual design of the platform.

Navigation

Page navigation is facilitated by a simple tab metaphor at the top of the UI, supporting navigation between the following page functions:

1. My Forms - shows form transactions that are assigned to the current user. This page is typically only surfaced when the user accesses the UI via the Chatter feed.
2. Live View - shows form transactions that are linked to the current Salesforce object (SObject).
3. New Form - is used to select and create a new form transaction based on an available form template.
4. Search - is customer service style search facility to find forms using a search term. Results may include forms that are anonymous and unlinked.

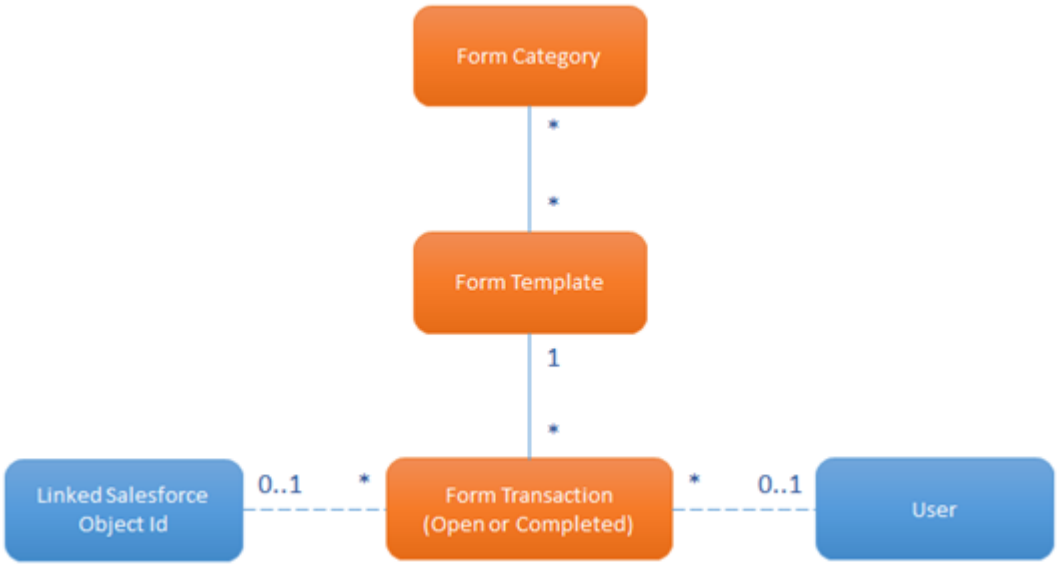
Information Architecture

A form template represents the code and configuration behind a form design and encapsulates the data model, business rules and presentation assets for a form. Within the Canvas UI, form templates are accessed by navigating to the 'New Form' page and selecting a form category. Form templates can be used to create new form transactions to be opened immediately or sent to a contact.

Form templates can belong to one or more form category. Form categories and the forms that belong to them are configured in Manager.

A form transaction may be assigned to a single user or assigned to no user (anonymous). Forms assigned to a user will appear under the **My Forms** page when that user is logged in to Salesforce and accessing the Canvas UI.

A form transaction may be linked with a single Salesforce object or unlinked. Forms linked to a Salesforce object will appear under the **Live View** page when viewing the detail layout for that object.



Note: Primary Entities in Avoka (orange) and Salesforce (blue) and their Relationships

Integration Approach

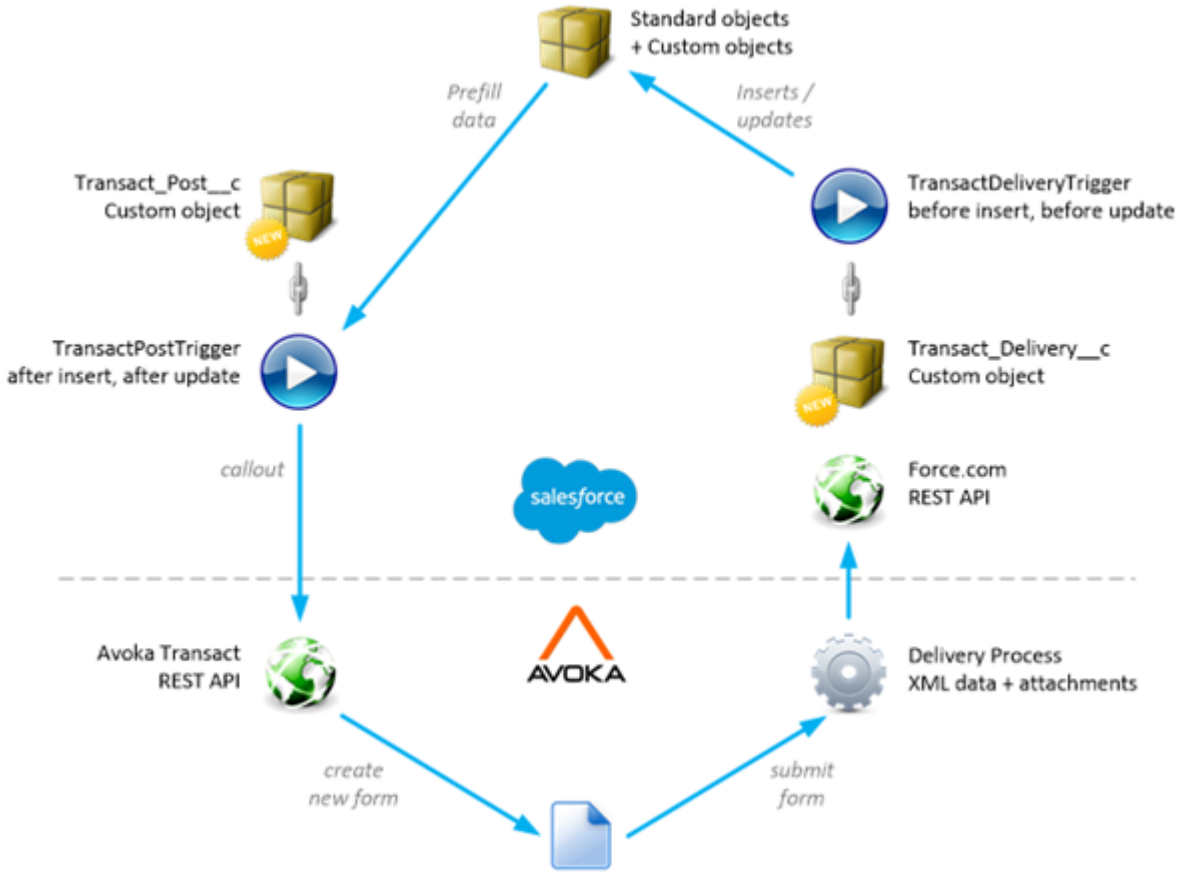
UI

The Transact for Salesforce App UI is built in and hosted on the Journey platform. This UI is integrated into the Salesforce page layouts by utilizing the Salesforce Canvas API.

APIs

API integration is facilitated by REST services on both platforms. Journey platform exposes REST services to accommodate the assignment of new forms from Salesforce, while REST APIs on the Salesforce side are utilized by Manager to deliver completed submissions.

Within Salesforce, these REST services are encapsulated by custom objects and associated triggers to facilitate the bi-directional communications, as illustrated in the following diagram.



Authentication / Identity Management

Authentication of Salesforce users into Journey Manager is done via OAuth2. Users who access the Transact for Salesforce App UI must be authenticated with Salesforce. Upon first time access to the Avoka UI, a placeholder account is created on the Manager server with the required permissions to perform the functions associated with the UI.

Data Security

This section provides an assessment of the data security considerations associated with the managed package. For detailed information on data security in the Journey platform, request a copy of the Journey platform security architecture documentation.

The base installation of the package does not require access to any Salesforce standard or custom objects other than those contained within the package itself.

Control over which data is shared with Manager for prefill into forms, and which data is updated by incoming submissions is under complete control of the Salesforce administrators. This is a manual configuration function that is not enabled by default.

Data Access Controls

The solution provides a framework for Salesforce administrators to release data to Manager for the purposes of prefilling forms before sending them to contacts to complete and submit. This framework is facilitated by a set of Apex classes that consume configuration based rules to build dynamic **SOQL**[1] queries for the retrieval of data.

Within this apex code, the following access controls are employed:

- With Sharing - all Apex classes in the package have the 'with sharing' control activated such that sharing rules of the current user are considered.
- Data Access Rights - where configuration requires that a field be prefilled into a form, the current user's data access rights are first checked before that field is retrieved with SOQL. Similarly, where data flows back into Salesforce, the user's permission to create or update fields and objects is checked prior to performing any change. The Apex class **`SecurityController`** has been included to accommodate these checks. The following code is an example of these checks in action: **`if(SecurityController.isFieldAccessible (objectType, fieldName))`**

Off-Platform Data Storage

The Journey platform is a system of engagement, not a system of record, so no data is retained in our platform, other than configuration data, longer than it is required in order to complete the data capture transaction and forward the data to the appropriate system of record.

Due to the requirements of our most sensitive clients, such as federal government agencies, police departments, and financial services organization, Manager data security is exceptionally strong. All data in transit and at rest is encrypted according to the industry's highest standard and flexible purge policies allow clients to ensure that sensitive data is removed from the system at the earliest opportunity.

---

[1] Salesforce Object Query Language (SOQL) is used to construct simple but powerful query strings in the queryString parameter in the query() call, in Apex statements, in Visualforce controllers and getter methods, or in the Schema Explorer of the Force.com IDE.

Salesforce Organization Impacts

Manager does not include pre-built data capture experiences targeting specific use cases, but rather provides a template based design environment with many pre-built components that can be strung together to rapidly produce these experiences. Each of our customers use Manager to facilitate different use cases within their organization, and these are typically built to spec.

With this consideration, it is difficult to provide a general assessment of the impacts the Salesforce package will have on the client's Salesforce Organization. Rather, the impacts need to be assessed on a case by case basis and Avoka will work with each client to ensure that the impacts understood.

The following sections outline the potential areas of impact and the considerations relating to each.

Licensing

The Transact for Salesforce App package can be used on any edition of Salesforce. Licenses are made available for a monthly fee on a per user basis. Any user who accesses any component included in the managed package must be licensed to use the solution. The primary functions required by users will be:

- Utilize the Canvas App to manage forms against a Salesforce object.
- Access delivery records of Manager transactions.
- Trigger functions within their Salesforce environment that in turn create a post request to Manager to assign a form.

Data Storage Requirements

The following data storage considerations need to be assessed on a case by case basis:

- Transaction Volumes - the volume of transactions being produced from the Journey platform and delivered into Salesforce will be a key factor in the determination of data storage requirements.
- Binary attachments - where transactions call for binary attachments to be provided, and these attachments are designated for delivery into Salesforce, the storage requirements for these binary attachment needs to be considered. Controls can be put in place in Journey platform to restrict the size of attachments that may be provided by users.
- Purge Strategies - the data in the Journey platform custom object records is transient by design. Data purge strategies can be effectively applied to remove records that are no longer required. Purge rules may be developed to remove only the binary attachments and submission XML while maintaining the metadata about each delivered submission.

API Usage Volumes

The following API usage considerations need to be assessed on a case by case basis:

- Transaction Volumes -the volume of transactions being produced from the Journey platform and delivered into Salesforce will be a key factor in the determination of API usage requirements. Each submission delivered into Salesforce will utilize a minimum of one API call, but more where attachments are to be delivered also.
- Prefill Generation - where a user is interacting with the Canvas App and attempts to send a form to a contact, this requires an API call to the prefill generation service.
- Where Journey Manager Post records are created as a result of events in the Salesforce environment, these trigger calls to the Journey platform.

Batch Processing

There are currently no batch processes included in the package. Clients may choose to implement their own purge process to remove post and delivery records older than a certain date.

Reporting

There are currently no reports or dashboards included in the package.

Next, learn how to configure Transact for Salesforce App.

Integrate Salesforce with Manager

Journey Platform | All Personas | 5.1 & Higher

You can configure Salesforce to integrate it with Managerfor the first time.

To configure your Salesforce environment for Journey Manager integration via APIs, follow the process below:

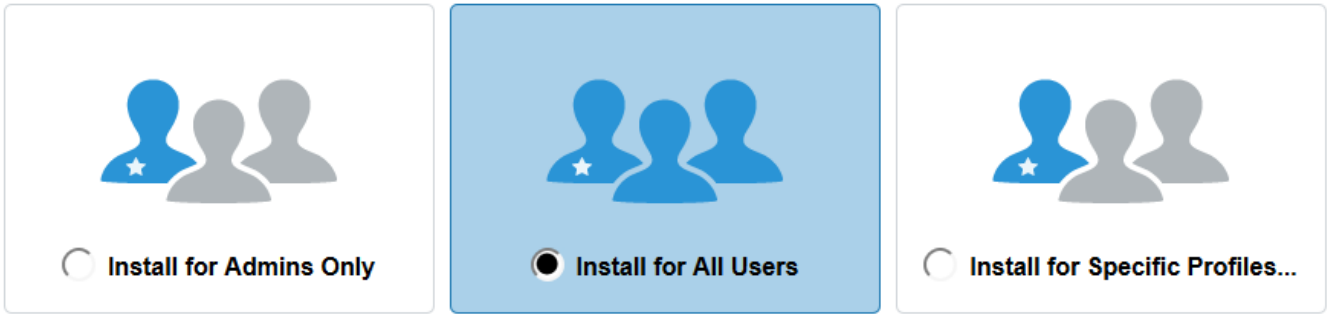
1. Install the Transact for Salesforce App.
2. Create the API User Account in Salesforce.
3. Create the Journey Manager Connected App in Salesforce.
4. Provide the User Account and Connected App details to the Journey Manager team.

Install the Transact for Salesforce App

Install the latest version of the Transact for Salesforce App that supports your version of Manager:

To commence installation of a Salesforce package, upon opening the package URL in the browser you are required to login to the Salesforce environment that you wish to install into.

When installing the package it is best to elect to Install for All Users as shown below. This will avoid users being unable to access the app due to insufficient privileges.



If you have already installed the app for Admins Only or Specific Profiles and some users are unable to access the app due to **Insufficient Privileges**, you can resolve this issue as described in Insufficient Privileges: You do not have the level of access necessary to perform the operation you requested...

Create the API User Account in Salesforce

We recommend you create a separate Salesforce user account dedicated to Manager integrations. This allows you to keep your real user credentials private and lets you restrict the permissions of the integration user as appropriate.

To create the user and its permission profile for API level integration, contact your Salesforce administrator or refer to the Salesforce documentation. The following items should be considered when doing so:

1. Suggested User Account attributes:
   - First Name - blank
   - Last Name - Transact
   - Alias - transact
   - Email - Suggest using the email address of your Salesforce administrator
   - Username - transact@yourdomain.com.<environment> . Append the environment name for non-prod environments
   - Nickname - Transact
2. The user must have API access enabled
3. See also: https://help.salesforce.com/HTViewHelpDoc?id=integrate_what_is_api.htm&language=en_US
4. The user must have full access to the Avoka custom objects to deliver submission records into Salesforce. The app package includes permission sets to help manage user access to Manager objects. Allocate the included Manager administrator permission set to this user.
5. The user must also have appropriate permissions to any objects you may wish to update when a submission is delivered as these update functions will be performed as this API user.
6. Ensure the user is licensed to use the Transact for Salesforce App - How to manage user licensing for the Salesforce App

It is important to understand the impacts of password expiry on integration user accounts.

Create the Manager Connected App in Salesforce

1. Under **App Setup**, click **Create > Apps**
2. Under **Connected Apps**, click **New**
3. Suggested Connected App attributes:
   - Connected App Name - Transact Connector
   - API Name - Transact_Connector
   - Contact Email - Suggest using the email address of your Salesforce administrator
4. Ensure **Enable OAuth Settings** is checked with the following attributes set:

- Enable OAuth Settings - Yes (selected)
- Callback URL - https://<Transact Domain>/sfdc/secure/account/home.htm , where <Transact Domain> is the domain of your Transact environment
- Use Digital Signatures - No (unselected)
- Selected OAuth Scopes - Access and manage your data, Access your basic information, Provide access to your data via the Web. It is important to set all and only these 3 scopes

5. Under **Web App Settings** enter the following values:
   - Start URL - https://<Transact Domain>/sfdc/servlet/SmartForm.html?formCode=AT4SF-canvas , where <Transact Domain> is the domain of your Manager environment
   - Enable SAML - No (unselected)
6. Under **Mobile App Settings** leave blank.
7. Under **Canvas App Settings** ensure **Force.com Canvas** is selected and enter the following values:
   - Force.com Canvas - Yes (selected)
   - Canvas App URL - https://<Transact Domain>/sfdc/servlet/SmartForm.html?formCode=AT4SF-canvas , where <Transact Domain> is the domain of your Manager environment
   - Access Method - Signed Request (POST)
   - SAML Initiation Method - None
   - Locations (Selected) - Chatter Tab, Layouts and Mobile Cards, Visualforce Page
   - Lifecycle Class - leave blank
8. Click **Save** to update the changes.

Configure Permitted Users for the Connected App

The app currently only supports Administrator Approved users.

1. Under 'Administration Setup', click Manage Apps > Connected Apps.
2. Locate the Manager connected app and click Edit.
3. Under 'OAuth Policies' selected 'Admin approved users are pre-authorized' for Permitted Users.
4. Click **Save** to update the changes.
5. Again under 'Administration Setup', click **Manage Apps > Connected Apps,** locate your application and click the App name link to view the detail page.
6. Under **Profiles**, click **Manage Profiles** and select the profiles you wish to pre-approve for app usage. This is typically System Administrator and Standard User.
7. Click **Save** to update the changes.

Create the API Named Credential in Salesforce

If your integration scenarios include pushing form task assignments from Salesforce to Avoka in batch or

within an Apex trigger (not very common), you will need to setup an API user in Avoka and configure the named credential in Salesforce. The named credential is not required for other scenarios and this step may be skipped.

1. Under Administration Setup, click Security Controls > Named Credentials
2. Under Named Credentials, click New Named Credential and enter the following Named Credential attribute values:

- Label - Transact API Credential
- Name - Transact_API_Credential, which must match this exactly
- URL - https://<Transact Domain>/manager/secure/rest/groovy-service-invoke/v2/at4sf-postrequest/v1/ , where <Transact Domain> is the domain of your Manager environment, the version number at the end (v1) must refer to the version of the AT4SF-PostRequest service in Manager that you wish to call (version 1 at the time of writing). For older Manager environment you may need to use the GroovyServiceServlet URL, such as https://<Transact Domain>/manager/secure/GroovyServiceServlet?sfmServiceName=AT4SF-PostRequest
- Certificate - leave blank unless you have certificate based authentication in place.
- Identity Type - Named Principal
- Authentication Protocol - Password Authentication
- Username and Password - the username and password used must correspond to a user who belongs to the required organization.

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

For the IP White-Listing, be aware that if your Manager server has IP white-listing enabled, you will need to add the Salesforce IP ranges to the white-list for the task creation calls to be successful. See:

- https://help.salesforce.com/articleView?id=000003652&type=1
- Post Request: 403 Forbidden - You don't have permission to access /manager/secure/GroovyServiceServlet on this server.

Information required by the Journey Manager Team

Once the integration User Account and Connected App have been correctly configured in Salesforce, the Manager administration staff will require the following information:

- Salesforce Login URL, for example, https://login.salesforce.com or https://test.salesforce.com.
- API User - Username
- API User - Password
- API User - Security Token
- Connected App - Consumer Key
- Connected App - Consumer Secret

To retrieve this information from Salesforce

- How to retrieve the Security Token for a Salesforce User Account.
- How to retrieve the Consumer Key and Secret for a Salesforce Connected App.

Next, learn how to make forms available for Salesforce users.

Enable Access to the Transact for Salesforce App

Journey Platform | All Personas | 5.1 & Higher

You can give user profiles access to the Transact for Salesforce App pages using the Salesforce App Menu.

To enable access to the TTransact for Salesforce App:

1. Under **App Setup**, click **Create > Apps**.
2. Under **Apps**, click **Edit** next to the **Transact App**.
3. Under **Assign to Profiles** on the App Edit Page, select the user profiles that you wish to give access to the Transact for Salesforce App.
4. Click **Save** to update the changes.

Next, learn how to group forms using form categories

Manage User Licenses

Journey Platform | All Personas | 5.1 & Higher

You can manage which users are licensed to use the Transact for Salesforce App when a limited number of user license is available.

> If your organization has a site license for the Transact for Salesforce App, all users will automatically be licensed and no license management will be required. This is relevant where your organization has purchased a limited number of seats.

To manage user license:

1. Under **App Setup > Installed Packages**, click **Manage Licenses** next to the Transact for Salesforce App package
2. Under Licensed Users, add and remove licensed users as required.
3. Click **Save** to update the changes.

Next, learn how to troubleshoot Transact for Salesforce App.

Retrieve the Consumer Key and Secret for the Salesforce App

Journey Platform | All Personas | 5.1 & Higher

If you're setting up the Salesforce integration using a Connected App, you'll need to get your Consumer Key, Secret and Security Token from Salesforce.

To retrieve the key and secret:

1. Login to Salesforce with the target user account.
2. Select **App Setup** and click **Create > Apps**.
3. Open the **Connected App** target and retrieve the **Consumer Key** and **Consumer Secret** from the API (oAuth) section.

   The Consumer Secret may be protected by a **Click to reveal** link.

Next, learn how to retrieve the security token for the Salesforce user account.

Retrieve the Security Token for the Salesforce User Account

Journey Platform | All Personas | 5.1 & Higher

Each Salesforce user account has an associated security token that is commonly required when integrating with Salesforce. Your Salesforce security token should have been emailed to you when you set up your Salesforce account or the last time you reset your password. If you don't know the security token for a selected user account, there is no way to retrieve it through the UI, so you may want to first search for it through old emails sent to the user's registered email address before resetting the security token.

> If the user account is already being used for integration purposes, resetting the security token may break these integrations until they are updated with the new token.

To retrieve the security token:

1. Login to Salesforce with the target user account.
2. Click **Username** in the top right and select **Setup** or **My Settings** from the menu.
3. Under **Personal Setup**, click **My Personal Information > Reset My Security Token**.

   > If you can't see the **Reset My Security Token** option, see Missing "Reset My Security Token" Option in the troubleshooting guide.

4. Click **Reset Security Token**.
5. Check the **User's email inbox** for an email from Salesforce advising of the new security token.

Next, learn how to manage user license for the Salesforce app.

Remove Password Expiry on a Salesforce User Account

Journey Platform | All Personas | 5.1 & Higher

When setting up an API integration user with Salesforce, a password expiry event will break the integration with little or no warning. You can disable password expiry for these selected users only and keep your integration channels open.

Creating a 'Password Never Expires' Permission Set

Follow this procedure to create a permission set that will disable password expiry:

1. Under Administration Setup, click Manage Users > Permission Sets
2. Create a new Permission Set with label 'Password Never Expires' and save
3. Edit the System Permissions against this Permission Set, select the 'Password Never Expires' permission and save

Adding the 'Password Never Expires' Permission Set to a User Account

Once you have created the 'Password Never Expires' Permission Set, follow this procedure to add the permission set to a selected user account:

1. Under Administration Setup, click Manage Users > Users
2. Open the target user account and edit Permission Set Assignments
3. Add the 'Password Never Expires' Permission Set.
4. Click **Save** to update the changes.

To improve security for integration user accounts you could look at one or more of the following strategies:

- Limit permissions to only the objects it needs to access
- Enable IP address restrictions to the IP ranges of the integrating system(s)
- Turn off integration clients (e.g. Outlook)
- Turn off all tabs

Next, learn how to restrict access for selected forms to user groups.

Configure the Transact for Salesforce App UI in Salesforce

Journey Platform | All Personas | 5.1 & Higher

You can configure the Transact for Salesforce AppCanvas UI in Salesforce so that users can access this UI on the object detail pages.

Create the Visualforce Page

A Visualforce page needs to be created for each Salesforce object type you wish to make the Transact for Salesforce App UI available for. In this example, we will outline the process for enabling the UI for the Contact object.

1. Select **App Setup** and click **Develop > Pages.**
2. Select **Visualforce Pages** and click **New**.
3. Suggested Page Attributes:
   - Label - Transact - Contact
   - Name - Transact_Contact
   - Description - Contains the view for the Contact object.
   - Leave defaults for other options
4. Enter the following Visualforce Markup and click save:

```
<apex:page standardController="Contact">
<!-- Begin Transact App -->
  <div id="transact-container"></div>
  <apex:canvasApp applicationName="Avoka_Transact_Connector"
        containerId="transact-container"
        width="100%"
        border="0"
        scrolling="no"
        height="350px"
        maxHeight="infinite"
        parameters="{
            SObjectId:'{!HTMLENCODE(Contact.Id)}',
            SObjectType:'Contact',
            ContactFirstName:'{!HTMLENCODE(Contact.FirstName)}',
            ContactLastName:'{!HTMLENCODE(Contact.LastName)}',
            ContactEmail:'{!HTMLENCODE(Contact.Email)}',
            EnableUserView:'false',
            EnableObjectView:'true',
            EnableNewForm:'true',
            EnableGlobalSearch:'true'
        }" />
  <!-- End Transact App -->
</apex:page>
```

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

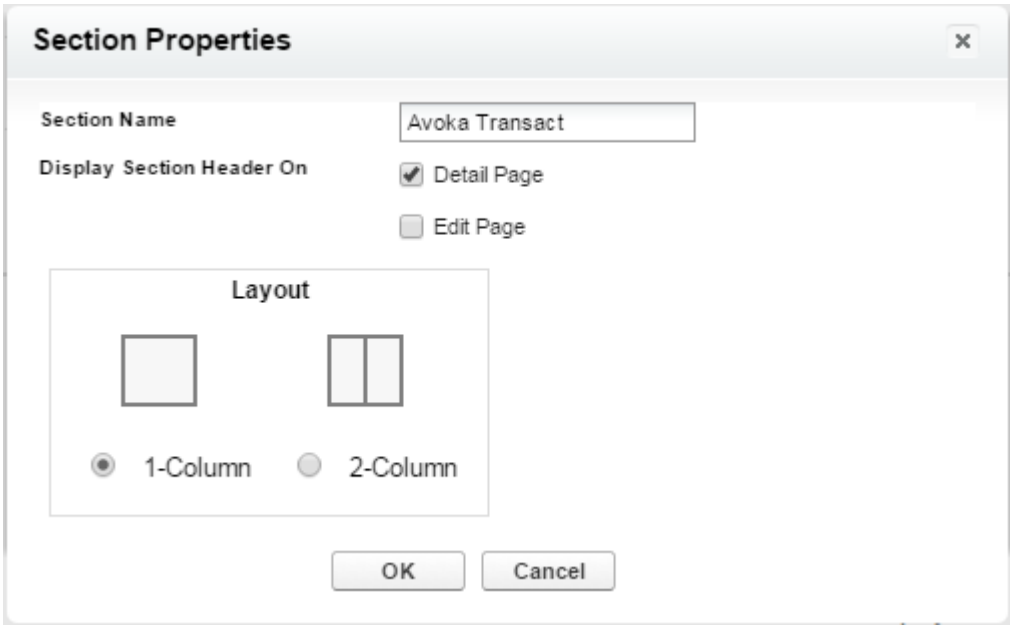
Visualforce Markup Attributes

- standardController="Contact" - should be set to the name of the object you wish to present the Visualforce page on - in this case 'Contact', if we were creating a Visualforce page for the Account object this value would be 'Account'
- applicationName="Avoka_Transact_Connector" - should be set to the name of the Avoka Transct connected app that has been configured - in this case Avoka_Transact_Connector
- containerId="transact-container" - set the containerId to the id of the div above the canvas app definition - in this case transact-container
- width="100%" - Stretch the canvas app to the full available width
- border="0" - Use no border on the frame
- scrolling="no" - set scrolling off because we enable the scrollbars in the page layout (see next section)
- height="350px" - default height of app
- maxHeight="infinite" - app can grow in height unconstrained
- Parameter SObjectId - (required) the Id of the Sobject being accessed - in this case just taking the Id of the current Contact record
- Parameter SObjectType - (required) a plain english name for the type of the SObject being accessed, may be used in some places throughout the Canvas UI
- Parameter ContactFirstName - (optional) The first name of the contact for 'Send to Contact' forms. As we are illustrating with the Contact record we are able to take the Contact's first name, but for other object types we may leave this parameter out
- Parameter ContactLastName - (optional) The lastname of the contact for 'Send to Contact' forms. As we are illustrating with the Contact record we are able to take the Contact's last name, but for other object types we may leave this parameter out
- Parameter ContactEmail - (optional) The default value to include for the email address of the contact for 'Send to Contact' forms. As we are illustrating with the Contact record we are able to take the Contact's email, but for other object types we may leave this parameter out
- Parameter EnableUserView - (optional) A 'true' or 'false' flag to indicate whether to show the User's My Forms page in the Canvas App. Default is false.
- Parameter EnableObjectView- (optional) A 'true' or 'false' flag to indicate whether to show the Object Forms page in the Canvas App. Default is false.
- Parameter EnableNewForm- (optional) A 'true' or 'false' flag to indicate whether to show the New Form page in the Canvas App. Default is false.
- Parameter EnableGlobalSearch- (optional) A 'true' or 'false' flag to indicate whether to show the Search page in the Canvas App. Default is false.
- Parameter InitialView - (optional) The name of the page to present when the Canvas App loads. Valid values are 'UserView', 'ObjectView', 'NewForm', 'GlobalSearch'. Default is to initialise on the first included page on the left.
- Parameter FormCategoryFilter - (optional) {Version 1.5} A comma separated list of form category names to present in the New Form view. Category names should exactly match the names in Transaction Manager. Without this parameter, all available categories containing forms will be presented.

The HTMLENCODE directive is required for parameter values to avoid unescaped characters.

Add the Visualforce page to the Page Layout

1. Under **App Setup**, click **Customize > Contacts > Page Layouts.**
2. Under Contact Page Layouts find Contact Layout and click edit.
3. In the widget palette at the top, choose Visualforce pages.
4. Drag a new Section onto the page in the location you want the UI to appear.
5. Name it as Transact.
6. Clear the **Edit Page** and choose **1-Column** layout.



7. Drag the Visualforce page into the section just created.
8. Click **Save** to update the changes.
9. Edit the Forms properties, set height to 350px and select **Show Scrollbars.**
10. Click **Save** to update the changes.

Next, learn how to enable access to restricted forms

Configure Transact for Salesforce App

Journey Platform | All Personas | 5.1 & Higher

You can configure Manager to integrate it with the Salesforce App, which has to be done once before the first use.

> Check that the Salesforce support was included when Manager was installed.

To check the Salesforce support is included:

1. Log on to Manager.
2. Select **Forms > Form Spaces** and search for the Salesforce space.
3. If the Salesforce space exists, the installation was done correctly and you can proceed with the following configuration.
4. If the Salesforce support is not installed, submit a request to the Temenos Cloud Hosting Team (ACS portal) to install the Salesforce support module.

To configure your Manager environment to support the Transact for Salesforce App:

1. Create the API User Account in Manager
2. Import the Manager extension packages
3. Configure the imported modules

Create the API User Account in Manager

> This step is only required if you are performing task assignment operations from Salesforce to Manager.

1. Select **Security > User Accounts**
2. Click **New**
3. Select the **User Details** tab and enter the following values:
   - Login Name - at4sf-client
   - User Type - Local
   - Password - take a note of the password you use and ensure the **Change Password After Login** checkbox is unselected
   - Given Name(s) - AT4SF
   - Family Name - Client User
   - Email - use the email address of your Manageror Salesforce administrator
   - Portal - Manager
4. Click **Save** to update the changes.

5. Select the **Roles** tab and assign the Administrator role to the user.
6. Click **Save** to update the changes.
7. Select the **Organizations** tab and assign your organization to the user.
8. Click **Save** to update the changes.

Configure the Salesforce API Client Details

1. Select **Services > Service Connections**
2. Locate the Salesforce Service Connection and configure its API credentials. Alternatively, for more restricted access you could create a new Salesforce API Invoke role with the following minimum permissions:
   - Admin Directory
   - Web Service Invoke
3. Click **Save** to update the changes.
4. Update the Security Manager settings as described in the Configure the Security Manager section below.

> You will need the connected app and Manager user details from Salesforce to complete this configuration. For more information, see how to retrieve the consumer key and secret for a Salesforce connected app and how to retrieve the security token for a Salesforce user account.

Import the Manager Extension Packages

To import the following extension packages, ideally in this order:

1. AT4SF-SecurityManager.zip - Contains the OAuth2 Single Sign-On Security Manager. Imported under 'Security' > 'Security Managers' > 'Import'
2. AT4SF-Services.zip - Imported under 'Services' > 'All Services' > 'Import'
3. AT4SF-Canvas.zip - Contains the Canvas UI that is presented in the Salesforce Desktop environment. Imported under 'Forms' > ' Forms' > 'Import Form Version'. Select the target organization in the import wizard.
4. AT4SF-TestForm.zip - Optional test form handy for verifying the connectivity. Imported under 'Forms' > 'Forms' > 'Import Form Version'

Configure the Salesforce User Space

> Check that the Salesforce support was included when Manager was installed.

To associate the forms with the space:

1. Select **Forms>Organizations**
2. Click on your organization

3. Select the **Spaces** tab and select the Salesforce space as an 'Assigned Space'
4. Click **Save** to update the changes.
5. Select **Forms>Forms**
6. For each of the forms you wish to show under the portal, click on the form name to edit the form
7. On the Spaces tab select the Salesforce space as an 'Assigned Space' and save
8. Don't forget to do this for the Salesforce App form 'AT4SF Canvas'

Configure the Security Manager

> You will need the app connection details from Salesforce to complete this configuration.

The security manager needs to be configured against the target Salesforce connected app and assigned to the Salesforce User Space as follows:

1. Select **Security > Security Managers**
2. Locate and open the AT4SF-UserSecurityManager and navigate to the Parameters tab.
3. Edit the 'Client Id' and 'Client Secret' parameters and update to the Consumer Key and Consumer Secret values from the Salesforce connected app (see article linked above).
4. If required, edit the URI parameters to ensure the correct domain name is being used. By default the domain will be the primary Salesforce login domain but for sandbox environments this may need to be changed to test.salesforce.com.
5. If you are using a test Salesforce instance (where the login page is under https://test.salesforce.com/) you will need to update each of the 3 URI parameters accordingly (Auth Uri, Token Uri, User Info Uri):
   - Auth Uri - https://login.salesforce.com/services/oauth2/authorize https://test.-salesforce.com/services/oauth2/authorize
   - Token Uri - https://login.salesforce.com/services/oauth2/token https://test.-salesforce.com/services/oauth2/token
   - User Info Uri - https://login.salesforce.com/services/oauth2/userinfo https://test.-salesforce.com/services/oauth2/userinfo
6. Click **Save** to update the changes.
7. Select **Forms > Form Spaces**, locate and edit the 'Salesforce' user space to ensure the Salesforce User Space is configured to use the correct security manager
8. On the 'Space' tab, ensure the 'AT4SF-UserSecurityManager' is selected in the 'Security Manager' field
9. Click **Save** to update the changes.

Configure the Delivery Service and Delivery Channel

Before delivery into Salesforce can occur the Delivery Service needs to be referenced by an organization Delivery Channel as follows:

1. Select **Forms > Organizations**
2. Locate and open your organization and navigate to the 'Delivery Channels' tab.
3. Click 'New' and enter the following values
   - Name - Salesforce Push Delivery
   - Delivery Method - Delivery Process
   - Description Transact for Salesforce App Generic Push Delivery Channel
   - Delivery Process - AT4SF-DeliveryPush
4. Click **Save** to update the changes.

Next, learn how to configure a prefill generator in Salesforce.

Make Forms Available for Salesforce Users

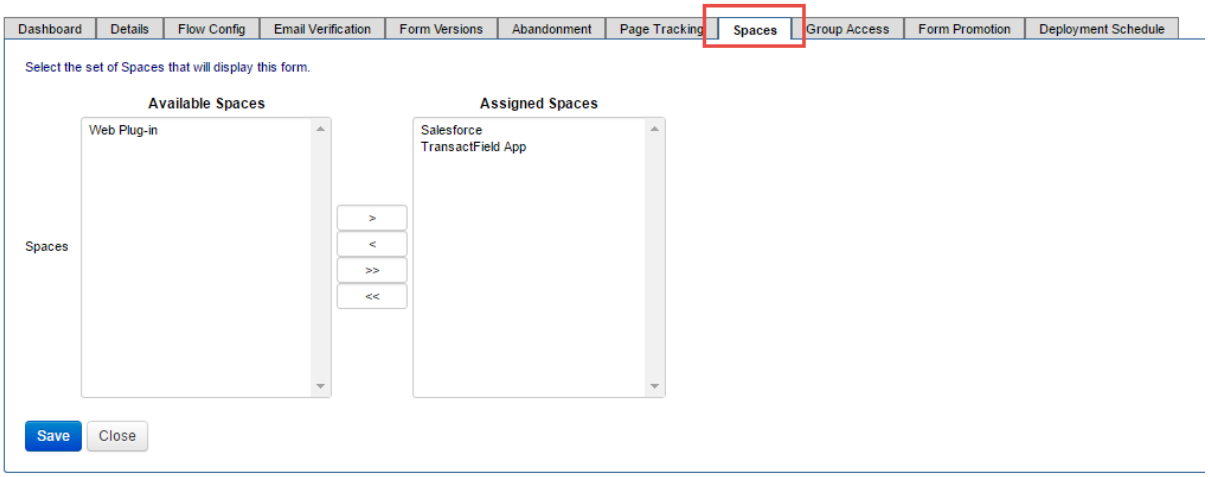
Journey Platform | All Personas | 5.1 & Higher

To make a form available to Salesforce users it must be assigned into at least one form category and published to the Salesforce User Portal.

> Forms that are not published to the Salesforce User Portal or are not assigned into at least one form category will not appear in the Salesforce Desktop UI.

To make forms available:

1. Log onto Manager
2. Select **Forms > Forms**.
3. Find the form you wish to publish to Salesforce users, open it and navigate to the Spaces tab, image below.
4. Assign the form into the Salesforce form space.



5. Ensure the form is assigned to at least one category otherwise it will not be accessible from Salesforce.
6. Click **Save** to update the changes.

Next, learn how to populate a form with Salesforce prefill data

Group Forms Using Form Categories

Journey Platform | All Personas | 5.1 & Higher

The Transact for Salesforce App uses form categories to visually group forms together, which allows users to find relevant forms quickly.

To group forms by form categories:

1. Log onto Manager
2. Select **Forms > Organizations**.
3. Locate an organization and select the **Form Categories** tab.
4. Click **New** to add the form categories.
5. Click **Save** to update the changes.
6. Select **Forms > Forms**.
7. Locate each active form version of the form you want to group and click **Edit**
8. Select the **Form Categories** tab and assign one or more categories to the form.
9. Click **Save** to update the changes.
10. Repeat the process for all forms you want to group.

Next, learn how to integrate Salesforce with Manager

Configure Salesforce Delivery Channel

Journey Platform | All Personas | 5.1 & Higher

Any form that must be delivered into Salesforce, for example, sent submissions to Salesforce, should be configured with the Salesforce Delivery Channel.

Completed Transaction Delivery

1. Select **Forms > Forms**.
2. Locate and open the form you wish to configure and select the **Details** tab.
3. Under the **Delivery Channels** section, select the Salesforce delivery channel for Production Delivery, Test Delivery or both.
4. Click **Save** to update the changes.

Abandoned Transaction Delivery

1. Select **Forms > Forms**.
2. Locate and open the form you wish to configure and select the **Abandonment** tab.
3. Under the **Abandoned Transaction Delivery** section, select the Salesforce delivery channel for Abandoned Delivery Channel.
4. Click **Save** to update the changes.

Saved Transaction Delivery

1. Select **Forms > Forms**.
2. Locate and open the form you wish to configure, open the active version record and select the **Services** tab.
3. Select the AT4SF-SavedProcessor service for Form Saved Processor.
4. Click **Save** to update the changes.

Next, learn how to configure a delivery processor in Salesforce.

Configure a Prefill Generator in Salesforce

Journey Platform | All Personas | 5.1 & Higher

You can configure a prefill generator against a particular Manager form in Salesforce by following this process:

1. Identify the data in Salesforce that you wish to use for prefill. This configuration is done in Salesforce by defining a Prefill Generator in the Prefill Generation tab of the Manager app as described below.
2. Configure where that data will be injected into a form's XML data structure at render time. This configuration is done in the Form Data Config in Manager and is described here.

Create the Prefill Generator Record

1. In the **App Menu** select **Transact** or simply view **All Tabs**, then select the **Prefill Generation Tab** and click **New.**
2. Enter the form code of the form - this value must exactly match the form code in Manager
3. Retain the default Prefill Generator Class, `TransactPrefillGenerator`, which allows you to provide JSON configuration to control which fields are included in the prefill generation
4. Specify the Generator Configuration as a JSON string according to the specification defined in this article
5. You may also specify a Sample SObject Id that will allow you to generate a Sample Prefill XML value and test your configuration (Note the SObject Id for a Salesforce object can be obtained from the URL while viewing that object)
6. Save
7. To test the configuration (where you have provided a Sample SObject Id), click the Generate Sample button - the generated prefill XML data will be added to the Sample Prefill XML field if your configuration is correct.
8. You may copy the sample prefill XML and use this in Transaction Manager to define your Input XML prefill mappings for the form.

Managing Form Versions

Where you need to perform different prefill generation between multiple versions of a form you may append an array of versions to the form code attribute to target specific versions. For example, you may wish to have 2 separate prefill generators, one for the first version of the form and another for subsequent versions. This will allow you to configure the prefill generation for new form versions prior to activation of those versions.

To utilize this feature you must specify the specific version numbers as an array directly after the form code. Some samples assuming a form code of 'my-form':

- my-form - Default configuration to be used where no specific configuration is found for the requested version. Shorthand for use of wildcard my-form[*].
- my-form[1.0] - Configuration for version 1.0 of the form. Any versions other than 1.0 will fall back to the default configuration (where no version is specified).
- my-form[1.0,2.0,3.0] - Configuration for versions 1.0, 2.0 and 3.0 of the form.

JSON Configuration Format

This section describes the syntax and supported directives of the JSON configuration consumed by the TransactPrefillGenerator class. This article assumes knowledge of JSON formatting. The JSON configuration should conform to the following structure:

```
{
    "parameters":{
        "<param-name>": "<param-value>"
    },
    "sobjects":{
        "<object-type-name>": {
            "includeFields": ["<field-name>", ... ],
            "includeRelated": ["<relationship-name>", ... ],
            "includeCustom": {"<custom-label>": "<custom-value>", ... }
        },
        ...
    }
}
```

The following example configuration illustrates all supported directives:

```
{
    "parameters":{
        "rootElementName": "SalesforceData"
    },
    "sobjects":{
        "Contact": {
            "includeFields":["Id","FirstName","LastName","Email","Phone","Title"],
            "includeRelated":["Account"],
            "includeCustom":{"FullName": "{!FirstName} {!LastName}"}
        },
        "Account": {
            "includeFields":["Name","Type","Industry"],
            "includeRelated":["Owner","Opportunities"]
        },
        "Opportunity": {
            "includeFields":["Name","Amount"]
        },
        "User": {
            "includeFields":["Name","Email","Username","Manager.Name"]
        }
    }
}
```

Supported Parameters

rootElementName - The name to be given to the root element in the generated XML prefill data. Defaults to SFDCPrefillData. This is the optional parameter.

Object Type Definition

A single JSON configuration can support any number of Salesforce object types, thereby allowing the form to be prefilled with many different object types. For example, the same form may be prefilled from a contact record or a lead record.

The configuration for an object type must be contained in an attribute labeled with the name of that object type. Note the configuration below for the Account object:

```
"Account": {
    "includeFields":["Name","Type","Industry"]
}
```

Dealing with Complex Types

If you wish to prefill data from Salesforce that is contained in complex types (e.g. Addresses), you should configure the individual data sub-attributes in your prefill configuration. For example, if you want to prefill the Shipping Address from the Account object, your configuration should be similar to this:

```
{
    "sobjects":{
        "Account": {
            "includeFields":["Name","Ship-
ɔingStreet","ShippingCity","ShippingState","ShippingPostalCode","ShippingCountry"]
        }
    }
}
```

The includeFields Directive (Required)

The includeFields directive is required for all object prefill definitions and allows you to explicitly specify the fields to be included in the prefill data with the following considerations:

- The field list should be provided as a comma separated list of string litterals where each item specifies the field name, not the label of the field.
- Parent and referential relationships can be leveraged by using the 'dot' notation to traverse the relationships by the name of the relationship (E.g. "includeFields": "FirstName,LastName,Owner.Name"). Note: if a child relationship is specified it will be ignored, but child relationships are supported in the includeRelated directive.
- Include the Id field if you want to support form user edits to the object information. If the Id value is present when the competed submission is processed on its return to Salesforce, the delivery processor may use this Id to update the object according to the updates made in the form.

The includeRelated Directive (Optional)

The includeRelated directive is optional and allows you to specify related objects to include in the prefill data with the following considerations:

- Specifying a related entity requires that the object type definition exists in the JSON configuration for that related entity.
- The object type definition must specify the name of the object type, not the name of the relationship. As an example, consider the directive "includeRelated":"Owner". The relationship name is 'Owner' but the related entity is a 'User' so the JSON configuration must contain an object definition named 'User'.
- Child relationships are supported, provided that the object is not itself a child of the primary prefill entity.

The includeCustom Directive (Optional)

- The includeCustom directive is optional an allows you to specify custom fields to include in the prefill data with the following considerations:
- Hard coded values may be used. E.g. "includeCustom":{"CampaignCode":"FF102"}
- Field references to the object context may be used but the referenced fields must be specified in the includeFields directive. E.g. "includeCustom":{"FullName": "{!FirstName} {!LastName}"}

XML Prefill Data Generation

The TransactPrefillGenerator interprets the JSON configuration and generates the XML data used to prefill the form according to the following rules:

- XML root element is always named 'SFDCPrefillData' unless an alternative root element name is specified.
- Each sub-node within the root element represents a single Salesforce object or a list of objects
- All fields specified in the includeFields or includeCustom directives will be included in the object node
- Where a related field is specified in the includeFields directive (E.g. Manager.Name), that field will generated the parent object node in a containing element named according to the relationship
- Where the includeRelated directive includes a parent or referencial relationship (E.g. "includeRelated":"Owner"), the object node for the related object will be added to the XML root node and named according to the relationship, as shown below.

```
<Owner>
    <Name>Jeff Johnson</Name>
    <Manager>
        <Name>Jenny Quack</Name>
    </Manager>
</Owner>
```

- Where the includeRelated directive includes a child relationship, a list container node will be added to the XML root node and named according to the relationship, each object node within the container will be named with the object type name, as shown below

```
<Opportunities>
    <Opportunity>
        <Name>Chicago City Store Displays</Name>
        <Amount>$50,000.00</Amount>
    </Opportunity>
    <Opportunity>
        <Name>San Francisco Mobile Signage</Name>
        <Amount>$28,000.00</Amount>
    </Opportunity>
</Opportunities>
```

The following block illustrates an example of the XML prefill data produced by the generation engine:

```
<?xml version="1.0" encoding="UTF-8"?>
<SFDCPrefillData>
    <Contact>
        <Id>00390000015C9LBAA0</Id>
        <FirstName>Kristen</FirstName>
        <LastName>Akin</LastName>
        <Title>Director, Warehouse Mgmt</Title>
        <Phone>(434) 555-3100</Phone>
        <Email>kakin@sony.com</Email>
        <FullName>Kristen Akin</FullName>
    </Contact>
    <Account>
        <Name>Volley Music</Name>
        <Type>Customer - Direct</Type>
        <Industry>Retail</Industry>
    </Account>
    <Owner>
        <Name>Jeff Johnson</Name>
        <Email>jj@domain.com</Email>
        <Username>jj@domain.com</Username>
        <Manager>
            <Name>Jenny Quack</Name>
        </Manager>
    </Owner>
    <Opportunities>
        <Opportunity>
            <Name>Chicago City Store Displays</Name>
            <Amount>$50,000.00</Amount>
        </Opportunity>
        <Opportunity>
            <Name>San Francisco Mobile Signage</Name>
            <Amount>$28,000.00</Amount>
        </Opportunity>
    </Opportunities>
    <User>
        <Name>Johny Lighning</Name>
        <Email>jl@domain.com</Email>
        <Username>jl@domain.com</Username>
        <Manager>
            <Name>Jack Sparrow</Name>
        </Manager>
    </User>
</SFDCPrefillData>
```

Custom Prefill Generators

Where the default prefill generator does not provide the level of control required, you may define your own prefill generator by developing an APEX class that implements the ITransactPrefillGenerator interface. To utilize your custom prefill generator, simply specify the class name in the relevant Prefill Generator record and any configuration required by the class.

```
global interface ITransactPrefillGenerator {

    /**
     * This function generates the prefill XML data based on the config provided.
     *
     * @param prefillObjectId The Id of the object to use for prefill generation
     * @param formPrefillConfig The string containing the generation config
     * @return The generated prefill XML data
     */
    String generatePrefillXml(Id entityId, String config);
}
```

Next, learn how to configure the Transact for Salesforce App UI in Salesforce

Populate a Form with Salesforce Prefill Data

Journey Platform | All Personas | 5.1 & Higher

Configuration of form pre-population from Salesforce generated prefill data is managed by the standard Input XML Prefill Mappings in the form data config. For more information on how to configure prefill data generation in Salesforce, see this documentation.

To populate a form with Salesforce prefill data:

1. Log onto Manager
2. Select **Forms > Forms**
3. Locate and open the form you wish to pre-populate.
4. From the Dashboard tab open the **Data Config** page of the active form version and navigate to the 'Input XML Prefill Mapping' tab.
5. Click **New Input XML** and past the Sample Prefill XML from the Salesforce Prefill Generator, enter a version name.
6. Click **Save** to update the changes.
7. Click **Edit XML Mapping** and map each input data element with the target element in the form XML.
8. Click **Save** to update the changes.

The Input XML Prefill Mappings do not accommodate mapping for repeating elements, so if you are generating pre-fill data that has multiple records of information in it (e.g. all cases related to a contact) there is currently no simple process available out-of-the-box to map this into the form and a more customized solution will need to be assessed.

Next, learn how to receive administrator alerts when delivery processing fails.

Configure a Delivery Processor in Salesforce

Journey Platform | All Personas | 5.1 & Higher

You can configure a delivery processing function against a particular Manager form in Salesforce, so that you can update your standard and custom Salesforce objects with data from the submitted form. The consideration as to which object to create or update in Salesforce is typically unique to the use case being targeted but often this involves the Contact or Lead objects.

For more information on the Salesforce standard objects, see Salesforce sales objects

Create the Delivery Processor Record

1. In the **App Menu** select **Transact** or simply view **All Tabs**, then select the **Delivery Processing Tab** and click new.
2. Enter the form code of the form - this value must exactly match the form code in Manager.
3. Retain the default Delivery Processor Class, **`TransactDeliveryProcessor`**, which allows you to provide JSON configuration to control what objects are affected by the submission and how.
4. Specify the Processing Configuration as a JSON string, as described below.
5. Click **Save** to update the changes.

Delivery Trigger Types

Delivery records are received from Manager according to the specified triggers. Transactions are typically delivered when completed, but they can also be delivered when the transaction has been saved or deemed to be abandoned, to facilitate lead generation and follow up.

Managing Form Versions

Where you need to perform different delivery processing functions between multiple versions of a form you may append an array of versions to the form code attribute to target specific versions. For example, you may wish to have 2 separate delivery processing functions, one for the first version of the form and another for subsequent versions. This will allow you to continue to accept deliveries for in-flight transactions on old form versions after a new version is activated.

To utilize this feature you must specify the specific version numbers as an array directly after the form code. Some samples assuming a form code of 'my-form':

- my-form Default configuration to be used where no specific configuration is found for the requested version. Shorthand for use of wildcard my-form[*].
- my-form[1.0] Configuration for version 1.0 of the form. Any versions other than 1.0 will fall back to the default configuration (where no version is specified).
- my-form[1.0,2.0,3.0] Configuration for versions 1.0, 2.0 and 3.0 of the form.

JSON Configuration Format

The TransactPrefillGenerator class uses the following JSON configuration:

```
{
    "parameters":{
        "<param-name>": "<param-value>"
    },
    "sobjects":{
        "<object-label>": {
            "type": "<object-api-name>",
            "matchFields": ["<field-name>", ... ],
            "insertFields": ["<field-name>", ... ],
            "updateFields": ["<field-name>", ... ],
            "sourceFields": {
                "<field-name>": "<field-mapping>",
                ...
            },
            "relations": {
                "<relationship-name>": "<object-label>",
                ...
            }
        },
        ...
    }
}
```

The following example configuration illustrates all supported directives:

```
{
    "parameters": {
        "formAssociationTarget":"new-case",
        "attachmentTarget":"new-case",
        "receiptTarget":"new-case",
        "caseNumber":"new-case.CaseNumber"
    },
    "sobjects": {
        "parent-account": {
            "type": "Account",
            "insertFields":[],
            "updateFields":[],
            "sourceFields" : {
                "Id":"[!XML://Account/AccountId]"
            }
        },
        "my-contact": {
            "type": "Contact",
            "matchFields":["LastName","Email"],
            "insertFields":["*"],
            "updateFields":["Email","Phone","MobilePhone","Title","Department"],
            "sourceFields" : {
                "Id":"[!XML://Contact/ContactId]",
                "Salutation":"[!XML://Contact/Title]",
                "FirstName":"[!XML://Contact/FirstName]",
                "LastName":"[!XML://Contact/LastName]",
                "Phone":"[!XML://Contact/Phone]",
                "MobilePhone":"[!XML://Contact/Mobile]",
                "Email":"[!XML://Contact/Email]",
                "Title":"[!XML://Contact/Position]",
                "Department":"[!XML://Contact/Department]"
            },
            "relations": {
                "Account": "parent-account"
            }
        },
        "new-case": {
            "type": "Case",
            "insertFields":["*"],
            "updateFields":["*"],
            "sourceFields" : {
                "Type":"[!XML://IssueDetails/IssueType]",
                "Reason":"[!XML://IssueDetails/Reason]",
                "Subject":"[!XML://IssueDetails/Summary]",
```

```
                "Description":"[!XML://IssueDetails/Description]",
                "Origin":"Web"
            },
            "relations": {
                "Account": "parent-account",
                "Contact": "my-contact"
            }
        }
    }
}
```

The list of supported parameters is shown below:

- formAssociationTarget - (Optional) The object (specified by referencing the relevant <object-label>) to associate the Manager form with so that it appears against that object in the Canvas App.
- attachmentTarget - (Optional) The object (specified by referencing the relevant <object-label>) to add the submitted attachments to.
- receiptTarget - (Optional) The object (specified by referencing the relevant <object-label>) to add the generated PDF receipt to.
- caseNumber - (Optional) The field (specified by referencing the <object-label>.<field-name>) from which to retrieve the Salesforce case number. This case number is returned to Manager and stored against the submission.

Object Type Definition

A single JSON configuration can define impacts to any number of Salesforce objects. Each object configuration must be given a label that is unique in the configuration. Note the configuration below for a Contact object:

```
"my-contact": {
    "type": "Contact",
    "matchFields":["LastName","Email"],
    "insertFields":["*"],
    "updateFields":["Email","Phone","MobilePhone","Title","Department"],
    "sourceFields" : {
        "Id":"[!XML://Contact/ContactId]",
        "Salutation":"[!XML://Contact/Title]",
        "FirstName":"[!XML://Contact/FirstName]",
        "LastName":"[!XML://Contact/LastName]",
        "Phone":"[!XML://Contact/Phone]",
        "MobilePhone":"[!XML://Contact/Mobile]",
        "Email":"[!XML://Contact/Email]",
        "Title":"[!XML://Contact/Position]",
        "Department":"[!XML://Contact/Department]"
    },
    "relations": {
        "Account": "parent-account"
    }
}
```

The matchFields Directive (Optional)

The Transact for Salesforce App version 1.5. The matchFields directive provides a basic de-duplication function by first searching for a matching record before determining whether to perform an insert or update. To utilize this optional function, provide an array of Salesforce field names that should be used to identify an existing record. If a record is found in Salesforce where ALL the listed fields are an exact match for the information provided in the form, this record will be updated.

- The field list should be provided as a comma separated list with no spaces or new lines where each item specifies the field name, not the label of the field.

For example, the following directive will cause the delivery processor to search for a Contact object with matching LastName and Email:

```
"my-contact": {
    "type": "Contact",
    "matchFields":["LastName","Email"],
    ...
```

The insertFields Directive (Required)

The insertFields directive is required for all object definitions and allows you to explicitly define which fields will be used in an insert scenario (when an Id value is not available).

- The field list should be provided as a comma separated list with no spaces or new lines where each item specifies the field name, not the label of the field.
- The field list should not include the "Id" field.
- A wild card value of "*" may be used to denote that all sourced fields should be affected in the insert.

The updateFields Directive (Required)

The updateFields directive is required for all object definitions and allows you to explicitly define which fields will be used in an update scenario (when an Id value is available).

- The field list should be provided as a comma separated list with no spaces or new lines where each item specifies the field name, not the label of the field.
- There is no need to include the "Id" field - it will be added automatically if it is available.
- A wild card value of "*" may be used to denote that all sourced fields should be affected in the update.

The sourceFields Directive (Required)

The sourcefields directive is required for all object definitions and allows you to define the mappings from the submitted XML data into the object fields.

```
"sourceFields" : {
    "Id":"[!XML://Contact/ContactId]",
    "Salutation":"[!XML://Contact/Title]",
    "FirstName":"[!XML://Contact/FirstName]",
    "LastName":"[!XML://Contact/LastName]",
    ....
}
```

- Hard-coded values may be used instead of XML mappings where appropriate, for example, **"Origin":"Web"**
- XML mappings can be specified such that values from the XML submission data are injected into the fields at processing time. XML mappings must be structured, for example, **[!XML://<xml location>]**

  where <xml location> is the URI to the XML element containing the required value. Note the root XML node is not specified in the location.
  If your data contains repeating nodes than you may target a specific node index to pull data from a specific node by specifying the node index (starting at zero) in parentheses. For example the following mapping will pull the Email data element from the 2nd Applicant node in the XML:
  [!XML://Applicants/Applicant(1)/Email]

- Version 1.8. Lookup fields can be populated with record Ids of related objects by specifying a LOOKUP function that must be structured, for example, **[!LOOKUP://<object-name>?<field-name>=<field-value>[&<field-name>=<field-value>]**

  For example, the following lookup function will search for a Contact record with matching first name and last name and insert the resulting record Id into the Contact__c lookup field: **"Contact__c":"[!LOOKUP://Contact?FirstName=[!XML://Contact/FirstName]&LastName=[!XML://Contact/LastName]]"**

  The value will be left empty if no matching record is found. If multiple records are found, an exception will result.

The relations Directive (Optional)

The relations directive is optional and allows you to specify object relationships to other objects defined in the configuration.

```
"relations": {
    "Account": "parent-account",
    "Contact": "my-contact"
}
```

For each relationship specified, the key value should match the name of a relationship against the object and the reference value should match an object label of the correct type specified in the configuration.

XML Consumption

The example configuration above will consume the following XML submission data structure to create the new Case object and make updates to the contact information for the Contact:

```
<?xml version="1.0" encoding="UTF-8"?>
<AvokaSmartForm>
    <Account>
        <AccountId>00128000002gH6DAAU</AccountId>
        <AccountName>Big Top Enterprises</AccountName>
        <Industry>Entertainment</Industry>
        <Website>www.bigtop.com</Website>
        <AccountRep>Ben Warner</AccountRep>
    </Account>
    <Contact>
        <ContactId>00328000001tCzaAAE</ContactId>
        <Title>Mr.</Title>
        <FirstName>Jolly</FirstName>
        <LastName>Roger</LastName>
        <Position>CEO</Position>
        <Department>Management</Department>
        <Phone>+61 2 9955 6600</Phone>
        <Mobile>0411 803 240</Mobile>
        <Email>jolly@bigtop.com</Email>
    </Contact>
    <IssueDetails>
        <IssueType>Problem</IssueType>
        <Reason>Instructions not clear</Reason>
        <Summary>Can't find where screw C3 goes</Summary>
        <Description>It is holding together but for how long?</Description>
    </IssueDetails>
</AvokaSmartForm>
```

Custom Delivery Processors

Where the default delivery processor does not provide the level of control required, you may define your own delivery processor by developing an APEX class that implements the ITransactDeliveryProcessor interface. To utilize your custom delivery processor, simply specify the class name in the relevant Delivery Processor record and any configuration required by the class.

```
global interface ITransactDeliveryProcessor {

    /**
     * This function processes the delivery record based on the config provided.
     *
     * @param delivery The Transact Delivery object to be processed
     * @param processorConfig The string containing the processing config
     */
    void processDelivery(Transact_Delivery__c delivery, String config);
}
```

The following custom delivery processor class is configured to send an email to an inbox when the submission is delivered to Salesforce.

```
global class CustomDeliveryProcessor implements avoka.ITransactDeliveryProcessor {

    public void processDelivery(avoka__Transact_Delivery__c delivery, String pro-
cessorConfig){
        System.debug('Custom Delivery Processor: ' + delivery.Name);

        Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
        String[] toAddresses = new String[] {'salesforce@avoka.com'};
        mail.setToAddresses(toAddresses);
        mail.setSubject('Custom Delivery Processor: ' + delivery.Name);
        mail.setPlainTextBody('Processing complete: ' + delivery.Name);
        Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
    }
}
```

To configure your custom delivery processor to be used you must enter the class name in the Delivery Processor object with the appropriate form code. The default value is 'TransactDeliveryProcessor' - you can modify this to be your own class name and it will be used instead of the default. You can also specify your own configuration data if required - your processor class will receive this configuration in the 'config' string parameter in the processDelivery () method signature.

If you have any trouble getting your custom delivery processor class to be used, use the **`global`** keyword in the class declaration to ensure it is visible to the package.

Next, learn how to configure Manager to support the Transact for Salesforce App.

Restrict Access for Selected Forms to User Groups

Journey Platform | All Personas | 5.1 & Higher

You can configure access restrictions for selected forms to specific user groups. By default, access to forms is unrestricted within the security context of the form. To restrict access for select form to certain user groups we need to first create a form group and then allocate users and forms into that group.

Create the Form Group

1. Log onto Manager.
2. Select **Security > Groups**.
3. Click **New**.
4. Enter the name that you'd like to call the group, relevant to the type of users who will be members of the group.
5. Select **Form** in the **Type** dropdown list.
6. Enter a description and select the required group user access control settings.
7. Its recommended to enable all settings unless there are specific reasons not to.
8. Click **Save** to update the changes.

Add Users to the Form Group

Users will not have access to forms in form group until they are added to the form group.

1. Log onto Manager.
2. Select **Security > Groups**.
3. Select the group you wish to manage and click the **Members** tab and add users to the group as required.
4. Click **Save** to update the changes.

Assign Forms to the Form Group

1. Log onto Manager.
2. Select **Forms > Forms**.
3. Select the form you wish to manage.
4. Click the **Group Access** tab and assign the form into the available groups as required.
5. Click **Save** to update the changes.

Next, learn how to retrieve the consumer key and secret for the Salesforce app.

Enable Access to Restricted Forms

Journey Platform | All Personas | 5.1 & Higher

Salesforce administrators can enable access to restricted forms to certain user groups within Salesforce if the restricted form groups have already been established in Manager.

To enable access to restricted form groups, a Salesforce Public Group must be created with the identical name to the Form Group in Manager.

1. Under **Administration Setup**, select **Manage Users**, and click **Public Groups**.
2. Click **New**.
3. Enter the label and name of the group, ensuring that the Label is identical to associated Form Group name in Manager.
4. Configure the membership to the group.
5. Click **Save** to update the changes.

Now, users who are members of the public group will automatically gain access to the restricted forms the next time they load the Canvas app.

Next, learn how to enable access to Transact for Salesforce App

Receive Administrator Alerts when Delivery Processing Fails

Journey Platform | All Personas | 5.1 & Higher

You can activate administrator alerts to one or more users, when a delivery processing failure occurs.

When delivery processing for a particular transaction fails, it will be identified in the Delivery Log with a status of 'Failed' and failure reason will be available in the Processing Message field.

Delivery processing failure alerts are sent out to anyone who has the **Transact Administrator** permission set on their account. To add this permission set to a user account:

1. Select **Administration Setup > Manage Users > Users.**
2. Locate and open the user account.
3. Select **Permission Set Assignments** and click **Edit Assignments**.
4. Move the **Transact Administrator** permission set from the **Available** to the **Enabled** column.
5. Click **Save** to update the changes.

Next, learn how to remove password expiry on a Salesforce user account.

Transact for Salesforce App Troubleshooting

Journey Platform | All Personas | 5.1 & Higher

If the integration between Manager and Salesforce is not working as expected, you can start troubleshooting by reviewing the error log, which can quickly reveal the cause of the issue.

The list of the common problems - solutions is listed below.

Content cannot be displayed: You do not have sufficient privileges to access the page: /apex/Avoka_Transact_Lead

Problem

Some of your users are getting a message displayed when they try to access the Transact for Salesforce App visualforce page in a page layout:

> Content cannot be displayed: You do not have sufficient privileges to access the page: /apex/Avoka_Transact_Lead

This issue may occur if you have enabled security around the visualforce page that does not include the profile of the user trying to access the page.

Solution

To resolve this issue you must enable page access to the users profile:

1. Select **Setup > Develop** and then select the Visualforce Pages.
2. Click the **Security** link next to the target page.
3. Add the profiles that require access to the page.
4. Click **Save** to update the changes.

Reference

https://help.salesforce.com/apex/HTViewHelpDoc?id=pages_security_page_def.htm&language=en_US

Could not identify SObject type <X>

Problem

When processing a delivery an exception is thrown with error message:

Could not identify SObject type <X>

where <X> is an object type specified in your delivery processor configuration.

Solution

This issue indicates that there is a problem with the delivery processor configuration. You need to check that you have specified all the SObject types correctly in your configuration.

Delivery Error: No such column 'avoka__Delivery_Trigger_Type__c' on sobject of type avoka__Transact_Delivery__c

Problem

The Manager delivery process fails with error message:

```
SalesForceException[response=[ { "message" : "No such column 'avoka__Delivery_Trigger_Type__c' on sobject of type avoka__Transact_Delivery__c", "errorCode" : "INVALID_FIELD" } ], operation=perform
```

This issue relates to insufficient permissions on the Salesforce side and is caused by missing permissions for Manager administrator permission set prior to version 1.10 of the in the Transact for Salesforce App.

The Manager administrator permissions should have full permissions on the Delivery Log, Delivery Processing, Post Log, and Prefill Generation objects . In Transact for Salesforce App versions prior to 1.10, you will notice a number of missing permissions when you view the object settings of this packaged permission set, as shown below.

Field Permissions

Field Name	Read	Edit
Affected Sobject Id	☑	☑
Attachment Count	☑	☐
Attachment Metadata	☐	☐
Case Number	☑	☑
Created By	☑	☐
Data Extracts	☐	☐
Delivery Trigger Type	☐	☐
Form Code	☑	☑
Form Name	☐	☐
Form Version	☐	☐
Job Reference Code	☐	☐
Last Modified By	☑	☐
Owner	☑	☑
Payment Amount	☑	☐
Payment Gateway Receipt Number	☑	☐
Payment Gateway Timestamp	☑	☐
Payment Gateway Transaction Number	☑	☐
Portal Name	☐	☐
Processed Timestamp	☑	☐
Processing Message	☑	☑
Processing Status	☑	☐
Submission Data	☑	☑
Submission Key	☑	☑
Submission Timestamp	☑	☐
Sync Status	☐	☐
Tracking Code	☑	☑
Transaction Score	☑	☑
Transaction Status	☐	☐
User Name	☑	☑

Solution

This issue is only applicable to versions of the Transact for Salesforce App prior to 1.10. The permission set in version 1.10 has the missing permissions added.

As the Manager administrator permission set is part of a managed package, it cannot be modified in your org. To resolve this issue, you will need to create a new permission set that has all permissions enabled for the 4 objects.

1. Under Administer, click Manage Users > Permission Sets
2. Under Permission Sets, click New and enter the label of the new permission set
   - Label - Transact Administrator 1
   - Description - Used to activate missing permissions in the Manager administrator permission set
   - User License - blank
3. Click **Save** to update the changes.
4. Edit the Object Settings of the permission set to activate all object and field permissions for the 4 Avoka objects:
   - Delivery Log
   - Delivery Processing
   - Prefill Generation
   - Post Log
5. Ensure all read and edit permissions are granted to these objects.
6. Click **Save** to update the changes.
7. Allocate this new permission set to the Manager user.

Delivery Error: The requested resource does not exist

Problem

The Manager delivery process in fails with error message:

```
SalesForceException[response=[ { "errorCode" : "NOT_FOUND", "message" : "The requested resource does not exist" } ]
```

Solution

This error indicates that the delivery process cannot find the delivery target objects in Salesforce. To resolve this issue:

1. Ensure that the app is installed in Salesforce.
2. If the license is limited to a number of seats, then the API user must be licensed for the app in order to access the Manager delivery objects.

Delivery Processing Error: Field <X> on <Y> is not CREATEABLE by <Z>

Problem

When processing a delivery an error is reported with message:

Delivery Processing Error: Field <X> on <Y> is not CREATEABLE by <Z>

X is the target object name, Y is the field being updated and Z is the user name. The same error may also appear with the key word UPDATEABLE.

Solution

This is a permission related problem and relates to the security around field updates on the object. Ensure the active user specified in the error message has the appropriate permissions to the target field.

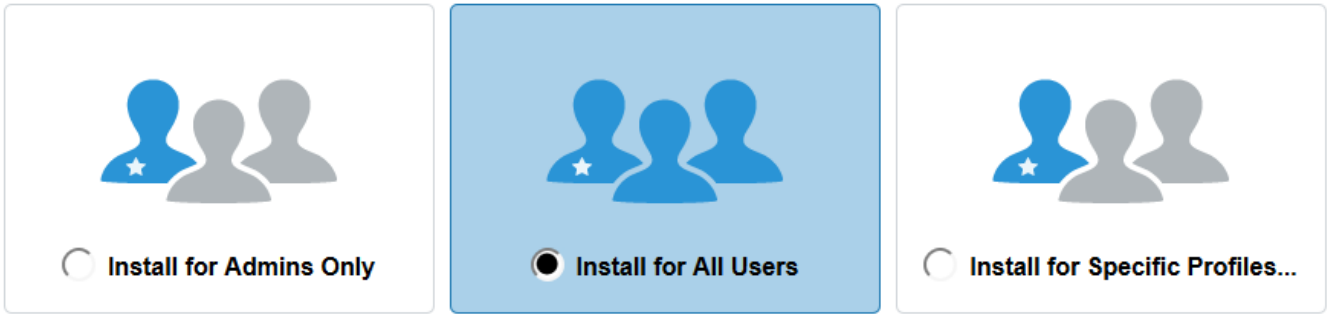
Insufficient Privileges: You do not have the level of access necessary to perform the operation you requested...

Problem

While trying to access the Transact for Salesforce App Visualforce page component some users are denied access with the following message:

Insufficient Privileges
You do not have the level of access necessary to perform the operation you requested. Please contact the owner of the record or your administrator if access is necessary. For more information, see Insufficient Privileges Errors.

This problem may be encountered if the package was installed for access by Admins Only or Selected Profiles instead of for All Users.



Solution

To resolve this issue, add the Manager user permission set to all required user accounts:

1. In the Setup menu search for Permission Sets and select the menu item to open the Permission Sets page.
2. Find the Journey Manager User permission set and open it, then select the **Manage Assignments** button.
3. Select **Add Assignments** and assign all required users to the permission set.

Missing "Reset My Security Token" Option

Problem

After creating the Manager user in Salesforce, no security token was emailed out and when logged in as that user the option to **Reset My Security Token** is not available as described here.

Solution

If you do not see the option to reset your token, you may be using IP Login Restrictions in Salesforce. For more information on why this menu option may not be available to your user, see the Salesforce documentation.

If you have further questions about IP Login Restrictions, contact Salesforce support for further assistance.

Post Request: 403 Forbidden - You don't have permission to access /manager/secure/GroovyServiceServlet on this server

Problem

When attempting to push a task from Salesforce to Manager using the post request object in Salesforce you may receive an error indicating that the HTTP request failed:

```
403 Forbidden - You don't have permission to access /manager/secure/GroovyServiceServlet on this server.
```

This may be due to IP white-listing on the Manager server rejecting access from the Salesforce server.

Solution

Add the Salesforce IP ranges to the Manager IP while-list as described here.

Salesforce API Authentication Failure (Invalid Grant)

Problem

Calls to Salesforce from Manager are failing with an authentication error:

```
SalesForceException[response={"error_description":"authentication failure","error":"invalid_grant"}
```

If you are receiving delivery error when trying to deliver to Salesforce, check the error log for the error message above.

Solution

This issue can be caused due to one of the following reasons:

- Invalid or incorrect credentials being used in the Salesforce Connection parameters
- Insufficient privileges assigned to the API user in Salesforce

To resolve this issue, try the following steps:

1. Re-enter the username, password and security token for the Salesforce Service Connection.
2. Ensure that the Salesforce API user has the following required permissions:
   - The user must have API access.
   - The user must have permission to write to the Manager custom objects.
   - Ensure the user is associated with a profile that is Admin approved for use of the Manager connected app.
3. If you have IP Restrictions in place in your Salesforce environment, double check that the IP ranges cover the Manager environment you are receiving the error on. You can check with the Manager cloud hosting team to verify the IP ranges for your environments.

Sorry, you are not licensed to use the Transact solution

Problem

Trying to access the Transact for Salesforce App Visualforce page component and getting a message saying:

> Sorry, you are not licensed to use the Transact solution. Please consult your Salesforce Administrator.

Solution

Your user is not licensed for use of the package, a Salesforce administrator needs to add you as a licensed user.

Next, learn about Transact for Salesforce App.

Transact for Salesforce App Installation

Journey Platform | All Personas | 5.1 & Higher

For any new deployment of Transact for Salesforce App, you should use the latest available version compatible with your Manager instance, as shown in the compatibility matrix below.

Transact for Salesforce App Version	Release Date	Improvements	Manager Version
1.11	August 24, 2017	Resolved issue with encoding of Post Request data	4.3
1.10	March 7th, 2016	Point release to accommodate customer driven features and bug fixes	4.3
1.8	December 16th, 2015	Point release to accommodate customer driven features and bug fixes	4.3
1.6	December 2nd, 2015	Minor patch release to improve permissions around Apex class access. No functional changes are included.	4.3
1.5	November 28th, 2015	Point release to provide client requested features and bug fixes.	4.3
1.4	September 23rd, 2015	This is the original AppExchange version containing baseline features.	4.2

The latest Salesforce portal war files are listed below.

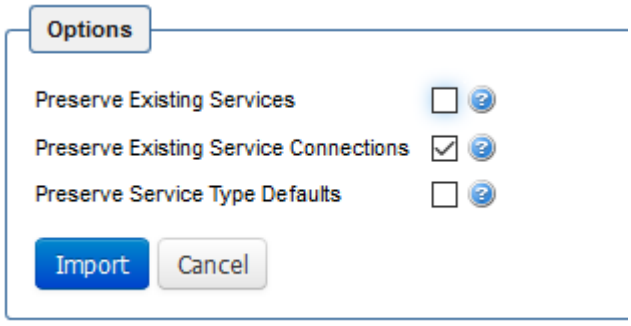
Manager Version	Portal WAR File
4.3.4+	Manager version 4.3.4 and higher has the Salesforce form space war file built into the Journey Manager installer. avoka-sf-portal-salesforce433.war
4.3.3	Journey Manager version 4.3.3 is EOL, so you must upgrade to newest Journey Manager, which includes the Salesforce portal. avoka-sf-portal-salesforce432.war
4.3.2	

> Journey Manager version 4.3.2 is EOL, so you must upgrade to newest Journey Manager, which includes the Salesforce portal.

The Salesforce app can be installed into the Salesforce Organization by clicking this link and following the prompts.

Service Definition Import Options

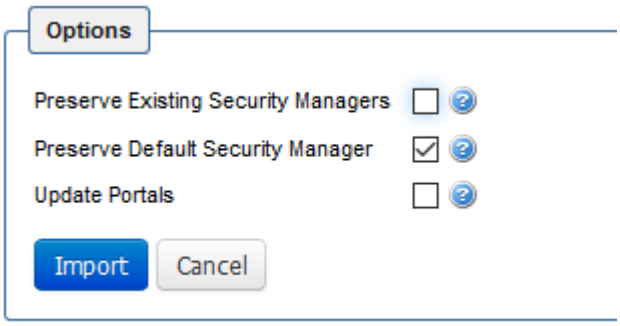
When importing service definitions deselect the **Preserve Existing Services** option in the import wizard while leaving the **Preserve Existing Service Connections** option selected:



Once you have imported new versions of the services you are required to manually activate these service versions and where appropriate, update the configurations that refer to the old version of the service to point to the new version.

Security Manager Import Options

When importing the security manager, deselect the **Preserve Existing Security Managers** option, as shown below.



Importing the security manager in this way will overwrite the parameters you have specified against this security manager, so make a backup copy of them first. In particular, the Client Id and Client Secret properties should be backed up and re-set after import.

Once you have imported new versions of the services you are required to manually activate these service versions and where appropriate, update the configurations that refer to the old version of the service to point to the new version.

Next, learn about Transact for Salesforce App troubleshooting.

Installation Assets Version 1.4

Journey Platform | All Personas | 5.1 & Higher

The Salesforce app version 1.4 is the original AppExchange version containing baseline features.

Release date is September 23rd, 2015.

It requires Manager version 4.2 and higher.

> This release version has been superseded. Please install the latest version of the App.

New Features

None

Issues Resolved

This is the original AppExchange version containing baseline features.

Installation Assets

The Salesforce App can be installed into the Salesforce Organization by clicking this link and following the prompts:

- Production: https://login.salesforce.com/packaging/installPackage.apexp?p0=04t900000002ZZ8

Next, learn how to install Transact for Salesforce App.

Installation Assets Version 1.5

Journey Platform | All Personas | 5.1 & Higher

The Salesforce app version 1.5 is a point release to provide client requested features and bug fixes.

Release date is November 28th, 2015.

It requires Manager version 4.3 and higher.

New Features

- SAE-085 Match Fields - You can now optionally specify the set of unique fields for an object type (e.g. LastName and Email for a Lead) and the delivery processor will find the existing record for update rather than insert. This feature was added to assist in the management of duplicate records.
- SAE-151 Administrator Alerts - Anyone with the **Transact Administrator** permission set in Salesforce will receive email notifications when delivery processing fails.
- SAE-055 Form Category Filter - In the visual force page to include the Canvas UI you can now optionally specify which categories you want to appear on the **New Form** page. This can be different for each instance of the Canvas. If not specified, all categories will be presented.
- SAE-146 Abandoned Forms – Abandoned forms will now appear in the search results (requires Transaction Manager 4.3). When a form becomes abandoned, customers cannot access it anymore. We've added a Reactivate link that is presented for abandoned forms where the data has not yet been purged. The reactivate link will re-open the form for access.
- SAE-147 Enhanced Customer Link - Provided the ability to copy a portal specific URL for an active form by selecting the portal from a drop down. This is to cater for the situation where customers interact with a different portal to staff.
- SAE-152 Reload page button - On all UI pages other than Search, the reload button will refresh the content of the page.

Issues Resolved

- SAE-135 Authentication Provider Issues - Resolved issues where existing users were not getting added to the current organization and portal when they re-authenticate under certain conditions.
- SAE-148 Incorrect Created Time - Resolved issue where the created time of a form was showing incorrect values under certain conditions.

Installation Assets

The Salesforce App can be installed into the Salesforce Organization by clicking this link and following the prompts:

- Production: https://login.salesforce.com/packaging/installPackage.apexp?p0=04t900000002ZjX

The following installation archives should be used to setup Manager:

- Service Definitions: AT4SF-Services-1.5.zip
- Security Manager: AT4SF-SecurityManager-1.5.zip
- Canvas UI Form Version: AT4SF-Canvas-1.5.zip
- Test Form Version: AT4SF-TestForm-1.0.zip

Next, learn how to install Transact for Salesforce App.

Installation Assets Version 1.6

Journey Platform | All Personas | 5.1 & Higher

The Salesforce app version 1.6 is a minor patch release to improve permissions around Apex class access. No functional changes are included.

Release date is December 2nd, 2015.

It requires Manager version 4.3 and higher.

New Features

None

Issues Resolved

- Improves permissions around Apex class access.

Installation Assets

The Salesforce App can be installed into the Salesforce Organization by clicking this link and following the prompts:

- Production: https://login.salesforce.com/packaging/installPackage.apexp?p0=04t90000000Y1N0

No change to the Manager installation assets since version 1.5.

Next, learn how to install Transact for Salesforce App.

Installation Assets Version 1.8

Journey Platform | All Personas | 5.1 & Higher

The Salesforce app version 1.8 resolves issues to accommodate customer driven features and bug fixes.

Release date is December 16th, 2015.

It requires Manager version 4.3 and higher.

New Features

- SAE-166 Lookup Fields - Add support for the mapping of form data to lookup field references.

Issues Resolved

- SAE-161 Relationship between the transaction and the current object not created where no prefill service is specified for the form
- SAE-163 Exception thrown if Delivery Processor not configured for specified form code
- SAE-164 Null Pointer if SObjectType not found
- SAE-167 Exception in delivery mapping to Boolean, Decimal or Date fields

Installation Assets

The Salesforce App can be installed into the Salesforce Organization by clicking this link and following the prompts:

- Production: https://login.salesforce.com/packaging/installPackage.apexp?p0=04t90000000Y1SP

The following installation archives should be used to setup Manager:

- Service Definitions: AT4SF-Services-1.8.zip
- Security Manager: AT4SF-SecurityManager-1.5.zip
- Canvas UI Form Version: AT4SF-Canvas-1.5.zip
- Test Form Version: AT4SF-TestForm-1.0.zip

Next, learn how to install Transact for Salesforce App.

Installation Assets Version 1.10

Journey Platform | All Personas | 5.1 & Higher

The Salesforce app version 1.10 resolves issues to accommodate customer driven features and bug fixes.

Release date is March 7th, 2016.

It requires Manager version 4.3 and higher.

New Features

- SAE-173 - Support for date time fields in delivery mapping
- SAE-175 - New Preprocessor service option
- SAE-176 - New Completed Processor service option

Issues Resolved

- SAE-168 - Exception if relationship parent is not found in delivery processing
- SAE-169 - Auto role assignment not working in Auth Provider due to error code
- SAE-174 - Manual Delivery Processing ignores trigger type (always executes Completed Transaction processing)

Installation Assets

The Salesforce App can be installed into the Salesforce Organization by clicking this link and following the prompts:

- Production: https://login.salesforce.com/packaging/installPackage.apexp?p0=04t90000000Y1cA
- Sandbox: https://test.salesforce.com/packaging/installPackage.apexp?p0=04t90000000Y1cA

The following installation archives should be used to setup Manager:

- Service Definitions: AT4SF-Services-1.10.zip
- New service AT4SF-PushToSalesforce - performs submission delivery to Salesforce and is used as a sub process by other services
- New service AT4SF-CompletedProcessor - provides a service option for delivery to Salesforce on the Submission Completed event
- New service AT4SF-Preprocessor - provides a service option for delivery to Salesforce on the Submission Preprocessor event
- Updated service AT4SF-DeliveryPush (v3) - now offloads delivery process to AT4SF-PushToSalesforce
- Updated service AT4SF-SaveProcessor (v2) - now offloads delivery process to AT4SF-PushToSalesforce
- Security Manager: AT4SF-SecurityManager-1.10.zip
- Canvas UI Form Version: AT4SF-Canvas-1.5.zip
- Test Form Version: AT4SF-TestForm-1.0.zip

Next, learn how to install Transact for Salesforce App.

Installation Assets Version 1.11

Journey Platform | All Personas | 5.1 & Higher

The Salesforce app version 1.11 resolves issue with encoding of Post Request data.

Release date is August 24, 2017.

It requires Manager version 4.3 and higher.

New Features

None.

Issues Resolved

- SAE-179 Post log fails if data contains '&' symbol - When using the Post Log with prefill data from Salesforce objects that contains special symbols like '&' (e.g. "Walker & Sons") the post request content is corrupted.

Installation Assets

The Salesforce App can be installed into the Salesforce Organization by clicking this link and following the prompts:

- Production: https://login.salesforce.com/packaging/installPackage.apexp?p0=04t90000000JDR9
- Sandbox: https://test.salesforce.com/packaging/installPackage.apexp?p0=04t90000000JDR9

The following installation archives should be used to setup Transaction Manager:

- Service Definitions: AT4SF-Services-1.10.zip
- Security Manager: AT4SF-SecurityManager-1.11.zip
- Canvas UI Form Version: AT4SF-Canvas-1.5.zip
- Test Form Version: AT4SF-TestForm-1.0.zip

Next, learn how to install Transact for Salesforce App.

Transact Integration Gateway Overview

Journey Platform | System Manager / DevOps | v4.3 & Higher

Although Transact Integration Gateway (TIG) continues to receive support, it is recommended that Journey Platform solution builders move away from using TIG and instead use Journey Manager REST APIs to build Journey Platform solutions. Continuing to use TIG may complicate an upgrade path when TIG is eventually deprecated in a future release.

Overview

Transact Integration Gateway (TIG) is a lightweight gateway for performing on premise business integration. Customers using a cloud environment often have integration requirements for:

- Dynamic data lookups or data validations to internal systems.
- Straight through processing, when customers submit applications.
- Background transaction delivery integration in internal systems.
- Uploading form prefill reference data to the cloud servers.

Using the Transact Integration Gateway, customers can implement integrations like these quickly without additional software licenses.

If you already have integration gateway capabilities with solutions such as IBM Power Gateway or Mulesoft API Gateway, you may choose to use these. However, if you don't need the complexity of these solutions, the simpler Transact Integration Gateway may be suitable, and is an excellent option for performing delivery integration.

The Journey SDK for Java provides a Java API for Manager services, so you can build applications that integrate directly with them. Journey SDK documentation is available online as well as being included in the Journey SDK download package (authentication required) available from our resources web site. (authentication required).

TIG is largely unchanged since TIG v4.3, so it is compatible with current Transact and Journey platform releases.

Usage and Deployment Models

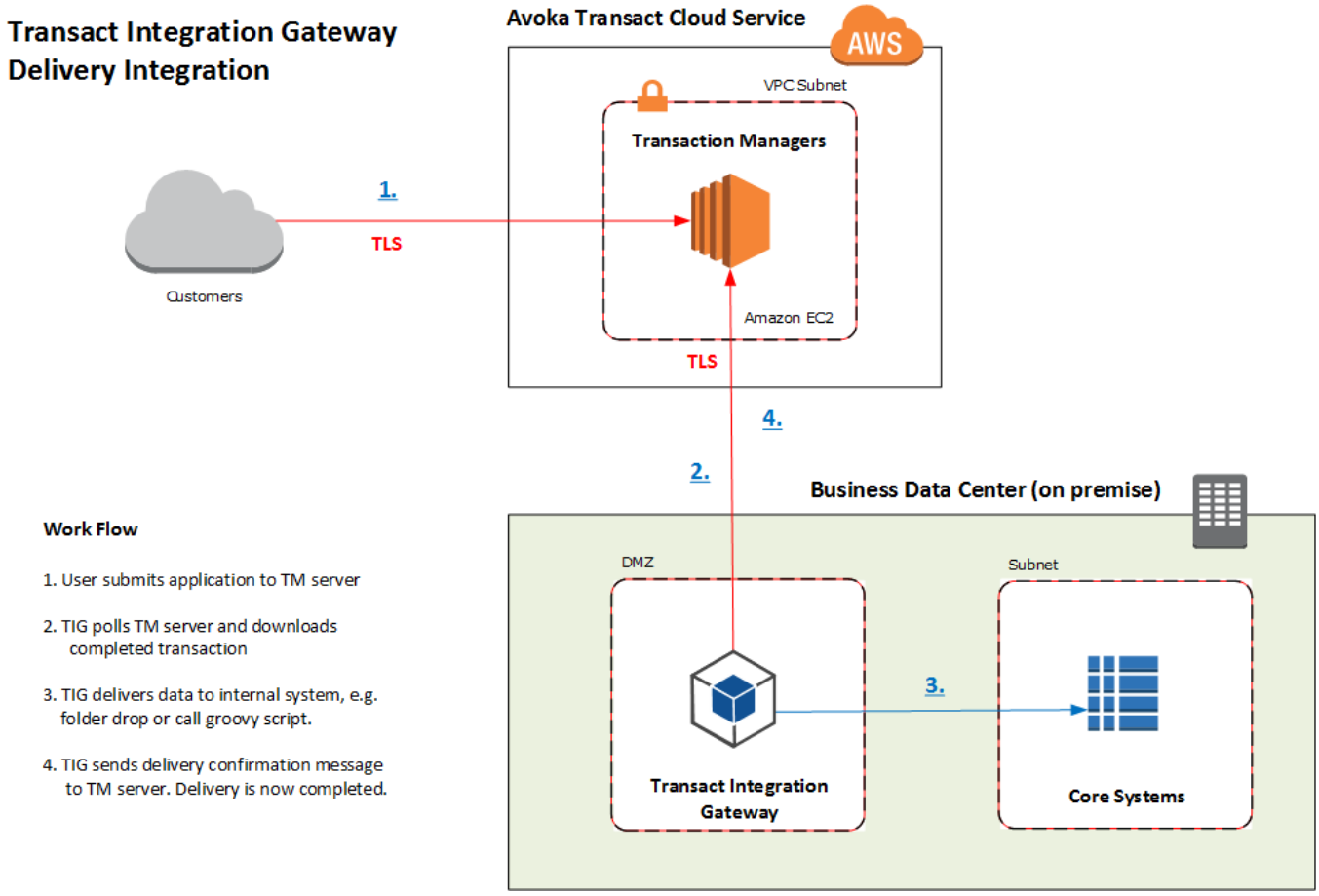
The deployment model for TIG is to install the application on premise inside a DMZ or a secure location. The application is designed to be very easy to install and configure. You can literally install the application and perform transaction data delivery within minutes.

For High Availability (HA) deployment models you would typically install the two TIG applications on separate servers in different availability zones. TIG uses a stateless architecture, with configuration maintained in local JSON[1] configuration files. There is no requirement for a shared database to support HA usage.

Delivery Integration

A typical use of TIG is to perform transaction delivery integration. In this deployment model, TIG is used to pull down completed transactions from the Cloud Hosted Journey Manager servers, and deliver this data into internal systems.

This usage model is illustrated in the diagram below.



For straight delivery integration, there is no requirement for TIG to be externally accessible. All network calls are outgoing from TIG to Manager servers. To complete transaction delivery, TIG needs to be able to write transaction data files to a secure and high availability file store such as a SAN, or it should be able to call other internal systems to complete delivery.

---

[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.

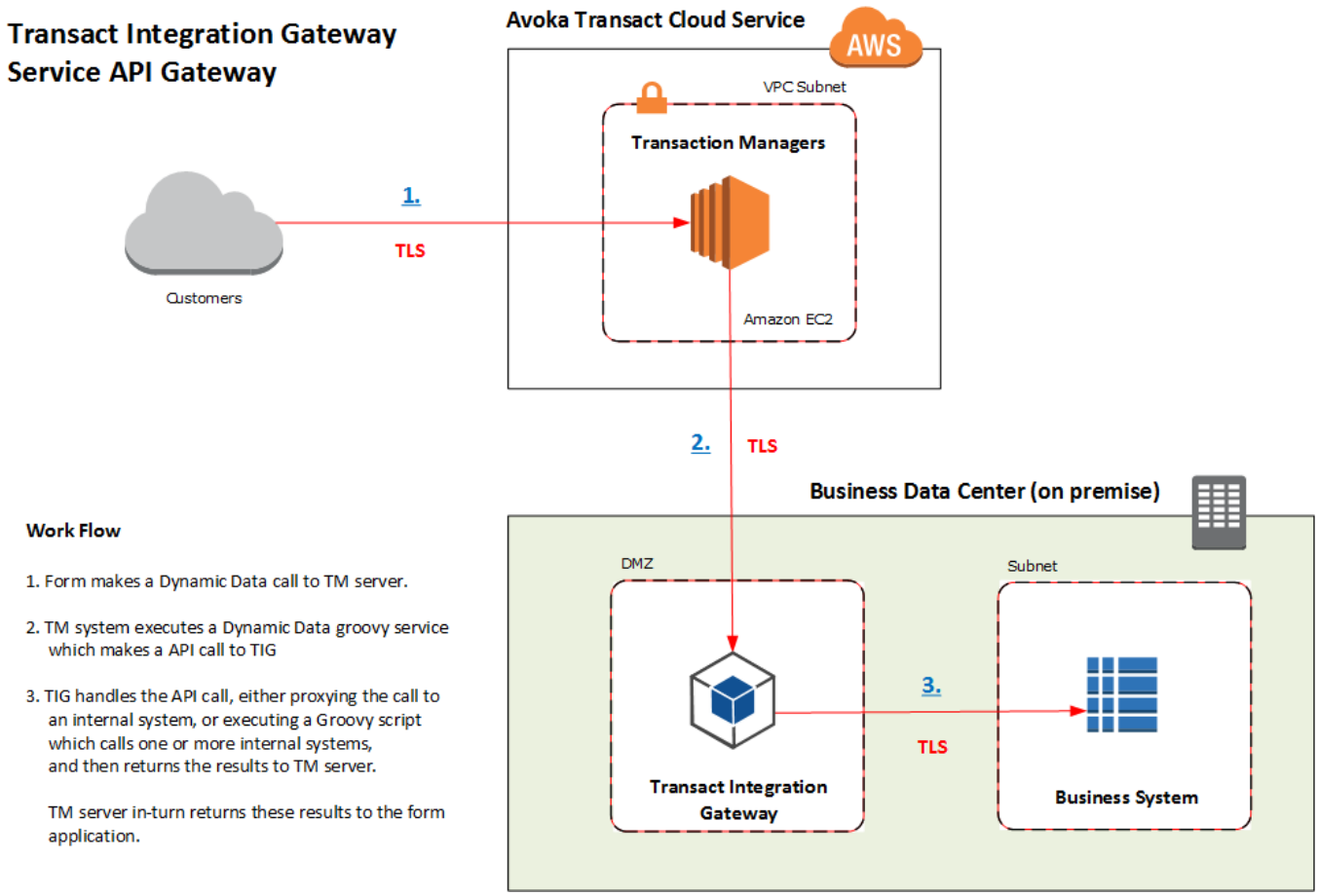
When internal transaction delivery has been completed, TIG will send a positive delivery confirmation message to the Manager server. Once this has been done, the transaction data on the server can either be deleted immediately or made available for scheduled data purging.

Service API Gateway

TIG can be used as a Service API Gateway to enable Cloud Manager servers to make real-time (synchronous) API calls to internal business systems. These calls are typically to perform operations like:

- perform a security check before the user opens a saved form
- prefill a form with the customer's profile data
- make a form dynamic data call to perform data verification, such as Identity Verification (IDV)
- straight through processing, when a customer submits an application resulting in an account being opened and the details returned to the user

This usage model is illustrated in the diagram below.



Operating as a Service API Gateway, TIG supports two modes of operation:

- **Service API Proxy** - TIG acts as a proxy routing the external call to an internal system
- **Groovy Service Execution** - TIG calls a **Groovy**[1] script to perform one or more calls to internal systems

By using TIG as a Service API Gateway, access to internal systems can be strictly controlled and no internal network information is exposed externally.

Gateway Security

To provide security access control policies to protect internal Service APIs, we recommend a combination of the following security controls:

- Transport Layer Security (TLS) protection for all network calls
- IP address white list with TIG only accepting calls from configured Manager servers (configured on switch and TIG)
- BASIC authentication over TLS, with TIG only allowing authenticated requests
- Virtual Private Network (VPN) between business data center and AWS Virtual Private Cloud (VPC)
- Mutual SSL authentication between Manager calls and TIG

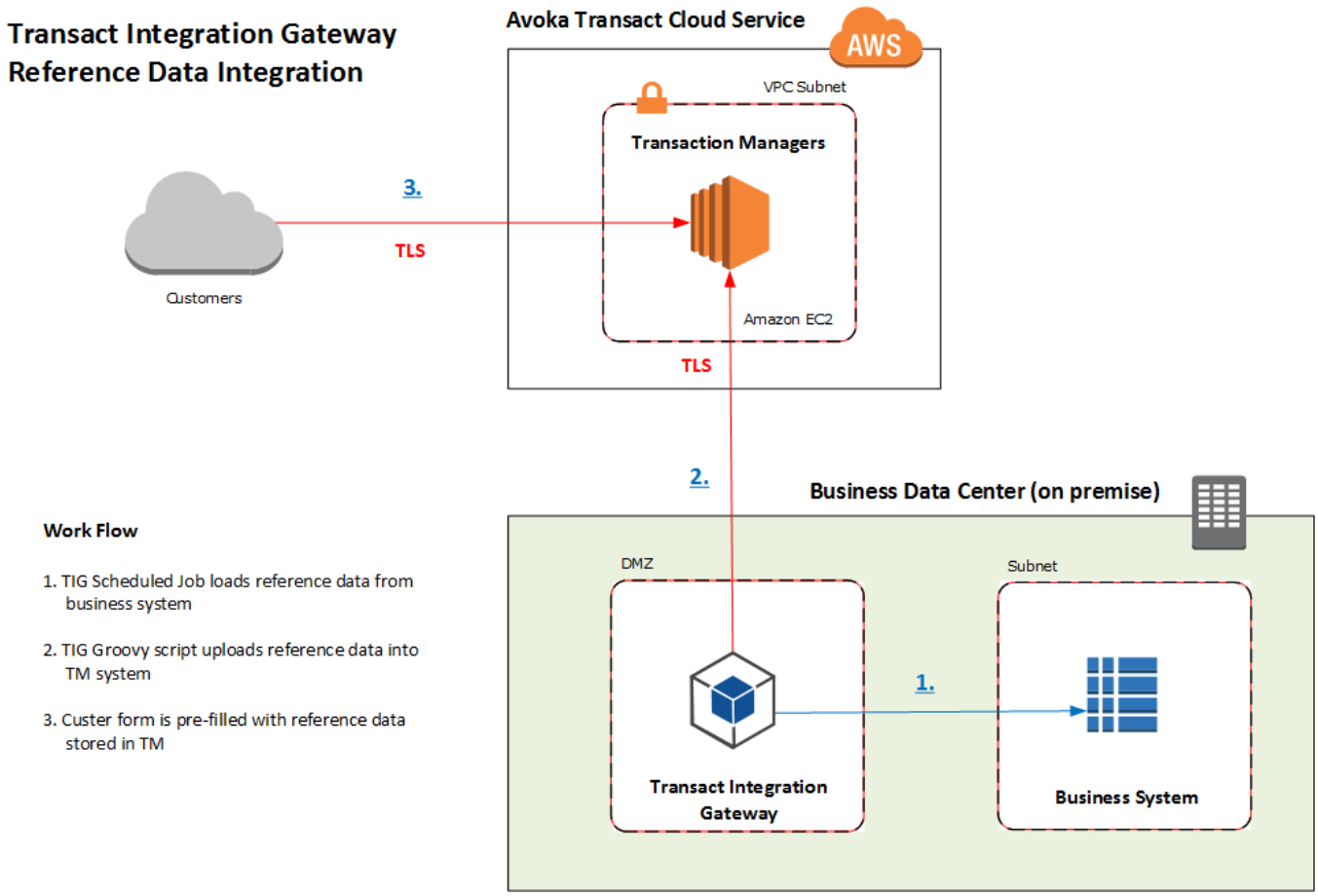
TIG utilizes Apache for TLS termination and uses ModSecurity as an Web Application Firewall (WAF) to provided additional protection to the Service API Gateway.

Reference Data Integration

Another use for TIG is to upload reference data from business systems into Cloud Hosted Manager servers where it can be used for form prefill or form dynamic data.

This usage model is illustrated in the diagram 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.



For uploading reference data, there is no requirement for TIG to be externally accessible. All network calls outgoing from TIG are to Manager servers.

TIG provides a Quartz Job scheduler facility which is used for performing background jobs. The job scheduler supports sophisticated **CRON**[1] expressions and simple interval based scheduling to invoke **Groovy**[2] scripts to perform integration tasks.

Next, learn about installing and configuring Transact Integration Gateway.

---

[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.

[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.

Transact Integration Gateway Setup

Jburney Platform | System Manager / DevOps | v4.3 & Higher

Although Transact Integration Gateway (TIG) continues to receive support, it is recommended that Jburney Platform solution builders move away from using TIG and instead use Jburney Manager REST APIs to build Jburney Platform solutions. Continuing to use TIG may complicate an upgrade path when TIG is eventually deprecated in a future release.

Prerequisites

You need a JDK installed on the system where you are installing Transact Integration Gateway (TIG). The JDK version you need is dependent upon the version of TIG you are installing, and can be identified as follows.

1. Identify the TIG version (**`tig-version`**) that you want to install.
2. Next, identify the JDK version (**`jdk-version`**) that is required when installing the **`tig-version`** version of Jburney Manager.
3. So, **`jdk-version`** is the JDK version required when installing the TIG version identified in the first step.

An example will help to clarify these steps.

1. Say you want to install TIG v18.11 (**`tig-version`** = v18.11).
2. Identify the JDK version required when installing Jburney Manager v18.11; in this case, it's JDK 11 (**`jdk-version`** = JDK 11).
3. So, we need JDK 11 installed on our system in order to install TIG v18.11.

TIG and Jburney Manager have been designed so that any version of TIG can communicate with any Jburney Manager version. That is, you do not need to install the same version of TIG and Jburney Manager.

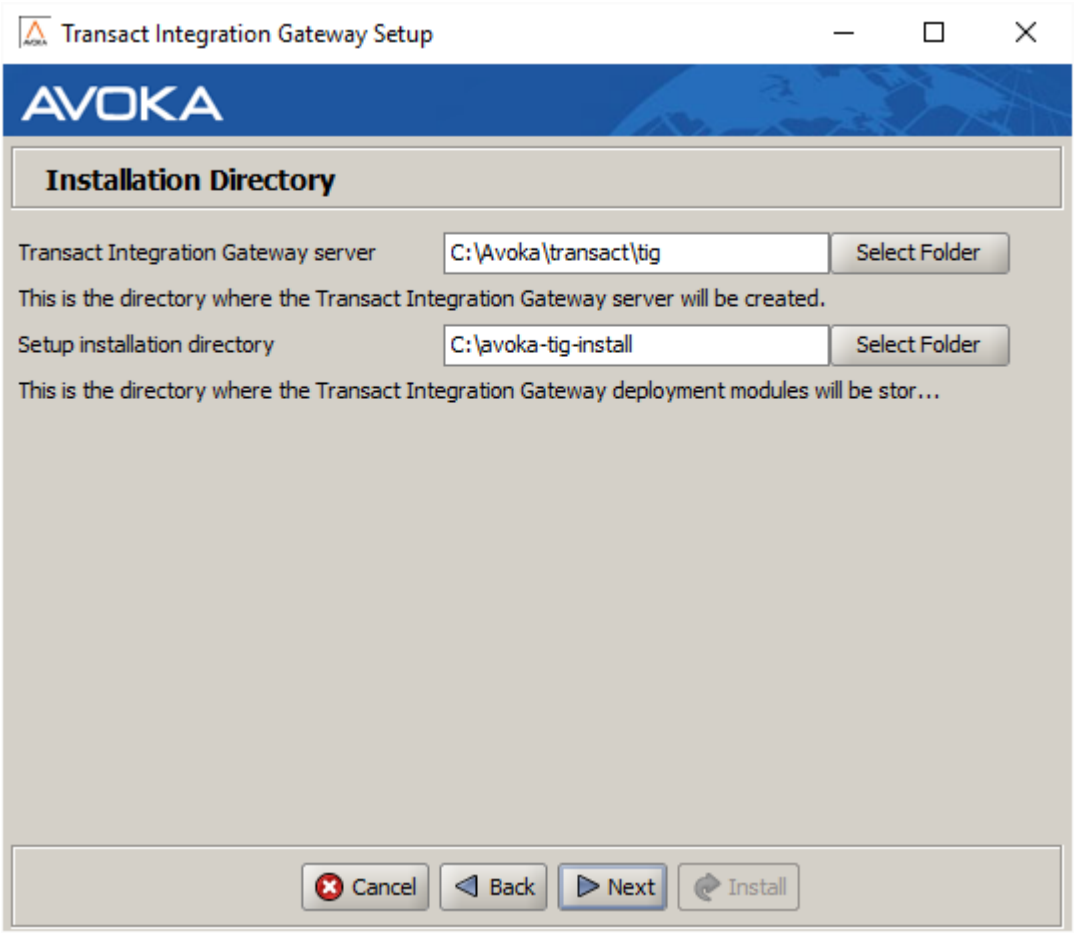
Transact Client SDK

If you're using TIG in your Jburney Platform solution, you may also want to use Transact Client SDK when building that solution. While Transact Client SDK is no longer under active development, it is compatible with later Jburney Platform versions, just like TIG. You can download the latest Transact Client SDK release which includes reference documentation that is also available online.

Installation

A TIG Setup Wizard application is provided to enable fast and easy installation.

On a Windows server with a JRE available, TIG can be installed and running as a Windows Service within a few minutes.



See the Transaction Integration Gateway Installation Guide for further details.

Configuration

TIG is configured through a series of JSON configuration files. These files are located in the configuration directory: **`C:\avoka\transact\tig\config\`**

Config File	Description
delivery-config.json	Transaction delivery channel configurations
proxy-config.json	Service API Gateway proxy configurations
services-config.json	Groovy Services configurations
scheduler-config.json	Scheduled Grooovy script job configurations
storage-config.json	Submission data storage configuration

The Groovy scripts directory is also under the configuration directory: **`C:\avoka\transact\tig\config\groovy\`**

This directory contains a number of sample Groovy scripts which are referenced in the **groovy-config.json** file including:

- **delivery-email.groovy**
- **delivery-example.groovy**
- **google-search.groovy**
- **job-example.groovy**
- **template-example.groovy**

Delivery Config

The delivery configuration file defines transaction delivery channels to retrieve completed form transactions using the Manager REST Delivery APIs.

The sample **delivery-config.json** installed with TIG is shown below.

```
{
	"pollingIntervalMins": 1,
	"deliveryChannels": [
		{
			"deliveryUrl": "http://localhost:9080/manager/secure/rest/delivery/v1/maguire/re
vice/",
			"user": "administrator",
			"password": "password",
			"dataDir": "C:/tig/delivery/",
			"subDir": "${submissionId}",
			"processingStatus": "Thank you your application is being processed",
			"serviceName": "delivery-example"
		}
	]
}
```

This configuration specifies a 1 minute polling interval to call the Manager REST Delivery API to see whether any new transactions are available.

The next element is a collection of delivery channels. A single TIG service can perform deliveries against many Manager Delivery Channels. For more information on how to configure a delivery channel, see the Transact Services Guide > REST API > Delivery API.

The JSON delivery channel attributes include:

Name	Description

deliveryUrl	Delivery channel URL as defined for the Manager organization delivery channel's REST Service URL
user	Manager user login name. Users must have the **REST Delivery** role assigned to them, be assigned to Manager console access, and organization access.
password	Manager user password It is recommended that you configure specific REST delivery users, with only the **REST Delivery** role and no other permissions.
dataDir	Base data directory where form transaction data will be written
subDir	A pattern sub directory under the base **dataDir** This folder is used to separate different transactions. There are a large number of patterns which can be used to create sub directories for data delivery (see below).
processingStatus	A processing status message which is returned to the Manager server when delivery is confirmed and may be displayed to the end user who submitted the form. This is typically use to provided a message informing them that their application is being processed. This message has a maximum length of 100 characters; longer values will be truncated.
deleteData	A flag informing the Manager server to delete the transaction data immediately when it receives the delivery confirmation message. Use this option only if you have delivered data into a high available storage system, and you have very strict data retention policies.
completedMarker	A flag to specify whether to create a zero length **completed.txt** file in the data delivery directory once all the transaction data has been delivered. This option is useful for folder watch deliver integration systems which will perform ELT operations when a marker signal file appears in a directory.
serviceName	The name of a Groovy script service to call when all the transaction data has been written into the delivery directory. Only delivery type services can be used for delivery integration.

The **subDir** pattern options include:

Pattern Variable	Description
clientCode	Form's organization client code identifier
formCode	Unique form code identifier
formVersion	Form version number
formStatus	Transaction form status: [**Completed** | **Abandoned**]
portal	Name of the portal or form space
submissionId	Transaction submission identifier
receiptNumber	Transaction receipt number
submitKey	Transaction submit key GUID
trackingCode	Transaction tracking code
transRefNumber	Transaction Trn Ref Number
yyyy	Year; for example, 2015
mm	Month of year; for example, 08
dd	Day of month; for example, 01

Note that TIG will monitor any changes to the delivery configuration file and make these available immediately. However if you change the polling internal configuration, you will need to restart the TIG service for this change to take effect.

Proxy Config

The proxy configuration file defines internal services which may be called via Gateway Proxy calls.

The sample **proxy-config.json** installed with TIG is shown below.

```
[
	{
		"route": "google-search",
		"serviceUrl": "https://ajax.googleapis.com/ajax/services/search/web",
		"connectionTimeout": 10000,
		"returnErrors": true,
		"parameters": [
			{ "name": "v", "value": "1.0" }
		]
	}
]
```

This configuration specifies a collection of proxy services routes. The proxy attributes include:

Name	Description
route	Proxy service route URL path
serviceUrl	Internal service API URL to call
headers	HTTP headers to add to the internal Service API call
parameters	HTTP request parameters to add to the internal Service API call
basicAuth	The HTTP Basic authentication header required to execute the call when performing a Service API Gateway call. The basic auth header is a Base64 encode username:password value.
connectionTimeout	The connection and socket transfer timeout in milliseconds for the internal Service API call. The default timeout is 10 seconds.
returnErrors	Return Groovy script errors to the caller when performing a Service API Gateway call. Use this option when developing scripts, but not in production. Default value is **`true`**.

