temenos

Maestro

JOURNEY MANAGER

VERSION 22.10

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 Headquarters SA - all rights reserved.

TOC

Journey Maestro Overview **55**

Maestro Features **59**

Main Features 59

Features by Release 59

23.04 60

22.10 60

22.04 60

21.11 60

21.05 61

20.11 62

20.05 62

19.11 62

19.05 63

18.11 63

Maestro 3rd Party Libraries **64**

Licenses 64

Libraries 64

Check Libraries 65

Maestro Roles Overview **68**

Template Designer 68

Form Builder 68

Log in to Maestro **70**

Session Management 72

Clear Browser Cache 72

Management Dashboard **73**
Welcome Tab 73
Recent Designs 73
Product Information 74
Latest Product Videos 74
Recent Tab 74
Release Tab 75
Navigator Pane 75
Help 76
Assets Save History **78**
Export a Maestro Asset **80**
Import a Maestro Asset **81**
Maestro Editor Overview **82**
Toolbar 83
Form 83
Receipt 83
Template 83
Component 83
Left Panel 85
Right Panel 85
Views 86
Help 86
Design Mode **87**
Rich Text Editor **90**
Icon Picker **94**

Use Image 95
Settings Menu **96**
Design History **99**
Form Resource Summary **100**
Palette Pane **102**
View Pane **106**
Structure 109
Sorting 109
Icons 111
View Component and Property Source **112**
Model Pane **116**
Properties Pane **121**
Styles Pane **124**
Selected Style 125
Styling Target 127
Item Appearance - Container 129
Text 130
Layout 131
Border 132
Background 132
Edit Custom CSS **137**
Integration Pane **140**
TM Form Data 141
Transact Integration 143
Hidden Data 144

Code View 145
Add a New Rule in Code View 147
Move Rules to a JavaScript File 148
Reference a JavaScript File into a Form 151
Download and Upload a JavaScript File 154
Preview Mode 156
Build a TM Form Version 158
Update a TM Form Version 166
Show Form Details 167
Maestro Organizations Overview 169
Organization Structure 171
Organization Set-up 173
Step 1: Create the Shared Components Project 174
Step 2: Create the Style Project 174
Step 3: Create the Shared Components Library 175
Step 4: Create the Style Library 175
View Organizations 176
Projects Overview 180
Create a Project 182
Upgrade a Project 184
Export a Project 186
Import a Project 187
View Project Details 189
View Project Assets and Resources 192
Forms 192

Components 193
Templates 194
Maestromo Libraries Overview 196
Use Case 1: Template Designer 197
Use Case 2: Form Builder 197
View Libraries 198
View Library Versions 201
Library Precedence 203
Library Scopes 205
Shared Organization Library 207
Shared 207
Global 207
How to Share Organization Libraries 208
Manage Shared Organization Libraries 214
Exclude Shared Organization Libraries 216
Un-share Global Organization Libraries 216
Add Sharing Permissions 217
Export a Library 219
Import a Library 220
Publish Resources To Another Library 221
How to Use Organization Libraries 223
Step 1: Create a Library 223
Step 2: Create a Project 226
Step 3: Create a Component 228
Step 4: Publish a Component 230

Step 5: Use a Published Component 232
The Advantage of This Approach 233
View Resources **234**
Load External JavaScript Libraries **238**
Use Page Tracking 238
Load Libraries with jQuery 239
Library Versions **242**
Library Version Numbers 242
Viewing Library Versions 242
Default Initial Library Version 243
Create a Library Version **245**
Delete a Library Version **247**
Map a Library Version to a Project **249**
Create a Tagged Library **251**
Maestro Forms **253**
Maestro Form Architecture **256**
Form View 256
Form Items 257
Form Data 257
Create a Form **259**
Open a Form **262**
Save a Form **263**
Close a Form **265**
Export Maestro Forms **266**
Import Maestro Forms **267**

Migrate Maestro Forms **268**

Change a Maestro Form Description **269**

Test Maestro Forms **270**

Test Checklist 270

Functional 270

Visual 270

Receipt 271

Accessibility 271

Dynamic Data 271

Devices & Browsers 271

Debug Maestro Forms with Chrome DevTools **272**

Prerequisites 272

Building a Form 272

Opening Chrome DevTools 273

Inspecting the Form Object 274

data 275

items 276

view 277

Debugging JavaScript Rules 278

Using Breakpoints 279

Rule Types and JavaScript Function Naming 282

Rule Type Codes 283

Action Rules 284

Dynamic Data Rules 284

Debugging Form Styles 285

Advanced Debugging of Maestro Forms **286**
Recommended Skill Set 286
Debugging Tools 286
Modern Browser 286
Developer Tools Enabled 287
AngularJS Browser Plugin 287
Publishing Forms for Debug 288
Activating a Debug Session 288
Shortcuts for Republishing Forms 289
Inspecting the Form Object 289
Logging an Item to the Console 291
Locating a JavaScript Rule for Debug 292
Rule Types and JavaScript Function Naming 294
Examples 295
UI Test Automation Overview **296**
Selenium & WebDriver 296
UI Testing Key Concepts 297
Issues in UI Testing 298
UI Test Automation of Maestro Forms **300**
Maestro Form.json 301
Form Item Definition 303
Item Definitions and Selenium Locators 305
Statically Typed Definitions 306
Field Validation 307
Data Model Lookup 307

Page Object Model Generation 308
Conclusion 310
Improve Page Load Time of Maestro Forms 311
Check Form Rendering Metrics 311
Check Form Resource Summary 311
Analyze Page Load Time 312
Enable Static Content Externalization 312
Enable GZIP Compression 313
Load Reference Data via Dynamic Data Calls 313
Use Correct Image Formats 314
Compress Images 314
Perform Load Testing 315
What Else? 315
Maestro Form Accessibility 316
Use Sequential Page Navigation 316
Use Page and Section Headers 317
Mark-up Lists 319
Add Alternate Text to Images 321
Decorative 322
Informative 322
Do Not Use Background Images 322
Use Fieldsets 323
Do Not Use Label Placement Left 325
Check Color Contrast 326
Do Not Use Placeholder Text 326

Do Not Replace Component Labels with Text 327
Do Not Disable Anything 328
Use Aria Label 329
Rule Templates and Rule Helpers in a Form 332
Form Options 334
Add a Form Option 336
Add Properties 336
Remove Properties 339
Configure a Form Option 340
Configure Properties 340
Form Name 343
Form Rules 346
User Experience 347
Policies 349
Maguire Template 349
Avalon Template 350
Analytics 351
Accessibility 352
Navigator Second Level 353
Wizard Navigation Bar 354
Form Background Save 355
Enable Form Background Save from Form Options 356
Enable Form Background Save from Content Container 357
Configure Form Background Save Interval 358
Form Save on Page Change vs Timed Save 360

Make a Place to Store the New Metadata 360
Update the Metadata Before Calling the Service 360
Update the Service to Use the New Metadata 361
Form Pages 362
Add a Page to a Form 362
Form Sections and Fieldsets 362
Form Two-Level Navigation 363
Form Versions Overview 365
View Form Versions 367
Form Version Number 371
Compare Form Versions 372
Method 1 372
Method 2 373
Delete Maestro Form Version 374
Export Form Versions 375
Nominate the Current Form Version 377
Open an Earlier Version of a Maestro Form 378
Method 1 378
Method 2 379
Compare Form Versions 379
Method 1 379
Method 2 380
Tagged (locked version) Form Version Overview 381
Create A Tagged Version of a Form 382
Create a New Form Version based on a Tagged Version 383

Change a Maestro Form Version Description **386**

How Applicants Experience a Maestro Form **387**

1. Load a form 387

2. Data is prefilled 387

3. Enter data into fields 388

4. Save and resume 388

5. Submit the form 388

Add Google Tag Manager to a Maestro Form **389**

Import the Widget Library into Maestro 390

Add the GTM Widget to a Form 391

Configure Manager to Enable Analytics 393

Connect the Form to GTM in Manager 394

Additional Forms Pages 394

Content Security Policy (CSP) Settings 396

Configure a GTM account 397

Import the Maestro GTM Container 397

Configure GTM to connect to Google Analytics 399

Specify the Virtual Page View URL Location 400

Navigate and Locate Items in GTM 402

In-form Page Views 403

Other Page Views 403

Autocomplete Fields for Dynamic Data Services **405**

Autocomplete Fields for Transact Functions **415**

Deep clearing of Chrome browser cache **429**

Maestro File Sizes **430**

Access to Form Reference Data **431**
JavaScript Library 431
Property Prefill 432
Dynamic Data Service 434
Add Cascading Dropdowns 437
Maestro Data Models **440**
Data Mapper **443**
Form XML Data Payload **445**
Dynamic Data in Forms **447**
Prefill Dynamic Data 448
Form Prefill Service 449
Prefill Mapping 449
Property Prefill 450
Property Types 450
Organization Properties 450
Form Properties 450
User Profile Properties 450
Prefill Caveats 450
Dynamic Data Services 451
Dynamic Data Button 451
DynamicData.call 452
What to fetch 453
When to fetch dynamic data 453
How to test dynamic data 453
Personal Data 454

Reference Data 455
Volatile Data 455
Accessing Prefill Data 455
Component Data Binding 457
Relative Binding 457
Absolute Binding 458
Exclude a Component 460
Components Using Entity XML Path 461
Check for Duplicate Data Bindings 463
Resolve Duplicate Data Bindings 464
User Mobile Device Detection 467
Components Overview 468
Core (Native) Components 468
Custom Shared Components 469
Custom Native Components 469
Component Common Properties 471
Name 471
XML Name 471
ID 471
Label 472
Rules 473
Layout 473
Component ID Overview 475
Component Identity Properties 477
Change a Component ID 479