An example of the proxy routing configuration is shown below:

1. URL request from Manager to TIG:

```
GET https://service.mycorp.com/tig/proxy/google-search/?q=Avoka
```

2. TIG extracts gateway route path:

```
address-lookup
```

3. TIG looks up proxy route and makes internal call using the **`serviceUrl`**, for example:

```
GET https://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=Avoka
```

Services Config

The services configuration file defines Groovy script services which may be invoked via Gateway Service calls, via Deliveries or via Scheduled Jobs.

The sample **`services-config.json`** installed with TIG is shown below.

```
[
	{
		"serviceName": "delivery-email",
		"serviceType": "delivery",
		"groovyScript": "$tig/config/groovy/delivery-email.groovy",
		"parameters": [
			{ "name": "emailUrl", "value": "https://outlook.office365.com/EWS/Exchange.asmx"
			{ "name": "emailUser", "value": "username@server.com" },
			{ "name": "emailPassword", "value": "password" }
		]
	},
	{
		"serviceName": "delivery-example",
		"serviceType": "delivery",
		"groovyScript": "$tig/config/groovy/delivery-example.groovy"
	},
	{
		"serviceName": "job-example",
		"serviceType": "scheduled",
		"groovyScript": "$tig/config/groovy/job-example.groovy"
	},
	{
		"serviceName": "google-search",
		"serviceType": "gateway",
		"groovyScript": "$tig/config/groovy/google-search.groovy",
		"groovyTypeChecked": false,
		"executionTimeout": 10000,
		"returnErrors": true,
		"parameters": [
			{ "name": "urlBase", "value": "http://ajax.googleapis.com/" }
		]
	},
	{
		"serviceName": "template-example",
		"serviceType": "gateway",
		"groovyScript": "$tig/config/groovy/template-example.groovy",
		"parameters": [
			{ "name": "Test Mode", "value": "false" }
		]
	}
]
```

This configuration specifies a collection of Groovy services. The service attributes include:

Name	Description

serviceName	Groovy service name, should be unique
serviceType	The type of Groovy service: [**delivery** | **gateway** | **scheduled**] • **delivery** services can be used for performing transaction delivery • **gateway** services can be called via REST service API • **scheduled** services can be called by scheduled Quartz Jobs
groovyScript	A Groovy script file path The **$tig** variable is the TIG relative path location.
groovyTypeChecked	Perform strict Java style Groovy type checking before executing the script. Default value is **false**.
executionTimeout	End execution of the Groovy script if it takes longer than the specified number of milliseconds. Default value is **0** meaning no timeout.
returnErrors	Return any Groovy script errors to the caller when performing a Service API Gateway call. Use this option when developing scripts, but not in production. Default value is **true**.
basicAuth	The HTTP Basic authentication header required to execute the call when performing a Service API Gateway call The basic auth header is a Base64 encode username:password value.
parameters	Collection of parameters to pass to the Groovy script as a parameters map. This is useful for externalizing configuration values.

TIG will monitor any changes to the Groovy configuration file and Groovy script files and make these available immediately.

Gateway Services Config

Groovy services with the type **gateway** can be called externally from Manager using the gateway service URL. The URL path is used to resolve the name of the service to execute.

An example of the gateway service configuration is illustrated below:

1. URL request from Manager to TIG:

```
GET https://service.mycorp.com/tig/service/google-search/?q=Avoka
```

2. TIG extracts service name path:

```
google-search
```

3. TIG looks up service name called "google-search" and invokes the **gateway** service type.

Only services with the type **gateway** can be called externally. Attempting to call a **delivery** or **scheduled** type service will result in a HTTP **404 Not Found** status code being returned.

Scheduler Config

The scheduler configuration file defines Quartz job schedules for invoking Groovy services. Only services with the **scheduled** service type can be called by Quartz scheduled jobs.

The sample **scheduler-config.json** installed with TIG is shown below.

```
[
	{
		"jobName": "Cron Job",
		"jobType": "cron",
		"cronExpression": "0 30 1 * * ?",
		"serviceName": "job-example"
	},
	{
		"jobName": "Simple Job",
		"jobType": "simple",
		"repeatIntervalMins": 10,
		"serviceName": "job-example"
	}
]
```

This configuration specifies a collection of Groovy service job schedules. The job schedule attributes include:

Name	Description
jobName	Name of the job
jobType	Job type: [**cron** | **simple**]
cronExpression	A cron scheduling expression, for **cron** type jobs only. For details, see Quartz Tutorials, Lesson 6: CronTrigger.
repeatIntervalMins	Repeat interval in minutes, for **simple** type jobs only

serviceName	Name of the Groovy service to call when the job is triggered

If you make changes to **scheduler-config.json**, you must restart the TIG server for these changes to take effect.

Storage Config

The storage configuration file defines customer transaction data storage options.

The sample **storage-config.json** installed with TIG is shown below.

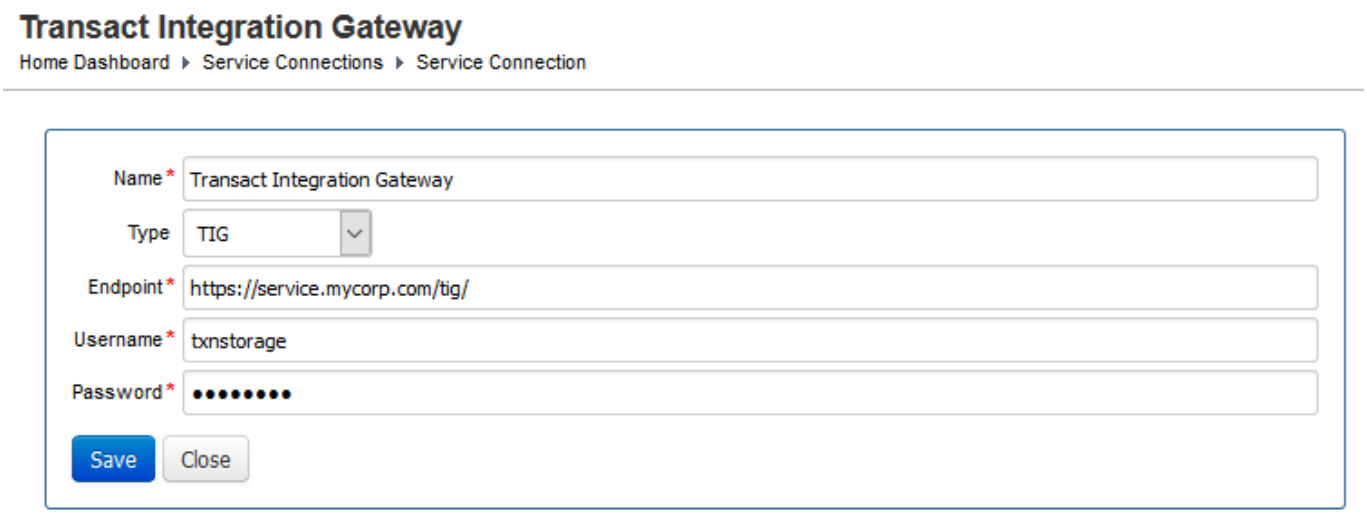
```
{
        "user": "administrator",
        "password": "tuart!23",
        "dataDir": "C:/tig/storage/"
}
```

The storage attributes include:

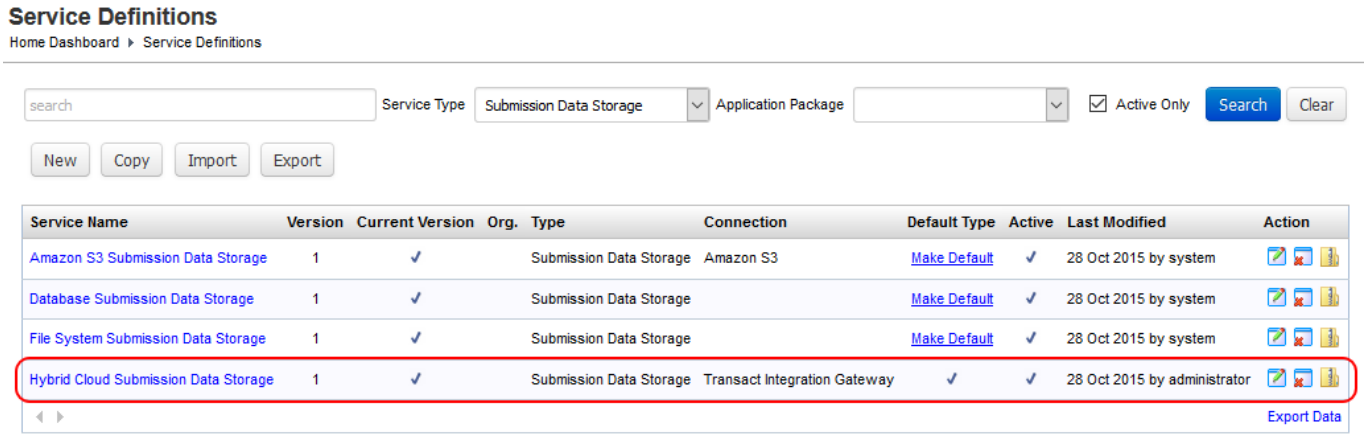
Name	Description
user	The BASIC authentication login user This value should match the Manager TIG Service Connection Username.
password	The BASIC authentication login password This value should match the Manager TIG Service Connection Username.
dataDir	The customer transaction data storage area

If you make changes to **storage-config.json**, you must restart the TIG server for these changes to take effect.

The image below shows the Manager configurations for a Transact Integration Gateway Service Connection.



The image below shows a Manager configuration for the Hybrid Cloud Submission Data Storage service using Transact Integration Gateway.



Next, learn about developing with Transact Integration Gateway.

Transact Integration Gateway Development

Journey Platform | System Manager / DevOps | v4.3 & Higher

Although Transact Integration Gateway (TIG) continues to receive support, it is recommended that Journey Platform solution builders move away from using TIG and instead use Journey Manager REST APIs to build Journey Platform solutions. Continuing to use TIG may complicate an upgrade path when TIG is eventually deprecated in a future release.

Groovy Development

TIG provides the same integration libraries as Manager for performing Groovy script development. This provides a wide range of options for performing simple and sophisticated integrations.

For details on the Groovy language, see the Groovy Language Syntax Specification.

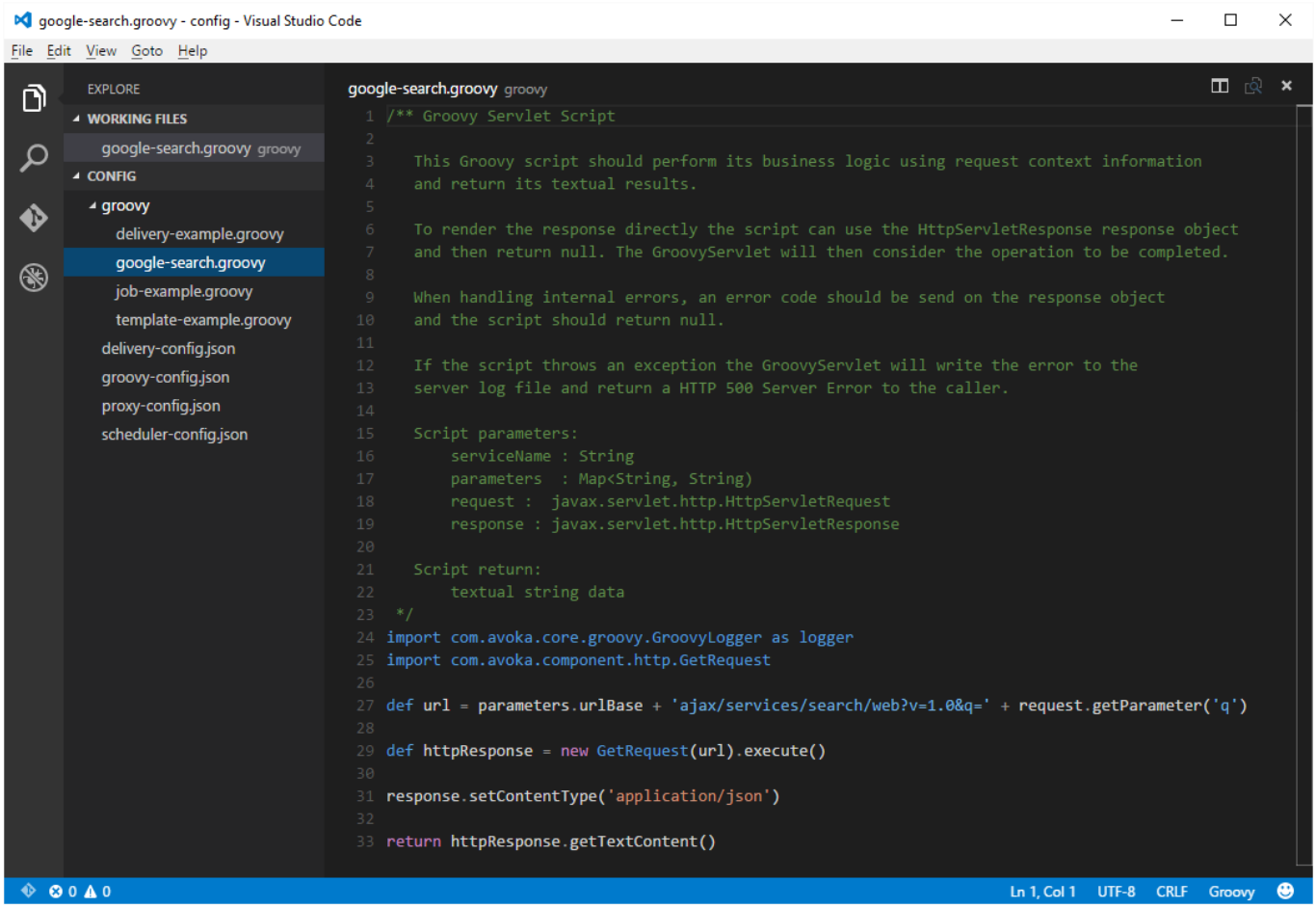
Security Restrictions

The TIG Groovy 2.4.5 runtime is the same one provided with Manager and includes a number of security restrictions to prevent scripts from terminating the application server.

Script Development

A good way of developing TIG Groovy scripts and making configuration changes is to use a code editor and point it at the TIG config directory. Any changes you make to your Groovy scripts will be loaded immediately by TIG, and your scripts will be interpreted dynamically. This enables you to quickly develop and test scripts without having to go through lengthy compile and deployment cycles. The development experience is much like writing JavaScript.

A good code editor to use for this is the free Microsoft Visual Studio Code. In Visual Studio Code, open a TIG configuration directory to edit your Groovy scripts and configurations.



Delivery Service Script

A **`delivery`** type service Groovy script example is shown below. This script details the Groovy script parameters and error handling.

```
/** Groovy Delivery Script

       This Groovy script should perform any transaction delivery.

       If the delivery integration fails, then the script can throw an exception and the
       delivery will not be completed in Transaction Manager and can be retried again at a
later point.

       If the script does not throw an exception, the transaction will be considered
       to have been delivered, and TM will be notified that delivery has been completed.

       At this point Transaction data will be available for purging.

       Script parameters:
               serviceName : String
               parameters  : Map<String, String>
               transaction : com.avoka.transact.service.delivery.Transaction
               deliveryDir : String
 */
import com.avoka.core.groovy.GroovyLogger as logger

def deliveryInfo = """
   serviceName: $serviceName
    parameters: $parameters
transaction.id: $transaction.id
   deliveryDir: $deliveryDir
"""

logger.info deliveryInfo
```

The Groovy **delivery** service example below performs an Email delivery to the Microsoft Exchange Email server and includes the form XML, PDF receipt and file uploads as attachments to the email. The example externalizes email service configurations to service parameters.

```
/** Groovy Delivery Script

       This Groovy script should perform any transaction delivery.

       If the delivery integration fails, then the script can throw an exception and the
       delivery will not be completed in Transaction Manager and can be retried again at a
later point.

       If the script does not throw an exception, the transaction will be considered
       to have been delivered, and TM will be notified that delivery has been completed.

       At this point Transaction data will be available for purging.

       Script parameters:
               serviceName : String
               parameters  : Map<String, String>
               transaction : com.avoka.transact.service.delivery.Transaction
               deliveryDir : String
 */
import com.avoka.core.groovy.GroovyLogger as logger
import com.avoka.transact.service.delivery.Transaction
import com.avoka.transact.service.delivery.Transaction.Attachment
import microsoft.exchange.webservices.data.core.*
import microsoft.exchange.webservices.data.credential.*
import microsoft.exchange.webservices.data.core.service.item.EmailMessage
import microsoft.exchange.webservices.data.core.enumeration.misc.ExchangeVersion
import microsoft.exchange.webservices.data.property.complex.MessageBody

logger.debug """$serviceName parameters: $parameters"""

// Create Exchange WS client
ExchangeService service = new ExchangeService(ExchangeVersion.Exchange2010_SP2)
service.setUrl(new URI(parameters.emailUrl))
service.setCredentials(new WebCredentials(parameters.emailUser, para-
meters.emailPassword))

// Create email subject line and message body
def subject = """$transaction.formCode Submission: $transaction.id"""

def body = """
<pre>
    Form Name: $transaction.formName
Tracking Code: $transaction.trackingCode
        Email: $transaction.contactEmailAddress
```

```
</pre>
"""

// Send email with attachments
EmailMessage msg = new EmailMessage(service)
msg.getToRecipients().add(parameters.emailUser)
msg.setSubject(subject)
msg.setBody(MessageBody.getMessageBodyFromText(body))
msg.getAttachments().addFileAttachment("form.xml", transaction.xmlData)
msg.getAttachments().addFileAttachment("receipt.pdf", transaction.receiptData)
for (Attachment attachment : transaction.attachments) {
       def name = "attachment-" + attachment.id + "-" + attachment.fileName
       msg.getAttachments().addFileAttachment(name, attachment.fileData)
}
msg.send()

// Log message
logger.info """$subject  email sent to: $parameters.emailUser"""
```

Gateway Service Script

A **gateway** type service Groovy script is shown below. This script details the Groovy script parameters and error handling.

```
/** Groovy Servlet Script

       This Groovy script should perform its business logic using request context information
       and return its textual results.

       To render the response directly the script can use the HttpServletResponse response
object
       and then return null. The GroovyServlet will then consider the operation to be com-
pleted.

       When handling internal errors, an error code should be send on the response object
       and the script should return null.

       If the script throws an exception the GroovyServlet will write the error to the
       server log file and return a HTTP 500 Server Error to the caller.

       Script parameters:
               serviceName : String
               parameters  : Map<String, String>
               request     :  javax.servlet.http.HttpServletRequest
               response    : javax.servlet.http.HttpServletResponse

       Script return:
               textual string data
 */
import com.avoka.core.groovy.GroovyLogger as logger
import com.avoka.component.http.GetRequest

def url = parameters.urlBase + 'ajax/services/search/web?v=1.0&q=' + request.getParameter
('q')

def httpResponse = new GetRequest(url).execute()

response.setContentType('application/json')

return httpResponse.getTextContent()
```

Scheduled Service Script

A **scheduled** type service Groovy script is shown below. This script details the Groovy script parameters and error handling.

```
/** Groovy Job Script

       This scheduled Groovy script should perform its business logic.

       Any exceptions thrown will be logged to the server log file.

       Script parameters:
               jobName     : String
               serviceName : String
               parameters  : Map<String, String>
 */
import com.avoka.core.groovy.GroovyLogger as logger

def jobInfo = """
       jobName:     $jobName
       serviceName: $serviceName
       parameters:  $parameters
"""

logger.info jobInfo
```

Adding JAR Libraries

To include additional Java libraries (JAR) for use by Groovy scripts, place these libraries in the placeholder JBoss modules folder:

```
C:\avoka\transact\tig\server\modules\com\avoka\custom\main
```

Next, edit the **`modules.xml`** file to reference your included JAR files:

```
C:\avoka\transact\tig\server\modules\com\avoka\custom\main\modules.xml
```

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.avoka.custom">
	<resources>
		<!-- Include your JAR libraries  here. -->
		<resource-root path="custom.jar" />
	</resources>
	<dependencies>
		<!-- Included your library module dependencies here. -->
		<module name="system" />
	</dependencies>
</module>
```

You must restart the TIG service for these libraries to become available.

If your JAR file has dependencies on third party libraries, you should also include these files in this modules directory.

If the server has troubles loading your JAR files, you will typically see these when the server starts up or when you call your Groovy script.

Logging

To monitor the activity of TIG, see the server log file:

```
C:\avoka\transact\tig\server\standalone\log\server.log
```

When developing Groovy scripts, you can use the **`GroovyLogger`** to log **`DEBUG`**, **`INFO`**, **`WARN`** and **`ERROR`** messages to the server log file

```
import com.avoka.core.groovy.GroovyLogger as logger

logger.debug 'Started service call'

// ...

logger.debug 'Completed service call'
```

To tune the server logging output, edit the **`standalone.xml`** configuration file in:

```
C:\avoka\transact\tig\server\standalone\configuration\standalone.xml
```

By default, the **GroovyLogger** will only log **INFO** and above level messages.

See the JBoss standalone XML elements below for the various configuration options:

```
<subsystem xmlns="urn:jboss:domain:logging:1.1">
	<console-handler name="CONSOLE" autoflush="true">
		<level name="INFO"/>
		<formatter>
			<pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n"/>
		</formatter>
	</console-handler>
	<periodic-rotating-file-handler name="FILE" autoflush="true">
		<level name="INFO"/>
		<formatter>
			<pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n"/>
		</formatter>
		<file relative-to="jboss.server.log.dir" path="server.log"/>
		<suffix value=".yyyy-MM-dd"/>
		<append value="true"/>
	</periodic-rotating-file-handler>
	<logger category="com.avoka.core.groovy.GroovyLogger">
		<level name="INFO"/>
	</logger>
	<root-logger>
		<level name="INFO"/>
		<handlers>
		<handler name="CONSOLE"/>
		<handler name="FILE"/>
		</handlers>
	</root-logger>
</subsystem>
```

Next, take a look at an overview of the Temenos Journey Platform.

Transact Manager Previous Releases

Journey Platform

You can find documentation for earlier version releases of Journey Manager (Transact Manager) using the following links:

- Transact Manager v4.3
- Transact Manager v5.0

> You must have an existing KB user accound to access these links. The KB account is different to the community account.

Transact Fluent SDK Overview

Journey Platform | Platform Developer | 5.1 & Higher | Deprecated in 17.10

Journey Manager is designed to be integrated with external systems. This makes the Journey platform highly extensible, allowing it to meet the growing demands of the most complex financial and government environments with low or no code changes by using configuration rather than development activities. Clients can extend the Manager core capabilities to achieve goals that aren't possible with built-in functionality, such as integration with a custom back-end system.



Manager comes with the Transact Fluent SDK, which developers can use to extend the Journey platform by writing **Groovy**[1] scripts that hook into Manager's services and extension points.

Transact Fluent SDK includes:

- Transact Fluent API references.
- Manager Fluent libraries.
- Manager Fluent project templates to develop Groovy services using modern **IDE**[2]s.
- Tools to help interact with Manager Rest APIs to support deployment and testing of services.

The Fluent Groovy APIs have been designed to enable a simplified Fluent programming style, improve learning, reduce programming complexity, increase productivity, and improve code readability. Upgrades and maintenance become smoother because of the simplified and unchanging Transact Fluent API. Finally, the new APIs provide enhanced levels of security and performance.

---

[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]Integrated Development Environment.

Transact Fluent SDK supports **IDE**[1]-based development that includes:

- Source control integration for team development projects
- Code modularization and reuse
- Unit testing
- Real-time code completion and error checking

Transact Fluent SDK creates a development environment for clients, partners, and the Customer Service team to rapidly extend Manager functionality with new back office Integrations, workflows, and service connections to various **FinTech** [2] services.

Next, learn about the benefits of using Groovy in Manager.

---

[1]Integrated Development Environment.

[2]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.

Benefits of Using Groovy in Journey Manager

Journey Platform | Platform Developer | 5.1 & Higher | Deprecated in 17.10

Journey Manager is written in **Java**[1] and runs on **Java Virtual Machine**[2] (JVM). Manager provides extensive support for using **Groovy**[3] language, which enables developers to quickly implement business and integration logic without having to deploy custom code as jar files and restart Manager servers.

Groovy has the following benefits, over other scripting languages, that make it the right candidate for Manager:

[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] 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.

[3] 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.

- Groovy is a dynamic scripting language, which allows developers to implement and customize Groovy scripts without re-compiling and re-deploying Manager applications. The scripts can be stored as files or as objects in the **DB**[1]. Manager extensively utilizes Groovy scripts to implement Transact Services. It has the advantage that the service does not need to be compiled nor the containing component rebuilt prior to use. For more information, see the service definitions extension points listed below.
- Groovy is developed for Java, so it runs on JVM and most of the code written in Java will compile and work as expected when you try to run it as a Groovy script. It seamlessly and transparently integrates and is compatible with Java and any third-party libraries.
- Groovy is Java syntax compatible, so learning Groovy won't be challenging for someone who knows Java already.
- Groovy has concise, brief and direct syntax, which makes it easy to learn, develop and read the code. It requires less lines of code to implement the same functionality as compared with Java.
- Groovy has native support for markup languages, such as XML and HTML, with which developers can quickly create sophisticated application logic and in less code. It also enables the definition and manipulation of many types of data assets, which is handy in data preparation and manipulation.
- Groovy comes with integrated support for converting between Groovy objects and **JSON**[2].
- Groovy is a testing-oriented development language, which is great for writing concise and maintainable tests, and for all your build and automation tasks.
- Groovy has powerful features - closures, builders, runtime and compile-time meta-programming, functional programming, type inference, and static compilation - which allows developers to implement complex application rules in a concise and well defined way.

Groovy was first introduced in Manager v3.3.0 and it has been extensively used since v3.5.0. Manager v17.10 supports Groovy v2.4.15. Every major Manager release includes the latest Groovy release. If you want to keep up-to-date with Groovy releases, refer to http://groovy-lang.org/index.html.

Manager uses service definitions, which provide Groovy extension points for the following service types.

[1]A database is a data structure that stores organized information. Most databases contain multiple tables, which may each include several different fields. For example, a company database may include tables for products, employees, and financial records. Each of these tables would have different fields that are relevant to the information stored in the table.

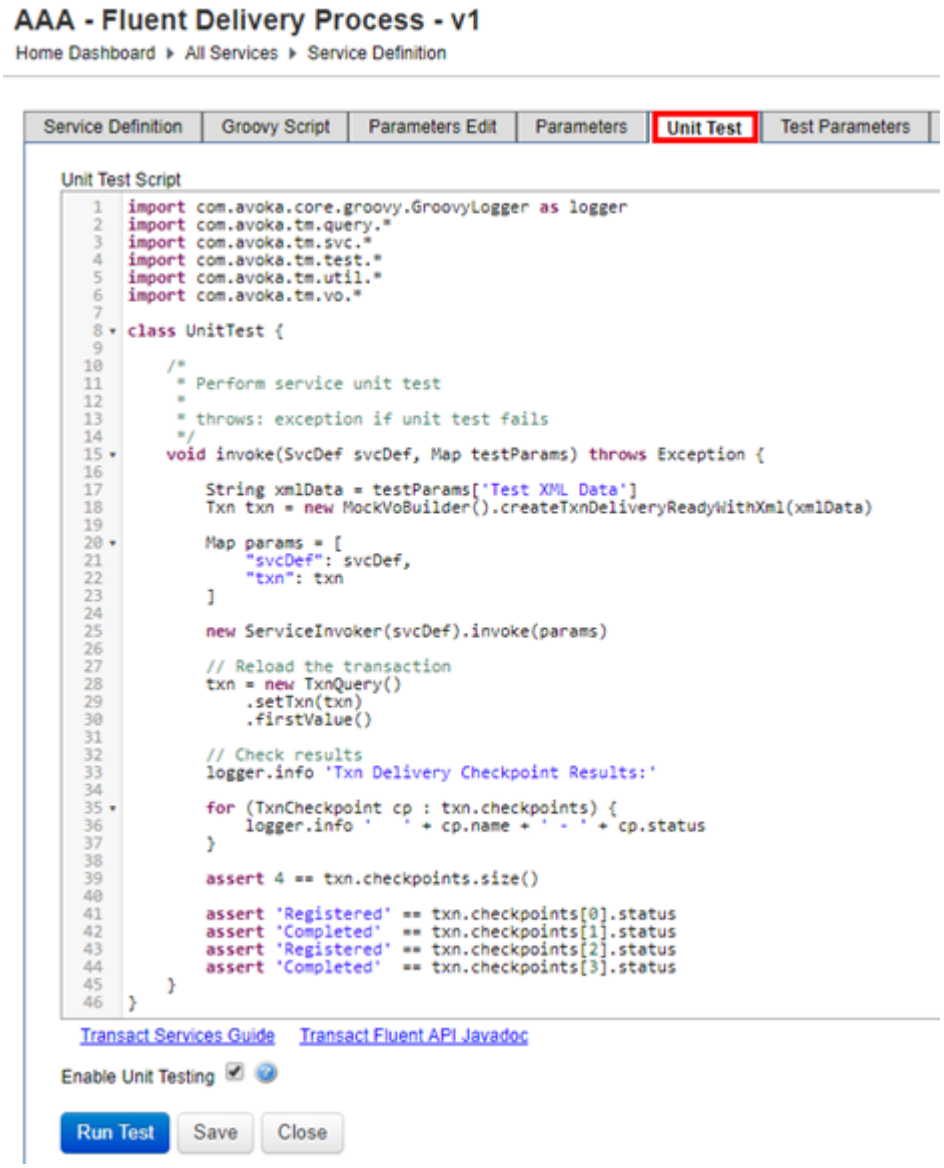
[2]JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.

- Delivery Process.
- Dynamic Data.
- Form Prefill.
- Form Saved Processor.
- Groovy Service.
- Receipt Number.
- Scheduled Service.
- Submission Complete Processor.
- Submission Preprocessor.
- Task Expiry.

All these services have Groovy tab that allows developers to write a Groovy script to implement specific requirements.



The script can be tested using the Unit Test tab, which significantly speeds up the development process and helps to find most of the errors before making use of this service.



Next, learn about benefits of using Fluent Groovy services.

Fluent Groovy Services vs Groovy Services

Journey Platform | Platform Developer | 5.1 & Higher | Deprecated in 17.10

Journey Manager is based on an extensible service-oriented architecture, which enables developer to implement highly configurable services written in **Groovy**[1] language. There are two types of service to choose from to implement a new Groovy service: Fluent Groovy service and Groovy service. The main difference between them is:

- Fluent Groovy service uses Transact Fluent API.
- Groovy service uses Transact Core API.

> It is strongly recommended to use the Fluent Groovy service for any new clients, services and applications development. If this can't be done due to limitations in the Fluent APIs, Groovy services must be used. However, some efforts must be made to migrate existing Groovy services to Fluent ones, whenever possible, because Manager future version will force everyone to only use the Fluent APIs.

The detailed difference between the services is outlined below.

Fluent Groovy Service

The Fluent Groovy service is a service that can only use Transact Fluent API. It is impossible to call any Transact Core API from this service. The reasons are as follows:

[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.

- Simplify and speed up service implementation for developers because they need to know only limited set of Fluent APIs (34 Fluent API classes vs 1042 Core API classes)
- Enable to use the Fluent programming model, see the example below:

```
import com.avoka.tm.svc.*
def params = [:] params.refNumber = ...
def result = new GroovyServiceInvoker()
.setServiceName("Ref Lookup")
.setClientCode("maguire")
.setVersion("0.2.1")
.invoke(params)
```

- Allows to use short names, and immutable value objects
- Reduce time to market and reduce development costs
- Faster to train developers and scale teams
- More supportable and stable API contract, because they are better regression tested
- Easier upgrades
- Improved system performance
- Better architecture for writing server-side business logic where services are tied to a single trigger point and can be used across a range of triggers and arbitrary invocations
- Allows Groovy types checking
- Restricted from using Transact Core APIs to prevent developers from running into implementation troubles or design confusions. For example, object mapping API is restricted, but value objects API is allowed.
- Enforces the Java-like class syntax, as opposed to script-like syntax. It makes development more structured. In the above example, if you create a new service from Fluent Dynamic Data template, your new Groovy script will look like this (compare to the script example in the Groovy Service section):

```
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import javax.servlet.http.*

class FluentDynamicData {
	public Logger logger
	String invoke(SvcDef svcDef, Txn txn, HttpServletRequest request, User user) {
		String data = '''{ "address": { "firstLine": "123 Wall Street" } }'''
		return data
	}
}
```

Developers are encouraged to create new services as Fluent Groovy services. Developers might encounter a few situations when it is difficult, if yet possible, to implement a service using Fluent Groovy service (due to Transact Fluent API limitations). However, it can be resolved on case by case basis. For more information, see Fluent Groovy Service Developer Guide.

Fluent service can only invoke Dynamic Groovy services, if the install time Groovy Secure API restriction is not enforced.

Groovy Service

The Groovy Service uses both Transact Core API and Transact Fluent API, which allows developers to implement nearly any service possible. However, there are number of disadvantages that are listed below:

- Implementation can become complex and labor extensive because there are hundreds of APIs to learn and use. This significantly increase the development cost.
- Developers should have deep understanding of the APIs to be able to write services.
- High possibility to execute potentially dangerous code that can crash the whole Manager server, for example **`System.exit()`**, or directly manipulate with database objects.
- Groovy code is not stable because it relays of Transact Core API that can change and will change over time to accommodate for bug fixes and Manager new features.
- There is no type checking or any other static checks on the Groovy service code, so often the first time you would find out something had broken was when it failed at runtime.
- Provides the script-like syntax. In the above example, if you create a new service from Groovy Dynamic Data template, your new Groovy script will look like this:

```
import com.avoka.core.groovy.GroovyLogger as logger
def data = '''{
       "address": {
               "firstLine": "123 Wall Street"
       }
}'''
return data
```

Developers are discouraged to create new services using core Groovy services, which are considered the legacy API and have been deprecated since Journey Manager version 19.11. Instead, whenever possible, use Fluent Groovy Service .

As the core API are deprecated, you can find all legacy Groovy services, which use these core API, by running the following script in the Groovy console with the **Secure Fluent API Only** checkbox unselected.

```
import com.avoka.fc.core.dao.*
import com.avoka.tm.query.*
import com.avoka.tm.vo.*

List<SvcDef> svcDefs = new SvcDefQuery()
      .withCurrentVersion()
      .setFetchLimit(50000)
      .listValues()

println "all services: " + svcDefs.size()

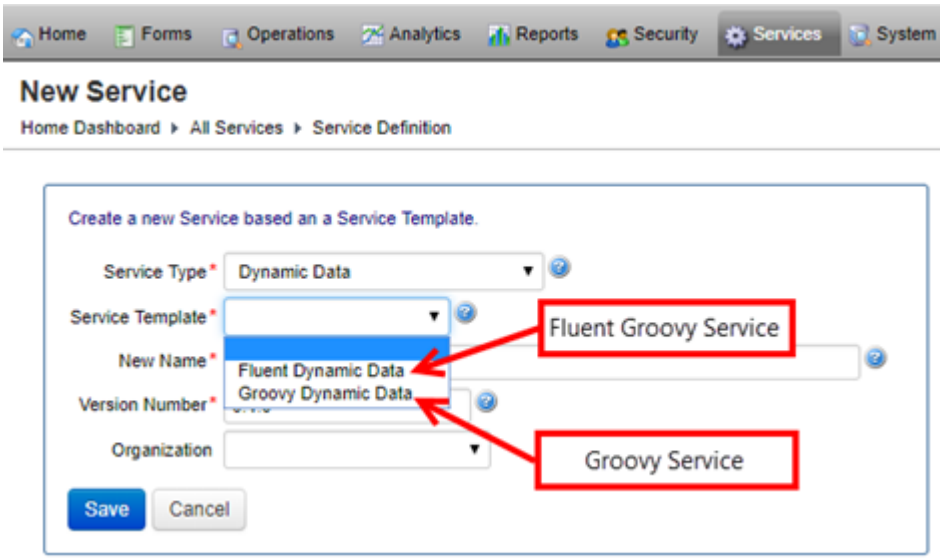
def legacyServices = svcDefs.collect{ svcDef ->
    def svc = DaoFactory.serviceDefinitionDao.getServiceDefinitionForPK(svcDef.id)
    return !svc.isFluentApiService() ? svc : null
}.findAll {it && it.getClassnameBeanname().contains("Groovy") }

println "all non fluent services: " + legacyServices.size()

legacyServices.collect { it.serviceName + " >> " + it.serviceType }.each {
  println it
}
```

Configuration Example

To create a new Dynamic Data service (Groovy service) from **Services** menu, you have an option to create this service based on the Fluent Dynamic Data Template (Fluent Groovy service), or Groovy Dynamic Data Template (Groovy service), as shown below.



It is recommended to use Fluent Groovy service.

Next, learn about Transact Fluent SDK.

Fluent API REST Service

Journey Platform | Platform Developer | 21.05

As you know, Journey Manager comes with an extensive set of private Fluent API enabling you to implement various complex business requirements. All you need to do is to just write a Groovy scrip that calls a few API and run it within your service. And it all works. However, what happens when you want to invoke the same Fluent API outside Manager. Now, we have a solution for you!

The 21.05 release of Journey Manager has introduced a new service (**`com.avoka.fc.admin.service.FluentApiService.java`**) which exposes some of Manager's Fluent API via **REST**[1] end points. This service is simple to understand and use and yet it provides you with ability to invoke rich in functionality Fluent API.

> This is the Beta release of the Fluent API REST service, which implements only a limited number of Fluent API. We may need to update or even change some service functionality or implementation depending on REST performance, usability, and your feedback.

Now, you are probably curious to find out how it all works. Well, it's rather simple. The Fluent API REST service is included in the new Transact Manager WAR module, along with other REST services, so if you've installed or upgraded Journey Manager to the 21.05 release, you can start using the Fluent API REST service. The new service architecture is illustrated in the diagram below:

Requests

You interaction with the Fluent API REST service starts by sending a REST request. Each Fluent API class and every of its methods (operations) form a single endpoint, which you call by sending a request to a URL in the following format:

```
https://{SERVER}/manager/secure/rest/fluentapi/{CLASSNAME}/{OPERATION}
```

Where:

- **`SERVER`** is your Manager server instance
- **`CLASSNAME`** is a case-insensitive name of Fluent API class, for example, **`txnupdater`** for the **`TxnUpdater`** class
- **`OPERATION`** is a case-insensitive name of a final method (oreration) to be called on this API, for example, **`update`**

[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.

We have chosen the combination of class names and methods to simplify the learning curve for developers, who are already familiar with our Fluent API.

You have to provide a body payload for an operation in a **JSON**[1] format, where each **set** method has its own entry in the JSON payload. The Fluent API REST server processes the request and sends back a response including a response code and a payload, depending on an operation.

Let's illustrate it with a simple example. Suppose, you want to update a transaction delivery status from a Groovy script. For that, you use the **TxnUpdater** class, as shown below.

```
import com.avoka.tm.svc.*
import com.avoka.tm.vo.*

new TxnUpdater(txn)
    .setId(733)
    .setDeliveryStatus(Txn.DELIVERY_READY)
    .update()
```

You can achieve the same result by executing this POST request with a payload:

```
POST https://{SERVER}/manager/secure/rest/fluentapi/txnupdater/update
Content-Type:application/json
Accept:application/json
{
    "setId" : "733",
    "setDeliveryStatus" : "Ready"
}
```

Let's examine the REST POST method:

- **txnupdater** is the **TxnUpdater** class
- **update** is the last method to be called on the instance of TxnUpdater class, which is the **update ()** method
- **setId** sets a transaction ID to be updated, which is the **setId()** method
- **setDeliveryStatus** sets the Ready status of a transaction, which is the **setDeliveryStatus()** method

You can see the REST method basically mimics the Fluent API call, which makes developer's life much easier.

---

[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.

Responses

Each Fluent API REST request has a corresponding response including an HTTP code and data in a JSON format or an error message. We've simplified and minimized the number of responses, so it's easy to use the service.

The Fluent API REST service returns the following HTTP response codes:

HTTP Code	Description
200 OK	This code is returned when a Fluent API REST call is successful.
400 Bad Request	This code is returned when a Fluent API REST call fails due to bad input. The error message in the response includes an error log ID so you can find more details about the error in the error log.
404 Not Found	This code is returned when a Fluent API REST call fails trying to update or access objects that don't match provided ID. The error message in the response includes an error log ID so you can find more details about the error in the error log.

Security

Manager has a new **REST Fluent API** core permission which you must assign to roles of users who will invoke the Fluent API REST service. The **Administrator** role has this permission set by default. We recommend creating a dedicated user with this permission.

Class Names

This release of Manager has the following CLASSNAMEs implemented:

CLASSNAME	Usage Example
ActionResultBuilder	
DeliveryTxnBuilder	
Emailer	
GroupQuery	
JobActionQuery	
JobBuilder	
JobQuery	
JobStepQuery	
JobUpdater	

Pay-
mentLo-
gQuery

Prop-
ertyQuery

PropertyBuilder RefDataSvc RefDataQuery RoleQuery SpaceQuery SvcConnQuery SvcDefQuery SvcConnUpdater TrackingCodeBuilder TxnBuilder TxnUpdater TxnHistoryQuery	
	Query by ID

TxnQuery	`https://localhost/manager/secure/rest/fluentapi/txnquery/firstValue`
	
	`{`
	`  "setId" : "733"`
	`}`

UserQuery	

UserBuilder	`https://localhost/manager/secure/rest/fluentapi/userbuilder/create`
	
	`{`
	`    "setEmail" : "person@temenos.com",`
	`    "setUserType" : "Local",`
	`    "setLoginName" : "loginName",`
	`    "setPassword" : "pAs3W0rd!@#"`
	`}`

UserEmailer

```
https://localhost/manager/secure/rest/fluentapi/useremailer/sendEmail

{
    "setLoginName" : "loginName",
    "setSpaceName" : "Work Space",
    "setType" : "WELCOME"
}
```

Next, learn about Fluent Groovy Services vs Groovy Services.

About Journey Platform New Releases

Journey Platform | All Personas | 5.1 & Higher

Product versioning and product releases go hand in hand with each other. We update and improve the Temenos Journey Platform products on a regular basis and make these changes available to our customers through individual product releases. We use major and minor releases to distribute new product versions. The kind of changes included in a product release depend on whether it is a major or minor release:

Major Release

A major release adds new product features, enhancements, libraries, and bug fixes, and, sometimes, can introduce behavioral changes that may affect Groovy services and integrations with other systems. While we try to minimize these changes, you need to review all product enhancements carefully to determine if and how they may affect your existing implementations.

A major release is denoted by a 0 as the third number in the release number, with the two first numbers representing the year and month of the release. For example, **18.05**.0, **19.05**.0, or **20.11**.0 or **21.11**.0.

Maintenance Release

A maintenance release usually includes only bug fixes and minor changes. However, there can be exceptions when a maintenance release includes new features. If this is the case, it will be clearly documented in the product release notes.

A minor release is denoted by the third number in the release number being greater than 0. For example, 18.05.**10**, or 19.05.**5**.

Release Notes Sitemap

Detailed information on all major and minor releases is available from the Release Notes Sitemap in our Journey Platform resources website.

What's New

For a quick summary of What's New in each major release, see:

What's New in 21.11

What's New in 21.05

What's New in 20.11

What's New in 20.05

What's New in 19.11

What's New in 19.05

Next, learn about our End-of-Life policy.

What's New in 21.11

Journey Platform | All Personas | 21.11

The 21.11 release has a number of new exciting features and enhancements across the whole range of Temenos Journey Platform products. The updated products and their changes are listed below.

Journey Maestro New Features

This release of Maestro includes long awaited updates and bug fixes of many early features, so you'll be delighted to upgrade your current product to the latest 21.11 version of our software.

Read the detailed Maestro release notes or check the main release features.

- Enhanced the TM Form Version building functionality so you can choose which brand to use from the Brands Options.
- Improved **white-labelling**[1] support so you can set up multiple brands in a single template much easier.
- Introduced the new Avalon Template, which replaces the Maguire template, and brings a more modern look and feel and encapsulates common design pattern, helping to speed development time frames.
- We have partnered with XLTS.dev who will provide on-going support for AngularJS for Maestro to ensures its longevity.

[1]White-label software is software that is purchased by a company from a service provider and re-branded as its own.

Journey Manager New Features

We are pleased to announce our latest 21.11 release of Journey Manager is ready for you to download! It's packed with new features and bug fixes, which you have been waiting for.

Read the detailed Journey Manager release notes or check the main release features.

- Introduced a new set of statuses for organizations, form spaces, forms and form versions, so you have fine control over each of the entities.
- Updated 3rd party libraries in line with our continuous software development.

Journey Analytics New Features

We are pleased to announce the latest 21.11 release of Journey Analytics is ready! The 21.11 enhancements improve existing functionality and incorporate the following updates.

Read the detailed Journey Analytics release notes or check the main release features.

- The Timeline View now provides users more options for aggregating data. The tooltip displayed on hover of the bar chart has also been improved to order the colour sequence to match the bar chart for better read-ability.
- Users can now select a custom date range in a single view in the calendar.
- Ability to allow users to see near-time data from the current day by adding a new **Today** option to the period selector.
- For those customers with forms across multiple organisations, the organisation name will now be displayed above the form name in the Scope Selector to help differentiate between forms of the same name across dif-ferent organisations.
- The User Journey View segmentation chart has been extended to accommodate the display of the top 10 segmentation values.

Journey Workspaces New Features

We are pleased to announce the latest 21.11 release of Journey Workspaces. This release introduces a new space for authenticated applicants in addition to improving the Workspaces experience for processing agents, helpdesk agents, assisted channel staff, and group managers.

Read the detailed Journey Workspaces release notes or check the main release features.

Key features included in this release:

- A new space for authenticated applicants as a self-service portal for managing application creation, follow-ups, and keeping track of application progress.
- An improved build and deployment process for Workspaces which is now available directly from the product Artifactory.
- Support for job properties to configure the Workspaces key info section and custom cards.
- Support for switching between languages used for the Workspaces UI, and other enhancements to the localization framework.
- An enhanced focused search feature that now allows searching on fields that are not displayed in the item list.

What's New in 21.05

Journey Platform | All Personas | 21.05

The 21.05 release has a number of new exciting features and enhancements across the whole range of Temenos Journey Platform products. The updated products and their changes are listed below.

Journey Maestro New Features

This release of Maestro includes long awaited updates and bug fixes of many early features, so you'll be delighted to upgrade your current product to the latest 21.05 version of our software.

Read the detailed Maestro release notes or check the main release features.

- Maestro Dashboard improvements
- Components space separator
- Security hardening
- Maestro Editor improvements

Journey Manager New Features

We are pleased to announce our latest 21.05 release of Journey Manager is ready for you to download! It's packed with new features and bug fixes, which you have been waiting for. One of the exciting features is a Fluent API REST service. As you know, Manager comes with an extensive set of private Fluent API enabling you to implement various complex business requirements. All you need to do is to just write a Groovy scrip that calls a few API and run it within your service. And it all works. However, what happens when you want to invoke the same Fluent API outside Manager. Now, we have a solution for you!

Read the detailed Journey Manager release notes or check the main release features.

- A new REST service - Proof of Concept
- Fluent API
- Manager installer
- Reports

What's New in 20.11

Journey Platform | All Personas | 20.11

The 20.11 release has a number of new exciting features and enhancements across the whole range of Temenos Journey Platform products:

Journey Maestro New Features

Read the detailed Journey Maestro release notes to learn about these new features and enhancements.

- Introduced the new concept of shared organization libraries
- Added drag and drop functionality to order libraries in the list
- Library exclude option retains the order of libraries
- Security improvements
- Performance improvements
- Developer experience enhancement

Journey Manager New Features

Read the detailed Journey Manager release notes or click the links below to learn about each new feature.

- A new magic byte scanning service
- A new Attachment File Name Regex deployment property
- Updated 3rd party libraries

SpringboardSmall Business (SMB) New Features

Read the detailed Springboard release notes or click the links below to learn about each new feature.

- Introduces a new Springboard Solution for small businesses - SMB
- Improved documentation to accommodate multiple Springboard solutions

What's New in 20.05

Journey Platform | All Personas | 20.05

The 20.05 release has a number of new exciting features and enhancements across the whole range of Temenos Journey Platform products:

Journey Maestro New Features

Read the detailed Journey Maestro release notes to learn about these new features and enhancements.

- Improved the command line offline build tool
- Added the right to left language support
- Enhanced component publishing dependency

Journey Manager v20.05 New Features

Read the detailed Journey Manager release notes or click the links below to learn about each new feature.

Azure Cloud support:

- 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:

- 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.

Journey Analytics New Features

Read the detailed Journey Analytics release notes or click the links below to learn about each new feature.

- Journey Analytics integration for Kony Visualiser web apps
- Springboard Out-of-the-Box Journey Analytics support
- New Milestone Funnel report
- UX and Design Improvements

Journey Workspaces New Features

Read the detailed Journey Workspaces release notes to learn about these new features and enhancements.

- Improved build and deployment process
- Switch between timeline tasks when viewing application details
- Task-specific actions, and a new View Notes action
- A new focused and faster search option
- A fresh new user interface
- Improved custom card – permission control, hide empty data
- Improved session logout after a period of inactivity
- Date picker enhancements: single date, custom formats
- Search/Filter options on item list retained in session
- Assignee full name displayed when assigning a task
- Enhanced application/task list – configure line wrapping in tables
- Attachment name now shown in Documents panel

Exchange New Features

Read the detailed Exchange release notes or click the links below to learn about each new feature.

- New Socure TAF package
- Updated AuthenticID TAF package
- Updated Worldpay TAF package
- Exchange Framework
- Updated FIS ChexSystems TIF package
- Updated Iovation TAF package
- Updated Plaid TAF package

Springboard v3.0.0 New Features

Read the detailed Springboard release notes or click the links below to learn about each new feature.

- A new Product Selector form with seamless transition to DAO form
- Upgrade to Maestro v19.11
- iOvation call removed for Workspaces
- Updated documentation
- Addressed some technical debt

What's New in 19.11

Journey Platform | All Personas | 19.11

The 19.11 release has a number of new exciting features and enhancements across the whole range of Temenos Journey Platform products:

Journey Maestro New Features

Read the detailed Journey Maestro release notes to learn about these new features and enhancements.

Improvements to shared components including:

- View a component's order of inheritance and source details
- View a component's properties and their source
- Display child components at any level
- Revert local overrides from properties panel or properties display
- Security hardening to address security issues uncovered during penetration testing

Journey Manager New Features

Read the detailed Journey Manager release notes or click the links below to learn about each new feature.

- General Fluent API enhancements
- Fluent API receipting enhancements
- Fluent Security Managers support added
- Adobe LiveCycle support removed
- Card Payment page removed from the Workspace
- Form version import options include **Form Function** now

Journey Analytics New Features

Read the detailed Journey Analytics release notes or click the links below to learn about each new feature.

- New A/B Testing View (now Custom Reports View) save feature and improved metrics
- Dropoff View improvements
- Improved User Journey save, zoom and segmentation features
- Scope selector improvements

Journey Workspaces New Features

Read the detailed Journey Workspaces release notes to learn about these new features and enhancements.

- Improved interface for applications with multiple applicants/products
- Supported actions offer custom actions and permission control
- Withdraw a saved application (NEW)
- Recover abandoned applications (NEW)
- Detailed console reporting to aid debugging of configuration issues (NEW)
- Monitor crucial service indicators, e.g. application SLAs, on the listing screen (NEW)
- Documents card enhancement to include/exclude documents using pattern matching
- Improved out-of-the-box templates for review, helpdesk, assisted channel, and manager

Exchange v19.11 New Features

Read the detailed Exchange release notes or click the links below to learn about each new feature.

- New AuthenticID package
- New Worldpay package
- Exchange Framework
- Updated Yodlee package
- Updated Mitek Tiden package
- Security hardening

What's New in 19.05

Journey Platform | All Personas | 19.05

The 19.05 release has a number of new exciting features and enhancements across the whole range of Temenos Journey Platform products:

Journey Maestro New Features

Read the detailed Journey Maestro release notes or click the links below to learn about each new feature.

- Receipt Designer
- Offline Builds (BETA)

Journey Manager New Features

Read the detailed Journey Manager release notes or click the links below to learn about each new feature.

- Journey Workspaces API security enhancements
- Chromium Puppeteer Receipt Service
- COR and security headers added to the form space standard properties
- A new **TxnUpdater Return Behavior** property added to deployment properties
- The organization Report Scheduler feature removed

Journey Analytics New Features

Read the detailed Journey Analytics release notes or click the links below to learn about each new feature.

- A/B Testing View (now Custom Reports View)
- Export

Journey Workspaces New Features

New and improved features - see Journey Workspaces release notes or click the links below to learn about each new feature.

- New role-based spaces – Helpdesk, Manage and Assisted Channel
- New management features – Assign and Unassign tasks
- Kick-start applications on behalf of customers
- Enhanced UI – Key app info, Applicants and Background checks
- Improved application timeline supporting complex workflow
- Out-of-the-box templates to get you going faster

Exchange New Features

Read the detailed Exchange release notes or click the links below to learn about each new feature.

- New Iovation TAF package
- New Plaid TAF package
- New Yodlee TAF package
- Updated Equifax IDMatrix package
- Updated Mitek Tiden package
- Updated Gov.au package

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.

Maestro Features

Maestro | Form Builder Platform Developer | 21.11

Journey Maestro 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

- Ready-to-use form components and complex native widgets that support standard JavaScript[1]
- Internationalization and right-to-left (R2L) language support
- Configurable form accessibility
- Security permissions and roles to access and design forms
- Advanced Maestro editor including WYSIWYG, undo editing operations, copy and paste across several forms, and source code editing functionality
- Seamless form building and deployment to Journey Manager using FAR files
- Offline Build to develop forms in your favorite IDE
- External source control
- Templates and brands to accelerate form development process
- Logging and error handling to simplify form troubleshooting
- Optimized performance
- All tools are accessible via a web browser so no local software installation is required

Features by Release

Each release of Maestro 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.

The key features for each major Maestro 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
?	? ?

[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.

For more information see Maestro 22.04 Release Notes.

21.11

Release Key Features	Description
Templates and brands	Introduced the new Avalon Template, which replaces the Maguire template, and brings a more modern look and feel and encapsulates common design pattern, helping to speed development time frames. Improved **white-labelling**[1] support so you can set up multiple brands in a single template much easier.

For more information see Maestro 21.11 Release Notes.

21.05

Release Key Features	Description
Maestro Dashboard	To improve the management of users' organizations, projects, forms, form versions, libraries, and resources, we've added ability to select multiple resources and perform the available actions on them, such as delete, export and publish. You, as a form developer, can now delete published projects, forms and form designs from the Maestrodashboard as you may not have access to the Managerdashboard on the shared environment. We've introduced the Upload new Resources button to the library resource list so you can easily add a resource to your project.
Components	We've added support to have a " " (space) as the thousand separator to the Decimal, Percentage and Currency Field components.
Security	We've updated all required libraries to harden our software security.
Maestro editor	We've added the ability to copy and paste a block of content from one form to another. It only works in a Chrome browser or any Chromium including Microsoft Edge. We've added an **Icon Picker**[2] in the rich text editor, which is used in many component's properties, for example a label.

[1]White-label software is software that is purchased by a company from a service provider and re-branded as its own.

[2]The Icon Picker is a selection dialog allowing you to pick an icon from the range of images shipped with Maestro.

	We've added support for scrolling through the entire list of Brand options (Branding for Form) to find a required configuration without clicking on each individual entries.

For more information see Maestro 21.05 Release Notes.

20.11

Release Key Features	Description
Libraries	Introduced the new concept of shared organization libraries Added drag and drop functionality to order libraries in the list Library exclude option retains the order of libraries

For more information see Maestro 20.11 Release Notes.

20.05

Release Key Features	Description
SCM	Improved the command line offline build tool
Language	Added the right to left language support
Components	Enhanced component publishing dependency

For more information see Maestro 20.05 Release Notes.

19.11

Release Key Features	Description
Shared components	Improvements to shared components: • View a component's order of inheritance and source details • View a component's properties and their source • Display child components at any level • Revert local overrides from properties panel or properties display
Security	Security hardening to address security issues uncovered during penetration tests

For more information see Maestro 19.11 Release Notes.

19.05

Release Key Features	Description
Receipts	Receipt Designer
SCM	Offline Builds (BETA)

For more information see Maestro19.05 Release Notes.

18.11

Release Key Features	Description
Libraries	Library Tagging
SCM	Maestro SCM Feature Branching
Forms	Form Dependencies Form Resource Summary

For more information see Maestro 18.11 Release Notes.

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

Next, learn more about Manager features.

Journey Analytics Features

Journey Analytics | Analytics User | Latest Version

Journey Analytics comes packed with a wide range of features empowering you to analyze complex customer onboarding and user journey solutions. The main product features have now become core functionality in the latest cloud-hosted version. Click on each feature to learn more.

Main Features

- The Dashboard provides a high-level overview of application performance.
- View an individual form's transaction metrics and better understand completion rates with Dropoff.
- Identify abandonment hotspots and trends in Field Analysis - with detailed field-level statistics.
- Visualize the User Journey milestones while segmenting users.
- Use the Timeline to compare application statuses over time.
- Custom Reports provides a customizable dashboard for many different metrics in a single view.
- Visualize a Collaboration Job and view it's key metrics for the job, step, action, section or field.
- Export analytics reports to CSV.
- Segmentation analysis to help understand the behavior of specific groups of users.
- Follow significant events in the user's journey through Custom Milestones.

Features by Release

Each release of Journey Analytics is hosted 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. All cloud-hosted customers are automatically provided with the latest release.

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

21.11

Release Key Features	Description
Scope Selector	Custom period date selector displays a single view for start and end date selection. Today option provided for the period with Last Day being renamed to Yesterday. For multi-organisation clients the organisation name is displayed with the form name in the form version selector.

	Sorting forms in the form version selector is also available for multi-organisation clients. Forms can be sorted by organisation name or form name.
Timeline View improvements	The Timeline View now provides users more options for aggregating data. The tooltip displayed on hover of the bar chart has also been improved to order the colour sequence to match the bar chart.
User Journey View segmentation chart	The User Journey View segmentation chart has been extended to display ten segmententation values compared to the previous five.

For more information see Analytics 21.11 Release Notes.

21.05

Release Key Features	Description
Custom Reports View	A/B Testing was renamed to Custom Reports. This view is a customizable dashboard that makes it easy to compare the performance of your application versions, across many different metrics in a single view. Additional reports have been added to the view, such as: Best, Worst, Avg - gauge the best, worst and average time taken to complete applications within the selected scope. Completion Rate - examine application performance in terms of user completion for the selected scope in a bar chart. Duration Summary - compare and sort the best, worst, average, and median times to complete each version in the selected scope using a table. Median Time - compare the median time to complete each version in the selected scope as a bar chart. Milestone Funnel - compare standard and custom milestone hit rates for the selected scope with a bar chart. Section Completion - see completion across sections for the selected scope using a line graph. Section Summary - compare and sort by section-level stats for each version in the selected scope in a table. Segment Split - discover the breakdown of values of a segment type for all transactions within the selected scope using a pie chart. Segment Switch - examine how users traverse values of a segment type for all transactions within the selected scope using a sankey graph. Session Map - investigate the number of sessions occurring per country for the selected scope using a choropleth graph.

	Top Fields - compare and sort by field-level stats for each version in the selected scope in a table. Transaction Summary - compare Transaction Statuses for the selected scope using this bar chart.
Improved analysis and metric readability	Tooltip enhancements for the Field Analysis and Dropoff Views to provide more detailed statistics.

For more information see Analytics 21.05 Release Notes.

20.05

Release Key Features	Description
Milestone Funnel report	A new Milestone Funnel report was introduced to the A/B Testing View. Use this report to understand how milestones are encountered across one or more applications. The chart is a bar chart where each bar represents the number of milestone transactions.
Springboard	Provided built-in analytics used by the Springboard solutions.

For more information see Analytics 20.05 Release Notes.

19.11

Release Key Features	Description
New reports available in the A/B Testing View	Segment Split, Duration Summary, Best, Worst Average Duration, and Sessions Map. Also rates have been added to the Segment Switch report. Transaction Recovery Support - Query changes were made to the Journey Analytics backend to support Workspace transaction recovery. This means that Abandoned transactions that were subsequently completed (recovered), will be reflected accurately in the Journey Analytics views.
Other enhancements	Help Tooltips - We improved help tooltips by ensuring a consistent look and feel to tooltips throughout the application. We also added more clarity to some areas via help text, namely the Period selector. Export Column Ordering - This new enhancement ensures that the order in which metrics (columns) are selected in the Export View UI, is reflected in the downloaded CSV.

For more information see Analytics 19.11 Release Notes.

19.05

Release Key Features	Description
A/B Testing View	The new Journey Analytics A/B Testing View brings a customizable dashboard that makes it easy to compare the performance of your application

	versions, across many different metrics in a single view.
Custom Data Export	The new Custom Data Export allows you to export analytics reports to CSV with the ability to select the granularity of your reports and retrieve exports for multiple applications in a single file.
Additional enhancements	Segment filtering - Added to the Global Filter Panel to allow users to view different segments of their analytics data. Job filtering - Added to the Cohorts filter on the Global Filter Panel to allow users to filter by one or more collaboration jobs, or versions of jobs. Last Analysis - Added to the Journey Analytics banner to show the date and time the analytics data was last consolidated and updated. It typically runs hourly.

For more information see Analytics 19.05 Release Notes.

18.11

Release Key Features	Description
Visualizer user interface	Insights has a brand-new user interface that brings together modern best practices to deliver a refreshed user experience. This was achieved by a major redesign of the Insights' frontend. This new user interface is called Visualizer, and will replace the Dashboard user interface from previous versions.
Open UX support	Insights extends support for event collection to include Open UX applications, including Out Of The Box React implementations, such as Springboard implementations and framework agnostic wrappers for ECCL integration. This means that open UX applications can now be analyzed using Insights, the same way Maestro applications can be analyzed.
Collaboration Job support	Insights now provides Collaboration Job analytics to monitor workflow level analytics.

For more information see Analytics 18.11 Release Notes.

Next, learn more about Manager features.

Temenos Journey Platform Features Compatibility

Journey Platform

Temenos Journey Platform software has two major releases per year, with several minor releases in between. A lot of effort goes into the product development to ensure there is backward compatibility between these releases. However, you may need to have a certain Journey Manager version installed to enable and use the following product features.

Manager

Manager Version	Product Feature	Compatibility Issue
18.11	Journey Analytics Visualizer UI.	You can only use the Journey Analytics Visualizer UI alongside Manager version 18.11. For earlier Manager versions, the original Journey Analytics Dashboard UI is used.
18.05	Manager segments.	You can only send segments from Manager to Journey Analytics from 18.05 or later. It is not possible to send segments from any earlier versions of Manager.
17.10	Manager custom milestones.	You can only send custom milestones from Manager to Journey Analytics from version 17.10 or later. It is not possible to send custom milestones from any earlier versions of Manager
17.10	Maestro semantic versioning.	You can use **semantic versioning**[1] in Maestro and Manager older than version 17.10, provided the form version number is no longer than 10 characters. For example, a version "1.0.1" is 5 characters and will work, but a version "1.0.2-develop" is 13 characters and won't work. However, if you truncate it to "1.0.3-dev", it will work.

Maestro

As we constantly update and improve our Maestro product, we make sure it can run on most recent Manager versions. You can check the compatibility matrix below to confirm which Maestro version you can run on your current Manager installation.

---

[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.

Maestro Version	Manager Version	Compatibility Issue
19.05+ 21.11	21.11	Journey Manager 21.11 introduced the CSRF Protection option for XSRF tokens support in the workspaces, workspace and web-plug form spaces. If this option is enabled, forms other than the latest Maestro maintenance releases 19.05+ or forms with custom version of the Attachment Field widget will be unable to upload attachments in the form. To resolve this, you will need to upgrade and re-deploy the form with this fix, available with the Maestro maintenance releases 19.05+, including 21.11.
19.11.2	18.11.2 and later	None
18.11	17.10	None
18.05	17.10 and later	The **`formInit`** event, which is used for performance metrics, is only supported in Manager version 17.10.8 and later.
17.10	17.10	The form version number field has changed from 10 to 20 characters, so a new form with a long version, for example, 1.0.1-develop, will not work in Manager prior to version 17.10. Transact Functions are only supported in Manager version 17.10 and later.

Composer

Composer Version	Manager Version	Compatibility Issue
4.1	19.11	If you want to use FTX forms.

Also, check Temenos Journey Platform deprecation schedule.

Product Suite Common Vulnerabilities and Exposures

Journey Platform | All Personas

Every product of the Temenos Journey platform is vigorously tested and validated in accordance with the current list of Common Vulnerabilities and Exposures (**CVE**[1]). As a part of software development cycle, we scan all our software libraries nightly to ensure we deliver as secure the product as possible and to guaranty our clients and applicants safe and risk-free journey when they use form applications built and run using our product suite.

Each CVE report is diligently analyzed against Temenos security policies defining what fixes within what time frames we need to apply for each level of vulnerability. For example, Critical and High level vulnerabilities must be resolved before a release can go into production, whilst Medium level vulnerabilities can be fixed within a month with a minor software release. For more information, see vulnerability levels.

Each release of product suite is accompanied with release notes detailed vulnerabilities resolved or proactively addressed by new software. We also maintain the list of libraries by each product release. This is to provide a clear picture of how we manage current and possible secure risks and threats to all our software users, being it a developer, DevOps or a sales representative.

The following list shows all 3rd party libraries used in our product suite:

- Manager libraries
- Maestro libraries
- Workspaces libraries, which use Manager libraries

Next, learn about Temenos Journey Platform features compatibility.

[1]Common Vulnerabilities and Exposures (CVE) is a list of publicly disclosed computer security flaws, where each security flaw has a CVE ID number assigned to it for further reference.

Journey Platform for Developers

Journey Platform | All Personas

As a developer, you want to build **user experiences**[1] that differentiate your digital products from competitors. Unfortunately, that also means that you need to handle many things that cross from front-end to back-end services and develop features that a front-end can't always manage. That's where the Temenos Journey Platform comes in handy as it gives you the ability to create best in class customer journeys for financial services.

Journey platform provides critical services for customer acquisition and onboarding, including:

- Save & resume
- Receipt generation
- **PII**[2] security
- Bundling of multiple forms into a submission
- Accessibility compliance
- Multiple applicants

Because we take care of developing, testing, and maintaining these complex features, organizations can focus on designing an effective customer journey without dealing with the complexity of the underlying platform.

Each customer journey is made of one or several web applications, also referred to as forms, which are hosted on and managed by Journey Manager. Our platform provides developers with two distinct paths for building web applications:

Journey Maestro Design Environment

Journey Maestro is the customer experience development environment of the Journey platform, purpose-built for the creation of financial customer experiences. It's a design tool that enables trained developers and forms designers to create, visualize, integrate third-party services, test, deploy, and iterate a financial customer journey. Typically, a user action triggers an integration. And often, there is a moderately sophisticated user interface that needs to communicate with the service. These can sometimes be challenging to design, build and test. To simplify this, we provide well-designed and implemented user interfaces for each of our integrations. And while not every integration has a user interface, many do.

---

[1]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.

[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.

To build applications in Maestro, developers are required to have the following skills:

- **JavaScript**[1]
- **AngularJS**[2]
- **HTML5**[3]
- **CSS**[4]
- **LESS**[5]
- Maestro product training

Open UX Environment

The Temenos Open UX is an Software Development Kit (SDK), including **REST APIs**[6], that enables skilled **ReactJS**[7] developers to create complex web applications that run on Manager. ReactJS developers, with their own **IDE**[8] and development tools, may choose our Open UX as the most efficient path to creating different user experiences. Banks with established UX development teams, especially those using JavaScript frameworks such as ReactJS, can use the services of Open UX to focus on their customer onboarding experience, while treating infrastructure as a "solved problem", eliminating much of the routine complexity accompanying UX development. Open UX is transparent to existing development pipelines, allowing bank teams to reuse and retain their investment in UX, design, and development environments while taking advantage of the Jbourney platform services.

To build Open UX web applications, developers are required to have the following skills:

---

[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.
[2]AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write. For more information, see http://angularjs.org/
[3]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.
[4]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.
[5]LESS is a dynamic preprocessor style sheet language that can be compiled into Cascading Style Sheets (CSS) and run on the client side or server side.
[6]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.
[7]ReactJS is a JavaScript library for building user interfaces.
[8]Integrated Development Environment.

- ReactJS
- IDEs
- REST API
- UX development expertise

Integration

The Journey platform advantage is its capability to integrate with a wide range of third-party services. There are two types of integrations:

Native Integration

It uses a combination of Groovy scripting language, plus several libraries and services that the Journey platform provides. Groovy is a scripting language that most developers will find simple to learn and powerful to use. The libraries provided alongside Groovy make it easy to invoke third-party services using SOAP, REST or HTTP, perform data transformations, and more.

Our Journey platform also comes with a wide range of pre-built third party integrations available from the Exchange.

3rd Party Integration

Instead of doing the bulk of the computational work in Groovy and having it run within the Journey platform, you can create the logic in a third-party integration and simply invoke it from your application during the appropriate life cycle event. The integration platform will perform the service invocations, data translations, and so on, using its capabilities. There are extensive Transact Fluent API that can be called if the integration needs to call back into the platform to retrieve or update transaction information.

Developers depend on the built-in features for routine services, pre-built integrations, and apply their knowledge to rapidly extend the functionality to meet their specific needs. In this case, they require Java and **Groovy**[1] skills.

Next, learn about the complete range of software development technologies and programming languages you can use in our platform.

[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.

Software Development Technologies

Journey Platform | All Personas | 5.1 & Higher

These technologies are used across the entire Temenos Journey Platform.

This topic lists the software development technologies and programming languages used in Journey Maestro.

A Maestro form is a dynamic HTML page designed to present information to users, guide them through the form to collect the required inputs, and submit data to Journey Manager for further processing. The design and purpose of a Maestro form depends on the organization that builds and implements it. The organization is free to use their preferred software development processes, and Maestro supports the following popular software development technologies and programming languages to implement forms. They are listed below in the order of importance for a form builder.

AngularJS

A Maestro form builder can utilize **AngularJS**[1] framework to build dynamic forms, which implement business logic. One example is a script that retrieves data from Manager or invokes a service to retrieve data from a third-party system. Experience with AngularJS is necessary to create custom native components.

JavaScript

A Maestro form builder can implement complex business rules to enhance a form using the **JavaScript**[2] language.

LESS

Maestro allows a form builder to use **LESS**[3] (Leaner Style Sheets) to create dynamic **CSS**[4] to implement a form. For more information, see the CSS section below.

---

[1] AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write. For more information, see http://angularjs.org/

[2] 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.

[3] LESS is a dynamic preprocessor style sheet language that can be compiled into Cascading Style Sheets (CSS) and run on the client side or server side.

[4] 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.

HTML5

Maestro forms are structured and built using **HTML 5**[1]. HTML is used as the standard markup language for creating Web pages, while HTML 5 describes the structure of a Maestro form.

JSON

A Maestro form builder can use **JSON**[2] format to store form data in Maestro forms or exchange data between different systems, such as Manager, and Maestro. For example, to transfer forms to another system, use the **Download JSON** feature in Maestro.

CSS

CSS[3] is used to define styles for a Maestro form that includes the design, layout, and variations in the form's display on different devices and screen sizes. A template designer primarily looks after CSS of Maestro forms by implementing the branding and styling of the form's assets. CSS can then be applied to a form template, individual components, and any other asset associated with a Maestro form.

CSV

Maestro stores all translation files as **CSV**[4] files, which can be edited by language translators to add new languages to a Maestro form.

XML

Maestro uses **XML**[5] to store and transfer form data, such as form submissions. However, JSON is now replacing XML to perform these functions.

---

[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.
[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.
[5]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.

Next, learn about the Maestro dashboard.

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.

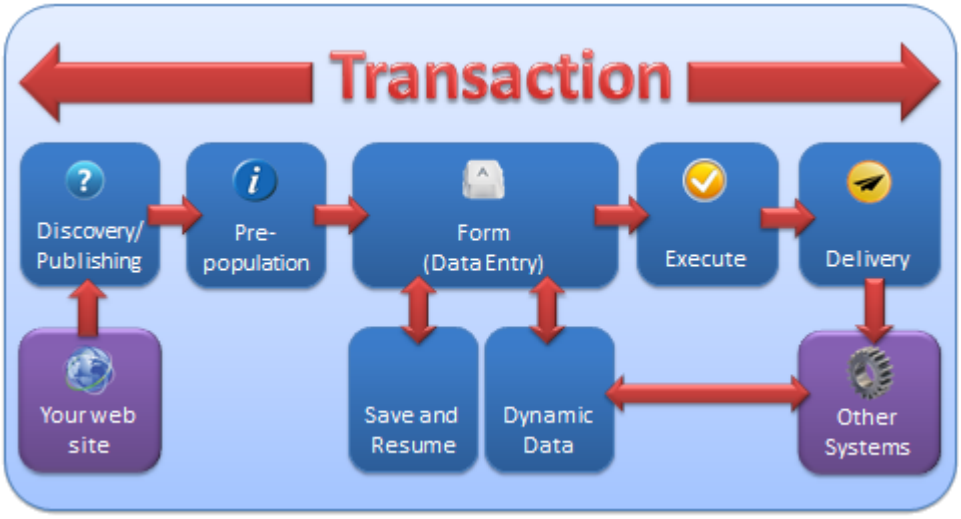
- Integration areas occur at different stages within the interaction between a user and the organization.
- Within each integration area, there are different integration implementations, to support different technologies used by different organizations.

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.

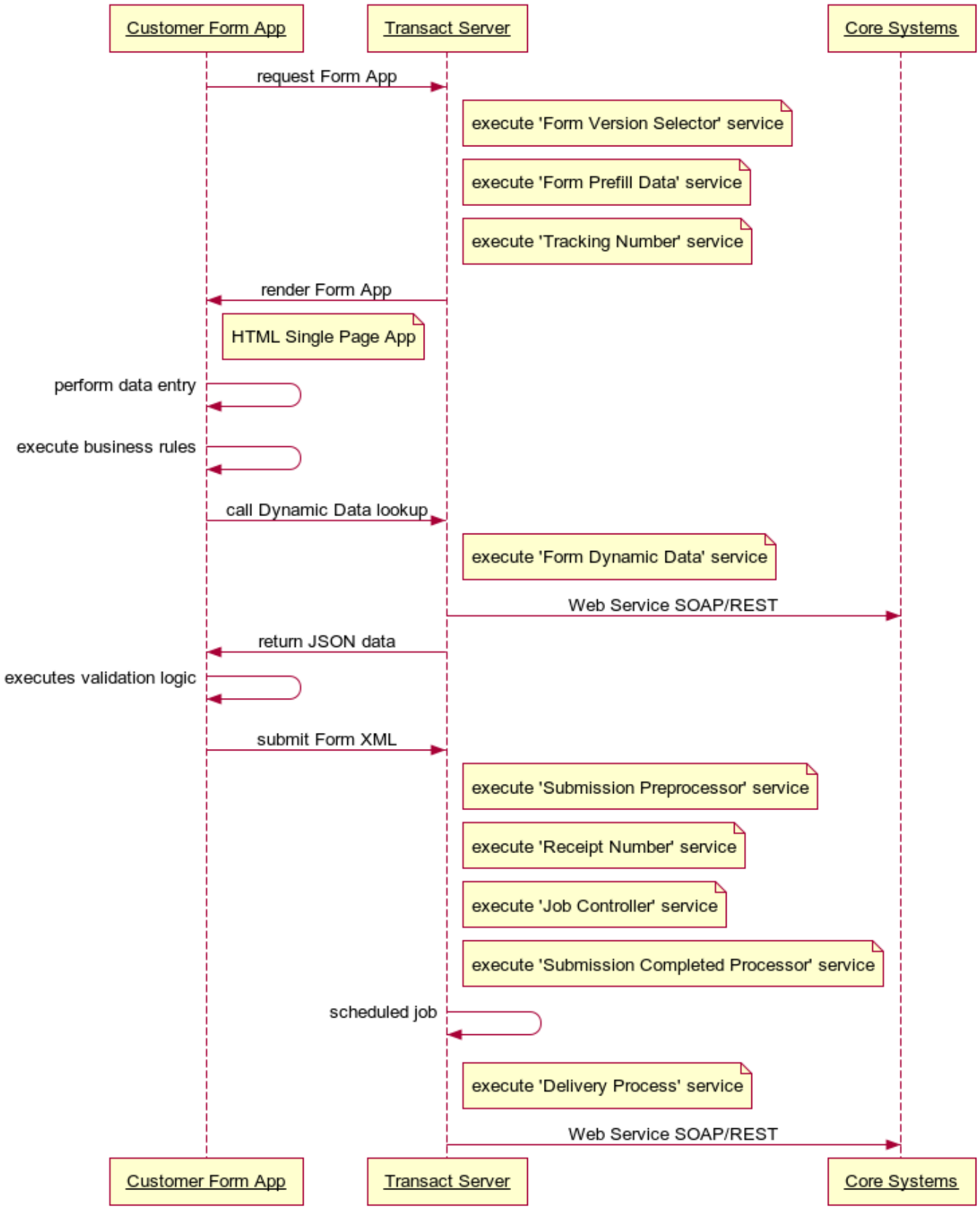
Because of the above, we generally recommend the following:

- 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.

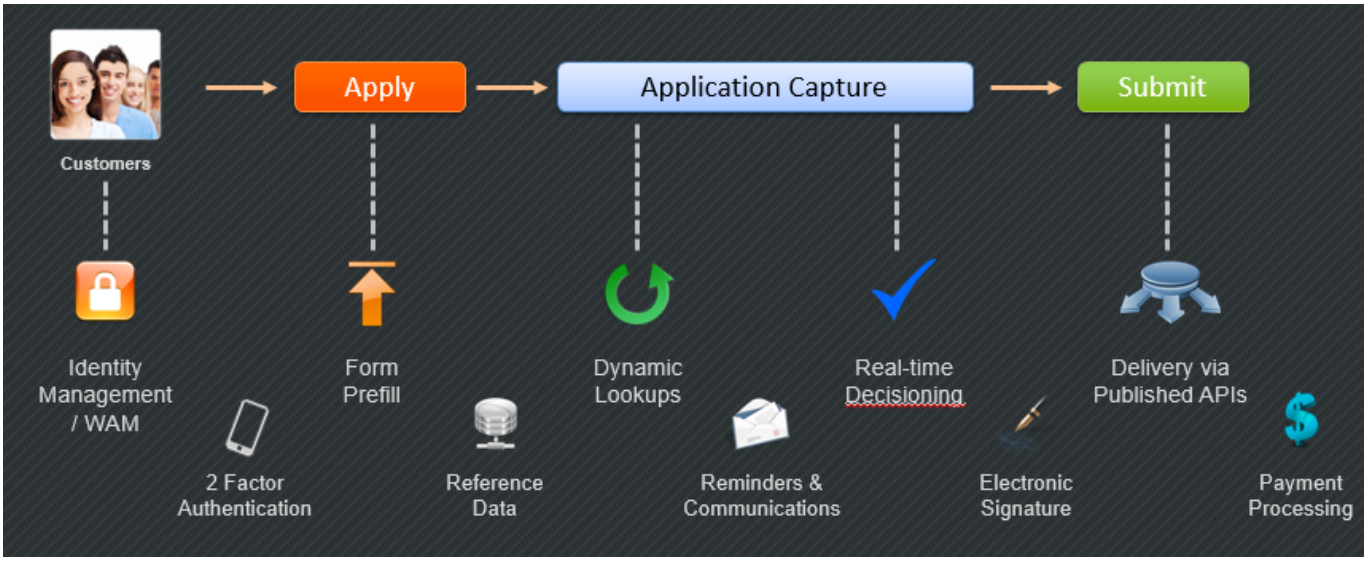
In some cases authenticated end users may be required for business reasons, and Transact does support this extremely well - but bear in mind that it may affect your time and cost of implementation.

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.

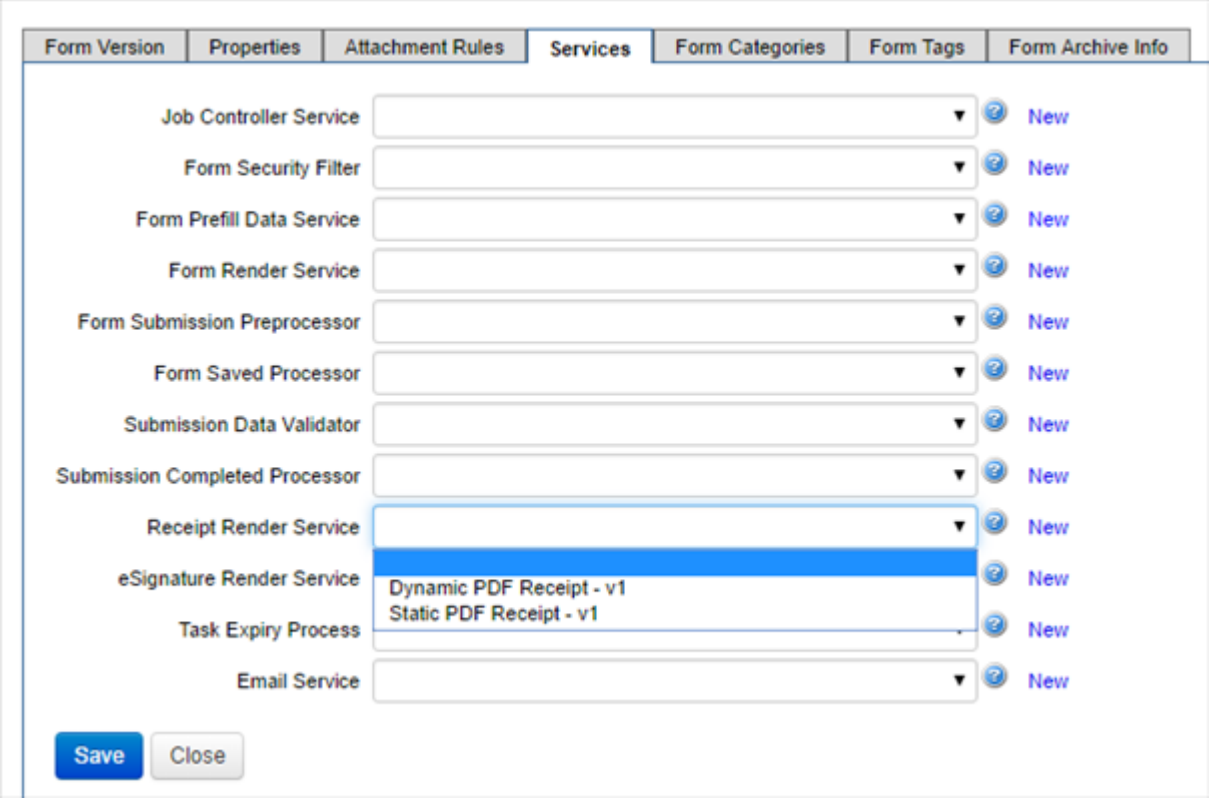

Customer Form App
Transact Server
Core Systems
request Form App
execute 'Form Version Selector' service
execute 'Form Prefill Data' service
execute 'Tracking Number' service
render Form App
HTML Single Page App
perform data entry
execute business rules
call Dynamic Data lookup
execute 'Form Dynamic Data' 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.

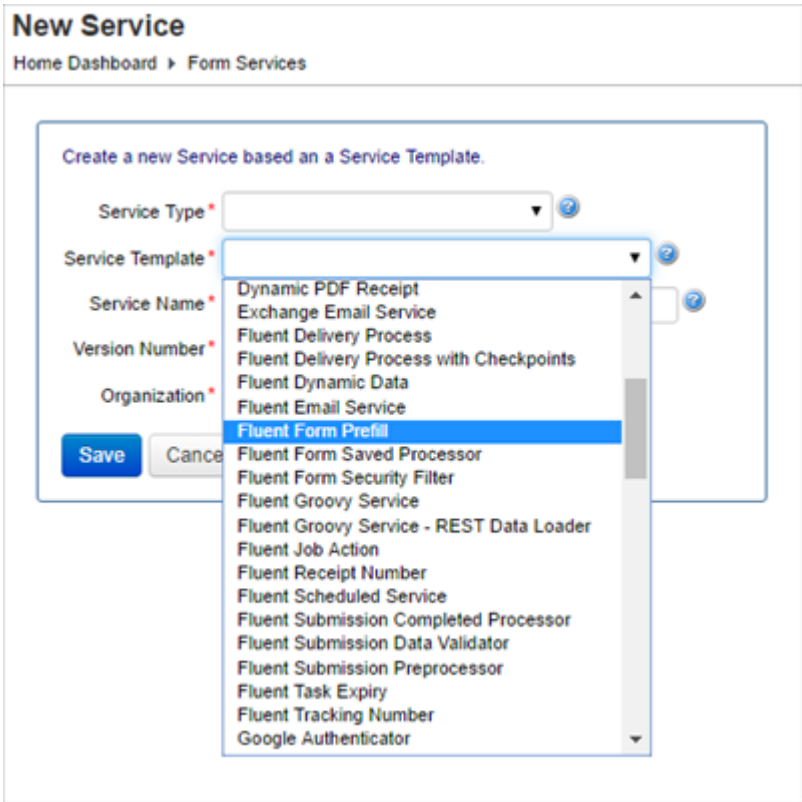
Manager provides extension points at many of these life-cycle events to allow integration between Manager and these other systems. The example of these extension points is shown below.



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)
}
```

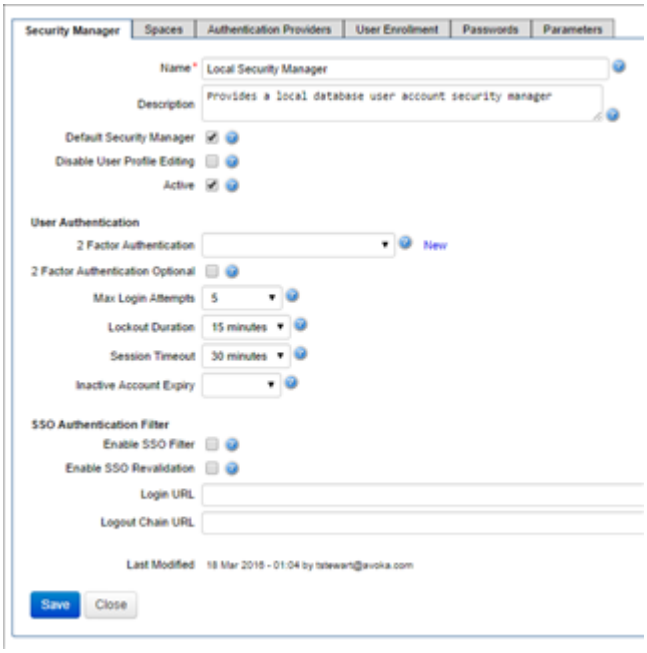
This illustrates how easy it is to invoke these external services. Of course, more sophisticated services can be developed.

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.

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.	
TMRDC	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.

Dynamic Data Service	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 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.

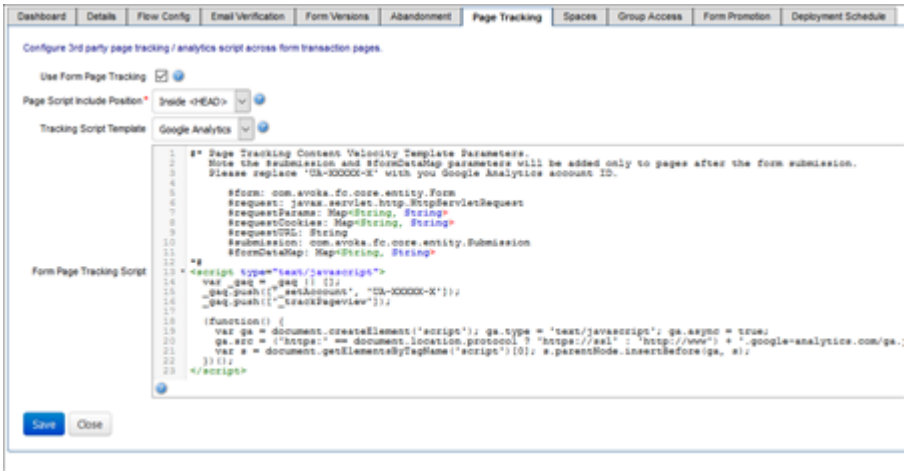
Google Analytics Virtual Page Integration

Journey Platform | All Personas

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

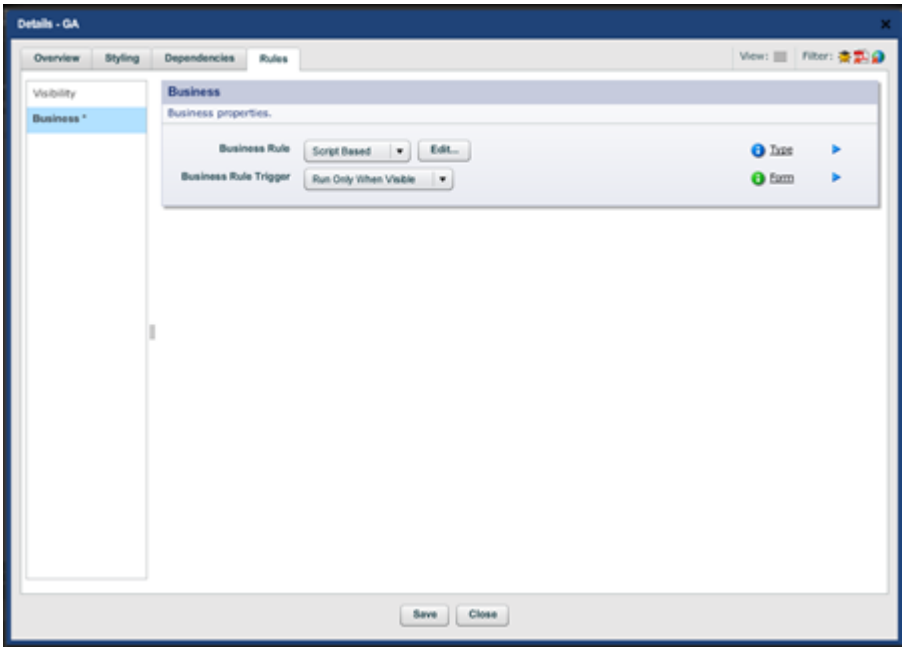
Add GA Code to Head of Page

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.

Continuous Integration Overview

Journey Platform

Continuous Integration[1] (CI) is an essential ingredient in a software development process, which helps improve software quality, provide reliable, repeatable, and well designed automated processes to save time, reduce the chance of human errors, and identify errors early. The Temenos Journey Platform provides the services and tools to facilitate in Continuous Integration of Journey applications.

Continuous Integration is the frequent building and testing of new code changes, assisted by software tools that automate the process. It is often referred to as a pipeline. A typical CI environment includes the following components:

- Version control systems: Git, Subversion, Visual Studio Team Services
- Build automation tools: Ant, Gradle
- Dependency management tools: Maven
- CI servers: Jenkins, TeamCity, Bamboo, GitLab CI
- Task automation tools: Ant, Maven, Gradle, Gulp, Make
- Unit testing frameworks: JUnit
- Mocking libraries: Mockito
- Deployment and configuration automation tools: Ansible, Chef
- In-house APIs: the REST APIs built into Journey Manager

CI environments are implemented using software tools that work alone or together to support the creation of pipelines. Implementation is usually achieved via configuration, rather than a specific development. Sometimes, as is the case with our Transact REST APIs, you might have to provide support services to enable application specific deployment or configuration.

There is often crossover in functionality between tools, which blurs the functional boundaries. In GitLab, for example, version control, build and CI services are all available in a single-source system.

We don't prescribe the use of specific tools to apply CI techniques to the Journey platform application development process. Customers are free to configure their existing Ant-compatible CI systems with Ant tasks from the Transact Fluent SDK to provide the underlying automation.

For customers who don't have existing CI infrastructure, we recommend the following tools:

---

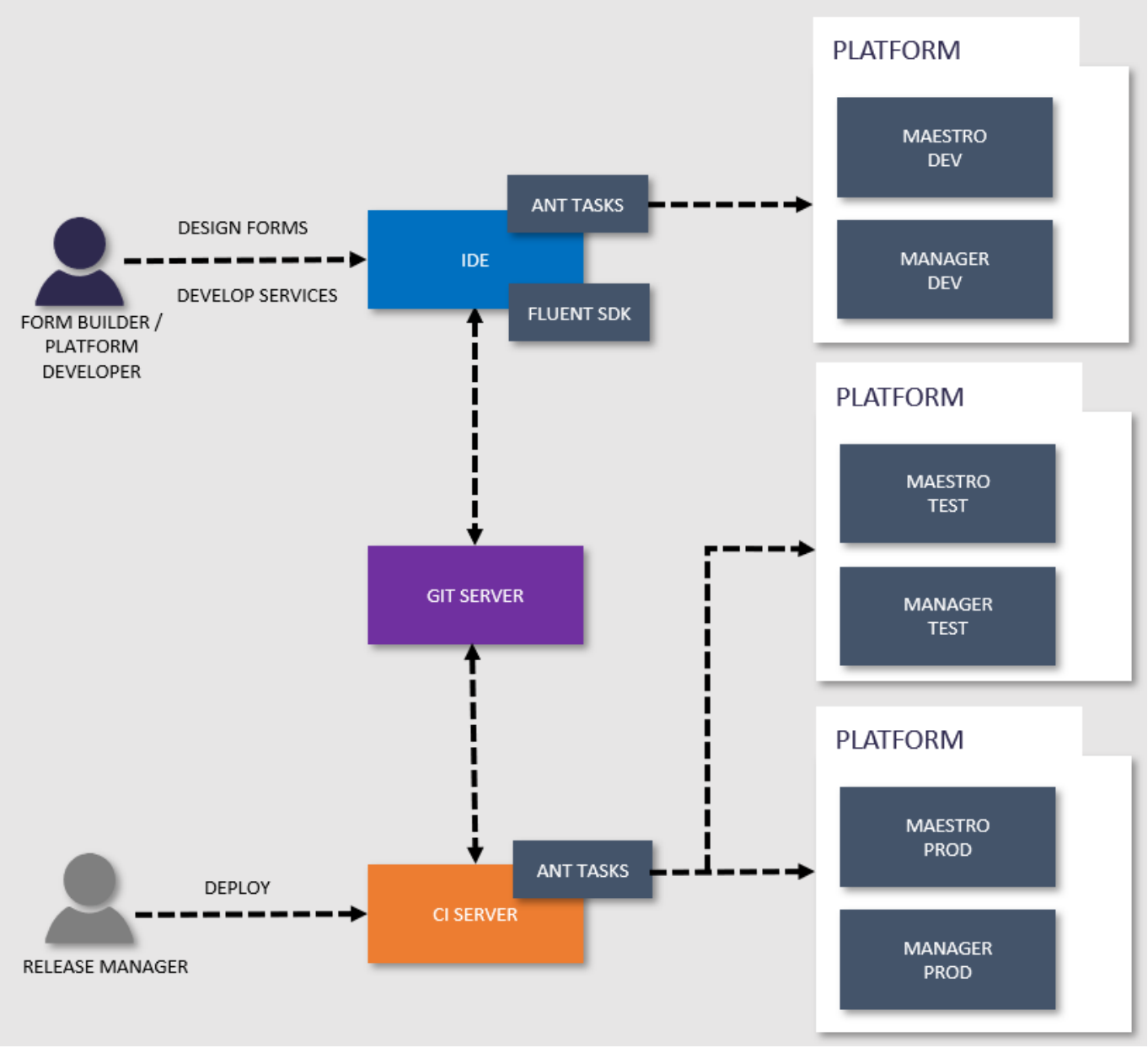
[1] Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

- GitLab - for source code version control.
- Maven - for dependency management of third-party content, and potentially for storing output from the build system.
- Jenkins - for integration to version control systems and orchestration of builds, dependency management and CI services.

Journey Manager supports Continuous Integration practices through the following three key features:

- Application packages that contain primary deployable artifacts, such as forms and services.
- Transact REST APIs references that include services to facilitate the deployment of application packages.
- Transact Fluent SDK which includes Ant tasks to leverage on the REST APIs

Journey platform implements CI model as shown below.



In this model, application developers, using Maestro, Manager, and other IDEs, collaborate in a development environment. They work with assets including forms, templates, organization properties, delivery channels, services, and service connections. They configure JSON files to define the contents of application packages. Changes in the development environment are committed to the version control system, from where they can be checked out, packaged into application packages and deployed to other environments by Ant tasks interacting with the Transact REST APIs. This is summarized below:

1. Check assets and supporting files into and out of the version control system
2. Trigger CI pipelines
3. Execute Ant tasks that form part of the Transact Fluent SDK
4. Package assets into application packages
5. Publish the application packages to Manager services exposed via the Transact REST APIs

Continuous Integration Workflow

The Manager CI workflow involves the following steps:

1. Download and install the Transact Fluent SDK. The Transact Fluent SDK is a part of the SDK download which you can find in the **Release Notes** of each Journey Manager release.
2. Define an application package:
   - Create configuration folder structure.
   - Obtain form version archive files from Manager, if required.
   - Create JSON configuration files to describe application package contents.
3. Configure your CI server with Ant tasks from the Transact Fluent SDK.
4. Create your CI workflow:
   - Build the application package.
   - Deploy the application package to one or more servers.
   - Run the remote tests on deployed servers.
5. Configure the CI server to work with source control.
6. Check the application package configuration files and resources into source control.
7. Check the state of the CI workflow on the CI server.

Next, learn about version control systems.

CI Version Control Systems

Journey Platform

A **version control system**[1] is the primary entry point for getting your new code into a system. A Continuous Integration pipeline starts with detecting changes to source code, which are triggered by source code check-ins or commits. These changes can also be detected by polling the version control system status or by broadcasting change events. You can choose which method to use depending on your preferences and the capability of your version control systems. Once triggered, the CI pipeline commences collecting all required assets from source control system. After that, the build system is triggered to gather additional dependencies, create the deployment artifacts, and run initial tests.

As with any software development project, the version control system, used for Journey Manager application development, should hold assets required to rebuild entire projects from scratch, including:

- Source code: Groovy, Java, and JavaScript files.
- Configuration: JSON, XML, and property files.
- Object Libraries: 3rd party libraries, in the absence of a dependency management system such as Maven.
- Resources: images, icons, and other visual components.
- Custom content: proprietary format content, such as Journey Maestro resources.
- Scripts: shell scripts to create folders, databases, seed tables, configure services, and execute any other actions required to create a development environment.

For more information on how to store content GitLab, see GitLab CI tutorial.

The version control system should not include unprotected sensitive and credential information where it can be easily viewed. User IDs and, especially, passwords do not belong in unprotected version control system. Either don't store such content in the first place or use encryption or vault technologies to secure the content. For more information on how to protect your sensitive content, use the following links:

- https://github.com/StackExchange/blackbox
- https://github.com/AGWA/git-crypt
- http://docs.ansible.com/ansible/latest/playbooks_vault.html
- https://github.com/hashicorp/vault
- https://square.github.io/keywhiz/

Next, learn about CI servers.

---

[1] Source Code Control System (SCCS) is a version control system designed to track changes in source code and other text files during the development of a piece of software. This allows the user to retrieve any of the previous versions of the original source code and the changes which are stored.

CI Servers

Journey Platform

A CI server allows you to build artifacts, run the unit tests periodically or after every commit, and send the results to developers. This makes it a critical part of a Continuous Integration pipeline. You typically need to configure your CI server with the following software:

- Java JDK
- Ant and Ant tasks
- The Transact Fluent SDK **`build.xml`** file.
- Version control system connection
- Dependency management system

You might need to configure triggers from the version control system to the CI server. Integration between GitLab and Jenkins CI server is described in here. For more information on Jenkins CI server configuration, see Jenkins CI tutorial.

Next, learn about CI build systems.

CI Build Systems

Journey Platform

A build system in the Continuous Integration pipeline is responsible for converting source assets into deployable artifacts. This could involve compiling source code into executable object code, or some other form of conversion from an input representation to an output object.

A build can be triggered by one of the following actions:

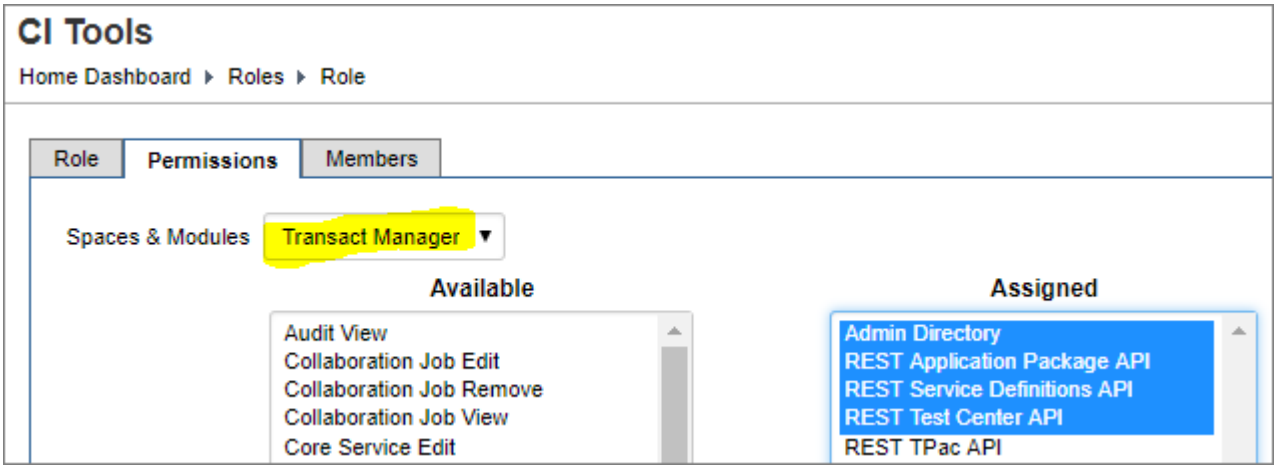
- Manual - run a script at a command prompt, or click a button on a web UI.
- Scheduled - a periodic trigger that runs at specified times.
- Event based - a configured event happens, such as a code check-in or commit to a version control system.

A build system can be a standalone, or integrated into a version control system, dependency management system, or the CI system.

The CI build can have a deployment configuration that uses Ant's **`ant-deploy`** task to deploy the built package. You need to ensure that Manager has the following permissions configured for the CI Tools user:

- Admin Directory
- REST Application Package API
- REST Service Definitions API
- REST Test Center API
- REST TPac API – optional, if you use TPacs

The organization, configured in your build.properties file, should be assigned to the CI Tools user to allow this user deploy and execute services under this organization.



Incorrect permissions can result in the HTTP 403 error thrown by the Ant task.

Next, learn about CI dependency management systems.

CI Dependency Management Systems

Journey Platform

A dependency management system is part of a Continuous Integration pipeline that is responsible for storing and managing 3rd party artifacts required to build application packages. A dependency management system, such as Maven, is backed by a repository system, where you can store build outputs. For more information on Maven, see https://maven.apache.org.

Next, learn about CI application packages.

CI Application Packages

Journey Platform

Application packages are the output of the Continuous Integration pipeline that can be deployed into Journey Manager servers. Application packages are ZIP files containing a folder structure and files that define:

- Delivery channels
- Forms
- Properties
- Service connections
- Services

An application package is created by the Ant **app-package** task. It contains a JSON definition file (shown below), which contains references to other definition files.

Main JSON

app-package-def.json

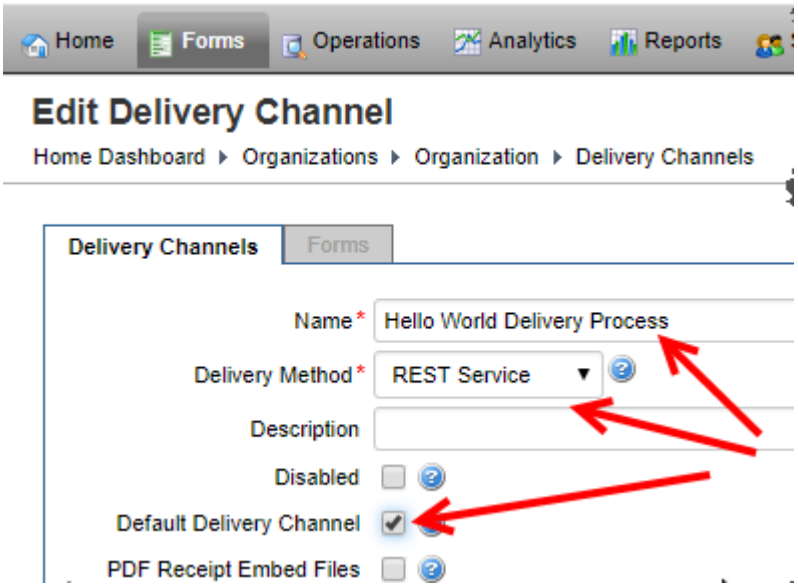
```
{
  "name": "Hello World App package",
  "description": "TODO...",
  "clientCode": "maguire",
  "delivery": [
    "delivery/rest-service-delivery-def.json",
    "delivery/trash-can-delivery-def.json"
  ],
  "forms": [
    "forms/CCA-MAESTRO-AP/cca-maestro-form-def.json",
    "forms/customer-enquiry-ap/customer-enquiry-form-def.json"
  ],
  "properties": [
    "properties/property-def.json",
    "properties/property-de-def.json",
    "properties/property-def1.json"
  ],
  "connections": [
    "connections/connection-def.json",
    "connections/connection-def2.json"
  ],
  "services": [
    "services/helloworld/service-def.json"
  ]
}
```

Delivery Channels JSON

delivery/rest-service-delivery-def.json

```
{
    "name": "Hello World Delivery Process",
    "deliveryMethod": "REST Service",
    "defaultChannel": true,
    "retryDelayMins": 15,
    "serviceName":"Hello World Delivery Process"
}
```

This delivery service definition appears in Manager as shown below.



Forms JSON

forms/CCA-MAESTRO-AP/cca-maestro-form-def.json

This definition causes a form to be uploaded into Maestro, and subsequently be visible in Manager.

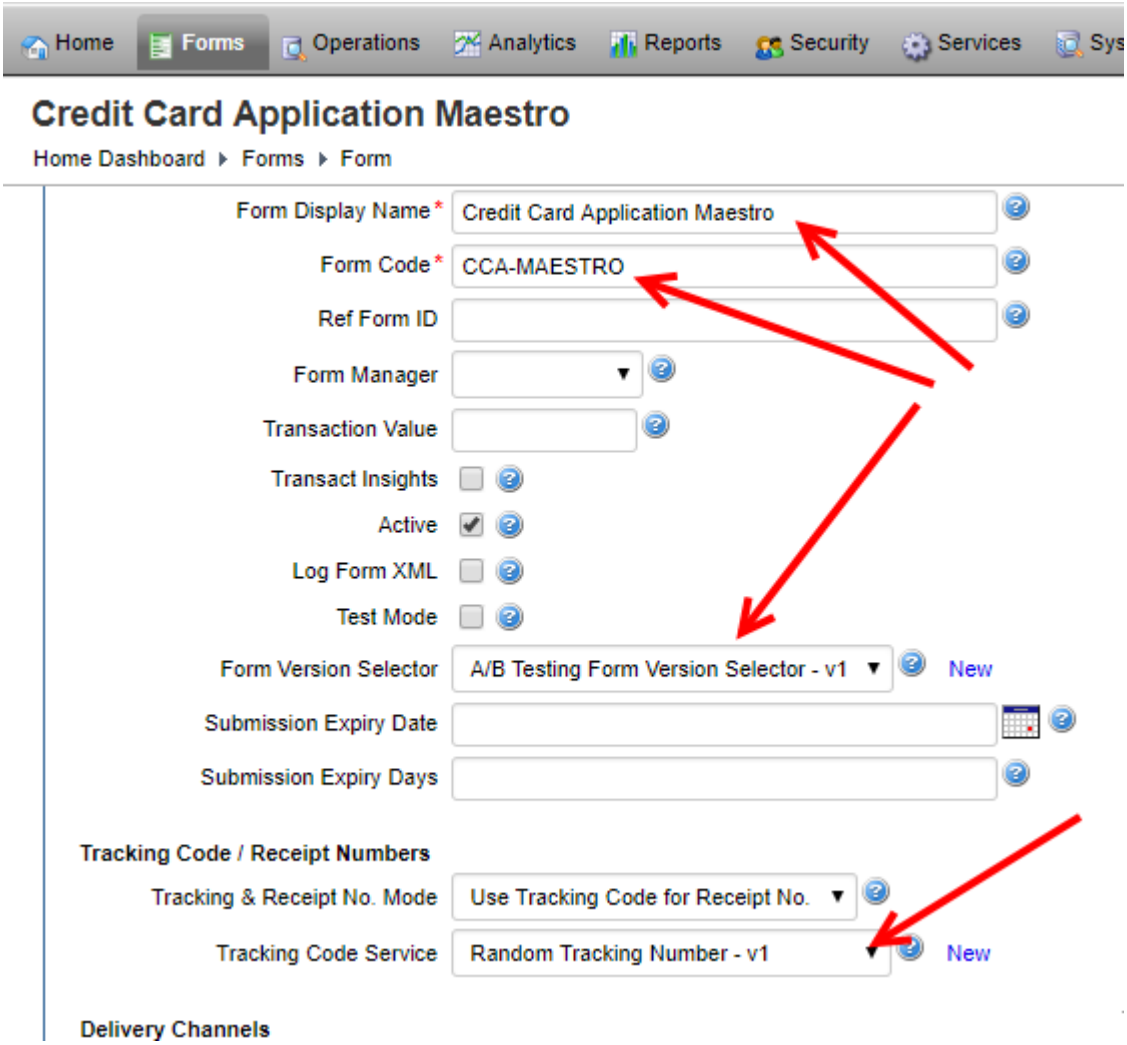
```
{
   "name":"Credit Card Application Maestro",
   "formCode":"CCA-MAESTRO-AP",
   "currentFormVersionNumber":"9",
   "formVersionSelector":{
         "serviceName":"A/B Testing Form Version Selector",
         "serviceVersion":1
   },
   "trackingCodeService":{
         "serviceName":"Random Tracking Number",
         "serviceVersion":1
   },
   "defaultChannels":{
      "productionDelivery":"Hello World Delivery Process",
      "abandonDelivery":"Hello World Delivery Process",
      "validationFailureDelivery":""
   },
   "dataRetention":{
      "savedDays":30,
      "finishedDays":5
   },
   "emailConfirmation":"Confirmation and PDF Receipt",
   "formSpaces":[
      {
         "name":"Maguire",
         "anonAccess":true,
         "authAccess":true
      }
   ],
   "formVersions":[
      {
         "versionNumber":"9",
         "formDataEncryption":true,
         "formType":"Maestro Form",
         "formVersionFile":"./form-version-Credit_Card_Application_Maestro-all-2017-06-
26.zip",
         "properties":[
            {
               "name":"Form Description",
               "description":"Form Description",
               "dataType":"String",
               "value":""
            },
            {
```

```
               "name":"Locale De",
               "description":"Form Description",
               "dataType":"JSON",
               "valueFile":"locale-de.json"
            }
         ],

         "formDataConfig":{
            "contactEmailXPath":"//ContactDetails/EmailAddress"
         }
      },
      {
         "versionNumber":"8",
         "formDataEncryption":true,
         "formType":"Maestro Form",
         "formVersionFile":"./form-version-Credit_Card_Application_Maestro-all-2017-06-
26.zip",
         "properties":[
            {
               "name":"Form Description",
               "description":"Form Description",
               "dataType":"String",
               "scope":"Form",
               "value":""
            }
         ],

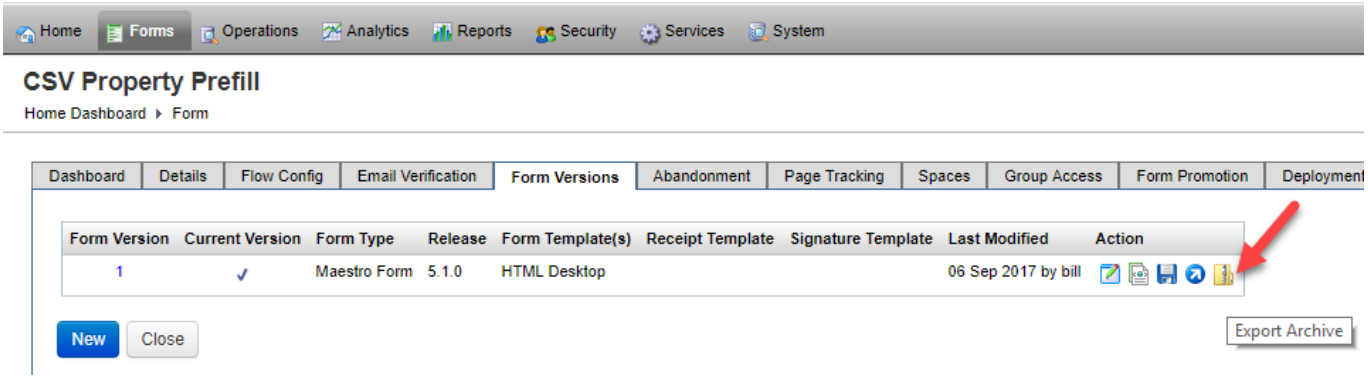
         "formDataConfig":{
            "contactEmailXPath":"//ContactDetails/EmailAddress"
         }
      }
   ]
}
```

The form appears in Manager as shown below.



When specifying forms, form version archive files need to be exported from Manager, saved locally, and checked into source control. They can then be referenced by a **`formVersionFile`** property in the **`formVersions`** array of the form definition JSON fil (shown above).

You can export a selected form in the Manager as shown below.



The **`emailConfirmation`** property, if specified, must be one of the following values:

- None
- Confirmation
- Confirmation and PDF Receipt
- Link to Receipt

forms/CCA-MAESTRO-AP/locale-de.json

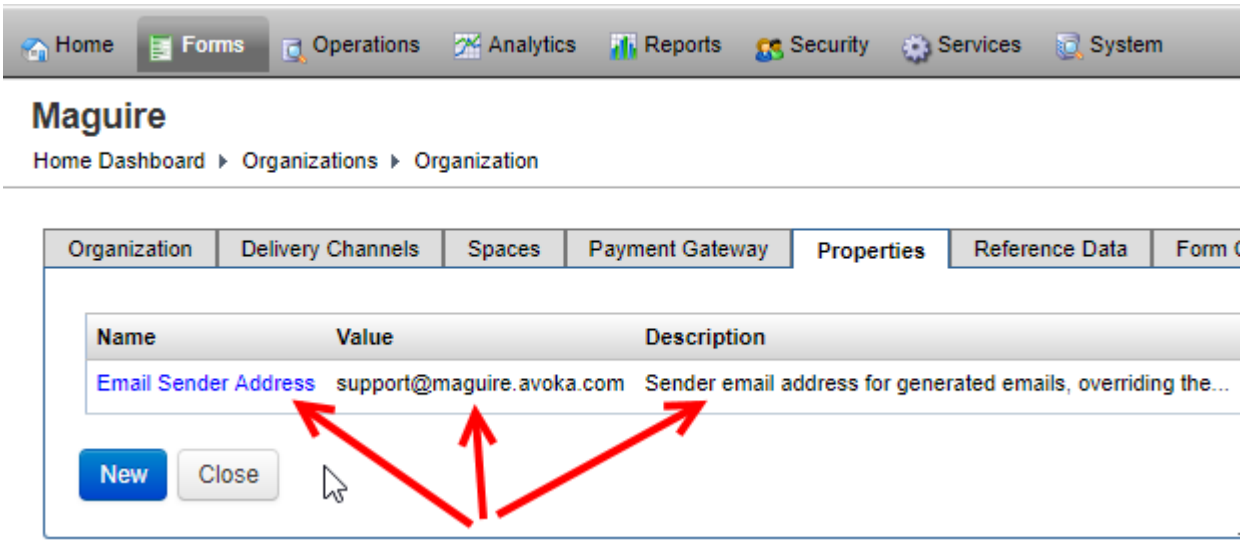
```
{
    "key.de": "Brüderlich zusammenhält"
}
```

Properties

properties/property-def.json

```
{
    "name": "Email Sender Address",
    "description": "Sender email address for generated emails, overriding the global
email sender address.",
    "scope": "Form",
    "dataType": "String",
    "value": "support@maguire.avoka.com"
}
```

This property appears in Manager as shown below.

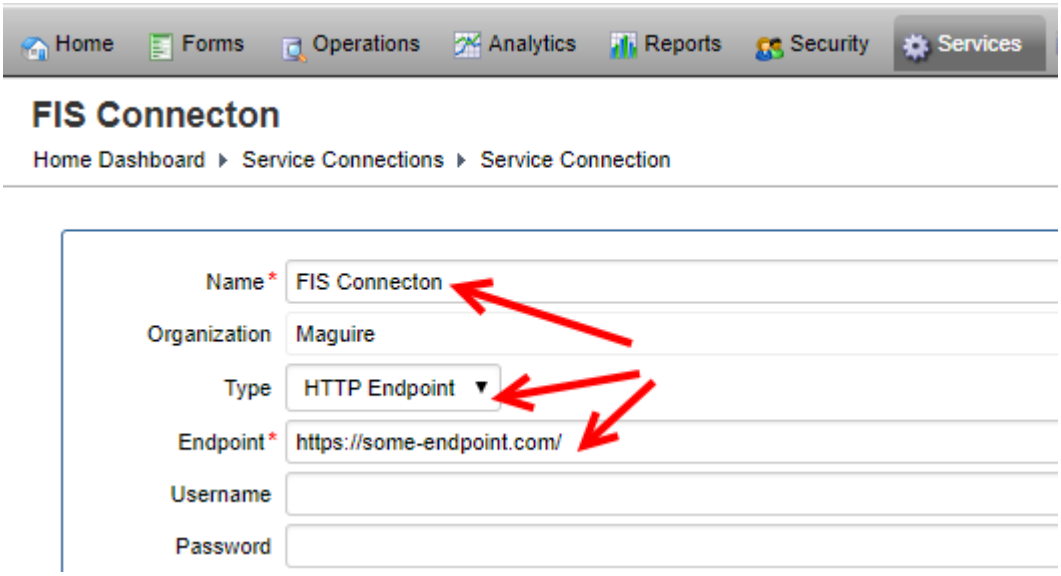


Service Connections

connections/connection-def.json

```
{
   "name": "FIS Connection",
   "type": "HTTP Endpoint",
   "endpoint": "https://some-endpoint.com/",
   "username": "username",
   "password": "password",
   "dataFile": ""
}
```

After installation of the application package, the above service connection details will be available to see in Manager:



Services

services/helloworld/service-def.json

```
{
  "name": "Hello World Delivery Process",
  "description": "",
  "type": "Delivery Process",
  "version": 1,
  "tmMinVersion": "5.0.0",
  "parameters": [
    {
      "name": "groovyScript",
      "filePath": "HelloWorld.groovy",
      "fileIncludes": [],
      "bind": true,
      "required": false,
      "clearOnExport": false,
      "readOnly": false
    },
    {
      "name": "Unit Test Script",
      "filePath": "HelloWorldTest.groovy",
      "fileIncludes": [],
      "bind": false,
      "required": false,
      "clearOnExport": false,
      "readOnly": false,
      "unitTest": true
    },
    {
      "name": "Help Doc",
      "type": "HTML",
      "filePath": "service-help.html",
      "bind": false,
      "required": false,
      "clearOnExport": false,
      "readOnly": false
    }
  ]
}
```

services/helloworld/HelloWorld.groovy

```
import com.avoka.core.groovy.GroovyLogger as logger
import com.avoka.tm.vo.*
import javax.servlet.http.*

class HelloWorld {

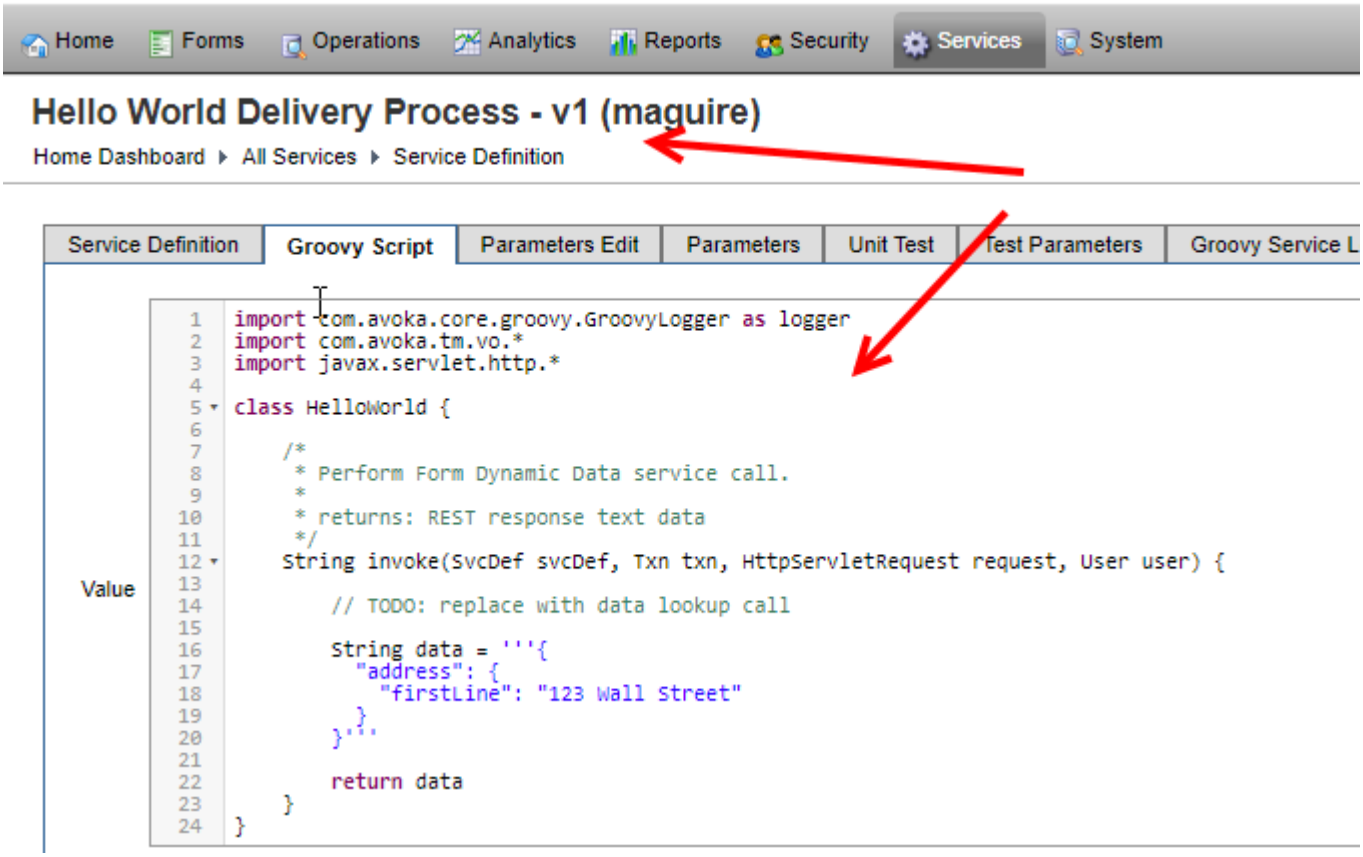
    /*
     * Perform Form Dynamic Data service call.
     *
     * returns: REST response text data
     */
    String invoke(SvcDef svcDef, Txn txn, HttpServletRequest request, User user) {

        // TODO: replace with data lookup call

        String data = '''{
          "address": {
            "firstLine": "123 Wall Street"
          }
        }'''

        return data
    }
}
```

The two files above will provision Manager with a new service with the following Groovy script.



Next, learn about application package JSON definitions.

Support

Welcome to our support site. We are committed to supporting our customers who have Journey platform solutions installed in the cloud or on premise.

Learn more about our Support Team, it's policies, procedures, and processes.

You can raise a technical support request for: product issues, applications, account access, or infrastructure related queries. Or track existing support tickets and view existing new feature requests.



SUPPORT HANDBOOK

keyboard_arrow_right Support Handbook

keyboard_arrow_right Reset Support Portal Password

keyboard_arrow_right New Product Releases

keyboard_arrow_right For Developers

SUPPORT PORTAL

keyboard_arrow_right Raise a Support Request

keyboard_arrow_right View Existing Support Requests

keyboard_arrow_right Raise a New Feature (Idea) Request

keyboard_arrow_right View Existing New Feature (Idea) Requests

POLICIES

keyboard_arrow_right End Of Life

keyboard_arrow_right Deprecation Schedule

RESOURCES

keyboard_arrow_right Release Notes

keyboard_arrow_right Q&A forum

keyboard_arrow_right Announcements

keyboard_arrow_right Articles

SYSTEM REQUIREMENTS

keyboard_arrow_right Journey Platform

keyboard_arrow_right Journey Manager

keyboard_arrow_right Journey Maestro

keyboard_arrow_right Journey Platform Forms

keyboard_arrow_right Mobile Browsers

LEGACY SYSTEM REQUIREMENTS

keyboard_arrow_right Transact Composer

keyboard_arrow_right Transact Integration Gateway

keyboard_arrow_right TransactField App

keyboard_arrow_right Legacy Browsers

Journey Platform Support Handbook

Journey Platform | All Personas | 5.1 & Higher

The Support Handbook contains the following sections:

- Roles and responsibilities
- Support definitions
- Severity level definitions
- Response times and update targets
- Contacting Customer Care
- Lifecycle of a support ticket
- Escalating a support ticket
- Product release definitions
- Getting more information

Thank you for your trust in Temenos as your partner in your Digital Customer Acquisition Journey. We are honored to be part of your business.

The Support Team's sole focus is your success in achieving your business goals.

Our passion is to remove roadblocks and other challenges on your journey. We intend to always be there when you need us and be your trusted advisor.

Our intention is to provide all the information required to engage with our Support Team, it's policies, procedures, and processes.

We continuously work on improving our services, and I encourage you to engage with the Support Team as we value your comments and feedback.

Roles and responsibilities

Following is a general description of the personnel involved in providing Journey Platform support under its various offerings.

Role	Will Do
Temenos Support Engineer	• Understand the business impact of the customer's issue upon logging of a support ticket • Provide technical expertise • Troubleshoot and resolve the customer's issue • Provide status updates throughout the resolution process
Temenos Customer Success Manager	• Understand the customer's requirements and business goals • Recommend solutions with Journey Platform technology that meets requirements • Communicate product updates and important news
Temenos Customer Success Architect	• Advanced knowledge of the customer's solution • Guide and foster strategic product discussion • Provide technical expertise • Troubleshoot and resolve the customer's issue • Provide status updates throughout the resolution process
Temenos Hosted Platform Engineer	• Monitor cloud-hosted environment • Troubleshoot and resolve cloud-hosted related issues • Conduct upgrade and updates on cloud-hosted platform
Temenos Regional Support Manager	• Ensure highest degree of technical know-how in department • Keep apprised of high-priority customer issues • Point of contact for customer escalation
Customer's Named Support Contacts	• Be trained and certified in Journey Platform products • Train internal users to appropriate degree on Journey Platform products • Communicate business impacts of any technical issues appropriately • Collect diagnostics and other required information in submitting cases • Engage technical and management resources appropriately • Provide equal resource availability

Support definitions

Platform Support

Definition

Platform Support is included in your annual platform subscription. Services differ depending on the support package purchased. Additional services may be available depending on the support package you purchased.

Platform Support is there to assist if the platform malfunctions due to a product defect or issue with the hosting services.

Example of Platform Issues

A platform issue is one where the expected behavior of the module is not observed. For example, a form won't render in Journey Manager, or a form field is not behaving as expected, submission is not delivered, etc.

Out of Scope

- Development assistance such as web application (form) development, custom services and scripts
- Product Training
- Third Party integration (except Mitek and Yodlee)
- New feature request

Development Assistance

We have self-service and guided options to help you if you need development assistance. Instructional documentation and online training (authentication required) is available on our resources website. Alternatively, you can contact your Account Executive to discuss commercial requirements for Temenos Development Assistance Services and product training.

New Feature Requests

All new feature requests should be logged in the Support Portal as an 'Idea'.

Severity level definitions

Our team uses severity to define the impact on your business and the urgency required for resolution. Below are summaries of the severity levels for Platform Support; please consult your contract with Temenos or contact your Temenos Account Executive for further details.

Severity 1

- Problem with the platform that results in continued unavailability of critical production operations with no workaround available.
- It has affected, or is reasonably likely to affect, the entire user community.
- Example: Platform is unresponsive, non-operational.

Severity 2

- Problem with the platform that results in continued degraded or intermittent performance of critical operations in production, or the continued unavailability or degraded or intermittent performance of operations in a test environment, if no reasonable workaround is available.
- It has affected, or is reasonably likely to affect, multiple users.
- Example: Email queue is stuck in Manager; a service is stopped and won't restart in Manager.

Severity 3

- Problem with the platform that results in impairment of non-critical operations in any environment.
- Example: Third-party integration is down and is affecting the form.

Response times and update targets

Response Times

When creating a support ticket, you can expect initial contact from a Support Engineer within the following targeted response times, depending on the Support Package you have purchased.

Severity level	Core	Extended	Premium
Severity 1	3 business hours	2 hours (24x7)	1 hour (24x7)
Severity 2	1 business day	6 business hours	4 business hours
Severity 3	2 business days	1 business day	1 business day

Update Targets

Target communication updates shall be governed by the following default times unless other statuses are set.

Severity level	Core	Extended	Premium
Severity 1	Daily	Every 2 hours (24/7)	Every 1 hour (24/7)
Severity 2	Weekly	Every 3 business days	Every 2 business days
Severity 3	Bi-Weekly	Weekly	Weekly

Business Hours are 9 AM-5 PM Mon-Fri MST (MT) AEST (AET) GMT (BST)

Business Hours Definition

Our centers and business working hours are:

Australia - Sydney	United Kingdom - London	Colorado USA - Broomfield
Mon - Fri	Mon - Fri	Mon - Fri
9:00 AM - 5:00 PM AEST	9:00 AM - 5:00 PM GMT	9:00 AM - 5:00 PM MST

After Hours Support

After hours support is defined by Support Package Response SLAs.

Customers with Extended or Premium customer care support packages will receive 24x7 platform support for severity 1 issues. Customers should log a ticket in the Temenos Support Portal to be promptly contacted by one of our support engineers.

Customers with Core support packages will receive support next business day, if a severity 1 ticket is logged outside of their business hours.

Contacting Support

Support Centers



Business Hours

Our centers and business working hours are:

Australia - Sydney	United Kingdom - London	Colorado USA - Broomfield
Mon - Fri	Mon - Fri	Mon - Fri
9:00 AM - 5:00 PM AEST	9:00 AM - 5:00 PM GMT	9:00 AM - 5:00 PM MST

After Hours Support

After hours support is defined by Support Package Response SLAs.

Customers with Premium support packages will receive 24x7 platform support for severity 1 issues. Customers should log a ticket in the Temenos Support Portal to be promptly contacted by one of our support engineers.

Customers with Core support packages will receive support next business day, if a severity 1 ticket is logged out-side of their business hours.

Temenos Support Portal

You can contact our Support Team by logging a ticket on our Temenos Support Portal.

Accessing the Temenos Support Portal

The Temenos Support Portal is accessible from the following URL.

Username and Password

Named contacts have been defined by your organization and only these contacts will be able to login using their email address as their username to the Temenos Support Portal.

To request new access, you have to ask the manager from your organization to request new user access through the Temenos Support Portal or contact the cloudjirasupport@temenos.com or jira_servicedesk_admin-s@temenos.com email addresses.

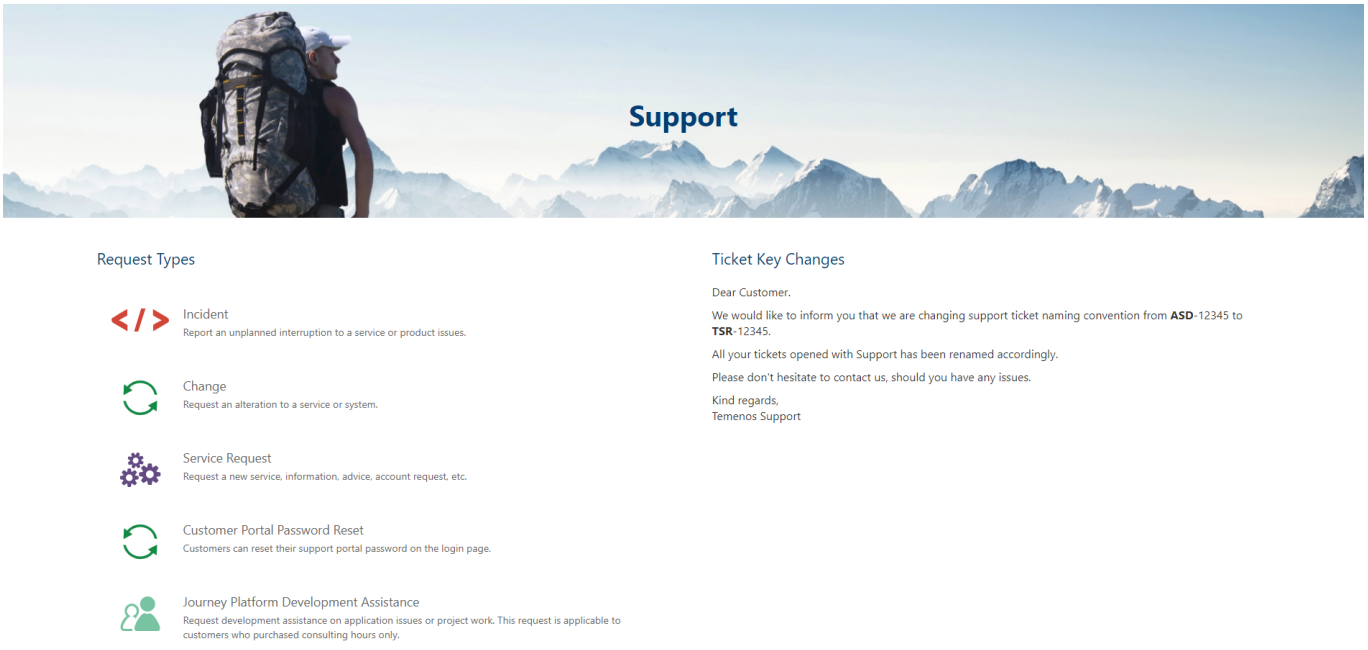
Open a new ticket

The support portal will allow you to:

- Log new tickets for different issues or requests
- Search our resources and documentation content when logging new issues
- View existing tickets
- Respond to tickets needing more information

You can open a support ticket for the following categories:

- Incident - Report an unplanned interruption to a service or product issue.
- Change - Request an alteration to a service or system.
- Service Request - Request a new service, information, advice, account request, etc.
- Support Password Reset - Customers can reset their support portal password on the login page.
- Journey Platform Development Assistance - Request development assistance on application issues or project work. This request is applicable to customers who purchased consulting hours only.



Describe your issue

You will be required to enter mandatory information to allow the Support Team to start investigation.

It is important that the ticket is submitted with relevant, accurate, and detailed information otherwise delays might be introduced to gather further information necessary to troubleshoot the issue.

Summary

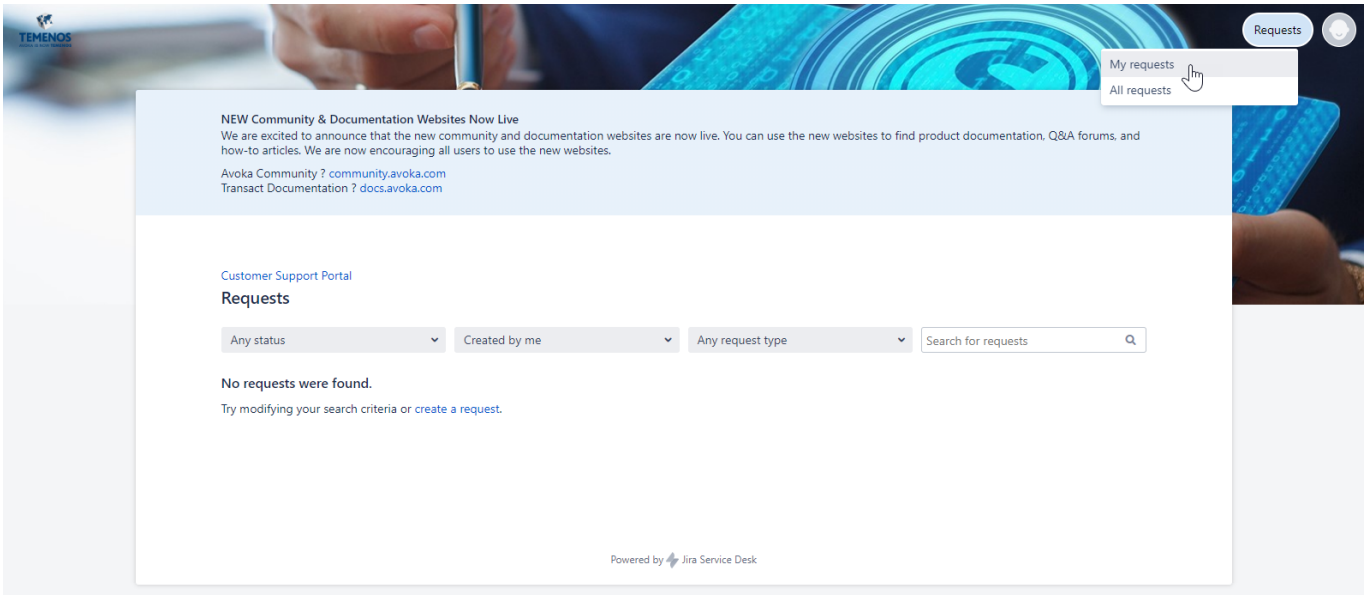
- The summary of the ticket should highlight the core problem.

Issue Description

- **Single issue** - Include a single problem per ticket to avoid misunderstanding and keep focus on the issue. Ensure there is no ambiguity in description of the issue.
- **Environment details** – Provide any environment details including product versions, form versions, 3rd party services involved, any resent changes or updates, etc.
- **Severity and business impact** – Describe business impact and context, including number of services, forms, users, customer impacted, frequency of the issue, project schedule at risk, etc.
- **Reproducible steps** - Clearly define the steps that we can take to reproduce the issue and try to reproduce issue on the latest deployment with simple configuration.
- **Actual behavior** – Describe and provide screenshots for the actual behavior when experiencing an issue.
- **Expected behavior** – Describe and provide mockups screenshots for the expected behavior.
- **Screenshots** – Include any related screenshots which can help us to better understand the issue. Take screenshots of the whole window rather than providing cropped images, without context.
- **Attachments** - If the issue relates to system outputs (e.g. receipt documents, or emails) please attach examples when submitting the ticket. The screenshots should be correctly named to match the description.
- **Your expectations** – Provide details on your expectations including time deadlines, versions where issues are expected to be fixed, etc.

View existing tickets

To review existing tickets, click **Requests** in the upper right corner of the Support Portal and use the selections provided.



You can also search for and view requests using the following filters:

- Status
- Created by
- Request type
- Search text

Ticket statuses explained

Status	Description
New	The ticket is awaiting triage by a Support Engineer.
In Progress	Temenos representative is actively working on the ticket.
Waiting for Customer	Ticket is waiting on information or update from the customer.
Under DEV Investigation	The issue is under investigation by development team.
Defect Acknowledged	Issue reported has been qualified to require a code change and has been assigned to the development team.
Awaiting Version Release	Issue has been fixed by code change in a specific version and the version is pending release.
On Hold	Ticket has been suspended by agreement with the customer.
Closed	Ticket is now closed and no longer actively worked on.

Lifecycle of a support ticket

Step 1

Open Support Ticket

To open a support ticket navigate to the Support Portal.

When a ticket is created, our system automatically verifies your Support Package and assigns the Response SLA based on the severity of the ticket.

Support Ticket Severity

Customers should select the severity level according to the severity description when the ticket is opened. The Support Engineer will review the severity when the ticket is opened and may reduce the severity if the description of the issue does not match the description of the severity.

Severity levels will not increase once established but may be reduced once the impact is mitigated.

Step 2

Engagement with Support Team

Your ticket will be assigned to a Support Engineer who will assist you to identify the issue and begin troubleshooting.

This may involve:

- Requesting more information including reproducible steps, logs and screenshots.
- Reproducing the issue internally.
- Requesting a screensharing session or remote access.
- Requesting an upgrade to new maintenance version which includes hotfixes.
- Implementing a workaround.
- Testing a solution in a non-production environment.
- Request opening a ticket with a third-party vendor.
- Support Engineer may engage a Senior Engineer when necessary.
- Once all action items have been exhausted and the problem is not resolved, the issue may be transferred to an Escalation Engineer who will assist you further.
- If the issue requires a code fix or code-level analysis, the Engineering team will be engaged.

As a customer you can review the progress of any open ticket in the Support Portal.

Step 3

Close a Support Ticket

You can request to close your ticket in the Support Portal.

Before closing a support ticket, you will be asked to confirm that the issue has been worked to your satisfaction and that the support ticket can be closed.

After the closure, a summary will be sent to you with a description of the problem and the reason for closure or resolution.

Issue Resolution

The Support team will consider a ticket resolved and closed with the following possible outcomes:

- Provide documentation/information of product expected behavior.
- Provide an acceptable workaround.
- Provide a software update.
- Provide a software upgrade.
- Update to product documentation.
- File an enhancement request.
- Customer decides to close the issue with no resolution.

Other reasons for closing a support ticket may include:

- Customer is not entitled to support.
- No response has been received after several attempts to contact you.

Step 4

Customer Satisfaction Survey

After case closure, you may receive a survey to ensure your case was worked to your satisfaction.

Step 5

Reopening a support ticket

You will have 30 days from the support ticket closure to reopen the ticket if necessary. If the 30 days have lapsed, a new support ticket must be opened.

Escalating a support ticket

Escalating support cases

Our goal is to provide a way for our customers to escalate an issue to the Support Manager if:

- There is an increased impact on your business and you need to raise the priority.
- You require faster response or more attention.
- You are dissatisfied with the progress of your issue, or the Support experience.

Our Support team will work with you through the life cycle of your ticket to ensure that the appropriate resources and level of attention are focused towards a timely resolution of your issue(s). If you are not satisfied with the experience you can raise the ticket to the attention of the Support Management Team.

How do I escalate the visibility of a support ticket to management?

To escalate an issue to Support Management Team, take the following steps:

1. Log into the Support Portal.
2. Navigate to your existing ticket.
3. Click the **Escalate** button.
4. Enter the reason for the escalation.
5. Click the **Escalate** button which will complete the Escalation process.

What happens after an escalation request is made?

Once your request for escalation has been received, we will acknowledge the receipt of your request and the Support Manager will contact you within 24 business hours.

The Support Engineer assigned to the issue will be notified to review to the issue and contact you within the working hours of your ticket.

Time-Based Escalation

We have an internal time-based escalation process in place to guarantee the right level of management visibility on progress of open support issues.

Our escalation hierarchy is:

- Customer Success Manager and Account Executive
- Regional Support Manager
- VP, Global Support

Time-based escalations are triggered on the following frequency:

Severity level	Core	Extended	Premium
Severity 1	1 week	3 business days	1 business day
Severity 2	1 month	2 weeks	1 week

Product release definitions

Release terminology

Major release

A major release is denoted by the first two digits of the release number. For example, **5.10**.0, **18.05**.0.

It contains significant or breaking changes. For example, changes or removal of existing APIs, significant changes to the user experience, or removal or addition of a major feature.

Maintenance release

A maintenance release is denoted by last digit of the release number. For example, 5.10.**12**, 18.05.**4**.

It contains product defect fixes, stability and performance improvements. Depending on the nature of the product defect fixes, they may introduce minor changes to existing features, but do not include new features or high risk changes, so can be adopted quickly. We recommend regularly upgrading to the latest maintenance release for your current version.

End-of-Life Policy

Our support will only provide assistance on supported product and platform versions. For more information, see our End-of-Life policy.

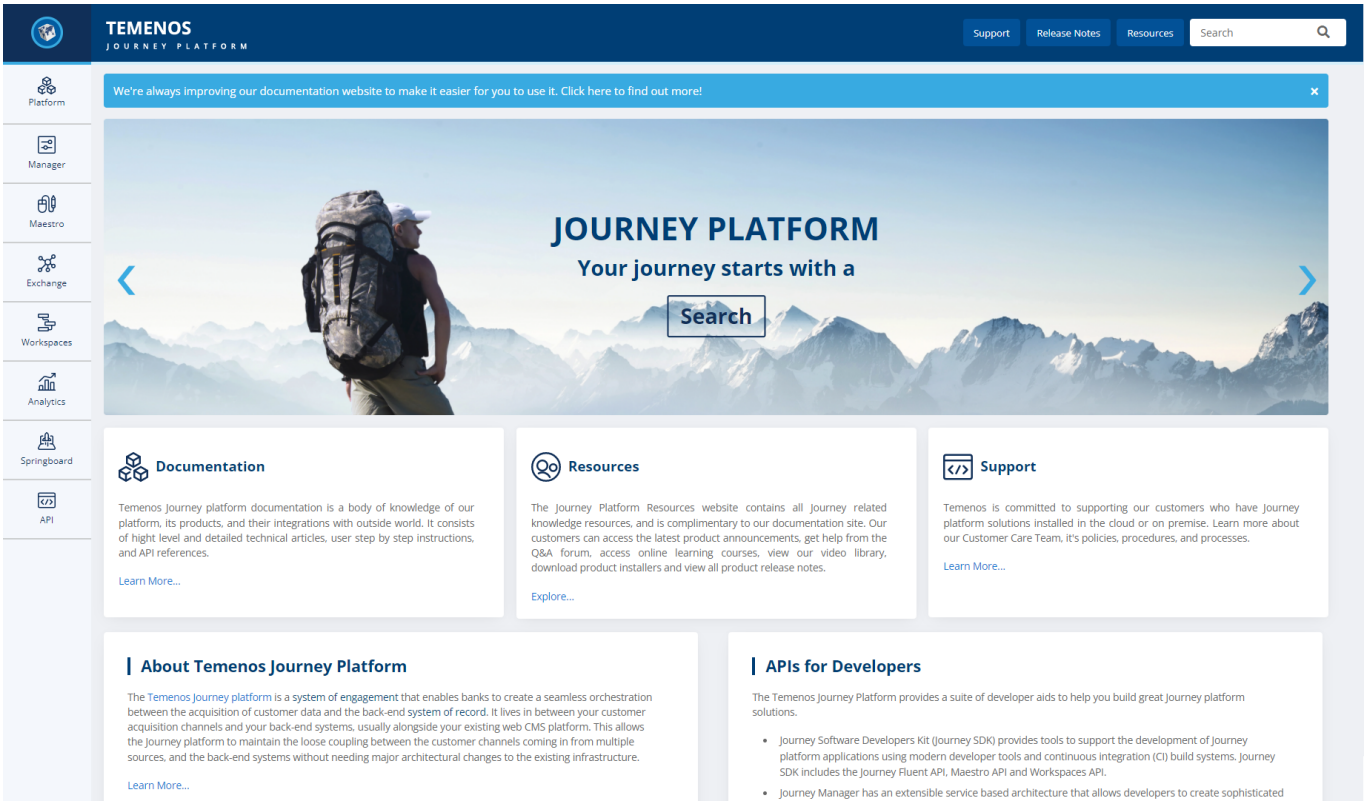
End -of-Life Support may be considered for an additional fee. Contact your Account Executive for more details.

While support will be provided for all versions until End-of-Life Date, it is likely that in the case of issues related to older versions, the Support Engineer will require an upgrade to resolve the issue.

Getting more information

Documentation website

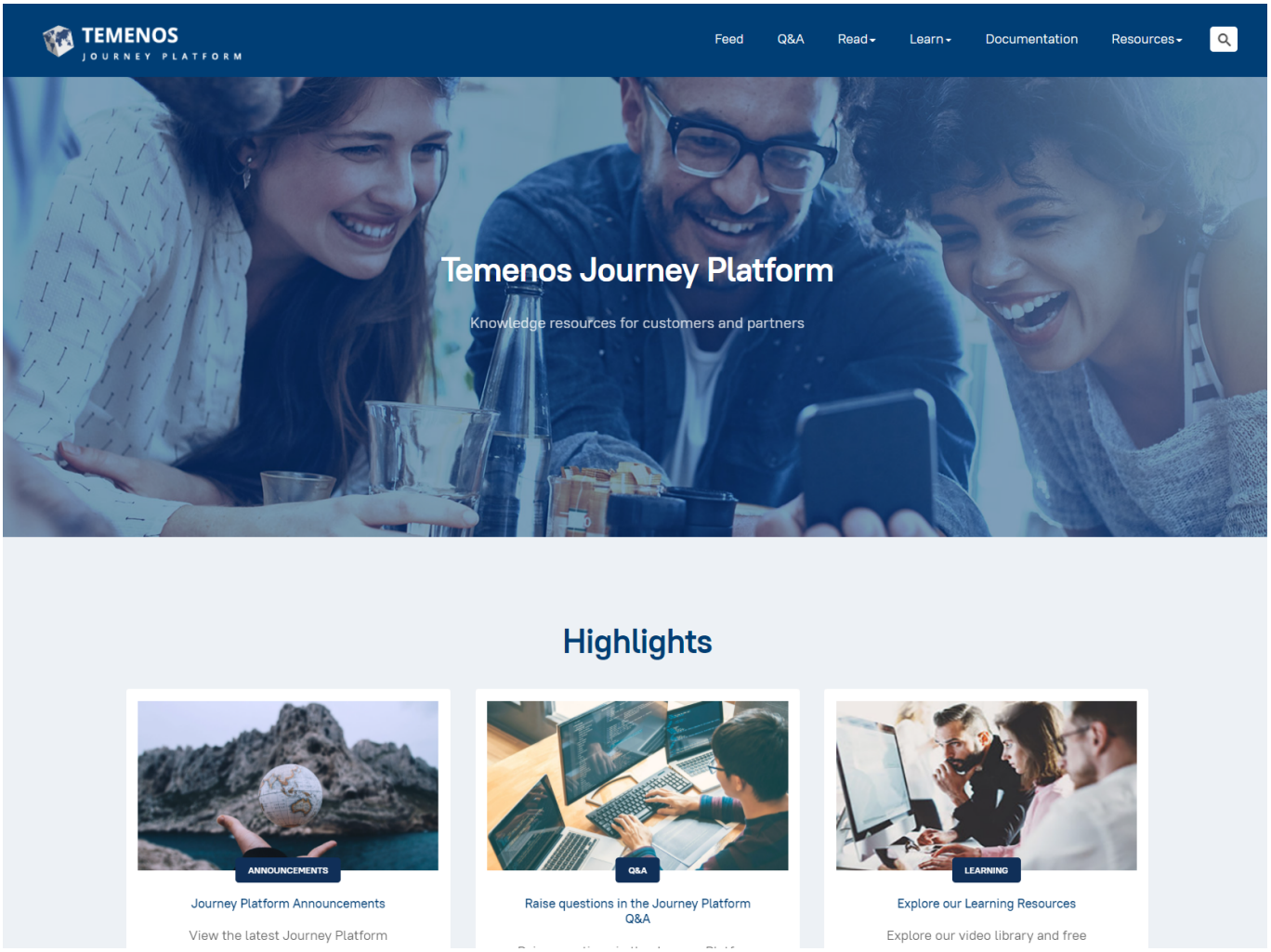
Our product documentation can be found at docs.avoka.com.



You can easily find a wealth of information on all our Journey Platform products; Manager, Maestro, Exchange, Workspaces and Journey Analytics (Insights). As well as product documentation, you will also find developer tech docs in the API section, including architectural documents and SDK documentation.

Resources website

Our Resources website is located at journey.temenos.com.



Here you will find announcements, release notes and product downloads, as well as our Q&A forum.

Announcements

All product announcements and advisories can be found in the Announcements section.

Release notes

Product release notes for all Journey Platform products, including detailed features, enhancements, and resolved issues for each release.

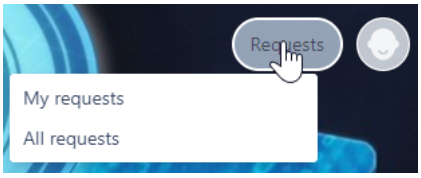
Q&A

This is the space for our customers to ask questions to the wider user community. The Q&A is offered to foster user community discussion to gather feedback and crowd source ideas.

While it is expected to often be technical, this section is separate from the Support system, and is intended as a place to facilitate useful discussions and browse topics. Our staff will contribute where helpful.

Ideas and New Feature Requests

To log New Feature Requests, navigate to Temenos Support Portal > Ideas. You can submit your ideas and vote for existing ideas. To view existing ideas, click on **Requests** > **All Ideas**.



Journey Platform End-of-Life Policy

Journey Platform | All Personas

Each release of a product will reach its **End-of-Life**[1] (EOL) date 3 years after its **General Availability**[2] (GA) date.

This policy relates to our core product modules, including:

- Journey Manager
- Journey Maestro
- Transact Composer
- TransactField App

Each product has its own release numbering and GA dates.

All product releases are supported and maintained until their EOL date. After that, out-of-support releases will be supported only to facilitate an upgrade to a supported release version.

Support	Period	Support & Maintenance
Normal support	Release to EOL date (years 1 to 3)	Full access to the support portal per your support package. Monthly Maintenance Release available against the most recent major version only. For older Manager versions - Critical issues fixed on request and with approval from the Product Owners and company management. Access to new releases.
	After Support Period (years 4+)	No support. Access to new releases.

Feature and Maintenance Releases

New features are added to the product in major releases. Maintenance releases provide fixes and security improvements.

---

[1] End-of-life (EOL) is a term used with respect to a product supplied to customers, indicating that the product is in the end of its useful life (from the vendor's point of view), and a vendor stops marketing, selling, or rework sustaining it.

[2] General availability (GA) is the marketing phase when all commercialization activities pertaining to the software product have been completed and it is available for purchase. It is also known as production release.

New features are added to the current release only and are not ported to previous releases. For example, features added in version 18.11.0 will not be available in version 18.05.x.

Monthly Maintenance Releases are available against the most recent major version only, unless, in our view, there is a significant reason to patch a previous release, such as a major security issue. For example, a fix in v18.11.5 will not be released for v18.05.

Latest Release Notes

Check out the latest release notes for all products.

Feature Lists

Check out the feature lists for Manager and Maestro, divided by each major release.

End-of-Life Schedules

The following are the End-of-Life schedules for our products.

Journey Manager

Supported versions

Version	GA Release Date	EOL Date
21.11	Dec 3, 2021	Dec 3, 2024
21.05	June 15, 2021	June 15, 2024
20.11	Jan 12, 2021	Jan 12, 2024
20.05	May 1, 2020	May 1, 2023
19.11	Nov 30, 2019	Nov 30, 2022
19.05	Jun 14, 2019	Jun 14, 2022

Unsupported (EOL) versions

Version	GA Release Date	EOL Date
18.11	Nov 26, 2018	Nov 26, 2021
18.05	Jun 1, 2018	Jun 1, 2021
17.10	Dec 8, 2017	Dec 8, 2020
5.1	Apr 12, 2017	Apr 12, 2020

5.0	Oct 17, 2016	Oct 17, 2019
4.3	Nov 25, 2015	Nov 25, 2018
4.2	Jul 15, 2015	Jul 15, 2018
4.1	Nov 6, 2014	Nov 6, 2017
4.0	Apr 15, 2015	Apr 15, 2017
3.6	Aug 15, 2013	Aug 15, 2016
3.5	Apr 8, 2013	Apr 8, 2016
3.4	Sep 28, 2012	Sep 28, 2015

Journey Maestro

Supported versions

Version	GA Release Date	EOL Date
21.11	Dec 3, 2021	Dec 3, 2024
21.05	July 6, 2021	July 6, 2024
20.11	Jan 11, 2021	Jan 11, 2024
20.05	May 31, 2020	May 31, 2023
19.11	Dec 13, 2019	Dec 13, 2022
19.05	July 5, 2019	July 5, 2022

Unsupported (EOL) versions

Version	GA Release Date	EOL Date
18.11	Dec 7, 2018	Dec 7, 2021
18.05	May 31, 2018	May 31, 2021
17.10	Dec 8, 2017	Dec 8, 2020
5.1	May 5, 2017	May 5, 2020
5.0	Oct 17, 2016	Oct 17, 2019

Strategy for Continued Journey Maestro Support after AngularJS End-of-Life from 31 December 2021

Maestro will continue to be supported after the AngularJS EOL.

TheMaestro product uses AngularJS as its underlying web framework. From 31 December 2021, Google, the project owner for AngularJS, will end its support per an EOL announcement on the official website: https://-docs.angularjs.org/misc/version-support-status.

This means that security patches and fixes will no longer be provided by the AngularJS project. It does not mean AngularJS apps will stop working:

All AngularJS applications that work now, will continue to work in the future. For more information, see https://b-log.angular.io/stable-angularjs-and-long-term-support-7e077635ee9c

This notification outlines Temenos' strategy and commitment to fully support the Journey Maestro product after the AngularJS EOL and into the foreseeable future.

Temenos is currently in negotiation with a third-party support provider to provide a commercial support contract, service level agreement, and license for a version of AngularJS that is maintained by the third-party through to at least 2026 (with an option to extend).

The commercial support is identical to what we currently receive and covers security weaknesses and fixes caused by web browser and jQuery changes.

This version of AngularJS will be included in the Q4 release of Maestro - 21.11.

Please be assured that there will be no impact to Journey Maestro customers. The product team will continue the practice of applying, regression testing, and releasing the latest patch versions of AngularJS as they are made available.

Transact Composer

Unsupported (EOL) versions

Version	GA Release Date	EOL Date
4.5	May 3, 2018	May 3, 2021
4.4	Oct 17, 2016	Oct 17, 2019
4.3	Nov 3, 2015	Nov 3, 2018
4.2	Jul 15, 2015	Jul 15, 2018
4.1	Nov 6, 2014	Nov 6, 2017
4.0	Apr 15, 2014	Apr 15, 2017

3.6	Aug 15, 2013	Aug 15, 2016
3.5	Apr 8, 2013	Apr 8, 2016
3.4	Sep 28, 2012	Sep 28, 2015

TransactField App

Unsupported (EOL) versions

Version	GA Release Date	EOL Date	EOL Reached
4.3	Mar 1, 2016	May 3, 2021	
4.2	Jul 9, 2015	Jul 9, 2018	
4.1	Nov 6, 2014	Nov 6, 2017	
4.0	Apr 15, 2014	Apr 15, 2017	
3.6	Aug 15, 2013	Aug 15, 2016	

Next, learn about our product support.

Journey Platform Deprecation Schedule

Journey Platform | All Personas

As the Temenos Journey Platform continuously invests to innovate, improve and stay current with security hardening, there is a need to deprecate some older or obsolete features of the platform.

The reasons for deprecating features include:

- Reliance on 3rd party components which have reached **End-of-Life**[1] and are no longer supported.
- Features which pose a security risk to the platform and our customers.
- Features which are not used or barely used by customers.
- Features which add significant complexity to the platform without delivering commensurate business value.

When deprecated features are removed in a future Journey platform release, they will still be supported in their current Journey platform version until EOL of that version.

Feature	Last Supported Release of Journey platform [1]	Reason for Deprecation	Migration Strategy
Oracle JDK 8	18.05	Oracle EOL Policy (Jan 2019)	Upgrade to Journey platform 18.11 or later to obtain OpenJDK Java 11 security updates
Payment Gateway APIs (2 party)	18.11	PCI Compliance & Gateway Vendor EOL Policies	Use redirect gateway integrations (3 party)
Eclipse BIRT Reports	18.11	Security Hardening Reasons	Use the Licensing reports and new Operational reporting features in Temenos Journey platform 18.11
Adobe LiveCycle	19.05	Adobe EOL Policy	Migrate Adobe LiveCycle Forms to Maestro Forms. Migrate Adobe LiveCycle Output PDF receipts to use Manager dynamic PDF receipts.

[1] End-of-life (EOL) is a term used with respect to a product supplied to customers, indicating that the product is in the end of its useful life (from the vendor's point of view), and a vendor stops marketing, selling, or rework sustaining it.

Groovy Access to internal APIs	19.11	Security Hardening and Platform Support Reasons	Migrate scripts to supported Transact Fluent API
Work Space (v1)	20.11	Platform Support and Performance Reasons	Use Workspaces (v2) replacement module

[1] - Future release numbers and actual release dates are subject to change. For more information, see latest announcements.

Next, learn more about the Journey Platform End-of-Life Policy.

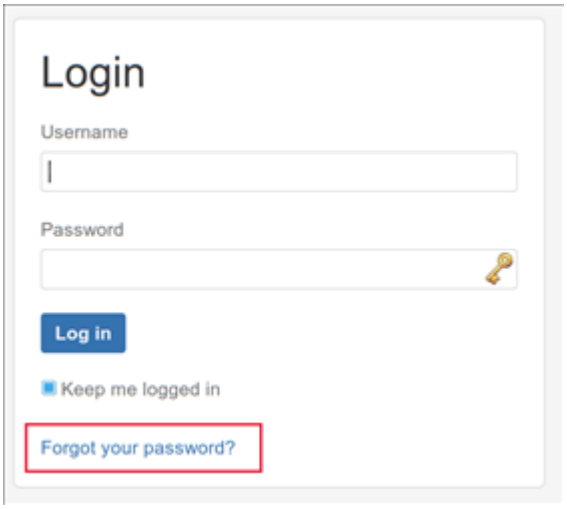
Reset Support Portal Password

Journey Platform | All Personas | 5.1 & Higher

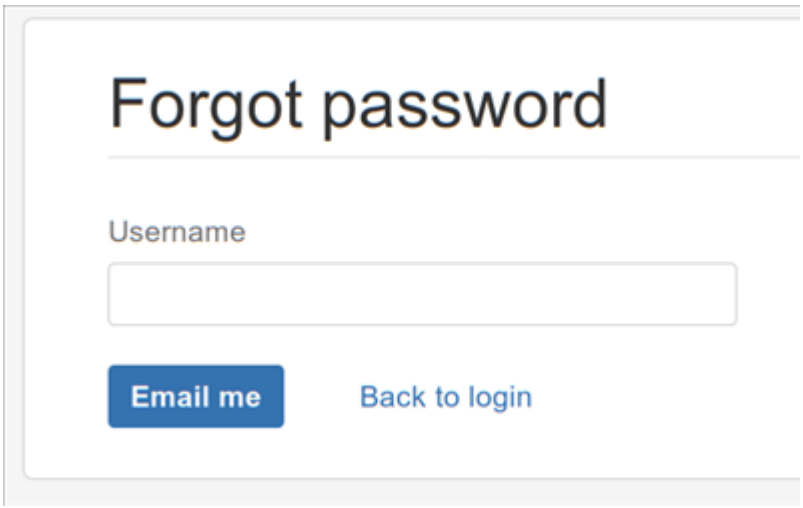
Every now and then you may need to reset your Support Portal password. If you still need help with your access, contact jira_servicedesk_admins@temenos.com or cloudjirasupport@temenos.com.

To reset the password:

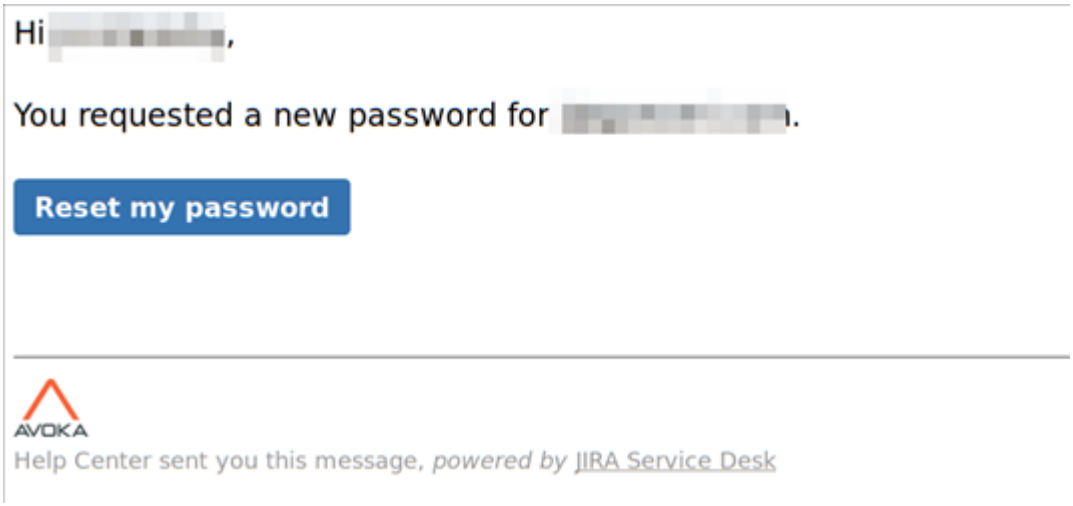
1. Go to Support Portal.
2. Click on **Forgot your passport?**



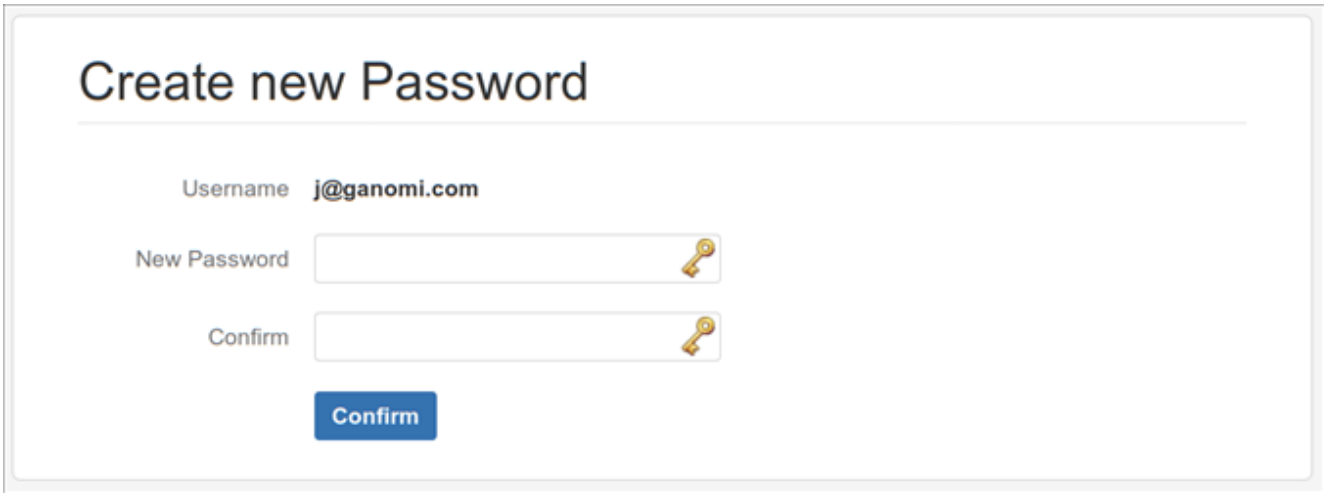
3. Enter your username (email address) and click **Email me**.



4. A link to reset your password will be send to your email address. Click **Reset my password** in the email.



5. Enter your new password and click **Confirm**.



Don't use your last 3 passwords, otherwise your password will not be reset. If you make a mistake and enter any of your last 3 passwords, you have to reset the password again.

6. Your password has been reset and you should be logged into the Support Portal automatically.

Changes to Our Product Names

Journey Platform | All Personas | 5.1 & Higher

Avoka is now a part of Temenos. In December 2018 Temenos announced and completed the acquisition of Avoka. Temenos chose Avoka to further its leadership with a digital banking product focused on customer experience, and has integrated Avoka as the Journey Manager product in the newly announced Temenos Infinity suite of products.

As part of the integration with Temenos, all product names have been changed and reflected in our documentation. Both names now appear in the labels at the top of each topic. Refer to the table below to see the original and new product names.

Original Name (Avoka)	New Name (Temenos)
Transact Platform	Journey Platform
Transact Manager	Journey Manager
Transact Maestro	Journey Maestro
Transact Insights	Journey Analytics
Avoka Exchange	Journey Exchange
Transact SDK	Journey SDK
Transact Open UX	Journey Open UX
Springboard	Journey Springboard

For example, if you are looking for the **Transact Manager Overview** topic, you should now search for the **Journey Manager Overview**.

Name Changes from the original Knowledge Base

We have always believed in providing high quality documentation. Prior to hosting on docs.avoka.com, we hosted the Knowledge Base via our Confluence portal at https://support.avoka.com/kb/. Since then, the platform and its products evolved, and we rewrote our content as **topic-based**[1] articles and improved its accessibility and search capability with the Algolia search engine.

The improvements are quite significant, affecting the structure, look and feel of the content and its discovery, so we recommend you become familiar with the website by checking our getting started page. If you are experiencing difficulties locating old KB documentation, we have mapped the most popular old KB content with the new topics, which are the entry points to all related content. Check the tables below to see how of the old KB articles map to new documentation topics.

Old KB Articles	New Documentation Topics

[1] Topic-based authoring is a modular approach to content creation where content is structured around topics that can be mixed and reused in different contexts.

Transact Manager Administration Guide	Journey Manager Overview
Transact Manager Configuration	Journey Manager Home Dashboard
Organizations	Organizations Overview
User Accounts	User Accounts Overview
Form Spaces	Form Spaces Overview
System Logs	System Logs
Configure Virus Scanning for Attachments	Symantec Attachment Virus Scanning Service ClamAV Attachment Virus Scanning Service
Core Global Services	Core Global Services Overview
Maestro Form Configuration in TM	Form Dashboard
Multi-Environment	Multi-Environment Support
Server Nodes	Server Nodes Overview
Outgoing Email	Outgoing Email Service Overview

For more information on our products prior to version 5.1, see Other products under Products.

If you can't find an article you are looking for, but you remember its name as it was in the KB, try to search for that name using our Algolia search engine.

Next, learn about our new documentation.

Journey Platform System Requirements

Journey Platform | All Personas

Each product of the Temenos Journey platform has its own requirements for running in the cloud or on-premise. These requirements are listed below:

- Journey Manager requirements
- Journey Maestro requirements
- Journey Platform forms requirements
- Journey Workspaces requirements
- Mobile Browsers requirements

Legacy

- Transact Composer requirements
- TransactField App requirements
- Transact Integration Gateway requirements
- Legacy Browsers requirements

Next, learn about the Journey Platform deprecation schedule.

Journey Manager System Requirements

Journey Manager (JM) | All Personas

Journey Manager is a secure, high performance application server that processes, manages, and monitors transactions associated with customer onboarding journeys. To provide the best customer experience while handling a significant number of transactions, the application server requires specific system configurations described below.

Journey Manager can be run in the cloud or on premise, depending on your requirements. We recommend running our software in the cloud environment provisioned by Temenos Journey Cloud Service, so you can have new services commissioned quickly and additional capacity allocated to you dynamically as required.

Cloud

Journey Manager can be run as a hosted service in the following clouds:

- **Amazon**[1] cloud
- **Azure**[2] cloud | 20.05

Journey Manager is provisioned in the following regions:

- APAC: AWS Sydney region / Azure Australia East
- North America: AWS Oregon region / Azure US East
- EMEA: AWS Ireland region / Azure North Europe

We can provision Journey Manager in other regions as well, as long as they support the services that our software requires.

On Premise

You can install Journey Manager in local environments giving you a complete control of the whole infrastructure.

Browser

- Google Chrome - current version.
- Microsoft Edge – current version.
- Microsoft Internet Explorer 11.0.
- Mozilla Firefox - current version.

---

[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.

Operating Systems

Journey Manager is certified on the following OS:

- **Microsoft Windows Server**[1] 2016, 2012 or 2008 R2 64 bit Edition on Intel x86-64
- **CentOS Linux**[2] 7 on Intel x86-64 (recommended)
- CentOS Linux 6 on Intel x86-64 (supported)
- **Oracle Linux**[3] 6 (Unbreakable Enterprise Kernel) on Intel x86-64 (supported)
- **Red Hat Enterprise Linux Server**[4] AP 6 on Intel x86-64 (supported)

Databases

Journey Manager is certified on the following DBs:

- **AWS Aurora RDS**[5] (MySQL-Compatible Edition)
- **Microsoft SQL Server**[6] 2017, 2014, 2012
- **Oracle MySQL**[7] 5.7 (recommended), 5.6 and 5.5 (supported)
- **Oracle RDBMS**[8] 19c | 20.05
- Oracle RDBMS 12c and 11g

---

[1]Microsoft Windows Server OS (operating system) is a series of enterprise-class server operating systems designed to share services with multiple users and provide extensive administrative control of data storage, applications and corporate networks.

[2]CentOS (Community Enterprise Operating System) is a Linux distribution that provides a free, enterprise-class, community-supported computing platform functionally compatible with its upstream source, Red Hat Enterprise Linux (RHEL).

[3]Oracle Linux (OL, formerly known as Oracle Enterprise Linux) is a Linux distribution packaged and freely distributed by Oracle, available partially under the GNU General Public License since late 2006. It is compiled from Red Hat Enterprise Linux (RHEL) source code, replacing Red Hat branding with Oracle's. It is also used by Oracle Cloud and Oracle Engineered Systems such as Oracle Exadata and others.

[4]is a distribution of the Linux operating system developed for the business market.

[5]Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks.

[6]Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network (including the Internet).

[7]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,

Hardware

Journey Manager runs best on latest Intel x86-64 Xeon CPU architectures with high CPU performance. We recommend using server **virtualization**[1], such as VMware ESX, to enable better hardware utilization and improve operational support.

The recommended server specification is:

- Intel Xeon Processors 8 Cores with 2.4 GHz or greater clock speed (4 Cores minimum)
- 16 GB of RAM (8 GB minimum)
- 100 GB of disk storage

To get a complete picture of all system requirements you might need, check other Journey Platform products that you will need to use along with Journey Manager to build different customer journeys.

Next, learn about Journey platform support.

[1]Virtualization is the creation of a virtual resource such as a server, desktop, operating system, file, storage or network to manage workloads by radically transforming traditional computing to make it more scalable.

Journey Maestro System Requirements

Maestro | All Personas

Journey Maestro is web-based design tool for creating HTML applications which are hosted on Journey Manager.

> To deploy a Maestro form, Journey Manager v5.0 or higher is required.

We support the following configurations:

Cloud

Available as an **Amazon**[1] hosted service, currently provided in the AWS regions US West (Oregon), EU (Ireland) and Asia Pacific (Sydney).

Browser

- Google Chrome - current version. This is the recommended browser.
- Mozilla Firefox - current version.

Recommended Hardware

- Operating System: Microsoft supported Windows 7 or Windows 10 versions, and Apple supported macOS versions
- RAM: 8GB
- CPU: Intel Core i5 or above
- Screen size: 1920 x 1080 or larger. This is not a hard requirement, however, the greater the screen size the better it is to design pages in Maestro tool
- Disk space: 10GB

Next, learn about Journey platform support.

[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.

Journey Platform Forms System Requirements

Journey Platform | All Personas

The Temenos Journey platform ensures that forms built with Journey Maestro are supported on both desktop and mobile platforms.

Desktop Browsers

The Journey Platform supports Maestro forms being used on the following desktop browsers:

- Apple Safari - current version
- Google Chrome - current version
- Microsoft Edge - current version
- Microsoft Internet Explorer - 11.0
- Mozilla Firefox - current version

Legacy Desktop Browsers

The Journey Platform no longer supports the following browsers because Microsoft no longer provides security patches or fixes for these browsers, which poses a significant security risk.

- Internet Explorer 8 (IE8)
- Internet Explorer 9 (IE9)
- Internet Explorer 10 (IE10),

For further details, refer to Microsoft's website:

- https://support.microsoft.com/en-us/lifecycle#gp/Microsoft-Internet-Explorer
- https://www.microsoft.com/en-au/WindowsForBusiness/End-of-IE-support

In addition, these legacy browsers do not support modern security standards required for security transactions including:

- TLS 1.2 which is required for secure credit card transactions (PCI policy)
- **Content Security Policy**[1] (CSP) to prevent JavaScript injection (XSS) attacks. For more information, see https://www.w3.org/TR/CSP/

[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/

For Journey Manager 4.3.3 and later, customers must explicitly configure their environments to support IE10, IE9, or IE8. The default Journey Manager configuration does not support these browsers. Customers enabling support for legacy browsers do so at their own risk.

Mobile Browsers

The Journey Platform supports Maestro forms in the following mobile browsers:

- Apple Safari on iOS - 7.0 or later
- Google Chrome - 4.0 or later

The Google Search App (GSA) is not supported. You should use Google Chrome instead.

The Journey Platform forms have been tested across a wide range of browsers on mobile devices including those from Apple, Samsung, HTC, and Nokia.

Microsoft Windows Phone 8 devices are not supported as these are EOL by Microsoft, and we do not support Windows 10 Mobile phone devices as these have not been successful in market and are scheduled EOL in 2019 by Microsoft.

Exhaustive testing across market leading devices and browsers is performed. However, the proliferation of mobile platform combinations (mobile device + OS + patches + browser) preclude us from testing Maestro form functionality against all mobile platform combinations.

If you're using TransactField App in a Bring Your Own Device (BYOD) deployment model, we recommend testing Journey Platform forms against your target devices.

Next, learn about Journey platform support.

Mobile Browsers Requirements

Journey Platform | All Personas

Temenos diligently tests HTML forms across a wide range of browsers on devices from the following companies:

- Apple
- Samsung
- HTC
- Nokia

Temenos doesn't support the following mobile devices:

- Microsoft Windows 8 phone as it's now **EOL**[1].
- Microsoft Windows 10 phone as it has not been successful on the market and its scheduled EOL is 2019.

While we perform exhaustive testing across market leading devices and browsers, there is a large number of mobile devices, browsers, and device patch combinations which means we can't test against all these combinations.

If you are using TransactField App in a Bring Your Own Device (BYOD) deployment model, we recommend that you test HTML forms against your target devices.

Next, learn about Temenos support.

[1]End-of-life (EOL) is a term used with respect to a product supplied to customers, indicating that the product is in the end of its useful life (from the vendor's point of view), and a vendor stops marketing, selling, or rework sustaining it.

Transact Composer Requirements

Composer | All Personas

Transact Composer is design tool for creating HTML applications which are hosted on Journey Manager . The Composer design tool is an Adobe Flash based application which can be used web browsers supporting Flash or as a desktop Adobe AIR application.

We support the following configurations:

Cloud

Available as an **Amazon**[1] hosted service, currently provided in the AWS regions US West (Oregon), EU (Ireland) and Asia Pacific (Sydney).

Adobe Flash

Requires browsers supporting **Adobe Flash**[2] Player, or the Adobe AIR runtime environment for desktops.

Browser

- Google Chrome - current version
- Microsoft Edge – current version
- Microsoft Internet Explorer 11.0
- Mozilla Firefox - current version

Composer is built with Flash, which is scheduled for deprecation in 2020. Once this happens, Composer will need to be run via Adobe Air.

[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] 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.

Recommended Hardware

- Operating System: Microsoft supported Windows 7 or Windows 10 versions, and Apple supported macOS versions
- RAM: 8GB
- CPU: Intel Core i5 or above
- Screen size: 1920 x 1080 or larger (recommended)
- Disk space: 10 GB freely available

For more information see Journey platform support overview.

Transact Integration Gateway System Requirements

Journey Platform | All Personas

The Transact Integration Gateway (TIG) is a light weight software appliance for deploying on premise to enable hybrid cloud Transact integration.

We support the following configurations:

Certified Operating Systems

- Microsoft Windows Server 2012 or 2008 R2 64 bit Edition on Intel x86-64
- CentOS Linux 7 on Intel x86-64 (recommended)
- CentOS Linux 6 on Intel x86-64 (supported)
- Red Hat Enterprise Linux Server AP 6 on Intel x86-64

Recommended Hardware

Transact Integration Gateway runs best on latest Intel x86-64 CPU architectures. We recommend server virtualization such as VMware ESX to provide enable better hardware utilization and improve operational support.

The Transact Integration Gateway has the following server requirements:

- Intel x86-64 Processors (2-4 Cores)
- 4-8 GB of RAM
- 100 GB of disk storage

For more information, see the Transact Integration Gateway Installation Guide documentation.

Next, learn about Temenos support.

TransactField App System Requirements

Journey Platform | All Personas

TransactField App supports the following devices for existing customers:

- Apple iOS 7 or later devices
- Google Android version 4 or later devices
- Microsoft Windows 8.1 devices via the Windows Store
- Microsoft Windows 7 or later PCs with Windows App

Next, learn about Journey platform support.

Legacy Browsers Requirements

Journey Platform | All Personas

The Journey platform doesn't support the following legacy desktop browsers:

- Internet Explorer 8
- Internet Explorer 9
- Internet Explorer 10

Microsoft no longer supports these browsers and doesn't provide security patches or fixes for zero day exploits making them a significant security risk and liability to customers, form users, and Temenos. For more information, refer to the following documentation:

- https://support.microsoft.com/en-us/lifecycle#gp/Microsoft-Internet-Explorer
- https://www.microsoft.com/en-au/WindowsForBusiness/End-of-IE-support

In addition, these legacy browsers don't support modern security standards required for secure transactions to take place:

- **TLS**[1] 1.2 which is required for secure credit card transactions such as **PCI policy**[2].
- **Content Security Policy**[3] (CSP) to prevent JavaScript injection (XSS) attacks.

In Journey Manager 4.3.3 and higher, customers must explicitly configure their environments to support IE10, IE9 or IE8 because its default configuration doesn't support them.

Customers enabling legacy browsers do so at their own risk.

Next, learn what Temenos supports.

---

[1]Transport Layer Security (TLS) is a protocol that provides privacy and data integrity between two communicating applications. It's the most widely deployed security protocol used today, and is used for Web browsers and other applications that require data to be securely exchanged over a network, such as file transfers, VPN connections, instant messaging and voice over IP.

[2]PCI policy is a type of security policy that covers how an organization addresses the 12 requirements of the Payment Card Industry Data Security Standard (PCI DSS). A PCI policy is required of all merchants and service providers who store, process or transmit credit card holder data.

[3]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/

Journey Platform Persona Overview

Journey Platform | All Personas

To identify, deliver, maintain and support an outstanding Journey platform customer experience requires the efforts of individuals with a broad range of strengths and skills. They can be associated with specific Journey platform actions and responsibilities to define specific personas. Personas have roles associated with them, which provide a guideline and a reference for users performing specific tasks and actions. For example, a Form Builder is a persona that creates forms in Journey Maestro.

We have two special personas, Executive and Starter, that define the audience that need to understand and start using our Journey platform quickly. Both personas have well defined user paths, the Executive path and Starter path. We recommend reading at least one of them before you proceed with other user paths.

Personas can be grouped by teams. We define two main teams within the Journey platform:

- Implementation team
- Business as Usual (BAU) team

Implementation Team

The Implementation team develops new applications, such as forms, services, and delivery channels. The team typically consists of Implementation Managers, Form Builders, Template Designers, Platform Developers, Solution Architects, and Testers. Often there are multiple teams working concurrently to introduce new transactions to the environment. These teams are structured as a project team and depend on your organization's chosen delivery style. The Implementation team will have access to develop forms within the authoring tools and deploy them to Manager development and test environments. They will have access to implement new services that correspond and compliment the forms in development, or that are required to support forms already in production.

The Implementation team consists of the following personas, including their user paths:

- Form Builder | Form Builder user path
- Platform Developer | Platform Developer user path
- Solution Architect
- Maestro Template Designer
- Application Tester
- Implementation / Project Manager
- Analytics User

Business as Usual (BAU) Team

The BAU team maintains and monitors the platform, performs user management, and so on. It is imperative that customers assign the **BAU**[1], or Operations, team to ensure the daily on-going operation of the forms.

> We can provide documentation on recommended operational support practices to the BAU team.

The following activities should be included in BAU monitoring:

- Developing processes to manage deployments.
- Creating and enforcing user management processes.
- Building error alerts.
- Regular Manager transaction monitoring.
- Supporting communication lines.
- Reviewing Manager analytics screens.
- Monitoring server health for on premise installations.
- Deploying updates and applications to the production environment.

The BAU team will:

- Have access to Manager for monitoring, supporting and checking alerts.
- Contain members who can review development works, before these developments are moved from the development phase, thus providing a level of security for the production systems from poorly written code.
- Test all new developments once they are deemed to be production grade developments.
- This team will also regression test any production changes to ensure ongoing integrity of the system.
- Be responsible for user accounts and user management on the Testing, Staging and Production environments and also for provisioning new development environments when requested.
- Maintain the **UX**[2] component of all developments, ensuring compliance to company standards.
- Have attended our training courses and completed the Operational Support Training module.

The BAU team consists of the following personas:

---

[1]usiness As Usual (BAU) is the view of an organisation, business unit, work team or team member completing their everyday work in the normal way.

[2]User experience (UX) design is the process of creating products that provide meaningful and relevant experiences to users. This involves the design of the entire process of acquiring and integrating the product, including aspects of branding, design, usability, and function.

- Champion / System Owner
- System Administrator
- System Manager / DevOps
- Support Team

Maestro Form Builder

Journey Platform | Form Builder

A Form Builder is responsible for designing and testing the data capture and display portion of the solution built with Journey Maestro.

This may include creating forms using existing components and functionality. They may also create simple business scripts and shared components. At a more advanced level, Form Builders may also write simple to complex business and display logic using JavaScript, jQuery, and AngularJS tags. In addition, they will use the Transact Fluent SDK and an IDE to create native components. They will also work closely with the Journey Platform Developer for technical integration points and the Template Designer for styling.

To see a list of Journey platform features specific to the Form Builder persona, see the Form Builder user path.

Example tasks of a Form Builder include:

- Convert requirement documents into Maestro form and component designs.
- Develop and maintain shared components, including global shared binding and data schema.
- Build forms and import them into Manager in non-production environments.
- Implement visibility and validation logic.
- Use relevant **APIs**[1] for forms and services.
- Prepare deployment assets.
- Test functionality to ensure the application meets the requirement specification.

[1] Application Programming Interface.

Prerequisite Skillset of a Form Builder

- 1-2 years background in web page design (3-4 years preferred).
- An understanding of HTML5 tags and HTML5 structure.
- Ability to add functionality with JavaScript and JQuery.
- Ability to modify the view with AngularJS expressions.
- Ability to manage and describe data structure, such as JSON and XML.
- Work with **Source Code Control Systems**[1] and understand version control.
- Debug using browser development tools.
- Work with an IDE and Transact Fluent SDK to build native components and front-end using another development environment, such as **ReactJS**[2].
- Effective communication skills: ability to describe and understand integration points.

Objectives and Goals of a Form Builder

- Create Journey applications that follow the requirement specification documentation.
- Follow best practices when creating new application solutions.
- Design efficient, reusable components to share with other Journey applications.
- Become Temenos Journey certified.

Next, learn about the Implementation/Project Manager />persona.

---

[1] Source Code Control System (SCCS) is a version control system designed to track changes in source code and other text files during the development of a piece of software. This allows the user to retrieve any of the previous versions of the original source code and the changes which are stored.

[2] ReactJS is a JavaScript library for building user interfaces.

Journey Solution Architect

Journey Platform | Solution Architect

A Journey Solution Architect is responsible for the overall design of the application built on the Temenos Journey Platform. They work with the client to gather business requirements and translate these into the architecture for the solution: describing it through architecture and design artifacts. They also work as a single point of contact for Form Builders and Platform Developers on integration points and defining expected functionality. They are the technical lead for planning, analysis, development, testing and transitioning to from development to live customer solutions.

It is recommended that the Solution Architect accomplishes the above tasks following **Agile**[1] methodology; therefore, an understanding of Agile development methodology is usually required.

Example tasks of a Journey Solution Architect include:

- Complete code review for new developments.
- Communicate best practices and new Journey platform features to relevant stakeholders at all levels of the business.
- Formulate solutions to complex business problems.
- Document solution designs for later reference.
- Manage all aspects – planning, development, implementation, archiving, and hand-off – of the development cycle.

[1] Agile software development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer/end user.

Prerequisite Skillset of a Solution Architect

- Effective communication and interpersonal skills.
- 5+ years commercial software delivery experience in a vendor (consultancy or development).
- 2+ years as a web application developer with exposure to the different layers of the stack aside from their area of expertise.
- Experience with the following (or equivalent substitute):
  - Java/**Groovy**[1].
  - Principals of Object-Oriented Programming.
  - JavaScript.
  - HTML, CSS.
- Experience with the following (or equivalent substitute):
  - Manipulate data formats, such as **XSD**[2], **XML**[3], **JSON**[4], using tools and code.
  - WebService/Restful Endpoints.
  - JavaScript Frameworks.
  - Databases, for example, SQLServer, Oracle or MySQL.
  - Application deployment to Web Servers, App Servers, Databases, and various Operating Systems.

Objectives and Goals of a Solution Architect

- Define and follow best practices for building a Journey platform application.
- Lead a project from conception to completion.
- Discover integration points between application.
- Be known as an expert on the Journey platform.
- Liaison/buffer between the Temenos team and the client.
- Guide teams and follow governance practices for developing on the Journey platform.

Next, learn about the Support persona.

---

[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]XSD (XML Schema Definition) is a World Wide Web Consortium (W3C) recommendation that specifies how to formally describe the elements in an Extensible Markup Language (XML) document.

[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]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.

Journey Platform Developer

Journey Platform | Platform Developer

A Journey Platform Developer is responsible for designing and testing the back-end portion of an application built on the Temenos Journey Platform. This includes setting form and organizational properties, creating services that use the Transact Fluent API, as well as third-party APIs to manage, secure, and manipulate data and digital assets. They are also responsible for creating workflows based on business logic, designing and implementing unit and integration tests, documenting functionality and bundling applications for the various Journey Manager instances (Dev, UAT, etc.). In addition, they may be responsible for implementing CSS, CSS-Less, ReactJS or AngularJS design and functionality for form portals.

It is recommended that the developer accomplishes the above tasks following **Agile**[1] methodology; therefore, an understanding of Agile development methodology is usually required.

To see a list of Journey platform features specific to the Platform Developer persona, see the Platform Developer user path.

Example tasks of a Platform Developer include:

- Develop and maintain pages to be used as Portal/Dashboard UI screens.
- Develop and maintain Manager service definitions and Transact functions (**Groovy**[2] scripts).
- Implement business logic using collaboration jobs.
- Version and deploy artifacts using the portal and dashboard in DEV and TEST environments.
- Develop, package and deploy applications using Journey SDK.

[1]Agile software development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer/end user.
[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.

Prerequisite Skillset of a Platform Developer

- Experience with Object Oriented Analysis, Development and Design using Java and Groovy.
- Experience with **IDEs**[1], such as Eclipse or IntelliJ and Journey SDK to build projects using Ant and Maven.
- Understand the principal of continuous integration development.
- Ability to unit test and perform integration testing using Java objects and framework.
- Ability to create services for the Java platform.
- Manipulate data formats such as **XML**[2] and **JSON**[3].
- Integrate IDEs with **source code control tools**[4].
- Convert business logic to electronic workflows.
- Develop following the agile delivery methodology.
- Debug and optimize code.
- Create custom portals using a variety of programming languages and frameworks (Form Spaces).
- Create secure communication channels using credentials, certificates, and keystores.
- Intermediate to advanced knowledge of Groovy, Java, and JavaScript.
- Familiar with SOAP and REST APIs integration.

Objectives and Goals of a Platform Developer

- Design back-end solution of a Journey application.
- Create and maintain Manager service definitions (Groovy scripts).
- Solve technical problems.
- Optimize code.
- Collaborate with other team members.
- Work with the latest technology.
- Become Temenos Journey certified.

Next, learn about the Solution Architect persona.

---

[1]Integrated Development Environment.

[2]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.

[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]Source Code Control System (SCCS) is a version control system designed to track changes in source code and other text files during the development of a piece of software. This allows the user to retrieve any of the previous versions of the original source code and the changes which are stored.

Maestro Template Designer

Journey Platform | Template Designer

A Journey Template Designer is responsible for the creation and functionality of the templates used as the foundation for forms created with Journey Maestro. The template defines the look-and-feel of the form as well as the components based on requirements provided by the client. In addition, the template designer can dictate what elements can be accessed and modified by the form designer.

The Template Designer will create the initial template using the appropriate libraries and **IDE**[1] based on discussions with the client's **UX**[2] design team. The Template Designer will also make small modifications to the existing template using the Maestrotemplate designer interface.

Example tasks of a Template Designer include:

- Creating the custom template, including the placeholder template.
- Managing style libraries within the Maestro project and organization.
- Creating component styles for use in Maestro applications.
- Creating brands that can be applied in the Maestro form.

[1]Integrated Development Environment.

[2]User experience (UX) design is the process of creating products that provide meaningful and relevant experiences to users. This involves the design of the entire process of acquiring and integrating the product, including aspects of branding, design, usability, and function.

Prerequisite Skillset of a Template Designer

- Ability to develop and implement **CSS3**[1] and **CSS-Less**[2] libraries.
- 3 years+ background in web page design.
- 3+ years' experience as a Front-End Developer.
- Understanding of **HTML5**[3].
- Experience working with JavaScript frameworks.
- Experience with **source code control management**[4].
- 1-3 years Visual design, for example, wireframes, mock-ups, graphic design.
- Experience working with Corporate Style guides.
- Knowledge of best practices for accessibility.

Objectives and Goals of a Template Designer

- Review and analysis design specifications for use in the Maestro template.
- Ensure best practices are followed when new styles and templates are created.
- Ensure accessibility requirements are met.

Next, learn about the Application Tester persona.

---

[1]CSS3 is the latest evolution of the Cascading Style Sheets language and aims at extending CSS2.1. It brings a lot of long-awaited novelties, like rounded corners, shadows, gradients, transitions or animations, as well as new layouts like multi-columns, flexible box or grid layouts.

[2]Less is a dynamic preprocessor style sheet language that can be compiled into Cascading Style Sheets and run on the client side or server side.

[3]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.

[4]Source Code Control System (SCCS) is a version control system designed to track changes in source code and other text files during the development of a piece of software. This allows the user to retrieve any of the previous versions of the original source code and the changes which are stored.

Journey Application Tester

Journey Platform | Application Tester

A Journey Application Tester is responsible for designing testing scenarios for usability, application functionality, integration, and bug discovery. They conduct the testing, analyse the results and provide feedback to the development team. The Application Tester is also responsible for creating the testing scripts – usability storyboards, test cases, and automation scripts. For Temenos clients, the persona may be limited to user-acceptance testing, which is less technical but requires a deep understanding of the business rules associated with the application as well as an understanding of the various interfaces provided.

Example tasks of a JourneyApplication Tester include:

- Create test cases based upon documented requirements.
- Write test reports.
- Compare the form to the specification documentation.
- Complete breakage testing - try to break the form, get creative with what you check, etc.
- Check the visual aspects of the form - spacing, spelling, grammar, layout, etc.

Prerequisite Skillset of a Journey Application Tester

- Ability to review and understand requirement specifications.
- Ability to use the Manager dashboard interface.
- Detail oriented.
- Familiarity with **XML**[1] and **JSON**[2] data structure.
- Experience with Java-based software testing tools, such as Selenium Suite, SahiPro, etc.
- Familiarity with version control systems, such as Git, **SVN**[3], etc.
- Experience with Issue & Project Tracking Software, such as Jira.

---

[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] 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] Subversion (SVN) is a software versioning and revision control system.

Objectives and Goals of a Journey Application Tester

- Assess the quality of the application and provide feedback to the stakeholders.
- Provide a level of confidence to management and the developers that the application will meet expectations.
- Ensure the quality of development prior to being released to **User Acceptance Testing**[1] by running test cases.
- Re-test cases during System Testing and UAT processes.

Next, learn about the Analytics persona.

[1]User acceptance testing (UAT) is the last phase of the software testing process. During UAT, actual software users test the software to make sure it can handle required tasks in real-world scenarios, according to specifications.

Journey Analytics User

Journey Platform | Analytics User

A Journey Analytics persona is responsible for ensuring that the Temenos Journey Platform application is performing to an agreed upon set of **Key Performance Indicators**[1] and to identify current trends and design features that can be used to optimize the application using Journey Analytics and Journey Manager dashboard.

Prerequisite Skillset of Journey Analytics

- Understand business needs and objectives.
- Define application-level metrics.
- Extract reports from Journey Analytics and Manager.
- Build systems to transform raw data into actionable business insights.
- Ability to interpret data and improve application performance.
- Keep abreast of industry trends.
- Design and interpret results of A/B testing.

Objectives and Goals of Journey Analytics

- Identify areas of application optimization based on collected data.
- Confirm improvements in application based on metrics.

Next, learn about the Champion/System Owner persona.

[1]Key Performance Indicator (KPI) is a measurable value that demonstrates how effectively a company is achieving key business objectives. Organizations use KPIs to evaluate their success at reaching targets.

Journey Implementation/Project Manager

Journey Platform | Implementation / Project Manager

A Journey Implementation/Project Manager is responsible for the successful implementation of an Temenos Journey Platform custom application. They manage the budget, resources, relationships and set up clients for long-term success. In addition to working with the client, they work closely with solution architects and coordinate the IT and infrastructure teams, brand designers, integration specialists, and developers to ensure that the defined business requirements match the functionality of the application.

Example tasks of a Journey Implementation/Project Manager include:

- Drive the development work ensuring all resources are available, communication lines are clear, and time-frames are communicated and agreed upon.
- Communicate with all stakeholders at various levels of the organization.
- Manage risks and issues of the project, including issues associated with loading a new transaction into the system.

Prerequisite Skillset of a Journey Implementation/Project Manager

- Experience in a customer-facing technical project management persona with proven ability to deliver on time and on budget.
- Understanding of software development methodologies, including **Agile software development**[1].
- Bank experience either internally or as a vendor.
- Experience with project management methodologies and tools.
- Ability to work through problems independently and engage the team as needed.
- Effective communication and documentation skills.
- PMI and CSM certifications desirable.

[1]Agile software development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer/end user.

Objectives and Goals of a Journey Implementation/Project Manager

- Deliver on specific business value expected.
- Create professional services **SOWs**[1].
- Advise client on the best implementation methodology and adjust to deliver the best customer outcome. Contribute domain expertise to business discussions.
- Define new processes to increase team efficiency and improve the delivery cycle.
- Ensure the project keeps to time and budget constraints.
- Ensure Quality Assurance checks are complete.
- Ensure processes are followed and documentation is maintained.

Next, learn about the Platform Developer persona.

[1] A statement of work (SoW) is a document routinely employed in the field of project management. It defines project-specific activities, deliverables and timelines for a vendor providing services to the client.

Journey Champion/System Owner

Journey Platform | Champion / System Owner

A Journey Champion/System Owner is a client resource that recognizes the value of the Temenos Journey Platform and already has a relationship with our sales team. They understand the business requirements and can, at a high-level, map the requirements to capabilities of the Journey platform. This may be a technical or non-technical resource. Their job is to support the Temenos team in the presales phase as well as other divisions and decision makers.

Prerequisite Skillset of a Journey Champion/System Owner

- Ability to describe use-cases for the Journey platform.
- Ability to describe – at a high-level - the value proposition of the Journey platform.
- Able to describe – at a high-level - the technologies used in the Journey platform.
- Accesses to decision makers.
- Effective communication or a thought leader for technology in the company.

Objectives and Goals of a Journey Champion/System Owner

- Successful implementation of a Journey solution.
- Introduce the benefits of the platform to other members of the company.

Next, learn about the Form Builder persona.

Journey System Administrator

Journey Platform | System Administrator

A Journey System Administrator is responsible for the deployment of applications from Staging to Production and will work closely with the Journey System Manager/DevOps persona.

Example tasks of a Journey System Administrator include:

- Deploy new Journey applications.
- Set up and monitor daily alerts on the Staging and Production environments.
- Notify the business and other stakeholders of any issues that will affect regular operations such as planned and unplanned outages.

Prerequisite Skillset of a Journey System Administrator

- Journey Manager skills.
- Knowledge of Journey authoring tools.
- Excellent problem-solving skills.
- Good verbal and written communication skills.
- Outstanding customer service skills.
- Knowledge of cloud-based applications.

Objectives and Goals of a Journey System Administrator

- Ensure that the Journey platform and their applications are performing within acceptable parameters.
- Ensure that the correct resources, based on their roles and the environment (Dev, UAT, etc.), have access to the system.
- Work with the Journey System Manager/DevOps persona.

Next, learn about the System Manager/DevOps persona.

Journey System Manager/DevOps

Journey Platform | System Manager / DevOps

A Journey System Manager/DevOps is responsible for the management and maintenance of various Journey Manager (Dev, UAT, etc.) servers. They will provision users, deploy applications, receive notification of issues, and provide internal support as well as liaison with Temenos support should any issues arise. They are also responsible for training on the day-to-day usage of the platform (i.e. assisting with testing, reports, and reading logs). The role is technical in nature, but does not involve development; however, they will need to have a strong understanding of the Temenos Journey Platform and web applications in general.

Example tasks of a Journey System Manager/DevOps include:

- Responsible for governance and processes for the system.
- Liaise with the business stakeholder on new developments.
- Manager vendors.
- Approve requests for Temenos Assist support time to be used.

Prerequisite Skillset of a System Manager/DevOps

- Understand development life cycle and associated various environments (Dev, UAT Staging, and Production)
- Knowledge of the Journey platform stack and interfaces.
- Understanding of the roles and permissions when providing access to the various environments and product interfaces.
- Able to navigate the Manager dashboard and set properties.

Objectives and Goals of a System Manager/DevOps

- Become the single point of contact for questions regarding the custom application and the Journey platform.
- Cultivate a relationship with Temenos.
- Ensure the Operations team are following agreed monitoring and support processes.

Next, learn about the Template Designer persona.

Journey Support Team

Journey Platform | Support

A Journey Support Team is a technical persona focused on the maintenance of the Temenos Journey Platform. This includes monitoring system health, receiving and responding to alerts, troubleshooting by reading logs and answering technical questions about the platform, and the application(s) built on the platform. In addition to answering questions, the Support Team will need to diagnose hardware and software faults and solve technical and application problems, either over the phone or in person. They will also need to liaison with other experts on the platform or application to resolve or flag issues. The Support Team personamay also include producing procedural documentation and relevant reports.

Example tasks of a Journey Support Team include:

- Attend to any support issues arising from regular monitoring or as reported by the business.
- Ensure implementation teams are kept up to date on any new releases or patches that need to be applied to the test and development environments.
- Provide support to implementation teams when required.

Prerequisite Skillset of the Support Team

- The ability to think logically and possess a good memory of how the platform and application work.
- Effective listening and questioning skills combined with the ability to interact confidently with clients to establish what the problem is and explain or document the solution.
- The ability to work well in a team.
- Strong problem-solving skills and customer/client focus.
- Depending on the technical aspect of their persona - will they be supporting the platform and business users of the application or only the business users - a background in computer science, information technology, or experience working with software and hardware, web-based applications.

Objectives and Goals of a Support Team

- Assist users with Level 1 and Level 2 tier support issues.
- Resolve support issues as per any support agreement.

Next, learn about the System Administrator persona.

Instructor-Led Training and eLearning overview

Journey Platform | Platform Developer | Form Builder | 5.1 & Higher

Temenos offers instructor-led training and eLearning courses to help further your knowledge of the Journey Platform.

Instructor-Led Training ILT

Let our expert instructors teach you the skills you need to fully benefit from your investment. Training can be offered face-to-face or virtually.

To book a course contact Temenos Training or contact your Account Manager.

eLearning

Learn at your own pace at the new consolidated Journey Platform resources website. There are a number of online courses and videos available to view. The eLearning courses contain topics across the Journey Platform, each with a number of modules. There are also exercises throughout the courses that will allow you to practice your skills.

We recommend that you view each course in order, following the listed order of the modules. After a few modules you will be given an exercise which will allow you to practice what you have learned.

If you do not have an account please contact the Journey Knowledge Team.

Available eLearning Courses

- NEW Upgrading to Maestro v17.10
- NEW Upgrading to Maestro v18.05
- NEW Upgrading to Journey Analytics (formerly Insights) v18.05
- Maestro v5.1
- Introduction to Maestro Form Builder 1
- Introduction to Maestro Form Builder 2
- Maestro Form Builder Scripting
- Journey Manager v5.1
- Getting Started with Journey Manager
- Configure a Maestro Form in Journey Manager
- Submission ManagementForm Data Configuration
- Collaboration Jobs
- Journey Analytics (formerly Insights)
- Composer

eLearning Exercises

Exercises for eLearning courses can be found here.

Version 5.1

- Maestro v5.1 Exercises
- Journey Manager v5.1 Exercises

Version 17.10

- Maestro v17.10 Exercises

Help

For help view the Learning Center Help Guide.

If you have any questions, issues or feedback please feel free to contact the Journey Knowledge Team.

Learning Center Help Guide

Journey Platform | Platform Developer | Form Builder | v5.1 & Higher

The Temenos Learning Center has been migrated to our new consolidated Journey Platform resources website. This new consolidated website contains all our existing online training videos on products within the Journey Platform.

New consolidated Journey Platform resources website

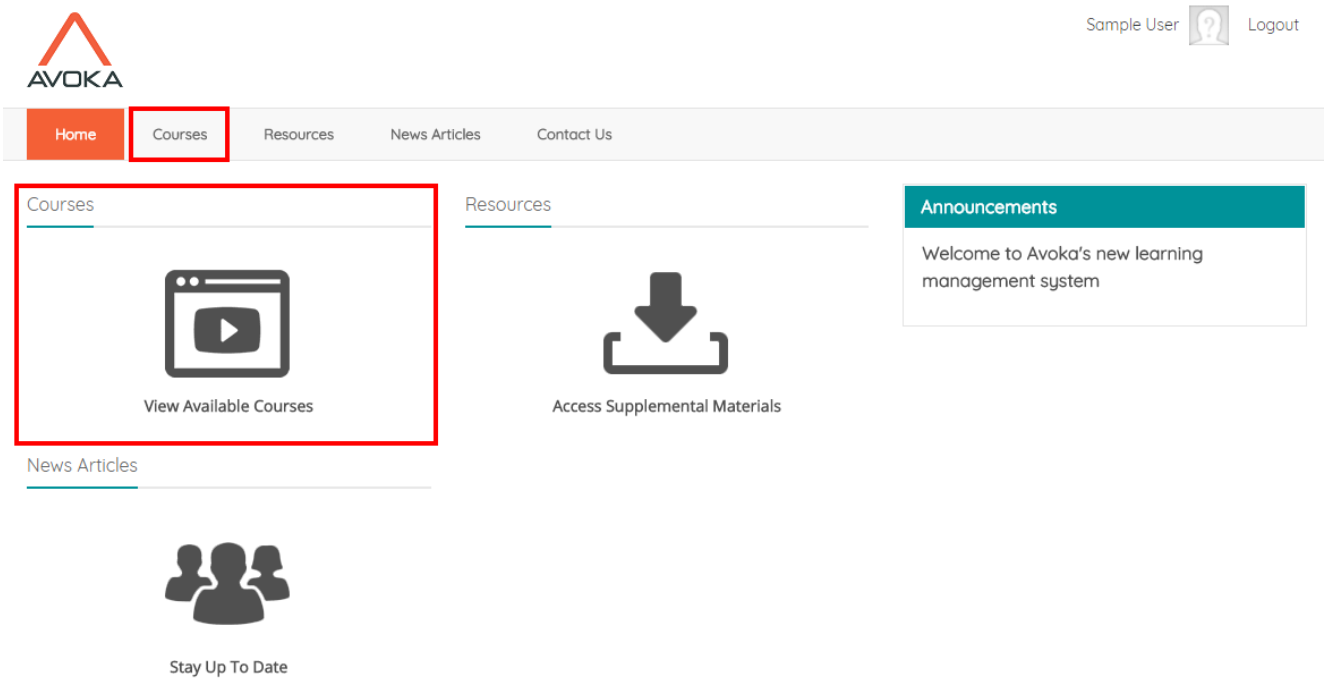
To request access to all the courses in the new consolidated Journey Platform resources website send your request to the Journey Knowledge Team. Once your access has been granted you will be able to access all the online learning materials. A video library and extensive product documentation with product overviews are also available from the Journey Platform resources website for self paced exploration.

Access the online courses at the new website - journey.temenos.com

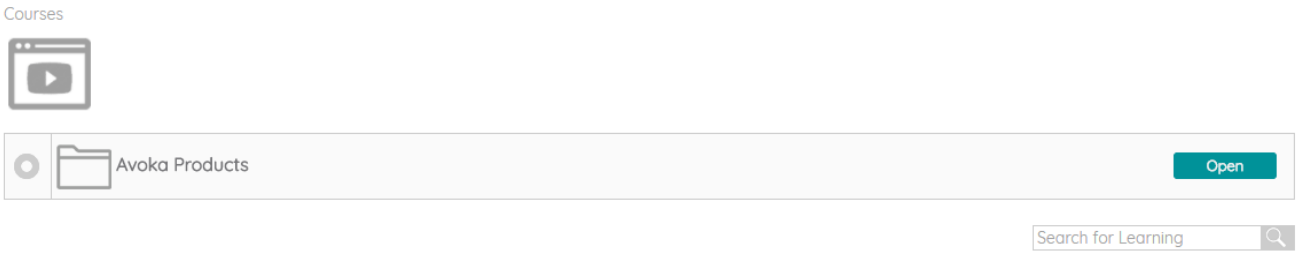
1. Simply visit the new consolidated Journey Platform resources website, or Sign In if you already have your user credentials.
2. Go to **Learn** on the main menu at the top of the page to view all courses.
3. If you don't see the **Learn** menu, send a request for access via the website, or email the Journey Knowledge Team.
4. Instructions on how to take a course can be found in the Learn article.

Access the online courses at the original location - learning.avoka.com

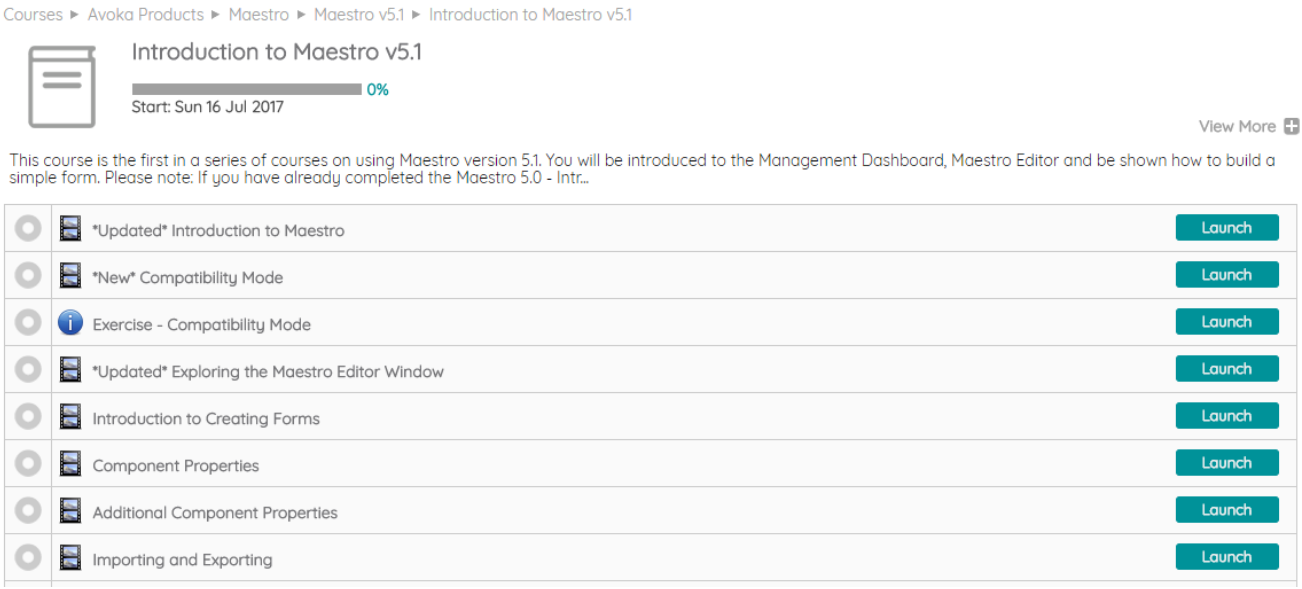
1. Login to learning.avoka.com.
2. Select the **Courses** menu or the **Courses** tile on the home page.



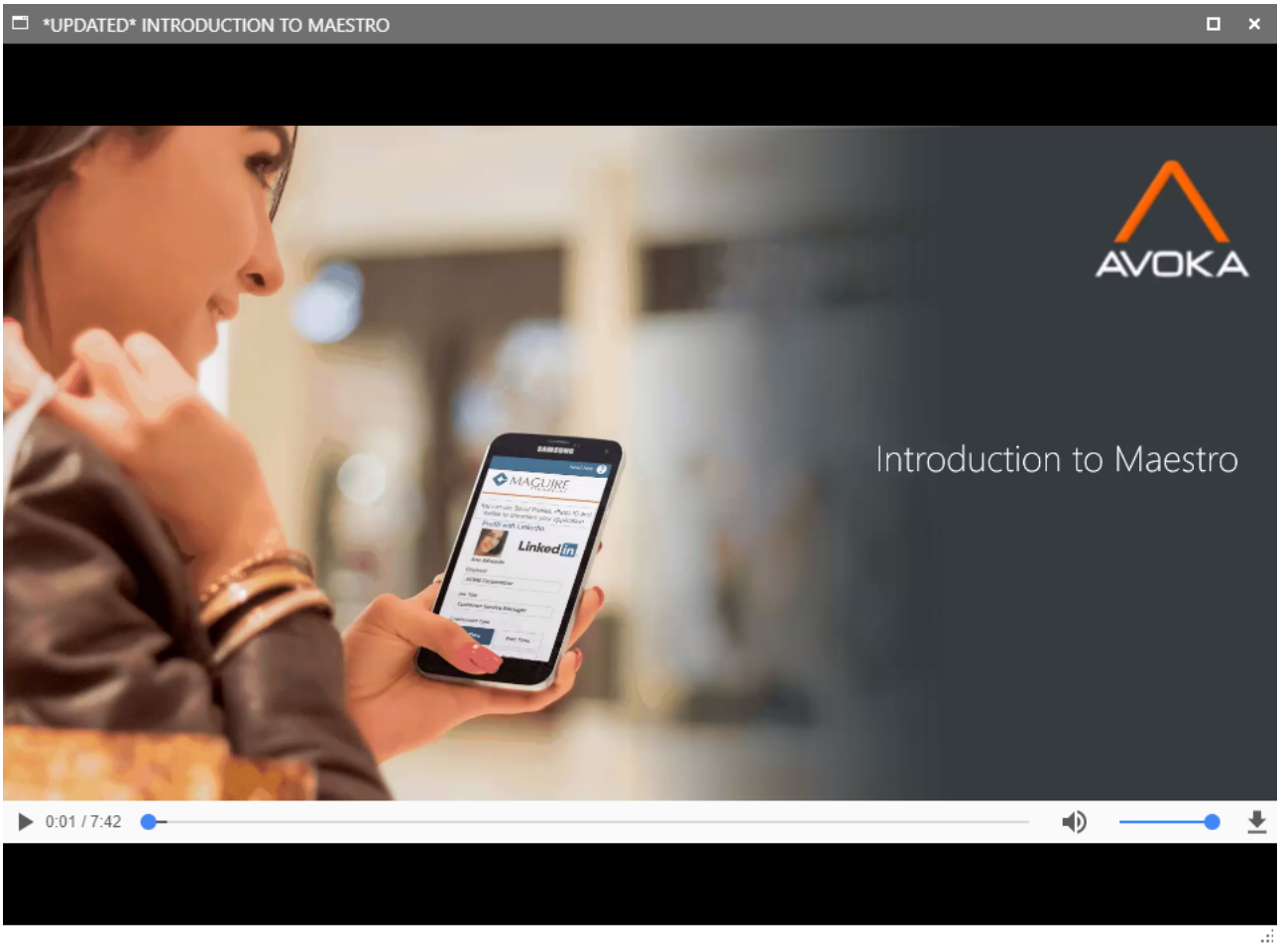
3. Click **Open** to view additional topics and/or courses.



4. Continue to **Open** the topics and/or courses, until you are viewing a course list.
5. Once you are inside the course, you will see a list of videos available to view.

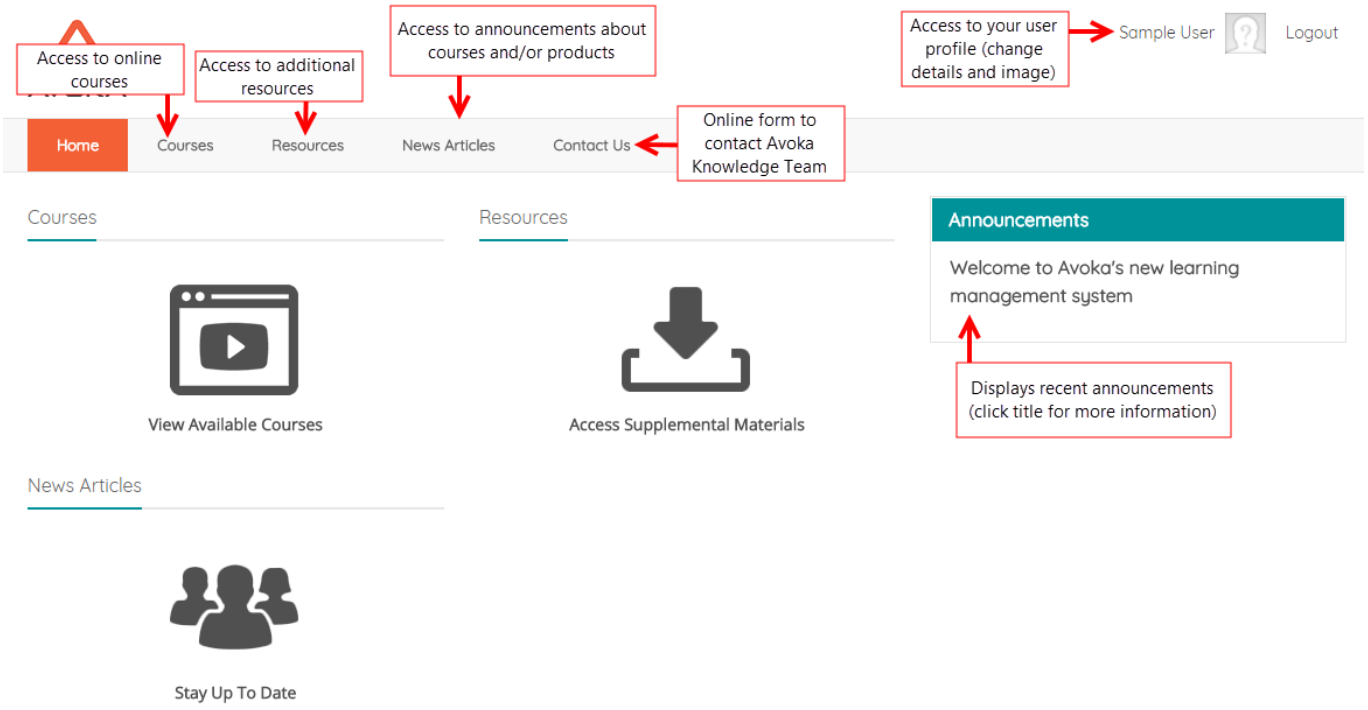


6. Click **Launch** to view the video. This will open a pop up window which will automatically begin to play the online training video.