Change Component ID 481
Component Labels **485**
Static Component Label 485
Dynamic Component Label 486
Hide a Label 487
Change Label Placement 488
Component Layout **490**
Configure a Component Layout 490
Example Layouts 492
Example 1 492
Example 2 493
Validation Message Layouts **494**
Fixing Vertical Alignment 496
Component Versions **498**
Create a Component Version from the Maestro Dashboard 498
Open a Specific Component Version 499
Method 1 499
Method 2 500
Add a Component **501**
Insert a Component **503**
Insert a component using the Maestro editor 503
Insert a component using the component hierarchy 503
Delete a Component **505**
Delete a component using the context menu 505
Delete a component using the toolbar 507

Delete a component using the keyboard 507
Delete a component version 507
Component Form Dependencies 508
Embedded Component Inheritance 511
Show References 514
Import and Export a Component 517
Repeating Components 518
Repeating Block Calculations 519
Re-usable Components 520
Re-use Components at the Component Level 520
Re-use Components at the Form Level 521
Reuse Components at the Style Level 522
Native Components Overview 524
Attachment Field 525
Usage 525
Properties 526
Error Messages 529
Help Text 530
Success Messages 531
Thumbnail Options 531
Transact Field | 20.11 This feature was removed in 20.11 532
Card Content Template Component 533
Checkbox Component 537
Currency Field 538
Usage 538

Properties 538
Accessibility 538
Help Text 539
Options 539
Data-Driven Dropdown Component **540**
Usage 540
Use Cases 541
Data-Driven Dropdown Component Based on a Repeating Component 544
Data-Driven Dropdown Component Based on a Range Component 546
Data-Driven Dropdown Component Based on Text Input 546
Cascading Dropdowns 547
Decimal Field **548**
Usage 548
Properties 548
Accessibility 548
Help Text 548
Options 549
Dialogs Overview **550**
Dialog Types 550
Creating Dialogs 552
Configuring Dialogs 553
Open a Dialog 553
Fieldset **554**
Usage 554
Properties 555

Help Text 555
Image Component **556**
Link Component **559**
Modal Pages Overview **561**
Modal Page Types 561
Create a Modal Page 562
Configure a Modal Page 562
Open a Modal Page 562
Multiline Text Area Component **563**
Page Component **566**
Percentage Field **568**
Usage 568
Properties 568
Accessibility 568
Help Text 569
Options 569
Radio Button Group Component **570**
Graphical Radio Button **575**
Usage 575
Properties 575
Appearance 575
Image 576
Graphical Radio Button Group **577**
Usage 577
Properties 577

Help Text 577
Icon Button **578**
Usage 578
Properties 578
Accessibility 579
Image **580**
Usage 580
Properties 580
Link **581**
Usage 581
Properties 581
Repeating Block Template **582**
Hide Delete Button for a Single Item in the Repeating Block Template 586
Save Challenge Dialog **589**
Save Confirm Dialog **590**
Select Language **591**
Usage 591
Properties 591
Accessibility 593
Help Text 594
Styles 594
Section Component **597**
Slider **600**
Usage 600
Properties 600

Configuration 600
Range Slider 602
Help Text 602
Text Field Component 603
Shared Components Overview 607
Create a Shared Component 609
Create a Shared Component from the Management Dashboard 609
Create a Shared Component from Existing Form Items 611
Component Options 613
Properties Options 613
Rule Helper Options 615
Rule Template Options 618
Override a Shared Component 621
Edit a Shared Component 623
Publish a Shared Component 625
Custom Native Components Overview 629
Custom Native Component Development 630
Create a Custom Native Component JSON File 630
Create a Custom Native Component HTML File 637
Create a Custom Native Component JavaScript File 639
Create a Custom Native Component jQuery File 640
Package a Custom Native Component 642
Enforcing Required Configuration Properties 642
Create a Native Component Library Automatically 644
Custom Native Component Property Types 645

Native Component Development Toolkit **647**
Installation 647
Using the Toolkit 647
Native Component Development Using Eclipse or IntelliJ **649**
Custom Button 650
Angular Dynamic Button 655
Auto-tabbing Component 657
Pack Resources 659
List of Maestro Components **660**
Attachment Field v2 ..... **666**
Usage 667
Properties 667
Additional Labels 668
Styles 669
Attachment Field **671**
Usage 671
Properties 672
Error Messages 675
Help Text 676
Success Messages 677
Thumbnail Options 677
Transact Field | 20.11 This feature was removed in 20.11 678
Landing Page Image **679**
Usage 679
Properties 679

Display Text 679
Options 680
Page View 680
Navigator 681
Styles 681
Modal Page Logo 682
Usage 682
Properties 682
Display Text 682
Options 683
Page View 683
Navigator 684
Styles 684
Avalon Info Dialog 685
Usage 685
Properties 685
Display Text 685
Options 686
Page View 686
Navigator 687
Styles 687
Progress Bar 688
Usage 688
Properties 688
Display Text 688

Options 689
Page View 689
Navigator 690
Styles 690
Session Timeout Controller 691
Usage 691
Properties 691
Display Text 691
Options 692
Page View 692
Navigator 693
Styles 693
Countries 694
Usage 694
Properties 694
Accessibility 695
Help Text 695
Gender 696
Usage 696
Properties 696
Accessibility 697
Help Text 697
State (Codes) (US) 698
Usage 698
Properties 698

Accessibility 699
Help Text 699
State (US) **700**
Usage 700
Properties 700
Accessibility 701
Help Text 701
State (AU) **702**
Usage 702
Properties 702
Accessibility 703
Help Text 703
Title **704**
Usage 704
Properties 704
Accessibility 705
Help Text 705
Yes/No Radio Buttons **706**
Usage 706
Properties 706
Appearance 706
Help Text 708
Email Address **709**
Usage 709
Properties 709

Accessibility 710
Help Text 710
Mobile Number (AU) **711**
Usage 711
Properties 711
Accessibility 712
Help Text 712
Mobile Number (US) **713**
Usage 713
Properties 713
Accessibility 714
Help Text 714
Phone Number (AU) **715**
Usage 715
Properties 715
Accessibility 716
Help Text 716
Phone Number (US) **717**
Usage 717
Properties 717
Accessibility 718
Help Text 718
Address (Global) **719**
Usage 719
Properties 719

Help Text 719
Address (USA) **720**
Usage 720
Properties 720
Help Text 720
Address (Australia) **721**
Usage 721
Properties 721
Help Text 721
ABN (Australian Business Number) **722**
Usage 722
Properties 722
Accessibility 723
Help Text 723
ACN (Australian Company Number) **724**
Usage 724
Properties 724
Accessibility 725
Help Text 725
Google Address Lookup **726**
Usage 726
Properties 726
Google Configuration 726
Help Text 727
Mastersoft Address Lookup **728**

Usage 728
Properties 728
Help Text 728
Dropdown (Years) **729**
Usage 729
Properties 729
Accessibility 730
Date of Birth Dropdown **731**
Usage 731
Properties 731
Help Text 731
Options 731
Validation 732
Video Capture **733**
Usage 733
Properties 733
Attachment Metadata 735
Configuration 735
TFN (Tax File Number) (AU) **737**
Usage 737
Properties 737
Accessibility 738
Help Text 738
Signature Pad **739**
Usage 739

Properties 740
Help Text 742
Popout Config 742
Button **744**
Usage 744
Properties 744
Accessibility 744
Click To Call **746**
Usage 746
Properties 746
Accessibility 747
Dynamic Data Button **748**
Usage 748
Properties 748
Accessibility 749
Configuration 749
Jump to Issue **751**
Usage 751
Properties 751
Accessibility 752
Icon Button **753**
Usage 753
Properties 753
Accessibility 754
Button Group **755**

Usage 755
Properties 755
Appearance 755
Help Text 757
Toggle Icon Button 758
Usage 758
Properties 758
Card Content 759
Usage 759
Properties 759
Card Content Template 760
Usage 760
Card Selector 761
Usage 761
Properties 761
Form Bundle Job Tasks 763
Usage 763
Properties 763
Dropdown Bundle Navigation 765
Usage 765
Properties 765
Accessibility 766
Help Text 766
Job Details Block 767
Usage 767

Properties 767
Accessibility 767
Job Number **768**
Usage 768
Properties 768
Accessibility 768
Step Name **769**
Usage 769
Properties 769
Accessibility 769
Preview Test Block **770**
Usage 770
Properties 770
Accessibility 770
Route Select **771**
Usage 771
Properties 771
Accessibility 771
Help Text 771
Fieldset **773**
Usage 773
Properties 774
Help Text 774
Block **775**
Usage 775

Properties 775
Accessibility 775
Modal Page **776**
Usage 776
Properties 776
Accessibility 777
Component 778
Google Analytics 778
Transact Manager 779
Dialog **780**
Usage 780
Properties 780
Accessibility 780
Block (Mandatory) **781**
Usage 781
Properties 781
Help Text 781
Options 781
Section (Avalon) **783**
Usage 783
Properties 783
Additional Layout 783
Styles 784
Section **785**
Usage 785

Properties 785
Additional Layout 785
Sticky Panel **787**
Usage 787
Properties 787
Autotab Block **788**
Usage 788
Properties 788
Format 788
Dialog **790**
Usage 790
Properties 790
Accessibility 790
Open Dialog **791**
Usage 791
Properties 791
Accessibility 792
Configuration 792
Dialog Navigation Bar **793**
Usage 793
Properties 793
Show XML **794**
Usage 794
Small Dialog **795**
Usage 795

Properties 795
Accessibility 795
Dialog (Avalon) **796**
Usage 796
Properties 796
Additional Layout 796
Styles 797
Form Help **798**
Usage 798
Save Challenge **799**
Usage 799
Save Confirm **800**
Usage 800
Cancel Confirm **801**
Usage 801
Form Background Image **802**
Usage 802
Properties 802
Form Header **803**
Usage 803
Properties 803
Tracking Code 803
Image **804**
Usage 804
Properties 804

Link **805**
Usage 805
Properties 805
Percentage Indicator **806**
Usage 806
Text Display **807**
Usage 807
Properties 807
Accessibility 807
Form Footer **808**
Usage 808
Properties 808
Accessibility 808
Form Footer (Avalon) **809**
Usage 809
Properties 809
Accessibility 809
Styles 809
LinkedIn Auto-Fill **810**
Usage 810
Properties 810
Api Config 810
Help Text 810
Options 810
Prefill Mappings 811