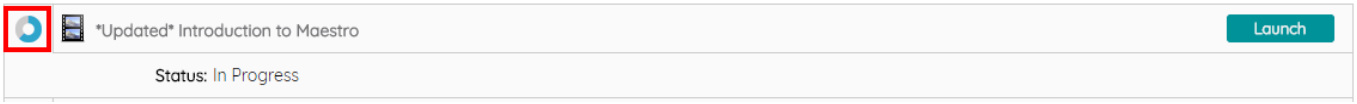
Navigate through the Temenos Learning Center

Home page and menu bar



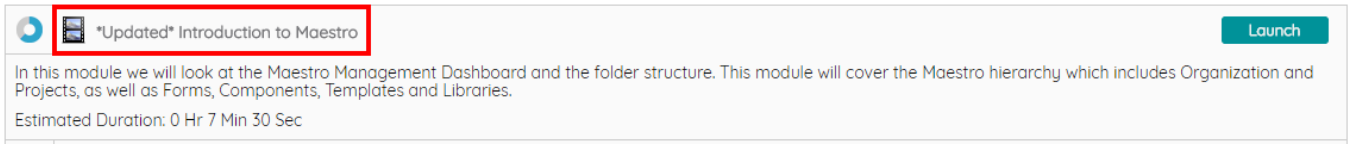
Status icon

Select the status icon to display the correct status of the course and/or online video.



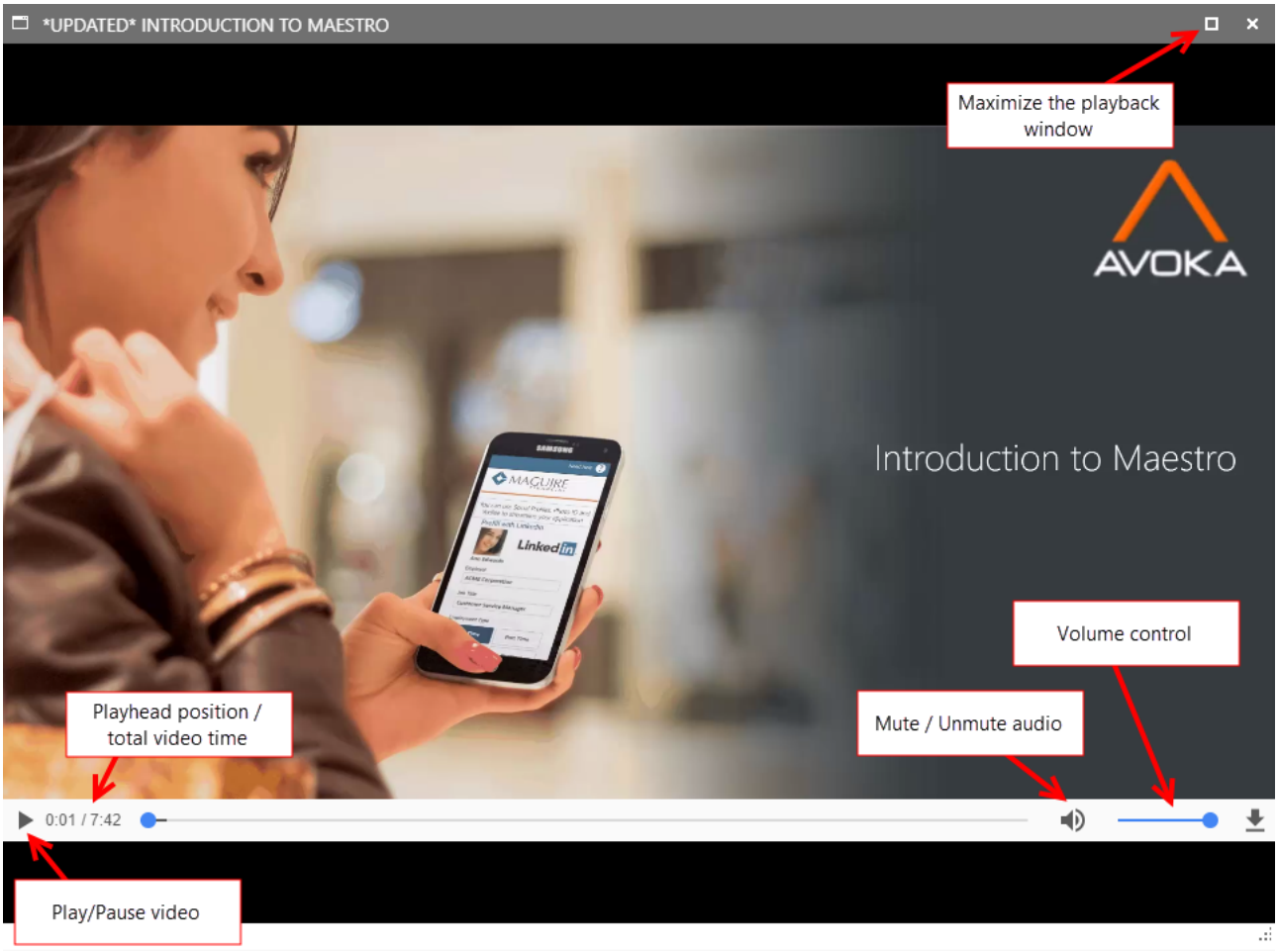
Course name

Select the course name or video name to display a description.



Launch video

Select **Launch** to open a pop up window which will automatically begin to play the online training video.


UPDATED INTRODUCTION TO MAESTRO
Maximize the playback window
AVOKA
Introduction to Maestro
Volume control
Playhead position / total video time
Mute / Unmute audio
0:01 / 7:42
Play/Pause video

eLearning Exercises for Maestro v17.10

Journey Platform | Platform Developer | Form Builder | v5.1 & Higher

View the eLearning courses available from **Learn** on the main menu of the new consolidated Journey Platform resources website.

We recommend that you view each course in order, following the listed order of the modules. After a few modules you will be given an exercise which will allow you to practice what you have learned. We strongly recommend you use these exercises to complete your training.

Pre-requisite

- First you will need to import the exercise files used for the Maestro 17.10 training exercises. Refer to Setup the Exercise Files.

eLearning Exercises for Maestro v17.10

Use the links below to jump directly to each exercise. Please follow the instructions for each exercise to identify which resources and assets you'll be using.

Upgrade to Transact Release 17.10:

- Setup the Exercise Files

Whats new?

Maestro ID Management:

- Exercise - ID Management
- Exercise - Duplicate IDs

Impact Analysis:

- Exercise - Impact Analysis

Entities:

- Exercise - Create an Entity (ID field)
- Exercise - Create an Entity (Entity panel)
- Exercise - Delete an Entity
- Exercise - Assign an Entity to multiple items

Semantic Versioning:

- Exercise - Semantic Versioning

Compatibility Mode:

- Exercise - Upgrade a project

Help

For help using the Learning Center refer to the Learning Center Help Guide.

If you have any questions, issues or feedback please feel free to contact the Journey Knowledge Team.

Exercise Setup - Importing the Maestro v17.10 Exercise Files

Journey Platform | Platform Developer | Form Builder | v5.1 & Higher

Follow the steps below to import the Maestro Project Exercise files.

- Use the **Import Project** button to import Version 17.10 Projects or later.
- For older projects exported from a 5.1.x environment use the **Import Legacy Project** button.

Step 1: Download the Exercise Project

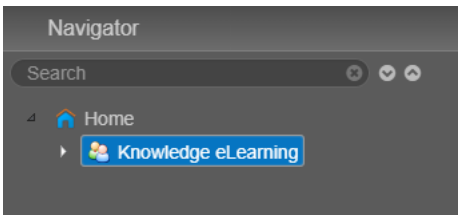
If you haven't done so already, download the Maestro Exercise Project.

1. For Version 17.10 projects or later, click Maestro_Project_Upgrade_17.10_20180424 to download the exercise project (zipped folder).
2. For older projects exported from a 5.1.x environment, click Maestro_Project_Legacy51_20180424 to download the exercise project (zipped folder).
3. Leave this file in the **Downloads** folder or move it to a different location. Just remember, you'll need to navigate to this file in the next step to import it into Maestro.

Step 2: Navigate to the Maestro Organization

Login to Maestro and navigate to the Organization.

1. Display the Maestro web page.
2. Login using your credentials.
3. Expand **Home** (in Navigator panel).
4. Select your Organization.

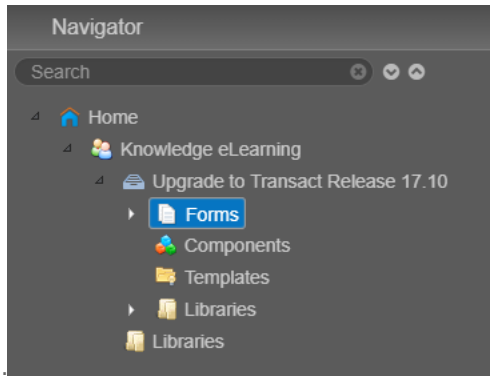


Step 3: Import Project

Import the exercise project.

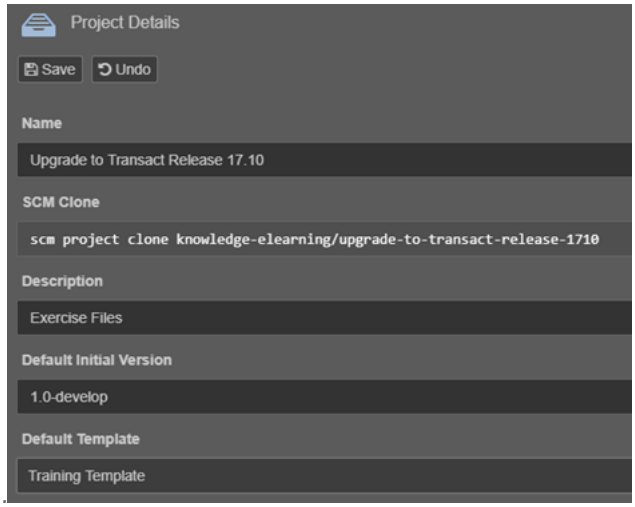
1. Click the **Import Project** button (17.10 or later) or **Import Legacy Project** button (5.1.x).
2. Enter **Upgrade to Transact Release 17.10** for the project name.
3. Click **Import**.
4. Navigate to where the downloaded Maestro Project (zipped folder) is saved.

5. Select the project and click **Open**



Step 4: Set the Default Initial Version and Default Template

1. Important: make sure the Project name is selected (i.e. Upgrade to Transact Release 17.10).
2. Click the **Project Details** tab (top right).
3. Enter **1.0-develop** as the Default Initial Version.
4. Select **Training Template** for the Default Template.
5. Click **Save** to finish



Once you've setup the project, you can access all the form, template and component assets.

Assign an Entity to Multiple Items Exercise v17.10

Journey Platform | Form Builder | v5.1 & Higher

You can assign an Entity to multiple items at once using the Structure panel.

Extra help is available at the end of this exercise if you get stuck.

Step 1: Add a new Section and Components

1. Display Page Two of the 'Exercise – Entities' form
2. Add a new section to the form and call it **HomeAddress**.
3. Add the following 'text field' components to 'HomeAddress':
   - StreetNumber
   - StreetName
   - Suburb
   - PostCode
   - Country
4. From the Entities panel, confirm the new components are 'Not Assigned' to any Entities

Step 2: Assign an Entity to multiple items

1. To relate an Entity to multiple components, you need to assign the Entity to the section first, i.e. where all the components are located.
2. Make sure the 'Home Address' section is selected.
3. Use the ID property field to assign the location Entity to the Section.
4. Display the **Structure** panel > right-click the **homeAddress** section and choose **Apply Entity Path to Children**.
5. From the 'Entities' panel, confirm all components are now related to the 'location' Entity
6. Select each component in the form and confirm the IDs have all been assigned to the **location** Entity.
7. Click **Save**, then close the form (Browser Tab).

Extra Help

Download the Exercise PDF for more detailed instructions to help you complete the exercise.

Create an Entity in the ID Field Exercise v17.10

Journey Platform | Form Builder | v5.1 & Higher

This exercise will demonstrate how to create an entity directly in the ID field.

Extra help is available at the end of this exercise if you get stuck.

Step 1: Create a Form

1. Create a new form and enter the follow details:
   - Form Name: Exercise - Entities
   - Description: This form is to demonstrate how to work with entities
   - Template: Training Template
2. Click **Create**.

Step 2: Create an Entity

1. Add a new 'Text Field' into 'About You'
2. To create a new Entity, enter 'applicant' followed by an underscore, '_'

> After the underscore (_) an inline warning displays 'An Entity called applicant will be created'

3. Enter 'FirstName' as the ID (then TAB or click away)

> The Label only displays 'First Name'

4. Display the 'Entities' panel and confirm the 'applicant' Entity has one related component (i.e. 'FirstName')
5. Click **Save**.

Step 3: Assign an Entity

1. Add a new 'Text Field' component below 'First Name'
2. Enter the first few letters of 'applicant' in the ID property field, i.e. 'app'

> Any matching Entities will display in dropdown

3. Select the 'applicant' Entity
4. Enter 'LastName' (after underscore) for the ID

Note: the label only displays 'Last Name'

5. Display the Entities panel and confirm the applicant Entity now has two related components.

Step 4: Create a nested Entity

1. Add another new 'Text Field' below 'Last Name'
2. Enter 'app' in the ID property field and select the 'applicant' Entity
3. Enter 'work_' as the new nested Entity, e.g. 'applicant_work_'

After the second underscore (_) an inline warning displays 'An Entity called work will be created'

4. Enter 'Email' for the ID, e.g. 'applicant_work_Email'
5. Press Tab or click away

Label only displays 'Email'

6. Display the Entities panel and confirm the new nested work Entity with one related component.
7. Click **Save**.

Existing IDs with an underscore will not automatically cause Entities to be created. Entities are only created when you modify an existing ID.
Any underscores followed by one or more digits (like suffixes applied by auto-generated IDs) are also ignored.

Extra Help

Download the Exercise PDF for more detailed instructions to help you complete the exercise.

Create an Entity in the Entity Panel Exercise v17.10

Journey Platform | Form Builder | v5.1 & Higher

Create a new Entity in the Entities panel by right-clicking where you want the new Entity to be placed.

Extra help is available at the end of this exercise if you get stuck.

Step 1: Create an Entity

1. Make sure the **Exercise – Entities** form is displayed in **Design** view.
2. From the Entities panel, right-click the 'Entities' icon and choose 'Add Entity'
3. Enter 'location' for the Entity name and click Save
4. Confirm the new location Entity now appears in the panel.

Step 2: Create a nested Entity

1. From the Entities panel, right-click the 'location' Entity and choose 'Add Entity'
2. Enter phone for the new Entity name and click Save.
3. Expand the location Entity to display the new nested phone Entity.
4. Add a new 'Text Field' component to the form below 'Email'
5. Enter the ID as 'HomePhone' and relate this component to the nested 'phone' Entity
6. From the 'Entities' panel, confirm 'Home Phone' is related to the nested 'phone' Entity
7. Click **Save**.

Extra Help

Download the Exercise PDF for more detailed instructions to help you complete the exercise.

Delete an Entity Exercise v17.10

Journey Platform | Form Builder | v5.1 & Higher

Note that you can't delete an Entity if any component IDs use that Entity.

Extra help is available at the end of this exercise if you get stuck.

Step 1: Delete an Entity

1. To delete the 'phone' Entity, you must delete or re-assign any related components
2. For this exercise, delete 'Home Phone' on the form
3. From the Entities panel, confirm the 'phone' Entity no longer has any related component IDs
4. Right-click the phone Entity and choose Delete Entity.
5. Click **OK** to confirm.
6. Click **Save**.

Extra Help

Download the Exercise PDF for more detailed instructions to help you complete the exercise.

Duplicate IDs Exercise v17.10

Journey Platform | Form Builder | v5.1 & Higher

This exercise will demonstrate the new duplicate ID functionality. You will:

1. Create a duplicate ID and Review
2. Create a duplicate ID and Accept

Extra help is available at the end of this exercise if you get stuck.

Step 1: Add a component with a duplicate ID and Review

1. Make sure the form you created for the ID Management exercise is open.
2. Add a new **Text Field** beneath First Name.
3. Update the ID to **FName**.

> This ID was deliberately used to create a duplicate ID.

4. An inline warning confirms **Value generates to a duplicate ID**.

> IDs are case sensitive - if you enter anything other than **FName**, the inline warning will not display.

5. At this point, you can either modify the ID directly (in the property field) ... or (click away to) **Review** or **Accept** the update.
6. For this example, click **Review** and update the ID to **MiddleName**.

> Modifying the ID means it doesn't cause a duplicate.

Step 2: Add a component with a duplicate ID and Accept

1. Add another new **Text Field** beneath 'Middle Name'.
2. Update the ID to **MiddleName** and click away.
3. For this example, click **Accept** in the popup.

> Choosing **Accept** appends a suffix to the ID, i.e. middleName_1.

4. Click **Save** and close the form (i.e. the browser tab).

If you ‘Accept’ a duplicate ID, ‘XML Name’ and ‘Full Path’ (‘Integration’ panel) keep their default names (i.e. ‘TextField’). To update these properties, uncheck ‘Use Auto-generated XML Name’ and manually update the ‘XML Name’.

Extra Help

Download the Exercise PDF for more detailed instructions to help you complete the exercise.

ID Management Exercise v17.10

Journey Platform | Form Builder | v5.1 & Higher

This exercise will demonstrate the updated functionality for component IDs. You will:

- create a new form
- add a new component
- update the component ID

Extra help is available at the end of this exercise if you get stuck.

Step 1: Create a new form

1. Create a new form using the following details:
   - Form Name: Exercise - ID Management
   - Description: This form is to demonstrate the new ID features
   - Template: Training Template

Step 2: Add a new Text Field component

1. Add a new **Text Field** into the **About You** section.

> The initial focus is set to ID Property field.

Step 3: Update the Text Field component

1. Update the **ID** Property value to **FirstName**.

> The Label Property has automatically updated to **First Name**.

2. Display the **Integration** pane.

> The XML Name and Full Path properties have updated to the new ID.

3. Display the **Properties** pane and update the ID to **FName**.

> The 'Label' remains unchanged and only updates when the initial 'ID' is entered.

4. Display the **Integration** pane.

'XML Name' and 'Full Path' have updated to the latest 'ID' (i.e. 'FName').

5. Click **Save**.

If 'Use Auto-generated XML Name' is not checked ('Integration' panel), 'XML Name' and 'Full Path' won't update if the ID changes and will need to be manually updated.

Extra Help

Download the Exercise PDF for more detailed instructions to help you complete the exercise.

Impact Analysis Exercise v17.10

Journey Platform | Form Builder | v5.1 & Higher

The following exercise will involve the Order form.

Extra help is available at the end of this exercise if you get stuck.

Step 1: Open the Order form

1. Open the **Order Form**.

Step 2: Review the current references

1. Right-click the First Name text field and choose **Show References to firstName**.
2. **Confirm** one existing reference to the FullName item.
3. **Close** the dialog.

Step 3: Add a new Text Field that references First Name

1. Display **Page Two** of the form.
2. Add a new Text Field and update the ID to **FirstNameP2**.
3. Update the Label to **First Name**.
4. With the new Text Field selected, click **Create Rule**.
5. Click **Script** (under Calculation).
6. In the dialog, expand Page One, then expand the first Section.
7. Double-click **First Name** to setup a reference to this component.
8. Click **Save** then save the form.
9. Preview the form, then enter a sample First Name on Page One, then confirm it displays in First Name on Page Two.
10. Switch back to Design view.

Step 4: Review the updated references

1. Right-click **First Name** on Page One and choose Show References to First Name.
2. Confirm a second reference to firstNameP2 (on Page Two) now exists.
3. **Close** the dialog.
4. **Save** the form, then close it (i.e. the Browser Tab).

Extra Help

Download the Exercise PDF for more detailed instructions to help you complete the exercise.

Semantic Versioning Exercise v17.10

Journey Platform | Form Builder | v5.1 & Higher

This exercise will demonstrate how to create multiple semantic versions of a form.

Extra help is available at the end of this exercise if you get stuck.

Step 1: Update the 1.0-develop version

1. Select the **Exercise - Entities** form.
2. Add a new text field to 'Getting Started' called 'PhoneNumber'(below 'Email')
3. Click **Save** and Close the form (Browser Tab).

Step 2: Create a Tagged version (based on 1.0-develop)

1. Select the **Exercise - Entities** form in the Navigator panel.
2. Click the **New Tag** button.
3. Confirm the new Semantic Version number is **1.0.1**.
4. Enter the Description as: This is the first tagged version.
5. Click **Create**.
6. Expand the **Exercise – Entities** form in the Navigator panel and confirm the new tagged **1.0.1** version is displayed.

Step 3: Open the Tagged version

1. Select the **Exercise - Entities** form in the Navigator panel.
2. Select the **1.0.1** tagged version (note the tick icon).

> Note: the Palette panel is empty, i.e. you can't add or delete components

3. Close the form (Browser Tab).

Step 4: Update the 1.0-develop version

Remember: all changes and updates are completed in 1.0-develop.

1. Open the **1.0-develop** version of the **Exercise - Entities** form.
2. Add a new text field to the Getting Started section called FaxNumber.
3. Click **Save** and Close the form (Browser Tab).

Step 5: Create another Tagged version (based on 1.0-develop)

1. Select the **Exercise - Entities** form in the Navigator panel.
2. Select **1.0-develop** and click **New Tag**.
3. Confirm the new Semantic Version number is **1.0.2**.
4. Enter the Description as: This is the second tagged version.
5. Click **Create**.
6. Expand the **Exercise – Entities** form the Navigator panel and confirm the tagged 1.0.2 version is displayed.

Extra Help

Download the Exercise PDF for more detailed instructions to help you complete the exercise.

Upgrade a Project Exercise v17.10

Journey Platform | Form Builder | v5.1 & Higher

You can upgrade an existing project to the latest Maestro release from the Management Dashboard.

Extra help is available at the end of this exercise if you get stuck.

Step 1: Import a Legacy project

To upgrade a project, you'll need to add a legacy project to your organization first.

1. If you haven't done so already, download the sample Legacy Project.
2. Navigate to the Maestro webpage and select the Organization
3. Click **Import Legacy Project**.
4. Name the project **Legacy Project** and click **Import**.
5. Navigate to the location of the downloaded project.
6. Select the project and click **Open**.
7. Confirm the imported project now appears under your organization.

Step 2: Upgrade the project

1. Select **Legacy Project** in the Navigator.
2. Click the **Project Details** tab.
3. Update the **Release Version** to **17.10.0**.

> When you select a new version, a warning message confirms you can't change the release version back to a previous version once the Project Details are saved.

4. Click OK in the warning message, then click **Save**.
5. The project has now been upgraded.

> Note : The Release Version dropdown no longer displays any previous release versions.

Extra Help

Download the Exercise PDF for more detailed instructions to help you complete the exercise.

eLearning Exercises for Maestro v5.1

Journey Platform | Platform Developer | Form Builder | v5.1 & Higher

View the eLearning courses available from **Learn** on the main menu of the new consolidated Journey Platform resources website.

We recommend that you view each course in order, following the listed order of the modules. After a few modules you will be given an exercise which will allow you to practice what you have learned. We strongly recommend you use these exercises to complete your training.

Pre-requisite

- First you will need to import the exercise files used for the Maestro 5.1 exercises. Refer to Exercise Setup - Importing the Maestro Exercise Files.

eLearning Exercises for Maestro v5.1

Use the following links to view the exercise guides. Please follow the instructions in each guide to identify the resources and assets you'll need to use.

Introduction to Maestro 5.1:

- Compatibility Mode Exercise
- Create a New Form Exercise
- Pages and Sections Exercise

Form Builder 1:

- Layout Exercise
- Creating Rules (without Scripting) Exercise
- Repeating and Shared Components Exercise

Form Builder 2:

- Creating Shared Components Exercise
- Two Level Navigation Exercise

Form Builder Scripting:

- Calculation Rules Exercise
- Creating Rules Exercise
- Validation Rules Exercise

Help

For help using the Learning Center refer to the Learning Center Help Guide.

If you have any questions, issues or feedback please feel free to contact the Journey Knowledge Team.

Exercise Setup - Importing the Maestro v5.1 Exercise Files

Journey Platform | Platform Developer | Form Builder | v5.1 & Higher

This guide explains how to import the Maestro exercise files as a Maestro project. This project contains all the required forms, templates, components and styles needed to complete the exercises.

You can also access a training module on the Avoka LMS that explains how to import and export in Maestro. You may want to view this module before importing the exercise project.

Step 1: Identify the Training Environment

1. Identify which of the following best describes the Training Environment you will use to complete the Maestro 5.1 training.

Training Environment	Uses Maestro version	Download…	Import Type
elearning.avoka.com	17.10 or later	Version 2018.03.08	Import Project
Your company	17.10 or later	Version 2018.03.08	Import Project
Your company	5.1.x or earlier	Version 2017.06.16	Import Project

+ Version 2017.06.16: was exported from Maestro 5.1.x environment
+ Version 2018.03.08: was exported from Maestro 17.10 (or later)

Step 2: Download the Exercise Project

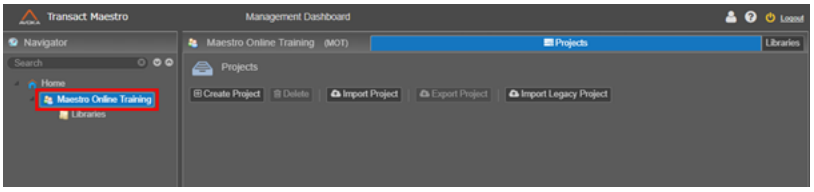
The file to be downloaded is a zipped folder that contains the Maestro training exercise project.

1. Click on the required download link above to download the Exercise Project to your PC.
2. Leave this file in the **Downloads** folder or move it to a different location. Just remember, you'll need to navigate to this file in the next step to import it into Maestro.

Step 3 Import the Exercise Project

Once you've selected the Organization, you can import a project into Maestro.

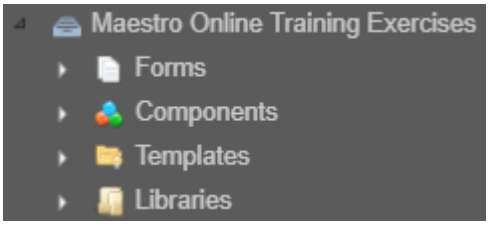
1. Login to Maestro.
2. Expand **Home** (in the Navigator panel).
3. Click on your **Organization**.



4. Click **Import Project** to import the Maestro Project.
5. Enter **Maestro Online Training Exercises** for the name of the project you're importing.
6. Click **Import**.
7. Navigate to and select the Maestro Exercise Project zipped folder you downloaded.
8. Click **Open**.

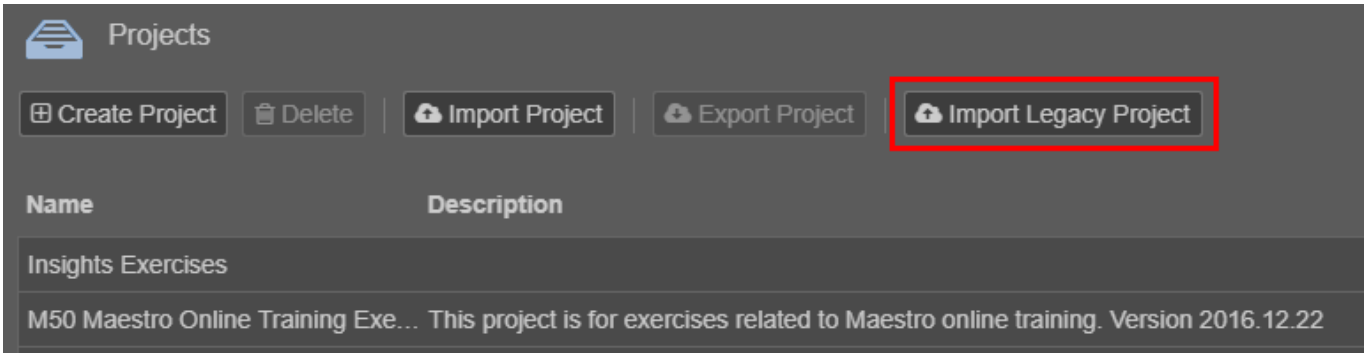
Result

After importing the project you'll have access to the exercise forms, templates and components.



Optional Step 4 Importing Legacy Projects

Maestro offers an additional import option for projects created and exported from legacy (or previous) versions of Maestro.



A project is considered Legacy if it was created and exported from any version of Maestro prior to v17.10. The option to Import Legacy Projects is only available to Maestro users assigned the **Maestro Site Administrator** Role. If you need to import a legacy project, speak to your Maestro account manager.

Step 5 Update Release Version

Once you've imported the Exercise Project, you may need to update the Default Initial Version and assign a Release Version (if required) in the Project Details.

Assigning the Release Version associates the version of Maestro used to access the forms in your imported Maestro project.

1. Select the imported project in the **Navigator** panel.

2. Click the **Project Details** tab.

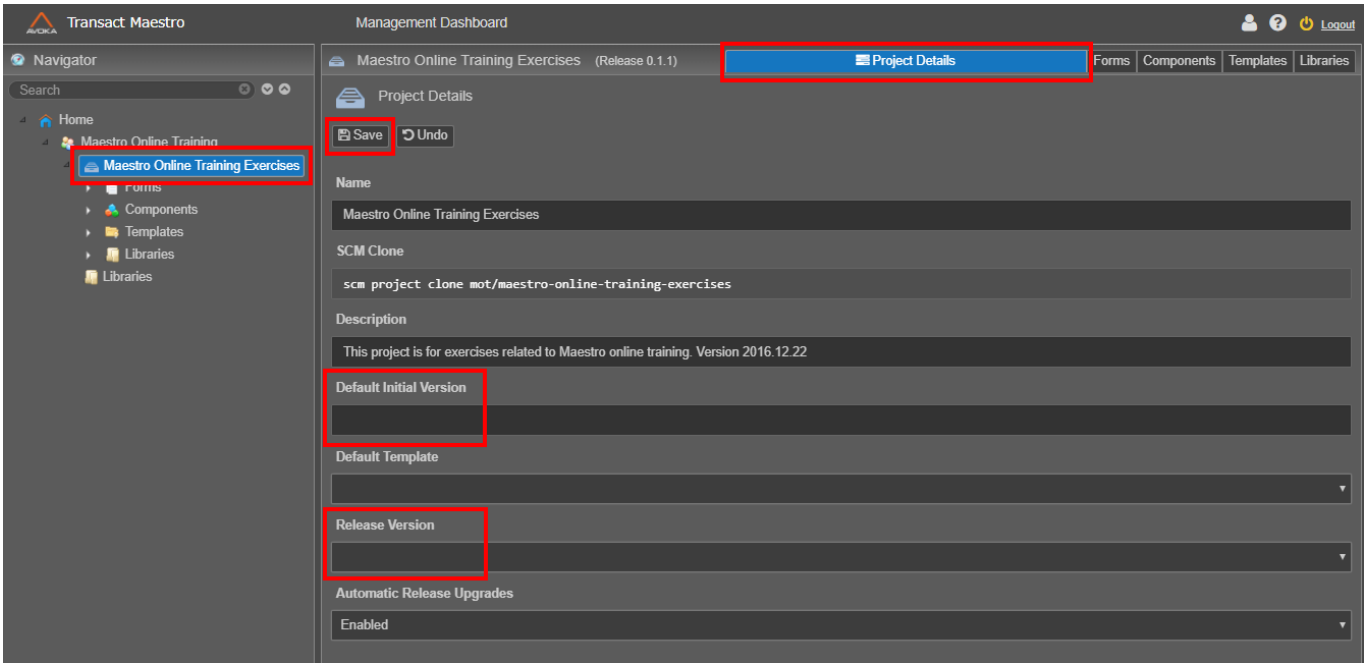
3. Enter **1.0-develop** for the **Default Initial Version**.

> For more information on the Default Initial Version see Form Versions.

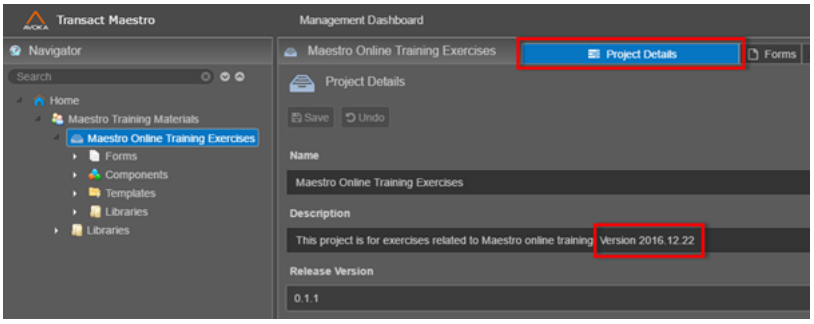
4. Select **5.1.4** (or the latest available version beginning 5.1.x) for the Release Version.

> Release Versions for Maestro Projects prior to v5.1 (i.e. v5.0) were named using a 1.x.x naming convention. For Maestro 5.1 projects, the Release Version followed a 5.x.x. naming convention. Starting with the v17.10 release, all Release Versions are based on a Year.Month naming convention, e.g. 17.10 = 2017 October.

5. Click **Save**.

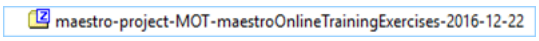


The date a Maestro Project is exported can be found on the Project Details tab, in the Description Field (as long as it's not been changed).



Any Maestro Projects exported prior to 2016.12.22 won't display a date in the Description field.

Instead, the export date can be found at the end of the zipped folder filename:



Extra Help

Need some extra help?

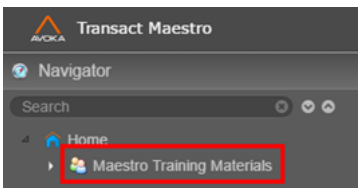
Use the information below to assist you in completing the exercise steps.

Importing a Project

When you import a project, that includes all the resources and assets within the project, e.g. the forms, templates, components, styles, and libraries etc.

To import a project, you need to have the Maestro Organization selected first.

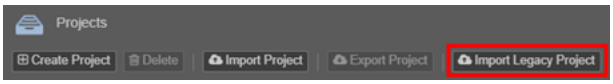
Once you login to Maestro, expand Home, then select the Organization in the Navigator panel (left side of the window).



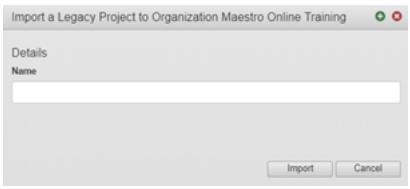
Once you select the Organization, a list of all the projects displays on the right.

The **Import Project** button is situated just above the list.

Use Import Legacy Project for Projects exported from Maestro 5.1.x or earlier.



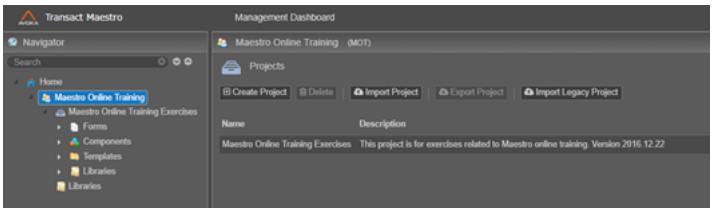
In the dialog, you must enter a name for the new project - then click **Import**.



For example, we use **Maestro Online Training Exercises** for the name.

Click **Import**, then navigate to the required Maestro Exercise project.

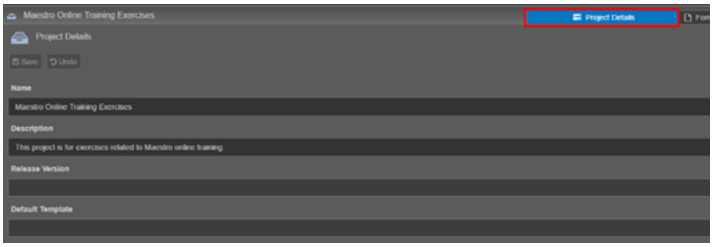
Once selected you can click **Open** and the project will be imported to your Maestro Organization.



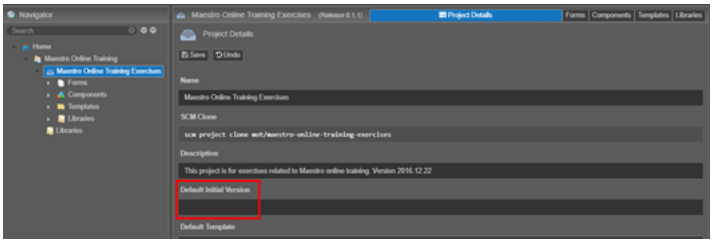
Select the project from the Navigator on the left side of the window.



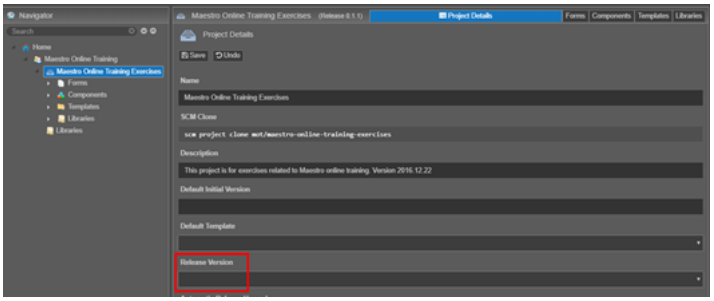
Select the **Project Details** tab.



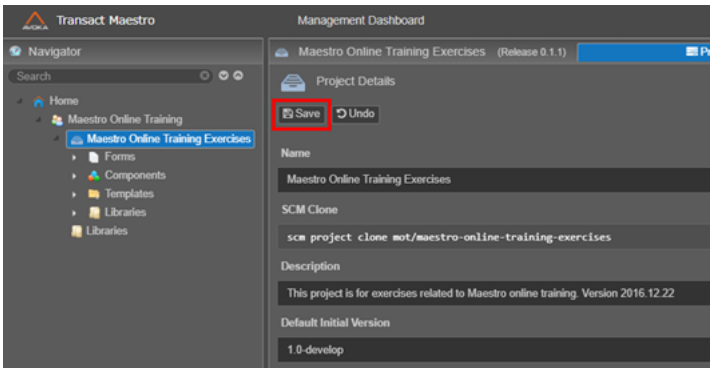
Enter a **Default Initial Version**.



Expand the **Release Version** dropdown and select the appropriate release version to apply to the project.



Click **Save**.



Calculation Rules Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating rules. During this exercise you will create a new form and add components. You will then create rules on the added components.

Do your best to follow the information provided and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find **Extra Help** at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the **Maestro Exercise Project: Maestro Training Exercises v5.1 > Calculation Rules Finish**. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Create a New Form

Create a new form using the details provided below.

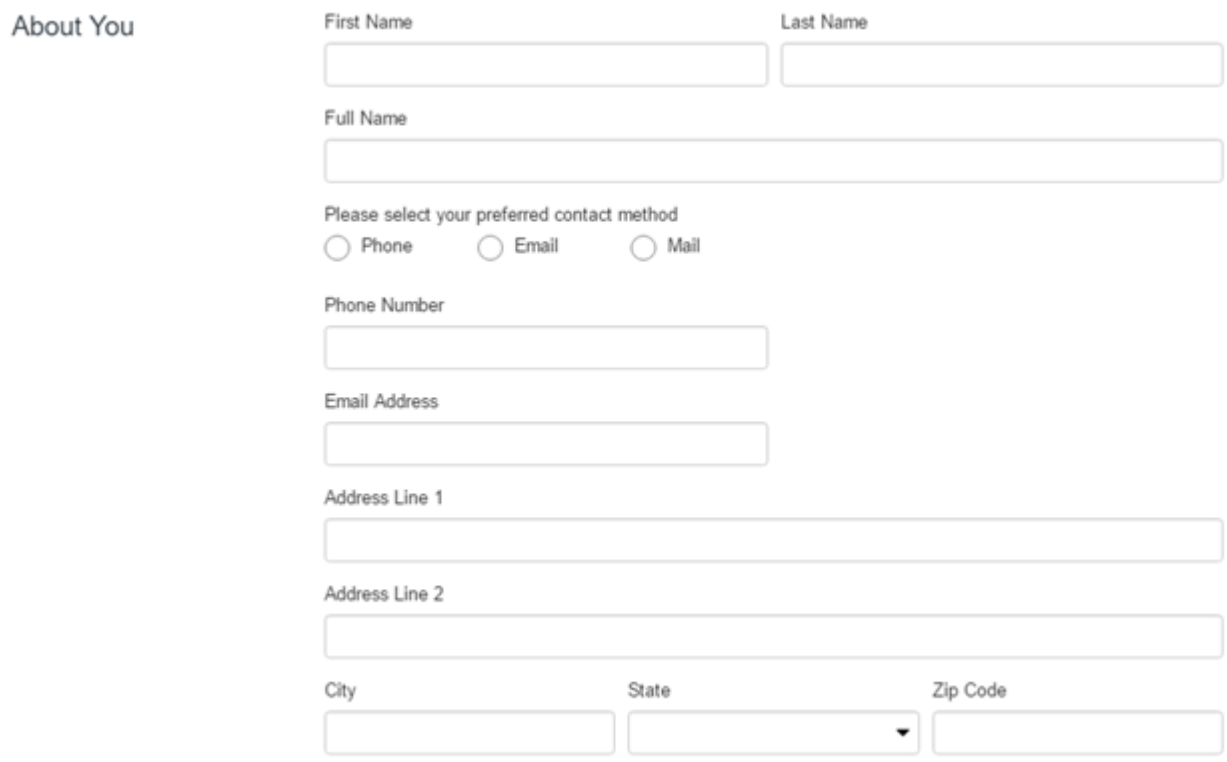
1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Click **Create Form**.
5. Enter details about the form:

- Name: Order Form
- New Version: 1.0-develop
  (should be pre-populated if Default Initial Version has been set up in Project Details)
- Description: This form is to practice creating calculation rules.
- Template: Exercise Template

6. Click **Create**.

Step 2 Add Components to Page One

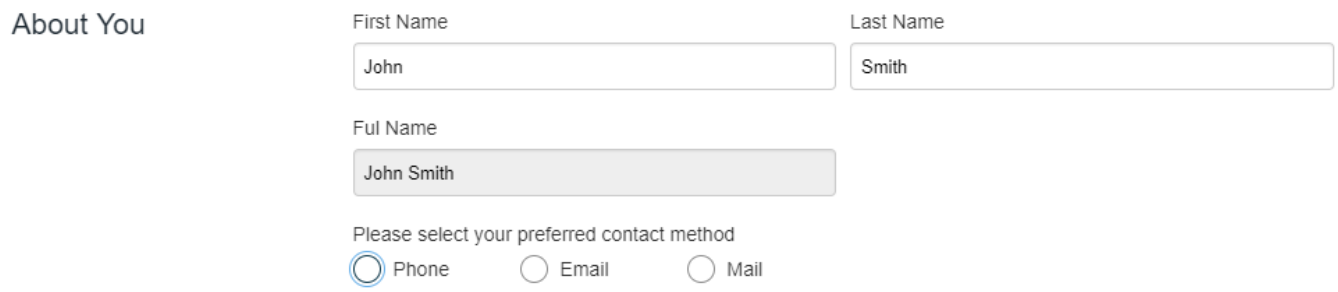
Add components to Page One so your form resembles the image below.



Click **Save**.

Step 3 Populate a Component Based on Entered Data

Populate the **Full Name** component based on a script that concatenates the data entered in **First Name** and **Last Name**.



Add a **Read Only** rule to **Full Name** so the populated data cannot be changed.

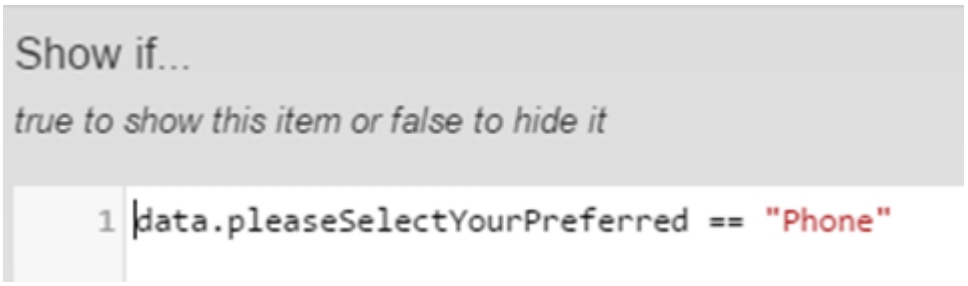
Preview the form to check the calculation rules.

Click **Save**.

Step 4 Add Show if Rules

Add **Show If** rules to the **Phone Number**, **Email Address** and **Address block** based on the preferred contact method selected.

Right-click the appropriate components on the left side of the **Create Rule** window to create the required **Show if** scripts (see the example below).

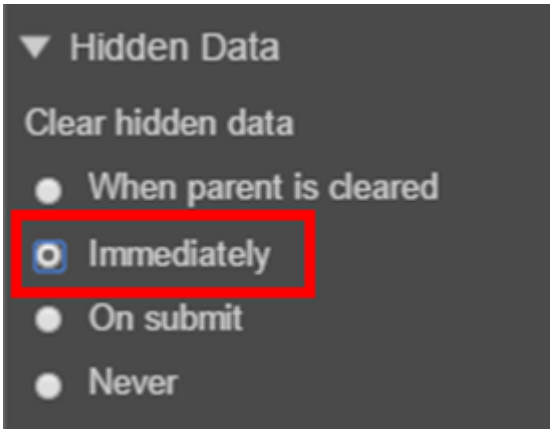


Save your form.

Preview the form to check the calculation rules.

Step 5 Change the Clear Hidden Data Option

Change the **Clear hidden data** option for each of the contact detail components so that when they are hidden, any data previously entered in the field is cleared immediately.

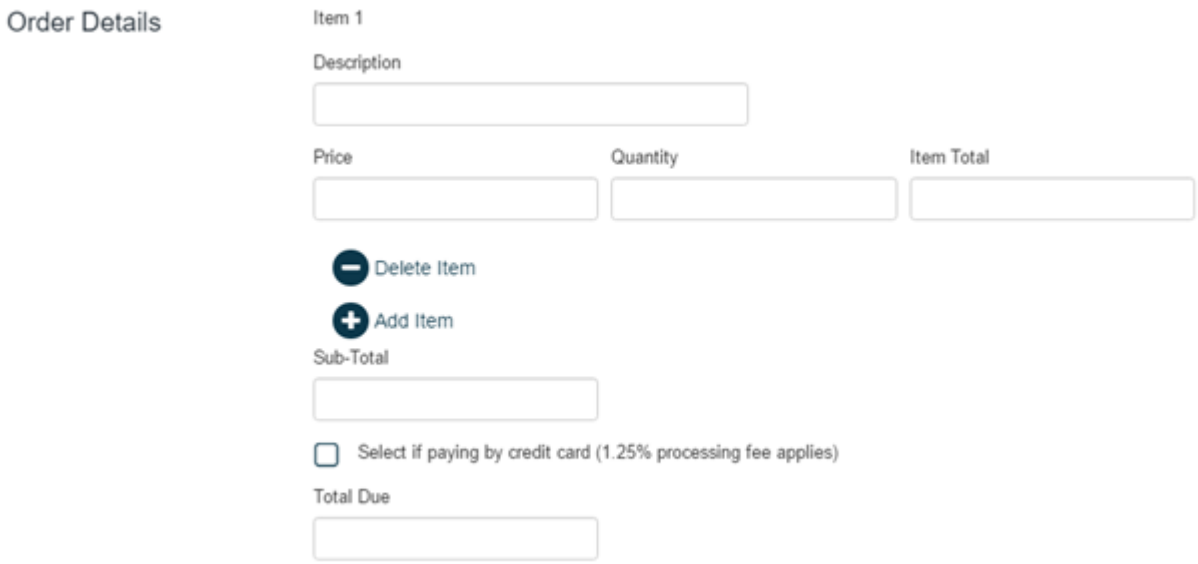


When you've changed the Hidden Data options, **Save** your form then click **Preview** to test the form.

Click **Save**.

Step 6 Add an Order Details Section to the Form

Add a new section below the **About You** section. Add components so the new section resembles the image below.

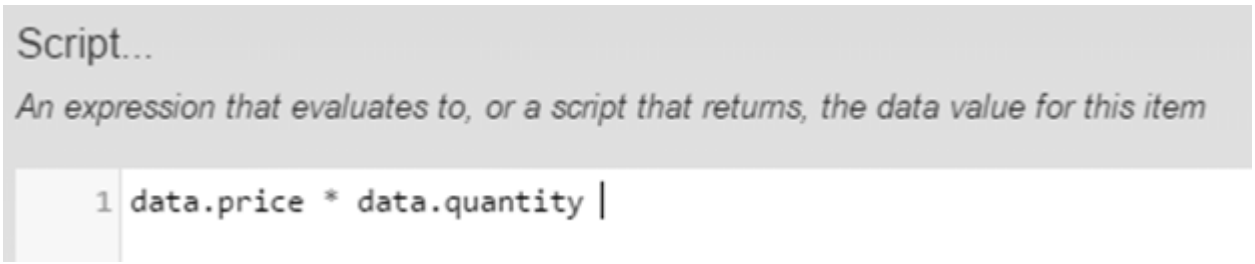


Use the **Decimal Field** component for **Price**, **Item Total**, **Sub-Total** and **Total Due** and set the decimal fields to 2 decimal places.

Click **Save**.

Step 7 Add a Calculation Rule to the Item Total

Create a calculation rule that multiples **Price** by **Quantity**.



Add a **Read Only** rule to **Item Total** so the user cannot change the calculated value.

Click **Save**.

Preview the form to check the calculation rules.

Step 8: Add a Calculation Rule to the Sub-Total Component

Create a calculation rule that sums the **Item Totals** for each repeat.

Script...

An expression that evaluates to, or a script that returns, the data value for this item

```
1 Util.sum(Util.getRepeatDataArray(Form.getItemFromPath('Form.data.content2.itemTotal')))
```

Add a **Read Only** rule so that the user cannot change the calculated value.

Click **Save**.

Preview the form to check the calculation rules.

Step 9: Add a Calculation Rule to the Total Due Component

Create a calculation rule that includes the **Sub-Total** multiplied by the credit card processing fee if the checkbox is selected.

Script...

An expression that evaluates to, or a script that returns, the data value for this item

```
1 if (data.selectIfPayingByCredit === true )
2   return data.subtotal * 1.025;
3 else
4   return data.subtotal;
```

Add a **Read Only** rule so that the user cannot change the calculated value.

Click **Save**.

Preview the form and check the calculation rules.

Extra Help

Need some extra help?

Use the information below to assist you in completing the exercise steps.

Create a Show If Rule

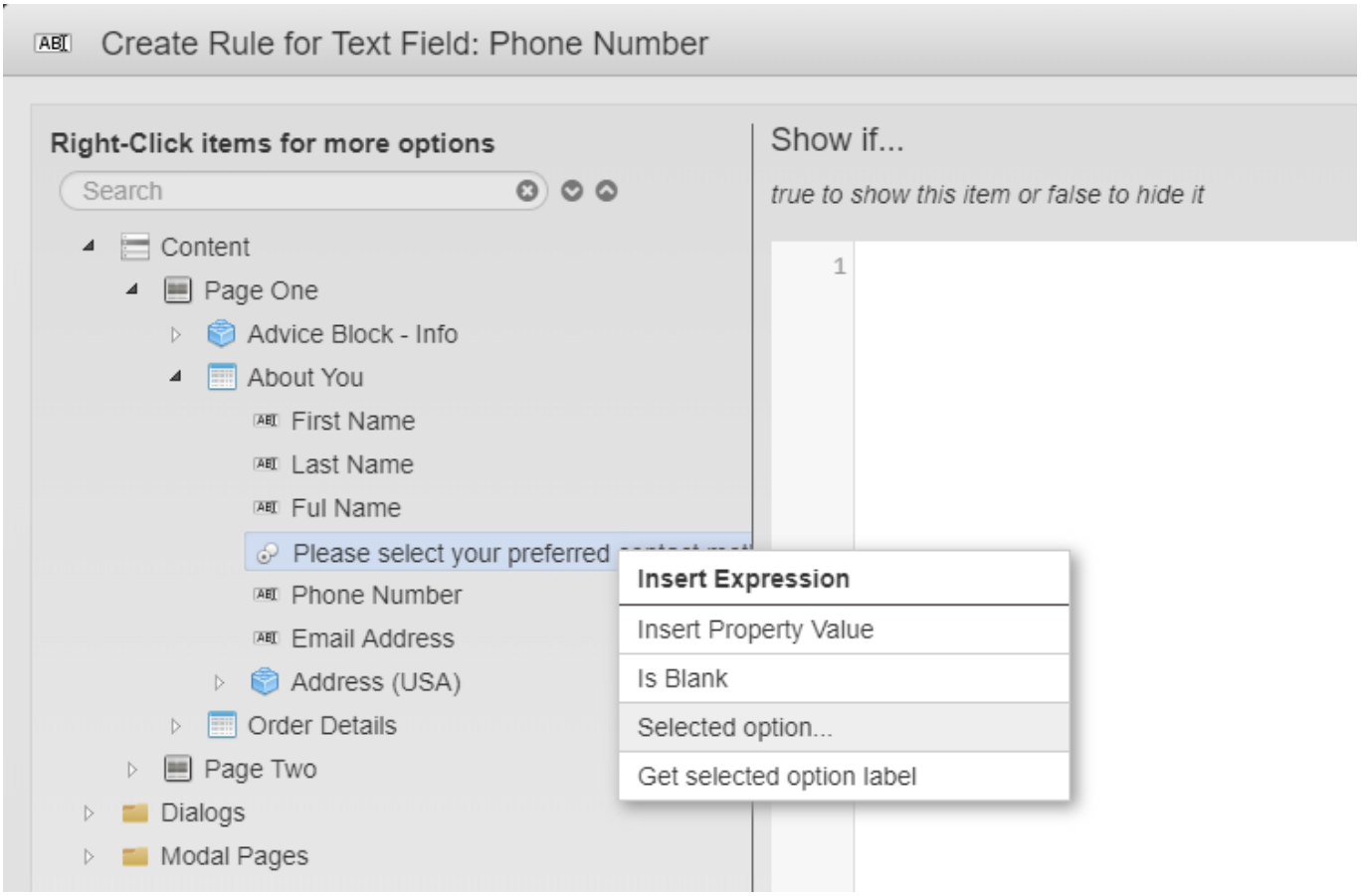
Select the component whose visibility is conditional (i.e. visible or not visible).

Click **Create Rule** from the **Rules** section (**Properties** panel).

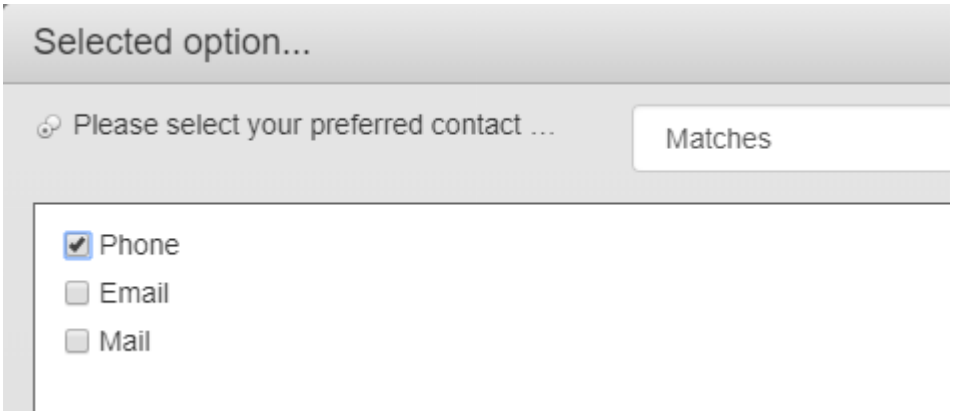
Select **Show If** from the **Create Rule** window.

Right-click the component that determines if a component is shown on the form

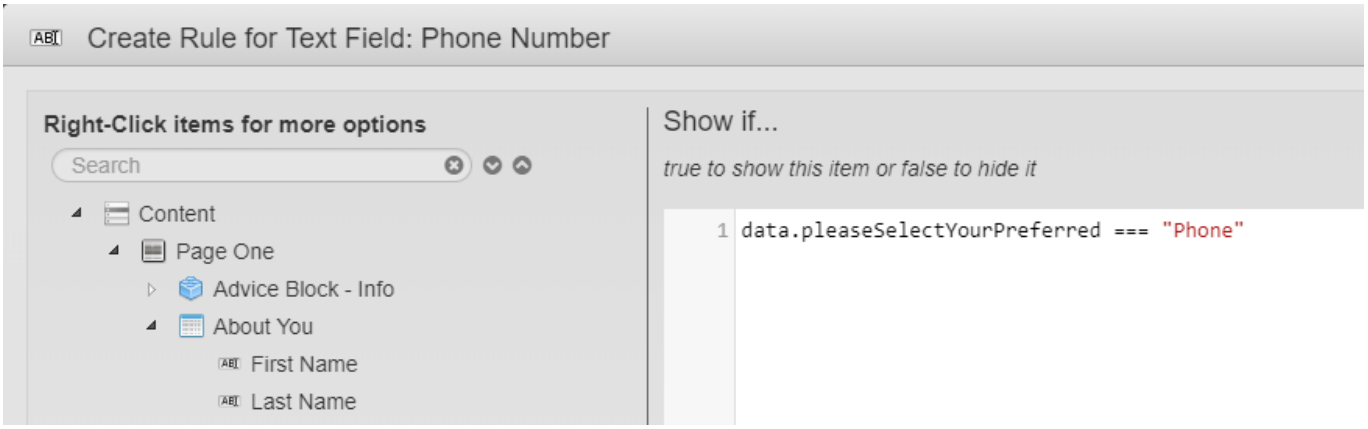
Choose **Selected option**.



Nominate the matching Option (e.g. Phone) and click **OK.**



The script is added to the **Create Rule** window.



Combine entered data into a component

Select the component where the data will be populated.

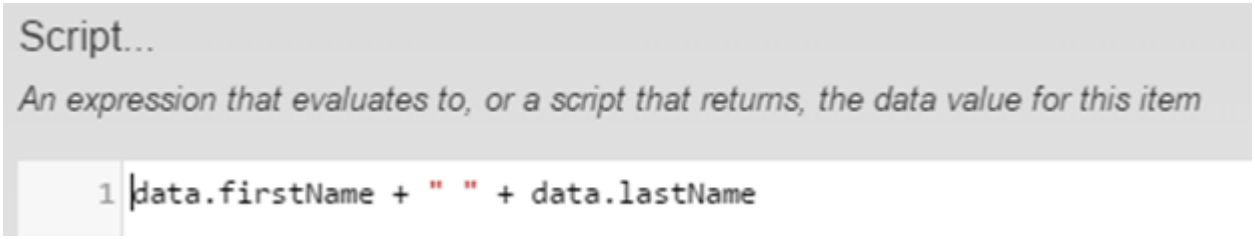
Click **Create Rule** and select **Script**.

Double-click the component you want to add to the script, e.g. double-click **First Name** to add **data.firstName** to the script window.

Enter a plus sign between the components you want to combine.

Any spaces or other characters to include in the script must be wrapped between quotation marks (“).

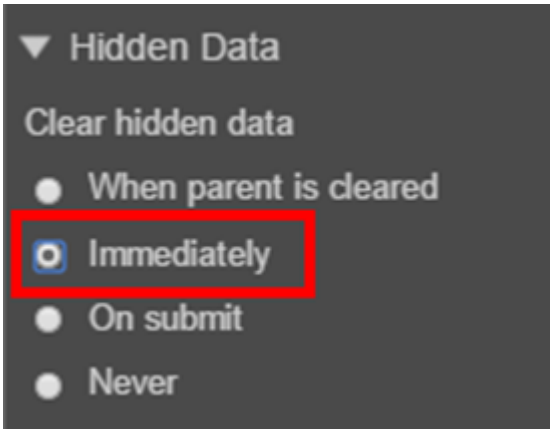
Double-click the next component to add to the script (see example below).



Clear Hidden Data Immediately

Select the component.

Select the **Data** tab, expand **Hidden Data** and choose **Immediately** from the options.



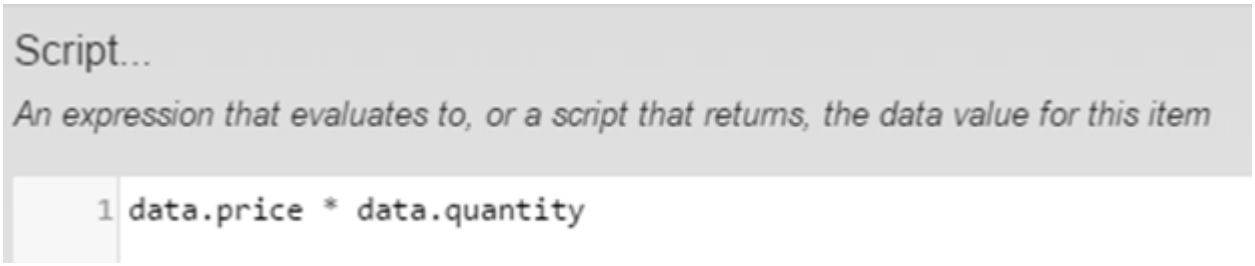
Create a Mathematical Calculation Rule

Select the component that will contain the calculation.

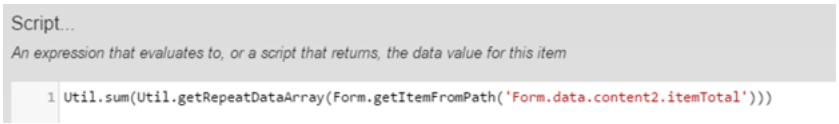
Click **Create Rule** then select **Script**.

Double-click any components on the left-side of the window to add them to the script window (on the right).

Insert the required calculation symbols, e.g. * / + -



The script is added to the window.



Create a Calculation Rule for Repeated Items

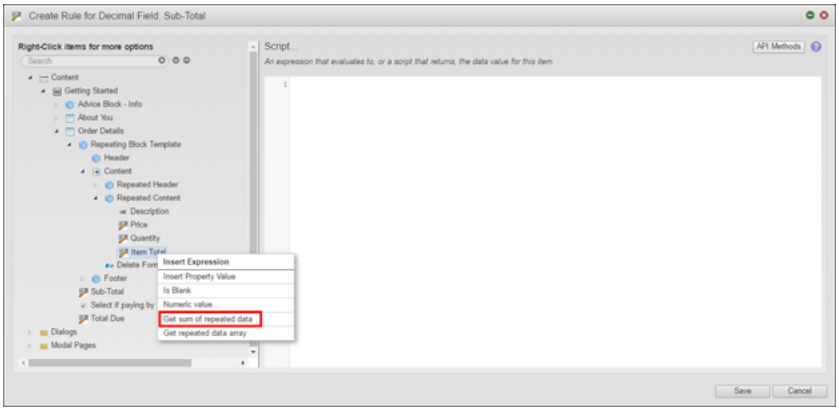
Select the component that will contain the calculation.

Click **Create Rule** then select **Script**.

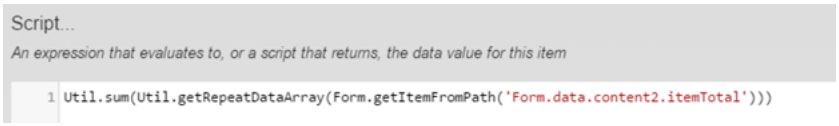
Expand the **Repeating Block Template**, then expand **Content**.

Expand **Repeated Content**.

Right-click the component you want to total and select **Get sum of repeated data**.



The script is added to the window.



Compatibility Mode Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills in upgrading an existing project to Maestro 5.1. During this exercise, you will import a new project into your organization and then change the release version to Maestro 5.1.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1: Import Compatibility Mode Project

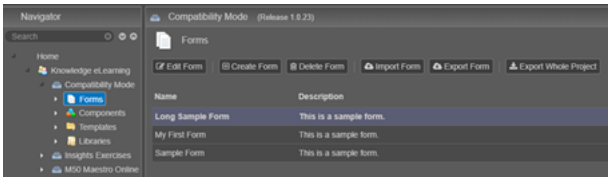
This exercise uses a different project to the one downloaded in the Exercise Setup Guide and will only be used for this exercise.

Identify which of the following best describes the Training Environment you will use to complete this exercise and download the appropriate project version.

Training Environment	Uses Maestro version	Download…	Import Type
elearning.avoka.com	17.10 or later	Version 2018.08.03	Import Project
Your company	17.10 or later	Version 2018.08.03	Import Project
Your company	5.1.x or earlier	Version 2017.06.01	Import Project

Once the project is downloaded, import the project into your Maestro organization.

1. Select your **Organization** then click **Import Project**.
   - Name: **Compatibility Mode**
2. Select your project and click **Open**.
3. The new project is displayed in your selected **Organization**.



Step 2: Update the Project to Maestro 5.1

Update the project to Maestro 5.1 by changing the Release Version dropdown on the Project Details tab.



Extra Help

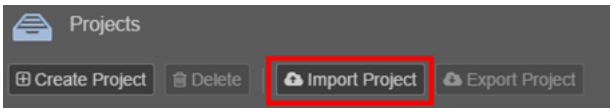
Need some extra help?

Use the information below to assist you in completing the exercise steps.

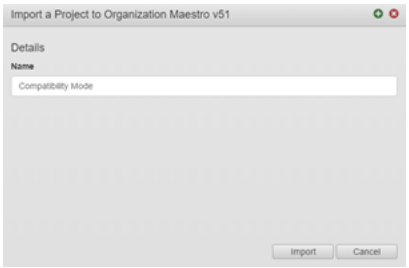
Import project

Select the Organization from the Navigator panel.

Click the Import Project button.

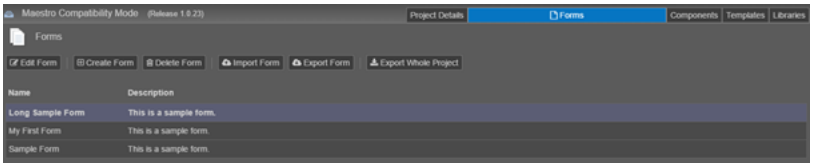


Enter a Name for the new project, then click Import.



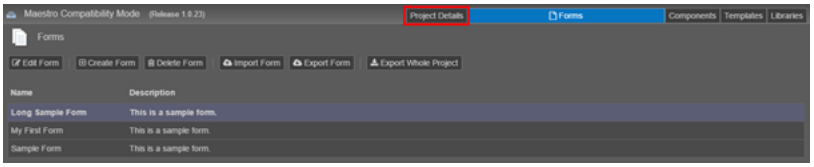
Select the project you want to import and click Open.

The new project will be created and displayed under your Organization.



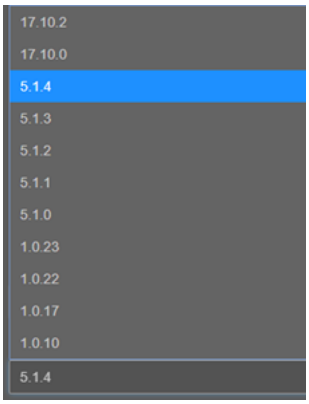
Change Release Version

Select the project from the Navigator panel, then select the Project Details tab.

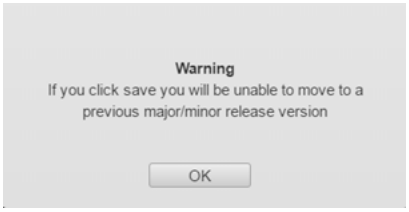


Use the Release Version dropdown to select 5.1.4.

This will associate the 5.1 version of Maestro to access the forms in your compatibility project.



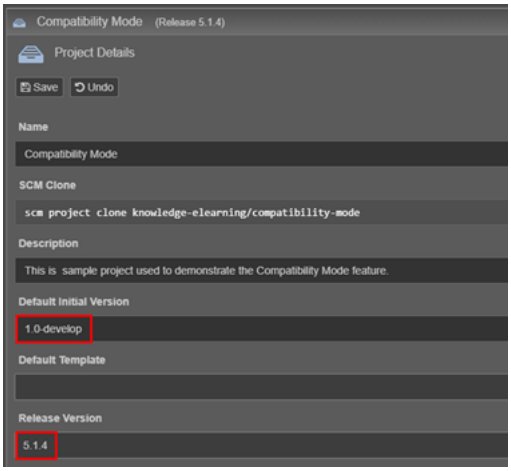
When you select a later version from the dropdown a warning message confirms that once you upgrade you can't change back to a previous release version.



Click **OK**.

Click Save to apply your changes.

> You may also need to enter a Default Initial Version before saving, i.e. 1.0-develop


Compatibility Mode (Release 5.1.4)
Project Details
Save
Undo
Name
Compatibility Mode
SCM Clone
scm project clone knowledge-elearning/compatibility-mode
Description
This is sample project used to demonstrate the Compatibility Mode feature.
Default Initial Version
1.0-develop
Default Template
Release Version
5.1.4

Create a New Form Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating new forms in Maestro. During this exercise, you will create a new form and add components.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises > Create New Form Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1: Create a New Form

Create a new form using the details provided below.

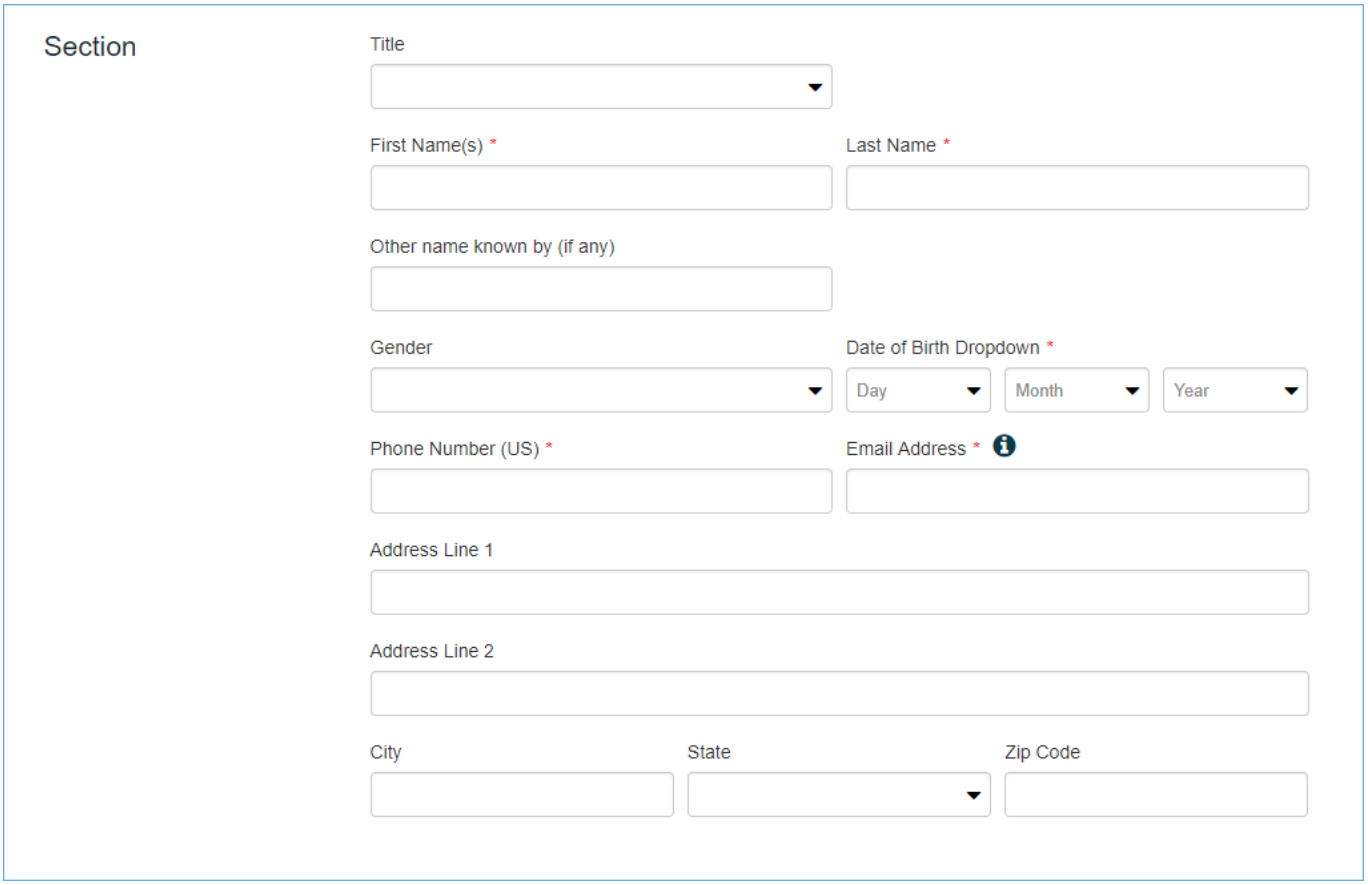
1. In the Navigator Panel, expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Click **Create Form**.
5. Enter the following details for your new form:
   - Name: Create New Form_[Your initials]
   - New Version: 1.0-develop

     (should be pre-populated if Default Initial Version has been setup in Project Details)

   - Description: This form is to practice creating new forms and adding components.
   - Template: Exercise Template
6. Click **Create**.

Step 2: Add Components to Page One

On Page One, you will find an empty **Section**. You can add (drag) components into this area. Use the image below as a guide for what you need to add.



You can **Preview** your form at any time to see how the components look and work.

the components marked as mandatory (*) and with the information icon (i).

When adding components, it may be helpful to display the Palette and Structure panel together. Click the Side-by-Side button in the top left corner of the Editor window.

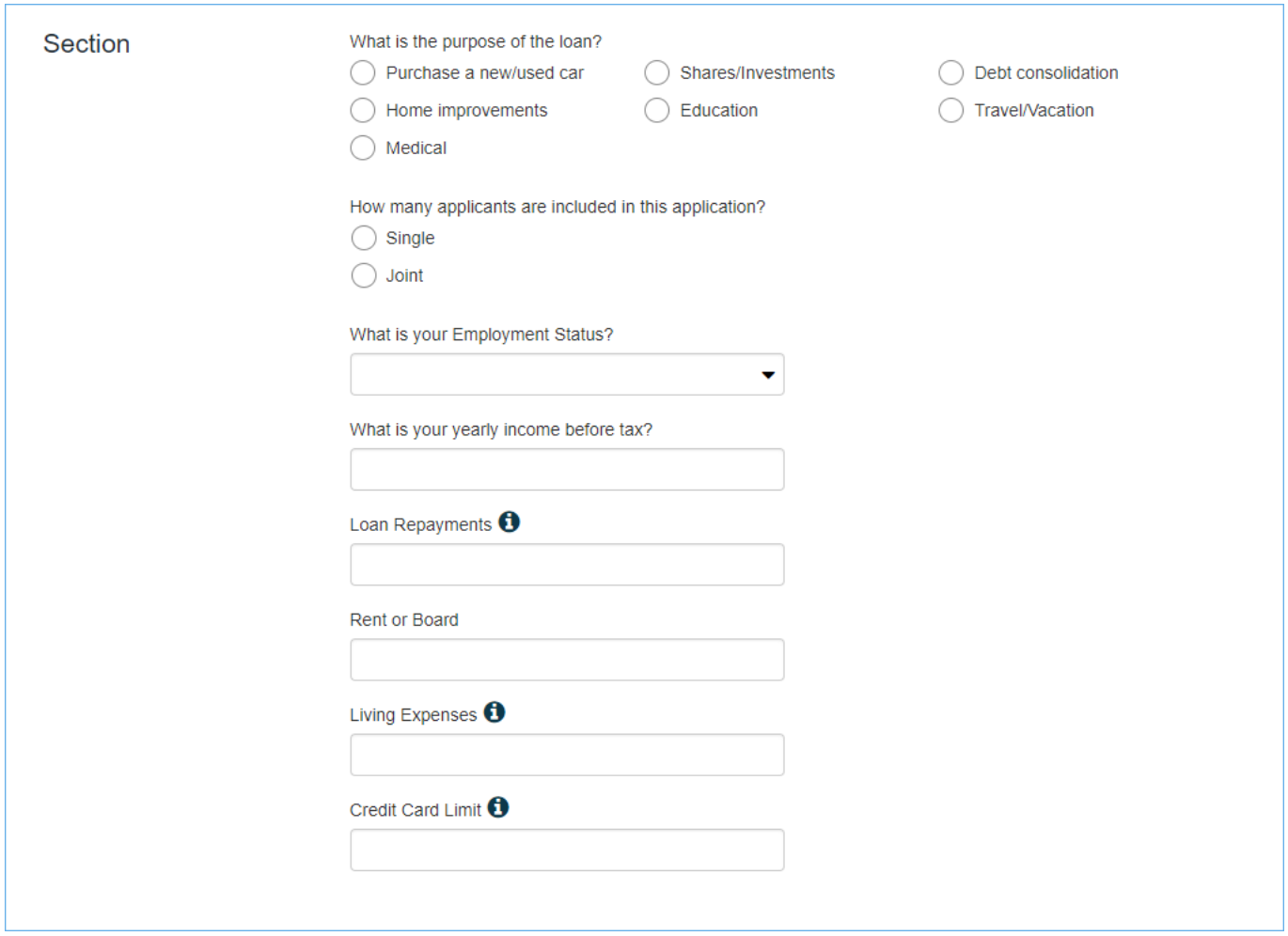
Remember: to use predefined components, rather than recreating your own. You can use Search in the Palette panel to find the required components, for example, the 'Title' component, which contains all the common titles (Mr, Mrs. etc.). The default options can be updated as required and is much easier than building a dropdown from scratch.

Use this for the **Email Address Help Tex**t: We use your email to send confirmation of the information you've provided.

Click **Save** on the toolbar.

Step 3: Add Components to Page Two

On Page Two, you can add (drag) components into the available section. Use the image below as a guide for what you need to add.



Some components have the information icon (i) next to the label.

Use the following entries for the **What is the purpose of the loan** radio buttons:

- Purchase a new/used car
- Shares/Investments
- Debt consolidation
- Home improvements
- Education
- Travel/Vacation
- Medical

Use the following entries for the **How many applicants are included in this application?** radio buttons:

- Single
- Joint

Use the following entries for the **What is your Employment Status?** dropdown:

- Employed
- Self-Employed
- Unemployed

Use the following details for the **Help Text**:

- Loan Repayments: Please include any mortgage repayments, personal loans or car loans.
- Living Expenses: Consider how much you spend on bills, groceries, transport and entertainment, etc.
- Credit Card Limit: Please include the total limit of all your credit cards.

Use the following for the expense questions:

- What is your Employment Status?
- What is your yearly income before tax?
- Loan Repayments
- Rent or Board
- Living Expenses
- Credit Card Limit

Step 4: Save the Form

Click **Save** on the toolbar to save all the changes to your form.

Extra Help

Need some extra help?

Use the information below to assist you in completing the exercise steps.

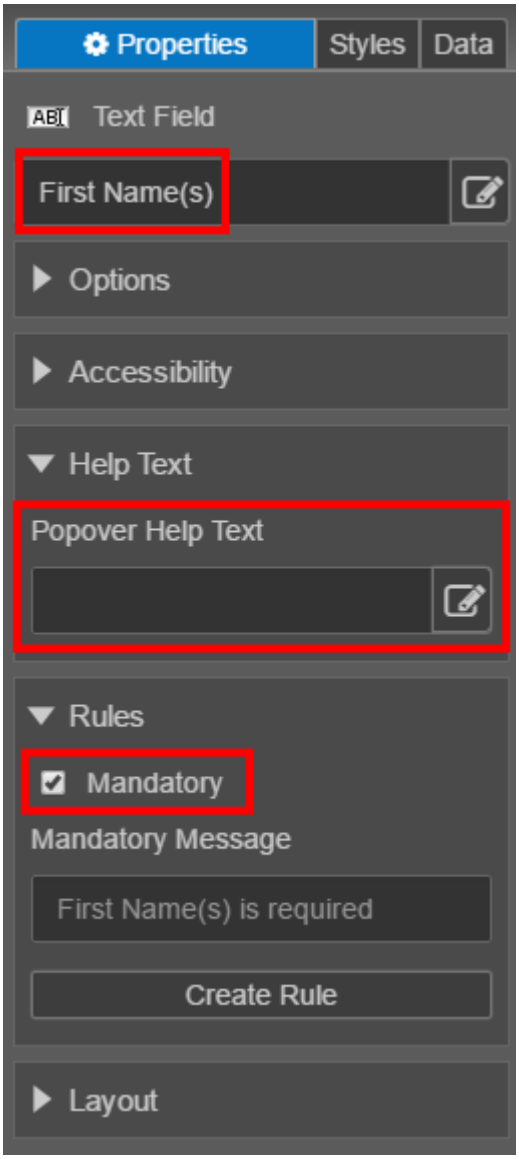
Component Properties

You can use the **Properties** panel to change the following settings:

- Component Label
- Add Help Text
- Set Component as Mandatory
- Radio Button Arrangement
- Radio Button Options
- Dropdown Options

The **Properties** panel displays the settings for the selected component.

For example, properties for the First Name(s) Text Field are displayed below.



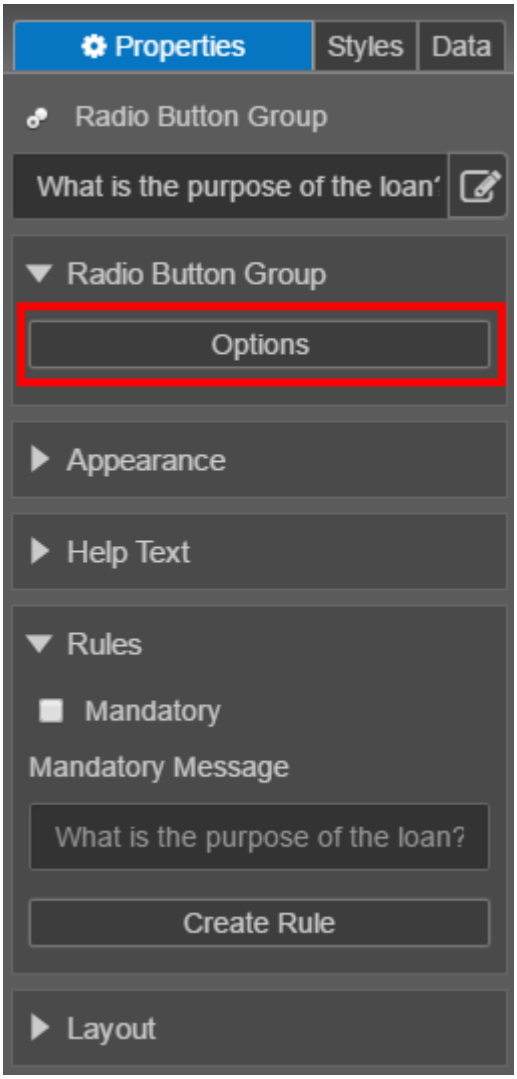
At the top, you can change the component label.

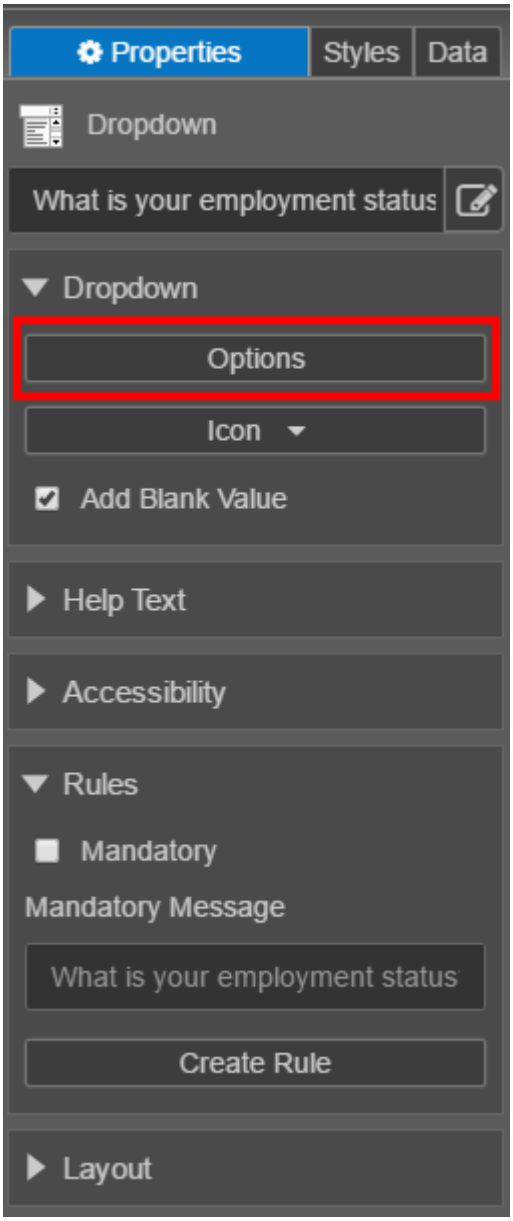
Popover Help Text is where you add the help text, which displays the information icon next to the component label.

The **Mandatory** option is under **Rules**. Select this to mark a component as mandatory. This will automatically add the red asterisk next to the label.

Radio Button and Dropdown Options

Use the **Properties** panel to customize the options available for **Radio Button Group** and **Dropdown**.



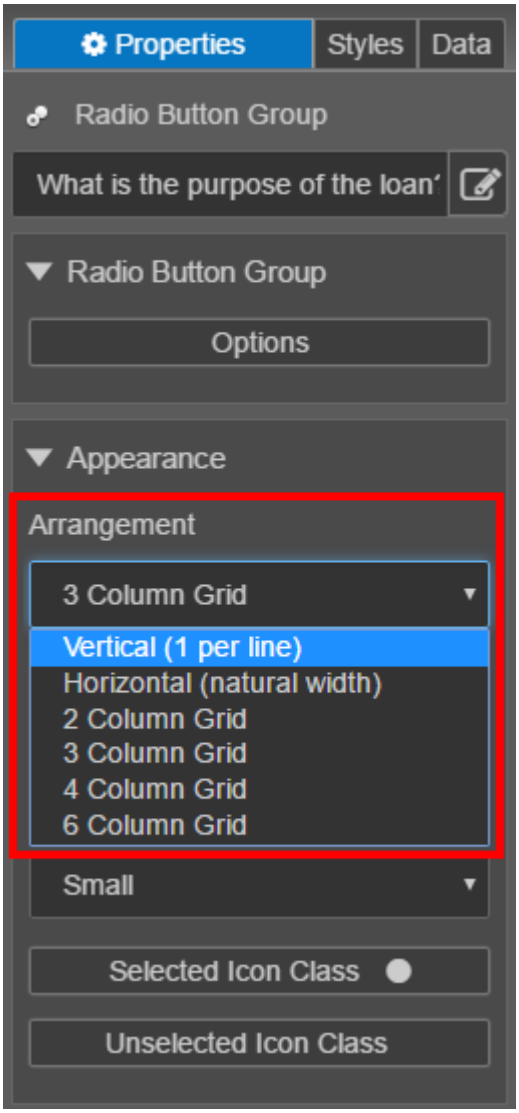


Click the **Options** button to open a new window, where you can add or edit the radio buttons or dropdown options.

Arrangement of Radio Button Options

You can change the appearance of the radio buttons, so they're stacked vertically or displayed in a horizontal grid.

To change the appearance, select the required **Arrangement**.


Properties
Styles
Data
Radio Button Group
What is the purpose of the loan'
Radio Button Group
Options
Appearance
Arrangement
3 Column Grid
Vertical (1 per line)
Horizontal (natural width)
2 Column Grid
3 Column Grid
4 Column Grid
6 Column Grid
Small
Selected Icon Class
Unselected Icon Class

Creating Rules Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating rules. During this exercise you will create a new form and add components. You will then create rules on the added components.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises > Creating Rules Finished. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

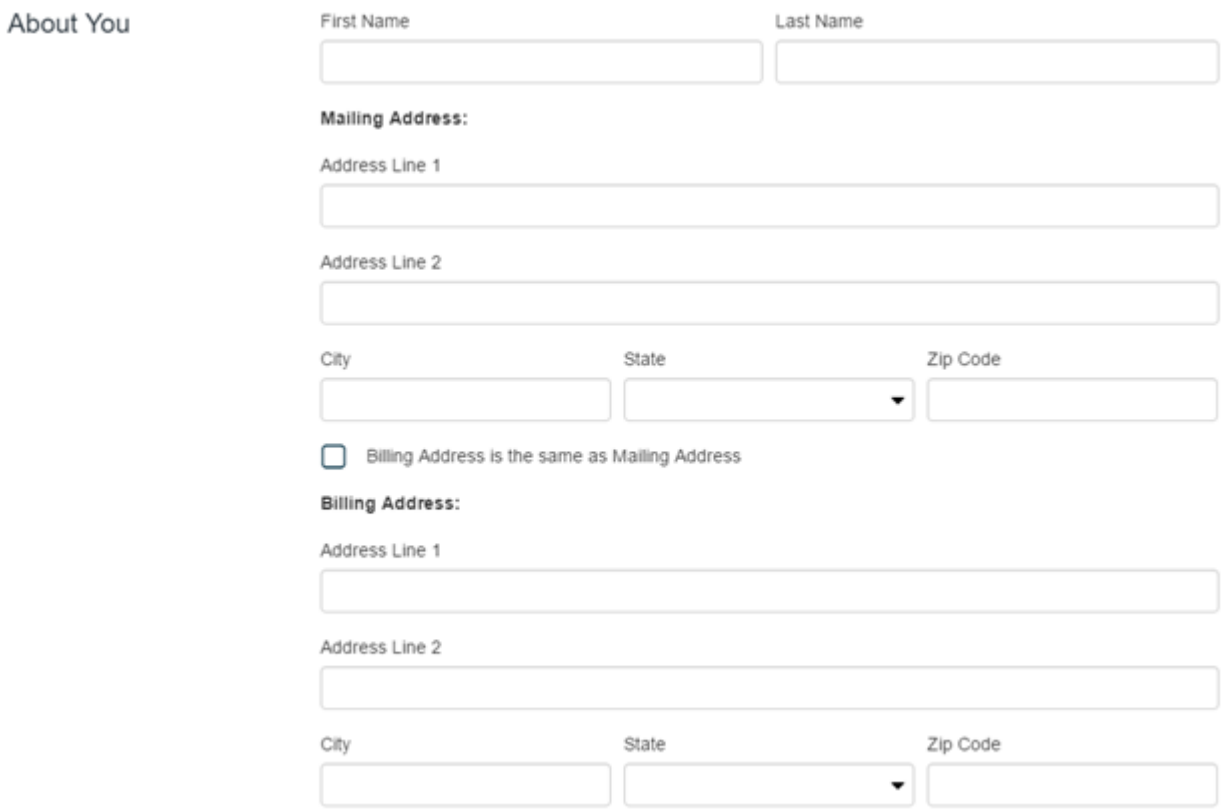
Step 1 Create a New Form

Create a new form using the details provided below.

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Click **Create Form**.
5. Enter details about the form:
   - Name: Mailing Address
   - New Version: 1.0-develop
     (should be pre-populated if Default Initial Version has been setup in Project Details)
   - Description: This form is to practice creating component rules.
   - Template: Exercise Template
6. Click **Create**.

Step 2 Add Components to Page One

On Page One you will find an empty section. Add new components to this area using the image below as a guide.



When adding components, it may be helpful to display the Palette and Structure panel together. Click the Side-by-Side button in the top left corner of the Editor window.

Remember to use predefined components (where appropriate), rather than recreating your own. You can use the search in the **Palette** panel to find components. For example, use the Address (USA) component rather than adding individual components. This component contains all the necessary fields, which you can customize if needed.

Step 3 Add a Rule to Billing Address

Add an **Editable if** rule to the Billing Address block that allows it to be editable if the checkbox is not selected.

You can double-click components on the left side of the Create Rule dialog to add them to the script window or type directly into the script window.

Editable if...

true if this item should be editable or false if this item should be locked

```
1 data.billingAddressIsTheSame !== true
```

Step 4 Add a Rule to the Checkbox

Add a **Change** rule to the checkbox that copies the entered values from Mailing Address to Billing Address if the checkbox is selected.

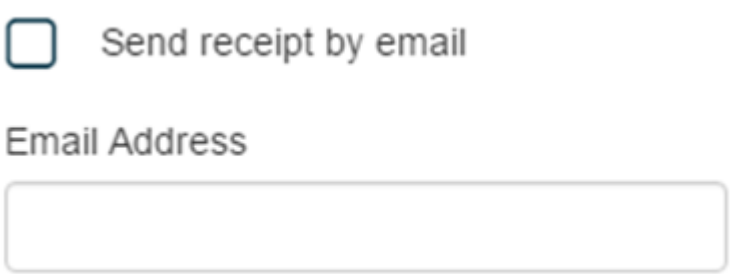
Change

A script to run when the user changes the data in this form item

```
1 if (data.billingAddressIsTheSame === true ) {
2   data.addressLine11 = data.addressLine1;
3   data.addressLine21 = data.addressLine2;
4   data.city1 = data.city;
5   data.state1 = data.state;
6   data.zipCode1 = data.zipCode;
7 }
```

Step 5 Add Components with a Mandatory If Rule

Add a checkbox and text field to the end of the form. Create a mandatory if rule on the text field that makes the field mandatory if the checkbox is selected.



Extra Help

Need some extra help?

Use the information below to assist you in completing the exercise steps.

Create an Editable If Rule

Select the required component from the **Structure** panel (or in the form).

Click the **Create Rule** button from the **Rules** section (**Properties** panel).

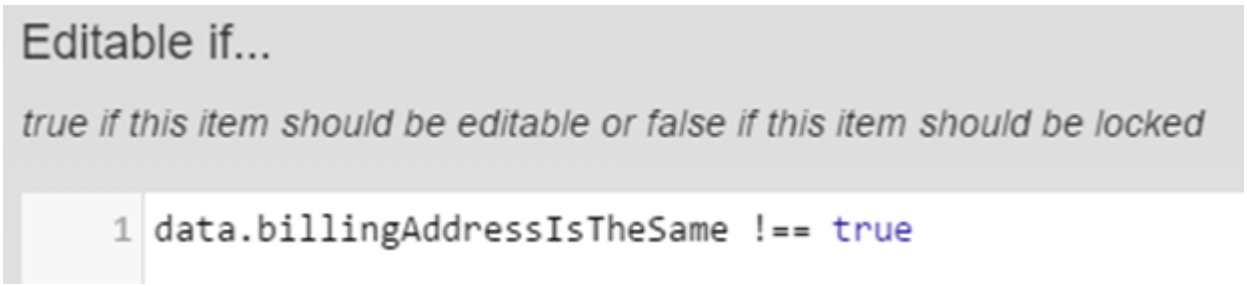
Select **Editable If**.

Right-click the checkbox component that determines when the address component is editable.



The options in the menu are determined by the component selected. The image (above) shows the options for a checkbox.

Based on the exercise requirements, you would reed to select the **Is not selected** option.



The script for this rule says the address block is editable if the checkbox is **not** selected.

Create a Change Rule

A Change rule will trigger when a component is changed.

To start, select the component you want to apply the rule to (i.e. the checkbox).

Click the **Create Rule** button from the **Rules** section (**Properties** panel).

Select **Change** under **Action**.

> You can double-click components on the left side of the Create Rule dialog to add them to the script window (right) or type directly into the script window.

Based on the exercise requirements, you will need to type at least some of the script. Enter the script as follows:

Type `if (`

Then right-click the checkbox and select **Is selected** to add `data.billingAddressIsTheSame === true`.

Then type `) {` and press **Enter**.

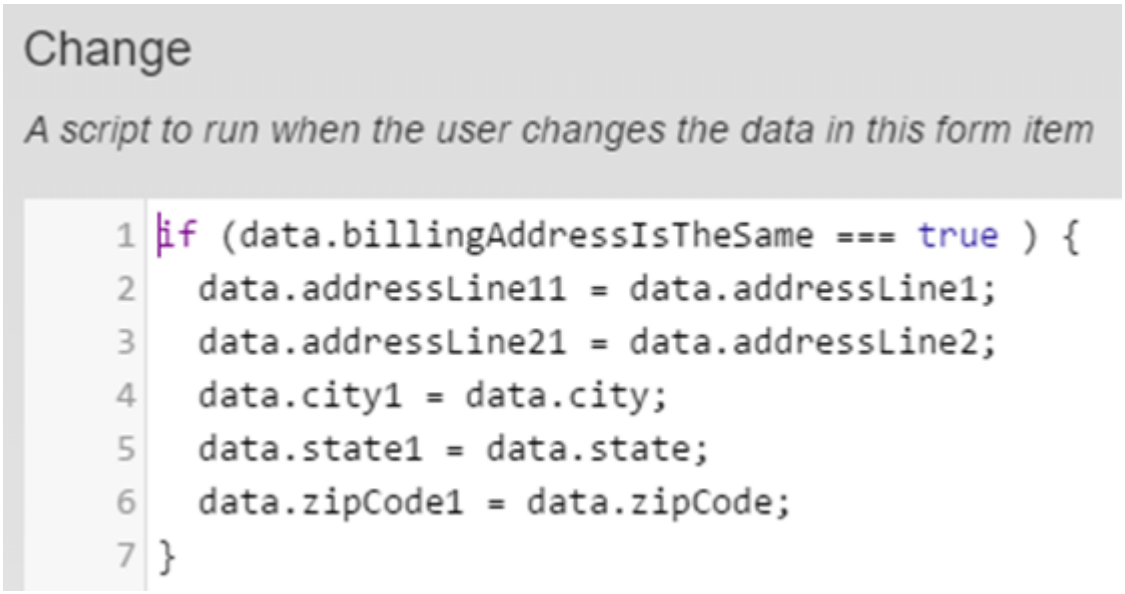
For the next line, double-click **Address Line 1** from the **Billing Address** block to add `data.addressLine11`.

Type **=** and a `space`.

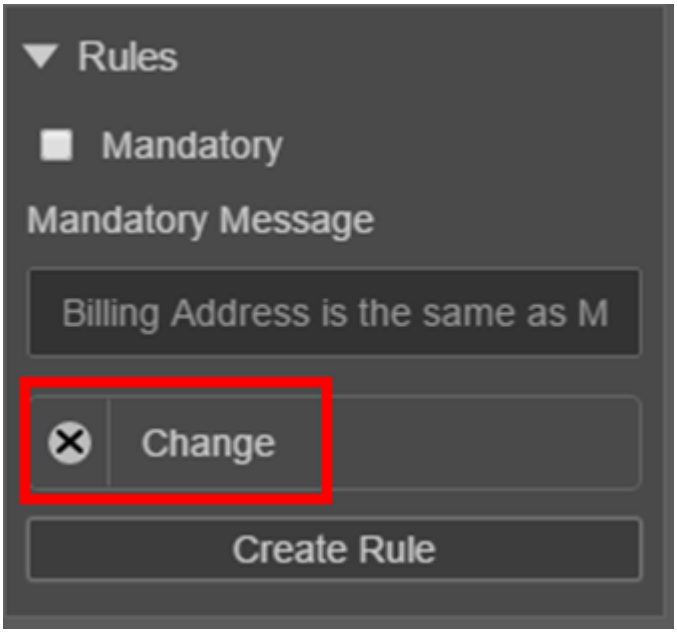
Double-click **Address Line 1** from the **Mailing Address** block to add `data.addressLine1`.

Type `;` (Semicolon)

Repeat this process for each of the address components. On the final line type `}` to end the script. See the image below for guidance.



Once you've finished creating the rule is, click **Save**.



The rule is added to the **Rules** section (**Properties** panel).

Create a Mandatory If Rule

Select the component to be mandatory based on specific conditions (i.e. Email Address).

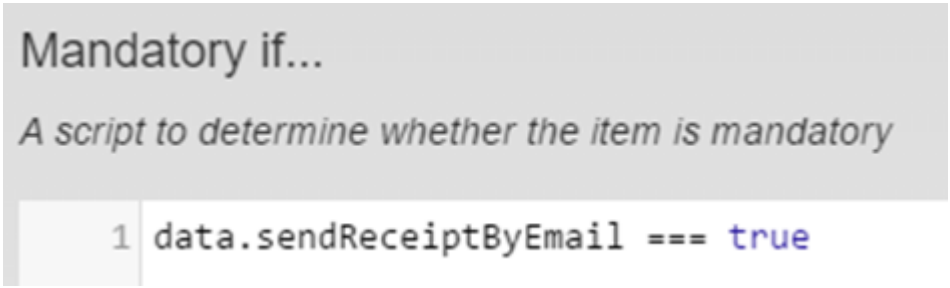
Click the **Create Rule** button from the **Rules** section (**Properties** panel).

Select **Mandatory If**.

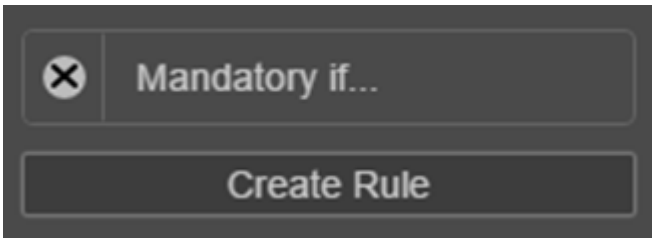


You can double-click components on the left side of the Create Rule dialog to add them to the script window (right) or type directly into the script window.

See the image below for guidance.



Click **Save**.



Creating Shared Components Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating shared components. During this exercise you will create a new shared component within a form.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find **Extra Help** at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the **Maestro Exercise Project: Maestro Training Exercises** > **Create Component Finish**. Please note that this form uses a shared component with a different name to the one suggested in the steps below. Shared components cannot have the same name as other shared components within the same project.

If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

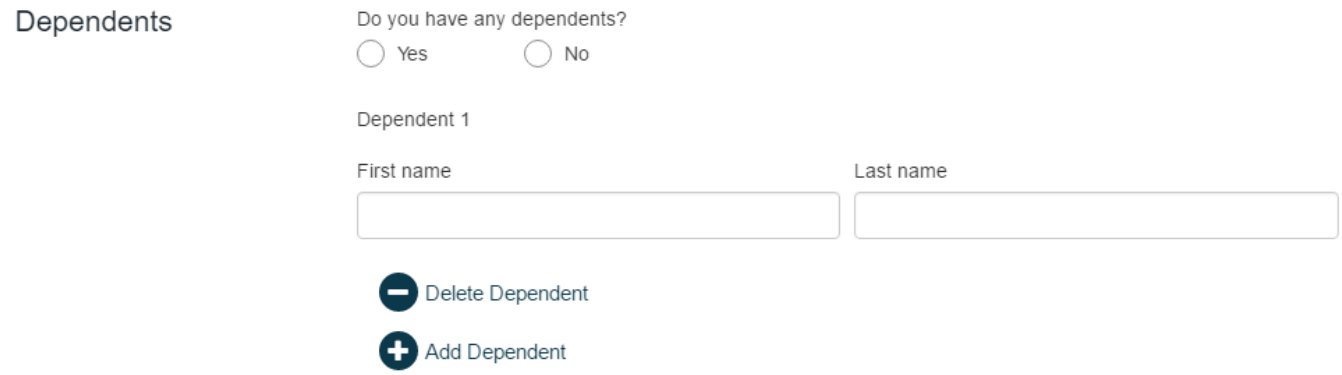
Step 1 Open Create Component Begin

1. Expand **Organization**.
2. Expand **Project**.
3. Select **Forms**.
4. Double-click **Create Component Begin**.
5. Click the **Edit Version** button to open the form

You can create components directly from the Components folder, rather than from within a form.

Step 2 Setup a Repeating Block and Add Components

Add a **Repeating Block Template** and components to the **Dependents** section so it matches the image below.

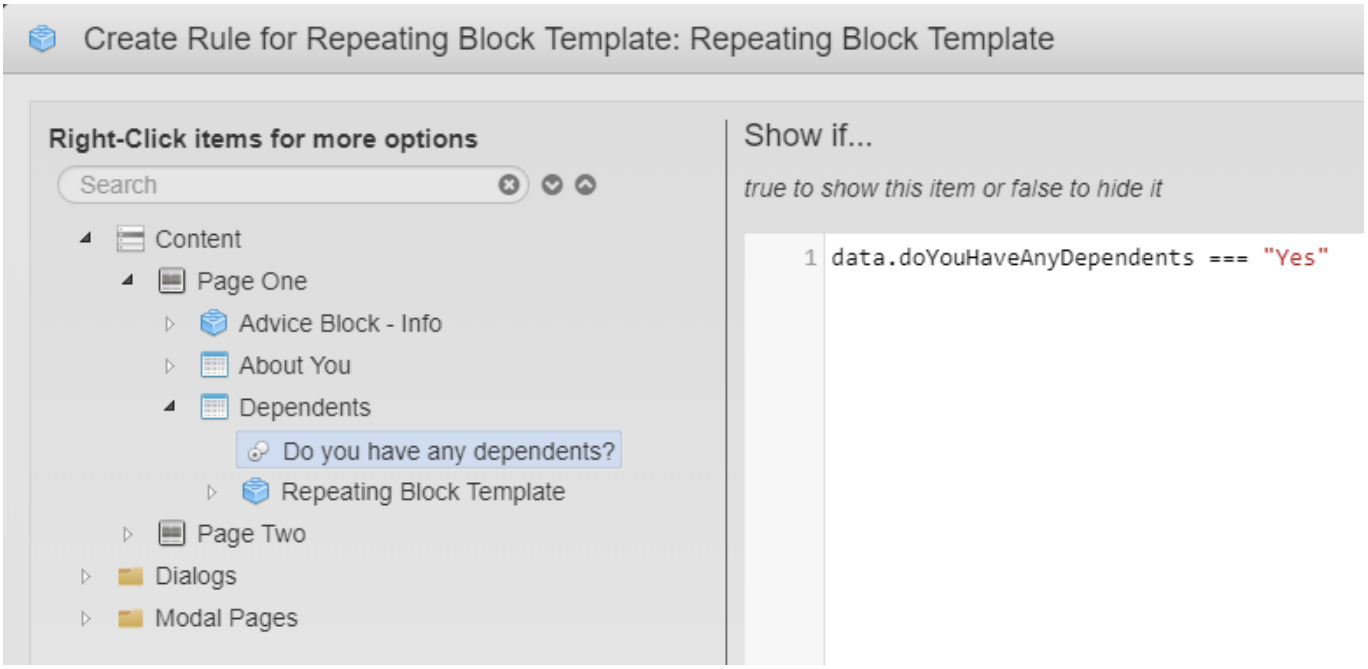


The First Name and Last Name components are within the Dependent 1 Repeating Block Template. Before you add components, make sure you add the Repeating Block Template first, then add the components.

Step 3 Create Visibility Rule

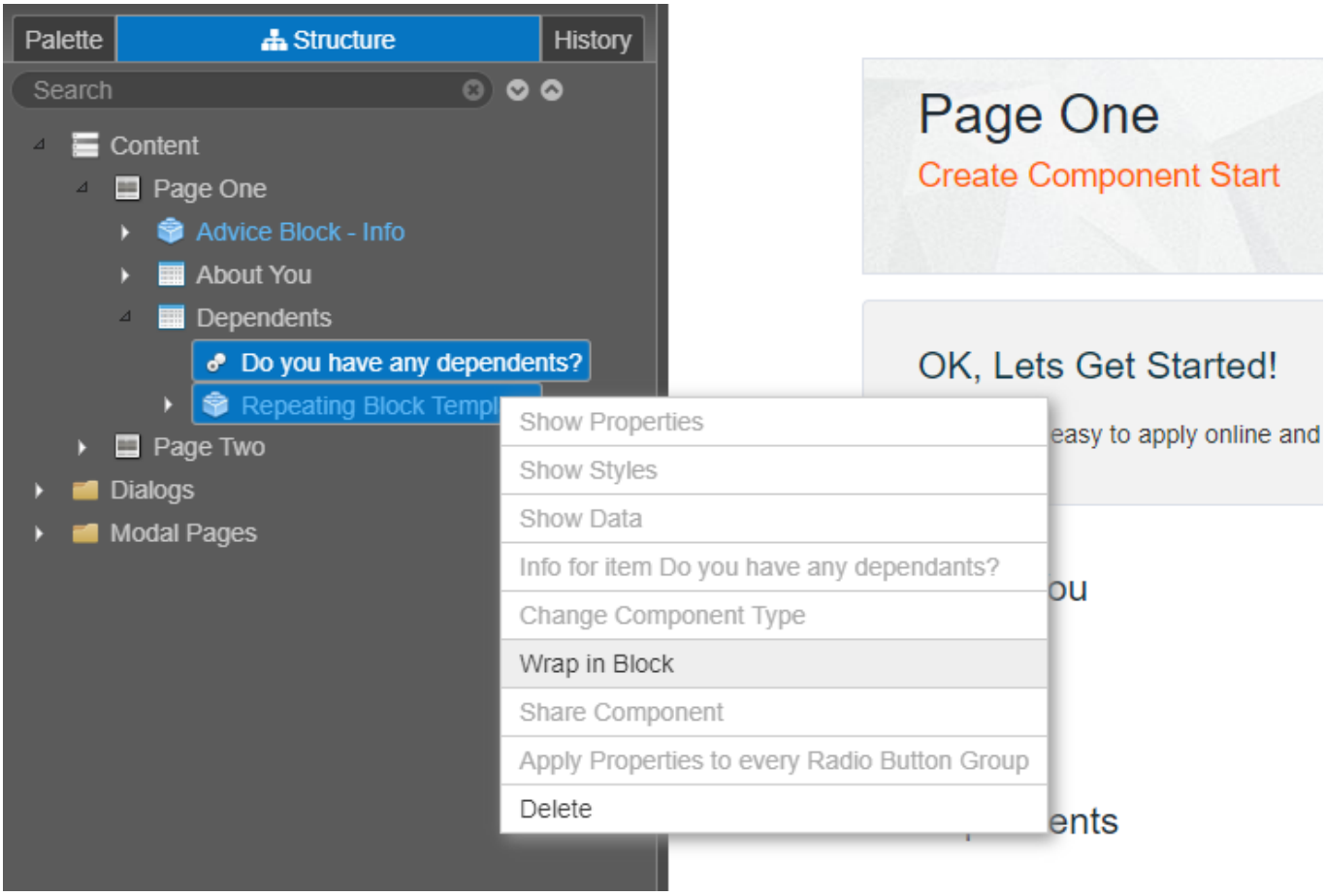
Create a visibility rule so that the repeating block does not show unless the user selects Yes to having dependents.

Create this rule on the **Repeating Block Template** (see below).

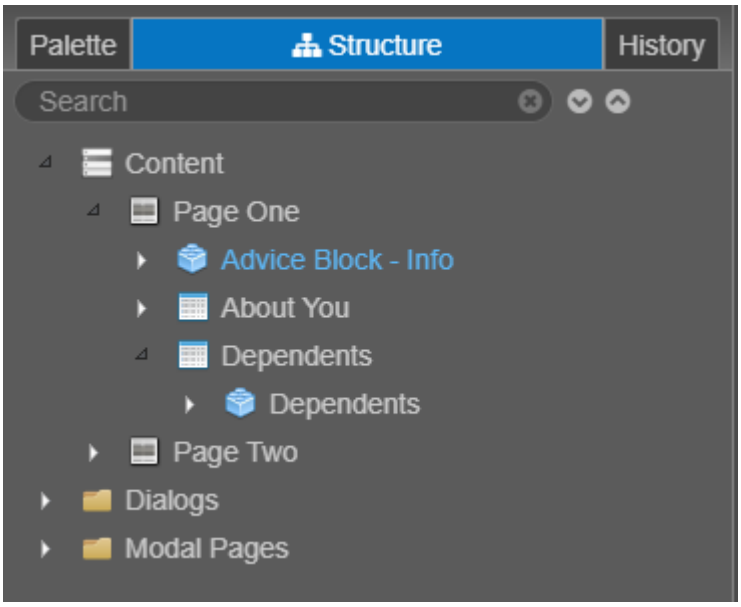


Step 4 Wrap in block

Wrap all the components (i.e. Radio Button component and Repeating Block) from the Dependents section into a block. See the image below for guidance.

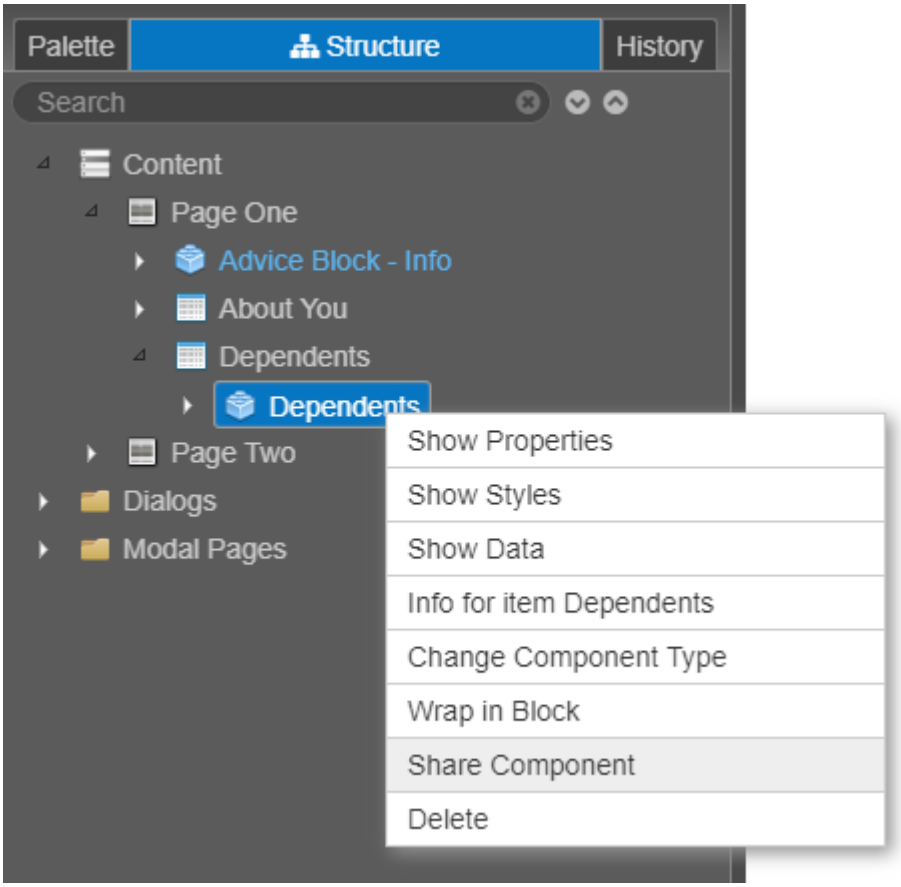


If more than one component is going to be part of a shared component, you must wrap the relevant components into a block. Your Structure panel should look like the image below.



Step 5 Create a Shared Component

Create a shared component that combines the components you just created. Your Structure panel should match the image below.



Use the following details for the shared component:

Name: Dependents_Exercise

Description: Contains the components for applicant's dependents.

Palette Folder: Custom Components

Instant Publish: Select Publish for immediate use and select Replace item with shared component

Publish Library: Select the appropriate project library (e.g. Maestro Training Exercises Library)

Remember: the instant publish options will publish the shared component to the library and replace the existing 'Block' with the shared component. This will ensure the current form is updated if the shared component changes.

To finish, click **Save** to save the **Create Component Begin** form - then close the form.

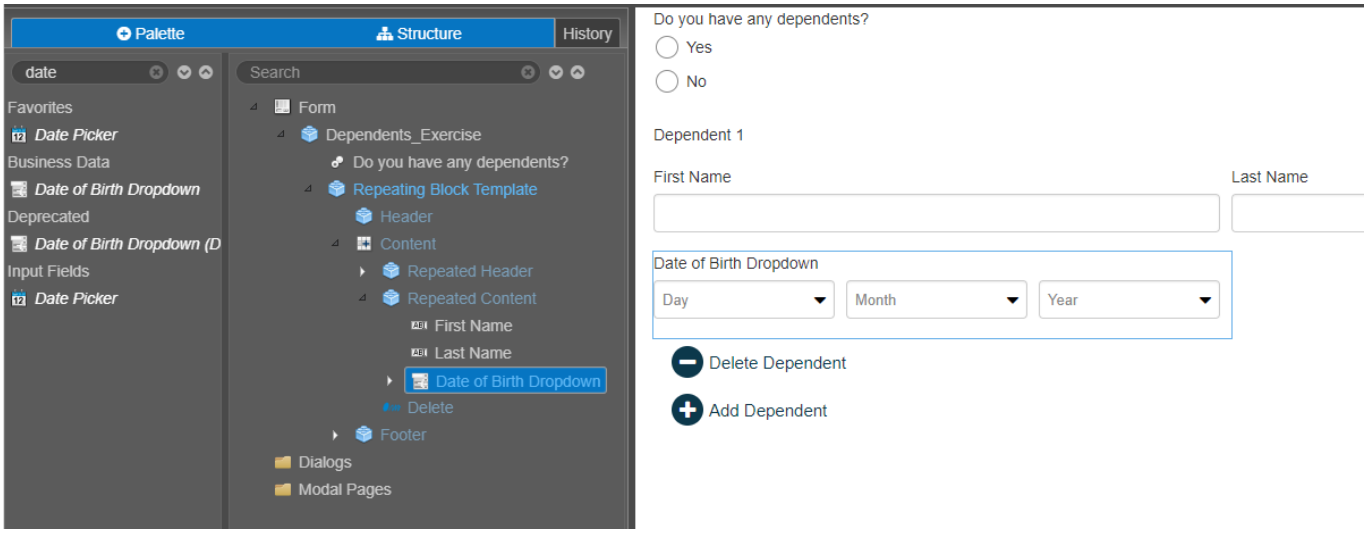
Step 6 Edit a Shared Component

Important: make sure your refresh the browser window (Press F5) that displays the Management Dashboard before you proceed.

From the Maestro Management Dashboard, open the shared component you just created and make a change to the component.

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Components**.
4. Double-click **Dependents_Exercise**.
5. Click **Edit Version** to open the Component.

Add the **Date of Birth** dropdown to the repeating block. Your shared component should match the image below.



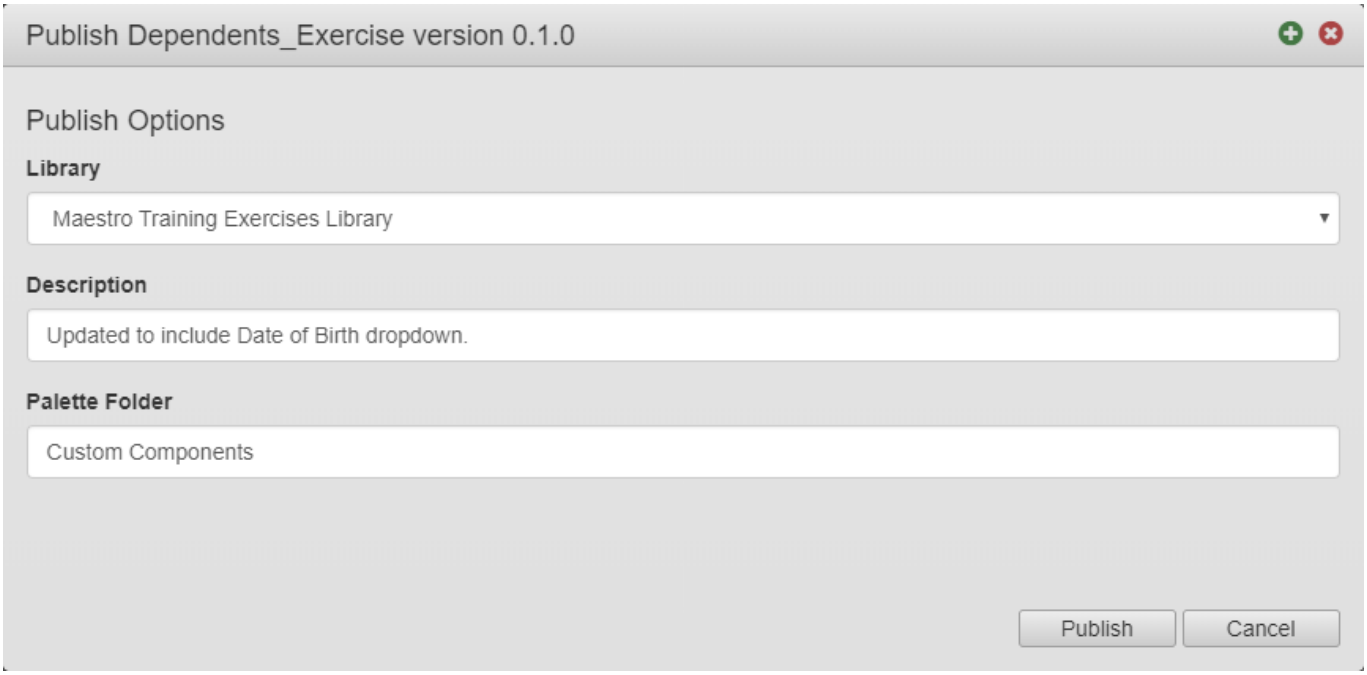
Note that the shared component won't include form elements like the navigator or sections because it's not linked to a specific template.

Click **Save** to save the changes to the shared component.

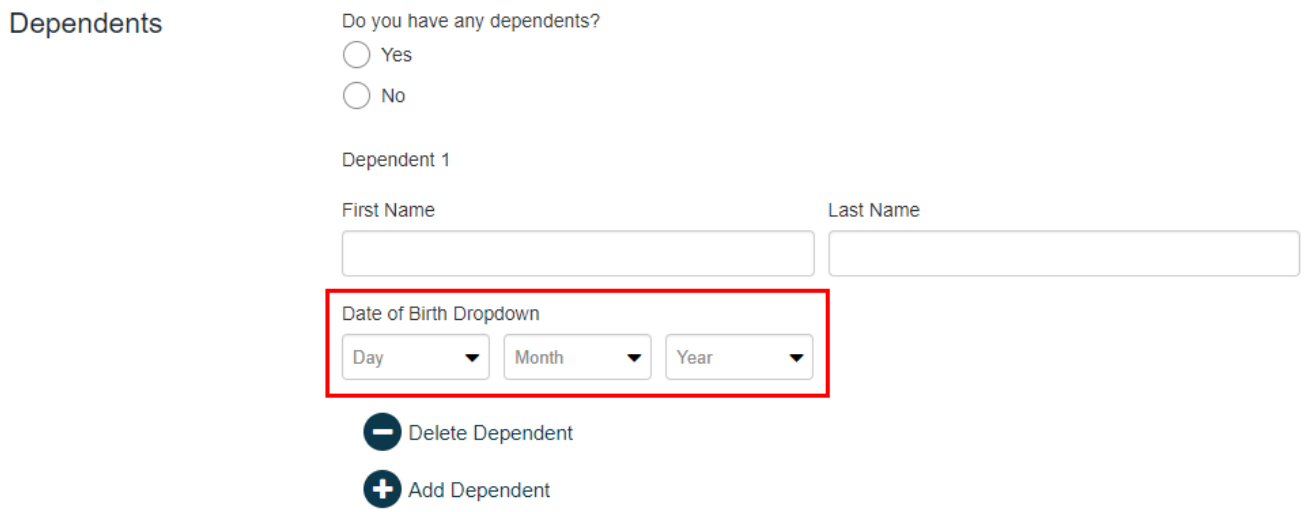
Step 7 Publish Updated Component

Once the component has been updated, **Publish** it so the changes are updated in forms currently using the component as well as any new uses of the shared component.

Use the image below for guidance.



Once you publish the updated component, **Close** the Component window and **Open** the **Create Component Begin** form to confirm the changes to the component (below).



Extra Help

Need some extra help?

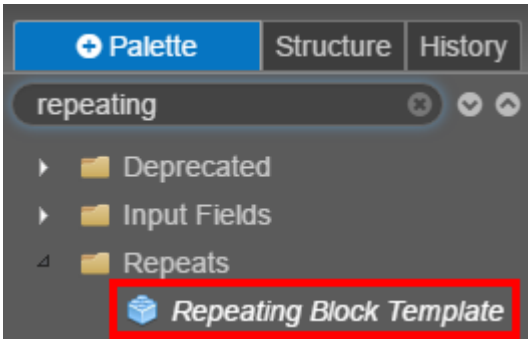
Use the information below to assist you in completing the exercise steps.

Using a Repeating Block Template

The **Repeating Block Template** enables you to easily setup repeating functionality.

Use the Palette Search to find the **Repeating Block Template**.

Drag the component onto your form.

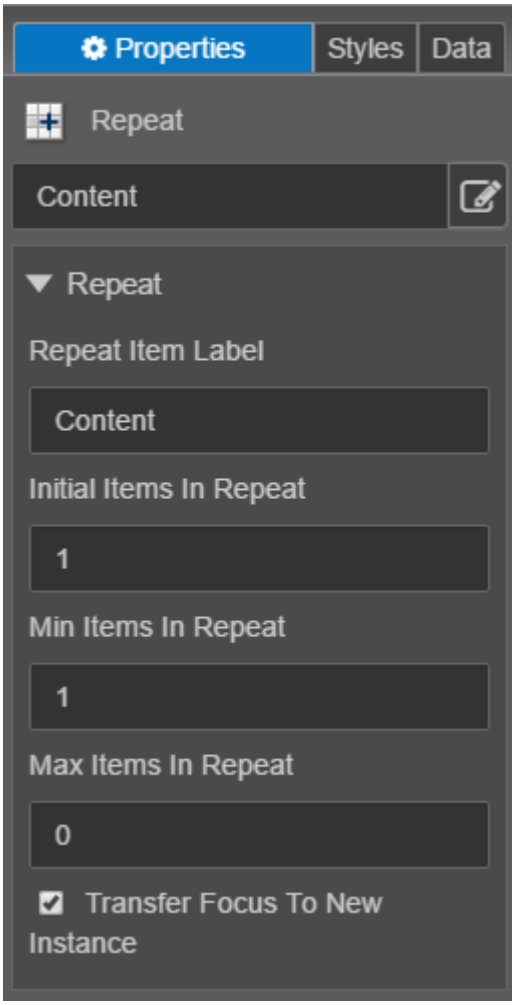


Add the components you want to repeat into the field area of the block.



To make adjustments to the Repeating Block properties, select **Content** within the Repeating Block Template (Structure panel).





Repeat Item Label: updates the label at the top of the block, as well as the button text.

Initial Items In Repeat: determines how many repeating components are on the form by default.

Min Items In Repeat: determines the minimum number of repeating components that must be displayed on the form.

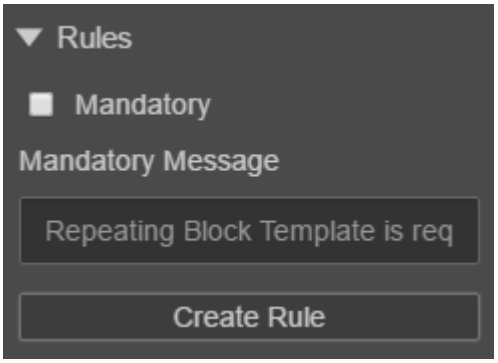
Max Items In Repeat: determines the maximum number of repeating components the user can add to the form.

Create a Show if Rule

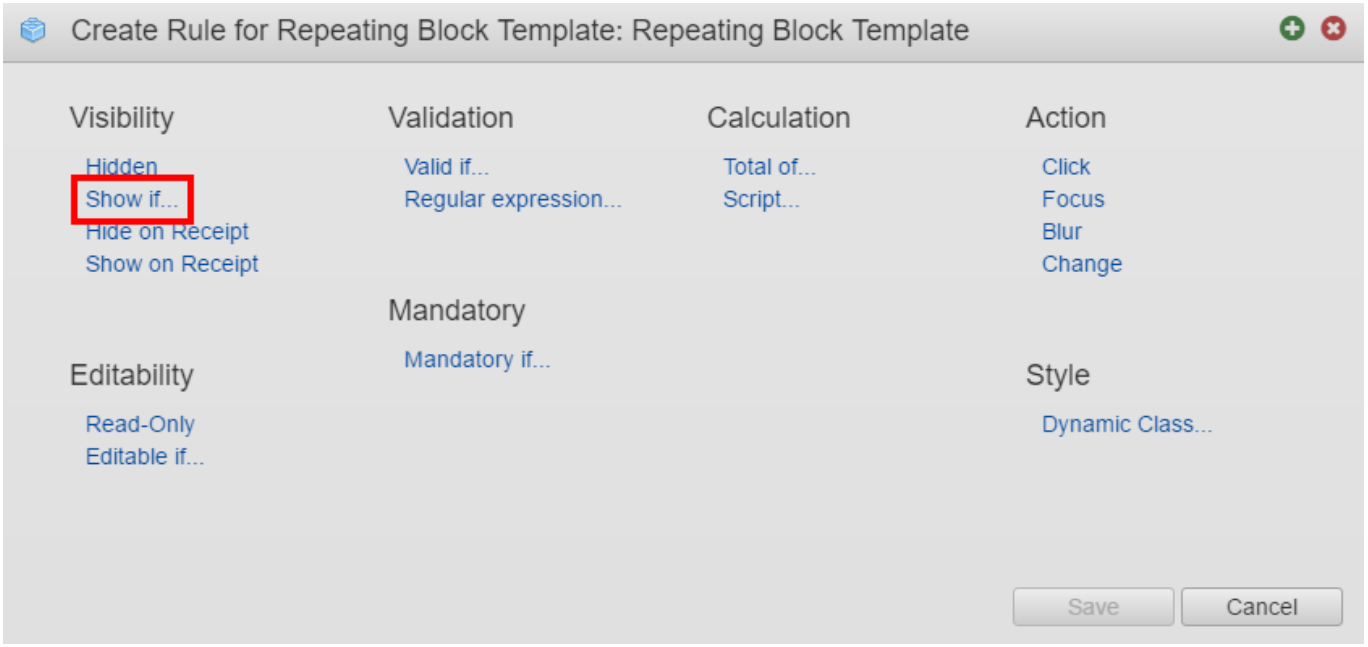
The Show if rule allows you to show or hide a component based on the selection of another.

Start by selecting the component you want to show or hide.

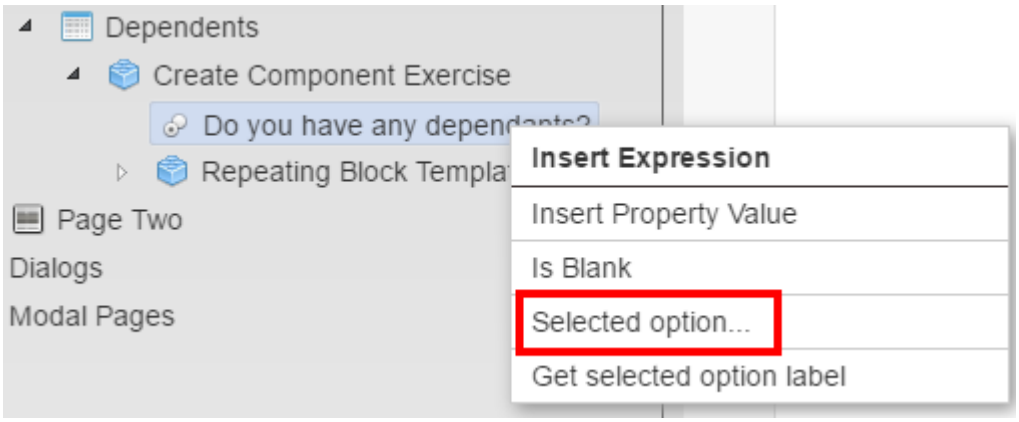
Click **Create Rule** in the **Rules** section of the **Properties** panel.



Select **Show if** (under **Visibility**).

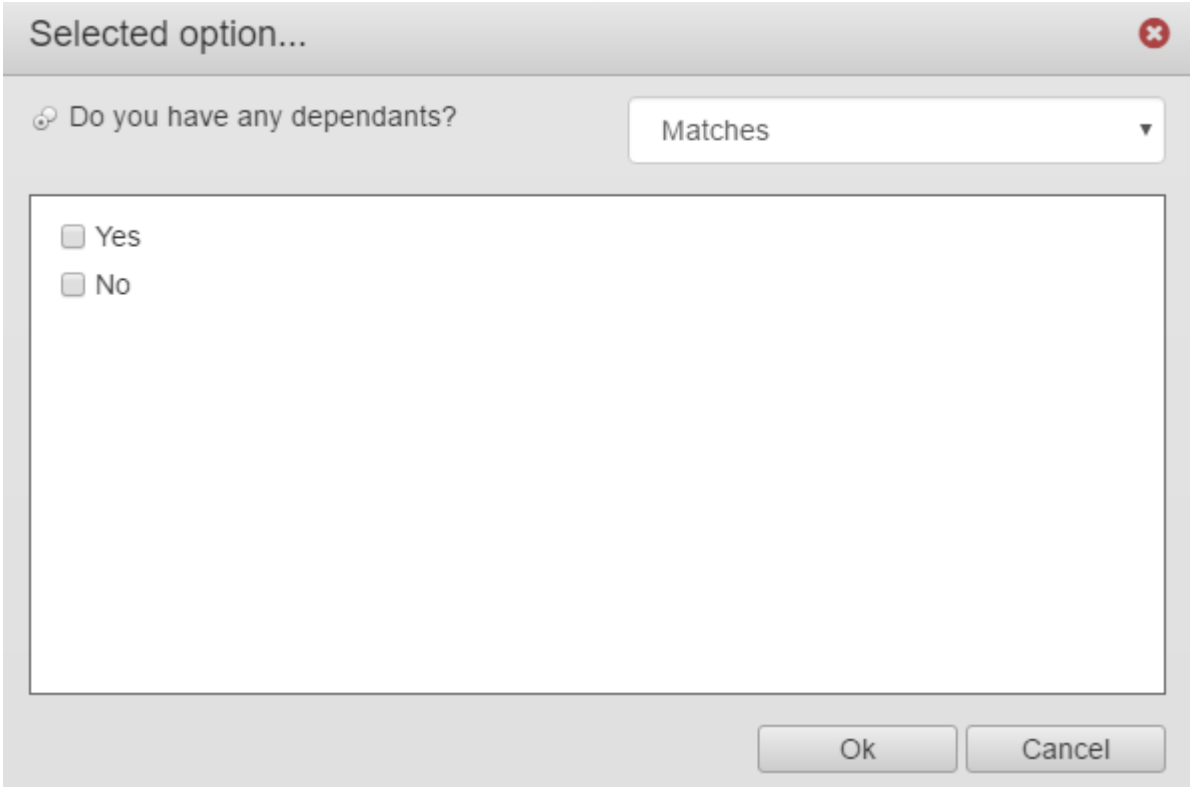


Right-click the component that determines if the selected component is displayed or hidden and click **Selected option**.



In the top right corner of the **Selected option** window, choose if the option should match or not match, then select the relevant options.

Click **OK** to close the window.



The script for the rule and selected components is displayed.

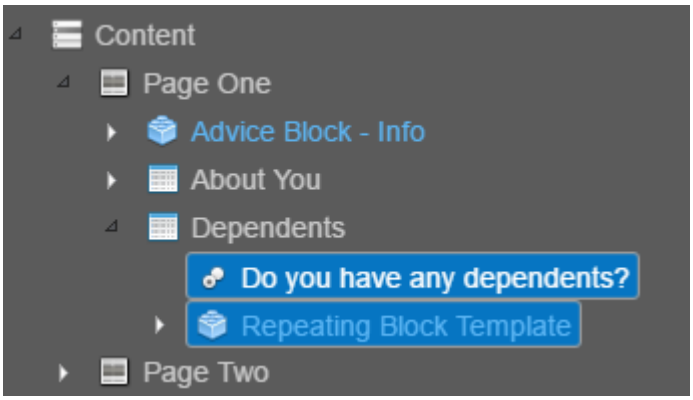
Click **Save** to continue.



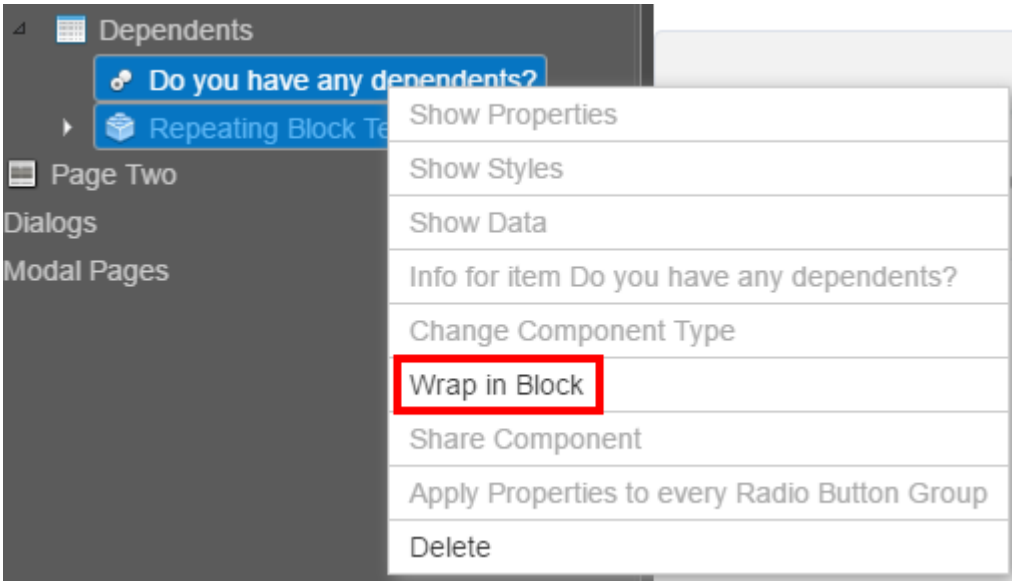
Wrap Components in a Block

If a shared component is made up of more than one component, you must wrap the relevant components in a block first, then create the shared component.

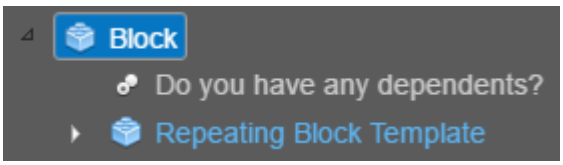
To wrap multiple components in a block, select all the components from the structure panel using either SHIFT or CTRL (see below).



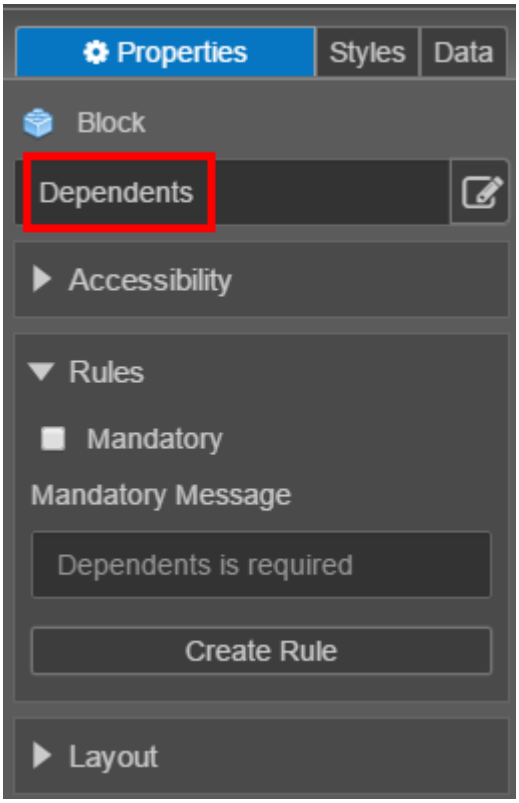
Right-click one of the selected items and select **Wrap in Block**.



The **Block** is created and the components are displayed within.



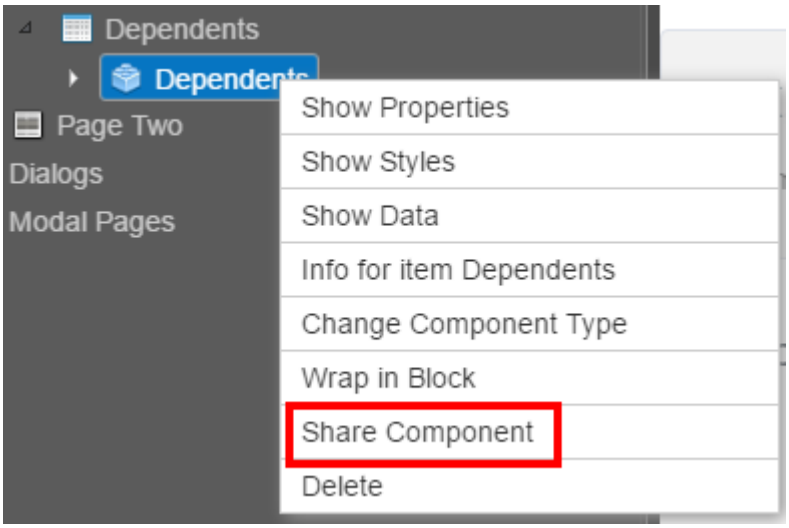
Use the **Properties** panel to rename the block.



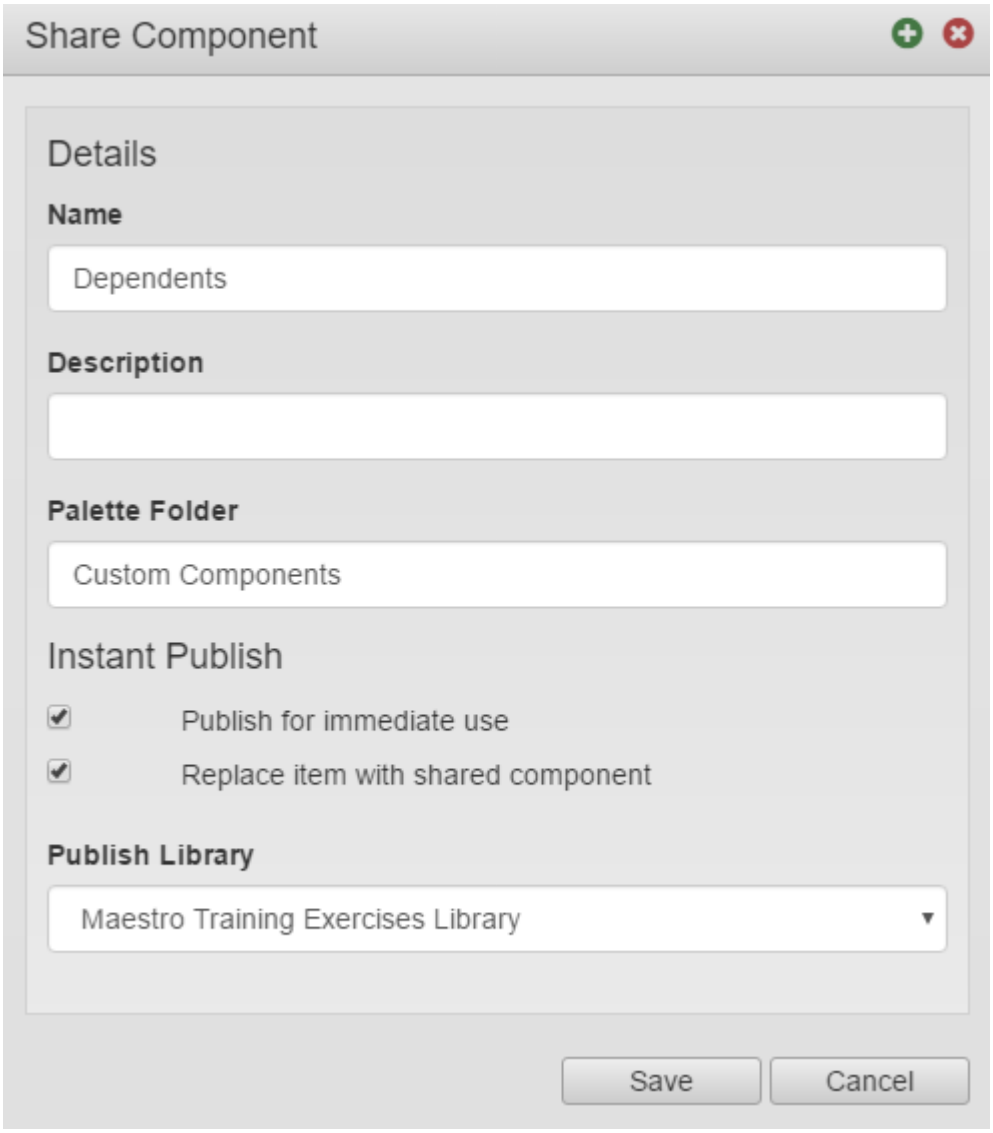
Create Shared Component

Shared components are stored outside a form, in the Components folder of the project. Once a Shared component is published, it is stored in a Library. Shared components can be updated outside of a form. When they're updated, any forms using that component are updated automatically.

To create a Shared Component, right-click the block and select **Share Component**.



The **Share Component** window will display.



Name: is the name of the shared component displayed in the Palette panel.

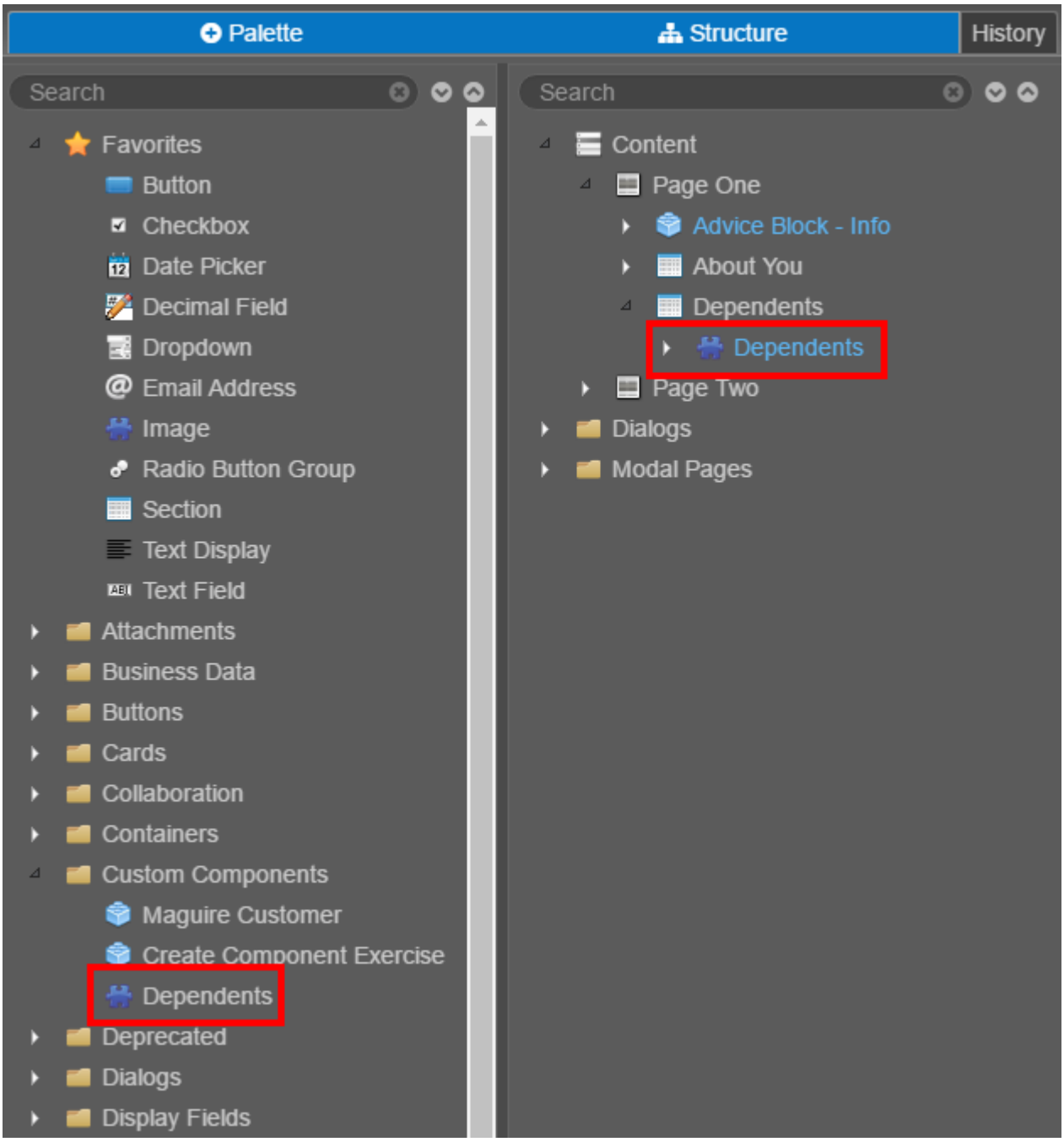
Description: contains details about the shared component like the components in contains.

Palette Folder: is the folder where the shared component is stored within the Palette structure. If you enter a folder name that's already created, the shared component will be added to the folder. If you enter a name not currently be used, a new folder will be created.

Instant Publish – Publish for immediate use: publishes the shared component as soon as you click Save. This adds the component to the library, so other forms can access and use the shared component.

Instant Publish – Replace item with shared component: removes the block currently in the form and replaces it with the shared component. This allows the component in the form to be updated whenever changes to the shared component occur.

The **Structure** panel is updated with the new shared component and the folder in the **Palette** panel will contain the shared component.



Layout Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for working with the layout of your form. During this exercise you will use an existing form to edit the layout of components.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

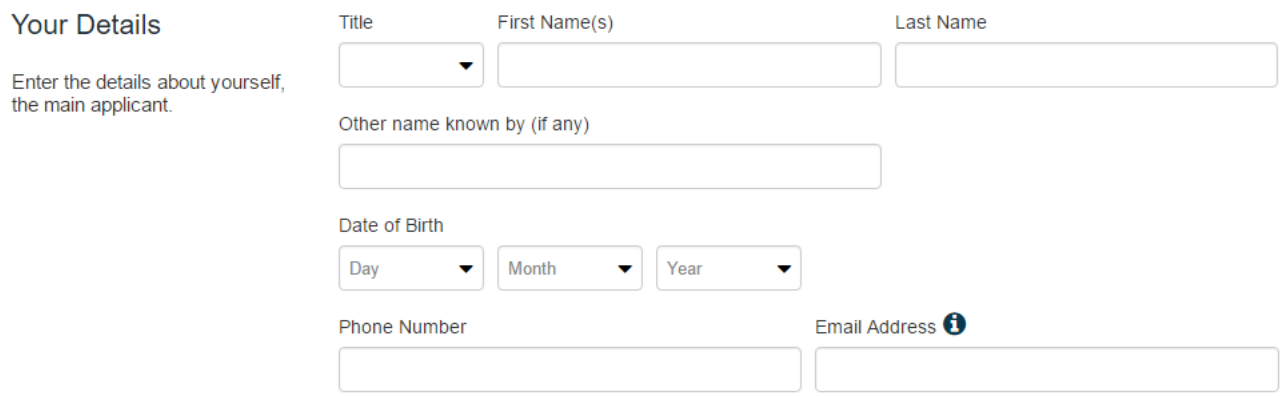
You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises v5.1 > Layout Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1: Open the Existing Form Layout Begin.

1. In the Navigator Panel, expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Double-click **Layout Begin**.
5. Click the **Edit Version** button to open the form.

Step 2: Adjust the Layout of Title, First Name and Last Name

On the Applicant Details page, adjust the components so they appear on the same line when viewing the form on large screens. Update the 'Your Details' section to match the image below.



Set Title to 2 columns wide.

Set First Name(s) to 5 columns wide.

Set Last Name to 5 columns wide.

Set Other name known by (if any) to 7 columns wide.

Set Phone Number to 6 columns wide.

Set Email Address to 6 columns wide.

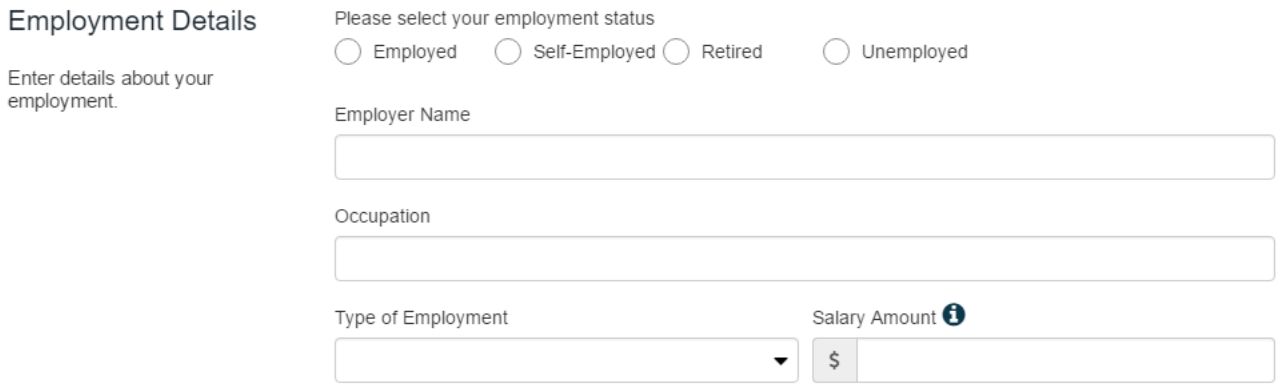
Best practice: only change the width values for the Large and Full Screen sizes.
In the Editor, you'll only see the changes made to the Full screen size.
Use Preview to check the form and make any necessary adjustments.

Once you've made room on a row, you can drag a component into position if it doesn't automatically move to the larger space.

Click **Save**.

Step 3: Adjust the Employment Details Components

Adjust the components so they appear like the screenshot below.



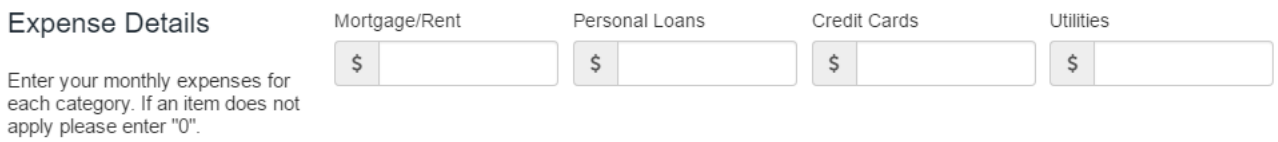
Set Employer Name and Occupation to 12 columns wide.

Set Type of Employment and Salary Amount to 6 columns wide.

Click **Save**.

Step 4: Adjust the Living Expenses Components

On the Income and Expenses page, adjust the components so they appear as four columns on the same row.



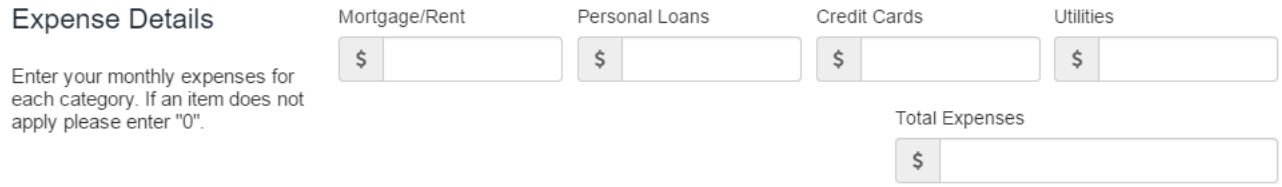
Set all four components to 3 columns wide.

Once you've made room on a row, you can drag a component into position if it doesn't automatically move to the larger space.

Click **Save**.

Step 5: Add a New Component to the Living Expenses Section

In the Expense Details section, add a new Total Expenses component.



> Notice: the currency symbol for Total Expenses is added by using the Currency Field.

Set Total Expenses to 5 columns wide.

Set the Offset for Total Expenses to 7 columns.

Click **Save**.

Extra Help

Need some extra help?

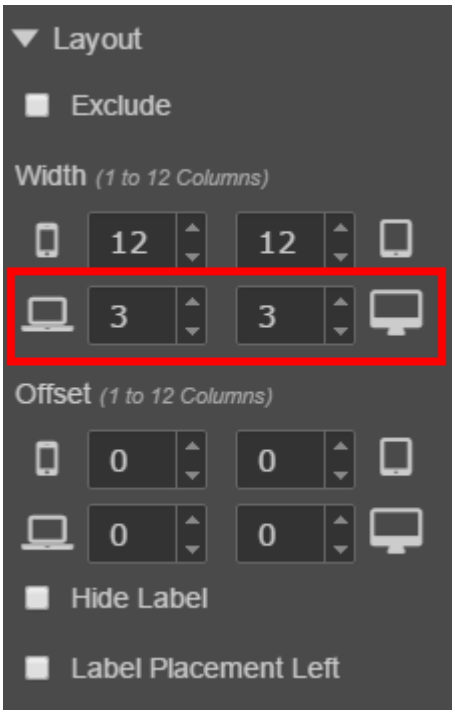
Use the information below to assist you in completing the exercise steps.

Change the layout of components

Select the component you want to change. Adjust the Width values within the Layout section (Properties panel) to make any changes.

Decrease the column width for the relevant screen size.

You will only see changes in the Editor when you adjust the full screen layout.

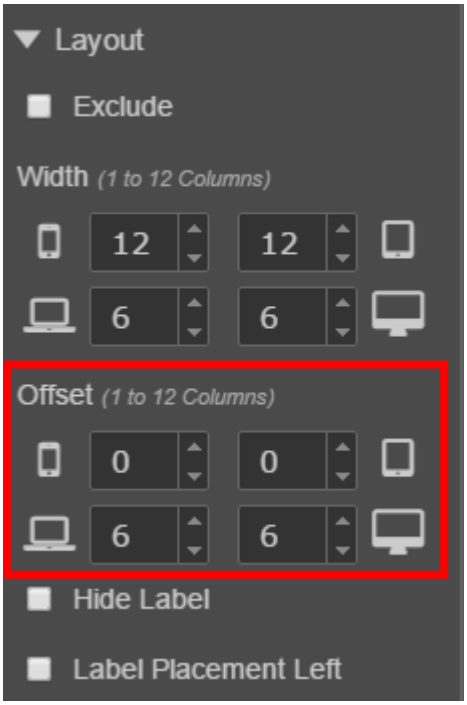


Change the position of components (using Offset)

Components can be shifted to the right or left within their row. Select the component you want to change.

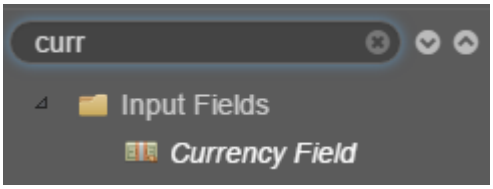
Adjust the Offset values within the Layout section (Properties panel) to make any changes.

You can increase the offset to move the component to the right or decrease the offset to move the component to the left.

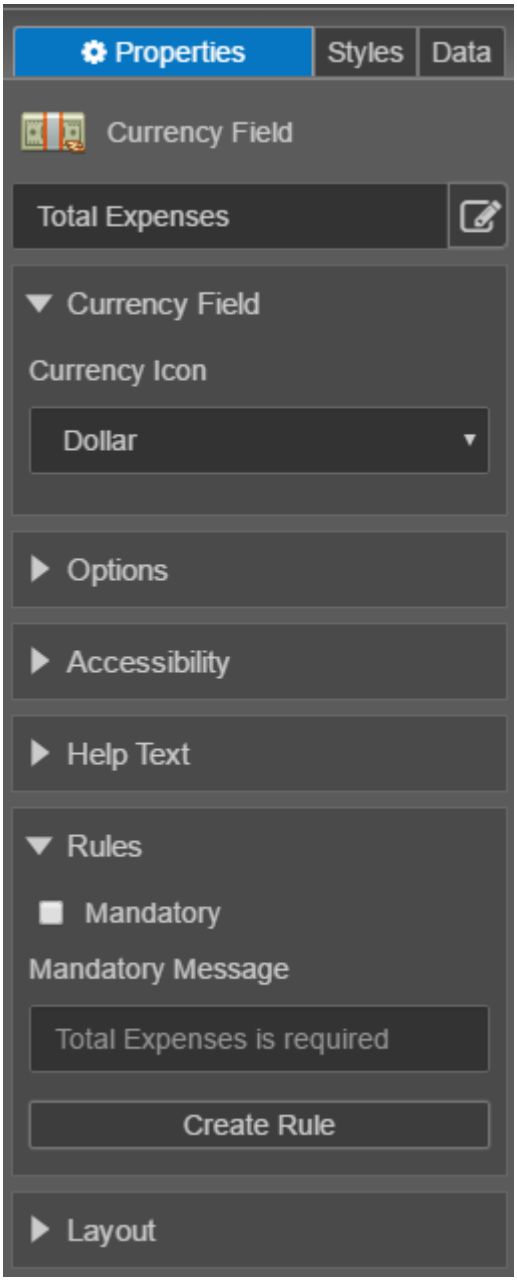


Add a Currency Field

The Currency Field component automatically includes a currency symbol. Use Search in the Palette to find the Currency Field and add (drag) it to your form.



Use the Properties panel to make changes to the field, including the currency icon used.


Properties
Styles
Data
Currency Field
Total Expenses
Currency Field
Currency Icon
Dollar
Options
Accessibility
Help Text
Rules
Mandatory
Mandatory Message
Total Expenses is required
Create Rule
Layout

Pages and Sections Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills with pages and sections. During this exercise, you will use an existing form to edit pages and sections as well as add new ones.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find **Extra Help** at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the **Maestro Exercise Project: Maestro Training Exercises v5.1** > **Pages and Sections Finish**. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1: Create a New Version

Create a new version of the existing form named **Pages and Sections Begin**.

Follow the steps below to create the new form.

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Select **Pages and Sections Begin**.
5. Click **New Version**.
6. Enter the following **Form Description**:

   Description: This form version is to practice working with sections.

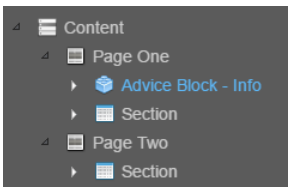
7. Click **Create**.
8. Click **Edit** to open the form.

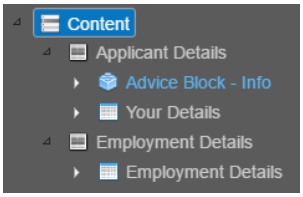
Step 2: Update the Names of the Existing Pages and Sections

In this form, you will find two pages, each containing a section. You will change the names of both pages and their sections.

Use the Structure panel to select the pages and sections you want to update.

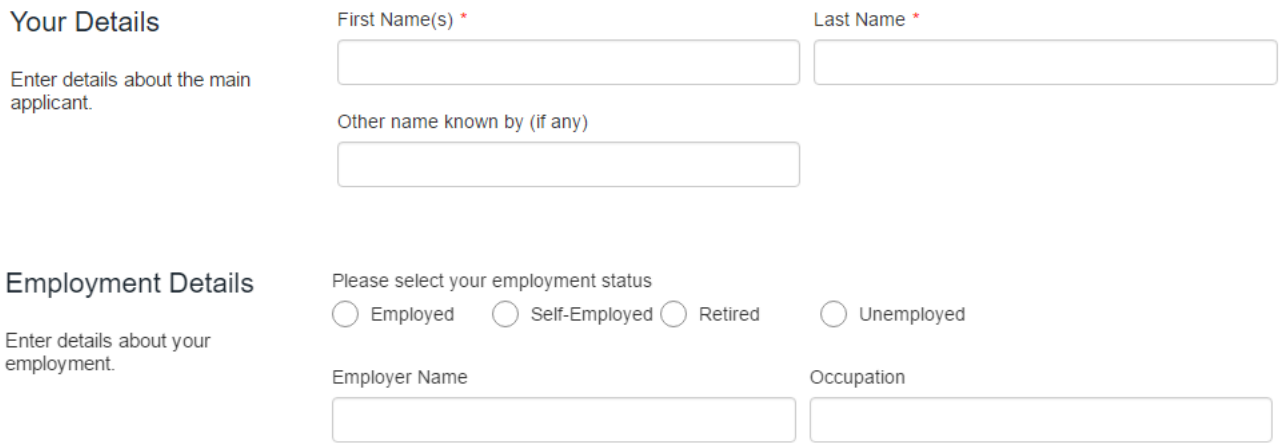
Starting Structure Panel | Updated Structure Panel





Update the Section - Additional Text (using the supplied text) so your form matches the images below.

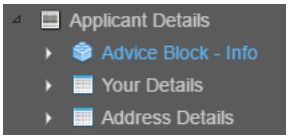
- Your Details: Enter the details about the main applicant.
- Employment Details: Enter details about your employment.



Click **Save**.

Step 3: Add a new section to the Applicant Details page

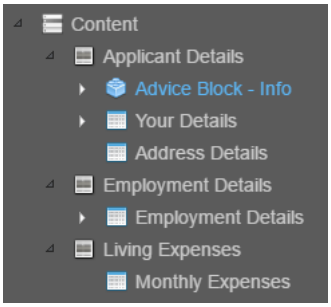
Add your new section after the existing section on page one. Your Structure panel should appear similar to the image displayed.



Click **Save**.

Step 4: Add a New Page with a Section

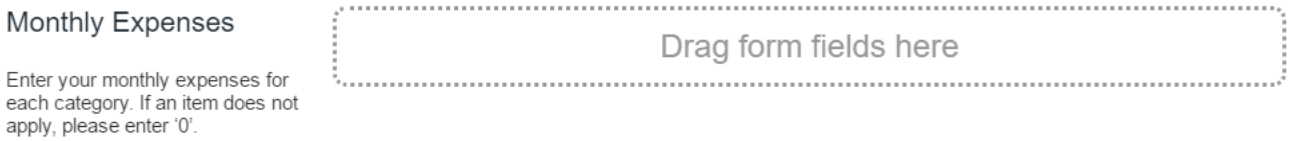
Add a new page (called 'Living Expenses') after the existing pages. Add a new section (called 'Monthly Expenses') to the page. Your Structure panel should match the image below.



Add the following Additional Text for the Monthly Expenses section (text below).

Monthly Expenses: Enter your monthly expenses for each category. If an item does not apply, please enter '0'.

The Living Expenses page should match the image below.



Click **Save**.

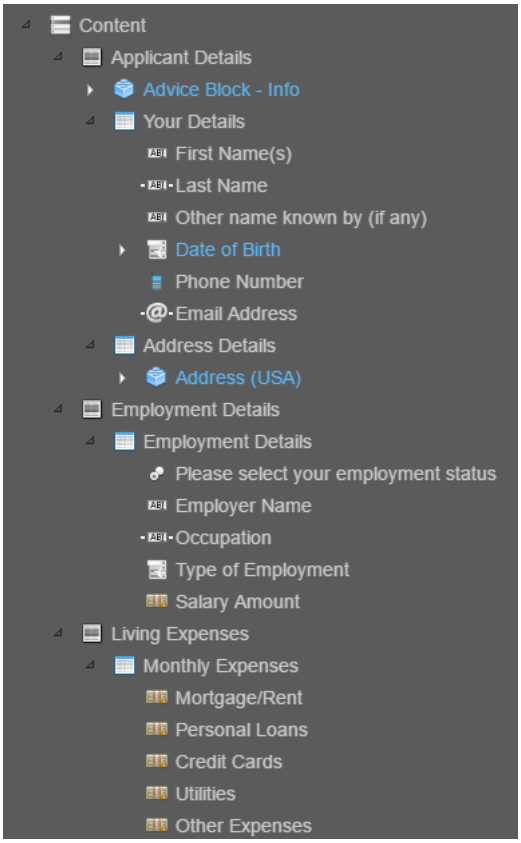
Step 5: Rearrange the Components

Using the Structure panel:

- Move the Address (USA) block to the Address Details section.
- Move the expense questions (Page Two) to the new Monthly Expenses section (Page Three).

> You can select multiple components at one time by using SHIFT or CTRL.

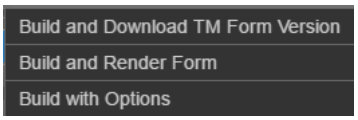
The image below shows how the Structure panel should look once the components have been rearranged.



Click **Save**.

Step 6: Build the TM Form Version

Build and Download the TM Form Version.



Extra Help

Need some Extra Help?

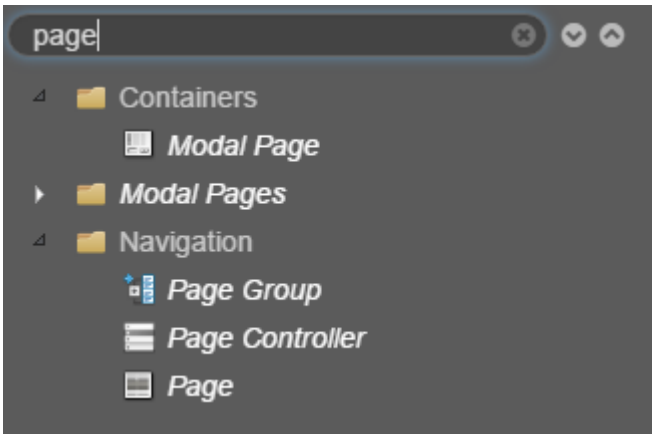
Use the information below to assist in completing the exercise steps.

Add a new page

Use the Search box in the Palette to find the Page component.

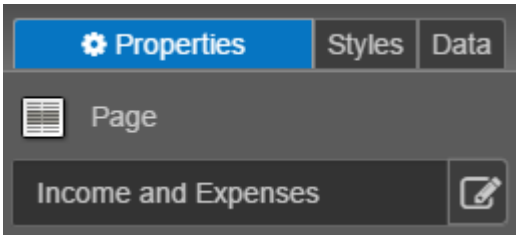
Drag the Page component to the Content container on the Structure panel.

Pages Groups and Pages cannot be added into the Editor pane.



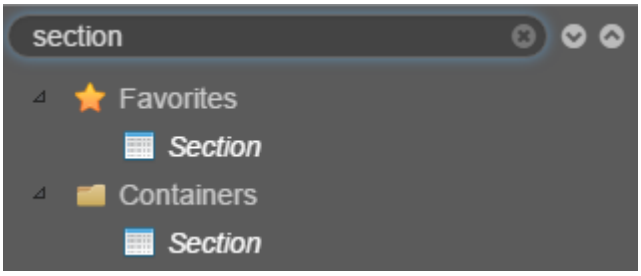
You can also right-click the Content folder (Structure panel) and choose Add Page Group.

Use the Properties panel to change the name of the page.



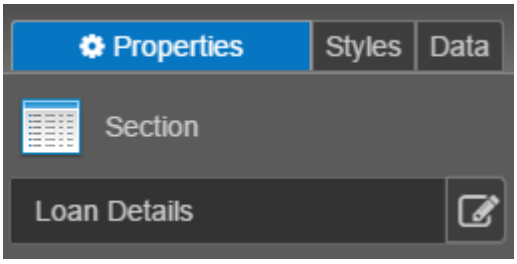
Add a new section

Sections can be added to pages to help organize the display of the components. Use the Search box in the Palette to find the Section component (or look under 'Favourites'). Drag the Section component to the Page container on the Structure panel.



You can also drag a new 'Section' component onto the form.

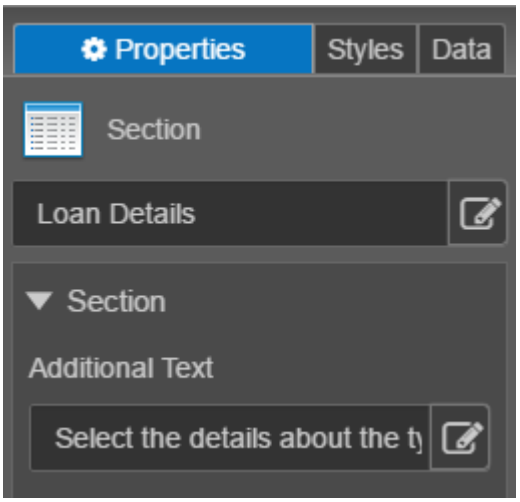
Use the Properties panel to change the name of the section.



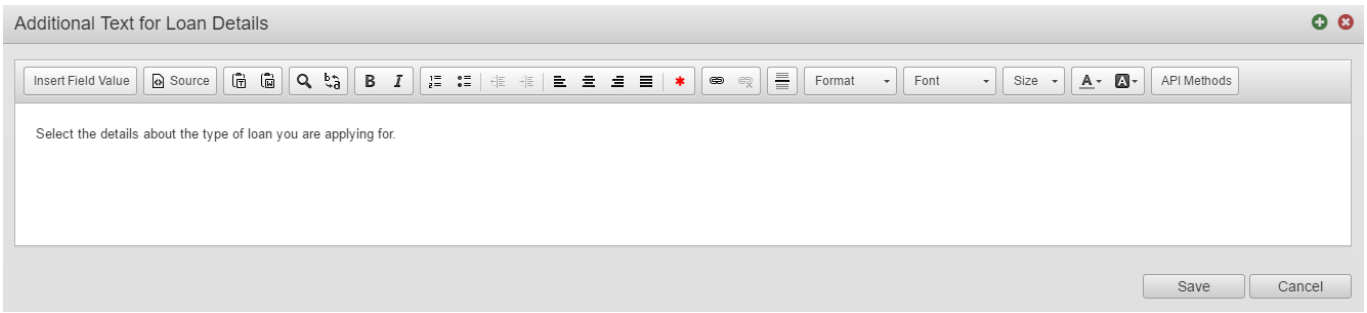
Additional Text

Use the Properties panel to customize the text below the section title.

In the Properties panel, enter text in the Additional Text entry box.

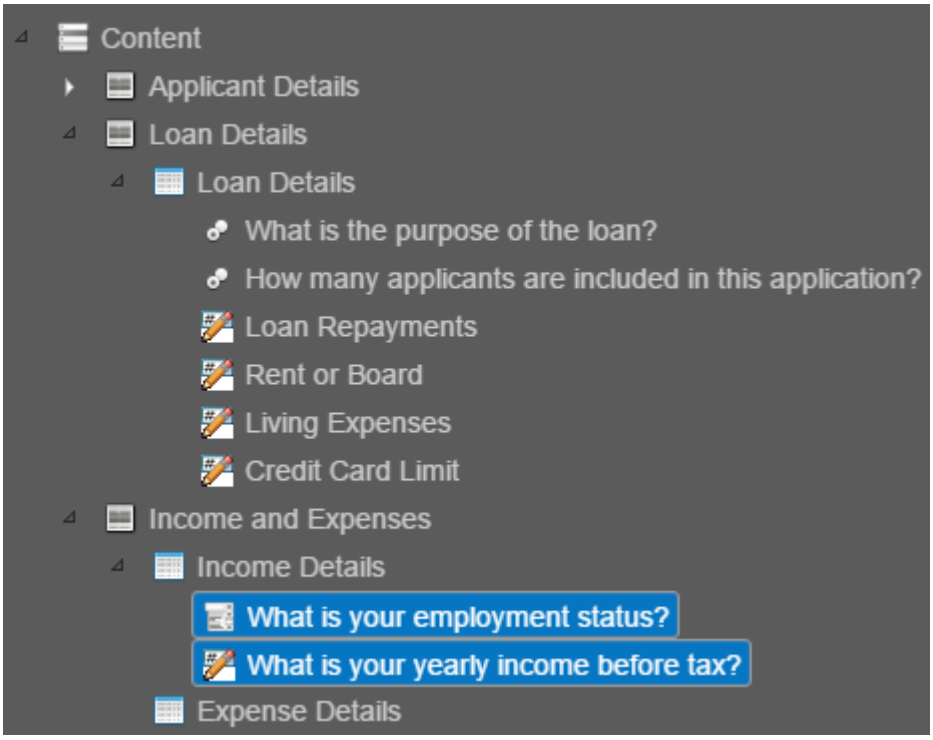


You can click the Edit button to display the Additional Text window. Here you can edit and format the additional text displayed on the form.



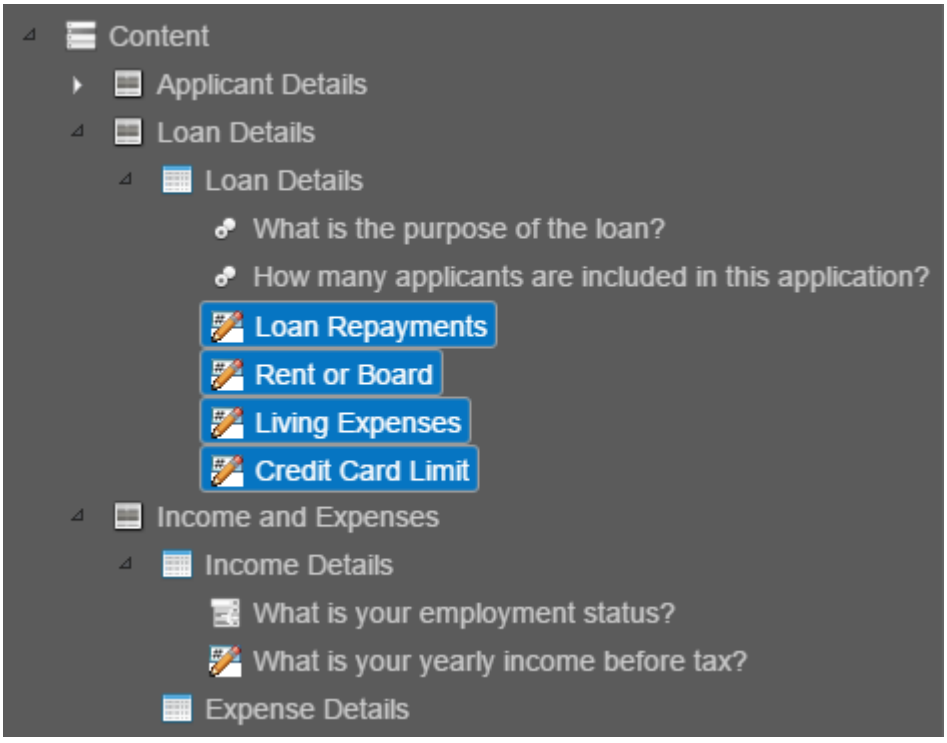
Rearrange Components

Use the Structure panel to drag and drop components between pages and/or sections.

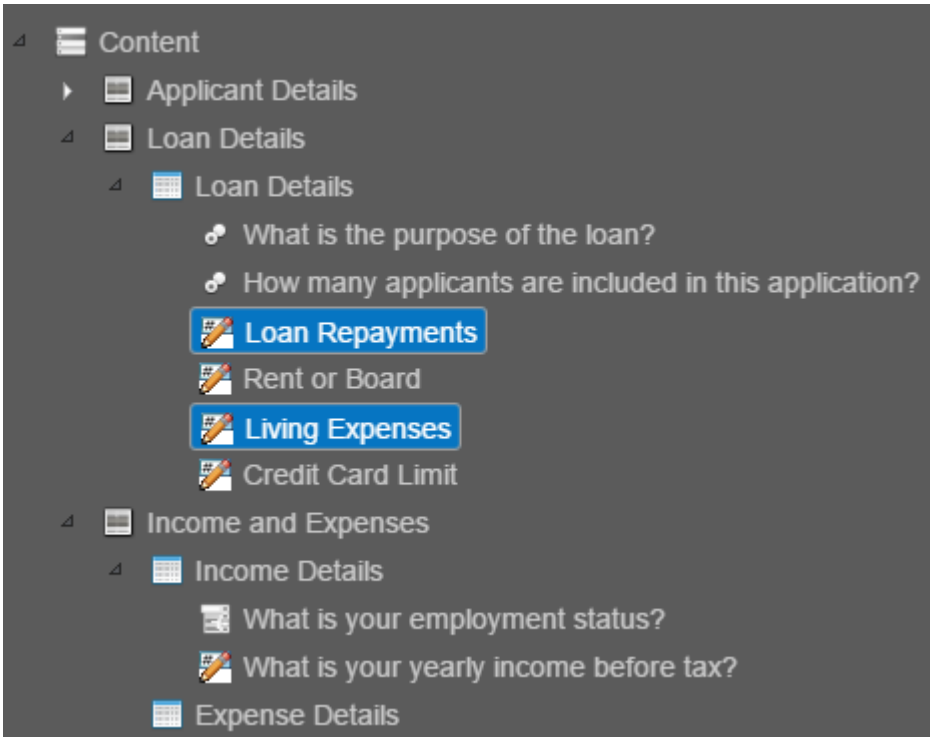


Select multiple components at one time using SHIFT or CTRL. Once selected, you can drag them to a different container.

Select adjacent components using SHIFT (below)



Select non-adjacent components using CTRL (below)


Content
Applicant Details
Loan Details
Loan Details
What is the purpose of the loan?
How many applicants are included in this application?
Loan Repayments
Rent or Board
Living Expenses
Credit Card Limit
Income and Expenses
Income Details
What is your employment status?
What is your yearly income before tax?
Expense Details

Repeating and Shared Components Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for using repeating components and shared components. During this exercise you will use an existing form to add repeating and shared components.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises > Repeating and Shared Components Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Open the Existing Form 'Repeating and Shared Components Begin'

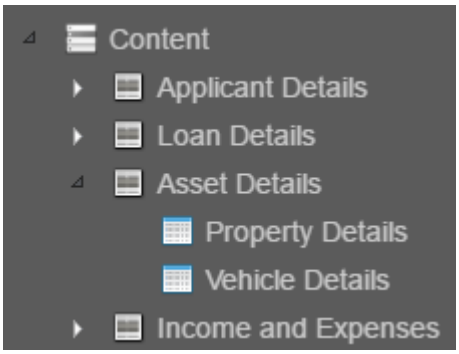
1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Double-click **Repeating and Shared Components Begin**.
5. Click the **Edit Version** button to open the form

Step 2 Add a New Page with Sections to Your Form

Add a new Page called **Asset Details**.

Add two new Sections to this page called **Property Details** and **Vehicle Details**.

Your Structure panel should match the image below.



Click **Save**.

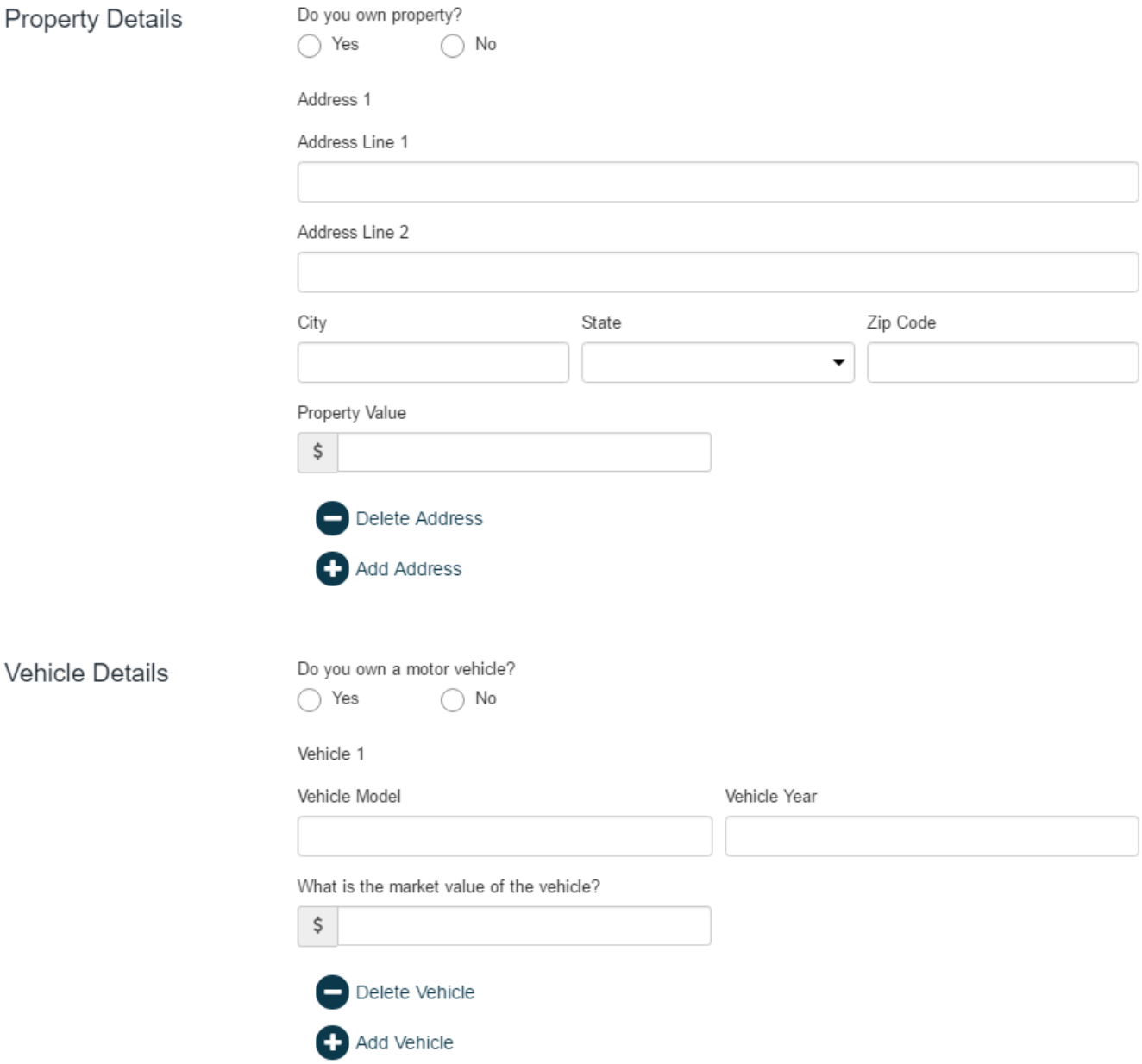
Step 3 Add Components to the Page

Add components to the new page so it matches the image below.

> Notice: the Address 1 block and Property Value component are within a Repeating Block Template. Before you add components to the page, you should add the Repeating Block Template first, then add the components to the block.

Create a Show if rule to only show the Repeating Block Templates when the user selects Yes to owning Property or a Vehicle.

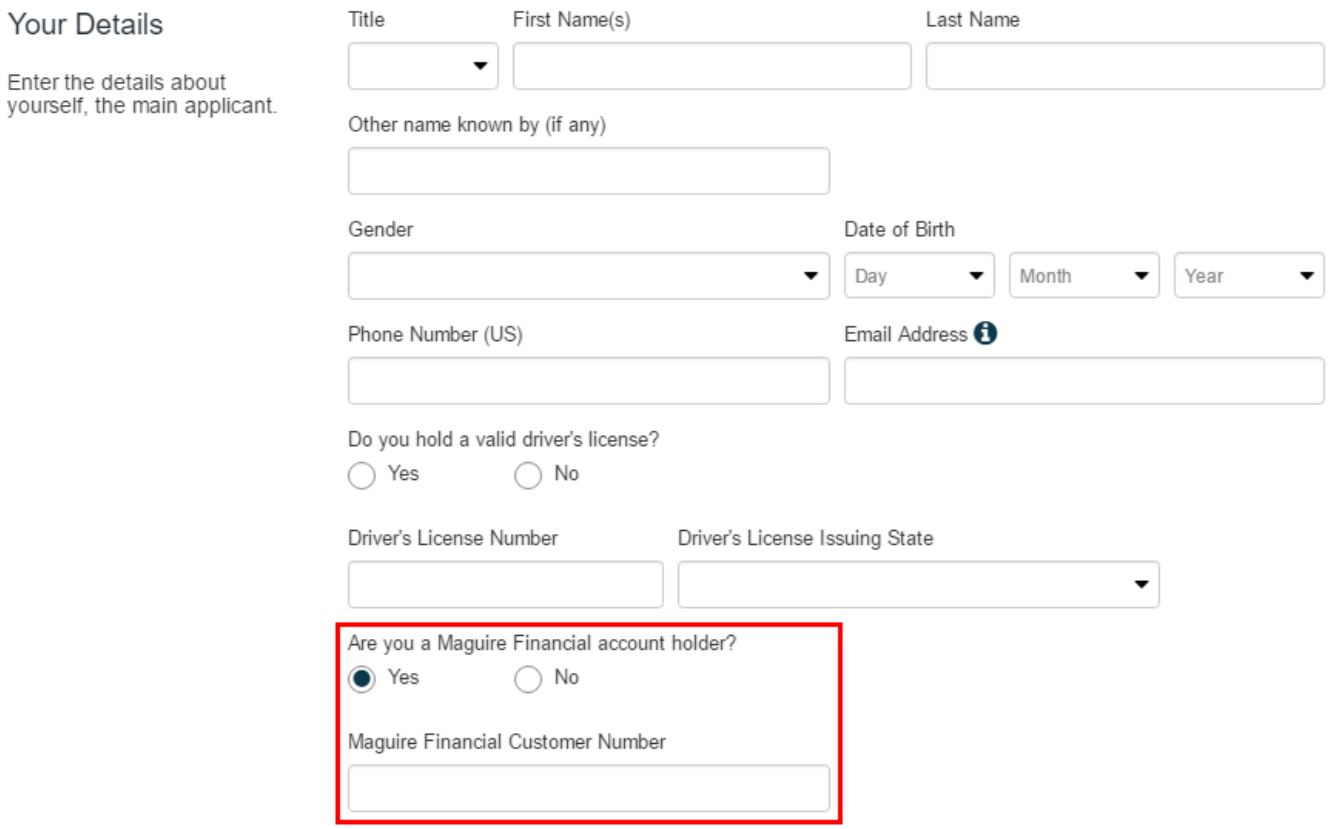
Click **Save**.



Step 4 Add a Shared Component

Add the Maguire Customer shared component to the bottom of the Your Details section.

Click **Save**.



Shared components can be stored anywhere in the Palette, so use the Search if you're unsure of a component's location.

Extra Help

Need some extra help?

Use the information below to assist you in completing the exercise steps.

Add a new Page and Section

Display the Palette and Structure panels side by side (use button in top left).

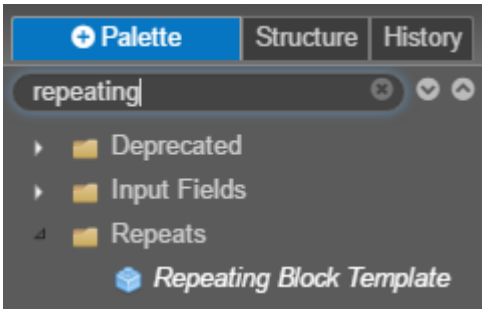
Search for the Page component and drag it onto the Content folder (Structure panel). Drag the new Page into the correct order.

Drag two new Section components onto the new Page.

Using a Repeating Block Template

The Repeating Block Template includes all of the necessary items to easily setup repeating functionality.

Use the Palette Search to find the Repeating Block Template (see below).



Drag the component onto your form.

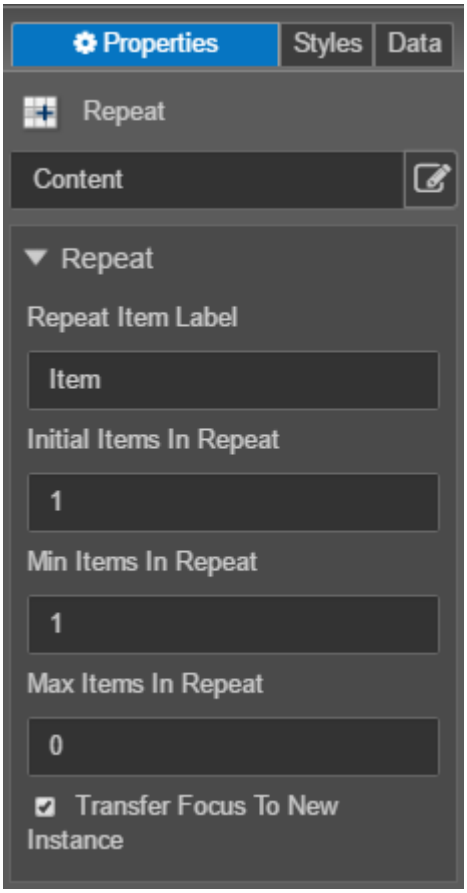
You can add the components you want to repeat into the field area of the repeating block.



To make any adjustments to your repeating block, select Content within the Repeating Block Template (Structure panel).



With Content selected you can edit values in the Properties panel.



Repeat Item Label: this updates the label at the top of the block, as well as the button text.

Initial Items In Repeat: determines how many repeating components are on the form by default.

Min Items In Repeat: determines the minimum number of repeating components that must be displayed on the form.

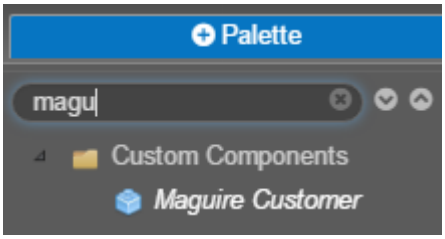
Max Items In Repeat: determines the maximum number of repeating components a user can add to the form.

Using a Shared Component

Shared components are stored outside of a form, in the Components folder within the project. A shared component can be updated outside the form. When updates are made, all forms using that component are updated automatically.

Use Search in the Palette to locate the shared component you want to use.

Drag the shared component to the form. The shared component will be added.



The shared component will contain the components along with any custom properties.

Are you a Maguire Financial account holder?

Yes No

Maguire Financial Customer Number *

Rules without Scripting Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating rules without scripting. During this exercise you will use an existing form to add non-scripted rules.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

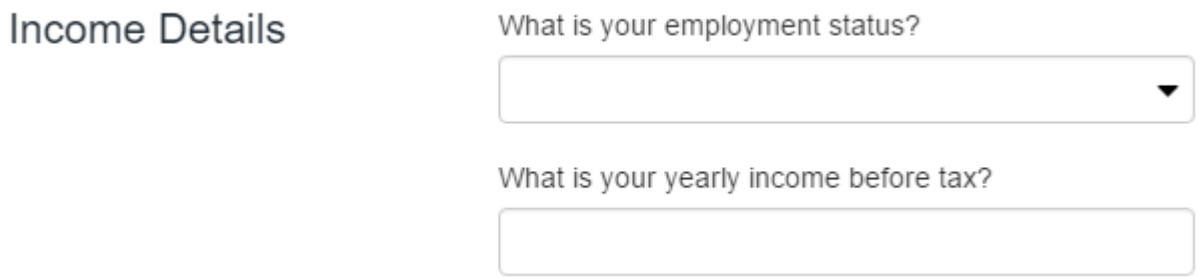
You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises v5.1 > Rules without Scripting Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Open the existing form Rules without Scripting Begin.

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Double-click **Rules without Scripting Begin**.
5. Click **Edit Version** to open the form.

Step 2 Create a Rule for Income Details

Create a Show If rule that displays the 'yearly income' component if the 'employment status' is set to 'Employed' or 'Self-Employed'.

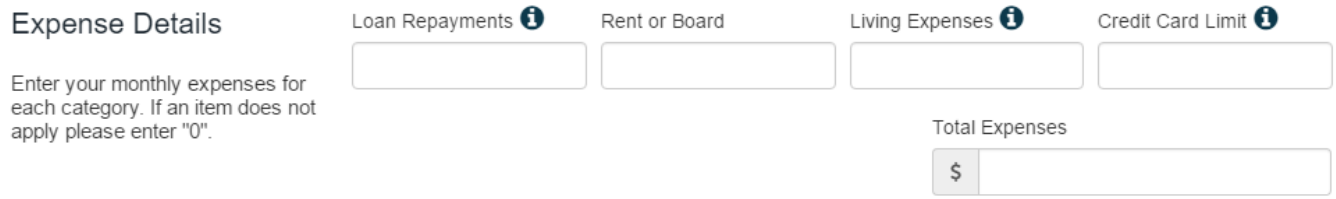


Click **Save**.

Step 3 Calculate the Total of the Four Expenses in the Expense Details Section

Calculate the total of the four expenses in the Total Expenses field.

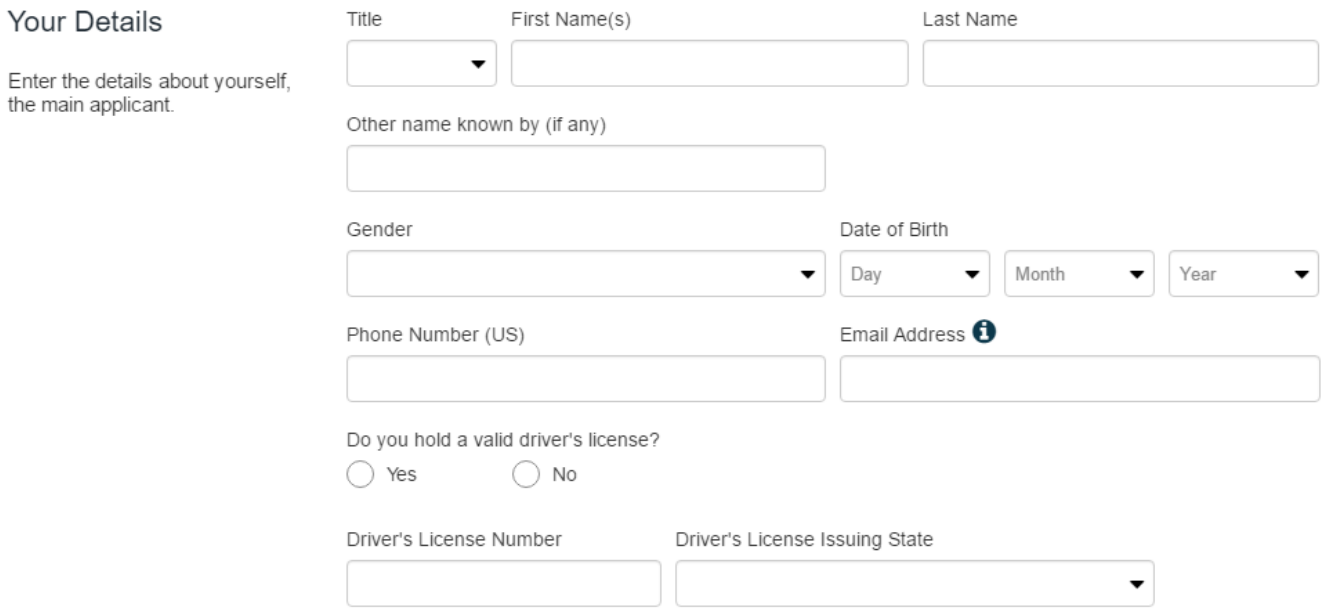
Set up another rule to make Total Expenses read only.



Click **Save**.

Step 4 Add the Driver License Components to the Your Details Section.

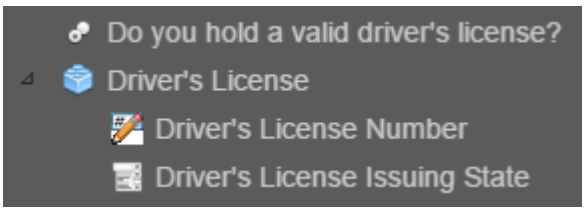
Add the Driver License components shown below.



Click **Save**.

Step 5 Create a Rule for Driver License

Wrap the Driver's License Number and Driver's License Issuing State components into a block, then add a Show If rule so the components only display when the user selects Yes to having a driver's license.



Click **Save**.

> Important: you should wrap Driver's License Number and Driver's License Issuing State in a block before creating the rule. This way you only have to apply the rule to the block, rather than applying to individual components.

Extra Help

Need some extra help?

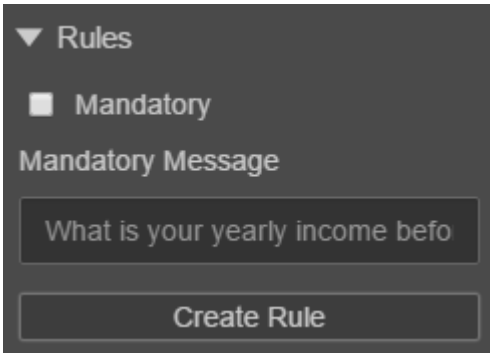
Use the information below to assist you in completing the exercise steps.

Create a Show If Rule

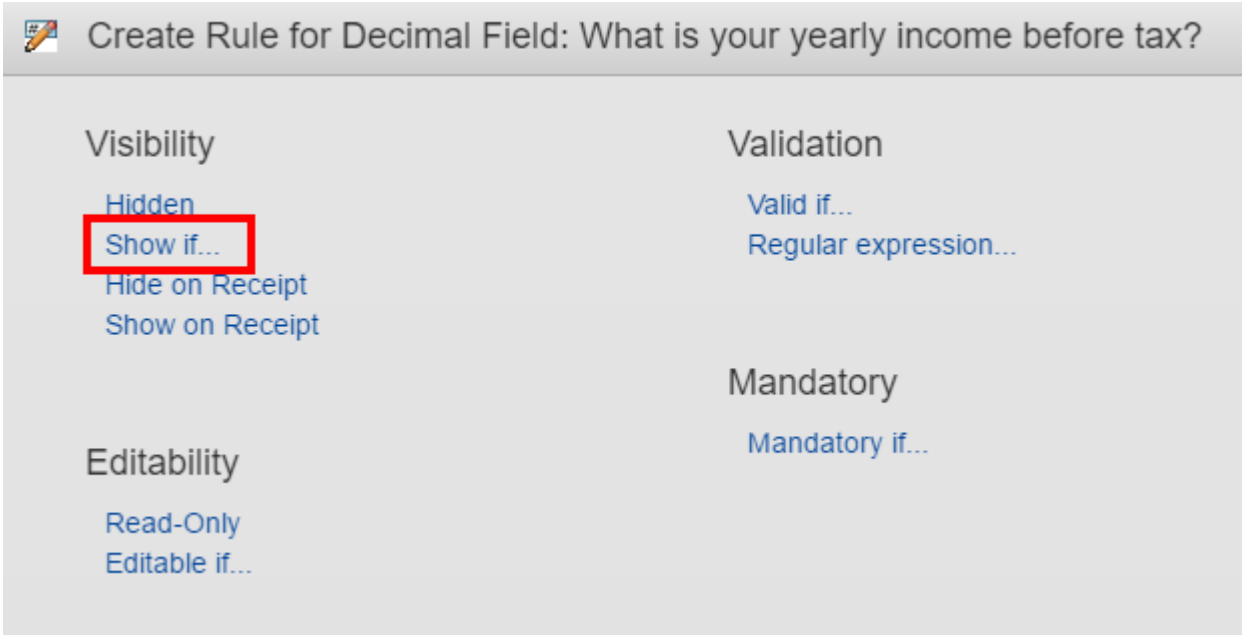
A Show If rule allows you to show or hide a component based on the selection of another component.

Select the component you want to show or hide.

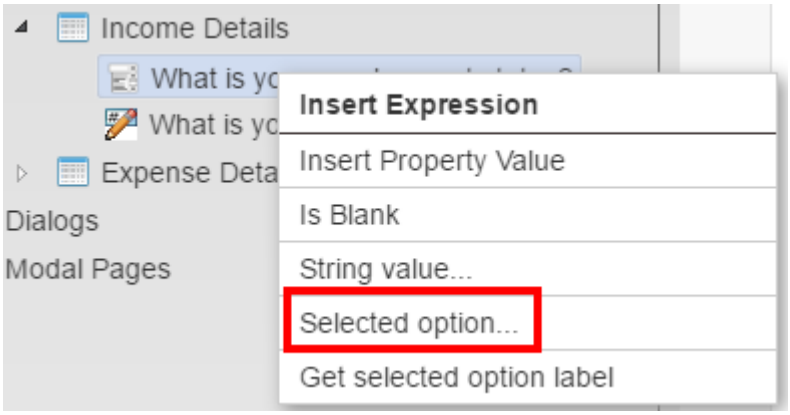
Click Create Rule in the Rules section (Properties panel).



Select Show If.

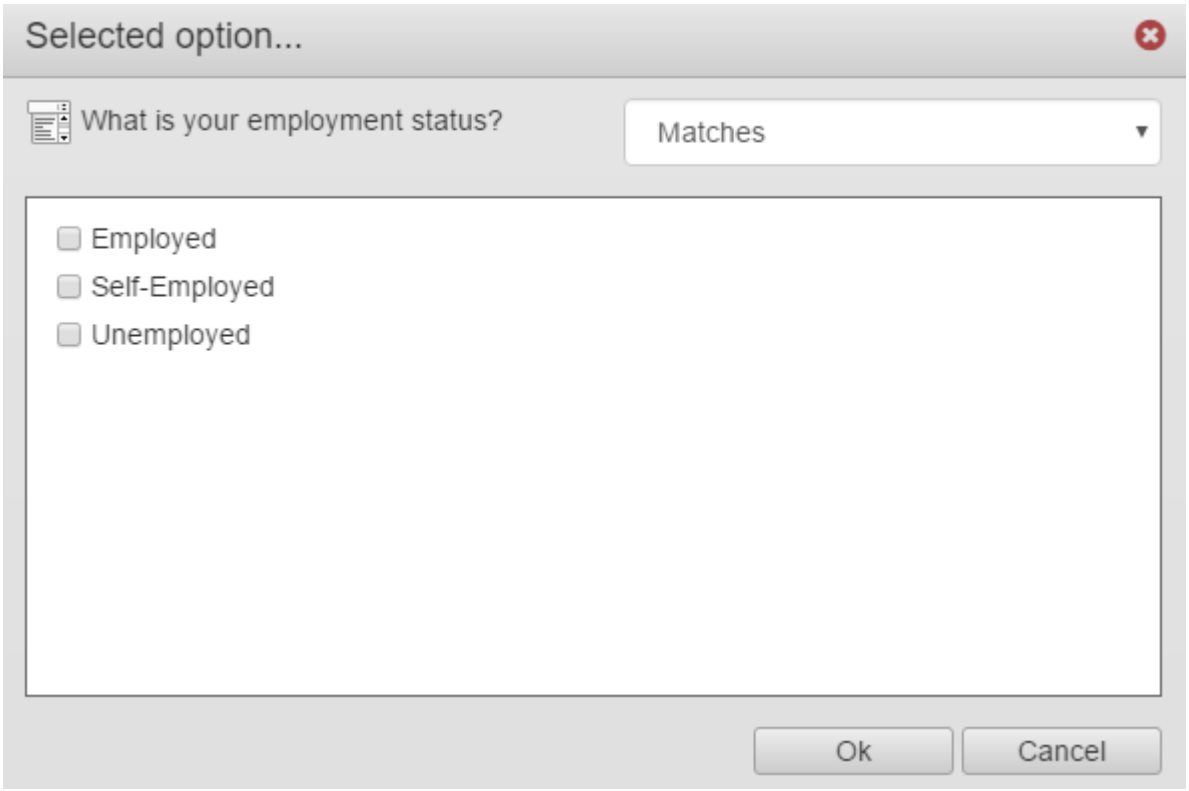


Right-click the component that determines if the selected component is displayed or hidden and click Selected option.

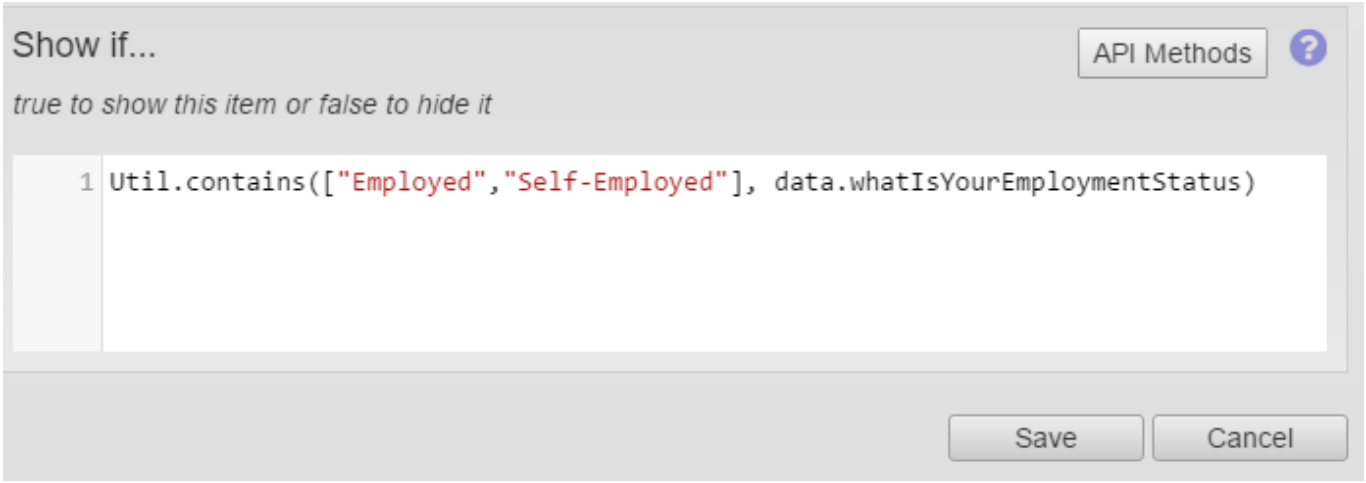


In the top right corner of the Selected option window, select if the option should match or not match, then select the relevant options.

Click OK.

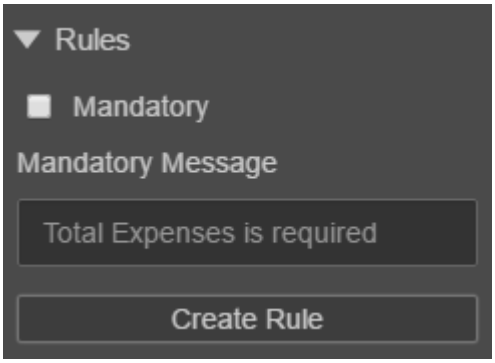


The script for the rule and selected components is displayed. Click Save.

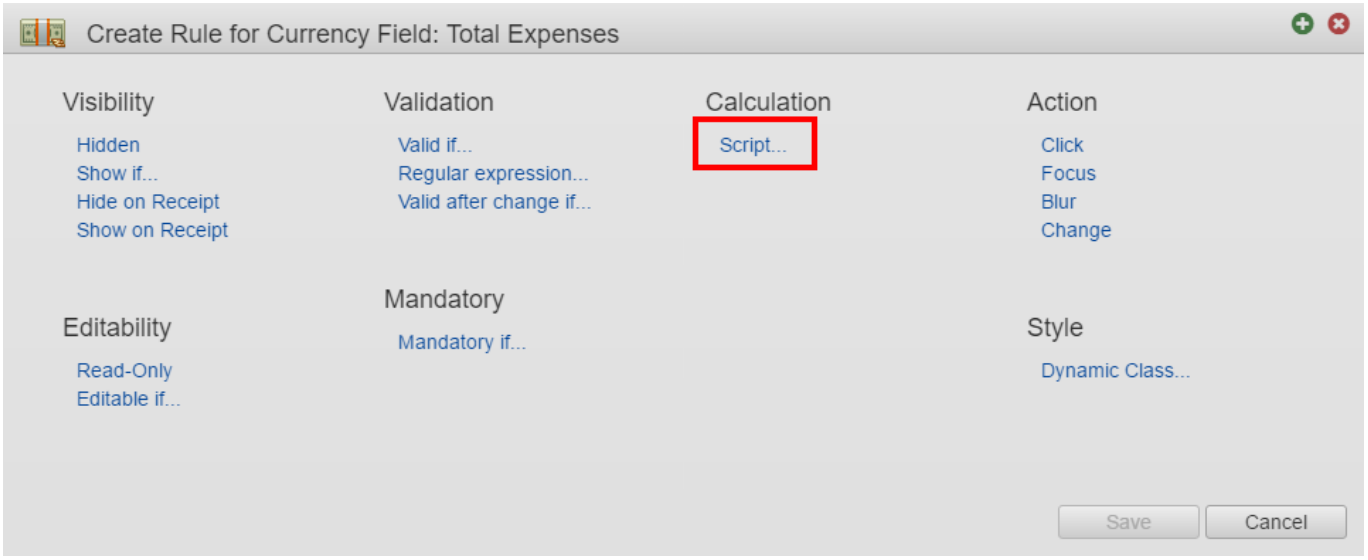


Create a Script Rule

A Script rule will perform calculations based on selected components. Select the component where the calculation will be displayed. Click Create Rule in the Rules section (Properties panel).

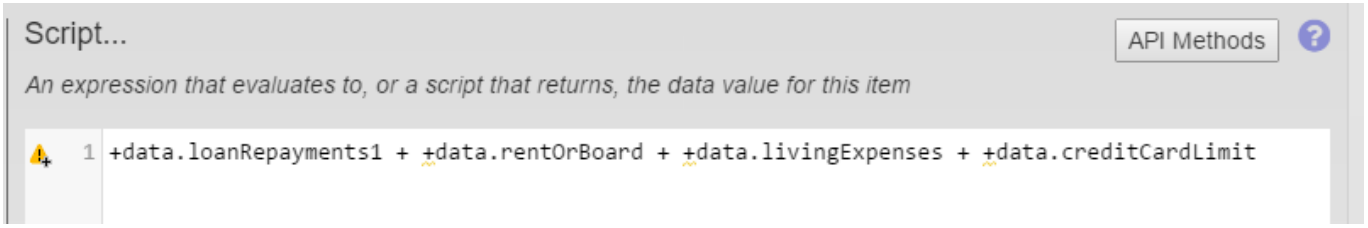


Click Script.



Double click each component you want to use in the calculation. Include a plus sign (+) between the components.

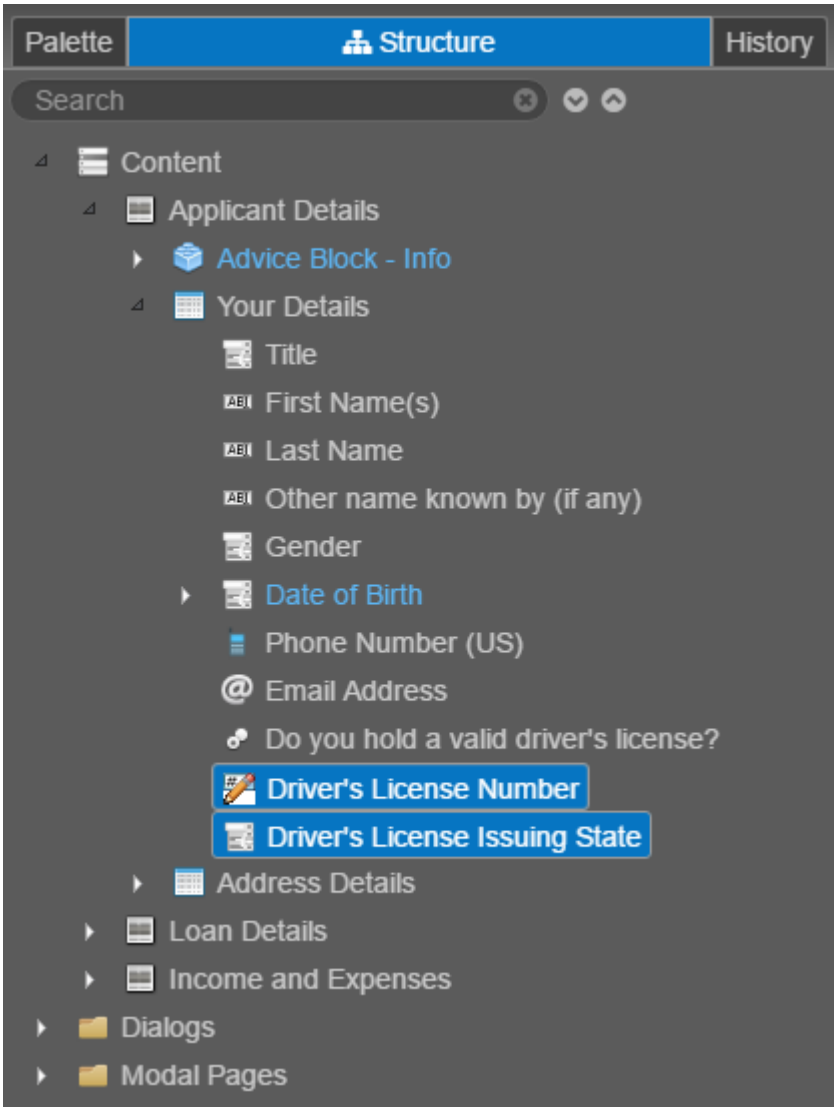
The selected components will display in the list. Click **Save**.



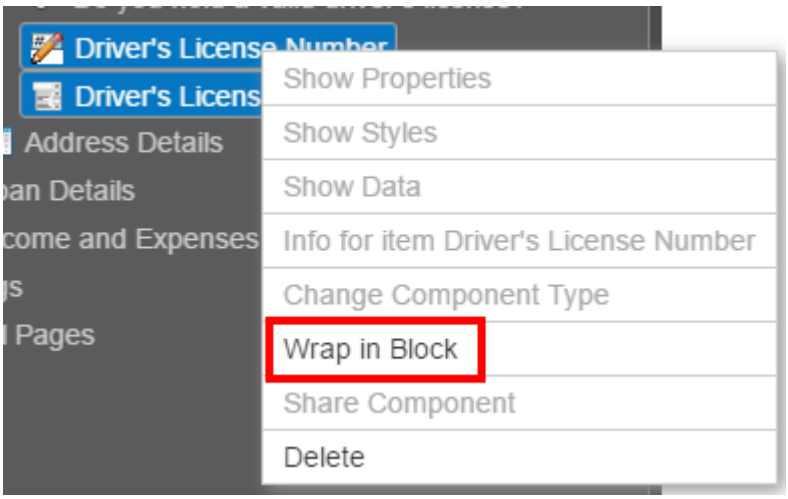
Wrap Components in a Block

You can wrap components in a block to easily manage related components.

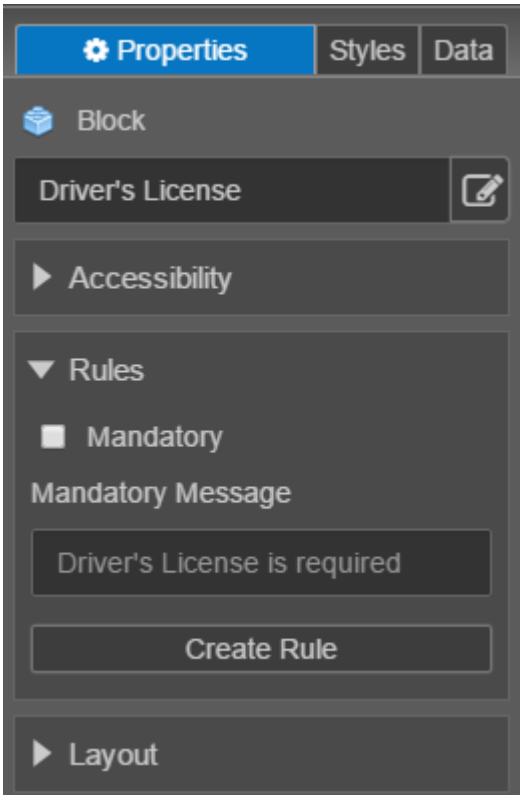
Select the components in the Structure panel you want to wrap in a block.



Right-click one of the selected components and click Wrap in Block.



With the block selected, use the Properties panel to rename the block.


Properties
Styles
Data
Block
Driver's License
Accessibility
Rules
Mandatory
Mandatory Message
Driver's License is required
Create Rule
Layout

Two Level Navigation Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for working with two level navigation. During this exercise you will create page groups, which will allow you to use two level navigation within your form.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises v5.1 > Two Level Navigation Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Open Two-Level Navigation Begin

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Double-click **Two Level Navigation Begin**.
5. Click **Edit Version** to open the form

Step 2 Add Page Groups to the Form

Add Page Groups to the form, then move the relevant pages into each group. Your Structure panel should match the After image below.

Before After





To add a Page Group, you need to drag the component from the Palette panel onto the Structure panel.