Checkbox **812**
Usage 812
Properties 812
Help Text 812
Currency Field **813**
Usage 813
Properties 813
Accessibility 813
Help Text 814
Options 814
Data-Driven Dropdown **815**
Usage 815
Properties 815
Accessibility 816
Help Text 816
Decimal Field **819**
Usage 819
Properties 819
Accessibility 819
Help Text 819
Options 820
Dropdown **821**
Usage 821
Properties 821
Accessibility 822

Help Text 822
Graphical Radio Button 823
Usage 823
Properties 823
Appearance 823
Image 824
Radio Button Group 825
Usage 825
Properties 825
Appearance 825
Help Text 827
Radio Button [Repeating] 828
Usage 828
Properties 828
Text Field 829
Usage 829
Properties 830
Accessibility 831
Help Text 831
Percentage Field 832
Usage 832
Properties 832
Accessibility 832
Help Text 833
Options 833

Time **834**
Usage 834
Properties 834
Help Text 834
Options 834
Data-Driven Checkbox Group **836**
Usage 836
Properties 836
Appearance 836
Help Text 837
Data-Driven Radio Group **838**
Usage 838
Properties 838
Appearance 839
Help Text 840
Date Picker **841**
Usage 841
Properties 841
Accessibility 843
Help Text 843
Terminology 843
Validation 844
Validation Messages 845
Autocomplete Field **846**
Usage 846

Properties 846
Accessibility 846
Configuration 847
Data Source 848
Dds Configuration 849
Help Text 850
Second Dds Configuration 850
Graphical Radio Button Group 852
Usage 852
Properties 852
Help Text 852
Slider 853
Usage 853
Properties 853
Configuration 853
Range Slider 855
Help Text 855
Multiline Text Area 856
Usage 856
Properties 856
Accessibility 857
Help Text 857
Options 857
Dropdown Multi Select 858
Usage 858

Properties 858
Accessibility 859
Help Text 859
Horizontal Divide **860**
Usage 860
Cancel Success **861**
Usage 861
Submit Success **862**
Usage 862
Save Success **863**
Usage 863
Modal Mobile Header **864**
Usage 864
Navigator Step Count **865**
Usage 865
Navigation Wizard Bar **866**
Usage 866
Properties 866
Options 866
Navigator **867**
Usage 867
Properties 867
Accessibility 867
Left Hand Navigation 868
Second Level Navigator **869**

Usage 869
Properties 869
Accessibility 869
Tramline Navigator **871**
Usage 871
Properties 871
Tramline Images 871
Navigator (Avalon) **872**
Usage 872
Properties 872
Accessibility 872
Left Hand Navigation 873
Styles 873
Page Group **874**
Usage 874
Properties 874
Display Text 874
Page Group (Avalon) **875**
Usage 875
Properties 875
Display Text 875
Styles 875
Navigator Wizard Bar **876**
Usage 876
Properties 876

Options 876
Page Controller **877**
Usage 877
Properties 877
Accessibility 878
Page Controller (Avalon) **879**
Usage 879
Properties 879
Accessibility 880
Styles 880
Page **881**
Usage 881
Properties 881
Display Text 881
Options 881
Page (Avalon) **882**
Usage 882
Properties 882
Display Text 882
Options 883
Page View 883
Navigator 884
Styles 884
Save and Close **887**
Usage 887

Repeat Index Text **888**
Usage 888
Properties 888
Accessibility 888
Repeat Add Button **889**
Usage 889
Properties 889
Repeat Delete Button **890**
Usage 890
Properties 890
Repeat **891**
Usage 891
Properties 891
Button Options 892
Repeating Block Template **894**
Usage 894
Properties 894
Accessibility 894
Repeating Block Template (Deprecated) **895**
Usage 895
Data Field **896**
Usage 896
Bundle Resource Library **897**
Usage 897
Properties 898

Bundle Resource Library 898
Runtime CSS Resource **900**
Usage 900
Properties 901
Runtime CSS Resource 901
Web Font **903**
Usage 903
Properties 906
Web Font 906
JavaScript Library **909**
Usage 909
Properties 909
Countdown Timer **910**
Usage 910
Properties 910
Options 910
Rules 911
Rule Helpers 912
Autotab Controller **914**
Usage 914
Properties 914
Format 914
Form **916**
Usage 916
Properties 916

Accessibility 918
Component 918
Google Analytics 918
Transact Manager 919
Receipts | 19.05 This feature was introduced in 19.05. 919
Rules 921
Layout 921
Submit **922**
Usage 922
Properties 926
Accessibility 926
Submit Button 927
Payment Type Selector **928**
Usage 928
Properties 929
Help Text 930
Image Resources 930
Payment Total 931
Supported Credit Cards 931
iFrame **933**
Usage 933
Properties 933
Advice Block - Info **934**
Usage 934
Advice Block - Promo **935**

Usage 935
Popover Text Display 936
Usage 936
Properties 936
Help Text 936
Popover Options 936
Popover Widget Container 937
Usage 937
Select Language 938
Usage 938
Properties 938
Accessibility 940
Help Text 941
Styles 941
Maestro Receipts Overview 944
Receipt Designer 945
Create a form 945
Create a receipt 946
Download the Form Version Archive file 950
Deploy the form to Manager 951
Test the receipt 951
Build a Receipt 954
Shared Styles 956
Create a Shared Style 957
Make a Shared Style Available 960

Apply a Shared Style **961**

Add a Shared Style **965**

Add a Shared Style to Multiple Components 965

Move a Shared Style **966**

Remove a Shared Style **967**

Template Overview **968**

Common Template Elements **972**

Avalon Template Overview **974**

Template Elements 975

Template-Specific Components 976

Avalon Components 976

Containers 976

Dialogs 976

Display Fields 976

Navigation 976

Avalon Template Structure **977**

Template's Components 977

Dialogs 979

Modal Pages 980

Avalon Template JSON Definition **981**

Maguire Template Overview **982**

Create a Template **984**

Create a Template 984

Set Extension Points 985

Publish the Template 986

Customize the Template ... 986
Tag a Template ... 987
Publish a Template ... 990
View Templates ... 992
Export a Template ... 995
Import a Template ... 996
Update Template Default Properties ... 997
Change a Template of a Form ... 999
Brands Overview ... 1003
Design-time branding ... 1006
Runtime branding ... 1006
Create a Brand ... 1008
Delete a Brand ... 1010
Apply a Brand to a Form ... 1011
Brand Theme Options ... 1013
Custom Theme Option ... 1015
View Custom Theme Options ... 1015
Create Custom Theme Options ... 1016
Brand Item Properties ... 1018
Change the Style of Components in Brands ... 1024
Runtime Branding ... 1025
JSON Structure ... 1025
css ... 1026
images ... 1026
translation ... 1027

translation_template 1027
dataValues 1027
Rules 1028
API 1029
Extension Points Overview 1030
Set an Extension Point 1032
Remove an Extension Point 1035
Setting Form Options for Extension Points 1037
Save Challenge Overview 1039
Create a Save Challenge 1041
Adding a Security Question 1042
Troubleshooting Dialogs 1045
Configure Security Questions for Save Confirm Dialog 1047
Configure Security Questions in Save Challenge Dialog 1048
Populate Security Questions from User Entered Data 1050
Setting up Security Questions with Multiple Components in the Save Confirm Dialog 1054
Configure Security Questions for Multiple Components in Save Challenge Dialog 1057
Populate Multiple Security Questions from User Entered Data 1060
Save Challenge for Multiple Components Best Practices 1061
Maestro Business Rules 1062
Types of Maestro Business Rules 1063
Introduction to Maestro Business Rules 1064
Values in Scope within a Maestro Rule Function 1064
Statements vs Expression 1064
Parameters 1065

Navigating the Data Model 1065
Navigating the View Model 1066
Global API Objects 1066
$rootScope 1066
$q 1066
$timeout 1067
Form 1067
Calc 1067
Resource 1067
Util 1067
DynamicData 1067
Scroll 1067
Insights 1068
Translation 1068
Other Shortcut Variables 1068
value 1068
Rules 1068
T 1069
SystemProfile 1069
Job 1069
Create a Maestro Business Rule 1070
Configure a Maestro Business Rule 1073
Business Rule Data Object 1076
Business Rule Info Object 1078
Business Rule Item Object 1081

Create a Form Load Rule 1082
How to Debug Maestro Business Rules 1083
Tips for Writing Rules 1084
Text Display vs Dynamic Content 1086
Calculation Rules 1087
Mathematical Calculation 1087
Duplicate Data 1087
Preset Data 1087
Concatenate Multiple Components 1088
Concatenate Multiple Components Rule 1089
Duplicate Data Calculation Rule 1091
Mathematical Calculation Rule 1093
Repeating Blocks Calculation Rule 1096
Editability Rule Overview 1098
Create a Read-Only Rule 1099
Editable If Rule Overview 1100
Create an Editable If Rule 1101
Salary Editable If Rule Example 1102
Mandatory Rule Overview 1106
Mandatory If Rule Overview 1107
Create a Mandatory If Rule 1108
Mandatory If Rule Credit Card Application Example 1109
Make a Component Mandatory 1112
Validation Rule Overview 1114
Create a Regular Expression Rule 1115

Create a Valid If Rule 1117
Basic Valid If Rule with Default Error Message 1118
Basic Valid If Rule with Custom Error Message 1118
Valid If Rule with multiple conditions 1118
Valid After Change If Rule Overview 1119
Create a Valid After Change If Rule 1120
Australian ID Validation Rule Example 1121
Exchange Rate Valid After Change If Rule Example 1126
Visibility Rule Overview 1130
Create a Hidden Visibility Rule 1131
Create a Hide on Receipt Rule 1132
Hide or Show on Receipt Overview 1133
Create a Show on Receipt Rule 1134
Show on Receipt Example 1135
Create a Show If Rule 1138
Show If Rule Examples 1139
Insert Expression in a Visibility Rule 1141
Translation 1142
Configure a New Language 1143
Translate a String in a JavaScript 1148
Translation CSV Files with One Language 1150
Translation CSV Files with Multiple Languages 1153
Maestro Right-to-Left Language Support 1156
Entity Overview 1157
Create an Entity 1159