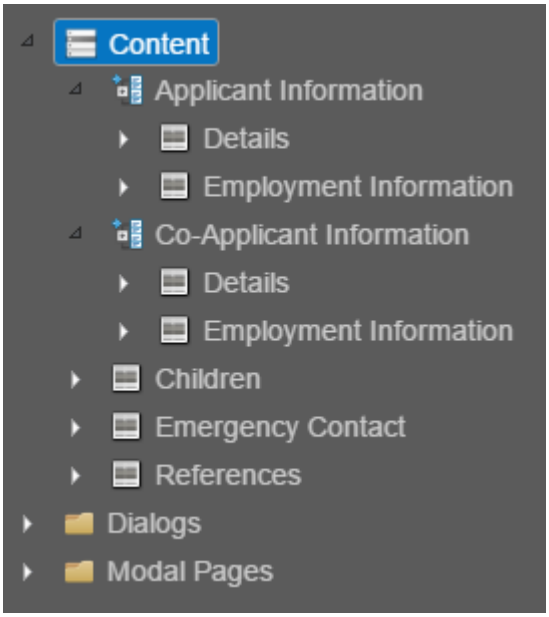
Page groups cannot be dragged directly onto a form or template.

> Notice: some of the pages have been rearranged.

Click **Save**.

Step 3: Rename Pages

Rename the Pages added to each Page Group. Your Structure panel should match the image below.



Step 4 Set Second Level Navigator to Equal Width

Change the width of the Pages on the second level navigator so that they are equal. Your second level navigator should match the image below.





To change the width of the Second Level Navigator:

1. Click **Form Options**.
2. Select **Navigator Second Level.**
3. Select **Equal Width.**
4. Check **Equal Width**.
5. Click **Save**.

Extra Help

Need some extra help?

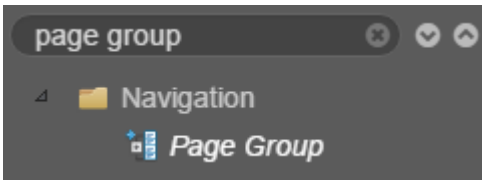
Use the information below to assist you in completing the exercise steps.

Using a Page Group

Page Groups let you group pages together. Adding page groups automatically creates a two level navigation. Each page group is displayed on the first level navigator and any pages in the page group are displayed on the second level navigator.

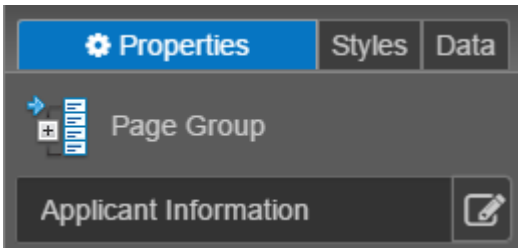
Use Search to find the Page Groups.

Drag the component onto the Structure panel.

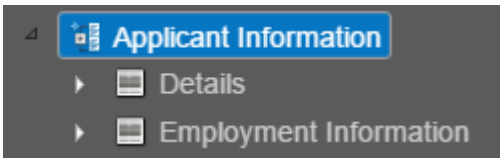


Use the Properties panel to rename the page group.

The Page Group label is automatically used for the first level navigator. If you want the label and menu text different, change the menu text in the Options section on the Properties panel.



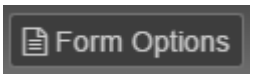
In the Structure panel, drag the pages you want grouped onto the page group component.



Set the Second Level Navigator to equal widths

By default, the width of Second Level Navigator menu items is determined by the length of the menu text. You can change this, so each menu item takes up the same width.

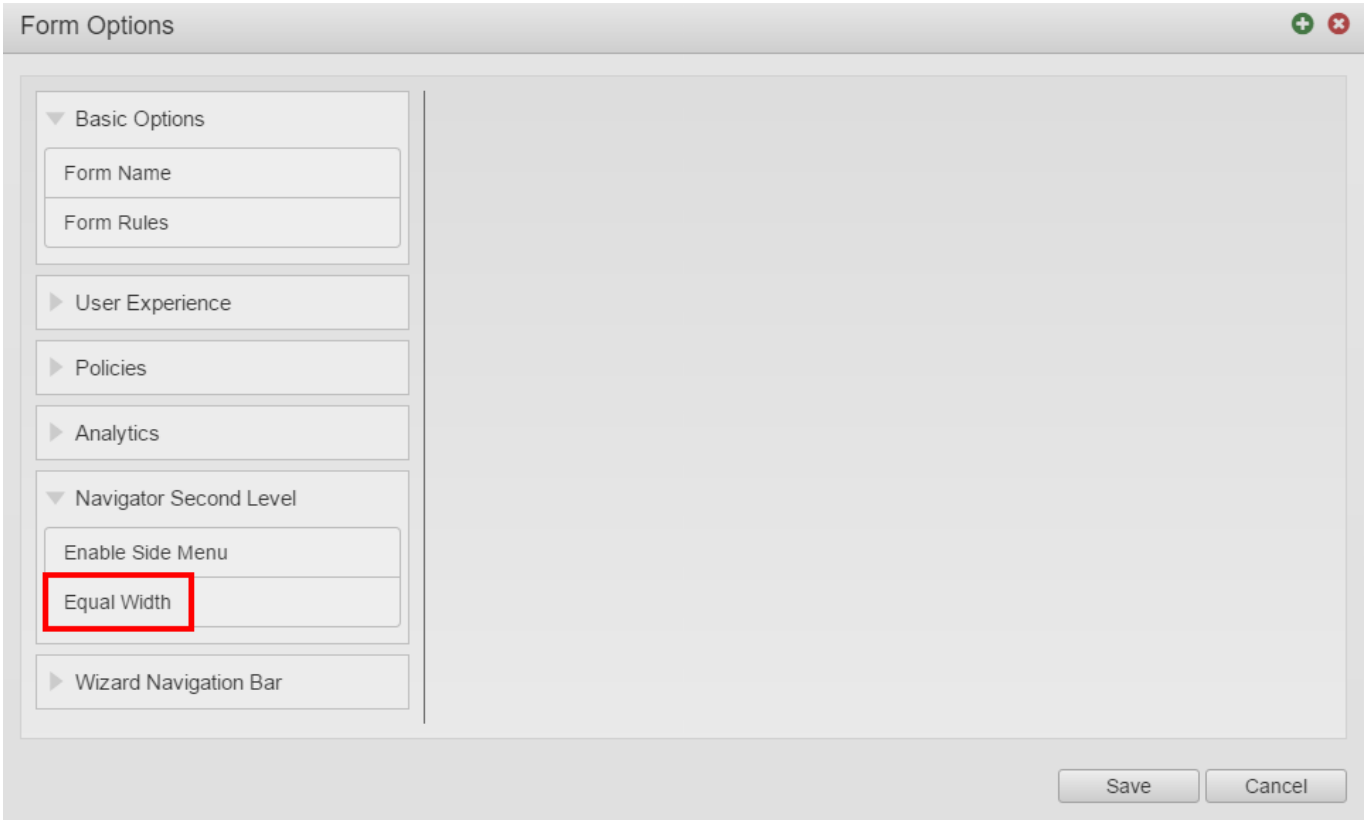
Click the Form Options button (towards the top of the Design window).



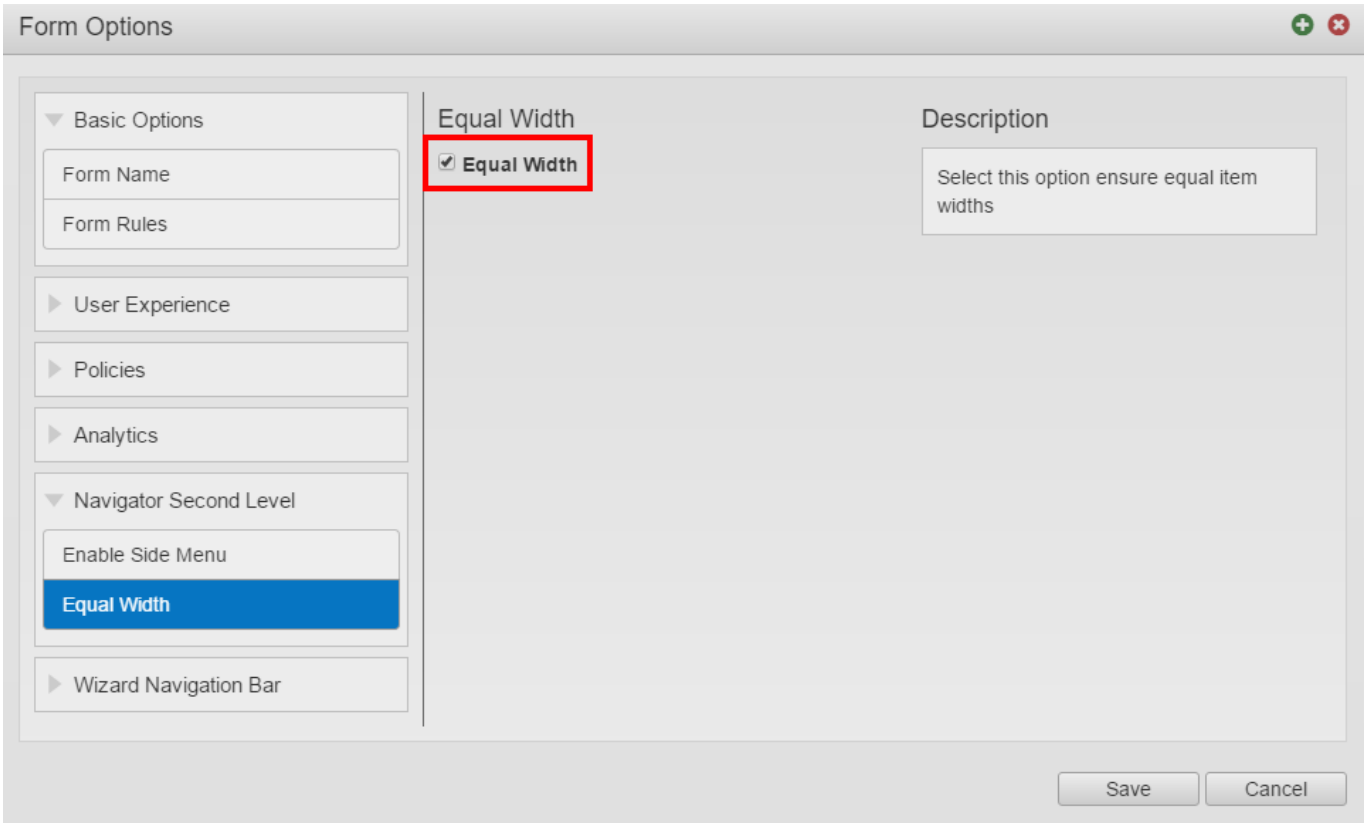
Expand Navigator Second Level.



Select Equal Width.



Check Equal Width, then click Save.


Form Options
Basic Options
Form Name
Form Rules
User Experience
Policies
Analytics
Navigator Second Level
Enable Side Menu
Equal Width
Wizard Navigation Bar
Equal Width
Equal Width
Description
Select this option ensure equal item widths
Save
Cancel

Validation Rules Exercise v5.1

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating validation rules. During this exercise you will create a new form and add components. You will then create rules on the added components.

Do your best to follow the information provided and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

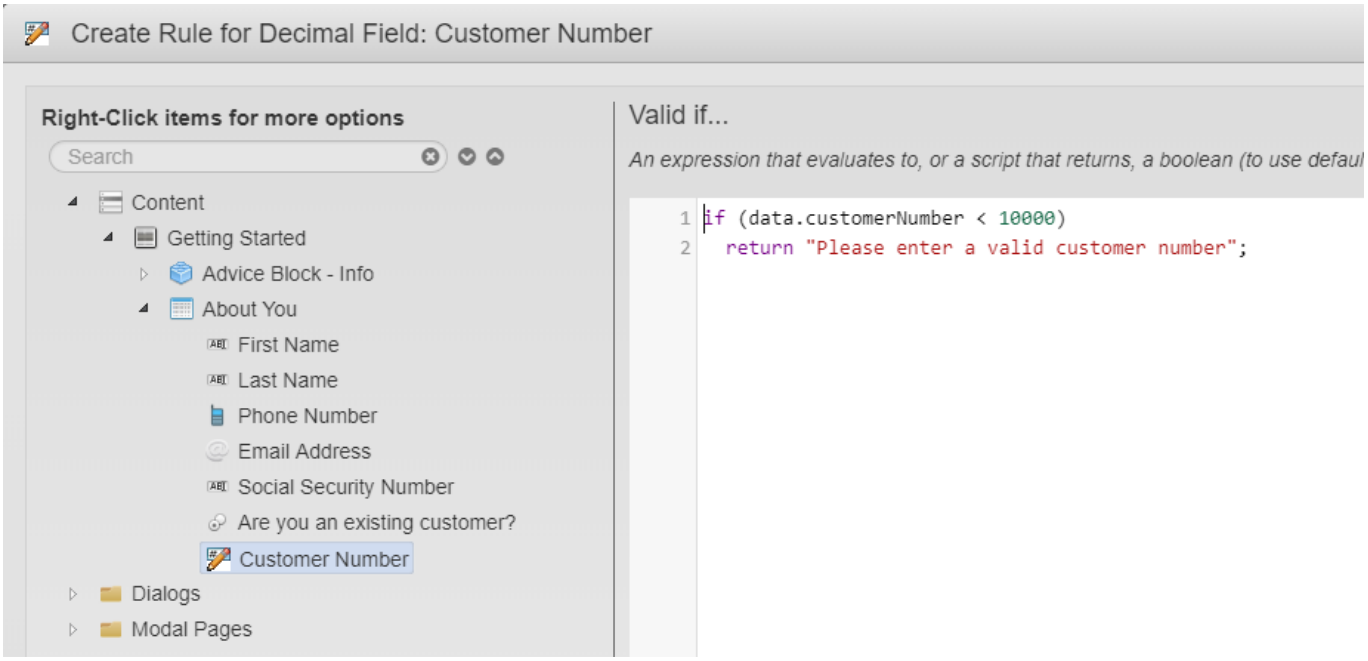
You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises v5.1 > Validation Rules Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Open Validation Rules Begin

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Double-click **Validation Rules Begin**.
5. Click **Edit Version** to open the form.

Step 2 Create a Valid If Rule

Use the Valid If rule for the Customer Number. Create a rule where the Customer Number is valid if greater than 10000.



Customer Number contains a visibility rule. You won't see this field in preview until you select Yes for the question 'Are you an existing customer?'

Step 3 Create a Regular Expression Validation Rule

You can search the Regular Expression Library (http://regexlib.com) to locate predefined expressions to use in your Maestro form.

Setup a Regular Expression rule to validate the Social Security Number component. An example expression is included in the image below.



Example Regular expression: `^(?!219-09-9999|078-05-1120)(?!666|000|9\d{2})\d{3}-(?!00)\d{2}-(?!0{4})\d{4}$`

Example Match Fails message: Please enter a number using this format ###-##-####

Once the validation rule is created, click **Save** and **Preview** the form.

Try entering this number first: 123-4-5678

Then try this number: 123-45-6789

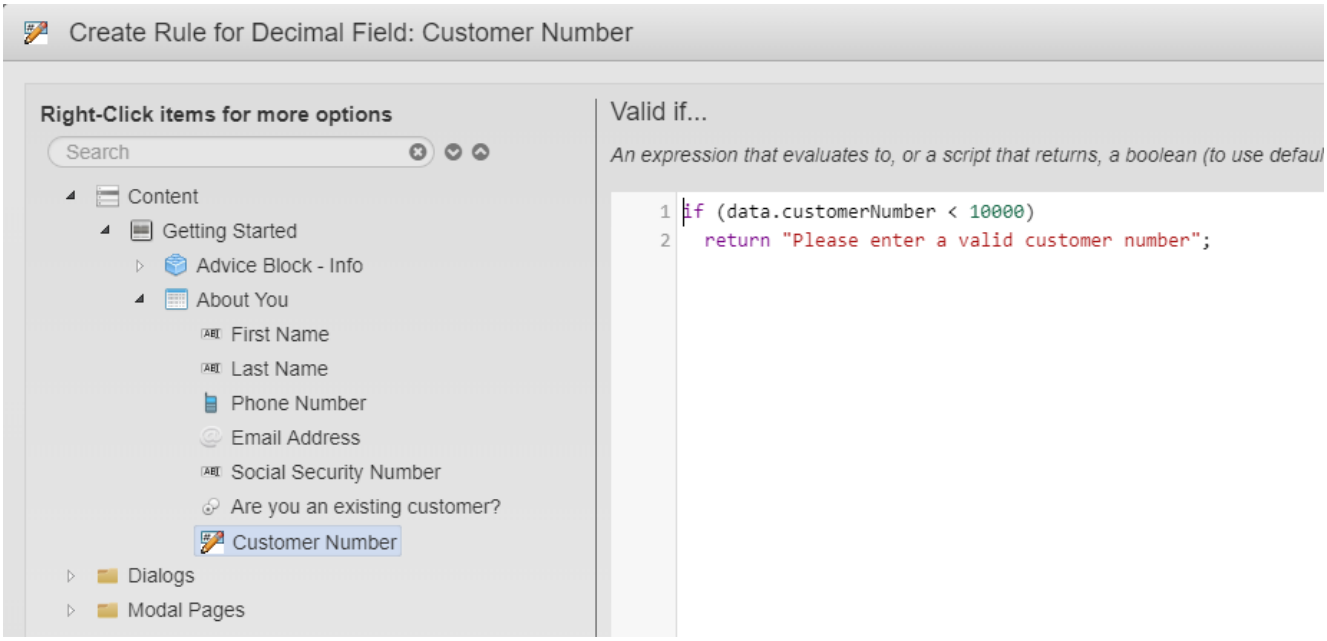
Extra Help

Need some extra help?

Use the information below to assist you in completing the exercise steps.

Create a Valid if rule

1. Select the component to contain the validation rule.
2. Click **Create Rule** then select **Valid if**.
3. Type `if (`.
4. Double-click the (Customer Number) component to add `data.customerNumber` to the script window.
5. Type `< 10000)` then press Enter.
6. Type `return "Please enter a valid Customer Number.";`.



7. Click **Save**.

Create a Regular Expression rule

1. Launch the Regular Expression Library - http://regexlib.com
2. Locate the pre-defined expression you want to use.
3. Copy the expression, then return to the Maestro form.
4. Select the component to contain the validation rule.
5. Click **Create Rule** then select **Regular Expression**.
6. Paste the pre-defined expression in the **Must Match this expression** box.
7. Enter an appropriate failure message.


Create Rule for Text Field: Social Security Number
Regular expression...
This item will be valid if it matches the provided regular expression
Must match this expression
^(?!219-09-9999|078-05-1120)(?!666|000|9\d{2})\d{3}-(?!00)\d{2}-(?!0{4})\d{4}$
Expression Flags
Ignore case
Multi-line
Global
If the match fails, display this message
Please enter a number using this format ###-##-####
Save
Cancel

8. Click **Save**.

eLearning Exercises for Maestro v5.0

Journey Platform | Platform Developer | Form Builder | v5.1 & Higher

View the eLearning courses available from the Temenos Learning Center.

We recommend that you view each course in order, following the listed order of the modules. After a few modules you will be given an exercise which will allow you to practice what you have learned. We strongly recommend you use these exercises to complete your training.

Pre-requisite

- First you will need to import the exercise files used for the Maestro 5.0 exercises. Refer to Exercise Setup - Importing the Maestro Exercise Files.

eLearning Exercises for Maestro v5.0

Use the following links to view the exercise guides. Please follow the instructions in each guide to identify the resources and assets you'll need to use.

Introduction to Maestro:

- Create a New Form
- Pages and Sections

Form Builder 1:

- Layout
- Creating Rules (without Scripting)
- Repeating and Shared Components

Form Builder 2:

- Creating Shared Components
- Two Level Navigation

Form Builder Scripting:

- Calculation Rules
- Creating Rules
- Validation Rules

Help

For help using the Learning Center refer to the Learning Center Help Guide.

If you have any questions, issues or feedback please feel free to contact the Manly Knowledge Team.

Exercise Setup - Importing the Maestro v5.0 Exercise Files

Journey Platform | Platform Developer | Form Builder | v5.1 & Higher

This guide explains how to import the Maestro exercise files as a Maestro project. This project contains all the required forms, templates, components and styles needed to complete the exercises.

You can also access a training module on the Avoka LMS that explains how to import and export in Maestro. You may want to view this module before importing the exercise project.

Step 1 Identify the Training Environment

1. Identify which of the following best describes the Training Environment you will use to complete the Maestro 5.0 training.

Training Environment	Uses Maestro version	Download…	Import Type
elearning.avoka.com	17.10 or later	Version 2018.08.02	Import Project
Your company	17.10 or later	Version 2018.08.02	Import Project
Your company	5.x or earlier	Version 2016.12.22	Import Project

+ Version 2016.12.22: was exported from Maestro 5.0.x environment
+ Version 2018.08.02: was exported from Maestro 17.10 (or later)

Step 2 Download the Exercise Project

The file to be downloaded is a zipped folder that contains the Maestro training exercise project.

1. Click on the required download link above to download the Exercise Project to your PC.
2. Leave this file in the **Downloads** folder or move it to a different location. Just remember, you'll need to navigate to this file in the next step to import it into Maestro.

Step 3 Import the Exercise Project

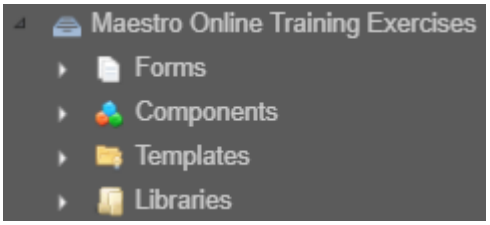
Once you've selected the Organization, you can import a project into Maestro.

1. Login to Maestro.
2. Click on your **Organization**.
3. Click **Import Project** to import the Maestro Project.
4. Enter **Maestro Online Training Exercises** for the name of the project you're importing.
5. Click **Import**.

6. Navigate to and select the Maestro Exercise Project zipped folder you downloaded.
7. Click **Open**.

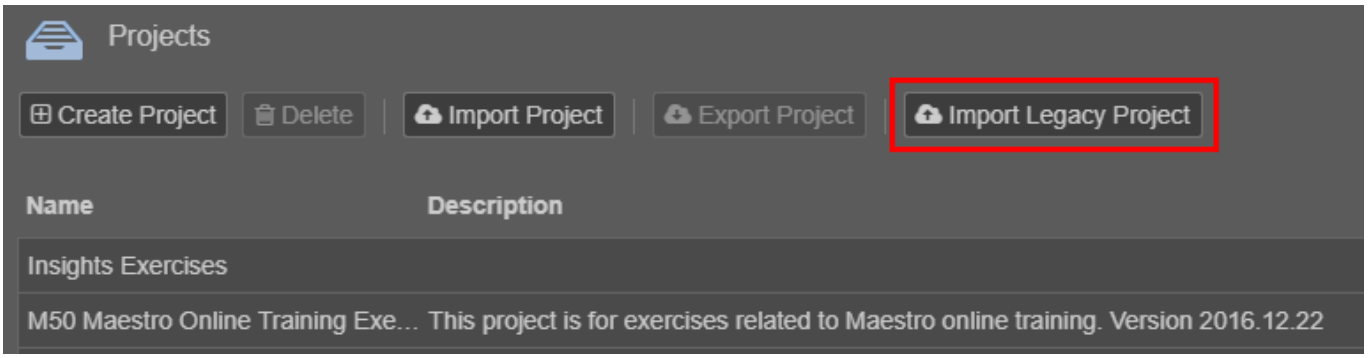
Result

After importing the project you'll have access to the exercise forms, templates and components.



Optional Step 4 Importing Legacy Projects

Maestro offers an additional import option for projects created and exported from legacy (or previous) versions of Maestro.



A project is considered Legacy if it was created and exported from any version of Maestro prior to v17.10. The option to Import Legacy Projects is only available to Maestro users assigned the **Maestro Site Administrator** Role. If you need to import a legacy project, speak to your Maestro account manager.

Calculation Rules Exercise v5.0

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating rules. During this exercise you will create a new form and add components. You will then create rules on the added components.

Do your best to follow the information provided and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find **Extra Help** at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the **Maestro Exercise Project: Maestro Training Exercises > Calculation Rules Finish**. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

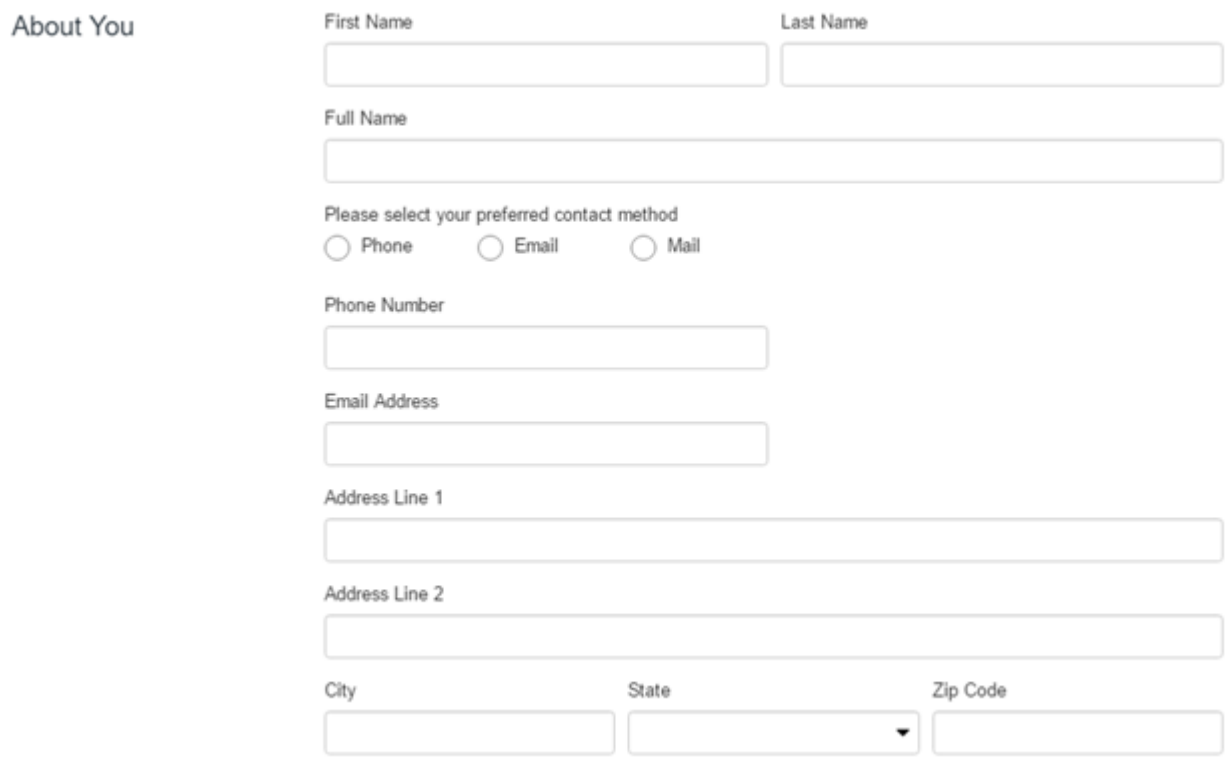
Step 1 Create a New Form

Create a new form using the details provided below.

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Click **Create Form**.
5. Enter details about the form:
   - Name: Order Form
   - New Version: 1.0-develop
     (should be pre-populated if Default Initial Version has been set up in Project Details)
   - Description: This form is to practice creating calculation rules.
   - Template: Exercise Template
6. Click **Create**.

Step 2 Add Components to Page One

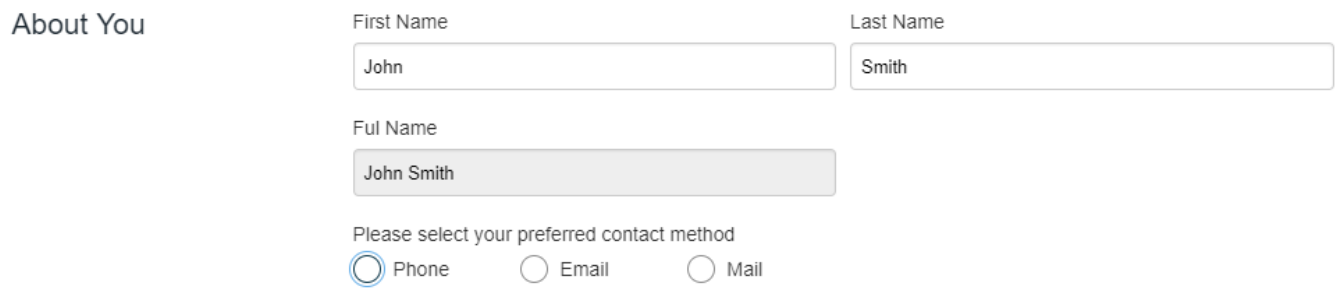
Add components to Page One so your form resembles the image below.



Click **Save**.

Step 3 Populate a Component Based on Entered Data

Populate the **Full Name** component based on a script that concatenates the data entered in **First Name** and **Last Name**.



Add a **Read Only** rule to **Full Name** so the populated data cannot be changed.

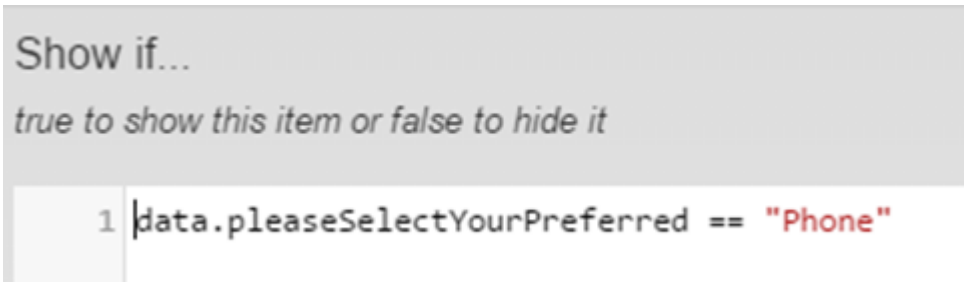
Preview the form to check the calculation rules.

Click **Save**.

Step 4 Add Show if Rules

Add **Show If** rules to the **Phone Number**, **Email Address** and **Address block** based on the preferred contact method selected.

Right-click the appropriate components on the left side of the **Create Rule** window to create the required **Show if** scripts (see the example below).

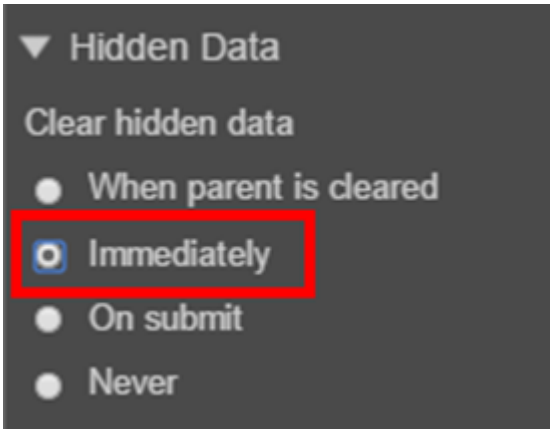


Save your form.

Preview the form to check the calculation rules.

Step 5 Change the Clear Hidden Data Option

Change the **Clear hidden data** option for each of the contact detail components so that when they are hidden, any data previously entered in the field is cleared immediately.

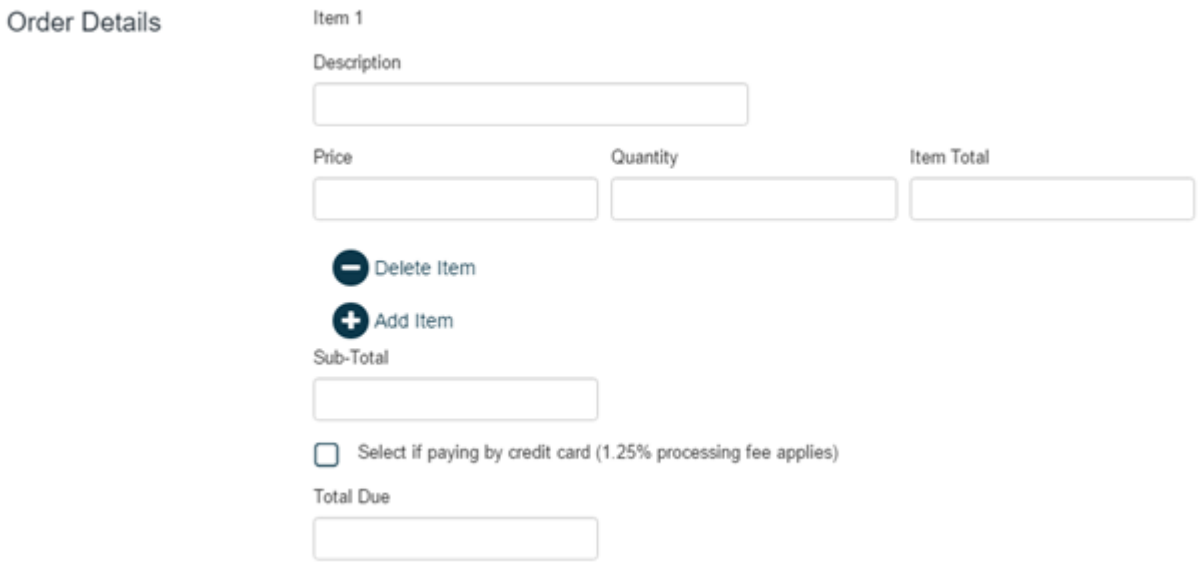


When you've changed the Hidden Data options, **Save** your form then click **Preview** to test the form.

Click **Save**.

Step 6 Add an Order Details Section to the Form

Add a new section below the **About You** section. Add components so the new section resembles the image below.

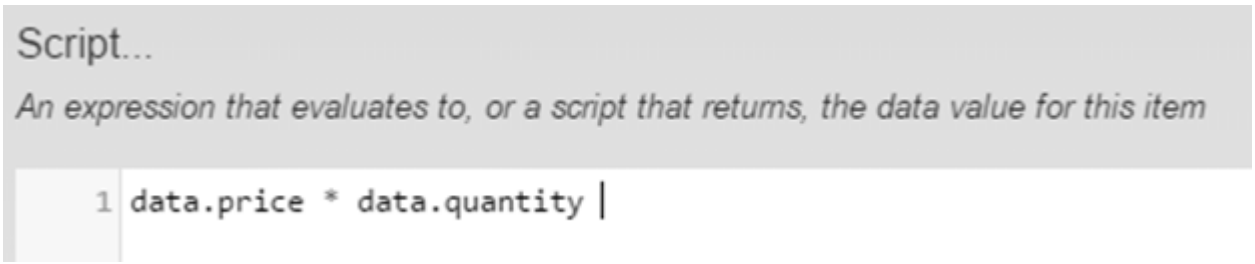


Use the **Decimal Field** component for **Price**, **Item Total**, **Sub-Total** and **Total Due** and set the decimal fields to 2 decimal places.

Click **Save**.

Step 7 Add a Calculation Rule to the Item Total

Create a calculation rule that multiples **Price** by **Quantity**.



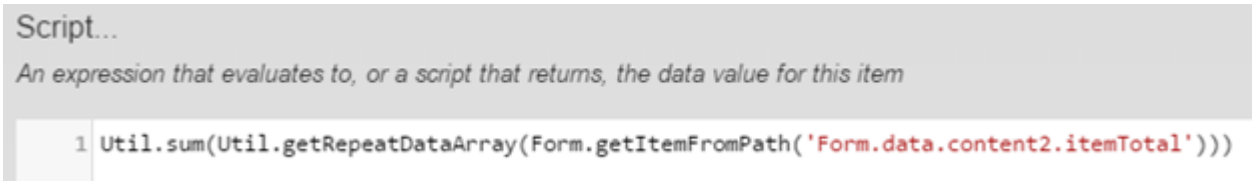
Add a **Read Only** rule to **Item Total** so the user cannot change the calculated value.

Click **Save**.

Preview the form to check the calculation rules.

Step 8: Add a Calculation Rule to the Sub-Total Component

Create a calculation rule that sums the **Item Totals** for each repeat.



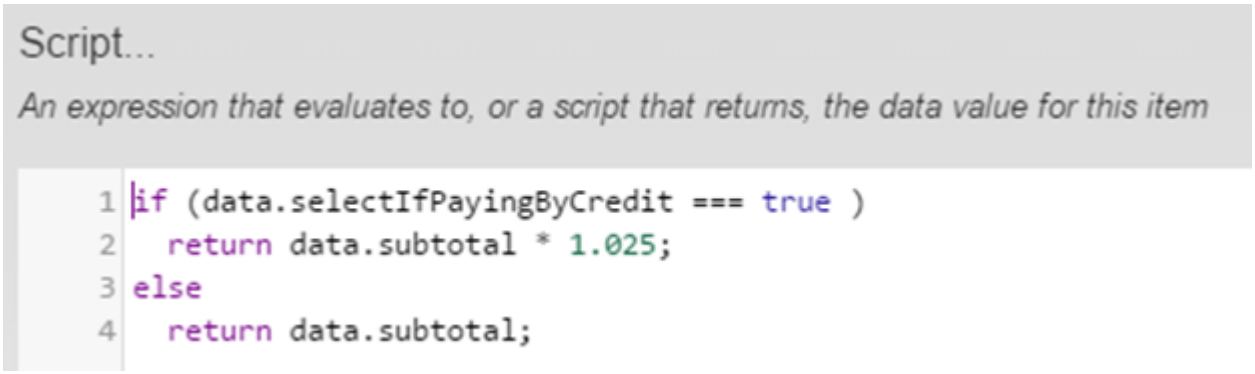
Add a **Read Only** rule so that the user cannot change the calculated value.

Click **Save**.

Preview the form to check the calculation rules.

Step 9: Add a Calculation Rule to the Total Due Component

Create a calculation rule that includes the **Sub-Total** multiplied by the credit card processing fee if the checkbox is selected.



Add a **Read Only** rule so that the user cannot change the calculated value.

Click **Save**.

Preview the form and check the calculation rules.

Extra Help

Need some extra help?

Use the information below to assist you in completing the exercise steps.

Create a Show If Rule

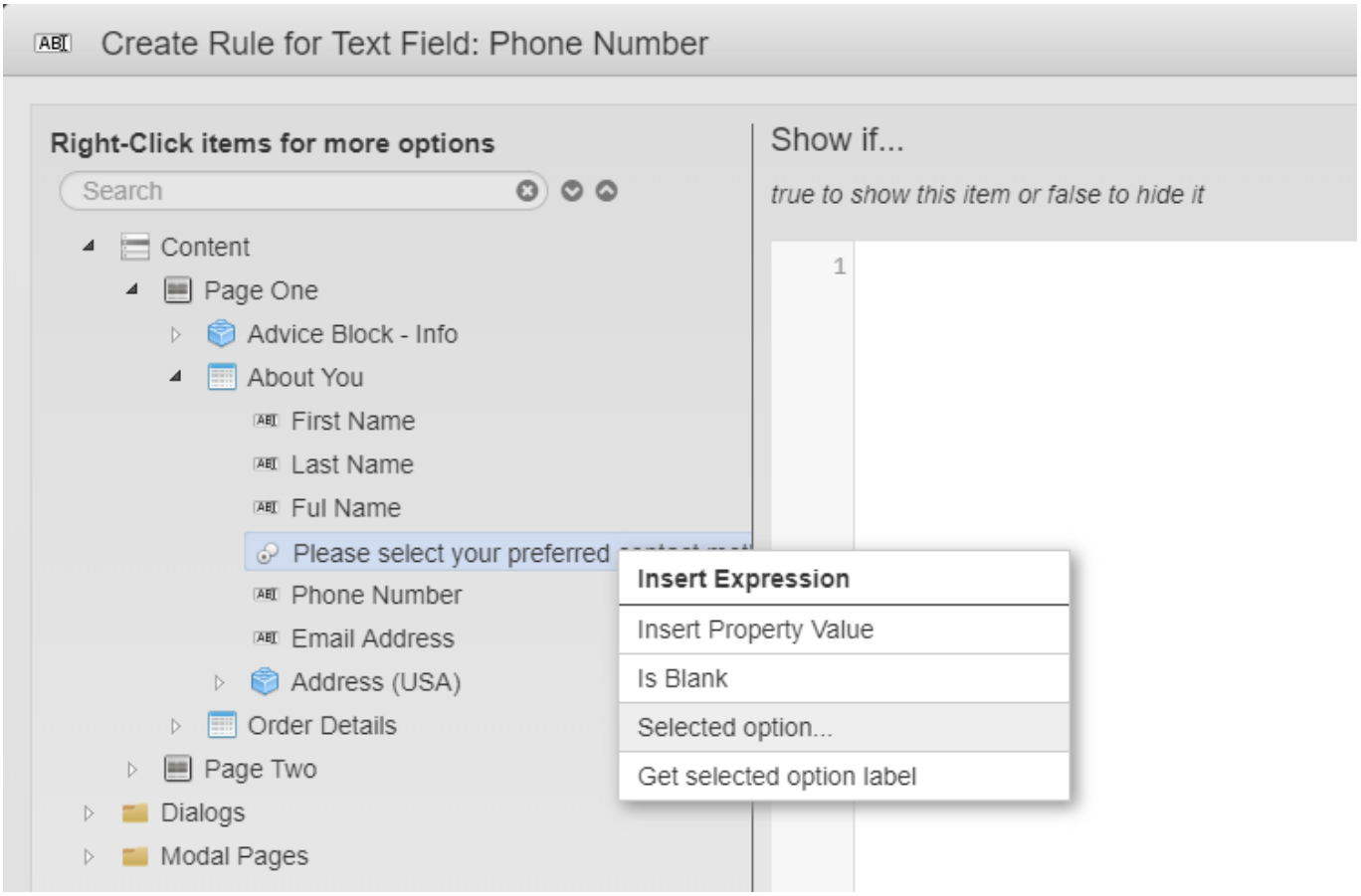
Select the component whose visibility is conditional (i.e. visible or not visible).

Click **Create Rule** from the **Rules** section (**Properties** panel).

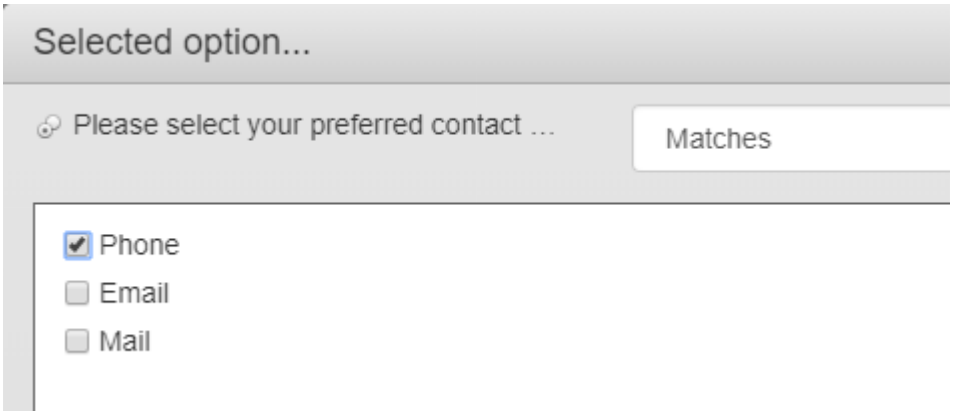
Select **Show If** from the **Create Rule** window.

Right-click the component that determines if a component is shown on the form

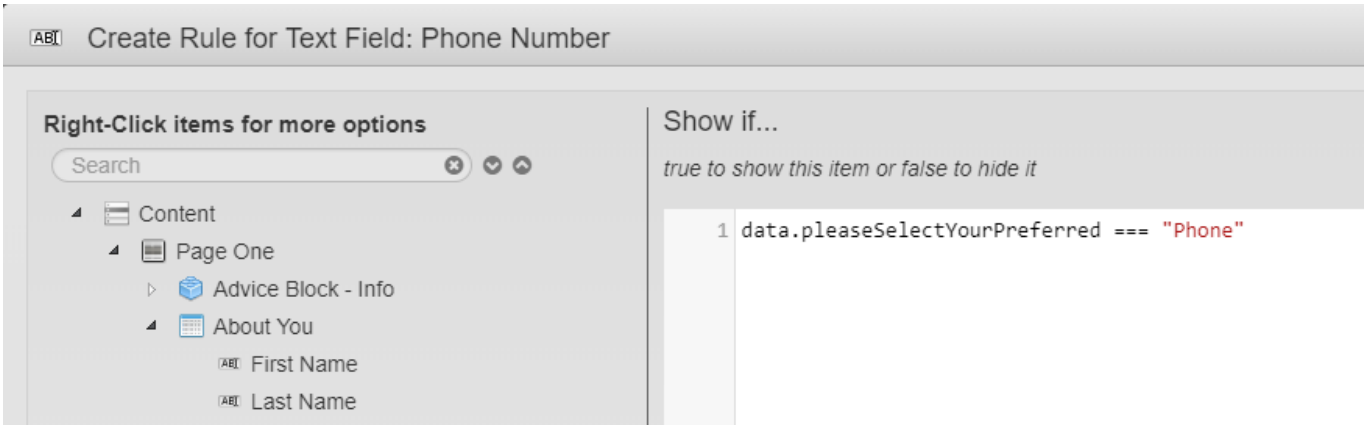
Choose **Selected option**.



Nominate the matching Option (e.g. Phone) and click **OK.**



The script is added to the **Create Rule** window.



Combine entered data into a component

Select the component where the data will be populated.

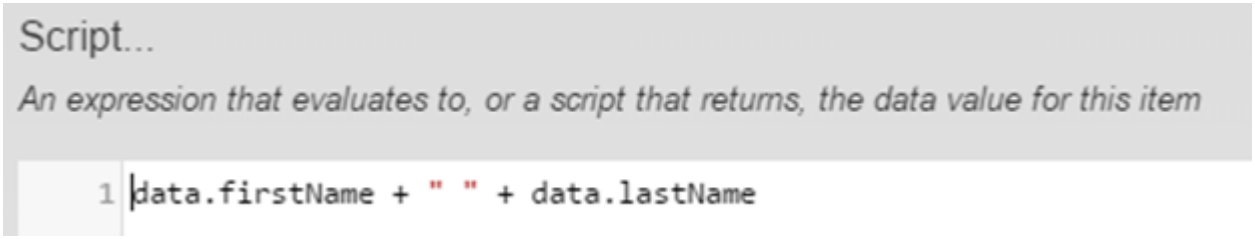
Click **Create Rule** and select **Script**.

Double-click the component you want to add to the script, e.g. double-click **First Name** to add **data.firstName** to the script window.

Enter a plus sign between the components you want to combine.

Any spaces or other characters to include in the script must be wrapped between quotation marks (“).

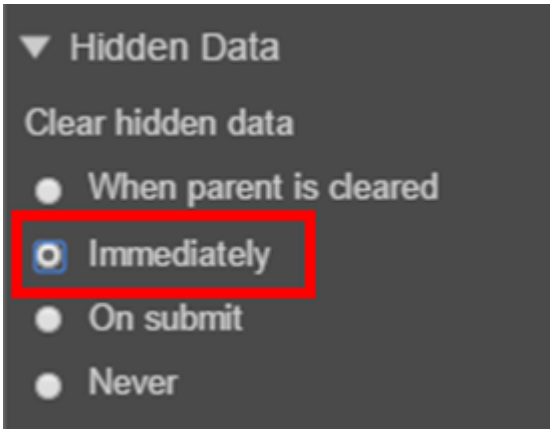
Double-click the next component to add to the script (see example below).



Clear Hidden Data Immediately

Select the component.

Select the **Data** tab, expand **Hidden Data** and choose **Immediately** from the options.



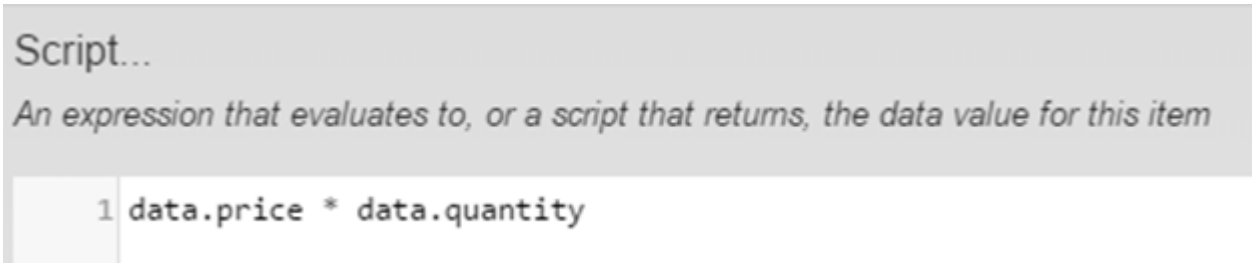
Create a Mathematical Calculation Rule

Select the component that will contain the calculation.

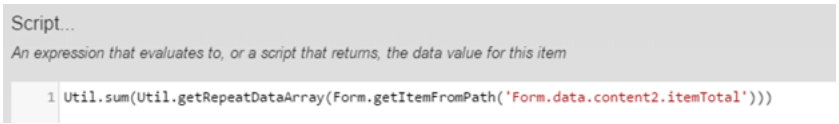
Click **Create Rule** then select **Script**.

Double-click any components on the left-side of the window to add them to the script window (on the right).

Insert the required calculation symbols, e.g. * / + -



The script is added to the window.



Create a Calculation Rule for Repeated Items

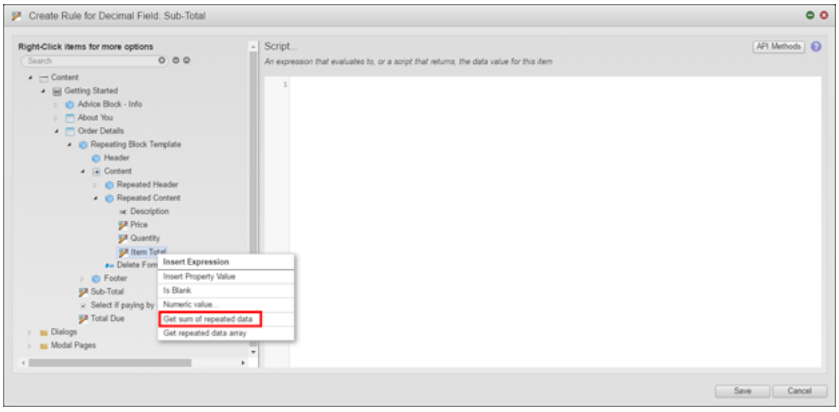
Select the component that will contain the calculation.

Click **Create Rule** then select **Script**.

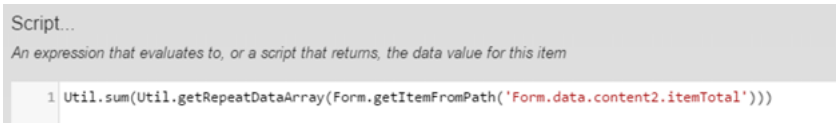
Expand the **Repeating Block Template**, then expand **Content**.

Expand **Repeated Content**.

Right-click the component you want to total and select **Get sum of repeated data**.



The script is added to the window.



Create a New Form Exercise v5.0

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating new forms in Maestro. During this exercise, you will create a new form and add components.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises > Create New Form Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Create a New Form

Create a new form using the details provided below.

1. In the Navigator Panel, expand the **Organization**.

2. Expand the **Project**.

3. Select **Forms**.

4. Click **Create Form**.

5. Enter the following details for your new form:

- Name: Create New Form_[Your initials]
- New Version: 1.0-develop

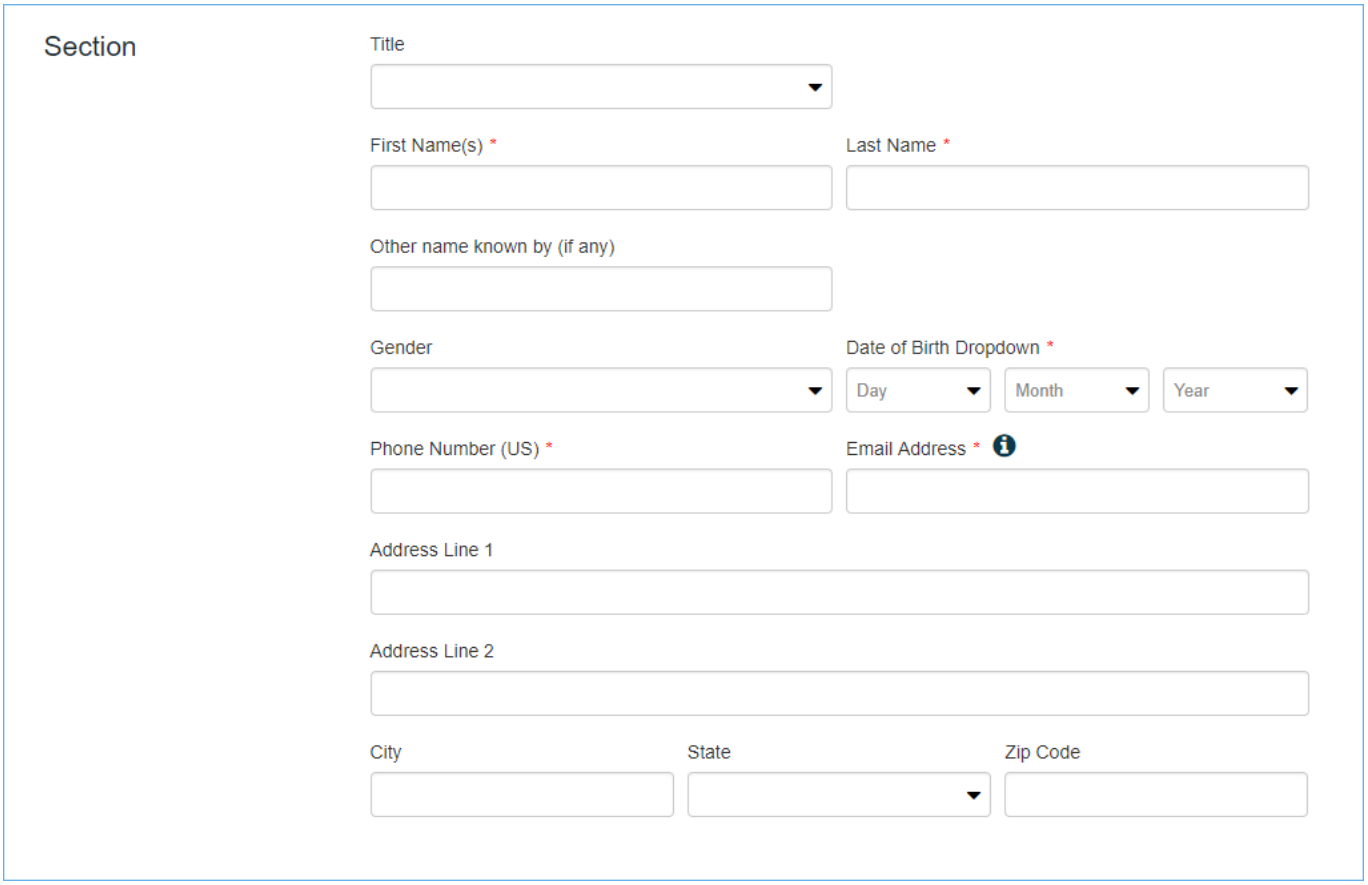
  (should be pre-populated if Default Initial Version has been setup in Project Details)

- Description: This form is to practice creating new forms and adding components.
- Template: Exercise Template

6. Click **Create**.

Step 2 Add Components to Page One

On Page One, you will find an empty **Section**. You can add (drag) components into this area. Use the image below as a guide for what you need to add.



You can **Preview** your form at any time to see how the components look and work.

The components marked as mandatory (*) and with the information icon (i).

When adding components, it may be helpful to display the Palette and Structure panel together. Click the Side-by-Side button in the top left corner of the Editor window.

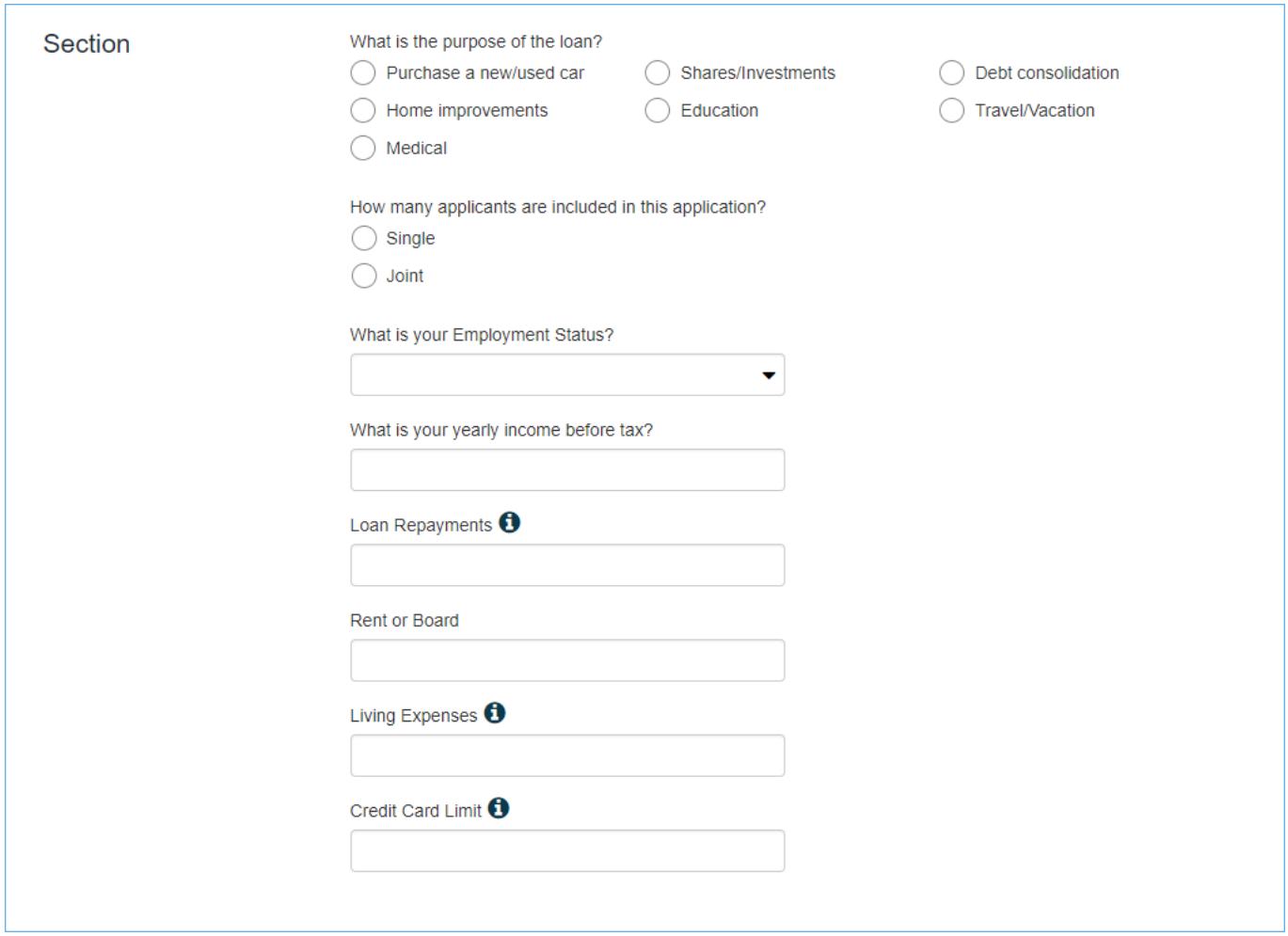
Remember: to use predefined components, rather than recreating your own. You can use Search in the Palette panel to find the required components, for example, the 'Title' component, which contains all the common titles (Mr, Mrs. etc.). The default options can be updated as required and is much easier than building a dropdown from scratch.

Use this for the **Email Address Help Tex**t: We use your email to send confirmation of the information you've provided.

Click **Save** on the toolbar.

Step 3 Add Components to Page Two

On Page Two, you can add (drag) components into the available section. Use the image below as a guide for what you need to add.



Some components have the information icon (i) next to the label.

Use the following entries for the **What is the purpose of the loan** radio buttons:

- Purchase a new/used car
- Shares/Investments
- Debt consolidation
- Home improvements
- Education
- Travel/Vacation
- Medical

Use the following entries for the **How many applicants are included in this application?** radio buttons:

- Single
- Joint

Use the following entries for the **What is your Employment Status?** dropdown:

- Employed
- Self-Employed
- Unemployed

Use the following details for the **Help Text**:

- Loan Repayments: Please include any mortgage repayments, personal loans or car loans.
- Living Expenses: Consider how much you spend on bills, groceries, transport and entertainment, etc.
- Credit Card Limit: Please include the total limit of all your credit cards.

Use the following for the expense questions:

- What is your Employment Status?
- What is your yearly income before tax?
- Loan Repayments
- Rent or Board
- Living Expenses
- Credit Card Limit

Step 4 Save the Form

Click **Save** on the toolbar to save all the changes to your form.

Extra Help

Need some extra help?

Use the information below to assist you in completing the exercise steps.

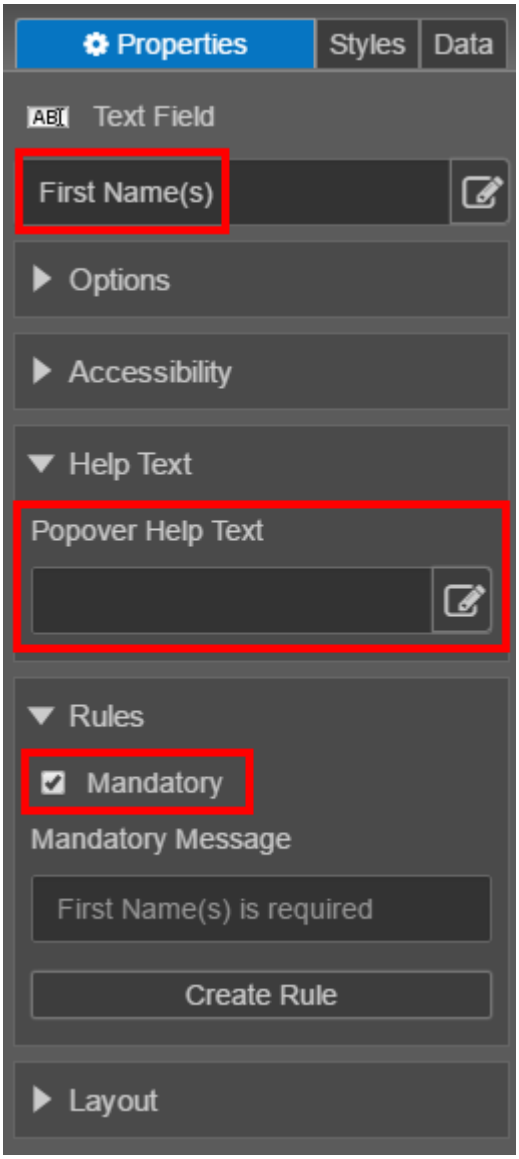
Component Properties

You can use the **Properties** panel to change the following settings:

- Component Label
- Add Help Text
- Set Component as Mandatory
- Radio Button Arrangement
- Radio Button Options
- Dropdown Options

The **Properties** panel displays the settings for the selected component.

For example, properties for the First Name(s) Text Field are displayed below.



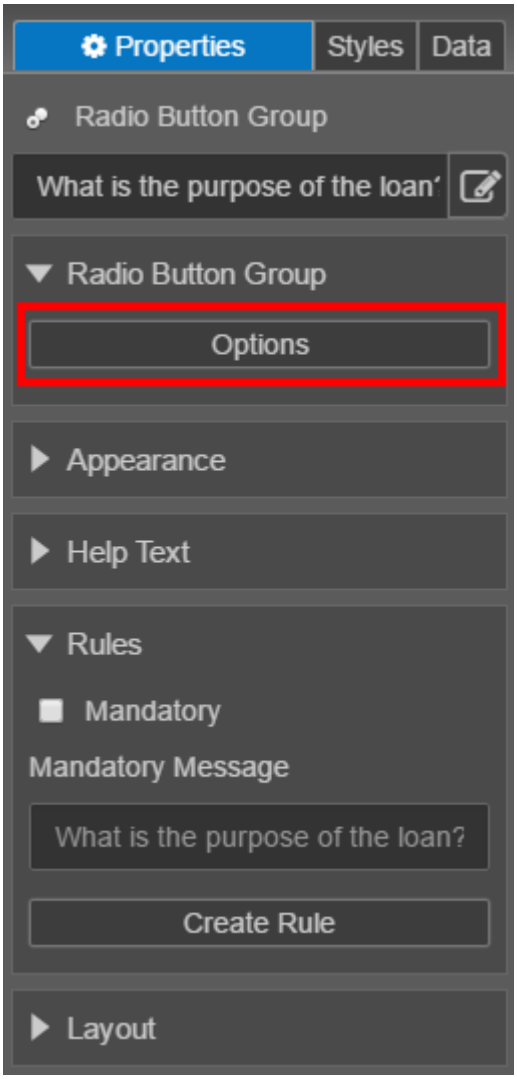
At the top, you can change the component label.

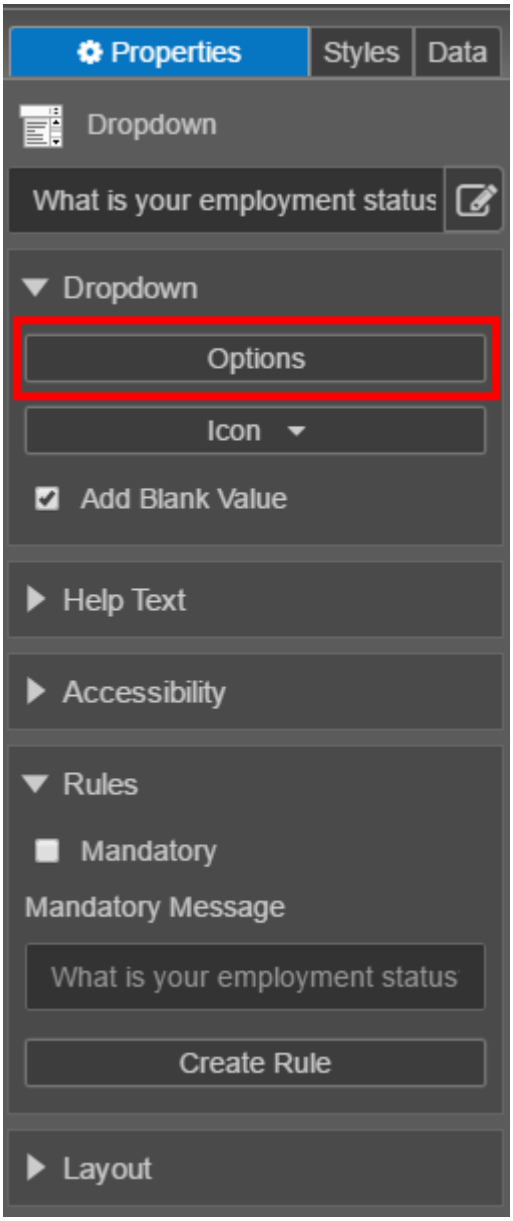
Popover Help Text is where you add the help text, which displays the information icon next to the component label.

The **Mandatory** option is under **Rules**. Select this to mark a component as mandatory. This will automatically add the red asterisk next to the label.

Radio Button and Dropdown Options

Use the **Properties** panel to customize the options available for **Radio Button Group** and **Dropdown**.



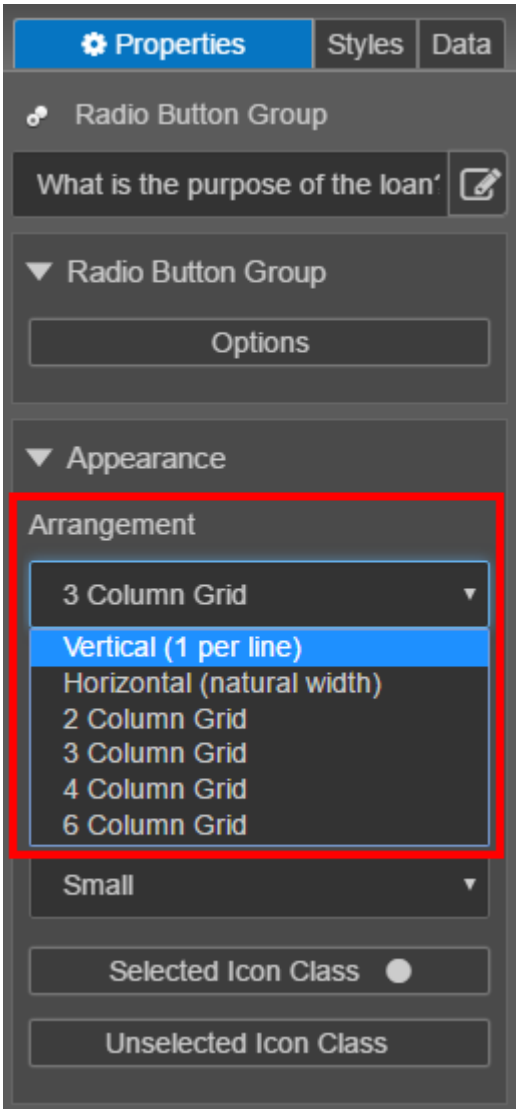


Click the **Options** button to open a new window, where you can add or edit the radio buttons or dropdown options.

Arrangement of Radio Button Options

You can change the appearance of the radio buttons, so they're stacked vertically or displayed in a horizontal grid.

To change the appearance, select the required **Arrangement**.


Properties
Styles
Data
Radio Button Group
What is the purpose of the loan'
Radio Button Group
Options
Appearance
Arrangement
3 Column Grid
Vertical (1 per line)
Horizontal (natural width)
2 Column Grid
3 Column Grid
4 Column Grid
6 Column Grid
Small
Selected Icon Class
Unselected Icon Class

Creating Rules Exercise v5.0

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating rules. During this exercise you will create a new form and add components. You will then create rules on the added components.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises > Creating Rules Finished. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

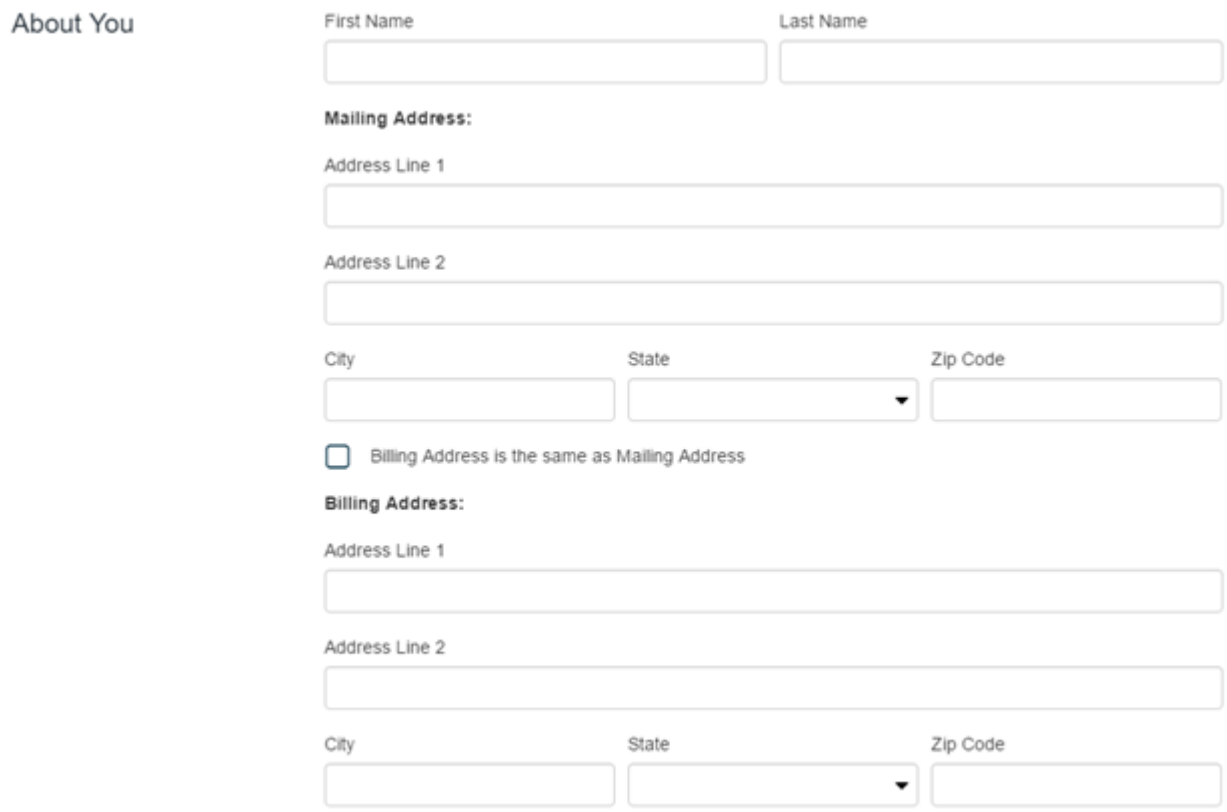
Step 1 Create a New Form

Create a new form using the details provided below.

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Click **Create Form**.
5. Enter details about the form:
   - Name: Mailing Address
   - New Version: 1.0-develop
     (should be pre-populated if Default Initial Version has been setup in Project Details)
   - Description: This form is to practice creating component rules.
   - Template: Exercise Template
6. Click **Create**.

Step 2 Add Components to Page One

On Page One you will find an empty section. Add new components to this area using the image below as a guide.



When adding components, it may be helpful to display the Palette and Structure panel together. Click the Side-by-Side button in the top left corner of the Editor window.

Remember to use predefined components (where appropriate), rather than recreating your own. You can use the search in the **Palette** panel to find components. For example, use the Address (USA) component rather than adding individual components. This component contains all the necessary fields, which you can customize if needed.

Step 3 Add a Rule to Billing Address

Add an **Editable if** rule to the Billing Address block that allows it to be editable if the checkbox is not selected.

You can double-click components on the left side of the Create Rule dialog to add them to the script window or type directly into the script window.

Editable if...

true if this item should be editable or false if this item should be locked

```
1 data.billingAddressIsTheSame !== true
```

Step 4 Add a Rule to the Checkbox

Add a **Change** rule to the checkbox that copies the entered values from Mailing Address to Billing Address if the checkbox is selected.

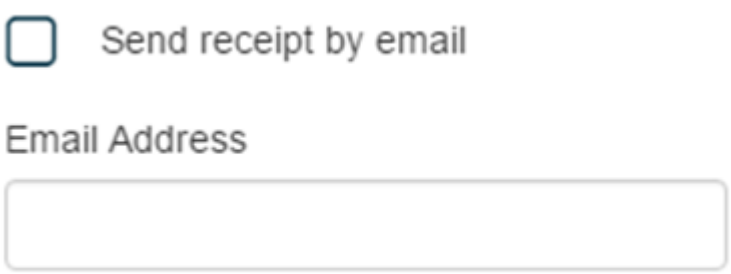
Change

A script to run when the user changes the data in this form item

```
1 if (data.billingAddressIsTheSame === true ) {
2   data.addressLine11 = data.addressLine1;
3   data.addressLine21 = data.addressLine2;
4   data.city1 = data.city;
5   data.state1 = data.state;
6   data.zipCode1 = data.zipCode;
7 }
```

Step 5 Add Components with a Mandatory If Rule

Add a checkbox and text field to the end of the form. Create a mandatory if rule on the text field that makes the field mandatory if the checkbox is selected.



Extra Help

Need some extra help?

Use the information below to assist you in completing the exercise steps.

Create an Editable If Rule

Select the required component from the **Structure** panel (or in the form).

Click the **Create Rule** button from the **Rules** section (**Properties** panel).

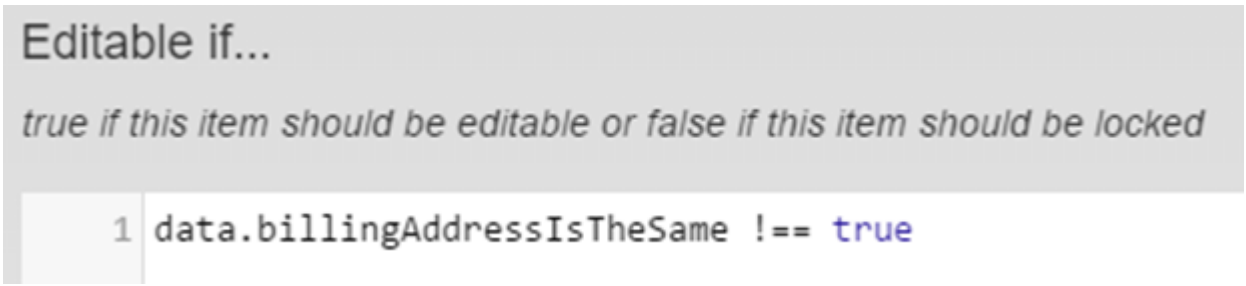
Select **Editable If**.

Right-click the checkbox component that determines when the address component is editable.



The options in the menu are determined by the component selected. The image (above) shows the options for a checkbox.

Based on the exercise requirements, you would reed to select the **Is not selected** option.



The script for this rule says the address block is editable if the checkbox is **not** selected.

Create a Change Rule

A Change rule will trigger when a component is changed.

To start, select the component you want to apply the rule to (i.e. the checkbox).

Click the **Create Rule** button from the **Rules** section (**Properties** panel).

Select **Change** under **Action**.

> You can double-click components on the left side of the Create Rule dialog to add them to the script window (right) or type directly into the script window.

Based on the exercise requirements, you will need to type at least some of the script. Enter the script as follows:

Type `if (`

Then right-click the checkbox and select **Is selected** to add `data.billingAddressIsTheSame === true`.

Then type `) {` and press **Enter**.

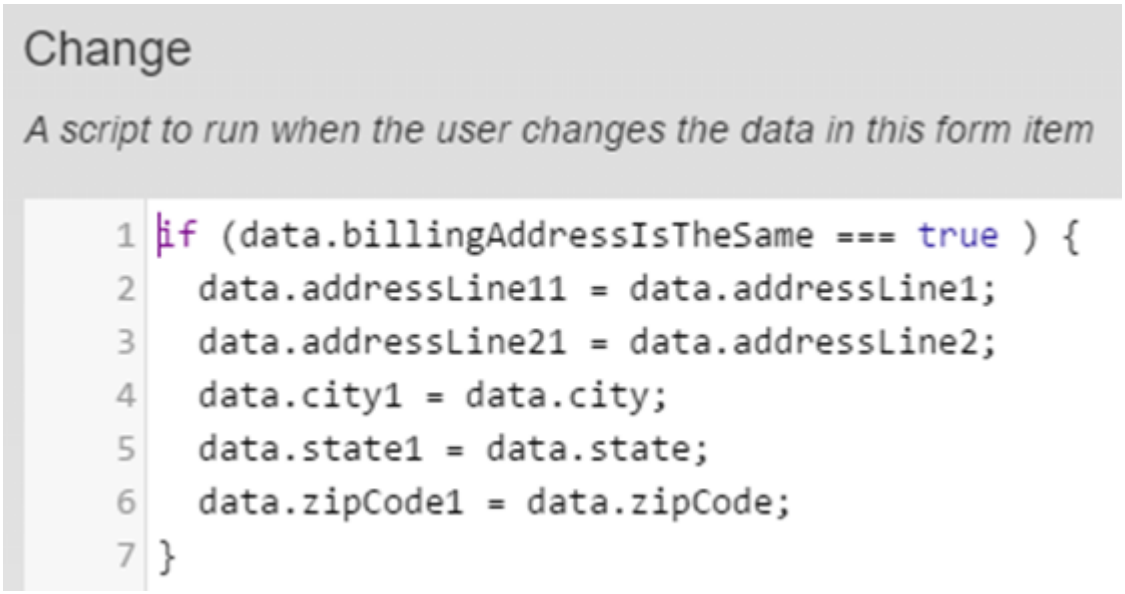
For the next line, double-click **Address Line 1** from the **Billing Address** block to add `data.addressLine11`.

Type **=** and a **space**.

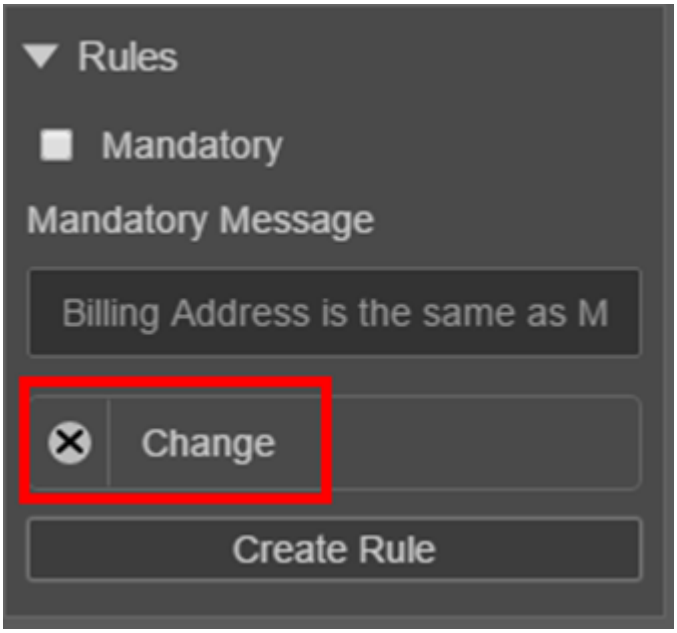
Double-click **Address Line 1** from the **Mailing Address** block to add `data.addressLine1`.

Type `;` (Semicolon)

Repeat this process for each of the address components. On the final line type `}` to end the script. See the image below for guidance.



Once you've finished creating the rule is, click **Save**.



The rule is added to the **Rules** section (**Properties** panel).

Create a Mandatory If Rule

Select the component to be mandatory based on specific conditions (i.e. Email Address).

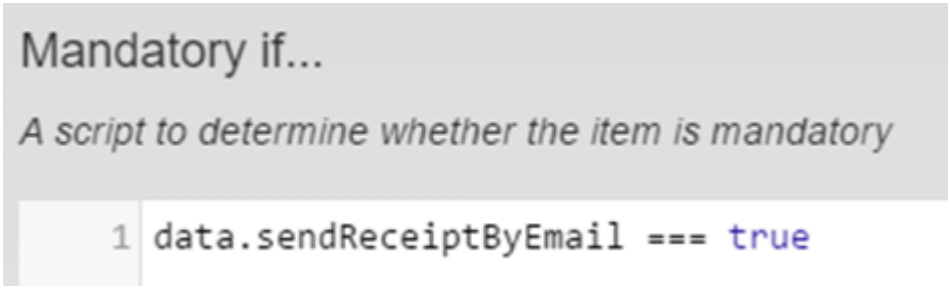
Click the **Create Rule** button from the **Rules** section (**Properties** panel).

Select **Mandatory If**.

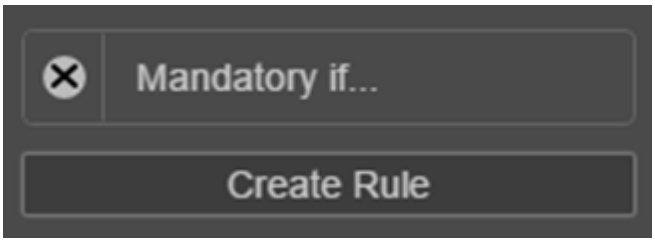


You can double-click components on the left side of the Create Rule dialog to add them to the script window (right) or type directly into the script window.

See the image below for guidance.



Click **Save**.



Creating Shared Components Exercise v5.0

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating shared components. During this exercise you will create a new shared component within a form.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find **Extra Help** at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the **Maestro Exercise Project: Maestro Training Exercises** > **Create Component Finish**. Please note that this form uses a shared component with a different name to the one suggested in the steps below. Shared components cannot have the same name as other shared components within the same project.

If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

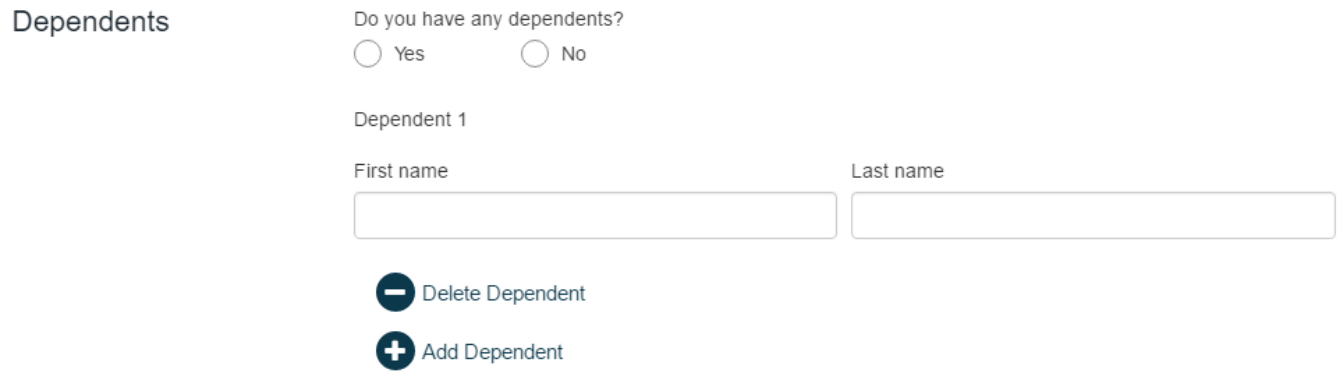
Step 1 Open Create Component Begin

1. Expand **Organization**.
2. Expand **Project**.
3. Select **Forms**.
4. Double-click **Create Component Begin**.
5. Click the **Edit Version** button to open the form

> You can create components directly from the Components folder, rather than from within a form.

Step 2 Setup a Repeating Block and Add Components

Add a **Repeating Block Template** and components to the **Dependents** section so it matches the image below.

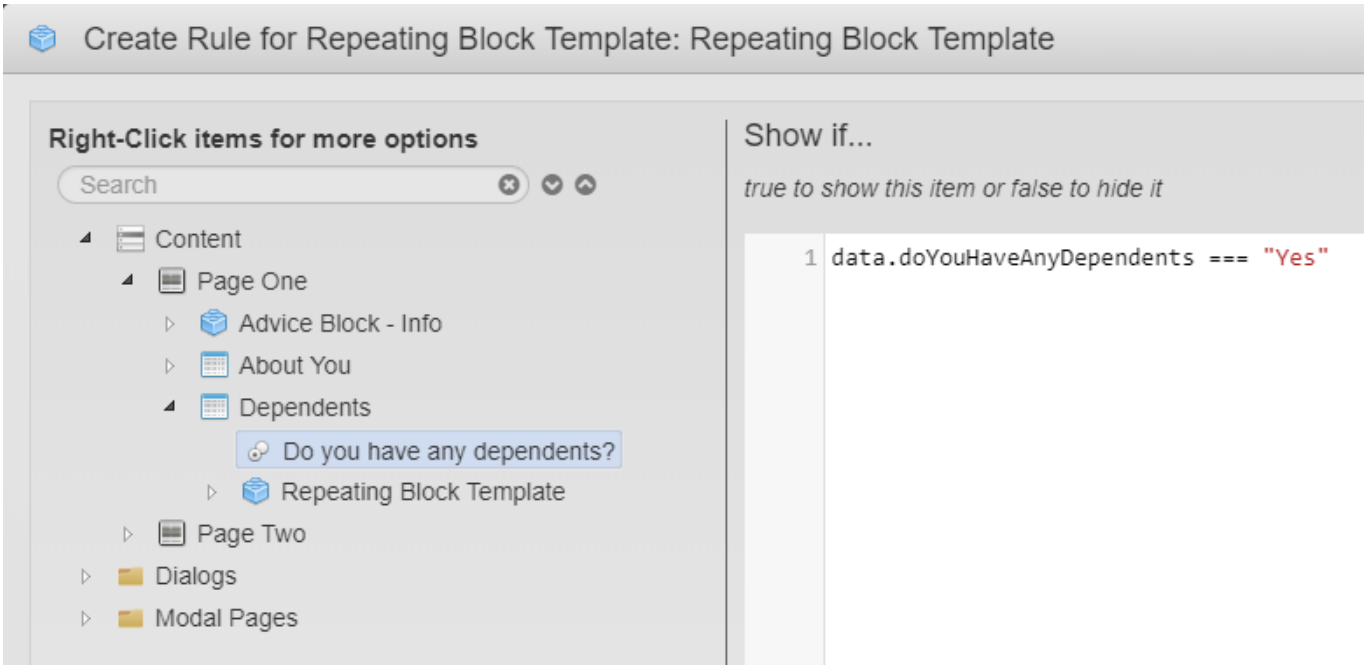


The First Name and Last Name components are within the Dependent 1 Repeating Block Template. Before you add components, make sure you add the Repeating Block Template first, then add the components.

Step 3 Create Visibility Rule

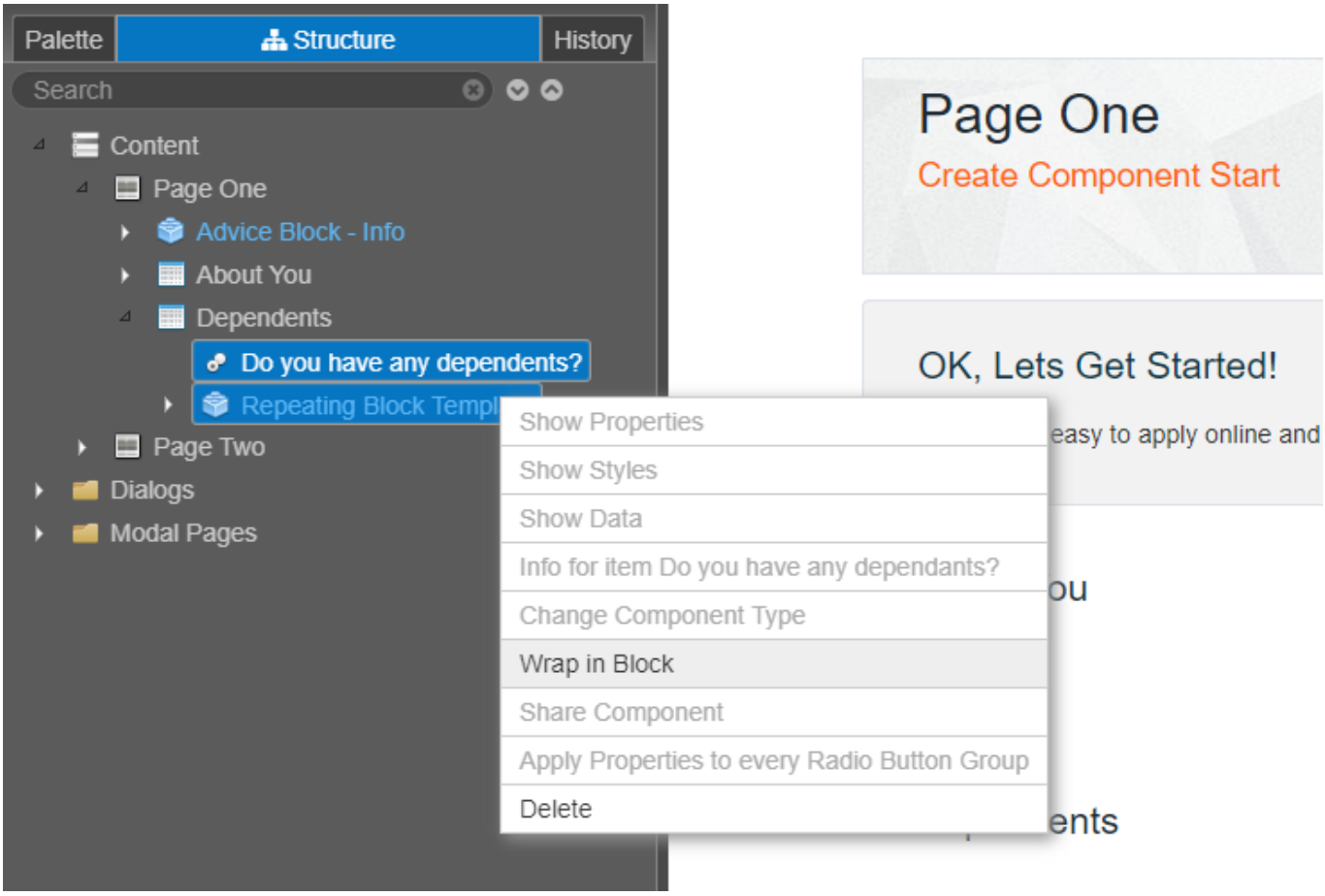
Create a visibility rule so that the repeating block does not show unless the user selects Yes to having dependents.

Create this rule on the **Repeating Block Template** (see below).

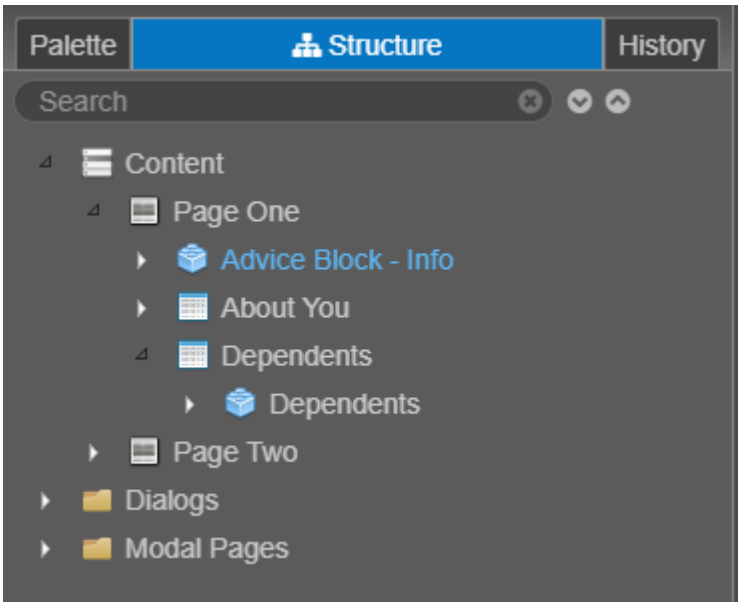


Step 4 Wrap in block

Wrap all the components (i.e. Radio Button component and Repeating Block) from the Dependents section into a block. See the image below for guidance.

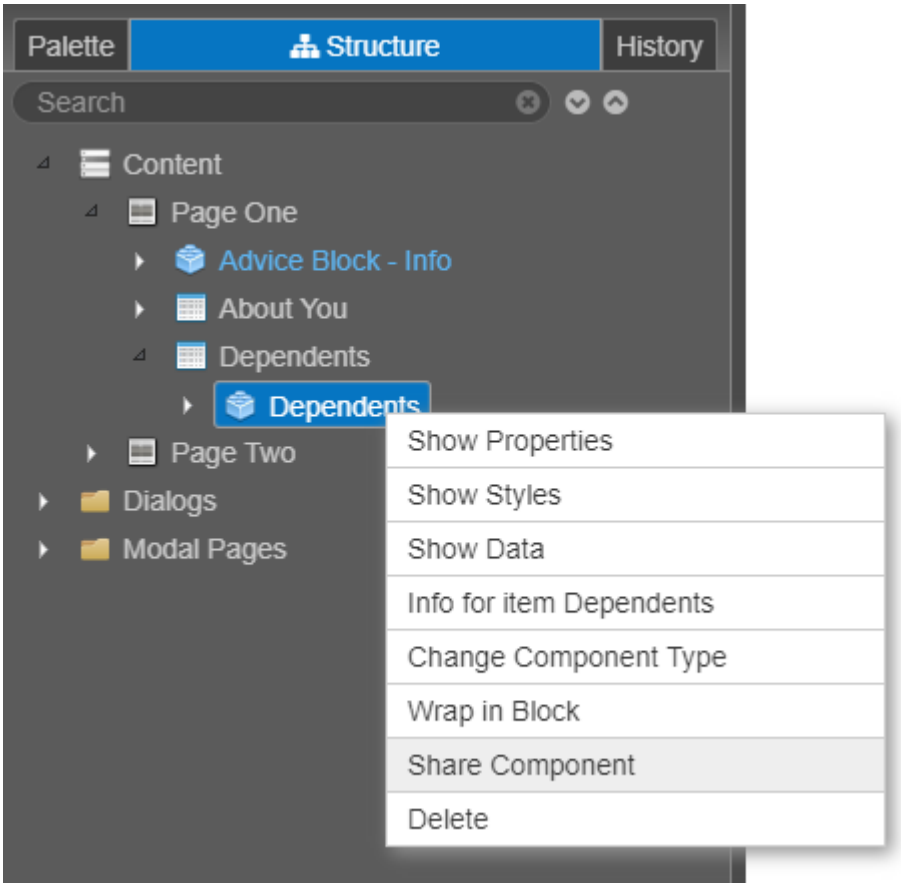


If more than one component is going to be part of a shared component, you must wrap the relevant components into a block. Your Structure panel should look like the image below.



Step 5 Create a Shared Component

Create a shared component that combines the components you just created. Your Structure panel should match the image below.



Use the following details for the shared component:

Name: Dependents_Exercise

Description: Contains the components for applicant's dependents.

Palette Folder: Custom Components

Instant Publish: Select Publish for immediate use and select Replace item with shared component

Publish Library: Select the appropriate project library (e.g. Maestro Training Exercises Library)

Remember: the instant publish options will publish the shared component to the library and replace the existing 'Block' with the shared component. This will ensure the current form is updated if the shared component changes.

To finish, click **Save** to save the **Create Component Begin** form - then close the form.

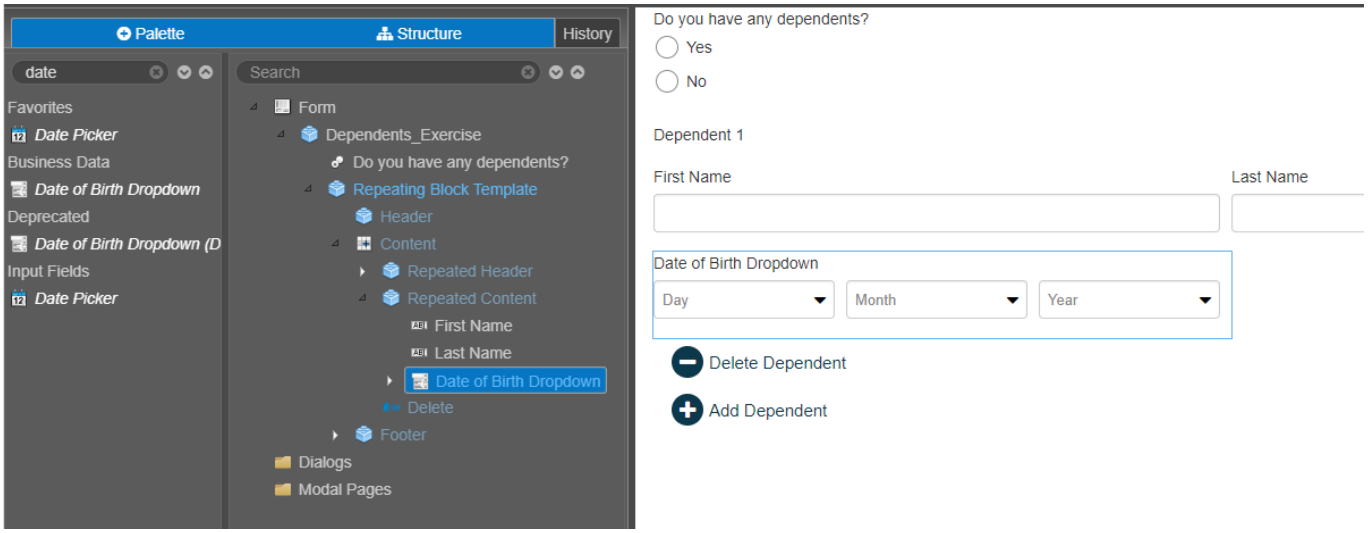
Step 6 Edit a Shared Component

Important: make sure your refresh the browser window (Press F5) that displays the Management Dashboard before you proceed.

From the Maestro Management Dashboard, open the shared component you just created and make a change to the component.

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Components**.
4. Double-click **Dependents_Exercise**.
5. Click **Edit Version** to open the Component.

Add the **Date of Birth** dropdown to the repeating block. Your shared component should match the image below.



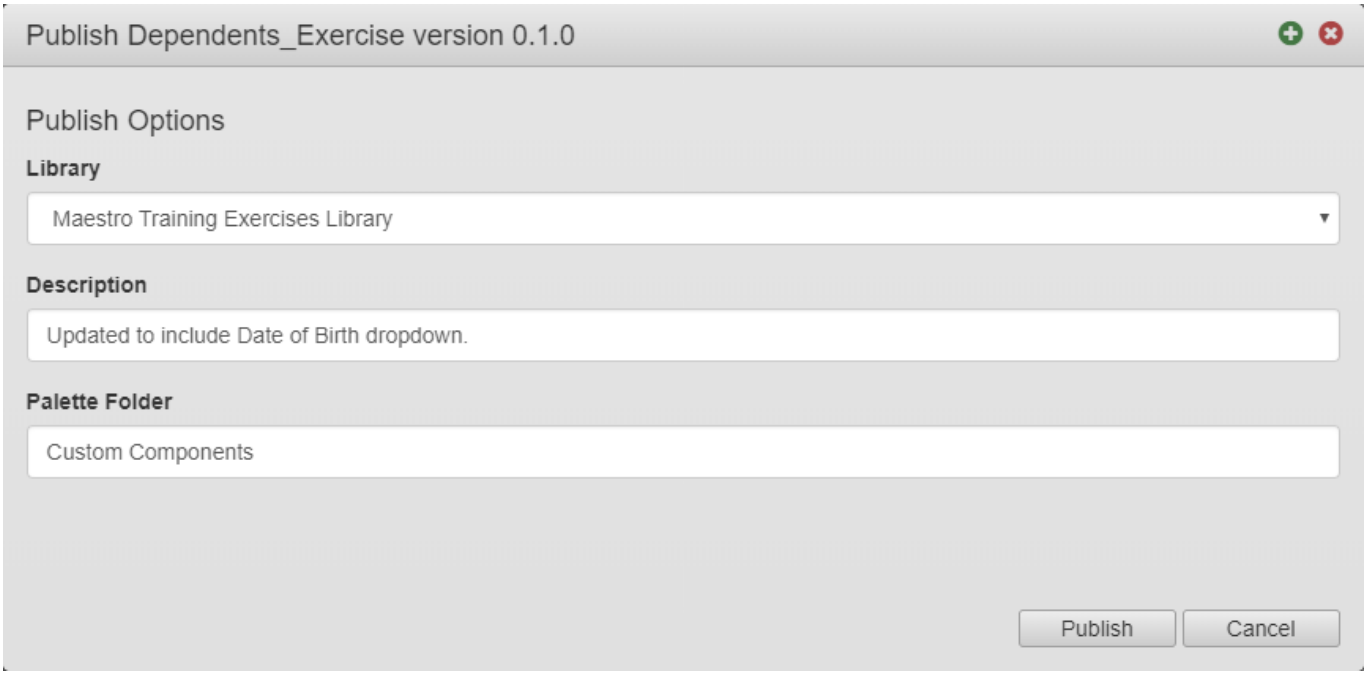
Note that the shared component won't include form elements like the navigator or sections because it's not linked to a specific template.

Click **Save** to save the changes to the shared component.

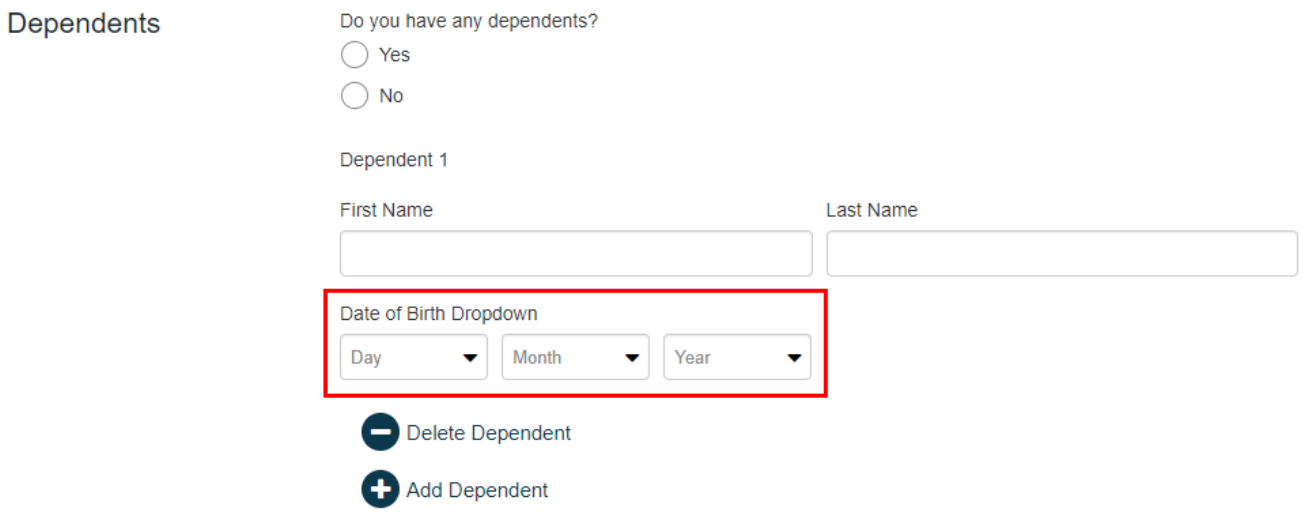
Step 7 Publish Updated Component

Once the component has been updated, **Publish** it so the changes are updated in forms currently using the component as well as any new uses of the shared component.

Use the image below for guidance.



Once you publish the updated component, **Close** the Component window and **Open** the **Create Component Begin** form to confirm the changes to the component (below).



Extra Help

Need some extra help?

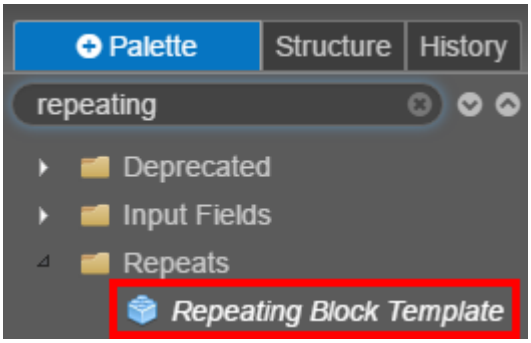
Use the information below to assist you in completing the exercise steps.

Using a Repeating Block Template

The **Repeating Block Template** enables you to easily setup repeating functionality.

Use the Palette Search to find the **Repeating Block Template**.

Drag the component onto your form.

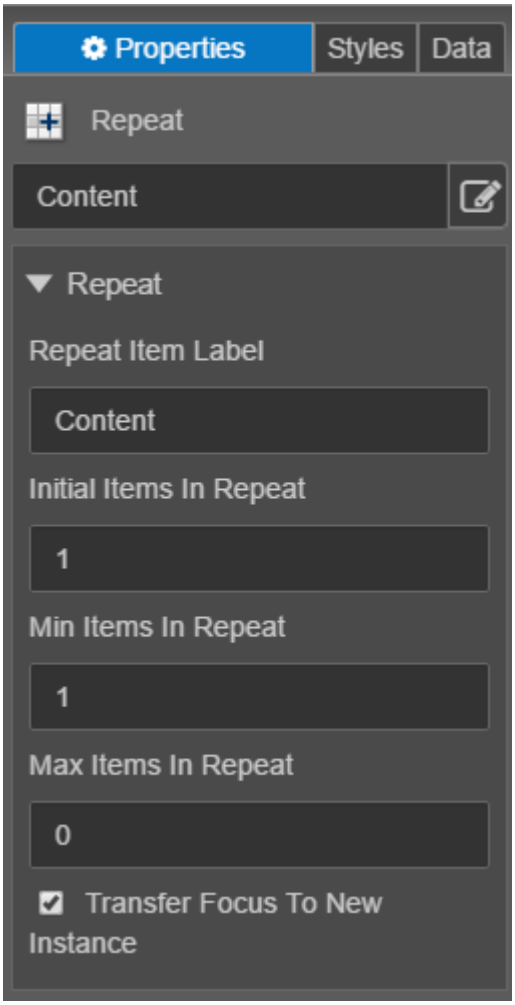


Add the components you want to repeat into the field area of the block.



To make adjustments to the Repeating Block properties, select **Content** within the Repeating Block Template (Structure panel).





Repeat Item Label: updates the label at the top of the block, as well as the button text.

Initial Items In Repeat: determines how many repeating components are on the form by default.

Min Items In Repeat: determines the minimum number of repeating components that must be displayed on the form.

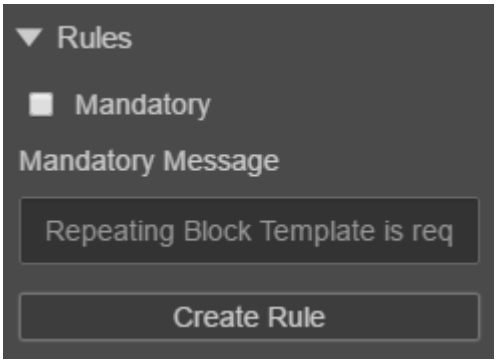
Max Items In Repeat: determines the maximum number of repeating components the user can add to the form.