Delete an Entity **1162**

Apply an Entity Path of a Component to its Children **1163**

Map a Component to an Entity Property **1165**

Map a Repeating Block Component 1167

Domain Model Overview **1170**

Create a Domain Model **1172**

Import a Domain Model **1175**

Share a Domain Model **1177**

Build a TM Form Version **1181**

TM Form Version Build Errors **1189**

A Form with This Name Already Exists 1189

Duplicate Binding 1189

For Composer Users 1190

Update a TM Form Version **1191**

Maestro SCM Overview **1192**

Getting Started 1192

Using Maestro SCM 1192

Best Practices for Maestro SCM Feature Branch Workflow **1195**

Introduction 1195

Prerequisites 1196

Maestro SCM Feature Branch Workflow 1196

Part 1: Getting Started 1197

Part 2: Feature Branches 1198

Part 3: Feature Development 1199

Part 4: Merging Features 1200

Part 5: Project Cleanup 1201
Getting Help 1201
Maestro Offline Build Tool Overview 1203
Migrate to a SCM Project 1204
Migration Challenges 1204
Migration Strategies 1205
Recreate the original project 1205
Converting a legacy project to the SCM project type 1205

Journey Maestro Overview

MaestroThe UI design product. | Form Builder ▪ Platform Developer | All versions This feature is related to all versions.

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 Manager platform, so it is seamlessly integrated with all other products to give you a maximum edge in creating web-based apps using latest technologies.

> NOTE
>
> Maestro 5.0 is the first version of our Maestro tool. It was versioned as 5.0 to keep it in line with Journey Manager version numbering. Therefore, Maestro forms are not supported in Manager prior to 5.0.

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

---

[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.

- Define rules that run arbitrary JavaScript code when events, such as a click, occur
- Maintain multiple translations of a single form
- Configure form accessibility

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

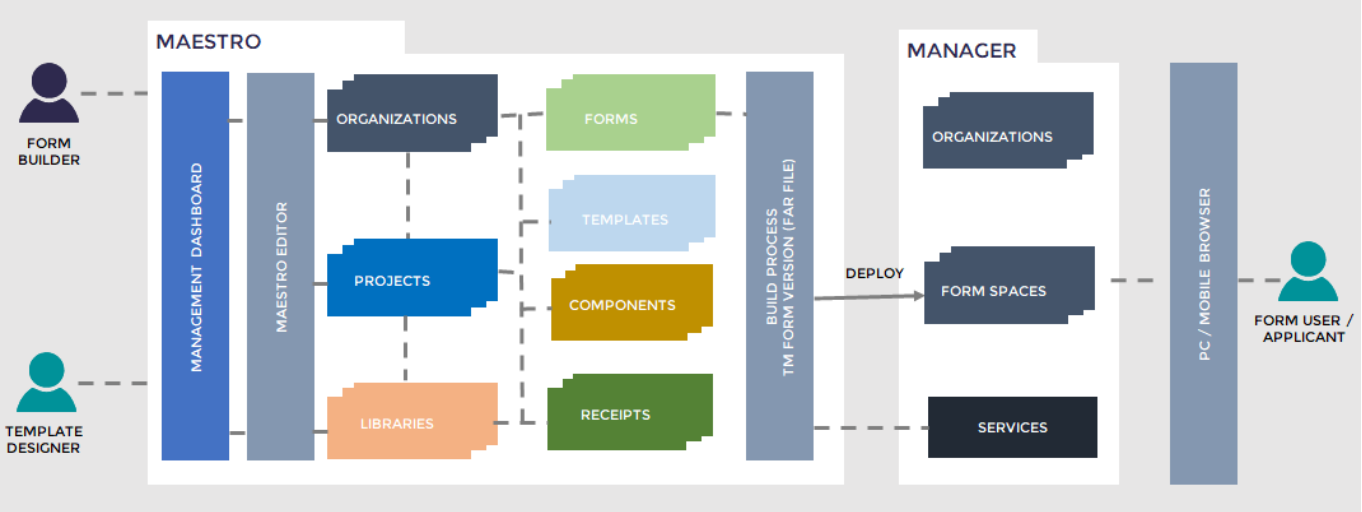
Maestro uses the following entities to enable you to design and build application forms efficiently:

- Organization - contains projects and organization level libraries
- Project - contains forms, components, templates and project level libraries
- Form - contains components to captured user data and pass it to Manager
- Template - a super form that defines and controls the look and feel of forms and receipts based on this template
- Receipt - a read-only statement for various stages of form application processes
- Component - a modular building block that can be dragged and dropped into all types of designs, including forms, receipts, templates, and even other components
- Library - related resources, such as components and shared components, images, core libraries, which can be used across different projects and even organizations

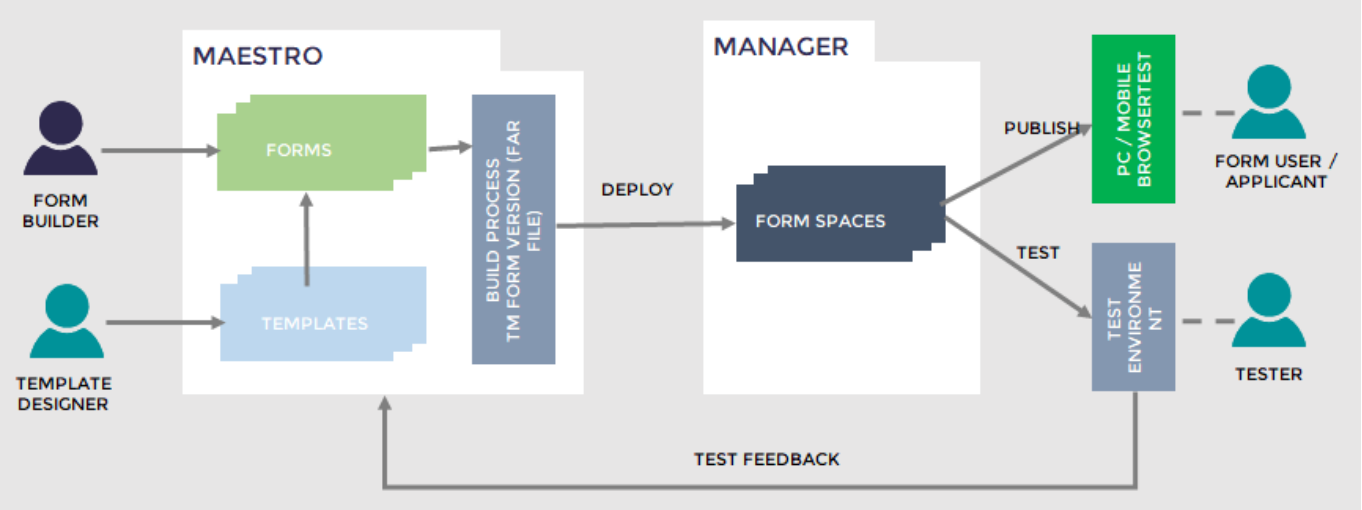
Organizations, projects and forms make up the three-tired hierarchy that allows you to minimize configuration effort yet maximize development efficiency by using a correct scope while defining and configuring a form.

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.

Maestro forms move through various events of the journey experience, which is a sequence of events that include prefill, data capture, save and resume, payments, notifications, and collaboration activities between parties.

Next, learn about Management Dashboard.

Maestro Features

MaestroThe UI design product. | Form Builder " Platform Developer | 23.04 This feature was updated in 23.04

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.

[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.

23.04

Release Key Features	Description
Libraries	Updated 3-rd party libraries and dependencies, including security updates

For more information see 23.04 Release Notes.

22.10

Release Key Features	Description
Libraries	Updated 3-rd party libraries and dependencies, including security updates
Form building	Improved the Offline / Online form building process
New look and fee	Rebranded the Maestro product according to Temenos' vision, mission and strategy

For more information see Maestro 22.10 Release Notes.

22.04

Release Key Features	Description
Libraries	Upgraded the Font Awesome library to version 5.15.4 from 4.5.0, so many new icons are now available in the icon picker in the rich text editor
	Updated 3-rd party libraries and dependencies, including support for ECMAScript 2019
	Improved accessibility in several included widgets
Components	Enhanced the validation framework to properly include component validation on pages that were not visited

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
	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.
Maestro Dashboard	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 rebranded 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
Components	Video Capture component added to provide you with a reliable user verification option
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.

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

Next, learn more about Manager features.

Maestro 3rd Party Libraries

MaestroThe UI design product. | Form Builder Template Designer | 23.04 This feature was updated in 23.04

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

Licenses

Maestro uses the 3rd party libraries according to their term and conditions, which you can find on their web sites listed in the table below.

Libraries

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

Library Name	19.11	20.05	20.11	21.05	21.11	22.04	22.10	23.04
AngularJS	1.5.9	1.8.2	1.8.2	1.8.2	1.8.2	1.8.7	1.9.0	1.9.0
Babel	6.26.0	6.26.0	6.26.0	6.26.0	6.26.0	7.16.12	7.16.12	7.16.12
FontAwesome	4.5.0	4.5.0	4.5.0	4.5.0	4.5.0	5.15	5.15.4	5.15.4
jQuery	3.5.0	3.5.0	3.5.1	3.6.0	3.6.0	3.6.0	3.6.0	3.6.0
jQuery UI	1.12.1	1.12.1	1.12.1	1.12.1	1.12.1	1.13.1	1.13.2	1.13.2
CK Editor	4.12.1	4.12.1	4.12.1	4.16.0	4.16.2	4.18.0	4.18.0	4.18.0
Less	1.7.3	1.7.3	1.7.3	1.7.3	1.7.3	1.7.3	1.7.3	1.7.3
Moment JS [1]	2.22.1	2.22.1	2.22.1	2.29.4	2.29.4	2.29.4	2.29.4	2.29.4

1 - A library is used in published forms only. A published form is a form that is built and rendered to a user in a browser.

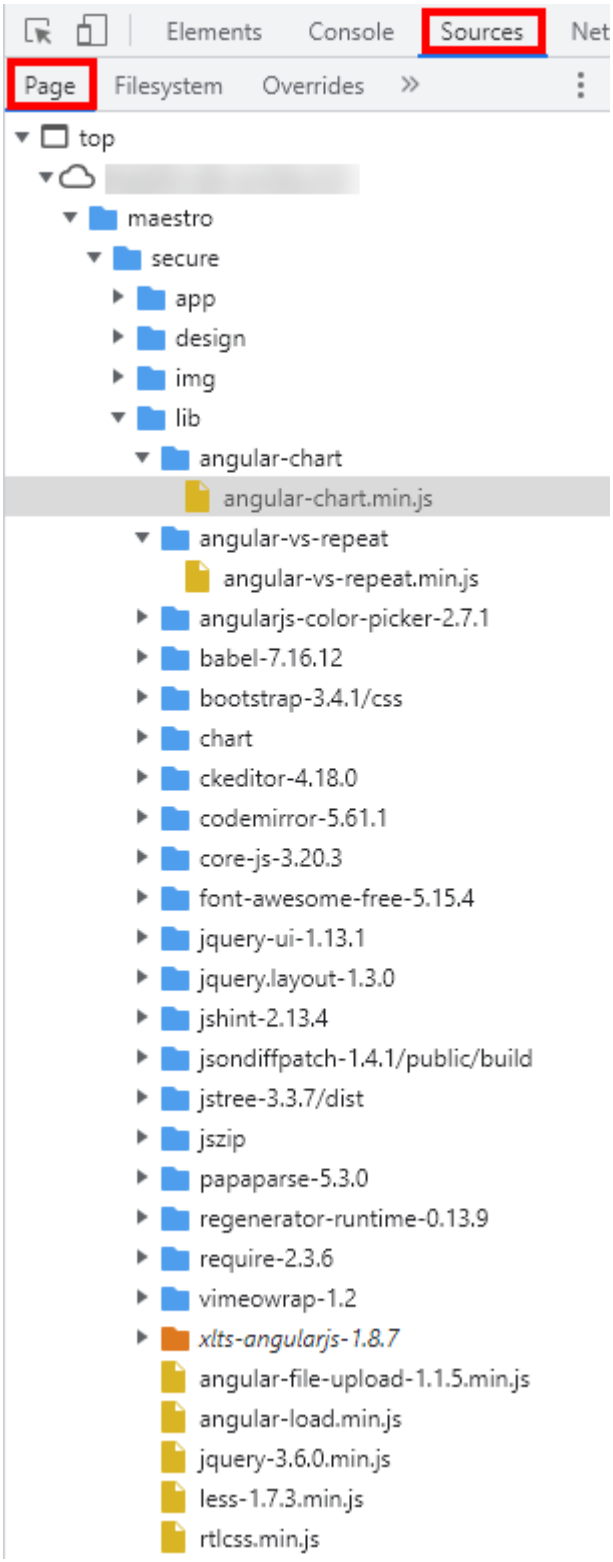
If you use one of End-of-Life Maestro versions, you can check their 3rd party libraries here.

Check Libraries

You can find out the complete list of libraries and their versions for each form for the release version that the form is built in.

To check what libraries a form uses:

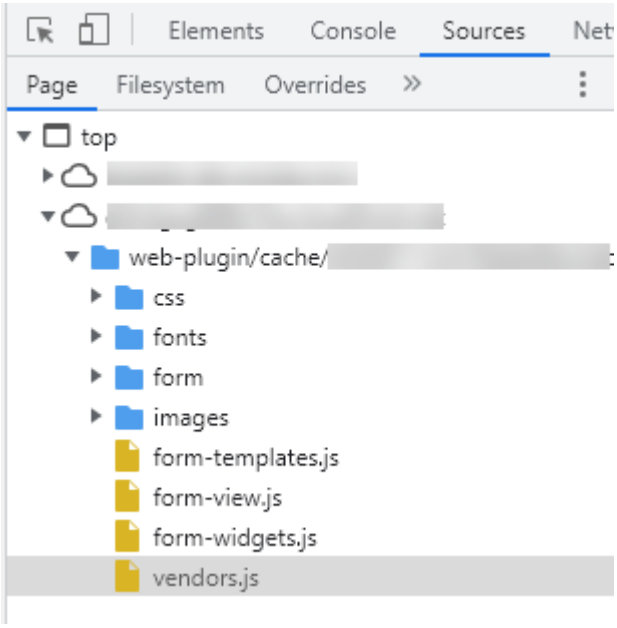
1. Open a form in the Maestro editor.
2. Open a browser's Development Tool, for example, select Chrome menu > More Tools > Developer Tools in the Chrome browser.
3. Select the Sources tab and click Page.
4. Expand the hierarchy under the server host name.
5. Select maestro > secure > libs to list all 3rd party libraries and their versions the form was built with.



6. Expand each library's folder to see its JavaScript file where you can find some additional info, such as what license it uses.

To check what libraries a published form uses:

1. Open a form in a browser.
2. Open a browser's Development Tool, for example, select Chrome menu > More Tools > Developer Tools in the Chrome browser.
3. Select the Sources tab and click Page.
4. Expand the hierarchy under the cloudfront.net name.



5. Double-click the `vendors.js` to view the source code with 3rd party libraries, which are used in published forms.

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

Maestro Roles Overview

MaestroThe UI design product. | Form Builder Template Designer | All versions This feature is related to all versions.

Although a single person is capable of creating a form with Journey Maestro, we generally recommend that at least two people are involved in the form creation process. We refer to these people as the Template Designer and the Form Builder.

Template Designer

Template Designers are responsible for how the form looks. They're the ones creating the graphics, choosing the colors, and making sure the final result is pixel-perfect, works across a range of display sizes, and follows the organization's style guides.

An effective Template Designer asks the following questions:

- What are the best ways to communicate meaning through color, layout, and typography?
- Can users with disabilities navigate the design? Or is it biased toward the able-bodied?
- How adaptable is the design to alternative contexts, such as different products or white-labeling opportunities?

If a form doesn't look good, the Template Designer will need to go back to the drawing board.

Form Builder

Form Builders are responsible for how the form feels. They're the ones laying out the fields, defining how the fields are validated, and constructing the overarching journey the users take while completing the form.

An effective Form Builder asks the following questions:

- Do the field validations work as expected?
- Are any of the steps confusing? Can they be made simpler?
- How can users be motivated to complete the form?

If the form isn't performing as expected – if user abandonment is high, for instance – the Form Builder will be the one taking the heat.

We suggest splitting the form development process between these roles because, while the roles are related, designing how a form looks and how a form functions are distinct tasks that

require distinct skills. Even if someone in your organization has the skills to handle both of them, our experience suggests that it's difficult to effectively prioritize both tasks. It seems inevitable that the look or function of the form will suffer if both takes are handled by the same individual.

Next, learn about Maestro organizations.

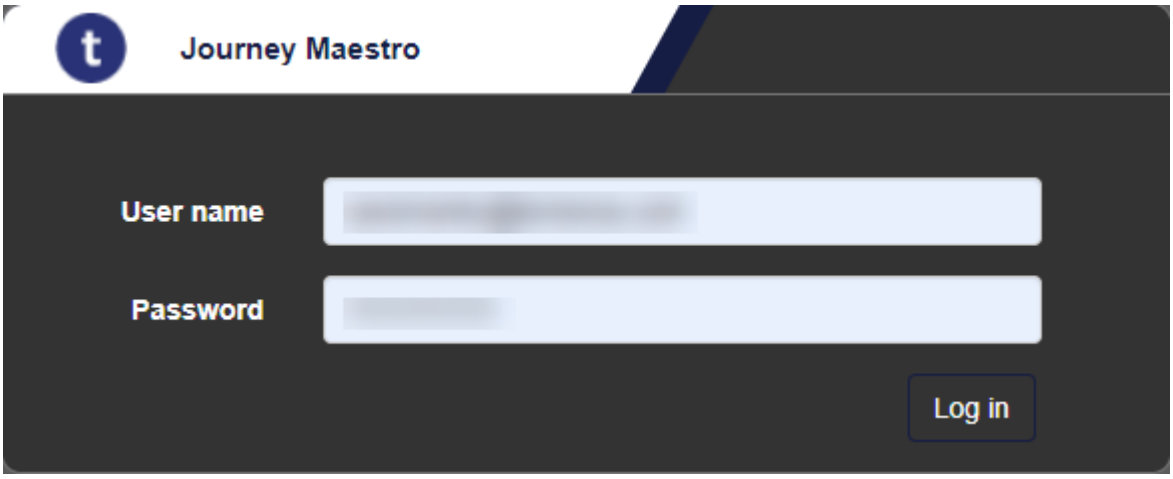
Log in to Maestro

MaestroThe UI design product. | Form Builder Platform Developer | All versions This feature is related to all versions.

Maestro requires all users to authenticate using a username and password before allowing them access to forms. A Journey Manager Administrator must create a user account for you and provide you with a username and password.

To log in to Maestro:

1. Go to `https://<HOST_NAME>/maestro/secure/dashboard` in your browser.



2. Enter your user name in the User name field.

   NOTE

   Your username is the email address associated with your account.