Create a Show if Rule

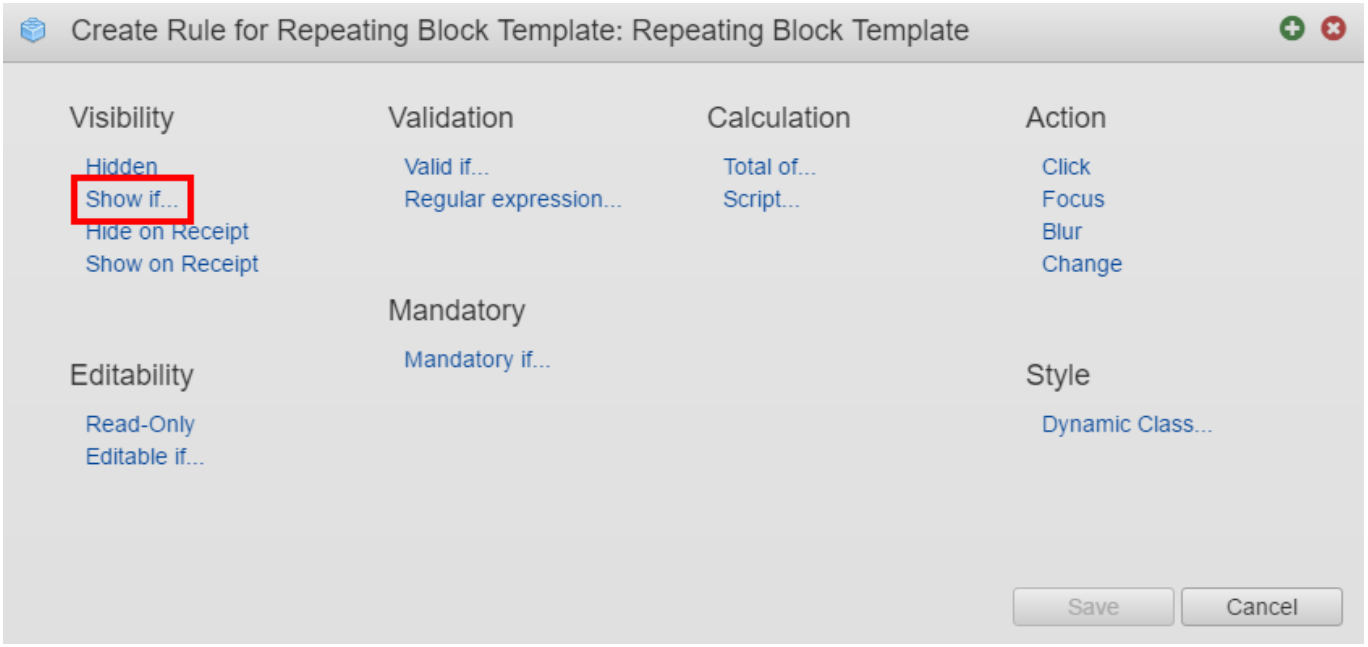
The Show if rule allows you to show or hide a component based on the selection of another.

Start by selecting the component you want to show or hide.

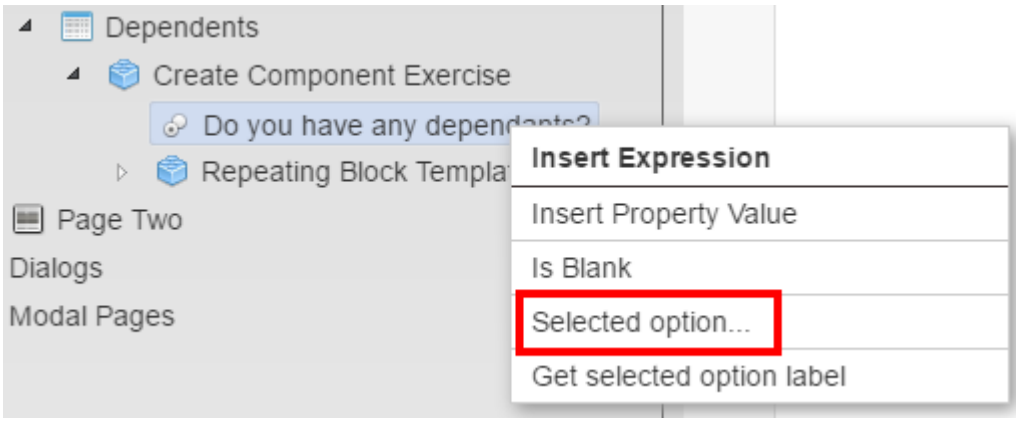
Click **Create Rule** in the **Rules** section of the **Properties** panel.



Select **Show if** (under **Visibility**).

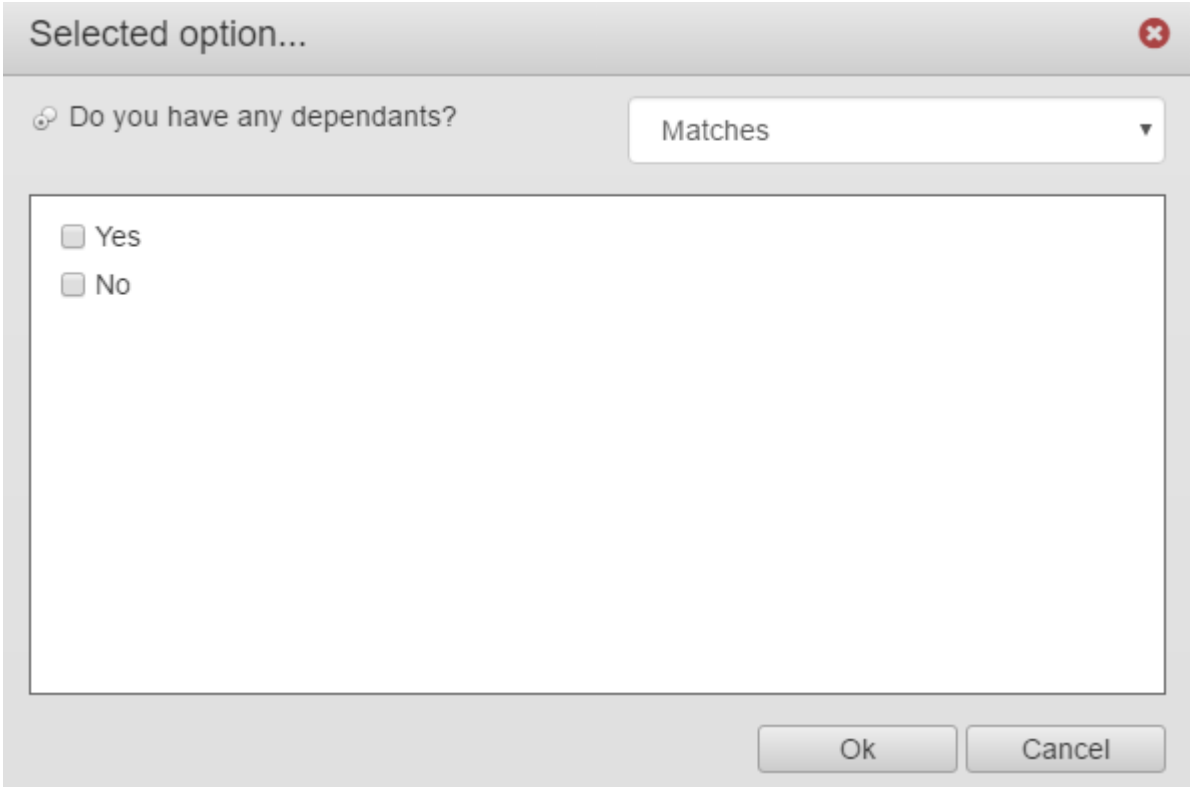


Right-click the component that determines if the selected component is displayed or hidden and click **Selected option**.



In the top right corner of the **Selected option** window, choose if the option should match or not match, then select the relevant options.

Click **OK** to close the window.



The script for the rule and selected components is displayed.

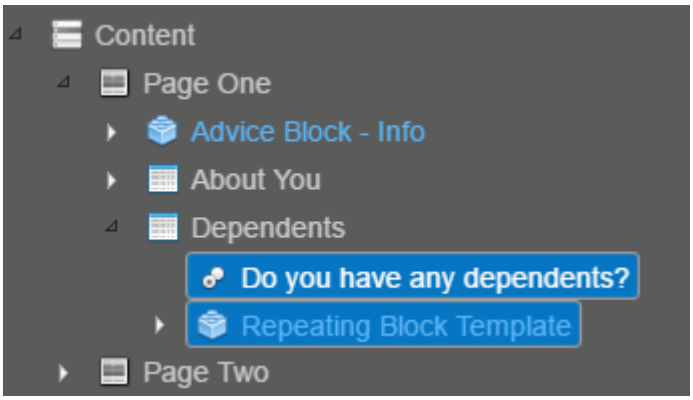
Click **Save** to continue.



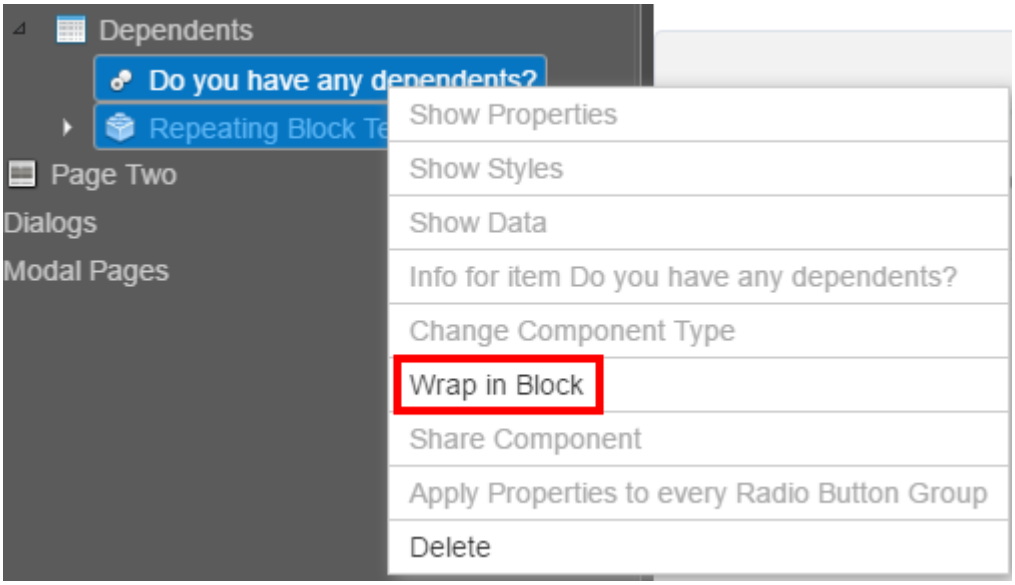
Wrap Components in a Block

If a shared component is made up of more than one component, you must wrap the relevant components in a block first, then create the shared component.

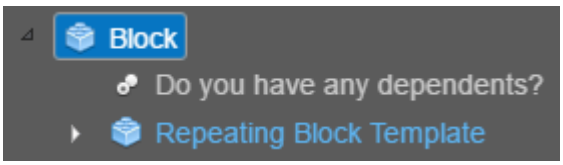
To wrap multiple components in a block, select all the components from the structure panel using either SHIFT or CTRL (see below).



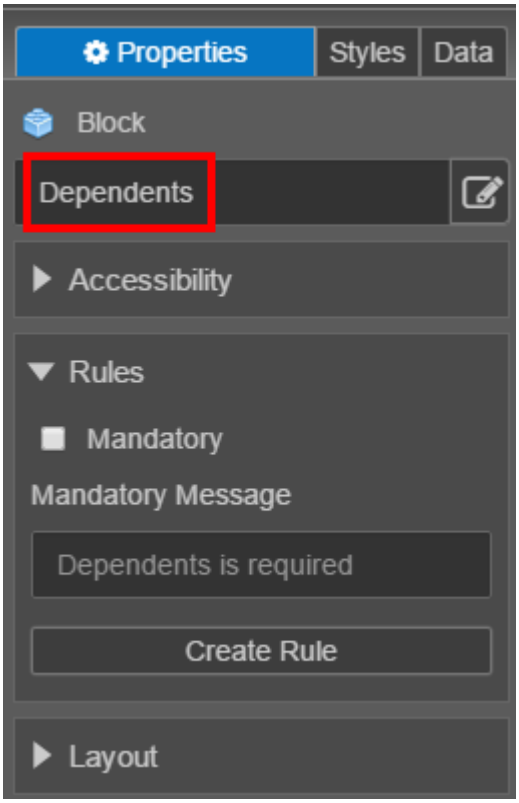
Right-click one of the selected items and select **Wrap in Block**.



The **Block** is created and the components are displayed within.



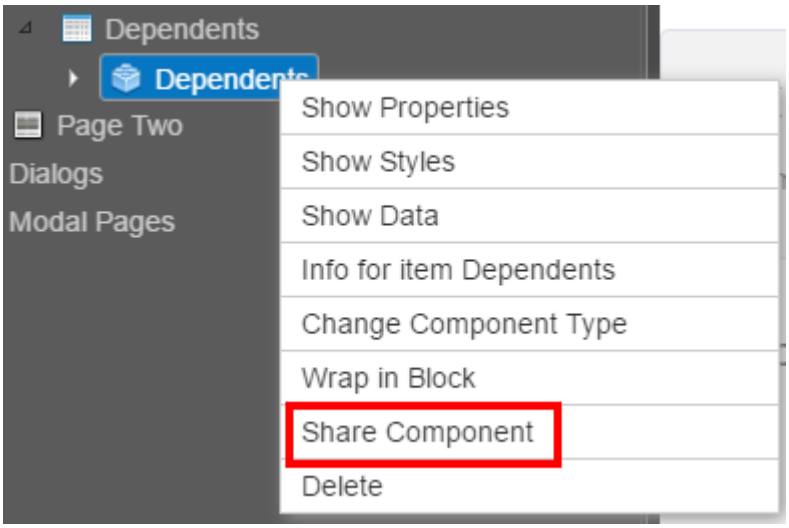
Use the **Properties** panel to rename the block.



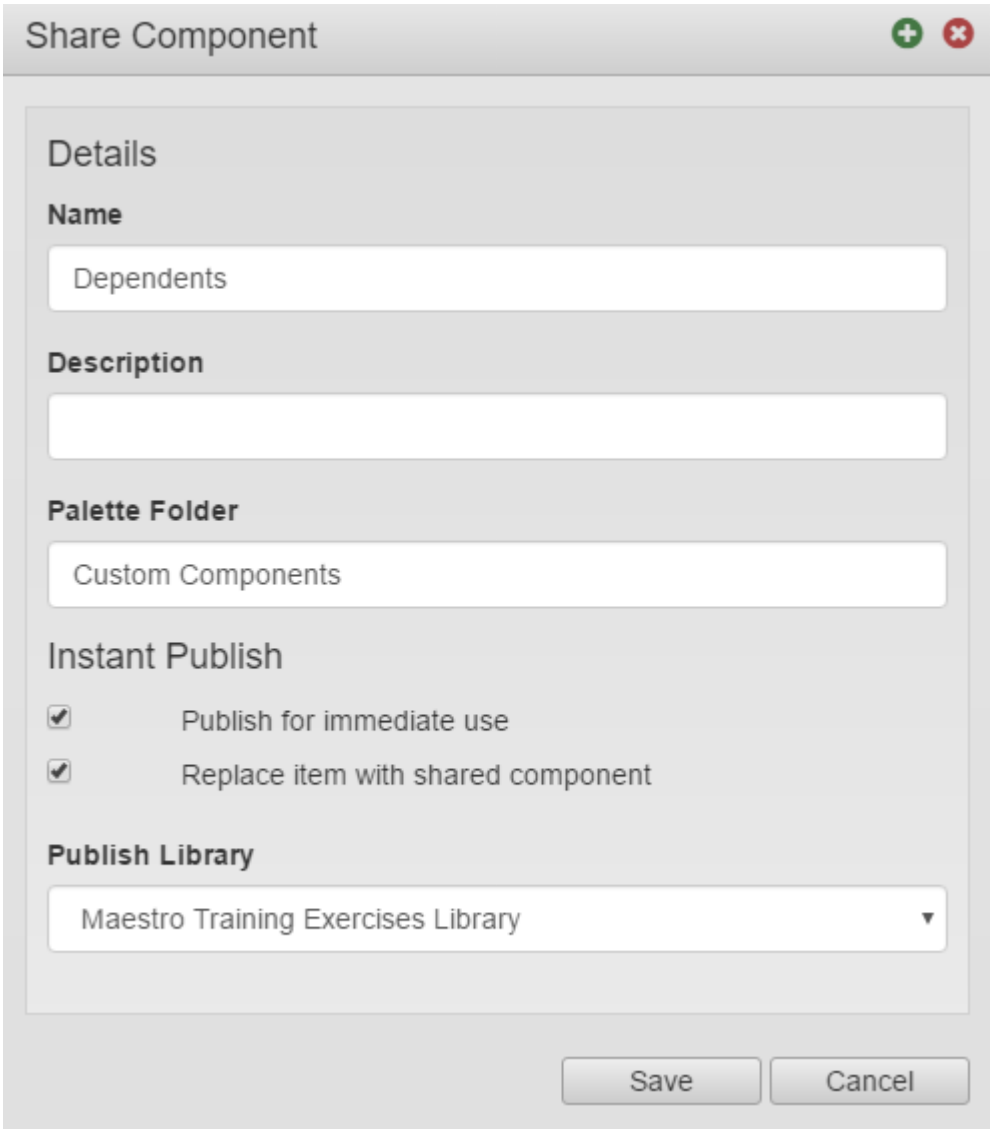
Create Shared Component

Shared components are stored outside a form, in the Components folder of the project. Once a Shared component is published, it is stored in a Library. Shared components can be updated outside of a form. When they're updated, any forms using that component are updated automatically.

To create a Shared Component, right-click the block and select **Share Component**.



The **Share Component** window will display.



Name: is the name of the shared component displayed in the Palette panel.

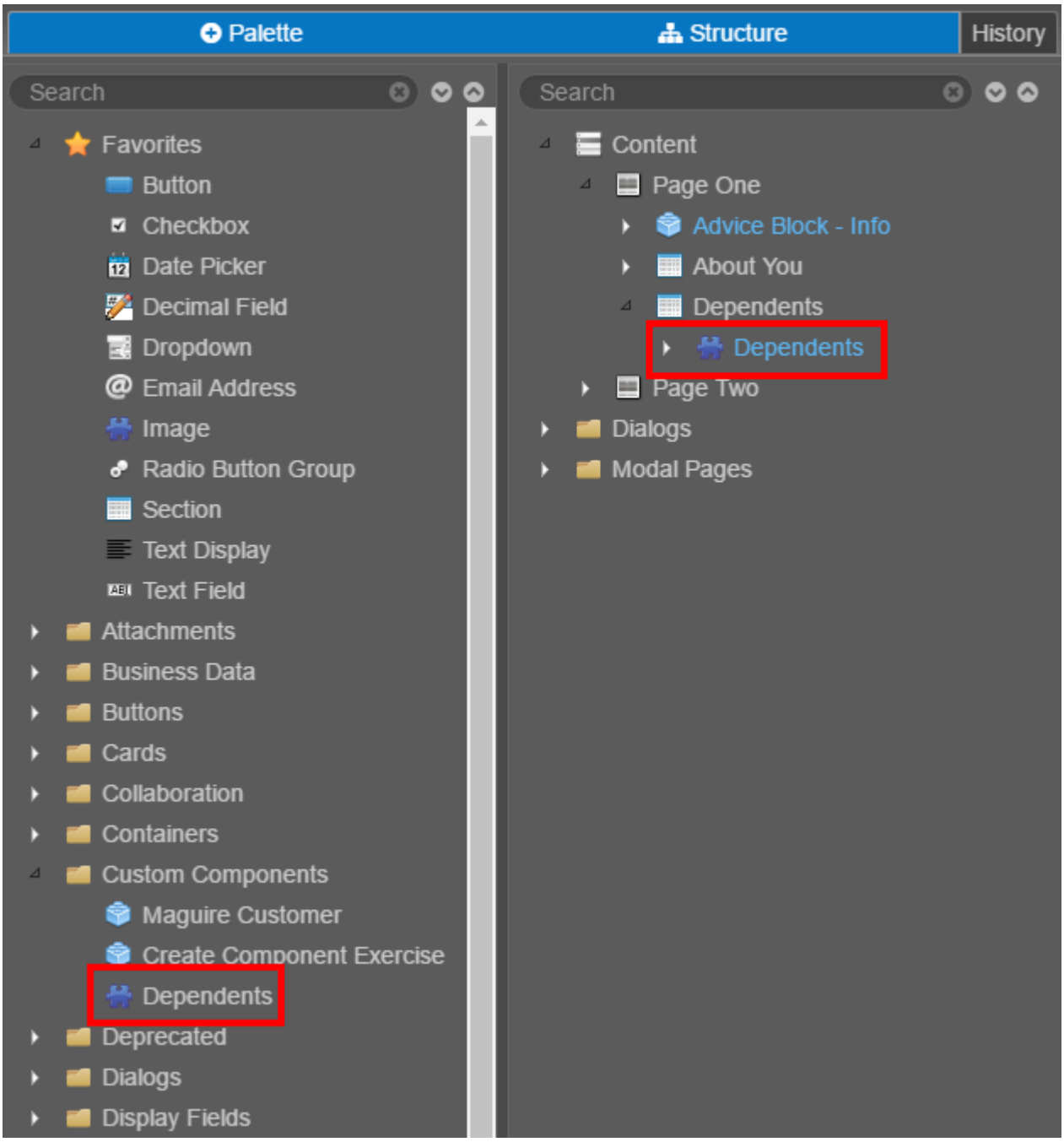
Description: contains details about the shared component like the components in contains.

Palette Folder: is the folder where the shared component is stored within the Palette structure. If you enter a folder name that's already created, the shared component will be added to the folder. If you enter a name not currently be used, a new folder will be created.

Instant Publish – Publish for immediate use: publishes the shared component as soon as you click Save. This adds the component to the library, so other forms can access and use the shared component.

Instant Publish – Replace item with shared component: removes the block currently in the form and replaces it with the shared component. This allows the component in the form to be updated whenever changes to the shared component occur.

The **Structure** panel is updated with the new shared component and the folder in the **Palette** panel will contain the shared component.



Layout Exercise v5.0

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for working with the layout of your form. During this exercise you will use an existing form to edit the layout of components.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

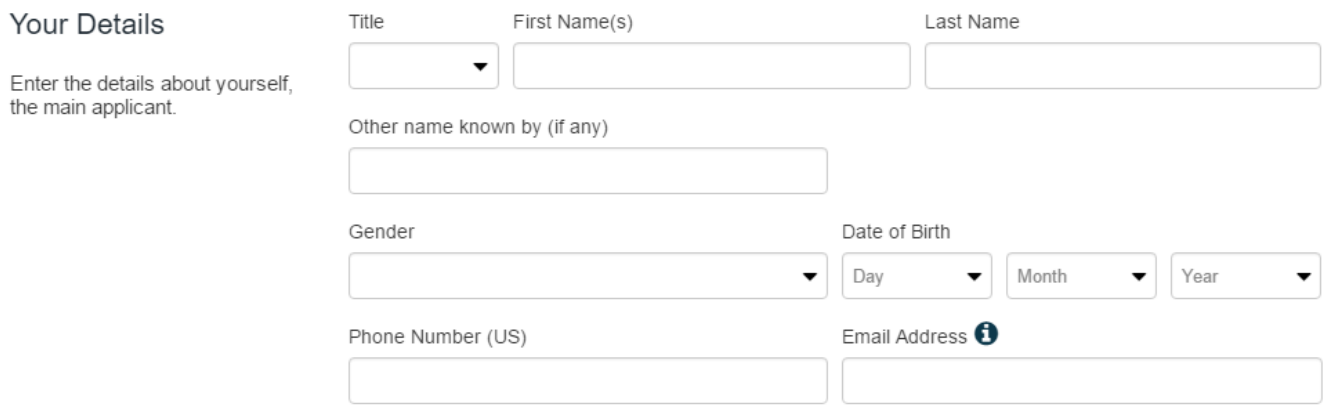
You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises > Layout Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1: Open the Existing Form Layout Begin.

1. In the Navigator Panel, expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Double-click **Layout Begin**.
5. Click the **Edit Version** button to open the form.

Step 2: Adjust the Layout of Title, First Name and Last Name

On the Applicant Details page, adjust the components so they appear on the same line when viewing the form on large screens. Update the 'Your Details' section to match the image below.



Set Title to 2 columns wide.

Set First Name(s) to 5 columns wide.

Set Last Name to 5 columns wide.

Set Other name known by (if any) to 7 columns wide.

Set Phone Number to 6 columns wide.

Set Email Address to 6 columns wide.

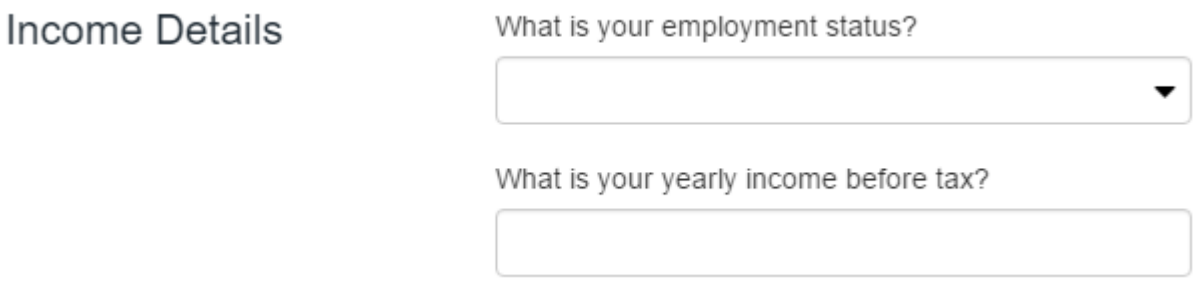
Best practice: only change the width values for the Large and Full Screen sizes.
In the Editor, you'll only see the changes made to the Full screen size.
Use Preview to check the form and make any necessary adjustments.

Once you've made room on a row, you can drag a component into position if it doesn't automatically move to the larger space.

Click **Save**.

Step 3: Adjust the Income Details Components

On the Income and Expenses page, adjust the components so they appear on their own row.

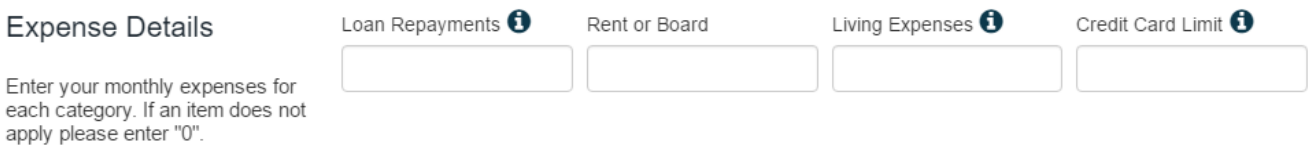


Both components are set to 6 columns wide (for the Large and Full Screen sizes).

Click **Save**.

Step 4: Adjust the Expenses Details Components

On the Income and Expenses page, adjust the components so they appear as four columns on the same row.



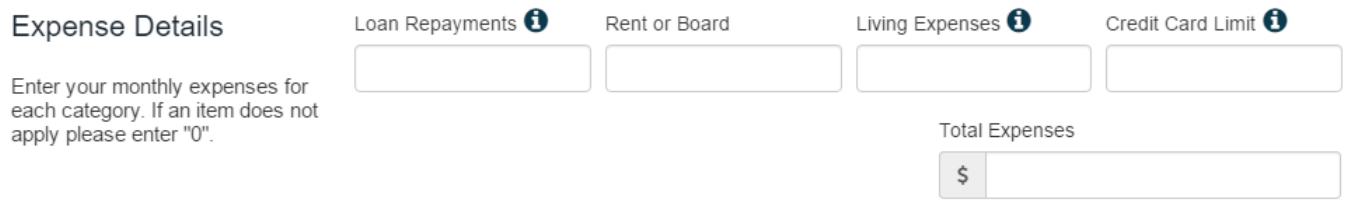
Set all four components to 3 columns wide.

Once you've made room on a row, you can drag a component into position if it doesn't automatically move to the larger space.

Click **Save**.

Step 5: Add a New Component to the Expense Details Section

In the Expense Details section, add a new Total Expenses component.



> Notice: the currency symbol for Total Expenses is added by using the Currency Field.

Set Total Expenses to 5 columns wide.

Set the Offset for Total Expenses to 7 columns.

Click **Save**.

Extra Help

Need some extra help?

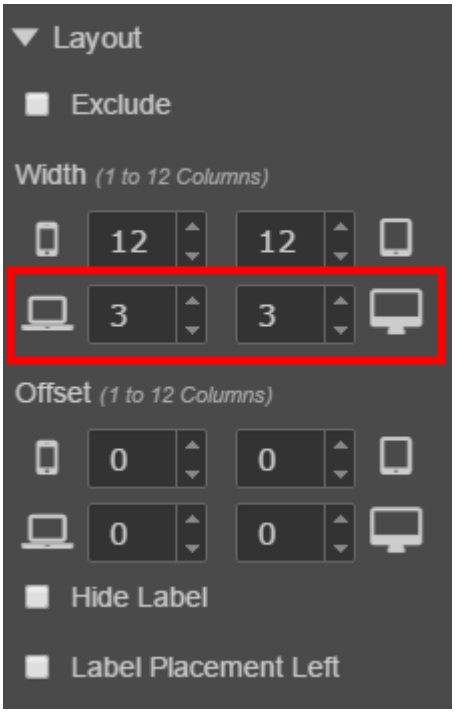
Use the information below to assist you in completing the exercise steps.

Change the layout of components

Select the component you want to change. Adjust the Width values within the Layout section (Properties panel) to make any changes.

Decrease the column width for the relevant screen size.

> You will only see changes in the Editor when you adjust the full screen layout.

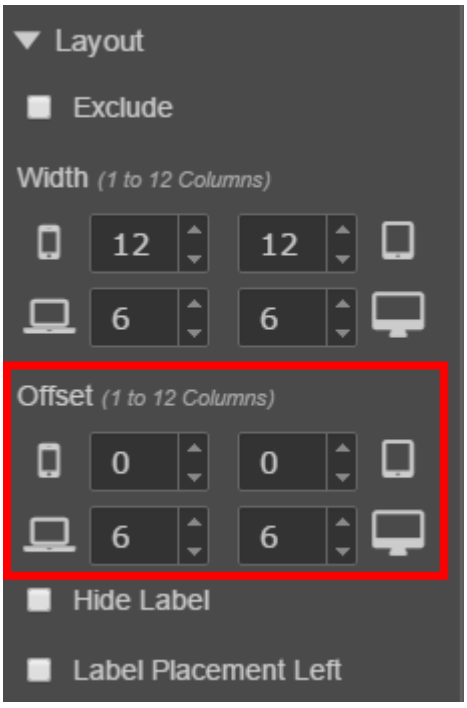


Change the position of components (using Offset)

Components can be shifted to the right or left within their row. Select the component you want to change.

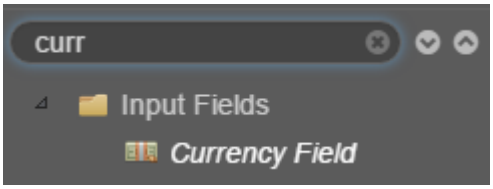
Adjust the Offset values within the Layout section (Properties panel) to make any changes.

You can increase the offset to move the component to the right or decrease the offset to move the component to the left.

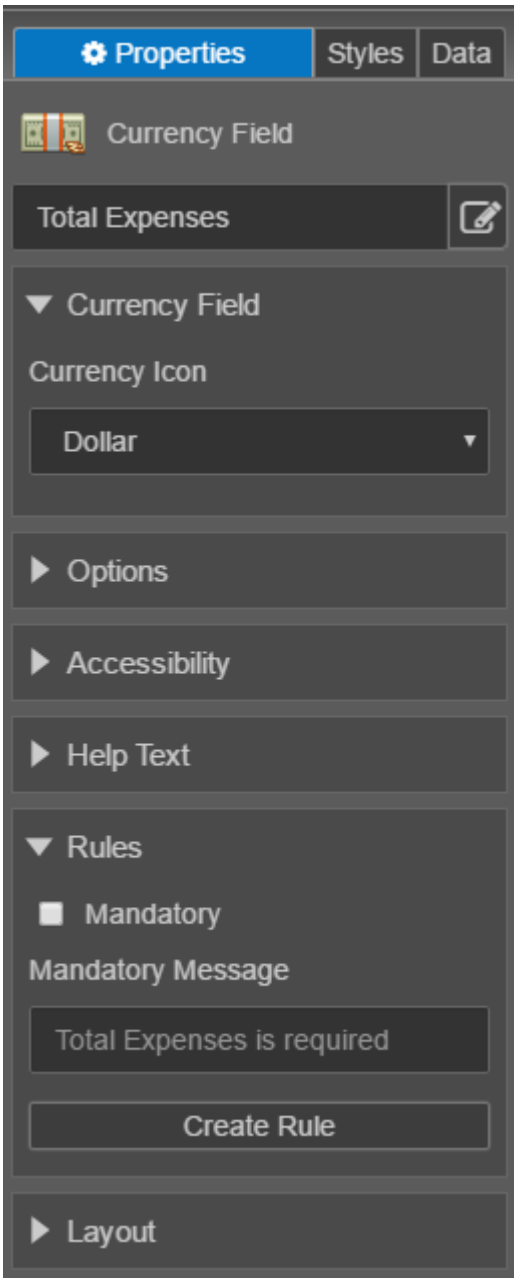


Add a Currency Field

The Currency Field component automatically includes a currency symbol. Use Search in the Palette to find the Currency Field and add (drag) it to your form.



Use the Properties panel to make changes to the field, including the currency icon used.



Pages and Sections Exercise v5.0

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills with pages and sections. During this exercise, you will use an existing form to edit pages and sections as well as add new ones.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find **Extra Help** at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the **Maestro Exercise Project: Maestro Training Exercises** > **Pages and Sections Finish**. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Create a New Version

Create a new version of the existing form named **Pages and Sections Begin**.

Follow the steps below to create the new form.

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Select **Pages and Sections Begin**.
5. Click **New Version**.
6. Enter the following **Form Description**:

   Description: This form version is to practice working with sections.

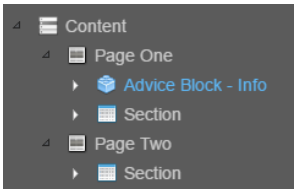
7. Click **Create**.
8. Click **Edit** to open the form.

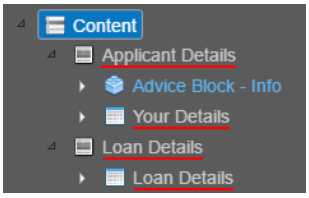
Step 2 Update the Names of the Existing Pages and Sections

In this form, you will find two pages, each containing a section. You will change the names of both pages and their sections.

Use the Structure panel to select the pages and sections you want to update.

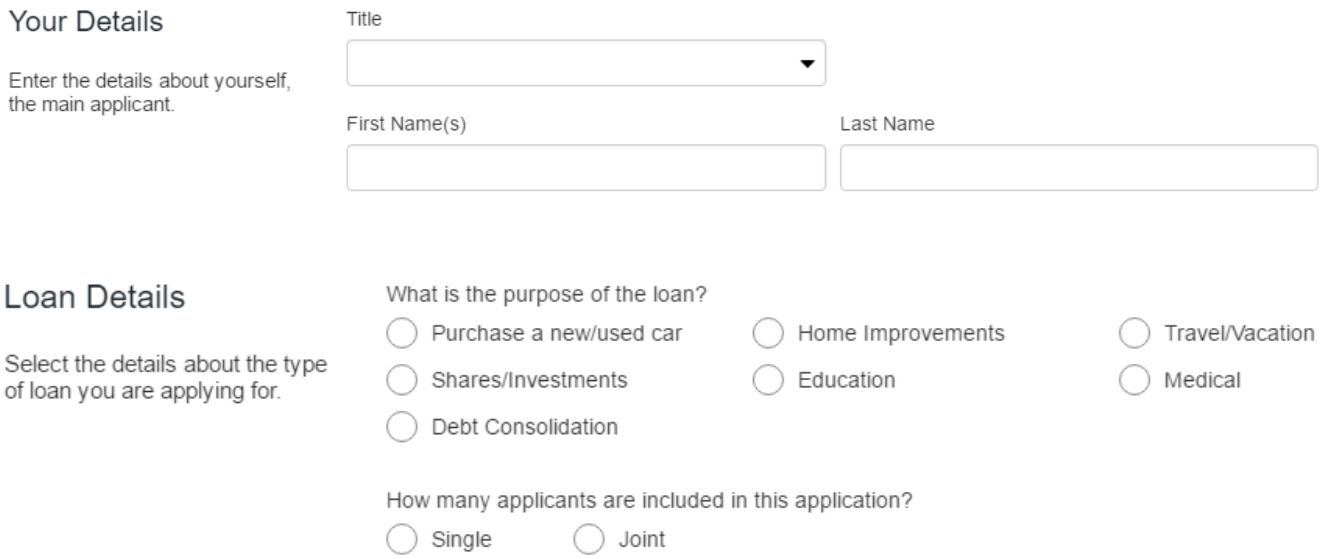
Starting Structure Panel Updated Structure Panel





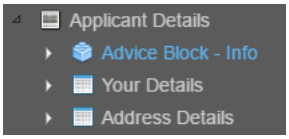
Update the Section - Additional Text (using the supplied text) so your form matches the image below.

- Your Details: Enter the details about yourself, the main applicant.
- Loan Details: Select the details about the type of loan you are applying for.



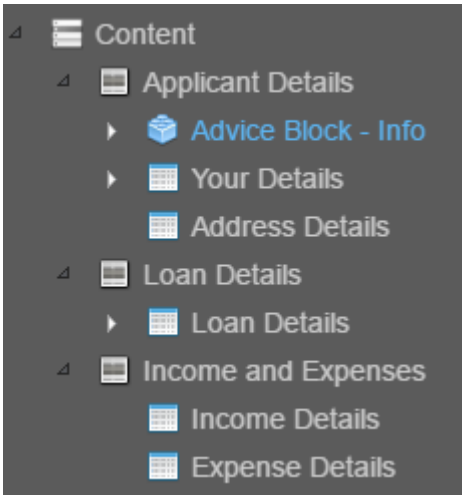
Step 3 Add a new section to the Applicant Details page

Add your new section after the existing section on page one. Your Structure panel should appear similar to the image displayed.

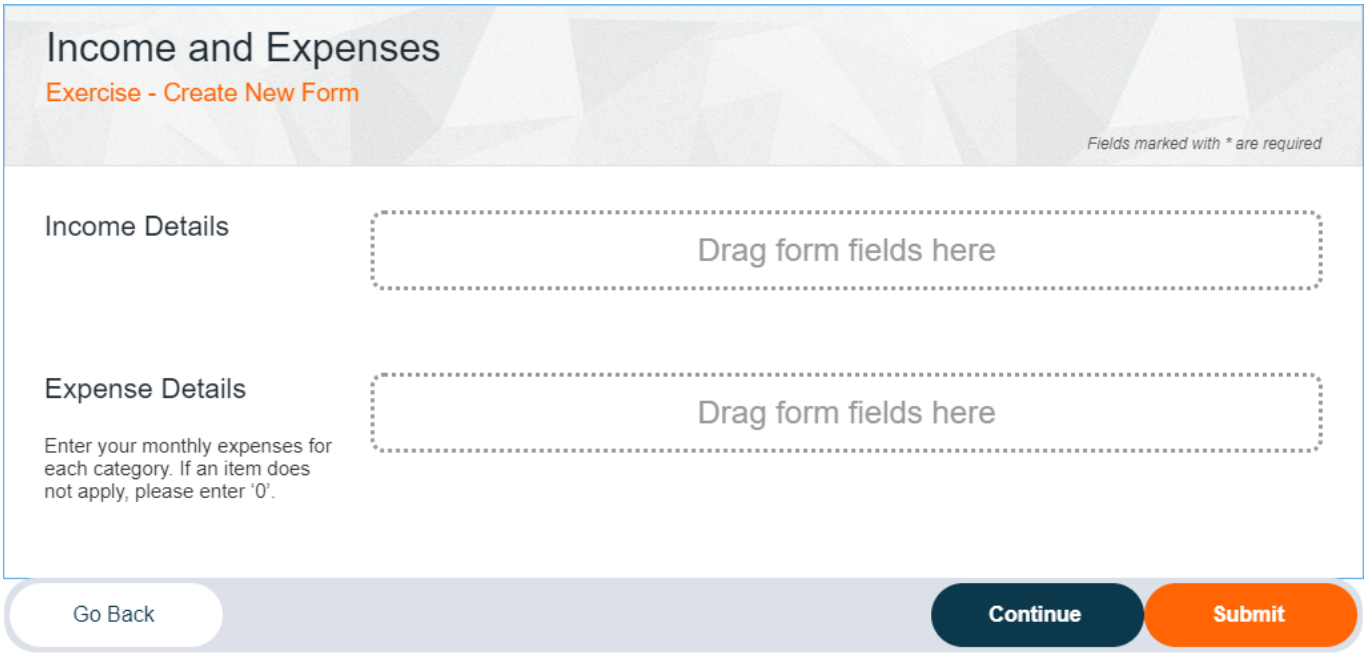


Step 4 Add a New Page with Two Sections

Add a new page (called 'Income and Expenses') after the existing pages. Add two sections to the page. Your Structure panel should match the image below.



The Income and Expenses page should match the image below.



Notice: the additional text for the Expense Details section (text below).

Expense Details: Enter your monthly expenses for each category. If an item does not apply, please enter '0'.

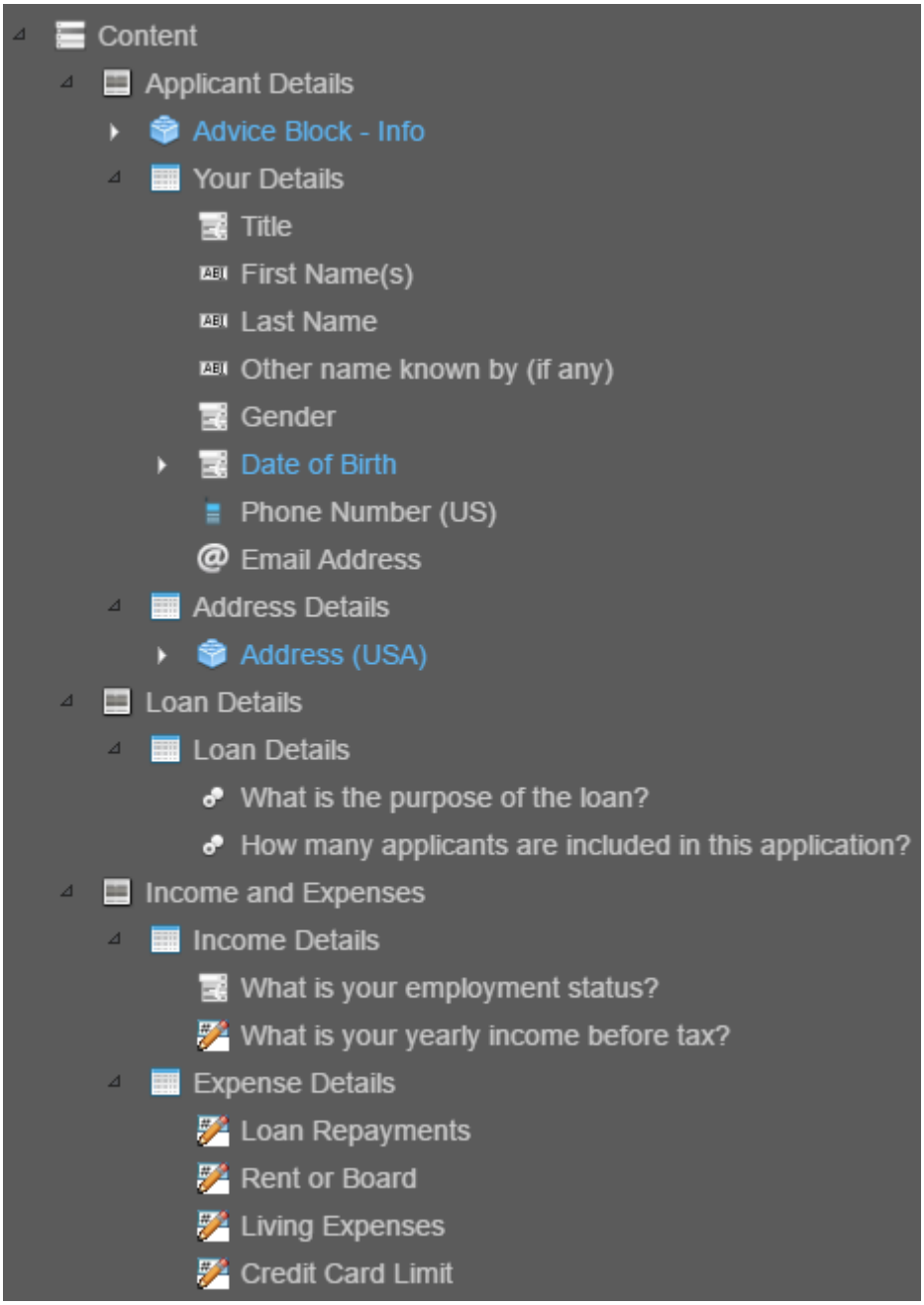
Step 5 Rearrange the Components

Using the Structure panel:

- Move the Address block to the Address Details section.
- Move (i.e. drag and drop) the income questions to the new Income Details section on page three.
- Move the expense questions to the new Expense Details section on page three.

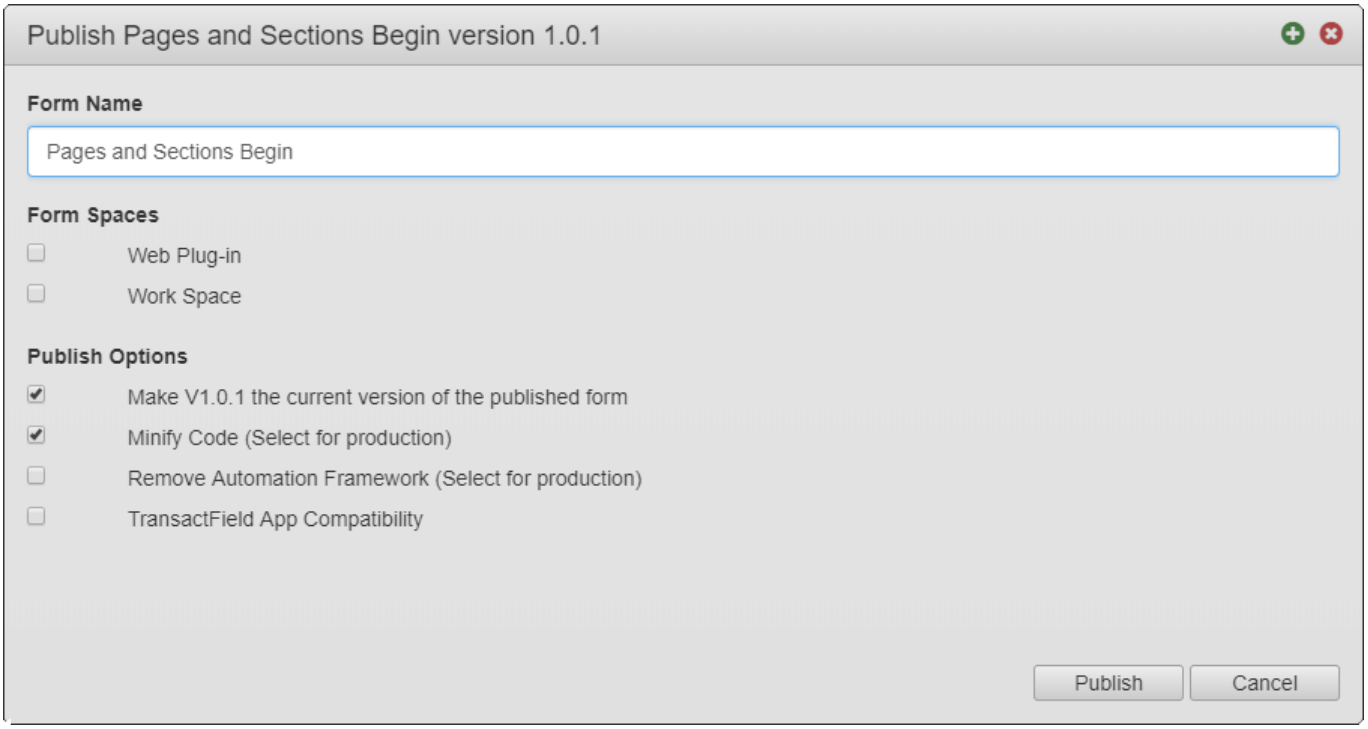
> You can select multiple components at one time by using SHIFT or CTRL.

The image below shows how the Structure panel should look once the components have been rearranged.



Step 6 Publish the Form

Click Publish on the toolbar to publish your form. The publish settings should match (or be like) the image below.



Extra Help

Need some Extra Help?

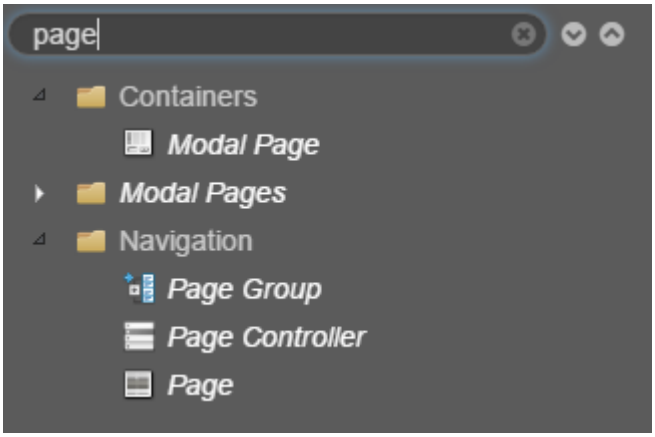
Use the information below to assist in completing the exercise steps.

Add a new page

Use the Search box in the Palette to find the Page component.

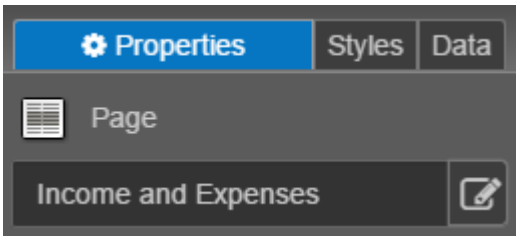
Drag the Page component to the Content container on the Structure panel.

> Pages Groups and Pages cannot be added into the Editor pane.



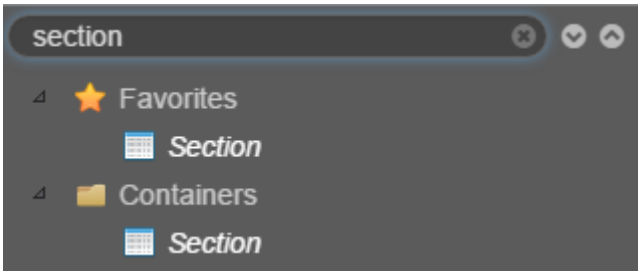
You can also right-click the Content folder (Structure panel) and choose Add Page Group.

Use the Properties panel to change the name of the page.



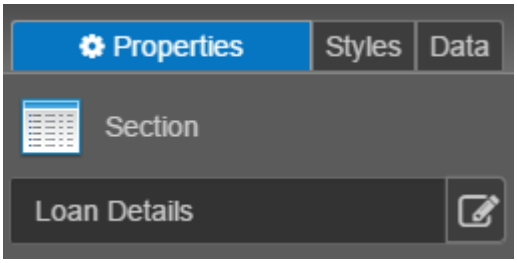
Add a new section

Sections can be added to pages to help organize the display of the components. Use the Search box in the Palette to find the Section component (or look under 'Favourites'). Drag the Section component to the Page container on the Structure panel.



You can also drag a new 'Section' component onto the form.

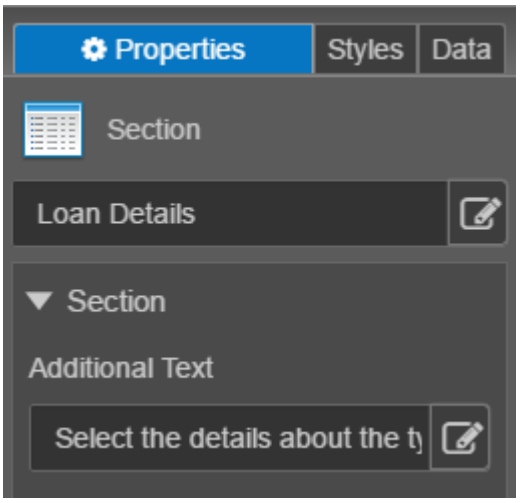
Use the Properties panel to change the name of the section.



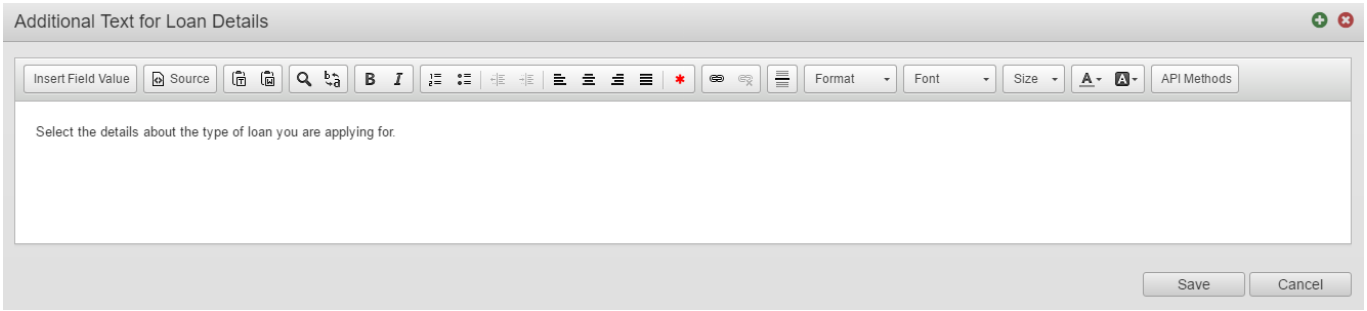
Additional Text

Use the Properties panel to customize the text below the section title.

In the Properties panel, enter text in the Additional Text entry box.

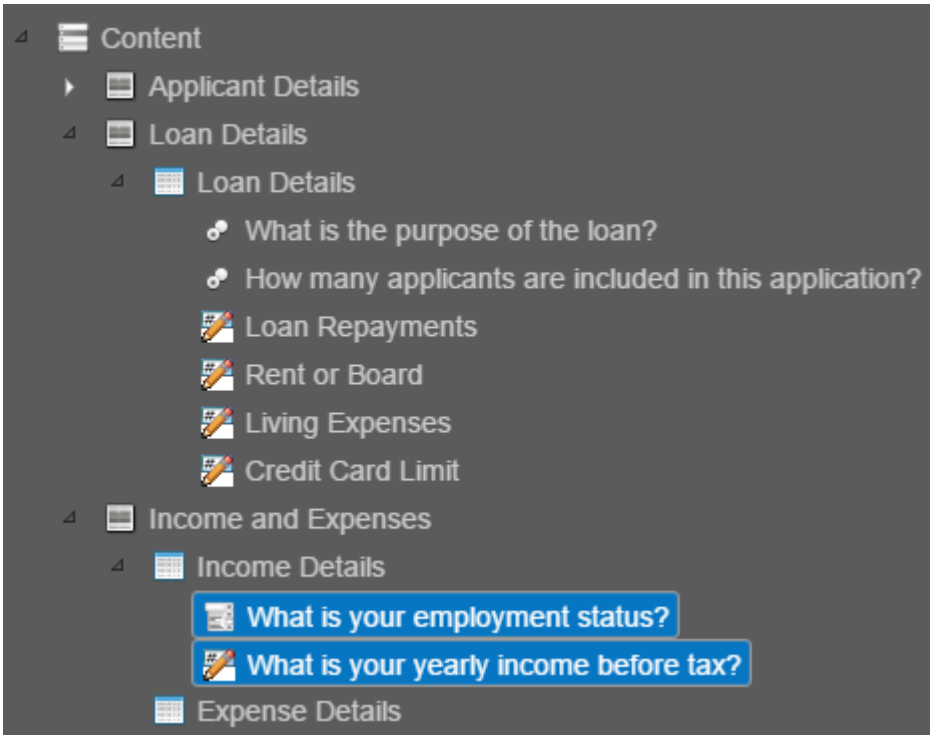


You can click the Edit button to display the Additional Text window. Here you can edit and format the additional text displayed on the form.



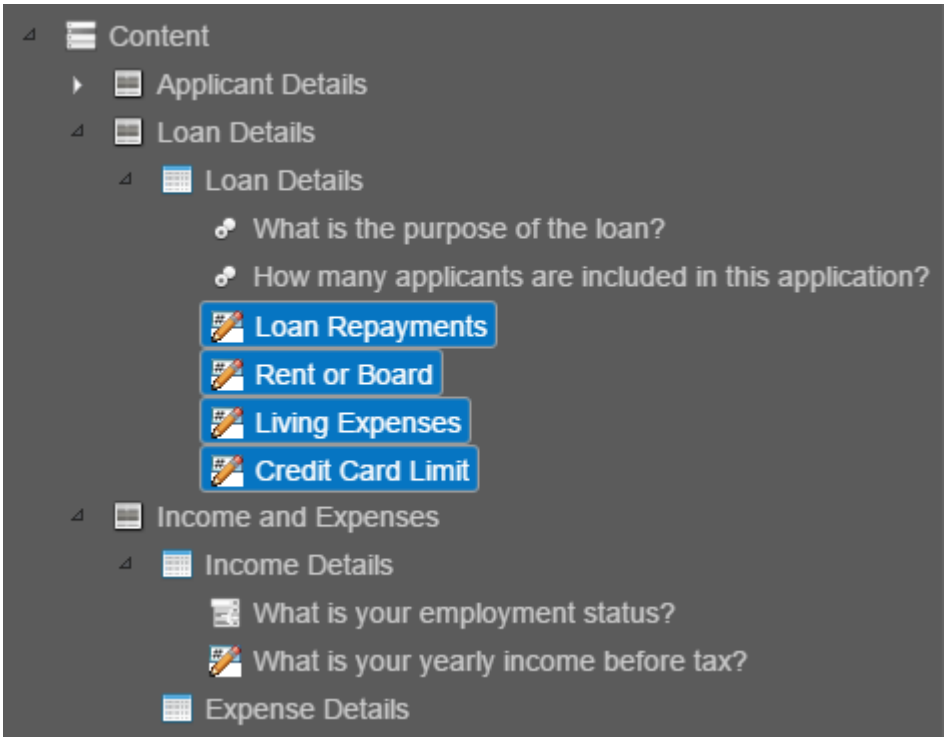
Rearrange Components

Use the Structure panel to drag and drop components between pages and/or sections.

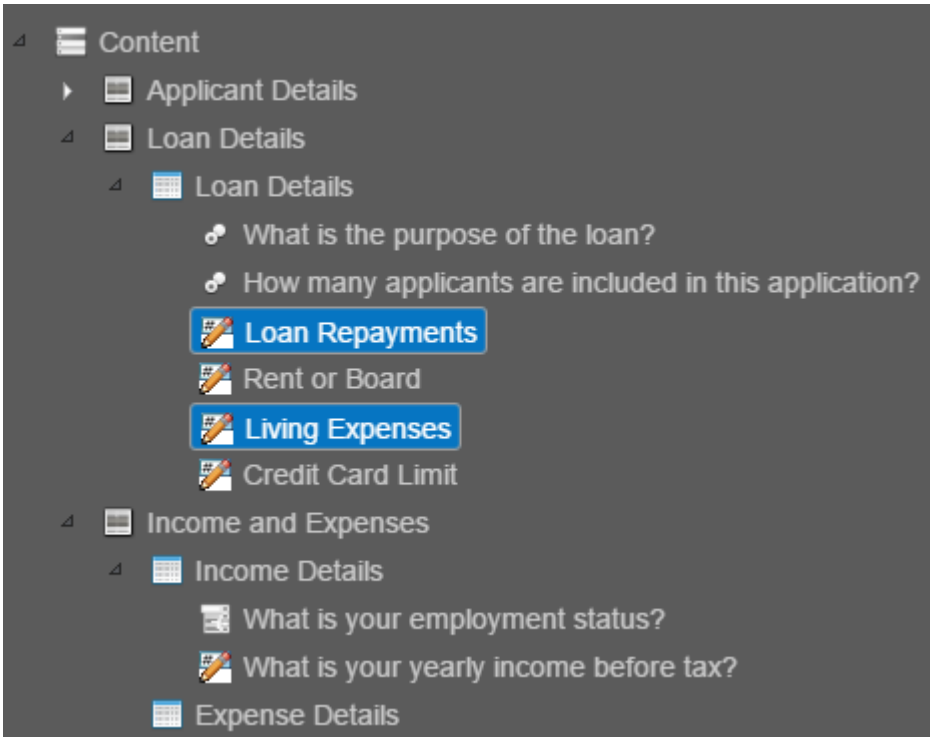


Select multiple components at one time using SHIFT or CTRL. Once selected, you can drag them to a different container.

Select adjacent components using SHIFT (below)



Select non-adjacent components using CTRL (below)


Content
Applicant Details
Loan Details
Loan Details
What is the purpose of the loan?
How many applicants are included in this application?
Loan Repayments
Rent or Board
Living Expenses
Credit Card Limit
Income and Expenses
Income Details
What is your employment status?
What is your yearly income before tax?
Expense Details

Repeating and Shared Components Exercise v5.0

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for using repeating components and shared components. During this exercise you will use an existing form to add repeating and shared components.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises > Repeating and Shared Components Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Open the Existing Form 'Repeating and Shared Components Begin'

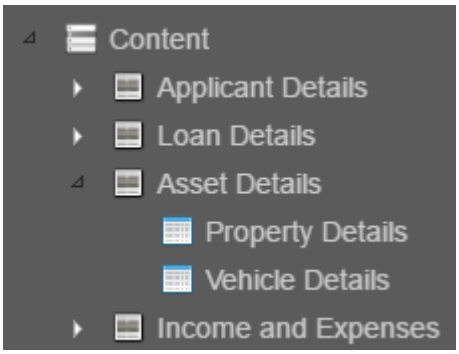
1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Double-click **Repeating and Shared Components Begin**.
5. Click the **Edit Version** button to open the form

Step 2 Add a New Page with Sections to Your Form

Add a new Page called **Asset Details**.

Add two new Sections to this page called **Property Details** and **Vehicle Details**.

Your Structure panel should match the image below.



Click **Save**.

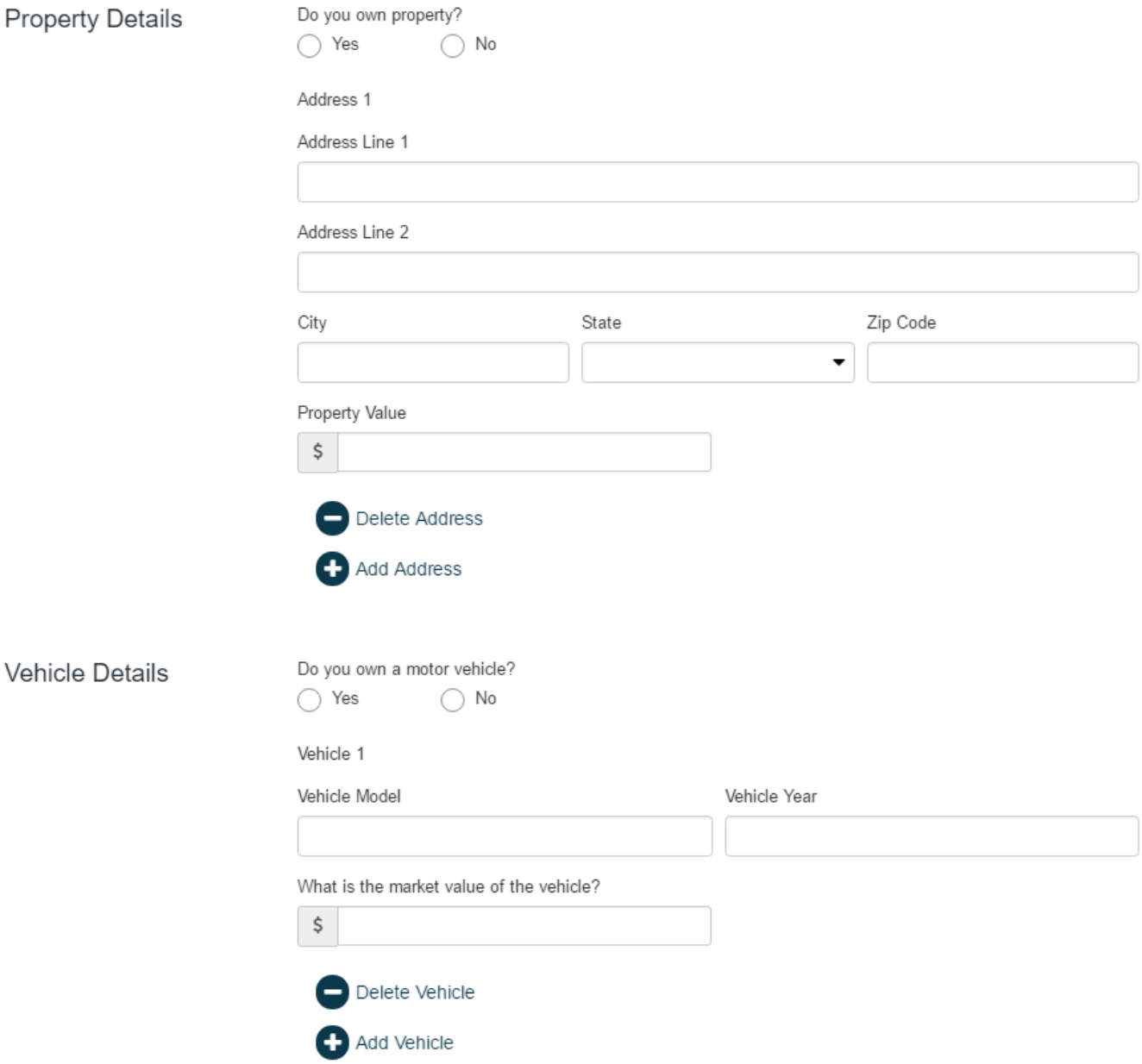
Step 3 Add Components to the Page

Add components to the new page so it matches the image below.

> Notice: the Address 1 block and Property Value component are within a Repeating Block Template. Before you add components to the page, you should add the Repeating Block Template first, then add the components to the block.

Create a Show if rule to only show the Repeating Block Templates when the user selects Yes to owning Property or a Vehicle.

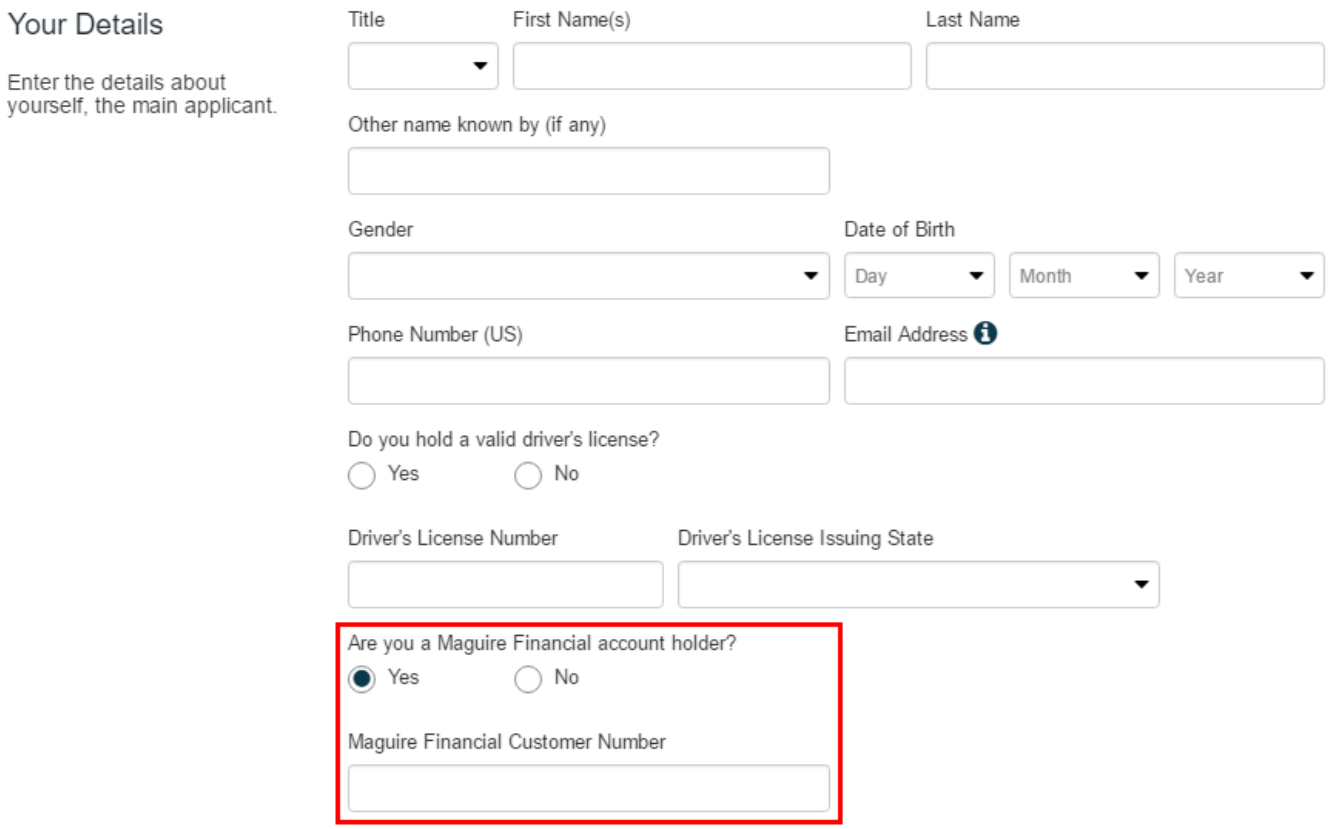
Click **Save**.



Step 4 Add a Shared Component

Add the Maguire Customer shared component to the bottom of the Your Details section.

Click **Save**.



Shared components can be stored anywhere in the Palette, so use the Search if you're unsure of a component's location.

Extra Help

Need some extra help?

Use the information below to assist you in completing the exercise steps.

Add a new Page and Section

Display the Palette and Structure panels side by side (use button in top left).

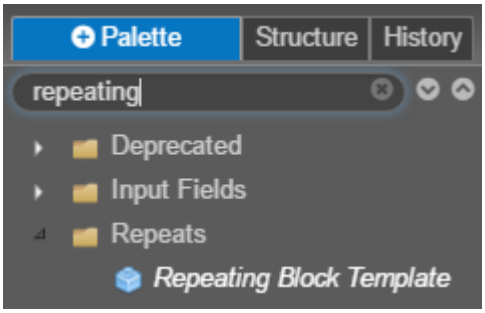
Search for the Page component and drag it onto the Content folder (Structure panel). Drag the new Page into the correct order.

Drag two new Section components onto the new Page.

Using a Repeating Block Template

The Repeating Block Template includes all of the necessary items to easily setup repeating functionality.

Use the Palette Search to find the Repeating Block Template (see below).



Drag the component onto your form.

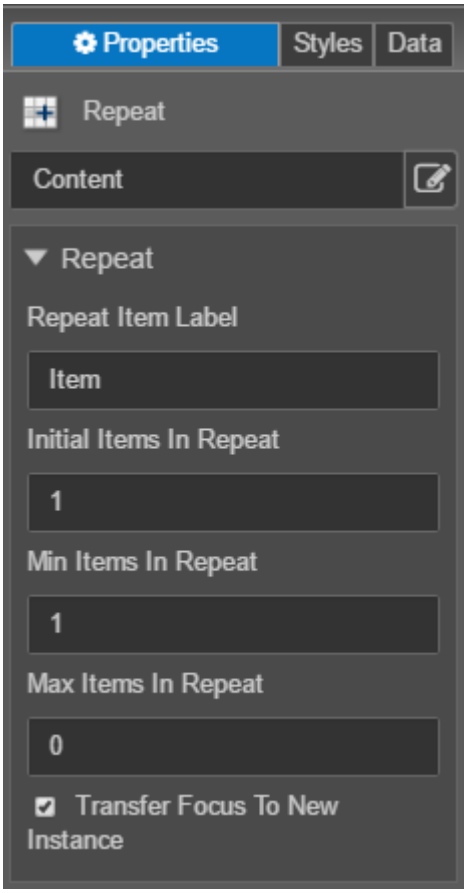
You can add the components you want to repeat into the field area of the repeating block.



To make any adjustments to your repeating block, select Content within the Repeating Block Template (Structure panel).



With Content selected you can edit values in the Properties panel.



Repeat Item Label: this updates the label at the top of the block, as well as the button text.

Initial Items In Repeat: determines how many repeating components are on the form by default.

Min Items In Repeat: determines the minimum number of repeating components that must be displayed on the form.

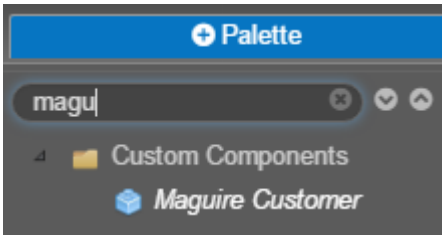
Max Items In Repeat: determines the maximum number of repeating components a user can add to the form.

Using a Shared Component

Shared components are stored outside of a form, in the Components folder within the project. A shared component can be updated outside the form. When updates are made, all forms using that component are updated automatically.

Use Search in the Palette to locate the shared component you want to use.

Drag the shared component to the form. The shared component will be added.



The shared component will contain the components along with any custom properties.

Are you a Maguire Financial account holder?

Yes No

Maguire Financial Customer Number *

Rules without Scripting Exercise v5.0

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating rules without scripting. During this exercise you will use an existing form to add non-scripted rules.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

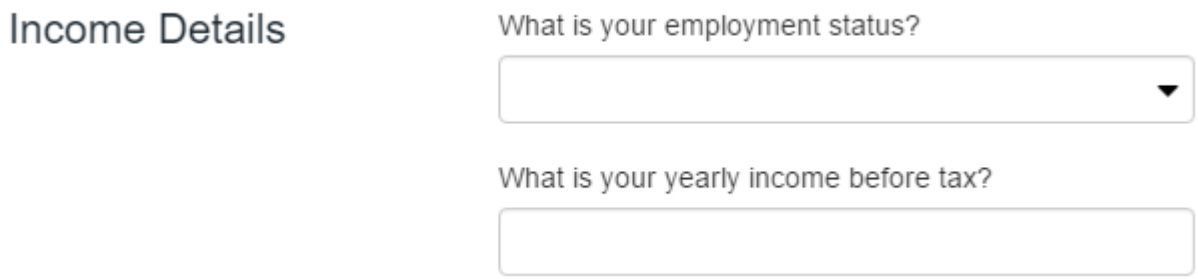
You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises > Rules without Scripting Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Open the existing form Rules without Scripting Begin.

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Double-click **Rules without Scripting Begin**.
5. Click **Edit Version** to open the form.

Step 2 Create a Rule for Income Details

Create a Show If rule that displays the 'yearly income' component if the 'employment status' is set to 'Employed' or 'Self-Employed'.

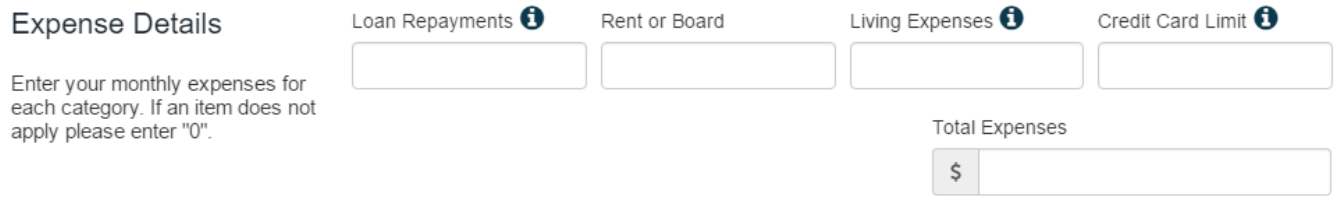


Click **Save**.

Step 3 Calculate the Total of the Four Expenses in the Expense Details Section

Calculate the total of the four expenses in the Total Expenses field.

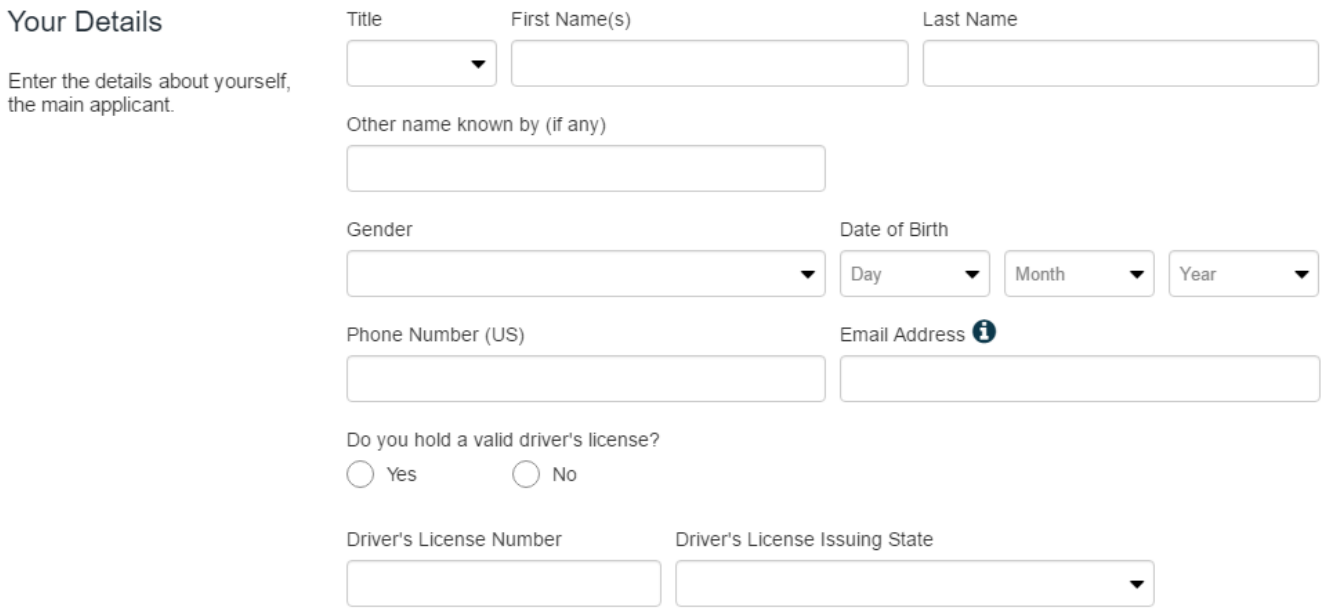
Set up another rule to make Total Expenses read only.



Click **Save**.

Step 4 Add the Driver License Components to the Your Details Section.

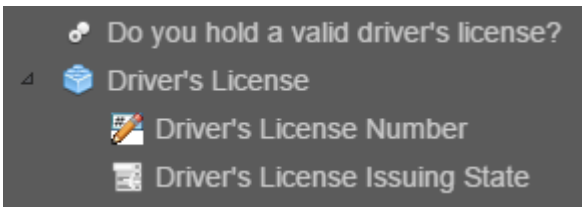
Add the Driver License components shown below.



Click **Save**.

Step 5 Create a Rule for Driver License

Wrap the Driver's License Number and Driver's License Issuing State components into a block, then add a Show If rule so the components only display when the user selects Yes to having a driver's license.



Click **Save**.

> Important: you should wrap Driver's License Number and Driver's License Issuing State in a block before creating the rule. This way you only have to apply the rule to the block, rather than applying to individual components.

Extra Help

Need some extra help?

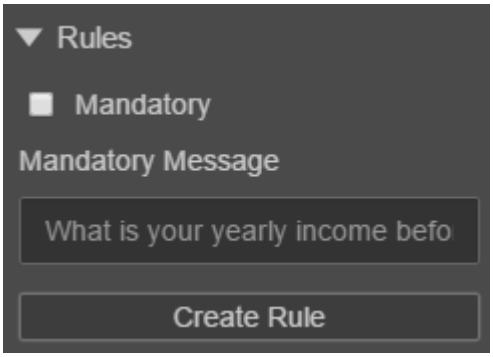
Use the information below to assist you in completing the exercise steps.

Create a Show If Rule

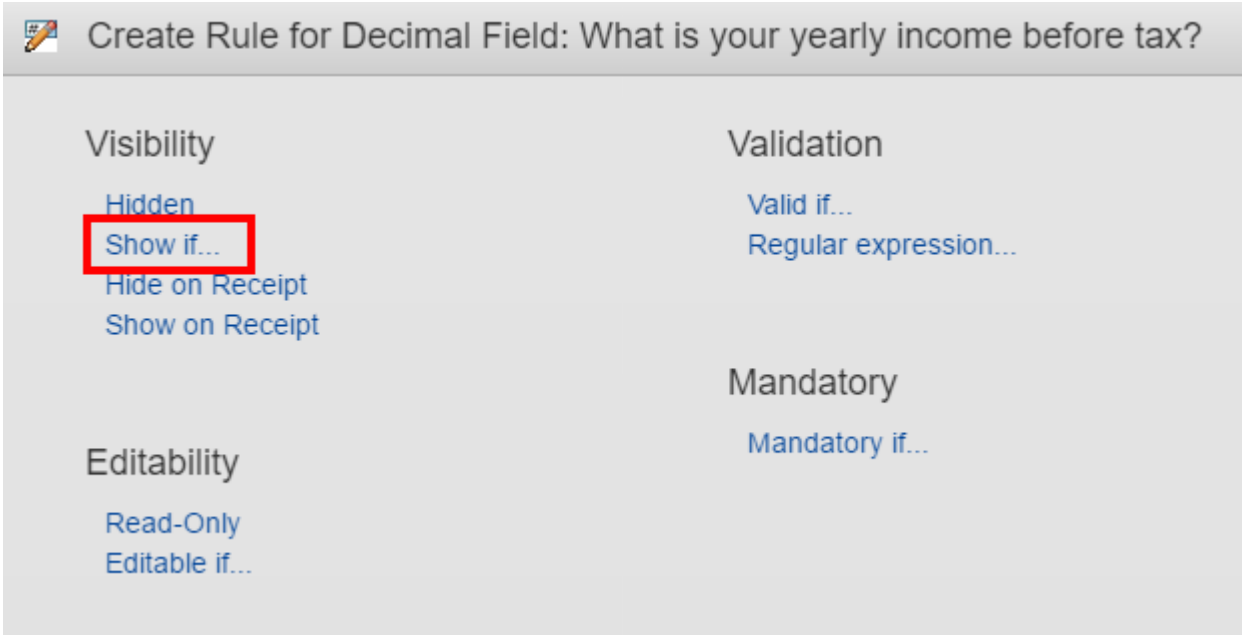
A Show If rule allows you to show or hide a component based on the selection of another component.

Select the component you want to show or hide.

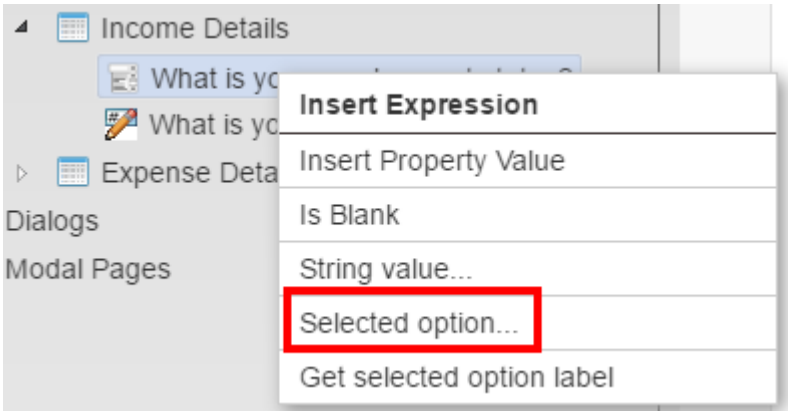
Click Create Rule in the Rules section (Properties panel).



Select Show If.

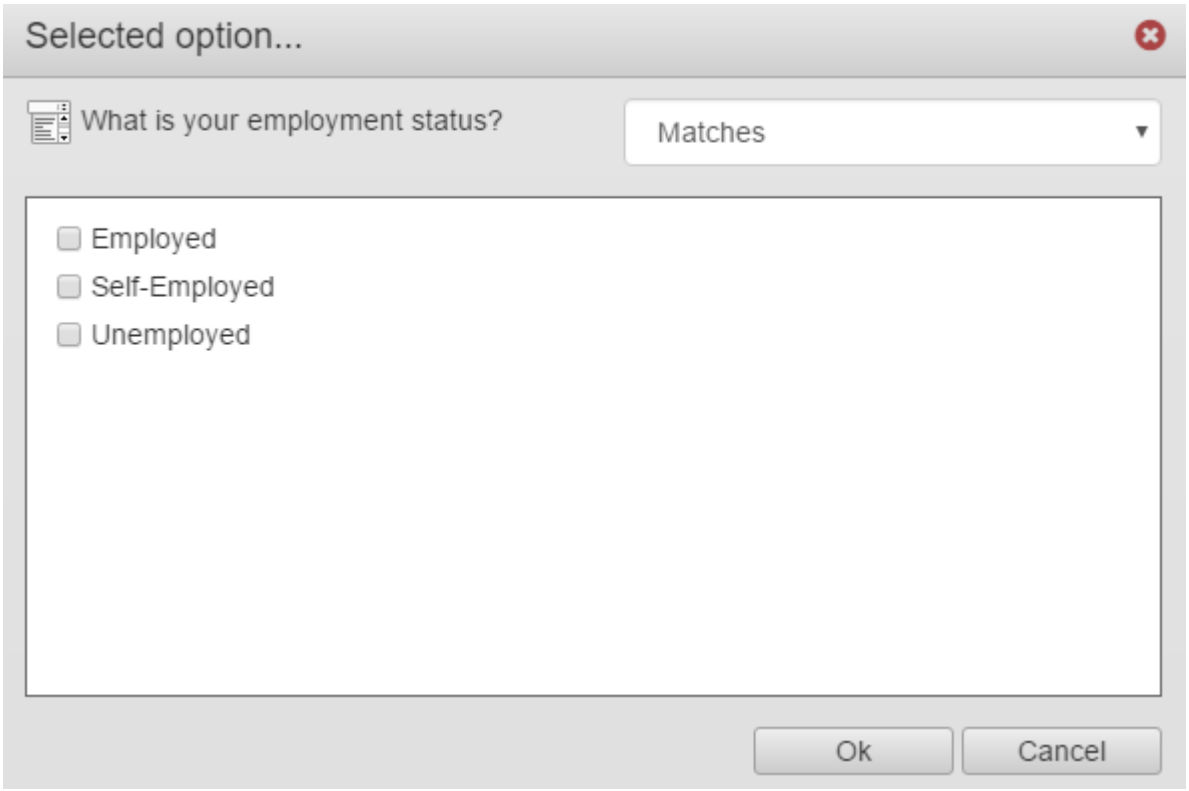


Right-click the component that determines if the selected component is displayed or hidden and click Selected option.

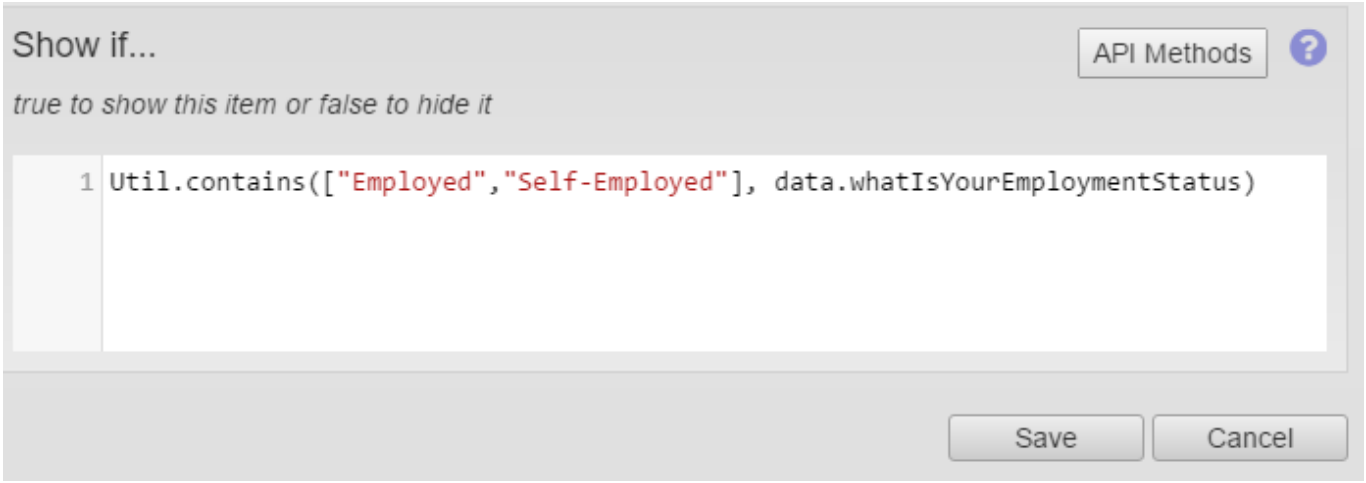


In the top right corner of the Selected option window, select if the option should match or not match, then select the relevant options.

Click OK.

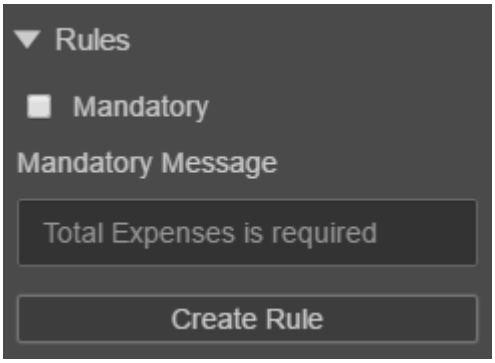


The script for the rule and selected components is displayed. Click Save.

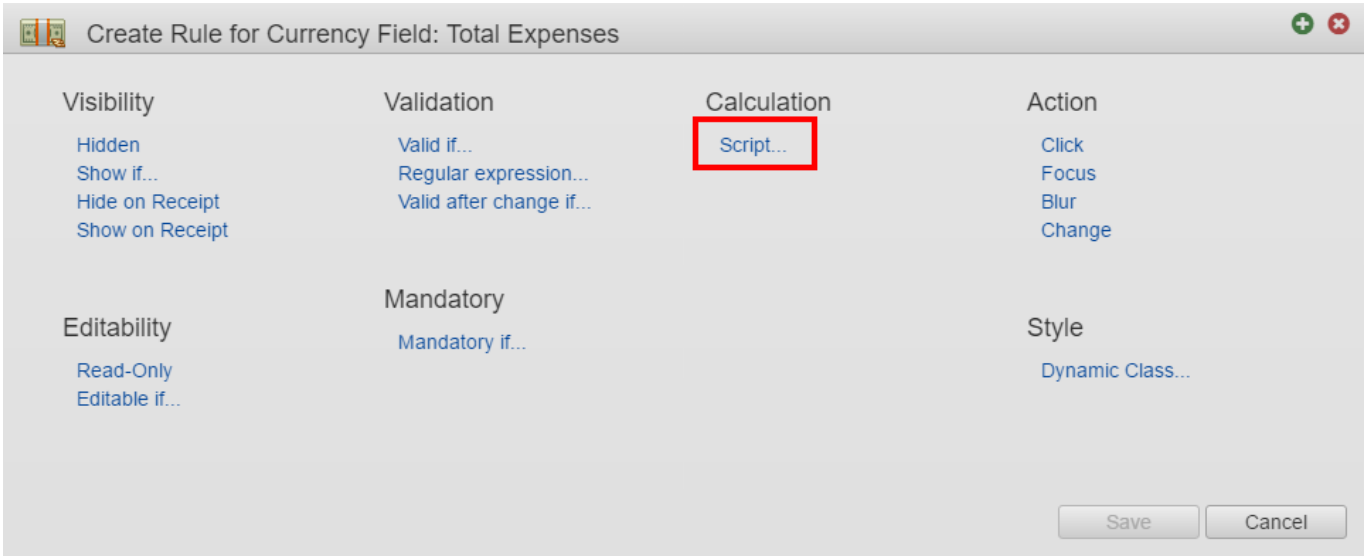


Create a Script Rule

A Script rule will perform calculations based on selected components. Select the component where the calculation will be displayed. Click Create Rule in the Rules section (Properties panel).

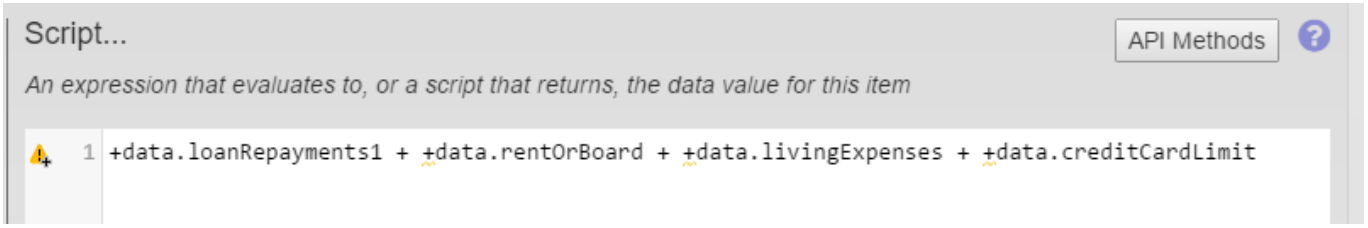


Click Script.



Double click each component you want to use in the calculation. Include a plus sign (+) between the components.

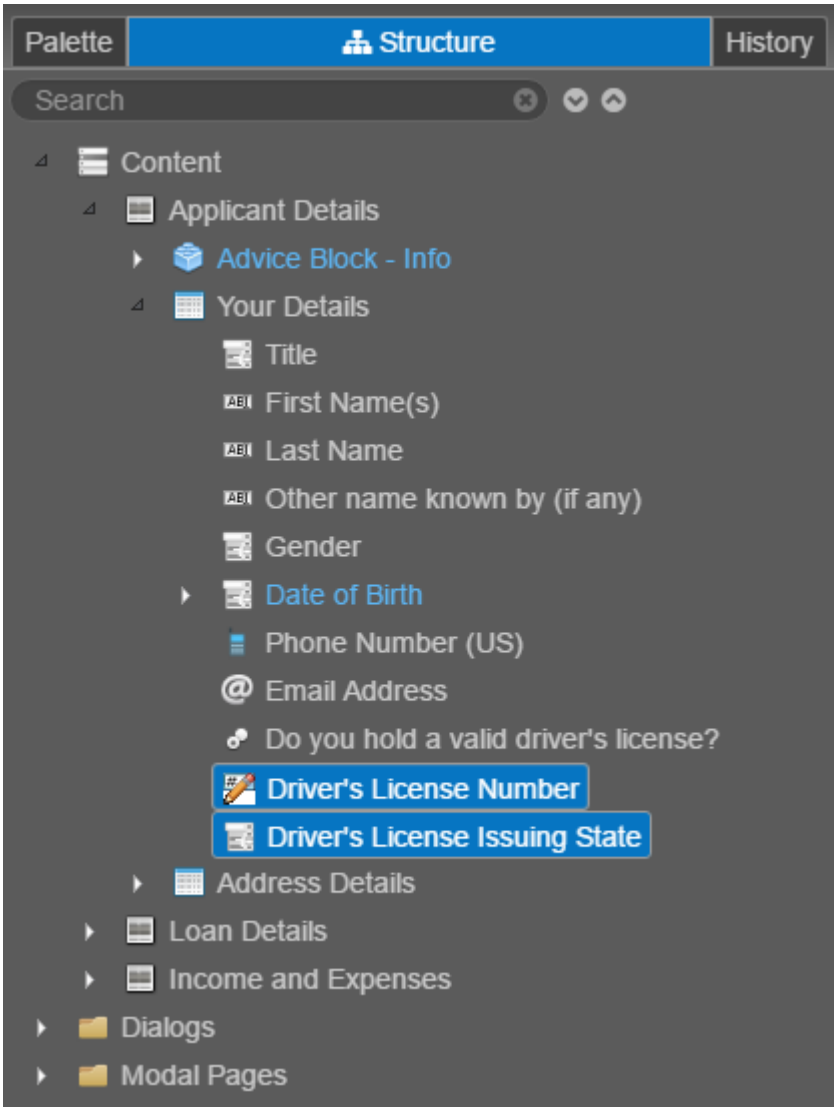
The selected components will display in the list. Click **Save**.



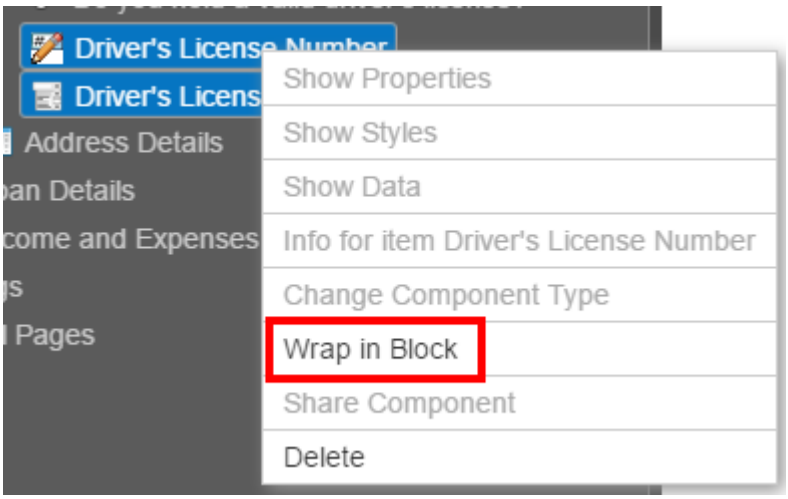
Wrap Components in a Block

You can wrap components in a block to easily manage related components.

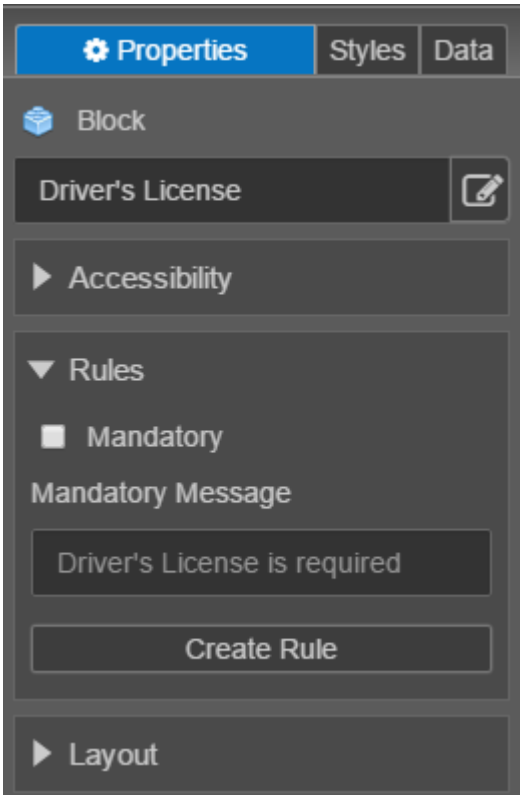
Select the components in the Structure panel you want to wrap in a block.



Right-click one of the selected components and click Wrap in Block.



With the block selected, use the Properties panel to rename the block.


Properties
Styles
Data
Block
Driver's License
Accessibility
Rules
Mandatory
Mandatory Message
Driver's License is required
Create Rule
Layout

Two Level Navigation Exercise v5.0

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for working with two level navigation. During this exercise you will create page groups, which will allow you to use two level navigation within your form.

Do your best to follow the provided information and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises > Two Level Navigation Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Open Two-Level Navigation Begin

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Double-click **Two Level Navigation Begin**.
5. Click **Edit Version** to open the form

Step 2 Add Page Groups to the Form

Add Page Groups to the form, then move the relevant pages into each group. Your Structure panel should match the After image below.

Before After





To add a Page Group, you need to drag the component from the Palette panel onto the Structure panel.

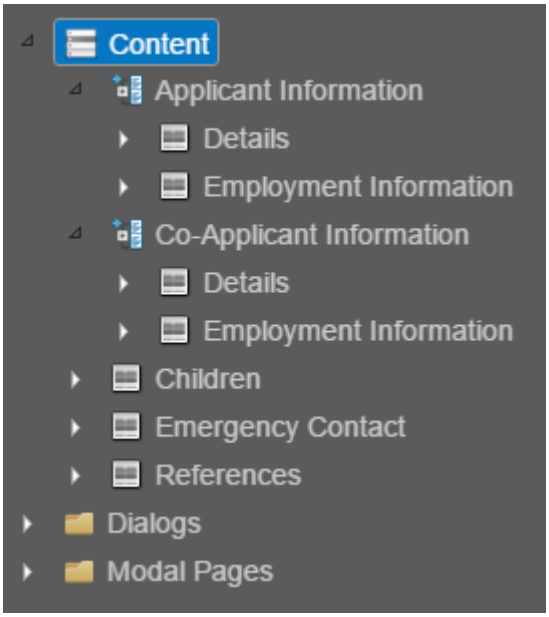
Page groups cannot be dragged directly onto a form or template.

> Notice: some of the pages have been rearranged.

Click **Save**.

Step 3: Rename Pages

Rename the Pages added to each Page Group. Your Structure panel should match the image below.



Step 4 Set Second Level Navigator to Equal Width

Change the width of the Pages on the second level navigator so that they are equal. Your second level navigator should match the image below.





To change the width of the Second Level Navigator:

1. Click **Form Options**.
2. Select **Navigator Second Level.**
3. Select **Equal Width.**
4. Check **Equal Width**.
5. Click **Save**.

Extra Help

Need some extra help?

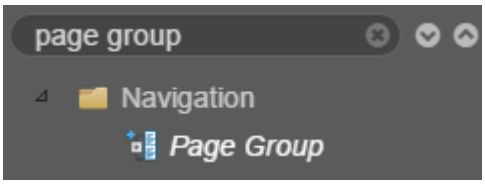
Use the information below to assist you in completing the exercise steps.

Using a Page Group

Page Groups let you group pages together. Adding page groups automatically creates a two level navigation. Each page group is displayed on the first level navigator and any pages in the page group are displayed on the second level navigator.

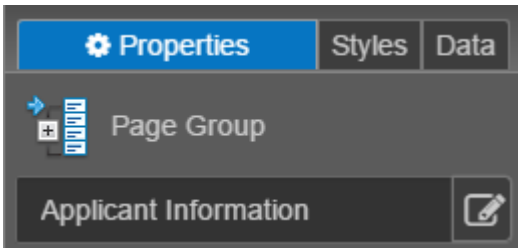
Use Search to find the Page Groups.

Drag the component onto the Structure panel.

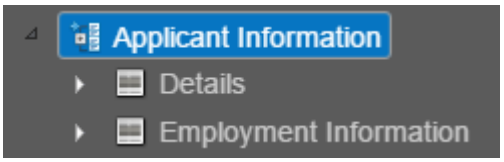


Use the Properties panel to rename the page group.

The Page Group label is automatically used for the first level navigator. If you want the label and menu text different, change the menu text in the Options section on the Properties panel.



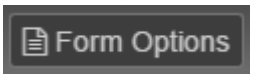
In the Structure panel, drag the pages you want grouped onto the page group component.



Set the Second Level Navigator to equal widths

By default, the width of Second Level Navigator menu items is determined by the length of the menu text. You can change this, so each menu item takes up the same width.

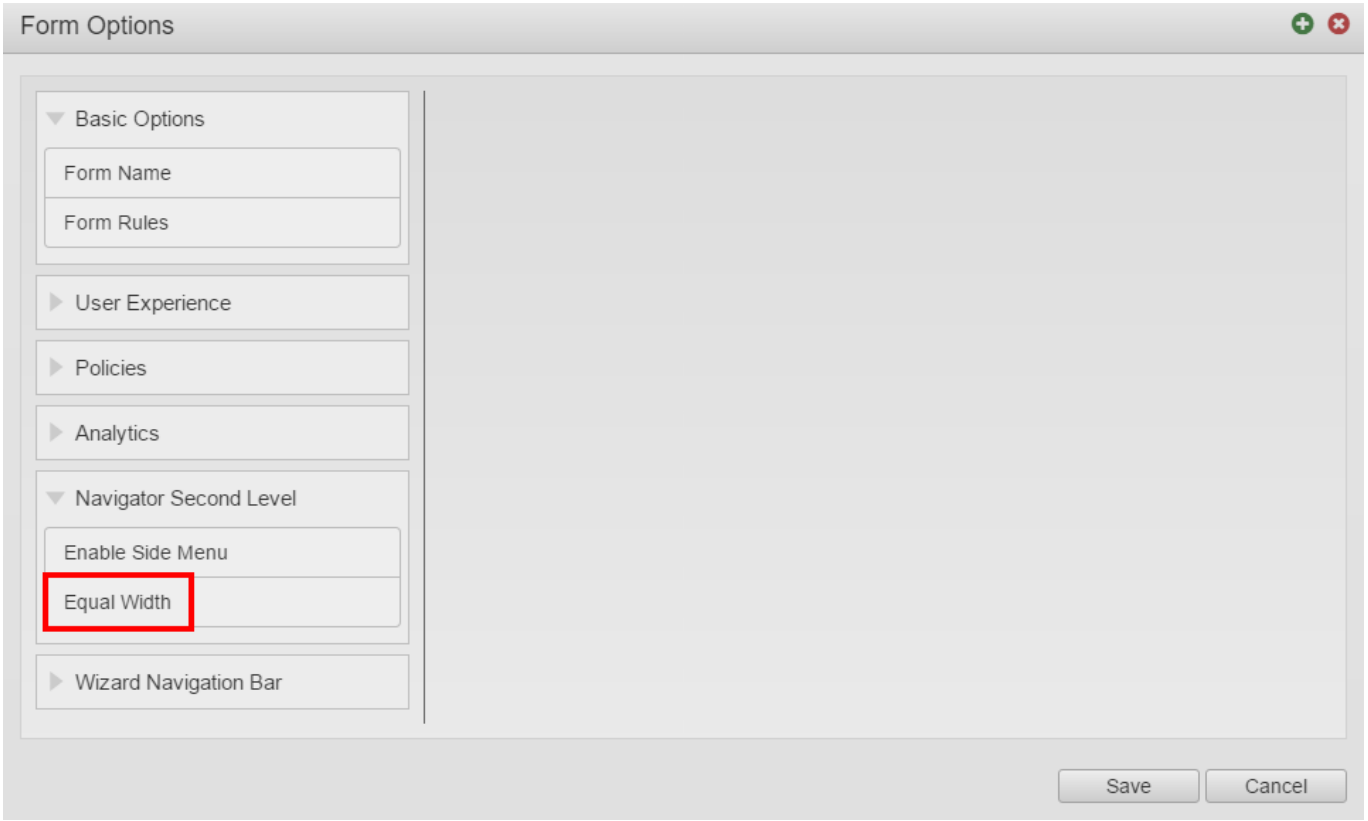
Click the Form Options button (towards the top of the Design window).



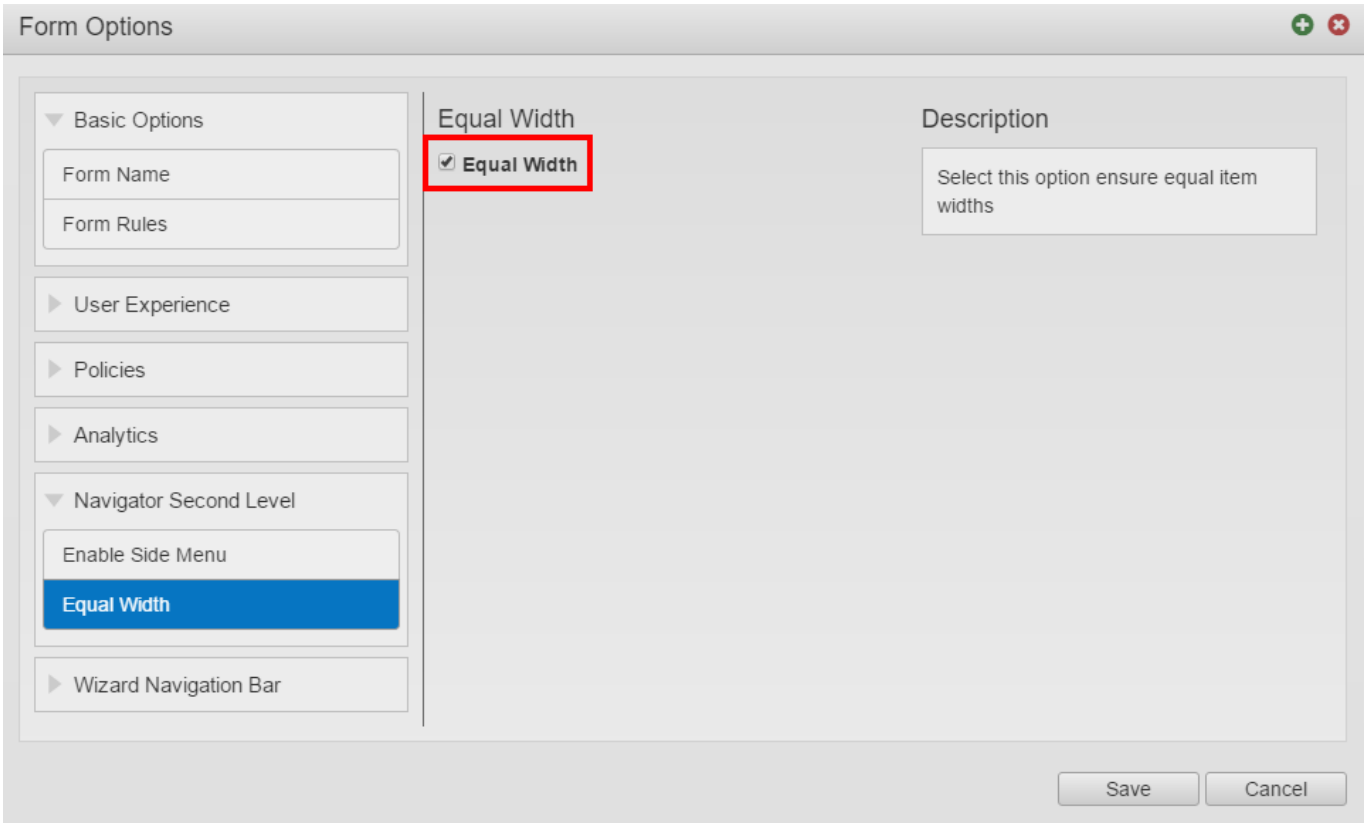
Expand Navigator Second Level.