3. Enter your password in the Password field.

   NOTE

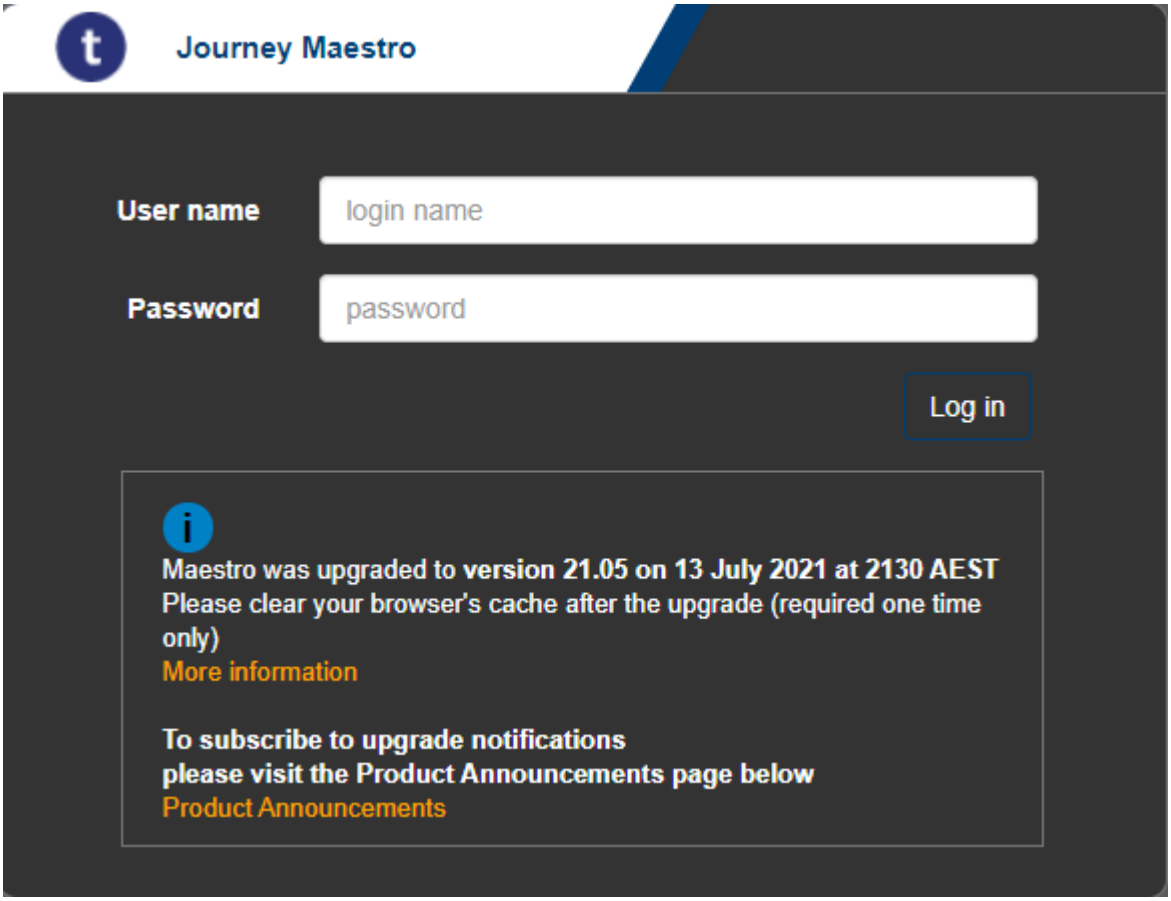
   If you don't have a password, ask your Journey Manager Administrator to get one.

4. Click Log in. If the login is successful, you are shown the Maestro home page. Otherwise, a error message is displayed, for example, Invalid User Name or Password, please try again. Please ensure Caps Lock is off.

NOTE

If you have any problems with login, contact your Journey Manager Administrator or our support team.
If you've forgotten your password, contact your Journey Manager Administrator and request a new one. You can't reset your own password.

The login screen can display some extra information, such as latest Maestro upgrade announcements, so you may be required to perform additional actions, for example, clear your browser's cache.



NOTE

You only have to clear your browser's cache once after an upgrade. The upgrade message, however, will continue to appear on return visits.

To log out of Maestro, click Logout. Your current session will be closed, so you must log in again to continue with your work.

Session Management

Maestro creates a session when a user logs in, which is valid for a limited time to prevent unauthorized access to the environment. There are a few important things about session management:

- Maestro sessions are only valid in a browser a user used to log in first. Maestro stores the session information as cookies in the browser. These cookies are shared among all the windows and tabs of the browser you first logged in with, but not with other types of browsers. For example, if you initially logged in to Maestro using Safari, you can open a new Safari tab or window and continue to use Maestro without logging in again. However, if you also try to access Maestro using Chrome, you'll be asked to re-authenticate with your Maestro's username and password.
- Maestro sessions expiration. A configuration setting in Journey Manager determines the session validity duration. When a session expires, Maestro pops up a new window and asks you to re-authenticate yourself, so you can return to the original window to continue working without losing any unsaved changes.

Clear Browser Cache

After upgrading to a new version of Maestro, the previous version of the application may still be cached in your browser, possibly leading to unpredictable behavior. To address this issue, we recommend clearing your browser cache after each upgrade.

The process of clearing your cache depends on your browser, so check the following links for further details:

- Google Chrome
- Microsoft Edge
- Mozilla Firefox
- Safari

Next, learn about the Management Dashboard

Management Dashboard

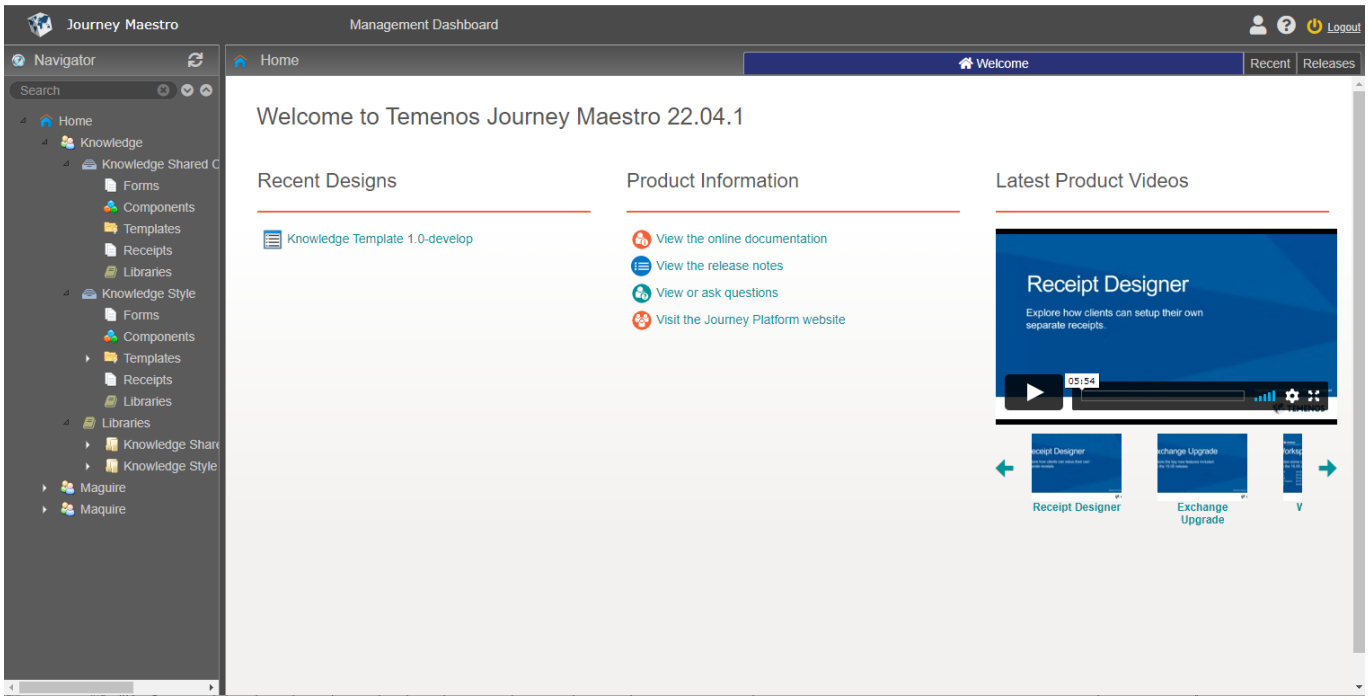
MaestroThe UI design product. | Form Builder ▪ Platform Developer | 21.05 This feature was updated in 21.05.

Maestro provides you with the Management Dashboard packed with useful features that help you perform day-to day tasks easier and quicker. The Management Dashboard is the home page where you land after a successful login to the Maestro editor.

The dashboard includes the following tabs that group related functionality and the navigation bar.

Welcome Tab

Select the Welcome tab to view the following:



Recent Designs

This list displays up to 10 of the most recently viewed designs, for example, a form or component. In order to maintain optimal software performance, Maestro has the total limit of recently opened forms set to 50.

Selecting a design will direct you to the latest Version Details for that design. You can view details about the design by pointing your mouse at the design name. These details include the

organization and project where the design is stored, the design name, version number, and the time and date the design was last opened.