Select Equal Width.



Check Equal Width, then click Save.


Form Options
Basic Options
Form Name
Form Rules
User Experience
Policies
Analytics
Navigator Second Level
Enable Side Menu
Equal Width
Wizard Navigation Bar
Equal Width
Equal Width
Description
Select this option ensure equal item widths
Save
Cancel

Validation Rules Exercise v5.0

Journey Platform | Form Builder | v5.1 & Higher

This exercise will allow you to practice your skills for creating validation rules. During this exercise you will create a new form and add components. You will then create rules on the added components.

Do your best to follow the information provided and complete each step.

If you get stuck, you can always re-watch any of the online training modules. This is a good way to refresh your form building skills. You'll also find Extra Help at the end of this exercise, which contains additional information to help you complete the exercise.

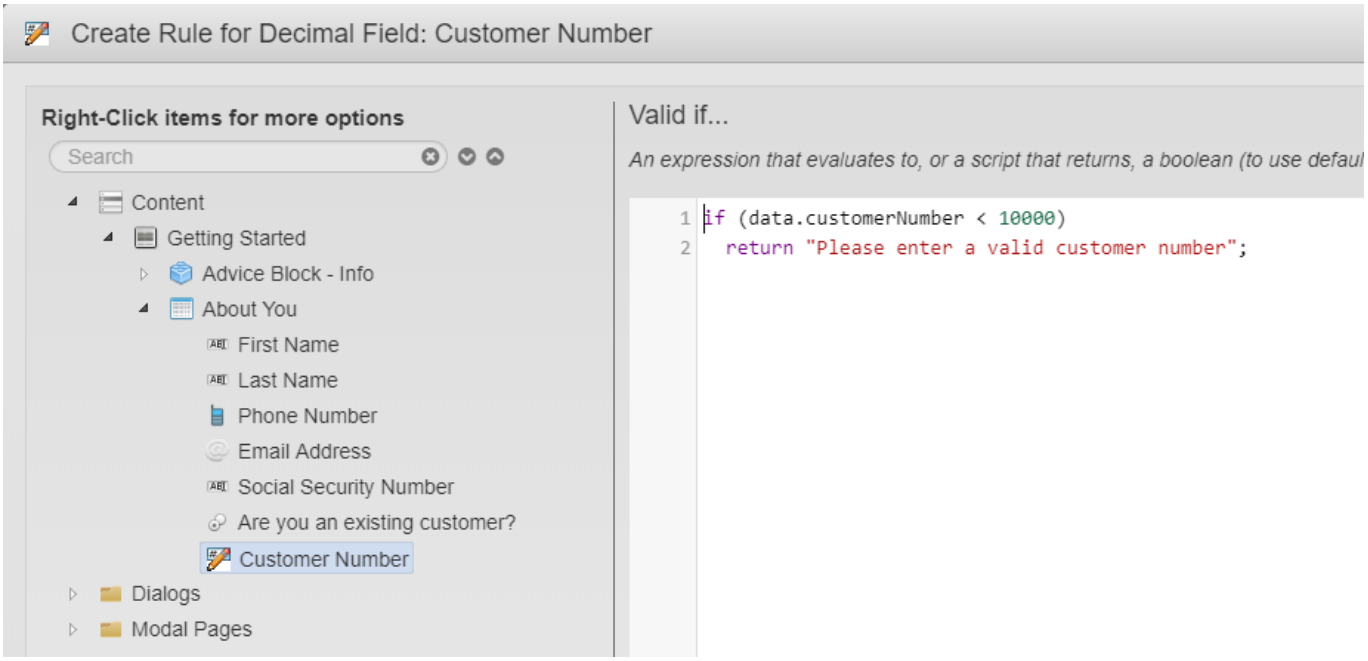
You can find a sample of the completed exercise form in the Maestro Exercise Project: Maestro Training Exercises > Validation Rules Finish. If you have not already imported the Maestro Exercise Project please visit Exercise Setup - Importing the Maestro Exercise Files for more information.

Step 1 Open Validation Rules Begin

1. Expand the **Organization**.
2. Expand the **Project**.
3. Select **Forms**.
4. Double-click **Validation Rules Begin**.
5. Click **Edit Version** to open the form.

Step 2 Create a Valid If Rule

Use the Valid If rule for the Customer Number. Create a rule where the Customer Number is valid if greater than 10000.



Customer Number contains a visibility rule. You won't see this field in preview until you select Yes for the question 'Are you an existing customer?'

Step 3 Create a Regular Expression Validation Rule

You can search the Regular Expression Library (http://regexlib.com) to locate predefined expressions to use in your Maestro form.

Setup a Regular Expression rule to validate the Social Security Number component. An example expression is included in the image below.



Example Regular expression: `^(?!219-09-9999|078-05-1120)(?!666|000|9\d{2})\d{3}-(?!00)\d{2}-(?!0{4})\d{4}$`

Example Match Fails message: Please enter a number using this format ###-##-####

Once the validation rule is created, click **Save** and **Preview** the form.

Try entering this number first: 123-4-5678

Then try this number: 123-45-6789

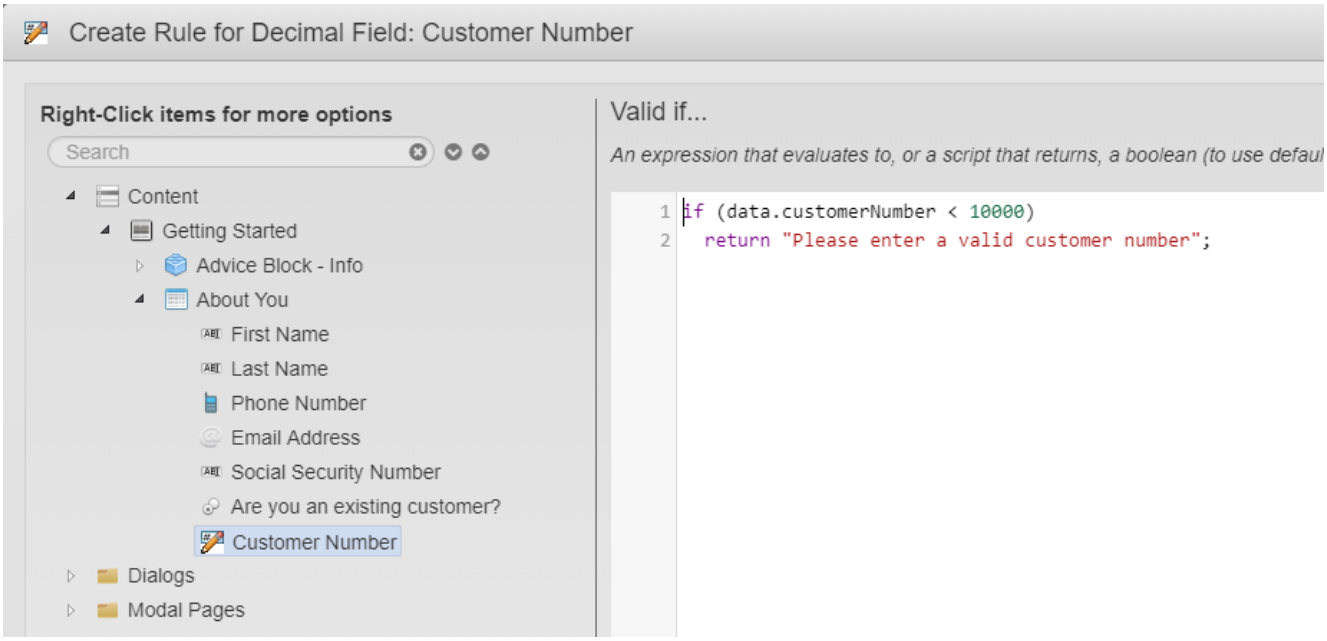
Extra Help

Need some extra help?

Use the information below to assist you in completing the exercise steps.

Create a Valid if rule

1. Select the component to contain the validation rule.
2. Click **Create Rule** then select **Valid if**.
3. Type `if (`.
4. Double-click the (Customer Number) component to add `data.customerNumber` to the script window.
5. Type `< 10000)` then press Enter.
6. Type `return "Please enter a valid Customer Number.";`.



7. Click **Save**.

Create a Regular Expression rule

1. Launch the Regular Expression Library - http://regexlib.com
2. Locate the pre-defined expression you want to use.
3. Copy the expression, then return to the Maestro form.
4. Select the component to contain the validation rule.
5. Click **Create Rule** then select **Regular Expression**.
6. Paste the pre-defined expression in the **Must Match this expression** box.
7. Enter an appropriate failure message.


Create Rule for Text Field: Social Security Number
Regular expression...
This item will be valid if it matches the provided regular expression
Must match this expression
^(?!219-09-9999|078-05-1120)(?!666|000|9\d{2})\d{3}-(?!00)\d{2}-(?!0{4})\d{4}$
Expression Flags
Ignore case
Multi-line
Global
If the match fails, display this message
Please enter a number using this format ###-##-####
Save
Cancel

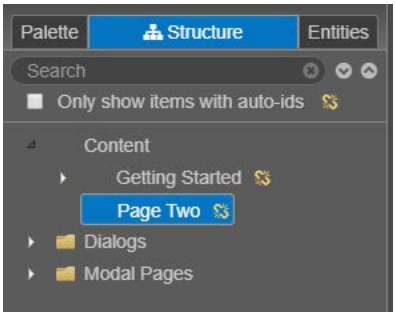
8. Click **Save**.

Exercise - Assign an Entity to multiple items

You can assign an Entity to multiple items at once using the Structure panel.

Step 1: Add a Section and Components

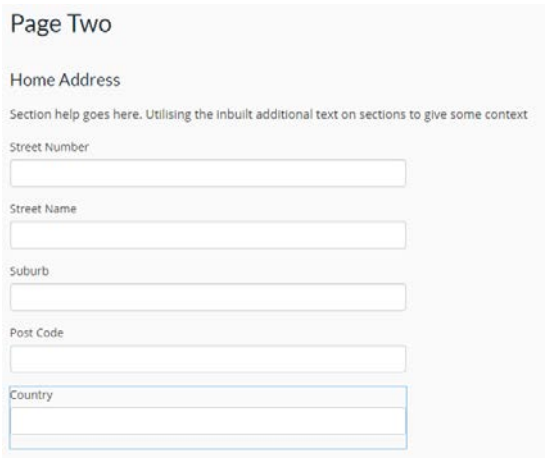
1. To go to Page Two, display the Structure panel and click Page Two (for the 'Exercise – Entities' form)



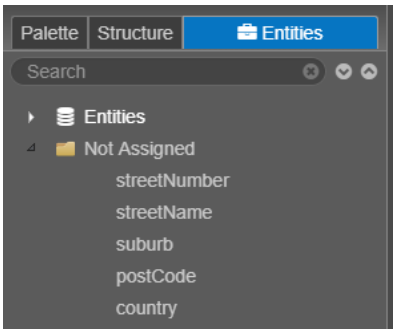
2. Add a new section to form called 'HomeAddress'



3. Add (and name) the following 'text field' components to the 'Address' Section:
   - StreetNumber
   - StreetName
   - Suburb
   - PostCode
   - Country



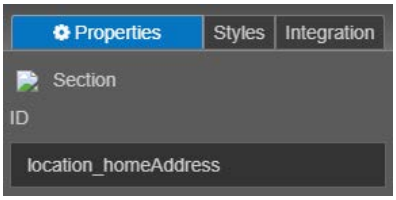
4. Display the Entities panel and confirm the new components are 'Not Assigned' to any Entities



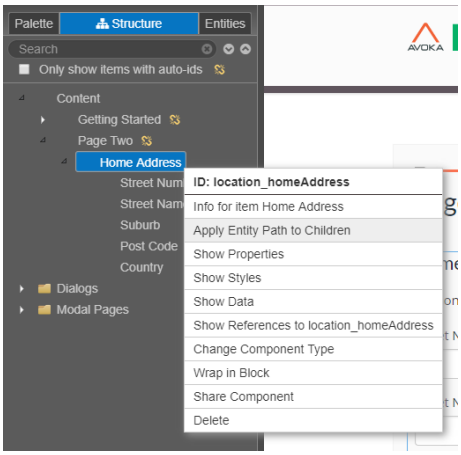
Step 2: Assign an Entity to multiple items

To apply the ‘location’ Entity to the new components, we need to assign it to the ‘Home Address’ section first, i.e. where all the components are located.

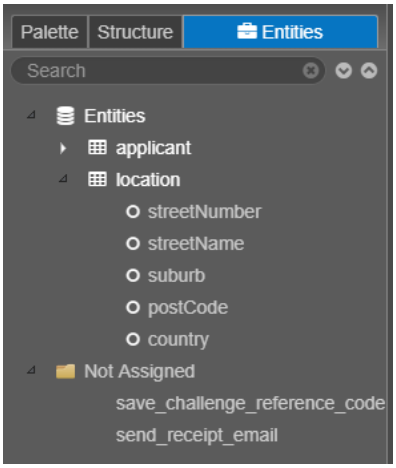
1. Select the ‘Home Address’ Section (the ID Property should show ’homeAddress’)
2. Use the ID property field to assign the ‘location’ Entity to the Section



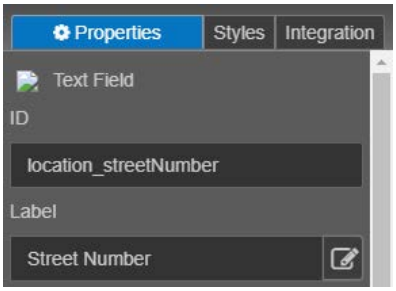
3. Display the ‘Structure’ panel > right-click the ‘homeAddress’ Section and choose ‘Apply Entity Path to Children’



4. Display the ‘Entities’ panel and confirm the components are assigned to the ‘location’ Entity



5. Select each component in the form and confirm the IDs have all been assigned to the ‘location’ Entity



6. Click Save

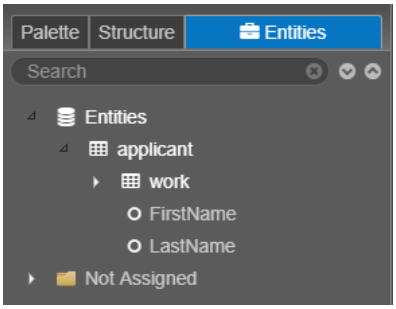
Close the form (Browser Tab)

Exercise - Create an Entity (Entity panel)

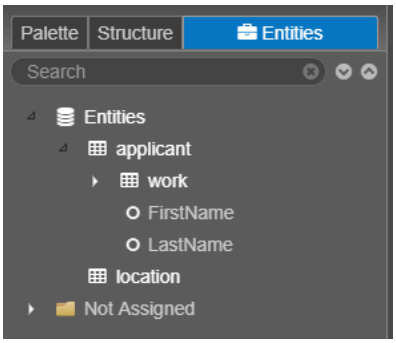
You can create a new Entity in the Entities panel by ‘right-clicking’ where you want the new Entity to be placed.

Step 1: Create an Entity (Entities panel)

1. Make sure the ‘Exercise – Entities’ form is displayed in Design view
2. Make sure the Entities panel is visible



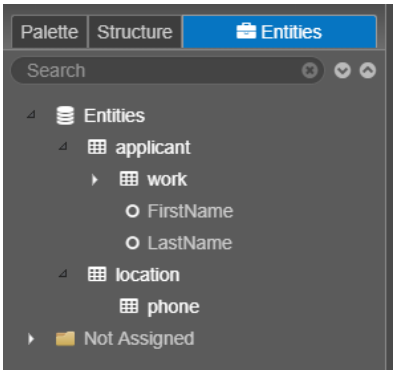
3. Right-click the ‘Entities’ icon
4. Choose ‘Add Entity’
5. Note: the ‘Create New Entity’ dialog is displayed
6. Enter ‘location’ for the Entity name and click Save
7. The new ‘location’ Entity now appears in the panel



Step 2: Create a ‘nested’ Entity (Entities panel)

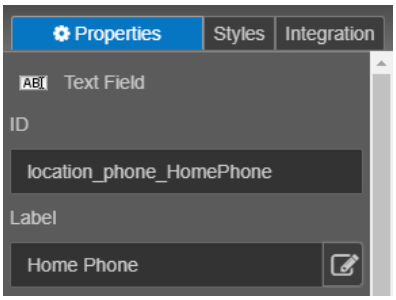
You can also create a nested Entity in the Entities panel.

1. Right-click the ‘location’ Entity
2. Click ‘Add Entity’
3. Enter ‘phone’ for the new nested Entity name and click Save
4. Expand the ‘location’ Entity to display the new nested ‘phone’ Entity

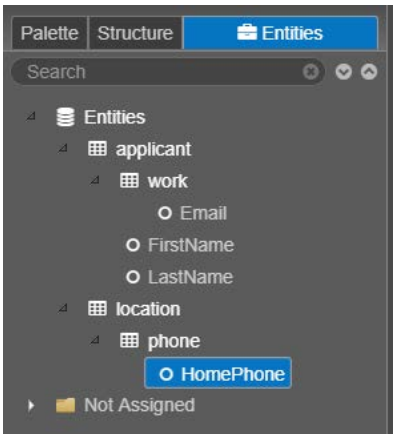


5. Add a new ‘Text Field’ component beneath ‘Email’

6. Use the ID Property field to add this to the nested ‘phone’ Entity and name it ‘HomePhone’



7. Display the ‘Entities’ panel to confirm the ‘Home Phone’ component is related to the nested ‘phone’ Entity



Click Save

Exercise - Create an Entity (ID field)

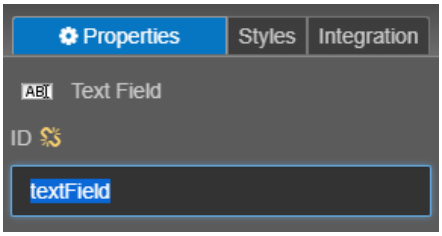
This exercise will demonstrate how to create an Entity directly in the ID field.

Step 1: Create a form

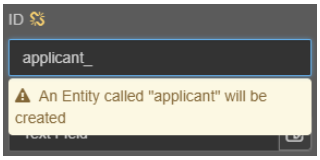
1. Make sure the Navigator Panel is visible
2. Select Forms
3. Click ‘Create Form’
4. Enter the follow details:
   - Form Name: Exercise - Entities
   - Description: This form is to demonstrate how to work with Entities.
   - Template: Training Template
5. Click ‘Create’
6. Note: a new form should display (new window tab) in Design view

Step 2: Create an Entity

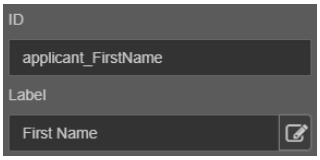
1. Make sure the Palette panel is visible
2. Add a new ‘Text Field’ into the ‘About You’ section
3. Notice: the focus is set by default to the ID property (Properties panel)



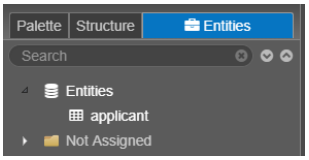
4. To create a new Entity, enter ‘applicant’ followed by an underscore, ‘_’
5. Notice: after you enter underscore (_) an inline warning displays beneath ‘An Entity called applicant will be created’



6. Enter ‘FirstName’ as the ID (after the underscore) then press TAB or click away.
7. Notice: the Label Property displays as ‘First Name’ (without the Entity name)



8. Display the ‘Entities’ panel and notice the ‘applicant’ Entity with one related ‘FirstName’ component

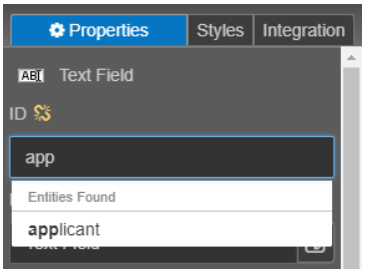


9. Click Save

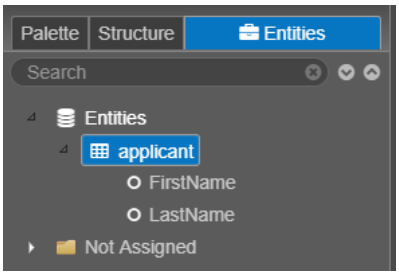
Step 3: Assign an Entity

An existing Entity can be assigned to a new component using the ID field.

1. Add another ‘Text Field’ component beneath ‘First Name’
2. Enter the first few letters of ‘applicant’ in the ID property field, i.e. ‘app’
3. Notice: as you type, any matching Entities will display in an inline dropdown menu



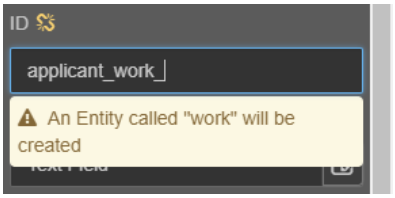
4. Select ‘applicant’ to populate the ID Property field
5. Enter ‘LastName’ (after the underscore) for the ID name.
6. Notice: the label name displays as ‘Last Name’ only
7. Display the ‘Entities’ panel and confirm the ‘applicant’ Entity now has two related components



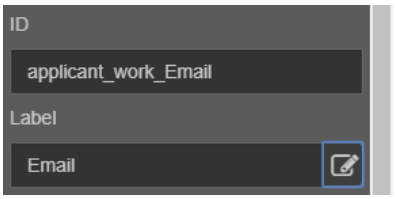
Step 4: Create a nested Entity

You can create Entities within Entities, i.e. a nested Entity.

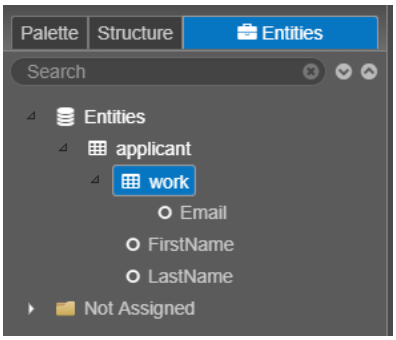
1. Add another new ‘Text Field’ beneath ‘Last Name’
2. Enter the first few letters of ‘applicant’ in the ID property field
3. Select the ‘applicant’ Entity
4. Enter ‘work_’ as the new nested Entity, e.g. ‘applicant1_work_’
5. Notice: after you enter the second underscore (_) an inline warning displays beneath ‘An Entity called work will be created’



6. Enter ‘Email’ for the ID name, e.g. ‘applicant1_work_Email’
7. Press Tab or click away from the ID property field
8. Notice: the label name displays as ‘Email’ only



9. Display the ‘Entities’ panel and confirm the nested ‘work’ Entity with one related component



10. Click Save

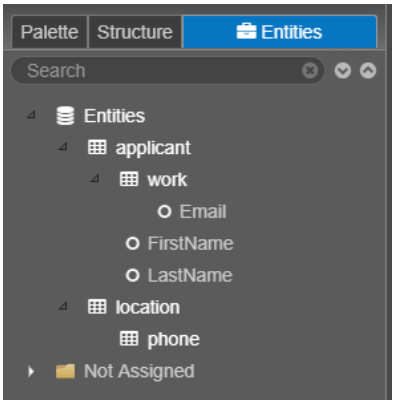
Please note: any existing IDs with an underscore will not automatically cause Entities to be created. Entities will only be created when you modify an existing ID. Any underscores followed by one or more digits (like suffixes applied by auto-generated IDs) will also be ignored.

Exercise - Delete an Entity

You can delete Entities using the Entities panel.

Please note: You can't delete an Entity if any component IDs use that Entity.

1. Your task is to delete the 'phone' Entity, but you'll need to delete or re-assign any related components.
2. For this exercise, delete the 'Home Phone' component on the form
3. Notice: the 'phone' Entity no longer has any related component IDs



4. Right-click the 'phone' Entity and choose 'Delete Entity'
5. Click 'OK' to confirm
6. The Entity has now been deleted from the Entities panel
7. Click Save

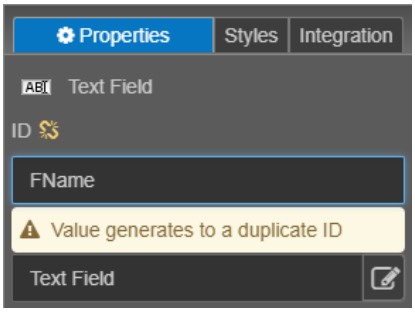
Exercise - Duplicate IDs

This exercise will demonstrate the new duplicate ID functionality. You will:

1. Create a duplicate ID and ‘Review’
2. Create a duplicate ID and ‘Accept’

Step 1: Add a component with a duplicate ID and ‘Review’

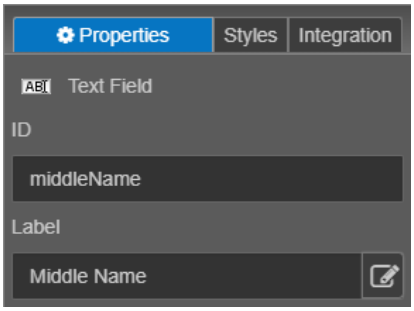
1. Note: Make sure the form you created for the 'ID Management exercise' is open
2. Add a new ‘Text Field’ into the ‘About You’ section (beneath ‘First Name’)
3. Update the ID Property to ‘FName’
4. An inline warning confirms ‘Value generates to a duplicate ID’
   Note: IDs are case sensitive - if you enter 'fname' instead of 'FName', the inline warning will not display



5. At this point, you can either modify the ID property directly in the field
6. Or if you ‘TAB’ (or click away), a popup will prompt you to ‘Review’ the duplicate ID or ‘Accept’ a suffix



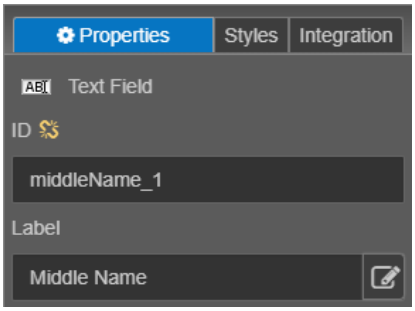
7. For this example, click ‘Review’ and update the ‘ID’ property to ‘MiddleName’



8. Choosing ‘Review’ would let you modify the ID property so that it doesn't cause a duplicate.

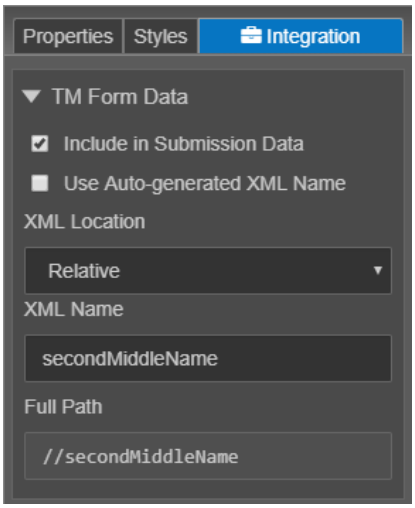
Step 2: Add a component with a duplicate ID and ‘Accept’

1. Add a new ‘Text Field’ into the ‘About You’ section (beneath ‘Middle Name’)
2. Update the ID Property ‘MiddleName’ and TAB away
3. For this example, click ‘Accept’ in the popup
4. Choosing ‘Accept’ appends a suffix to the ID Property (i.e. middleName_1)



5. Click ‘Save’
6. Close the form, i.e. the browser tab

Please Note: If you choose ‘Accept’ for duplicate IDs, the ‘XML Name’ and ‘Full Path’ properties (on the ‘Integration’ panel) keep their default names (i.e. ‘TextField’). To update these properties, you’ll need to uncheck ‘Use Auto-generated XML Name’ and manually update the ‘XML Name’.



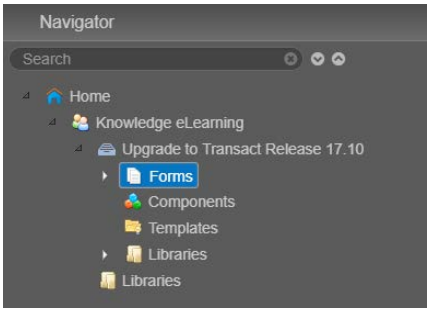
Exercise - ID Management

This exercise will demonstrate the updated functionality for component IDs. You will:

- create a new form
- add a new component
- update the component ID

Step 1: Create a new form

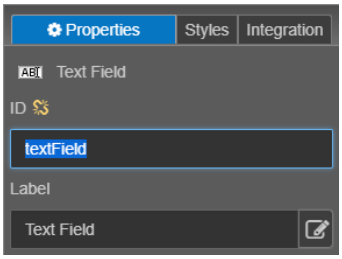
1. Expand the ‘Organization’ (Navigator Panel)
2. Expand the ‘Project’ (Upgrade to Transact Release 17.10)
3. Select ‘Forms’



4. Click the ‘Create Form’ button
5. Enter the following details:
   - Form Name: Exercise - ID Management
   - Description: This form is to demonstrate the new ID features.
   - Template: Training Template
6. Click the ‘Create’ button
7. Note: The form Design view should open in a new Tab

Step 2: Add a new ‘Text Field’ component

1. Make sure the ‘Palette’ panel is displayed
2. Add a new ‘Text Field’ into the ‘About You’ section
3. Note: the initial focus is set to the ID Property field



Step 3: Update the component

1. Update the ‘ID’ Property value to ‘FirstName’
2. Note: The ‘Label’ Property has automatically updated to ‘First Name’
3. Click the ‘Integration’ tab
4. Note: the ‘XML Name’ and ‘Full Path’ properties have updated to the new ‘ID’
5. Click the ‘Properties’ tab
6. Update the ‘ID’ Property to ‘FName’ (then TAB or click away)
7. Note: this time the ‘ID’ Property updates but the ‘Label’ Property remains unchanged. The ‘Label’ only updates when the default ‘ID’ Property is entered.
8. Click the ‘Integration’ tab
9. Note: the ‘XML Name’ and ‘Full Path’ properties have updated to the latest ‘ID’ (i.e. ‘FName’)
10. Click ‘Save’

Please note: If the ‘Use Auto-generated XML Name’ Property is not checked, the ‘XML Name’ and ‘Full Path’ properties won’t update if the ID changes. These properties will need to be manually updated if required.

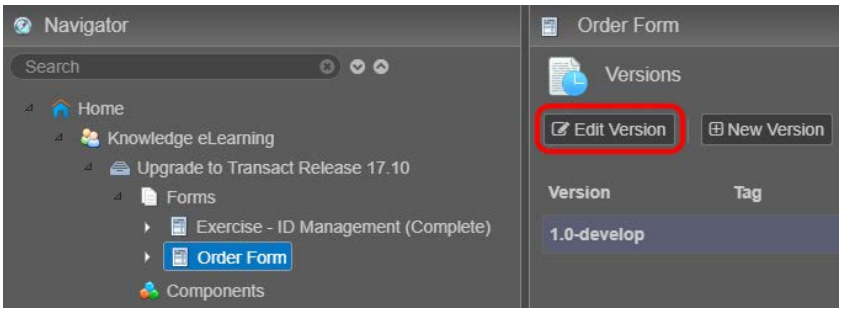
Exercise - Impact Analysis

The following exercise will involve the ‘Order’ form.

Step 1: Open the Order form

From the Maestro Transact Management Dashboard:

1. Expand the ‘Upgrade to Transact Release 17.10’
2. Expand ‘Forms’ and select ‘Order Form’
3. Click the ‘Edit Version’ button

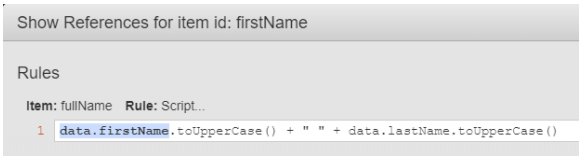


4. The ‘Order Form’ should open in a new browser tab

Step 2: Review the current references

Inspect references for the ‘First Name’ field.

1. Right-click the ‘First Name’ text field (in the form)
2. Choose ‘Show References to firstName’
3. Confirm one existing reference to the ‘FullName’ item

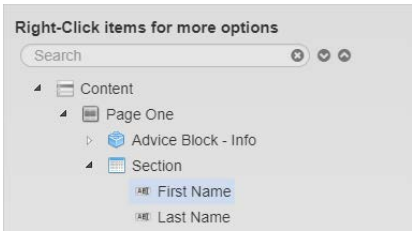


4. Click ‘Close’ (to close the window)

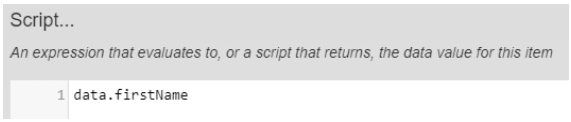
Step 3: Add a new Text Field that references ‘First Name’

Add a new Text Field to ‘Page Two’ that references the ‘First Name’ Text Field

1. Display the ‘Structure’ panel
2. Select ‘Page Two’ from the form structure
3. Re-display the ‘Palette’ panel
4. Add a new Text Field to the Page Two ‘Section’
5. Update the ID Property to ‘FirstNameP2’ and press TAB (focus moves to Label Property)
6. Update the Label Property to ‘First Name’ and press TAB
7. With the new Text Field selected, click ‘Create Rule’ (bottom of Properties panel)
8. Click ‘Script’ (under ‘Calculation’)
9. In the navigator column (on the left), expand ‘Page One’, then expand the first ‘Section’



10. Double-click ‘First Name’ to setup a reference to this component, i.e. ‘data.firstName’

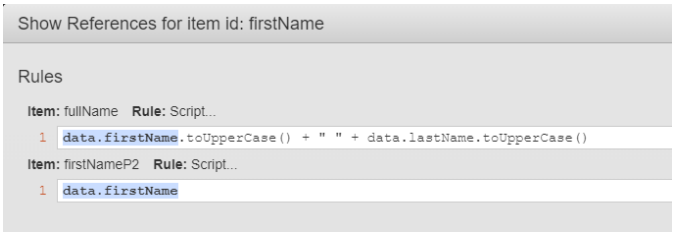


11. Click ‘Save’ (to close the window), then Save the form
12. Click ‘Preview’ to test the new reference, i.e. enter a ‘First Name’ on Page One and confirm it displays in ‘First Name’ on Page Two
13. When you’re ready, switch back to Design view

Step 4: Review the updated references

Inspect the references for the ‘First Name’ field on Page One

1. Display the ‘Structure’ panel
2. Select ‘Page One’ from the form structure
3. Re-display the ‘Palette’ panel
4. Right-click ‘First Name’ and click ‘Show References to ‘First Name’
5. Confirm a second reference to ‘firstNameP2’ (on Page Two) now exists



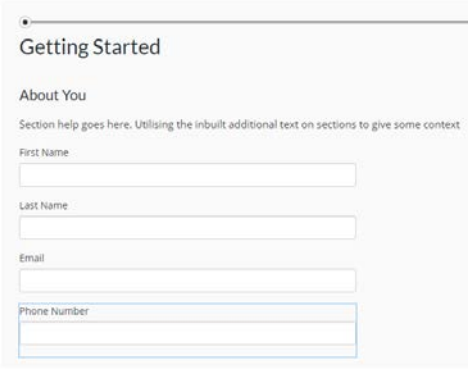
6. Click ‘Close’ (to close the dialog)
7. Click Save then close the form (i.e. the Browser Tab)

Exercise - Semantic Versioning

This exercise will demonstrate how to create multiple semantic versions of a form.

Step 1: Update the 1.0-develop version

1. Select the ‘Exercise - Entities’ form in the Navigator panel
2. Click ‘Edit Version’ to open the form
3. Add a new text field to the ‘Getting Started’ section called ‘PhoneNumber’

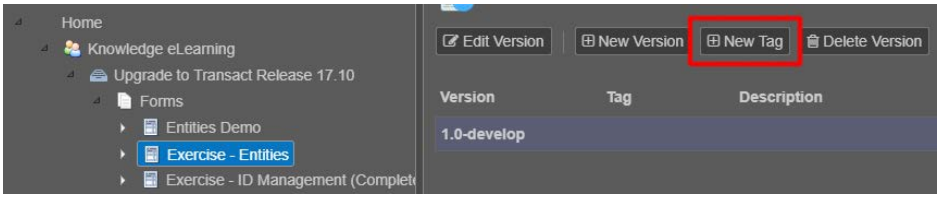


4. Save and Close the form (Browser Tab)

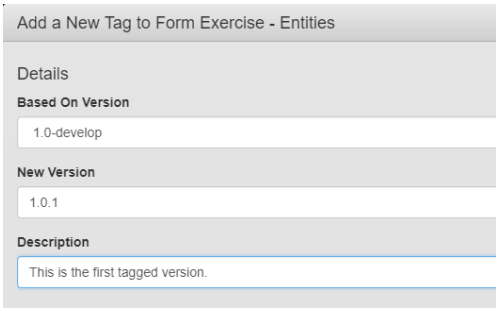
Step 2: Create a ‘Tagged’ version

Create the first read-only tagged version (based on ‘1.0-develop’)

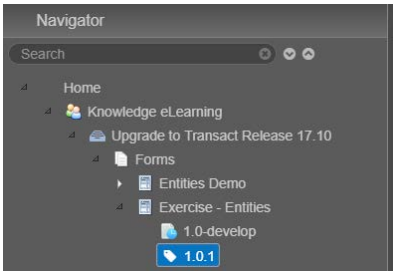
1. Select the ‘Exercise - Entities’ form in the Navigator panel
2. Click ‘New Tag’



3. Confirm the new Semantic Version number is 1.0.1
4. Enter Description: ‘This is the first tagged version.’
5. Click ‘Create’

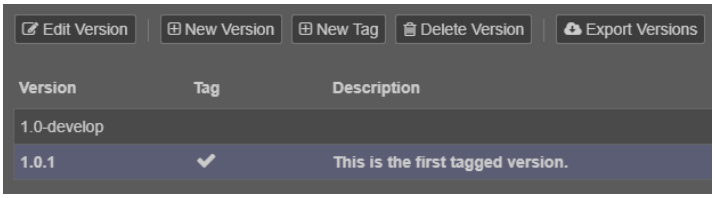


6. Expand the ‘Exercise – Entities’ form in the Navigator panel and confirm the tagged 1.0.1 version is displayed



Step 3: Open the ‘Tagged’ version

1. Select the ‘Exercise - Entities’ form in the Navigator panel
2. Select the ‘1.0.1’ tagged version (note the ‘tick’ icon)

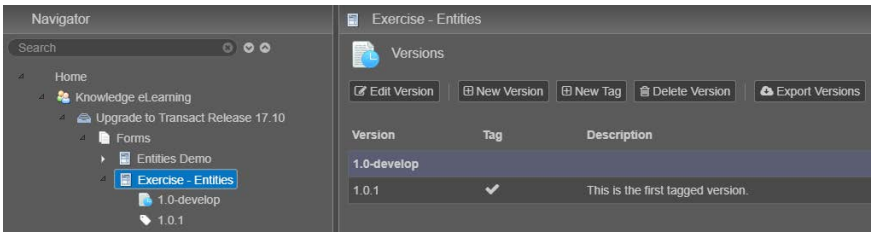


3. Click ‘Edit Version’ to open the form
4. Notice: the Palette panel is empty, i.e. you can’t add new or delete existing components
5. Close the form (Browser Tab)

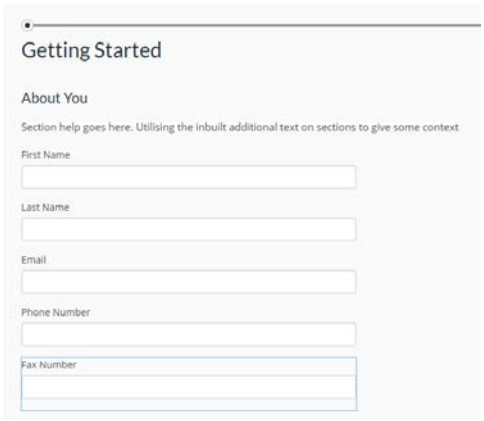
Step 4: Update the 1.0-develop version

Remember: all changes and updates are completed in 1.0-develop

1. Select the ‘Exercise - Entities’ form in the Navigator panel
2. Select the ‘1.0-develop’ version
3. Click ‘Edit Version’ to open the form



4. Add a new text field to the ‘Getting Started’ section called ‘FaxNumber’



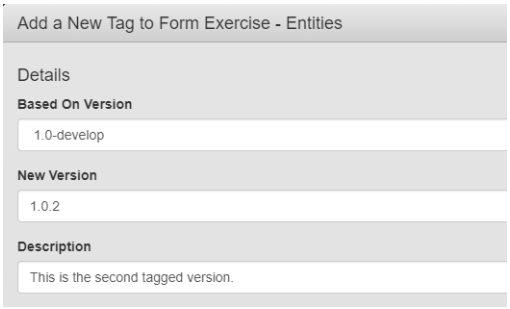
5. Save and Close the form (Browser Tab)

Step 5: Create another ‘Tagged’ version

Create another read-only version (based on ‘1.0-develop’)

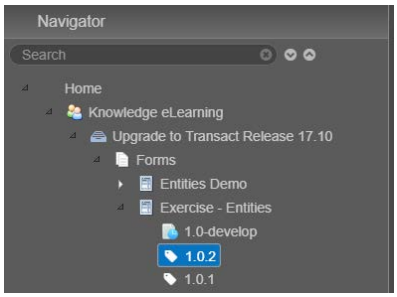
1. Select the ‘Exercise - Entities’ form in the Navigator panel
2. Select the ‘1.0-develop’ version
3. Click ‘New Tag’
4. Confirm the new Semantic Version number is 1.0.2

5. Enter Description: ‘This is the second tagged version.’



6. Click ‘Create’

Expand the ‘Exercise – Entities’ form in the Navigator panel and confirm the tagged 1.0.2 version is displayed



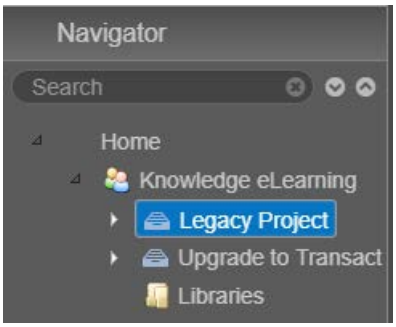
Exercise - Upgrade a project

You can upgrade an existing project to the latest Maestro release from the Management Dashboard.

Step 1: Import a project

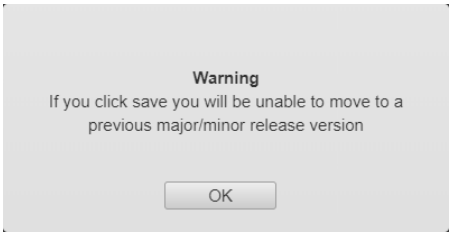
To upgrade a project, you'll need to add a legacy project to your organization first.

1. Download this sample project (zipped folder)
2. Remember where the project is stored, e.g. 'Downloads'
3. Navigate to the Maestro webpage
4. Expand Home (in Navigator Panel)
5. Select the Organization (the Organisation maybe your own name e.g. jsmith
6. Click 'Import Legacy Project'
7. Name the project - 'Legacy Project'
8. Click Import
9. Navigate to the location of the sample project
10. Select the project and click Open
11. The imported project should now appear under your organization



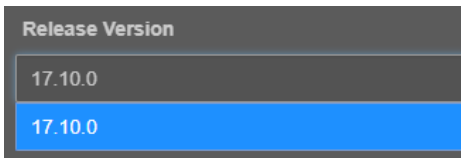
Step 2: Upgrade the project

1. Select 'Legacy Project' project in the Navigator
2. Click the Project Details tab
3. Display the 'Release Version' dropdown and choose '17.10.0'
4. Note: When you select a new version, a warning message confirms you can't change the release version back to a previous version once the Project Details are saved



5. Click OK in the warning message
6. Click Save
7. The project has now been upgraded

Notice: The Release Version dropdown no longer displays any previous release versions

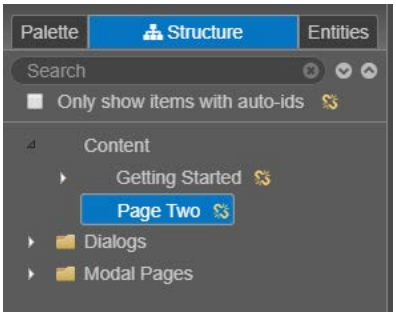


Exercise - Assign an Entity to multiple items

You can assign an Entity to multiple items at once using the Structure panel.

Step 1: Add a Section and Components

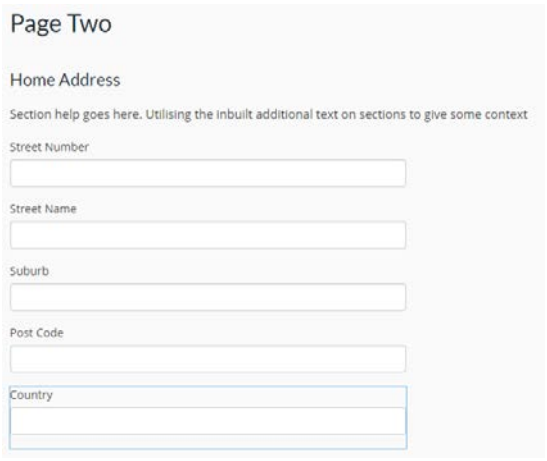
1. To go to Page Two, display the Structure panel and click Page Two (for the 'Exercise – Entities' form)



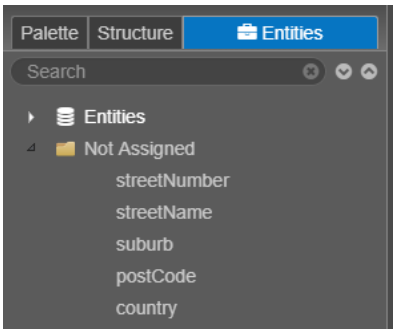
2. Add a new section to form called 'HomeAddress'



3. Add (and name) the following 'text field' components to the 'Address' Section:
   - StreetNumber
   - StreetName
   - Suburb
   - PostCode
   - Country



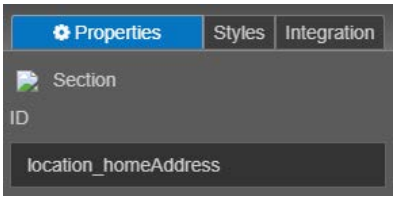
4. Display the Entities panel and confirm the new components are 'Not Assigned' to any Entities



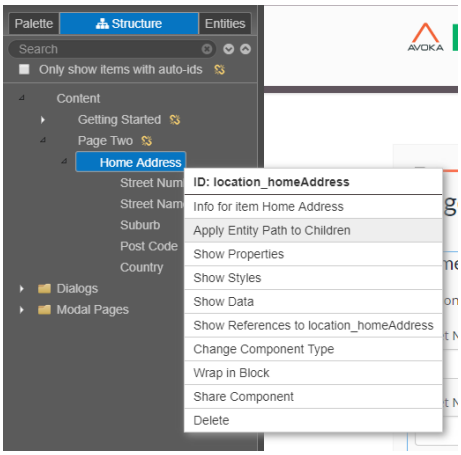
Step 2: Assign an Entity to multiple items

To apply the ‘location’ Entity to the new components, we need to assign it to the ‘Home Address’ section first, i.e. where all the components are located.

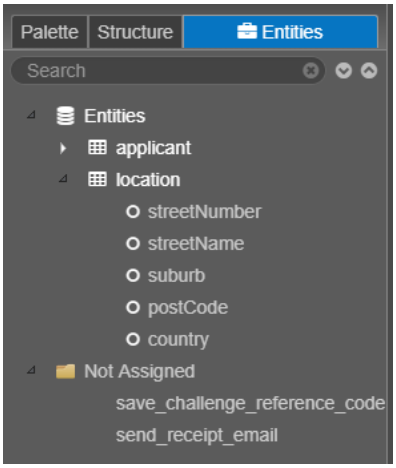
1. Select the ‘Home Address’ Section (the ID Property should show ’homeAddress’)
2. Use the ID property field to assign the ‘location’ Entity to the Section



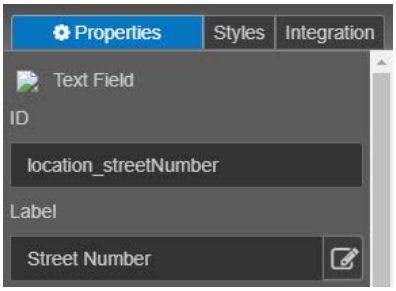
3. Display the ‘Structure’ panel > right-click the ‘homeAddress’ Section and choose ‘Apply Entity Path to Children’



4. Display the ‘Entities’ panel and confirm the components are assigned to the ‘location’ Entity



5. Select each component in the form and confirm the IDs have all been assigned to the ‘location’ Entity



6. Click Save

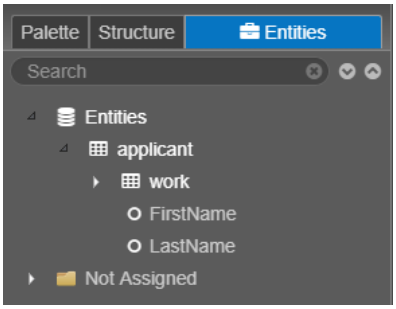
Close the form (Browser Tab)

Exercise - Create an Entity (Entity panel)

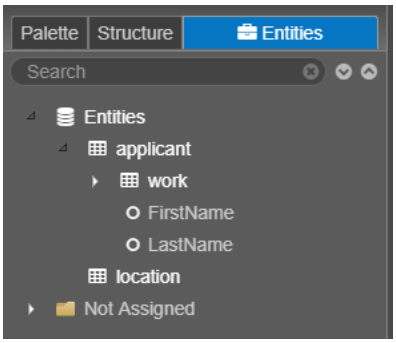
You can create a new Entity in the Entities panel by ‘right-clicking’ where you want the new Entity to be placed.

Step 1: Create an Entity (Entities panel)

1. Make sure the ‘Exercise – Entities’ form is displayed in Design view
2. Make sure the Entities panel is visible



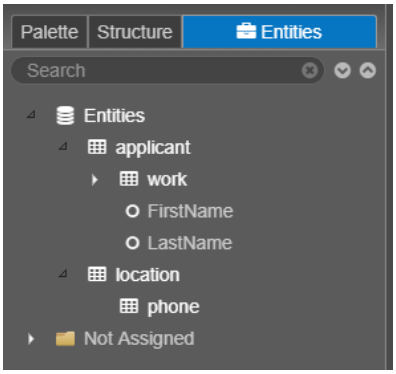
3. Right-click the ‘Entities’ icon
4. Choose ‘Add Entity’
5. Note: the ‘Create New Entity’ dialog is displayed
6. Enter ‘location’ for the Entity name and click Save
7. The new ‘location’ Entity now appears in the panel



Step 2: Create a ‘nested’ Entity (Entities panel)

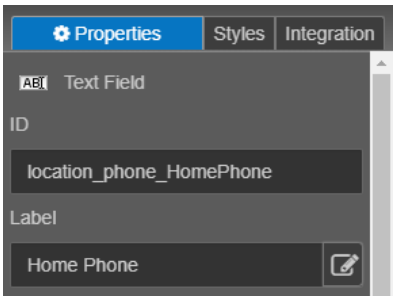
You can also create a nested Entity in the Entities panel.

1. Right-click the ‘location’ Entity
2. Click ‘Add Entity’
3. Enter ‘phone’ for the new nested Entity name and click Save
4. Expand the ‘location’ Entity to display the new nested ‘phone’ Entity

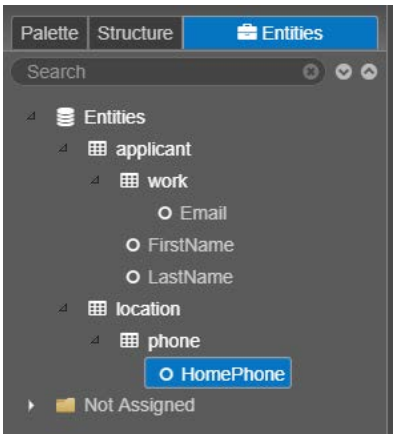


5. Add a new ‘Text Field’ component beneath ‘Email’

6. Use the ID Property field to add this to the nested ‘phone’ Entity and name it ‘HomePhone’



7. Display the ‘Entities’ panel to confirm the ‘Home Phone’ component is related to the nested ‘phone’ Entity



Click Save

Exercise - Create an Entity (ID field)

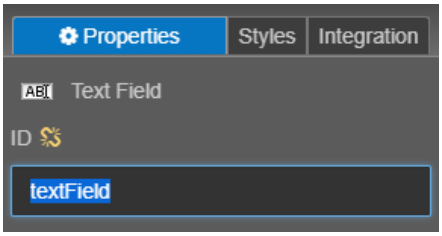
This exercise will demonstrate how to create an Entity directly in the ID field.

Step 1: Create a form

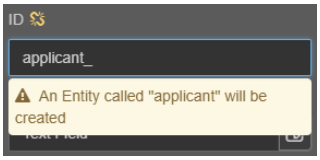
1. Make sure the Navigator Panel is visible
2. Select Forms
3. Click ‘Create Form’
4. Enter the follow details:
   - Form Name: Exercise - Entities
   - Description: This form is to demonstrate how to work with Entities.
   - Template: Training Template
5. Click ‘Create’
6. Note: a new form should display (new window tab) in Design view

Step 2: Create an Entity

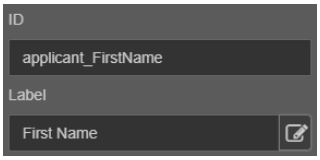
1. Make sure the Palette panel is visible
2. Add a new ‘Text Field’ into the ‘About You’ section
3. Notice: the focus is set by default to the ID property (Properties panel)



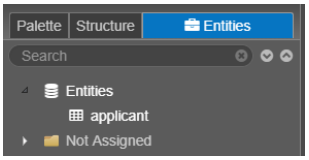
4. To create a new Entity, enter ‘applicant’ followed by an underscore, ‘_’
5. Notice: after you enter underscore (_) an inline warning displays beneath ‘An Entity called applicant will be created’



6. Enter ‘FirstName’ as the ID (after the underscore) then press TAB or click away.
7. Notice: the Label Property displays as ‘First Name’ (without the Entity name)



8. Display the ‘Entities’ panel and notice the ‘applicant’ Entity with one related ‘FirstName’ component

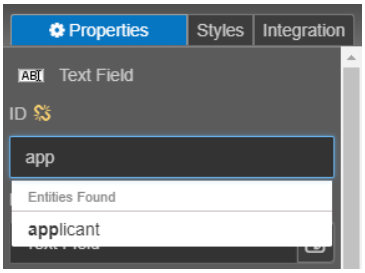


9. Click Save

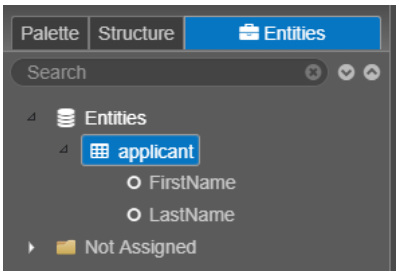
Step 3: Assign an Entity

An existing Entity can be assigned to a new component using the ID field.

1. Add another ‘Text Field’ component beneath ‘First Name’
2. Enter the first few letters of ‘applicant’ in the ID property field, i.e. ‘app’
3. Notice: as you type, any matching Entities will display in an inline dropdown menu



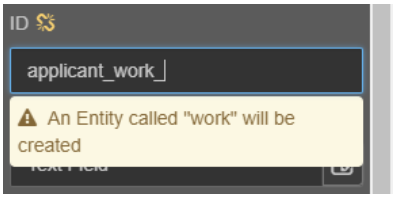
4. Select ‘applicant’ to populate the ID Property field
5. Enter ‘LastName’ (after the underscore) for the ID name.
6. Notice: the label name displays as ‘Last Name’ only
7. Display the ‘Entities’ panel and confirm the ‘applicant’ Entity now has two related components



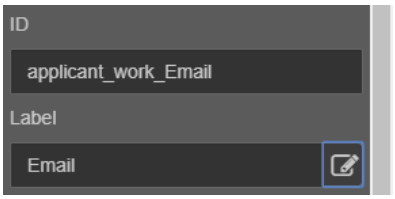
Step 4: Create a nested Entity

You can create Entities within Entities, i.e. a nested Entity.

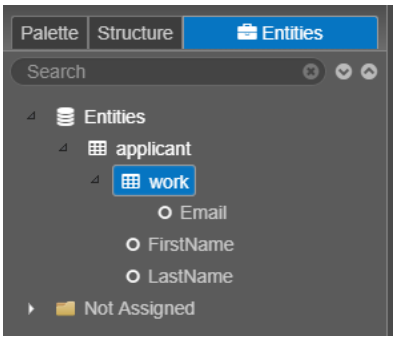
1. Add another new ‘Text Field’ beneath ‘Last Name’
2. Enter the first few letters of ‘applicant’ in the ID property field
3. Select the ‘applicant’ Entity
4. Enter ‘work_’ as the new nested Entity, e.g. ‘applicant1_work_’
5. Notice: after you enter the second underscore (_) an inline warning displays beneath ‘An Entity called work will be created’



6. Enter ‘Email’ for the ID name, e.g. ‘applicant1_work_Email’
7. Press Tab or click away from the ID property field
8. Notice: the label name displays as ‘Email’ only



9. Display the ‘Entities’ panel and confirm the nested ‘work’ Entity with one related component



10. Click Save

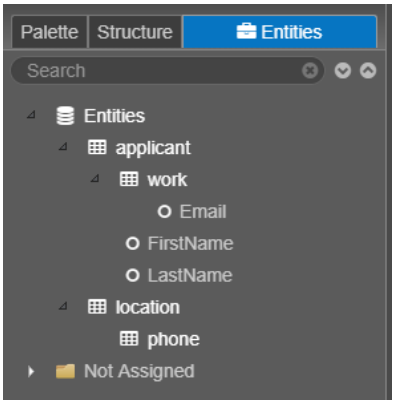
Please note: any existing IDs with an underscore will not automatically cause Entities to be created. Entities will only be created when you modify an existing ID. Any underscores followed by one or more digits (like suffixes applied by auto-generated IDs) will also be ignored.

Exercise - Delete an Entity

You can delete Entities using the Entities panel.

Please note: You can't delete an Entity if any component IDs use that Entity.

1. Your task is to delete the 'phone' Entity, but you'll need to delete or re-assign any related components.
2. For this exercise, delete the 'Home Phone' component on the form
3. Notice: the 'phone' Entity no longer has any related component IDs



4. Right-click the 'phone' Entity and choose 'Delete Entity'
5. Click 'OK' to confirm
6. The Entity has now been deleted from the Entities panel
7. Click Save

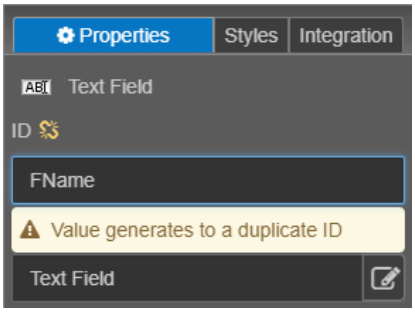
Exercise - Duplicate IDs

This exercise will demonstrate the new duplicate ID functionality. You will:

1. Create a duplicate ID and ‘Review’
2. Create a duplicate ID and ‘Accept’

Step 1: Add a component with a duplicate ID and ‘Review’

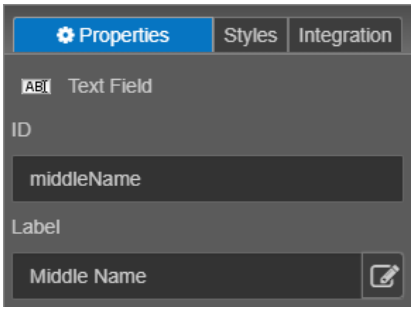
1. Note: Make sure the form you created for the 'ID Management exercise' is open
2. Add a new ‘Text Field’ into the ‘About You’ section (beneath ‘First Name’)
3. Update the ID Property to ‘FName’
4. An inline warning confirms ‘Value generates to a duplicate ID’
   Note: IDs are case sensitive - if you enter 'fname' instead of 'FName', the inline warning will not display



5. At this point, you can either modify the ID property directly in the field
6. Or if you ‘TAB’ (or click away), a popup will prompt you to ‘Review’ the duplicate ID or ‘Accept’ a suffix



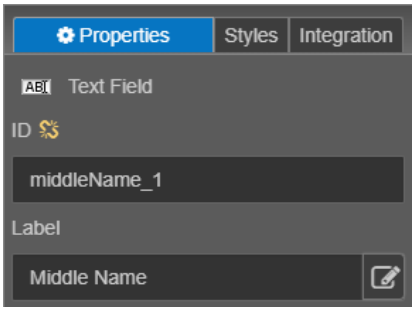
7. For this example, click ‘Review’ and update the ‘ID’ property to ‘MiddleName’



8. Choosing ‘Review’ would let you modify the ID property so that it doesn't cause a duplicate.

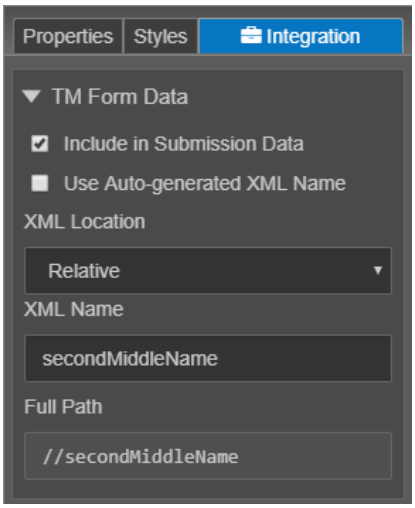
Step 2: Add a component with a duplicate ID and ‘Accept’

1. Add a new ‘Text Field’ into the ‘About You’ section (beneath ‘Middle Name’)
2. Update the ID Property ‘MiddleName’ and TAB away
3. For this example, click ‘Accept’ in the popup
4. Choosing ‘Accept’ appends a suffix to the ID Property (i.e. middleName_1)



5. Click ‘Save’
6. Close the form, i.e. the browser tab

Please Note: If you choose ‘Accept’ for duplicate IDs, the ‘XML Name’ and ‘Full Path’ properties (on the ‘Integration’ panel) keep their default names (i.e. ‘TextField’). To update these properties, you’ll need to uncheck ‘Use Auto-generated XML Name’ and manually update the ‘XML Name’.



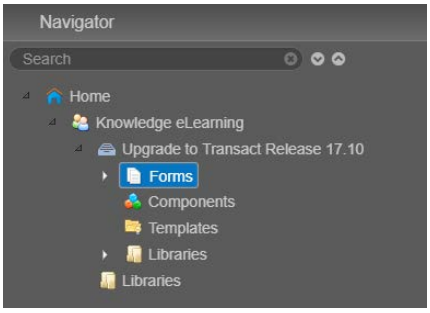
Exercise - ID Management

This exercise will demonstrate the updated functionality for component IDs. You will:

- create a new form
- add a new component
- update the component ID

Step 1: Create a new form

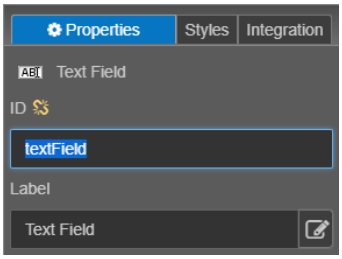
1. Expand the ‘Organization’ (Navigator Panel)
2. Expand the ‘Project’ (Upgrade to Transact Release 17.10)
3. Select ‘Forms’



4. Click the ‘Create Form’ button
5. Enter the following details:
   - Form Name: Exercise - ID Management
   - Description: This form is to demonstrate the new ID features.
   - Template: Training Template
6. Click the ‘Create’ button
7. Note: The form Design view should open in a new Tab

Step 2: Add a new ‘Text Field’ component

1. Make sure the ‘Palette’ panel is displayed
2. Add a new ‘Text Field’ into the ‘About You’ section
3. Note: the initial focus is set to the ID Property field



Step 3: Update the component

1. Update the ‘ID’ Property value to ‘FirstName’
2. Note: The ‘Label’ Property has automatically updated to ‘First Name’
3. Click the ‘Integration’ tab
4. Note: the ‘XML Name’ and ‘Full Path’ properties have updated to the new ‘ID’
5. Click the ‘Properties’ tab
6. Update the ‘ID’ Property to ‘FName’ (then TAB or click away)
7. Note: this time the ‘ID’ Property updates but the ‘Label’ Property remains unchanged. The ‘Label’ only updates when the default ‘ID’ Property is entered.
8. Click the ‘Integration’ tab
9. Note: the ‘XML Name’ and ‘Full Path’ properties have updated to the latest ‘ID’ (i.e. ‘FName’)
10. Click ‘Save’

Please note: If the ‘Use Auto-generated XML Name’ Property is not checked, the ‘XML Name’ and ‘Full Path’ properties won’t update if the ID changes. These properties will need to be manually updated if required.

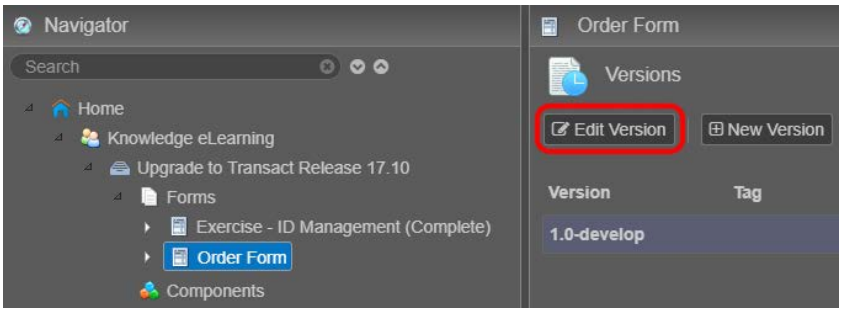
Exercise - Impact Analysis

The following exercise will involve the ‘Order’ form.

Step 1: Open the Order form

From the Maestro Transact Management Dashboard:

1. Expand the ‘Upgrade to Transact Release 17.10’
2. Expand ‘Forms’ and select ‘Order Form’
3. Click the ‘Edit Version’ button

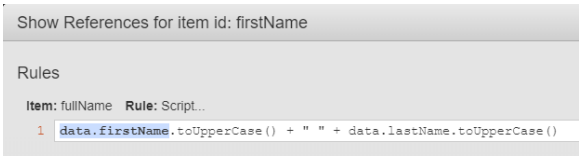


4. The ‘Order Form’ should open in a new browser tab

Step 2: Review the current references

Inspect references for the ‘First Name’ field.

1. Right-click the ‘First Name’ text field (in the form)
2. Choose ‘Show References to firstName’
3. Confirm one existing reference to the ‘FullName’ item

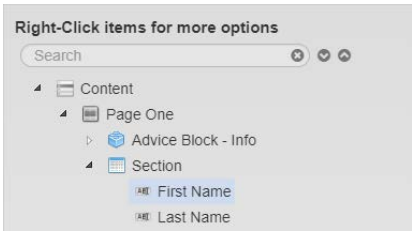


4. Click ‘Close’ (to close the window)

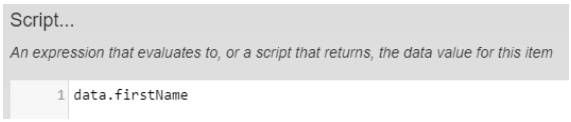
Step 3: Add a new Text Field that references ‘First Name’

Add a new Text Field to ‘Page Two’ that references the ‘First Name’ Text Field

1. Display the ‘Structure’ panel
2. Select ‘Page Two’ from the form structure
3. Re-display the ‘Palette’ panel
4. Add a new Text Field to the Page Two ‘Section’
5. Update the ID Property to ‘FirstNameP2’ and press TAB (focus moves to Label Property)
6. Update the Label Property to ‘First Name’ and press TAB
7. With the new Text Field selected, click ‘Create Rule’ (bottom of Properties panel)
8. Click ‘Script’ (under ‘Calculation’)
9. In the navigator column (on the left), expand ‘Page One’, then expand the first ‘Section’



10. Double-click ‘First Name’ to setup a reference to this component, i.e. ‘data.firstName’

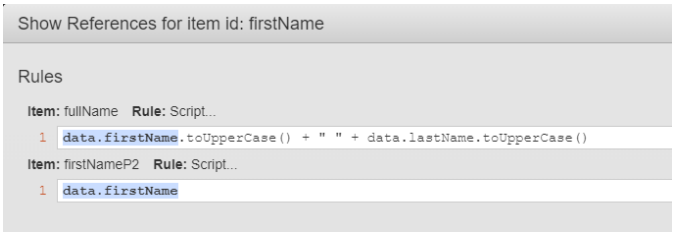


11. Click ‘Save’ (to close the window), then Save the form
12. Click ‘Preview’ to test the new reference, i.e. enter a ‘First Name’ on Page One and confirm it displays in ‘First Name’ on Page Two
13. When you’re ready, switch back to Design view

Step 4: Review the updated references

Inspect the references for the ‘First Name’ field on Page One

1. Display the ‘Structure’ panel
2. Select ‘Page One’ from the form structure
3. Re-display the ‘Palette’ panel
4. Right-click ‘First Name’ and click ‘Show References to ‘First Name’
5. Confirm a second reference to ‘firstNameP2’ (on Page Two) now exists



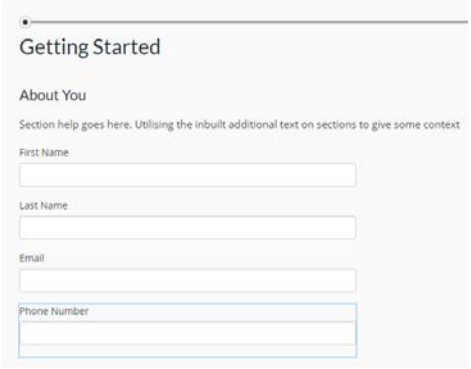
6. Click ‘Close’ (to close the dialog)
7. Click Save then close the form (i.e. the Browser Tab)

Exercise - Semantic Versioning

This exercise will demonstrate how to create multiple semantic versions of a form.

Step 1: Update the 1.0-develop version

1. Select the ‘Exercise - Entities’ form in the Navigator panel
2. Click ‘Edit Version’ to open the form
3. Add a new text field to the ‘Getting Started’ section called ‘PhoneNumber’

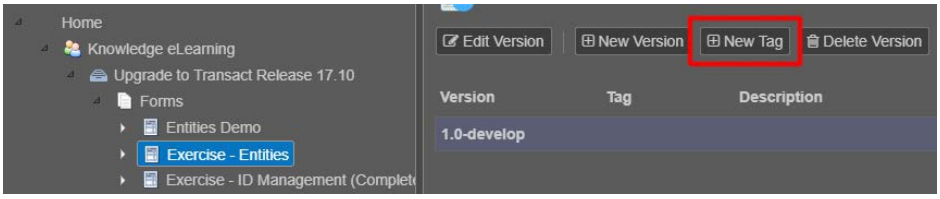


4. Save and Close the form (Browser Tab)

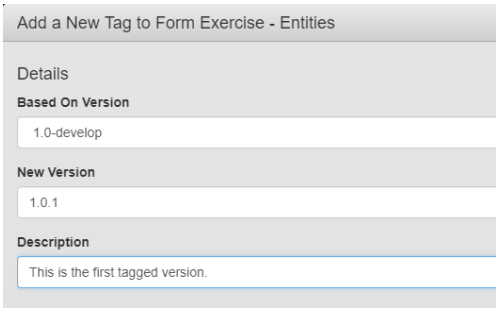
Step 2: Create a ‘Tagged’ version

Create the first read-only tagged version (based on ‘1.0-develop’)

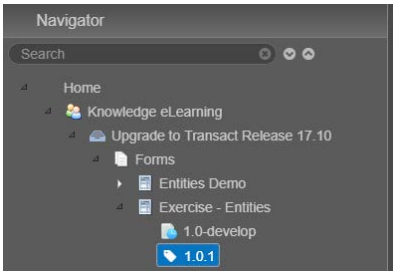
1. Select the ‘Exercise - Entities’ form in the Navigator panel
2. Click ‘New Tag’



3. Confirm the new Semantic Version number is 1.0.1
4. Enter Description: ‘This is the first tagged version.’
5. Click ‘Create’

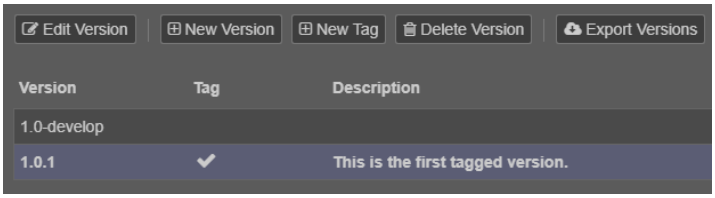


6. Expand the ‘Exercise – Entities’ form in the Navigator panel and confirm the tagged 1.0.1 version is displayed



Step 3: Open the ‘Tagged’ version

1. Select the ‘Exercise - Entities’ form in the Navigator panel
2. Select the ‘1.0.1’ tagged version (note the ‘tick’ icon)

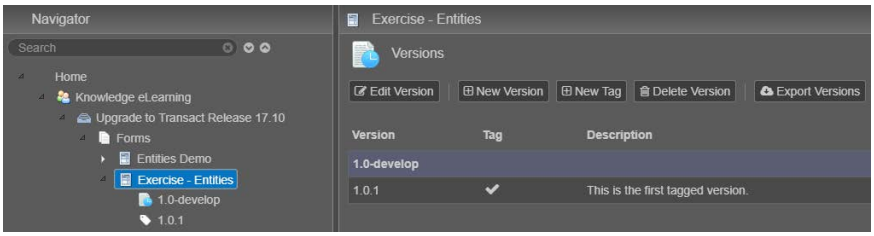


3. Click ‘Edit Version’ to open the form
4. Notice: the Palette panel is empty, i.e. you can’t add new or delete existing components
5. Close the form (Browser Tab)

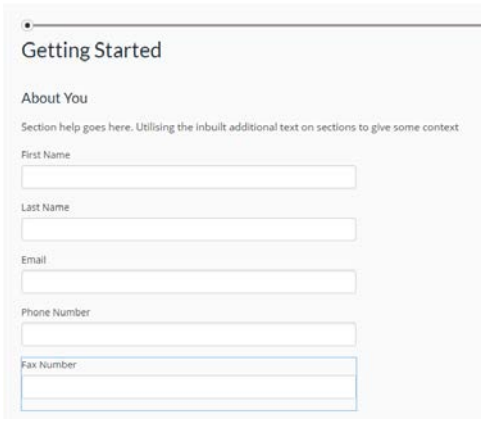
Step 4: Update the 1.0-develop version

Remember: all changes and updates are completed in 1.0-develop

1. Select the ‘Exercise - Entities’ form in the Navigator panel
2. Select the ‘1.0-develop’ version
3. Click ‘Edit Version’ to open the form



4. Add a new text field to the ‘Getting Started’ section called ‘FaxNumber’



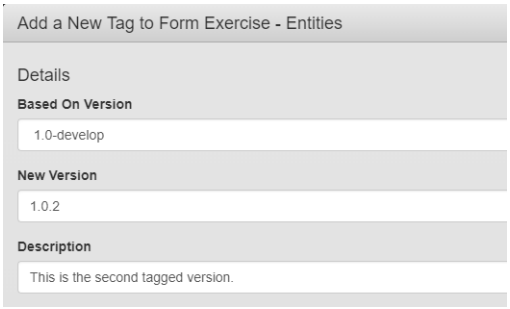
5. Save and Close the form (Browser Tab)

Step 5: Create another ‘Tagged’ version

Create another read-only version (based on ‘1.0-develop’)

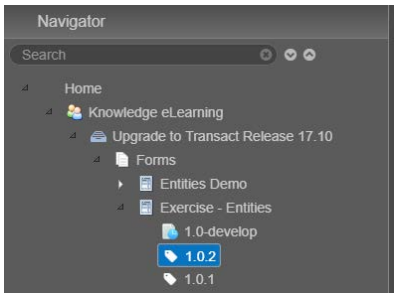
1. Select the ‘Exercise - Entities’ form in the Navigator panel
2. Select the ‘1.0-develop’ version
3. Click ‘New Tag’
4. Confirm the new Semantic Version number is 1.0.2

5. Enter Description: ‘This is the second tagged version.’



6. Click ‘Create’

Expand the ‘Exercise – Entities’ form in the Navigator panel and confirm the tagged 1.0.2 version is displayed



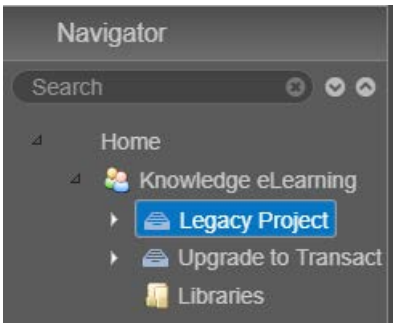
Exercise - Upgrade a project

You can upgrade an existing project to the latest Maestro release from the Management Dashboard.

Step 1: Import a project

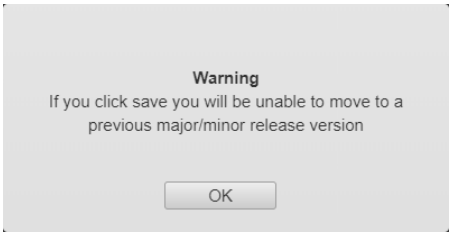
To upgrade a project, you'll need to add a legacy project to your organization first.

1. Download this sample project (zipped folder)
2. Remember where the project is stored, e.g. 'Downloads'
3. Navigate to the Maestro webpage
4. Expand Home (in Navigator Panel)
5. Select the Organization (the Organisation maybe your own name e.g. jsmith
6. Click 'Import Legacy Project'
7. Name the project - 'Legacy Project'
8. Click Import
9. Navigate to the location of the sample project
10. Select the project and click Open
11. The imported project should now appear under your organization



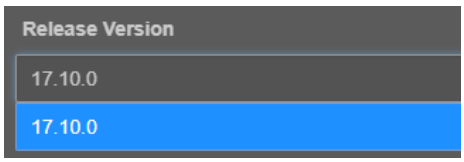
Step 2: Upgrade the project

1. Select 'Legacy Project' project in the Navigator
2. Click the Project Details tab
3. Display the 'Release Version' dropdown and choose '17.10.0'
4. Note: When you select a new version, a warning message confirms you can't change the release version back to a previous version once the Project Details are saved



5. Click OK in the warning message
6. Click Save
7. The project has now been upgraded

Notice: The Release Version dropdown no longer displays any previous release versions



Glossary

Journey Platform | All Personas

A

A/B testing

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.

Abandoned Transaction

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.

Acquisition

Acquisition is the process of capturing data to start the process of becoming a new customer

AcroForm

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.

ADFS

Active Directory Federation Services.

Adobe Flash

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.

Adobe LiveCycle ES

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.

Agile software development

Agile software development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer/end user.

AIM

Association for Automatic Identification and Mobility (AIM) is an industry trade group that developed and standardized bar codes, Automatic identification and data capture.

Amazon AWS Security Token Service

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).

Amazon CloudFront

Amazon CloudFront is a content delivery network (CDN). It is integrated with Manager to provide an easy way to distribute static content such as images to end users with low latency, and high data transfer speeds. This improves the speed at which forms load, providing a better customer experience. It also reduces load on theManager servers, resulting in a higher transaction throughput per server.

Amazon HSM

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.

Amazon Key Management Service

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.

Amazon Redshift

Amazon Redshift is an Internet hosting service and data warehouse product which forms part of the larger cloud-computing platform Amazon Web Services. It is built on top of technology from the massive parallel processing (MPP) data-warehouse company ParAccel to handle large scale data sets and database migrations.

Amazon Resource Name

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.

Amazon S3

Amazon S3 is object storage built to store and retrieve any amount of data from anywhere – web sites and mobile apps, corporate applications, and data from IoT sensors or devices. It is designed to deliver 99.999999999% durability, and stores data for millions of applications used by market leaders in every industry. S3 provides comprehensive security and compliance capabilities that meet even the most stringent regulatory requirements.

Amazon Simple Email Service

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.

Amazon Simple Queue Service

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.

Amazon Web Services

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.

Angular

Angular is a platform that makes it easy to build applications with the web. Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop.

AngularJS

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write. For more information, see http://angularjs.org/

Anti-Money Laundering

Anti-Money Laundering (AML) refers to existing laws or procedures meant to reduce illegally obtained income. The legal controls that require financial institutions and other regulated entities to prevent, detect, and report money laundering activities.

Apache Ant

Apache Ant is a software tool for automating software build processes, which originated from the Apache Tomcat project. It was a replacement for the Make build

tool of Unix.

Apache HTTP Server

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.

Apache Velocity Template

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

API

Application Programming Interface.

Artifactory

Artifactory is a product by JFrog that serves as a binary repository manager, which is an extension to the source code repository where it stores the outcome of your build process, often denoted as artifacts.

Assistive Technology

is an umbrella term that includes assistive, adaptive, and rehabilitative devices for people with disabilities while also including the process used in selecting, locating, and using them.

Automated Clearing House

Automated Clearing House (ACH) is the electronic network used to transfer money between accounts at different institutions.

AWS Aurora RDS

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks.

Azure

Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers.

Azure Blob Storage

Azure Blob Storage is Microsoft's object storage solution for the cloud. Blob storage is optimized for storing massive amounts of unstructured data. Unstructured data is data that doesn't adhere to a particular data model or definition, such as text or binary data.

Azure Key Vault

Azure Key Vault is a cloud service that provides a secure store for secrets. You can securely store keys, passwords, certificates, and other secrets.

Azure Queue Storage

Azure Queue Storage provides cloud messaging between application components. In designing applications for scale, application components are often decoupled so they can scale independently. Queue storage delivers asynchronous messaging between application components, whether they are running in the cloud, on the desktop, on an on-premises server, or on a mobile device. Queue storage also supports managing asynchronous tasks and building process work flows.

B

Bank Secrecy Act

Bank Secrecy Act (BSA)/Anti-Money Laundering (AML) is the BSA requires financial institutions to collaborate with the government in cases of suspected money laundering and fraud. Banks must conduct anti-money laundering checks and keep records of suspicious events.

Banking Platform

A plug-and-play business model that allows multiple participants (producers and consumers) to connect to it, interact with each other, and create and exchange value. Successful platforms attract both producers and consumers, match producers with consumers, and provide seamless integration among participants. Avoka Platform enables account opening and onboarding.

Barcode

Barcode is a machine-readable code in the form of numbers and a pattern of parallel lines of varying widths, printed on a commodity and used especially for stock control.

BAU

usiness As Usual (BAU) is the view of an organisation, business unit, work team or team member completing their everyday work in the normal way.

Biometrics

Is the electronically matching of a person's unique physical characteristics with their account.

BIRT

Business Intelligence and Reporting Tools (BIRT) is an open source data visualization and reporting platform. For more information, see http://www.eclipse.org/birt/

Blockchain

Also known as distributed ledger technology (DLT), is a software that is a shared record of information managed by a network of computers rather than a central authority. It is secured using cryptography.

Bounce or bounced transaction

Bounce is when a user views a form but never starts it and simply closes the window immediately or navigates away.

Bounced Transaction

Any transaction where the applicant (user) did not interact with the web application (form). Navigation across the form or scrolling the page does not count as an interaction. A legitimate interaction is one where a user inputs data into the form in some way, such as text input, radio button input, clicking on a checkbox, clicking the Save button, attaching a file etc. It typically takes 24-48 hours for a transaction in Open status to transition to Bounced status. This time will vary based on the settings configured in Manager.

BPMN

Business Process Modeling Notation (BPMN) is a method of illustrating business processes in the form of a diagram similar to a flowchart.

C

CentOS

CentOS (Community Enterprise Operating System) is a Linux distribution that provides a free, enterprise-class, community-supported computing platform functionally compatible with its upstream source, Red Hat Enterprise Linux (RHEL).

Changelog

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.

Checking Account

The basic account for easy access to your money.

ClamAV

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.

CLI

A command line interface (CLI) is a text-based user interface (UI) used to view and manage computer files.

Cloud computing

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.

Code Reuse

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.

Completed Transaction

Any transaction that has been completed by the user and submitted to Manager.

Component

A component is the general name for anything that can be dragged from the Palette onto a form. Components can be anything from a single data entry field, to a large complex re-usable block.

Content Delivery Network

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.

Content Security Policy

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/

Continuous Integration

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

Coordinated Universal Time

Coordinated Universal Time (UTC) is the primary time standard by which the world regulates clocks and time.

CORS

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.

Credit Rating

An evaluation of credit-worthiness based on financial resources and credit history. Usually expressed as a letter grade (A, B+, etc.)

Credit Report

A report on an individual or organisation identifying credit details such as: defaults, judgments, bankruptcies, etc.

Credit Score

In contrast to a credit rating, the credit score is a number (600, 700, etc.) indicating an individual's credit-worthiness.

Credit Union

A member-owned financial cooperative, controlled by its members and operated on the principle of people helping people, providing its members credit at competitive rates as well as other financial services. Credit unions offer many of the same financial services as banks, but often using a different terminology.

CRM

Customer relationship management (CRM) is an approach to manage a company's interaction with current and potential customers. It uses data analysis about customers' history with a company to improve business relationships with customers, specifically focusing on customer retention and ultimately driving sales growth.

Cron

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.

Cryptocurrency

A digital currency that relies on cryptography to validate and secure transactions.

CSRF

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.

CSS

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.

CSS3

CSS3 is the latest evolution of the Cascading Style Sheets language and aims at extending CSS2.1. It brings a lot of long-awaited novelties, like rounded corners, shadows, gradients, transitions or animations, as well as new layouts like multi-columns, flexible box or grid layouts.

CSV

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.

Current Account

A transaction account, checking account, demand deposit account, or share draft account (at credit unions) is a deposit account held at a bank or other financial institution.

Custom component

Sometimes it is useful to create a re-usable variant of either a simple or complex component. The main variations are: • A component with a re-usable business rule, particularly a validation rule. An email field is an example of this. • A component (radio button or dropdown list) with a specific set of values. An example of this is a dropdown list of States. • A component with a particular XML binding.

Customer Experience

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.

Customer Identification Program

Customer Identification Program (CIP) The CIP is a required provision under the USA PATRIOT Act. CIP is more commonly known as KYC.

CVE

Common Vulnerabilities and Exposures (CVE) is a list of publicly disclosed computer security flaws, where each security flaw has a CVE ID number assigned to it for further reference.

CX

Customer Experience (CX) is an aggregate of a customer’s A-Z experience with a brand. It is not a snapshot in time, such as a key touchpoint or a critical interaction.

D

DAO

Deposit Account Opening (DAO). DAO bank forms allow users to open Checking, Savings and Money Market Accounts.

DB

A database is a data structure that stores organized information. Most databases contain multiple tables, which may each include several different fields. For example, a company database may include tables for products, employees, and financial records. Each of these tables would have different fields that are relevant to the information stored in the table.

DDA

Direct Deposit Account (DDA) is simply a checking or savings account which offers the ability to send and receive funds electronically.

DDS

The Data Distribution Service (DDS) is a middleware protocol and API standard for data-centric connectivity from the Object Management Group (OMG). It integrates the components of a system together, providing low-latency data connectivity, extreme reliability, and a scalable architecture that business and mission-critical Internet of Things (IoT) applications need.

Design

A design is a generic name for a form, receipt, template, or component that you can create in Maestrousing the Maestro editor

Design Mode

Design mode displays the editor window so that Maestro assets can be created and updated

DevOps

DevOps is a software development methodology that combines software development (Dev) with information technology operations (Ops). The goal of DevOps is to shorten the systems development life cycle while also delivering features, fixes, and updates frequently in close alignment with business objectives.

Digital Adoption

Achieving a state in which digital tools are being used as intended, and to their fullest extent. It is about more than just basic functions — digital adoption happens when technology becomes second nature to its user.

Digital Banking

The move to online banking where banking services are delivered over the Internet. The advantages for banks and customers are providing more convenient and faster banking services.

Digital ID

Also known as Digital Credential, is the electronic equivalent of an individual's identity card. A digital ID can be presented electronically to prove an individual's identity and their right to access information or services online.

Digital Native

A person that grew-up in the digital age and is comfortable with technology.

Disaster-Recovery-as-a-Service

Disaster-Recovery-as-a-Service (DRAAS) is the hosting of servers by a third party in case of a disaster.

Document Object Model

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.

DocuSign eSignature

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.

E

EAR

Enterprise Archive.

Electronic Check Presentment

Electronic Check Presentment (ECP) is an electronic image of a check that can be processed by banks.

Electronic Deposit Verification

Electronic Deposit Verification (EDV) is a way to verify an account at another bank that you want to link to from the current institution.

Electronic Funds Transfer

Electronic Funds Transfer (EFT) is the transfer of money between accounts through electronic payment systems.

Electronic Identity Verification

Electronic Identity Verification (eIDV) is an electronic method to confirm the identification of an individual.

End-of-Life

End-of-life (EOL) is a term used with respect to a product supplied to customers, indicating that the product is in the end of its useful life (from the vendor's point of view), and a vendor stops marketing, selling, or rework sustaining it.

Endpoint

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.

EOT font

Embedded OpenType (EOT) fonts are a compact form of OpenType fonts designed by Microsoft for use as embedded fonts on web pages. These files use the extension . eot

ERP

Enterprise resource planning (ERP) is the integrated management of main business processes, often in real-time and mediated by software and technology.

ES6

ECMAScript (ES6) is a general-purpose programming language, standardized by Ecma International according to the document ECMA-262. It is a JavaScript standard meant to ensure the interoperability of Web pages across different Web browsers.

Europay, MasterCard, and Visa

Europay, MasterCard, and Visa (EMV) is a payment method based on agreed standards for use with smart payment cards (also referred to as chip or IC cards).

F

FAR

FAR is a Form Archive that contains a form and all related resources that can be deployed as a package in various Journey Manager environments.

Fast IDentity Online

Fast IDentity Online (FIDO) is an industry consortium with the aim to address the lack of interoperability among strong authentication devices and the problems users face creating and remembering multiple usernames and passwords. It comprises a set of technology-agnostic security specifications for strong authentication.

Field

A field is a specific type of component used for displaying and entering data into a form application. Examples include the Text Field or Decimal Field. In Journey Analytics only, a field also refers to any field, radio button or component in a form application used to enter data.

Financial Institution

An institution (public or private) that collects funds (from the public or other institutions) andinvests them in financial assets.

Financial Technology

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.

Financial Technology Terms

Financial technology, often referred to as FinTech or fintech, is an umbrella term describing new technology used to support or enable banking and financial

services.

Finovate

Also known as FinTech Innovation, a demo-focused conference series focused exclusively on showcasing the best and most innovative new financial and banking technologies.

FinServ

Anything that describes Financial Services.

Fluent

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.

Font Awesome

Font Awesome is a font and icon toolkit based on CSS and Less.

Form Builder

A Form Builder is the role name of the person that creates the form. A Form Builder focuses on the specific form content as well as the journey of the form user.

Form User

The form user is the person completing the form. They may also be known as "user".

Frictionless

Achieved with or involving little difficulty, or effortless. Avoka's Platform helps to provide frictionless customer onboarding and account opening experiences.

FTX

Form Transaction Experiences

Fully qualified domain name

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.

G

General Availability

General availability (GA) is the marketing phase when all commercialization activities pertaining to the software product have been completed and it is available for purchase. It is also known as production release.

Geolocation

Identifying person or device's geographical location. This may involve GPS or Global Positioning System.

Git

Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development,[8] but it can be used to keep track of changes in any set of files.

GitHub

GitHub Inc. is a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features.

GitLab

GitLab is a web-based Git-repository manager with wiki and issue-tracking features, using an open-source license, developed by GitLab Inc.

Google Tag Manager

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.

Groovy

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.

GUI

The graphical user interface (GUI) is a form of user interface that allows users to interact with electronic devices through graphical icons and audio indicator such as primary notation, instead of text-based user interfaces, typed command labels or text navigation.

GUID

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.

GZIP

GZIP is a file format and a software application used for file compression and decompression.

H

HA

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.

Happy path

A happy path is a default scenario featuring no exceptional or error conditions.

Home Mortgage Disclosure Act

Home Mortgage Disclosure Act (HMDA) is HMDA requires banks and credit unions to collect and publicly report data on particular types of mortgages and lines of credit.

HSM

A hardware security module (HSM) is a physical computing device that safeguards and manages digital keys for strong authentication and provides cryptoprocessing.

HTML5

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.

HTTP

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.

HTTP referer

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.

I

Icon Picker

The Icon Picker is a selection dialog allowing you to pick an icon from the range of images shipped with Maestro.

IDE

Integrated Development Environment.

In-Progress Transaction

Any transaction that is Opened, Started or Saved.

IP address

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.

iText

iText is a library for creating and manipulating PDF files in Java and .NET.

J

Java

Java is a programming language that produces software for multiple platforms. When a programmer writes a Java application, the compiled code (known as byte-code) 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.

Javadoc

Javadoc is a documentation generator created by Sun Microsystems for the Java language for generating API documentation in HTML format from Java source code.

JavaScript

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.

JDBC

JDBC (Java Database Connectivity) is the Java API that manages connecting to a data-base, issuing queries and commands, and handling result sets obtained from the database.

Joint account

A bank account held by more than one person. Joint holders of an account are regarded in law as together making up the 'owner' of the account.

JSON

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.

JVM

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.

K

KB

A knowledge base (KB) is a technology used to store complex structured and unstructured information used by a computer system.

Know your customer

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).

Knowledge-Based Authentication

Knowledge-Based Authentication (KBA) aid(s) used for fraud prevention? An example would be a “secret question” users must answer before being granted access to account information.

Kofax

Kofax software enables organizations to Work Like Tomorrow - today. It helps organizations transform information-intensive business processes, reduce manual work and errors, minimize costs, and improve customer engagement.

KPI

Key Performance Indicator (KPI) is a measurable value that demonstrates how effectively a company is achieving key business objectives. Organizations use KPIs to evaluate their success at reaching targets.

L

Landing Page

A web page that is accessed from another website or software application.

LDAP

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.

LESS

LESS is a dynamic preprocessor style sheet language that can be compiled into Cascading Style Sheets (CSS) and run on the client side or server side.

Less CSS

Less is a dynamic preprocessor style sheet language that can be compiled into Cascading Style Sheets and run on the client side or server side.

LexisNexis

LexisNexis is a corporation providing computer-assisted legal research (CALR) as well as business research and risk management services.

Linux

Linux is a family of free and open-source software operating systems built around the Linux kernel. Typically, Linux is packaged in a form known as a Linux distribution (or distro for short) for both desktop and server use.

LMS

Lifecycle Management Suiteis software designed for lending. For more information, see https://www.temenos.com/us/news/2015/02/10/temenos-purchases-us-based-financial-software-provider-akcelerant/

Loan to Value Ratio

Loan to Value Ratio (LVR) is the amount, as a percentage, borrowed for an asset against the value of the asset.

Locale

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.

Template Designer

The Template Designer is responsible for implementing and maintaining templates which are the starting point for all forms created in Maestro. The key activity of the Template Designer is to implement the template content such as common page components that typically surround specific form content, including header, navigator, and footer. The Template Designer will also create styles that the Form Builder will use in the creation of form content including text field styles, buttons, and section headers.

Multi-field component, or complex component

A multi-field component is made up of more sub-parts than a regular component. For example, a Section consists of the Heading text, sub-text, help text, a collapse/expand button and a significant amount of JavaScript and CSS. An address block contains several different simple data entry fields. A component such as a Driver's License scanning component may have multiple data entry fields, a camera button, descriptive images, and more. Multi-field components can be built as either a native component or a re-usable block.

Preview Mode

Preview mode displays the form so that you can interact with the components on the form as if it were live.

Re-usable Block

A custom block is a collection of simple components, that can be re-used. An example of a re-usable block is an Address Block, which consists of two lines of text plus City, State, and ZIP Code. Unlike a native multi-field component, each of the sub-elements of the custom block will be visible in the Structure tab and can be modified individually by the person using your block.

Native multi-field component

A native multi-field (or complex) component is still a completely self-contained component. There is no fixed boundary between a simple and a complex native component, although any component with more than one visible data entry area within it would usually be considered to be complex

Maestro form

Maestro form is a device used to collect information from a user. Its purpose will vary depending on the organization that is building and implementing the form. The development technologies and programming languages that are available to Form Builders will vary depending on these purposes. However, there are several development technologies and programming languages that are used in every Maestro form. These development technologies and programming languages are identified below, as well as their role in the development of a form or their place in the Journey platform.

Metadata

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.

Micro-deposits

Small deposits (nominal amount) made to verify an account.

Microsoft SQL Server

Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network (including the Internet).

Microsoft Windows Server

Microsoft Windows Server OS (operating system) is a series of enterprise-class server operating systems designed to share services with multiple users and provide extensive administrative control of data storage, applications and corporate networks.

Minification

Minification is the process of removing all unnecessary characters from the source code of interpreted programming languages or markup languages without changing its functionality.

Mitek

Mitek is a software company that specializes in digital identity verification and mobile capture built on artificial intelligence algorithms. The company's technology allows people to deposit checks via their mobile phones and open bank accounts from mobile devices. Additionally, the technology verifies people's identity with patented algorithms that capture, classify and authenticate photos of identity documents including passports, ID cards, and driver's licenses. The identity of the person holding the document is confirmed through a biometric facial comparison of the photo on the document to a selfie.

MongoDB

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

My Term

My definition

MySQL

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.

O

Object (computer science)

In computer science, an object can be a variable, a data structure, a function, or a method, and as such, is a value in memory referenced by an identifier. In the class-based object-oriented programming paradigm, object refers to a particular

instance of a class, where the object can be a combination of variables, functions, and data structures. In relational database management, an object can be a table or column, or an association between data and a database entity (such as relating a person's age to a specific person).

Office of Foreign Asset Control

Office of Foreign Asset Control (OFAC) enforces economic and trade sanctions. Banks must perform an OFAC screen on customers opening an account.

Omnichannel

Omnichannel is a service delivery model that integrates different interaction points for customers such as online, by phone or in store. Omnichannel Banking offers banking access across these three methods.

Onboarding

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

Open Banking

Indicated the use of open APIs by banks to enable software developers to create applications using the bank's data.

Open Transaction

All transactions, when created byManager, are considered to be in Open status. From this status, a transaction can either proceed to the Started or Bounced statuses.

Open UX

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.

OpenSAML

OpenSAML is a set of open source C++ & Java libraries used in support of the Shibboleth Project's implementation of the Security Assertion Markup Language (SAML).

Oracle Linux

Oracle Linux (OL, formerly known as Oracle Enterprise Linux) is a Linux distribution packaged and freely distributed by Oracle, available partially under the GNU General Public License since late 2006. It is compiled from Red Hat Enterprise Linux (RHEL) source code, replacing Red Hat branding with Oracle's. It is also used by Oracle Cloud and Oracle Engineered Systems such as Oracle Exadata and others.

Oracle RDBMS

Oracle Database is a multi-model database management system produced and marketed by Oracle Corporation. It is a database commonly used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads.

Origination

Origination is the process of setting up the approved customer (new or existing) with specific products and features, and get them completely configured in the core.

P

PaaS

Platform as a Service (PaaS) is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.

Page Base Content

A web page’s content as defined in the form space in Manager.

Payee

A person or business to whom a check is written.

Payment Card Industry Compliance

Payment Card Industry Compliance (PCI Compliance) is a set of security standards designed to protect card information during and after financial transactions.

Payment Gateway

A service provider that authorizes card payments.

PCI policy

PCI policy is a type of security policy that covers how an organization addresses the 12 requirements of the Payment Card Industry Data Security Standard (PCI DSS). A PCI policy is required of all merchants and service providers who store, process or transmit credit card holder data.

Pen Test

Penetration testing, also called pen testing or ethical hacking, is the practice of testing a computer system, network or web application to find security vulnerabilities that an attacker could exploit. Penetration testing can be automated with software applications or performed manually. Either way, the process involves gathering information about the target before the test, identifying possible entry points, attempting to break in, either virtually or for real, and reporting back the findings.

Personal Identification Number

Personal identification Number (PIN) is a number issued with your debit or credit card.

Personally identifiable information

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.

Personally Identifiable Information

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.

PhantomJS

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.

PII

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.

Proxy

A proxy server is a server (a computer system or an application) that acts as an intermediary (a gateway) for requests from clients seeking resources from other servers. A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource available from a different server and the proxy server evaluates the request as a way to simplify and control its complexity.

Puppeteer

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.

Q

QR Code

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.

R

RAF

RAF is an RAF file is a digital photograph captured by select Fuji digital cameras, such as the high-end FinePix models. It contains an image in the uncompressed

RAW Fuji (RAF) format. RAF files may also include a .JPG version of the same image within the file for thumbnail preview purposes.

ReactJS

ReactJS is a JavaScript library for building user interfaces.

Recovery Point Objective

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.

Recovery Time Objective

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.

Red Hat Enterprise Linux Server

is a distribution of the Linux operating system developed for the business market.

Reference Number

A reference number is a unique identifier assigned to each transaction.

Referer URL

Referer URL is the URL that was used to navigate to the form.

Regular Expression

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.

REST

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.

Routing Number

The numbers that appear at the bottom of a check to identify the financial institution.

S

SaaS

Software as a service (SaaS) is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted.

SalesForce

Salesforce is the customer relationship management (CRM) platform.

SAML

Security Assertion Markup Language (SAML) is an open standard that allows identity providers (IdP) to pass authorization credentials to service providers (SP). SAML transactions use Extensible Markup Language (XML) for standardized communications between the identity provider and service providers. SAML is the link between the authentication of a user's identity and the authorization to use a service.

Saved Transaction

Any transaction that has explicitly been Saved by the user. When a user resumes a Saved transaction, it can move to Completed status if the user submits the form. Or it can move to Abandoned status if the user abandons the application.

SCCS

Source Code Control System (SCCS) is a version control system designed to track changes in source code and other text files during the development of a piece of software. This allows the user to retrieve any of the previous versions of the original source code and the changes which are stored.

SDK

Software Development Kit (SDK) brings together a group of tools that enable the programming of mobile applications.

SDLC

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares.

Section

Applications built in Maestro or Composer are divided into sections. A section can be a page or modal page or modal dialogue.

Security Manager

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.

Semantic versioning

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.

SEP

System Event Publisher

Service-Level Agreement

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.

Session ID

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.

Session management

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.

SharePoint

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

Single Channel

The service delivery model that limits customers to a single interaction point or pathway. Customers are restricted to just online, phone, or in store/branch access, for example.

SLA

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.

SmartForm

A SmartForm is a dynamic electronic form with capabilities beyond a traditional paper or PDF form, allowing electronic completion, dynamic sections, database calls and electronic submission.

SMTP

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.

SoE

Systems of Engagement (SoE) is the technology used by an organization to help facilitate and orchestrate the customer journey via more personalized, seamless interactions across the various touchpoints. These include social media channels, email marketing platforms, mobile apps, and content management systems.

Software as a Service

Software as a Service (SaaS) is a software distribution model where logical support and data are stored in the servers of an IT company, and customers access the service online. This model is common in FinTech services because of scalability and security.

SOQL

Salesforce Object Query Language (SOQL) is used to construct simple but powerful query strings in the queryString parameter in the query() call, in Apex statements, in Visualforce controllers and getter methods, or in the Schema Explorer of the Force.com IDE.

SoR

System of Record (SOR) is an ISRS (information storage and retrieval system) that is the authoritative source for a particular data element in a system containing multiple sources of the same element. To ensure data integrity, there must be one -- and only one -- system of record for a given piece of information.

SoW

A statement of work (SoW) is a document routinely employed in the field of project management. It defines project-specific activities, deliverables and timelines for a vendor providing services to the client.

SSL

SSL (Secure Sockets Layer) and its successor, TLS (Transport Layer Security), are protocols for establishing authenticated and encrypted links between networked computers. Although the SSL protocol was deprecated with the release of TLS 1.0 in 1999, it is still common to refer to these related technologies as “SSL” or “SSL/TLS.”

SSO

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.

Started Transaction

Started Transaction is any transaction where a user interacted with a form application by inputting some data. A started transaction is a progressive step towards completion. Started transactions and Bounced transactions are mutually exclusive. For instance, a started transaction can never become a bounced transaction and vice versa.

Static Content Externalization

Static Content Externalization (SCE).

SVG font

An SVG font is a new version of the OpenType format, with SVG standing for Scalable Vector Graphics. The SVG glyph format allows the characters to be displayed in multiple colors and different transparencies, and some may even be animated.

SVN

Subversion (SVN) is a software versioning and revision control system.

T

Thread

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.

TIG

Transact Integration Gateway

TimeToCustomerValue

TTCV

TIS

Top Image Systems (TIS) is a global innovator of on-premise and cloud-based applications that optimize content-driven business processes such as procure to pay operations, remittance processing, integrated receivables, customer response management and more.

TLS

Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network.

Token-signing certificate

Token signing certificates are standard X509 certificates that are used to securely sign all tokens that the federation server issues. Token decryption certificates are standard X509 certificates that are used to decrypt any incoming tokens. They are also published in federation metadata.

Topic-based authoring

Topic-based authoring is a modular approach to content creation where content is structured around topics that can be mixed and reused in different contexts.

Transact Application

A Journey Application is what the applicant or user interacts with to submit their application. It is a Maestro or Transact Composer form, possibly with some Transact Exchange components, as well as other configuration information and services

code. Once the user submits the form application, it becomes a completed transaction in Manager.

Transact Application Version

A Journey Application Version is a version of a form application as published by Manager.

Transact Insights Field Name

Journey Analytics Field Name is another version of a form application field name configured in the Maestro Data tab to improve readability of the analytics data on Journey Analytics.

Transact Integration Agent

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.

Transact Manager

Journey Manager is a server application, deployed in the cloud or on-premise, that hosts forms, controls interaction with customers, calls on specialized services like identity verification or risk analysis and exchanges data with back office systems of record.

Transaction

A transaction is created when a user opens a form application.

Transpiler

Ttranspiler is a source-to-source translator, source-to-source compiler, transcompiler, which is a type of translator that takes the source code of a program written in a programming language as its input and produces an equivalent source code in the same or a different programming language.

Transport Layer Security

Transport Layer Security (TLS) is a protocol that provides privacy and data integrity between two communicating applications. It's the most widely deployed security protocol used today, and is used for Web browsers and other applications that

require data to be securely exchanged over a network, such as file transfers, VPN connections, instant messaging and voice over IP.

TTF font

TrueType (TTF) is an outline font standard developed by Apple in the late 1980s as a competitor to Adobe's Type 1 fonts used in PostScript. It has become the most common format for fonts on the classic Mac OS, macOS, and Microsoft Windows operating systems.

Two-factor authentication

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.

U

UAT

User acceptance testing (UAT) is the last phase of the software testing process. During UAT, actual software users test the software to make sure it can handle required tasks in real-world scenarios, according to specifications.

Underbanked

People who have limited or no access to proper banking or services.

Unicode

Unicode is an information technology standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems.

Unit Test

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.

USA PATRIOT Act

The purpose of the USA PATRIOT Act is to deter and punish terrorist acts in the United States and around the world and strengthen U.S. measures to prevent, detect and prosecute international money laundering and financing of terrorism.

User Agent

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.

User authentication

User authentication is the verification of an active human-to-machine transfer of credentials required for confirmation of a user's authenticity.

User authorization

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.

UTC

Universal Time Coordinated. Prior to 1972, this time was called Greenwich Mean Time (GMT) but is now referred to as Coordinated Universal Time or Universal Time Coordinated (UTC). It is a coordinated time scale, maintained by the Bureau International des Poids et Mesures (BIPM). It is also known as "Z time" or "Zulu Time".

UX

User experience (UX) design is the process of creating products that provide meaningful and relevant experiences to users. This involves the design of the entire process of acquiring and integrating the product, including aspects of branding, design, usability, and function.

UXP

A user experience platform (UXP) is an integrated set of technologies used to provide interaction between a user and a set of applications, processes, content, services or other users. A UXP has several components, including portals, mashup tools, content management, search, rich Internet application (RIA) tools, analytics, collaboration, social and mobile tools. It may be delivered as a suite of products or as a single product.

V

Virtual Data Centre

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

Virtual Machine

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.

Virtualization

Virtualization is the creation of a virtual resource such as a server, desktop, operating system, file, storage or network to manage workloads by radically transforming traditional computing to make it more scalable.

Virus Scanning Engine

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.

W

WAR

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.

White-label

White-label software is software that is purchased by a company from a service provider and re-branded as its own.

Wireframe

A wireframe is a diagram or a set of diagrams that consists of simple lines and shapes representing the skeleton of a website or an application's user interface (UI) and core functionality. In Maestro, it's the central part of the Design mode interface.

WOFF font

The Web Open Font Format (WOFF) is a font format for use in web pages. WOFF files are OpenType or TrueType fonts, with format-specific compression applied and additional XML metadata added.

WOFF2 font

WOFF2 font is a web font file created in the WOFF (Web Open Font Format) 2.0 format, an open format used to deliver webpage fonts on the fly. It is saved as a compressed container that supports TrueType (. TTF) and OpenType (. OTF) fonts. WOFF2 files also support font licensing metadata.

Workflow

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.

WYSIWYG

WYSIWYG, an acronym for What You See Is What You Get, is a system in which editing software allows content to be edited in a form that resembles its appearance when printed or displayed as a finished product, such as a printed document, web page, or slide presentation.

X-Frame-Options

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.

XDP

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.

XLS

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.

XML

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.

XPath

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).

XQuery

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.

XSD

XSD (XML Schema Definition) is a World Wide Web Consortium (W3C) recommendation that specifies how to formally describe the elements in an Extensible Markup Language (XML) document.