Product Information

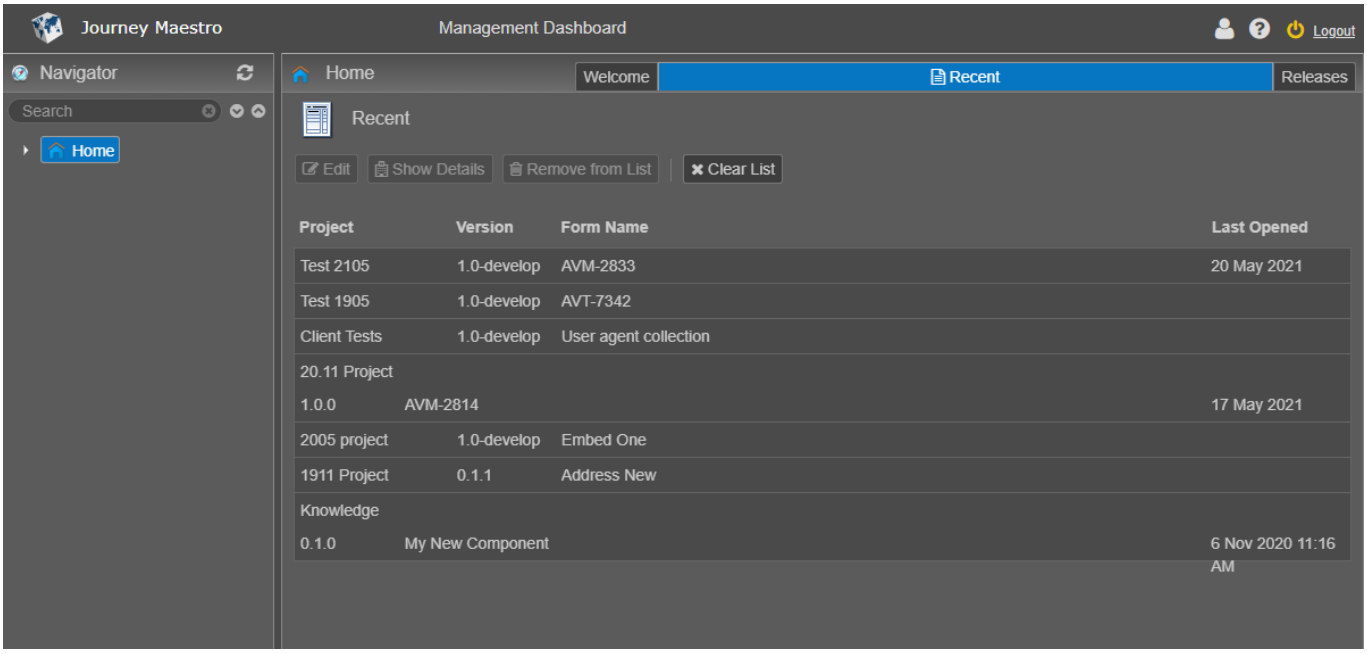
This list displays links to various online resources where you can get information about Temenos Journey Manager platform products.

Latest Product Videos

This list shows links to various online tutorial videos.

Recent Tab

Select the Recent tab to see your recent projects so you can easily go back and continue working on them.



Click Edit to open a selected design in the Editor window.

Click Show Details to see a selected project's details.

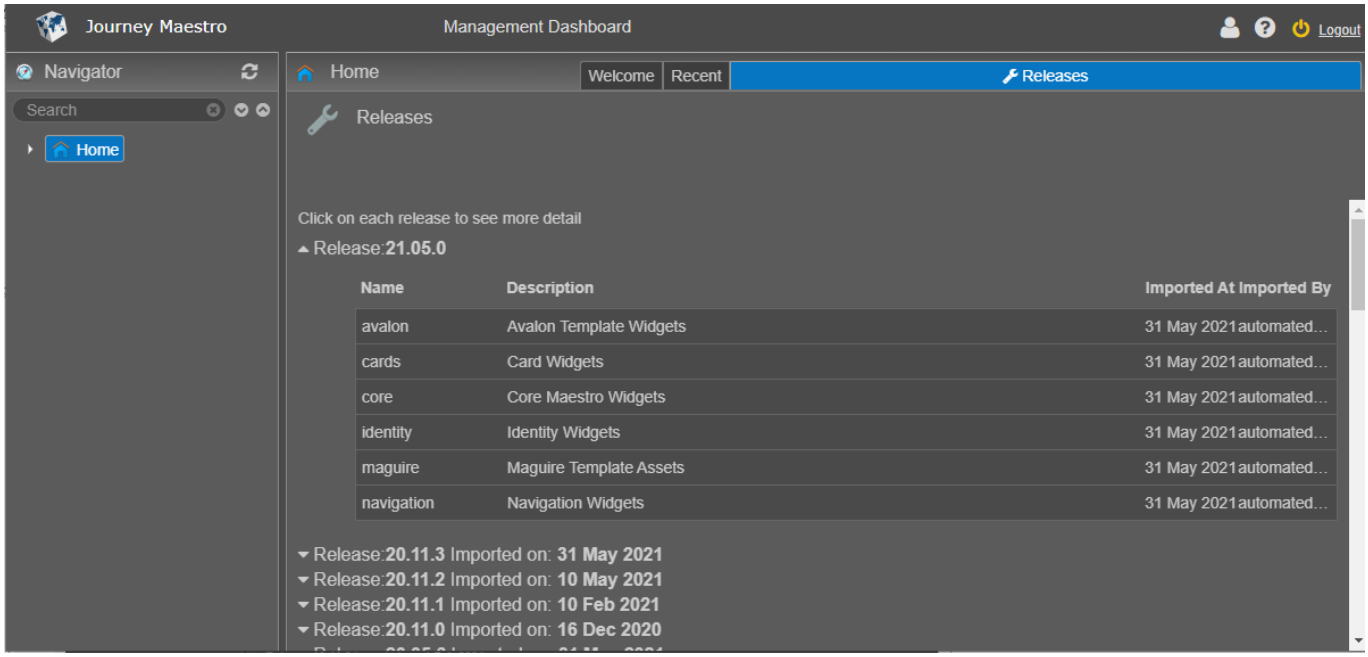
Click Remove from List to remove selected designs from the list.

> NOTE
>
> Use `Shift` or `Ctrl` key to select multiple projects. | 21.05 This feature was introduced in 21.05.

Click Clear List to clear the list completely.

Release Tab

Select the Releases tab to see which releases of Maestro software were installed on your Maestro instance.



Select a particular release to view detailed information on what libraries were changed. This is important for form compatibility and it helps in troubleshooting form issues.

Navigator Pane

The Navigator pane displays the organizations' structure consisting of projects, libraries, their assets and resources, which is available to you. If you can't see some organizations, it is because they are not assigned to your user account.

Select Home to show or collapse the list of organizations.

Type a name or a part of a name of a project, a form , or a library in the search field to quickly find it in the organizations structure.

Click the refresh icon  to reload the Navigator pane content.

Select an organization to view and manager its projects.

Select a project to view and manage its forms and associated resources.

Select Forms to view and manager forms of a project.

Select Components to view and manager components belonging to a project.

Select Templates to view and manager templates used a project.

Select Receipts to view and manager receipts of a project using Receipt Designer. It is a different mode in the Maestro editor that has certain things removed for receipts.

Select Libraries to view and manager libraries and their versions used a project.

Help

Maestro comes with built-in documentation, which are accessible in each development environment, as well as the links to the Temenos Journey Manager documentation web site.

To access Maestro documentation, click the help icon  and select one of the following:

- About Journey Maestro - shows the product information including a version.
- API Guide - redirects you to the Temenos Journey Manager documentation web site.
- API Reference - shows the Maestro API reference documentation shipped with this product version. The same documentation may be available on the documentation web site. However, we recommend using the local version of the documentation as it is tailored to your Maestro version.
- Online Help - redirects you to the Temenos Journey Manager community web site.
- Keyboard Shortcuts - lists the available keyboard shortcuts for Windows and macOS.

Action	Windows	macOS
Cut	CTRL + X	CMD + X
Copy	CTRL + C	CMD + C
Paste	CTRL + V	CMD + V
Undo	CTRL + Z	CMD + Z
Redo	CTRL + Y	CMD + Y
Publish	CTRL + P	CMD + P
Save Form	CTRL + S	CMD + S
Move to Next Item	CTRL + Enter	CMD + Return
Stop Progress / Close Dialog	ESC	ESC

NOTE

You can use Copy and Paste shortcuts across different forms. However, it may depend on each browser's ability to access the system clipboard. For more information, see browser compatibility.

Next, learn how to save history.

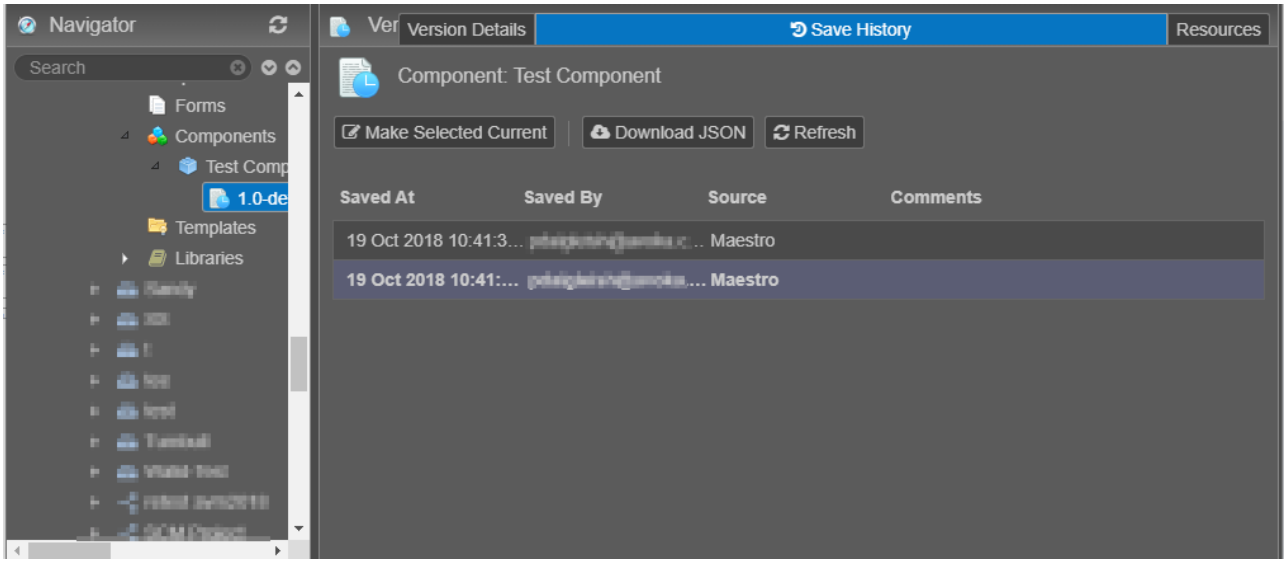
Assets Save History

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 21.05 This feature was updated in 21.05.

Every time Journey Maestro saves an asset, such as forms, templates, receipts and components, its compressed copy is sent to Manager where it is kept for historical references. This allows you to view assets' saved history and restore assets to previous saved states.

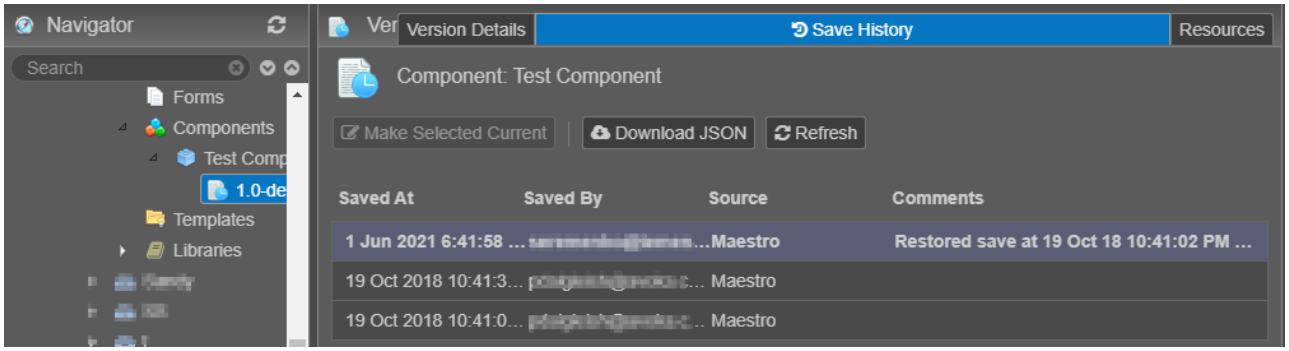
To view saved history of an asset:

1. Open the Management Dashboard.
2. Find an asset's version you want to view and select the Save History tab.



   The list of saved assets is shown of chronological order, with most recently saved asset on top, and includes the following information:

   - Saved At
   - Saved By
   - Source
   - Comment
3. Click Make Selected Current to make the selected historical asset current. You can use this feature to revert the selected asset to its previous save state. A copy of the previous saved instance of the asset will be created and added to the top of the list. A comment is added next to the saved instance that indicates which previous save was restored and is now the current instance of the asset.



4. Click Download JSON to download selected assets as JSON[1] files on your PC. You can use this functionality to compare previously saved instances of assets. Just download JSON files of assets you want to compare and then use an external tool to compare these files and see the differences.

   NOTE

   Use `Shift` or `Ctrl` key to select multiple libraries. | 21.05 This feature was introduced in 21.05.

5. Click Refresh to refresh the list.

Next, learn how to view assets' resources.

[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.

Export a Maestro Asset

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro allows you to export assets, such as forms, receipts, and templates, so you can easily move or copy the assets to a different environment. Once an asset has been exported, you can import it into the original project, which will create a copy of the asset, or you can import it into a different project, If you want to move the asset,

To export an asset:

1. Open a project in the Navigator pane and select a folder where the asset is stored.
2. Select the asset and then click Export. This will download the asset in a zipped folder to your computer.

Next, learn how to import an asset.

Import a Maestro Asset

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to import assets, such as forms, receipts, and templates, which is handy when you want to move or copy the assets to different environments. Once an asset has been exported, you can import it either back into the location it came from or a new location.

To import an asset:

1. Open a project in the Navigator pane and select the asset folder where you want to import the asset.

   NOTE

   It is important to import the asset into the correct location. For example, if you export a form, you must import the form into the project's forms folder.

   NOTE

   You can't import a form into the template's folder.

2. Click Import.
3. Choose the exported asset as the archive ZIP file and click Open.

   NOTE

   If an asset with the same name already exists, a number will be added to the end of the asset name. You can always edit the asset's name in the Details tab.

Next, learn how to export an asset.

Maestro Editor Overview

MaestroThe UI design product. | Form Builder | 18.05 This feature was updated in 18.05.

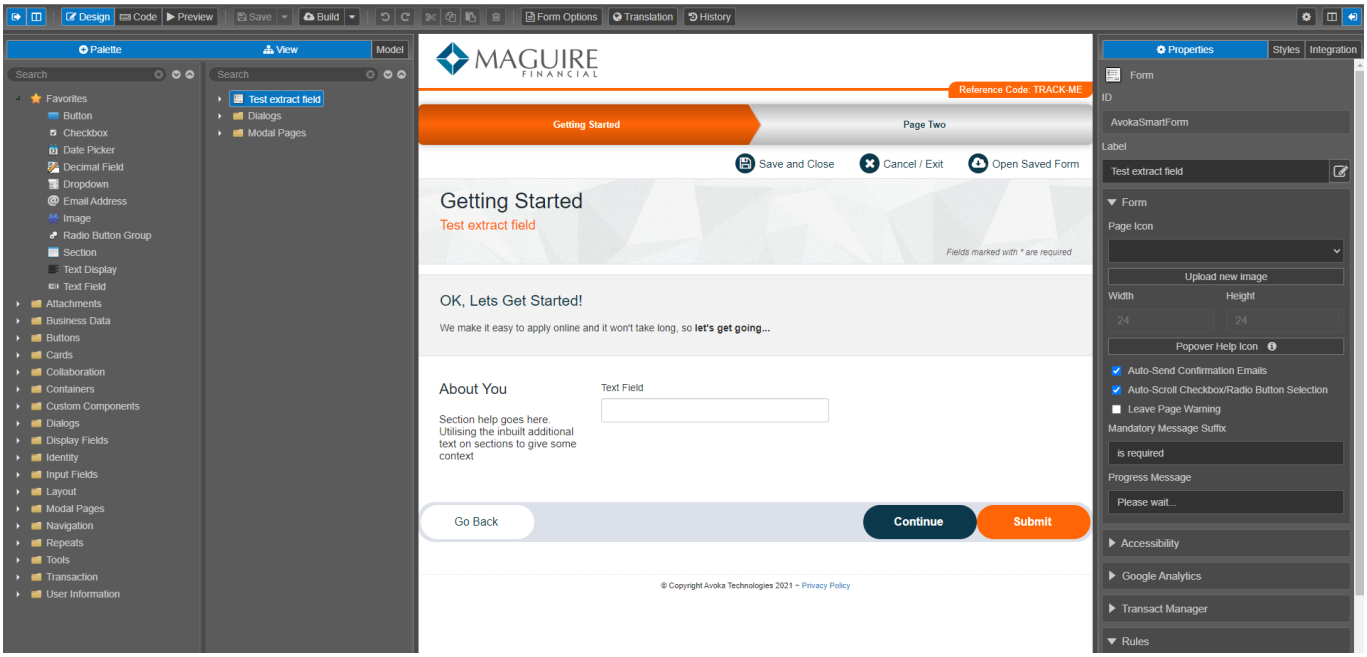
Maestro comes with the Maestro editor for creating and managing forms, components, templates, receipts and related assets and libraries. You use this editor to build complex web applications, such as onboarding customer journeys.

You can create four types of designs:

1. Forms - web applications designed to capture data from applicants.
2. Receipts - documents of record that provide organizations and applicants with a reference of their submitted data.
3. Templates - used to provide a consistent look and feel to multiple forms and receipts.
4. Components - modular building blocks that can be drag and dropped into forms, receipts, and templates.

For more information, see design types.

When you view forms or assets in Maestro and want to start working on a selected form, you click Edit Form Version that opens the form in the Design mode in the Maestro editor.



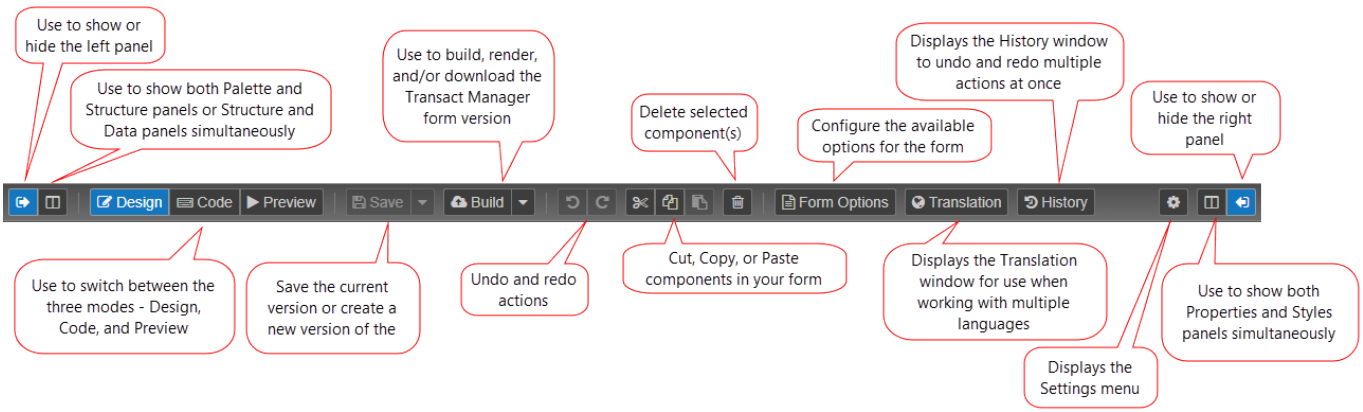
You visually arrange components within a design by dragging and dropping components and updating their properties using the central part of the editor - the Wireframe, which is Maestro's WYSIWYG editor.

The Maestro editor consists of several panes allowing you to perform specific editing and configuration tasks. The editor comes with the default panel structure and layout, which is configured to display one pane on each side of the editor's central window. However, you can use the toolbar functions to change the layout to display more information or less depending on your editing task at hand.

Toolbar

The Toolbar is located at the top of the editor. It contains a range of options for interacting with the current design. Most of the options are available for all types of designs, but some options are only available based on the type of design being edited. It has a slightly different layout when you work on a form, a component, or a template, as shown below:

Form



Receipt



Template



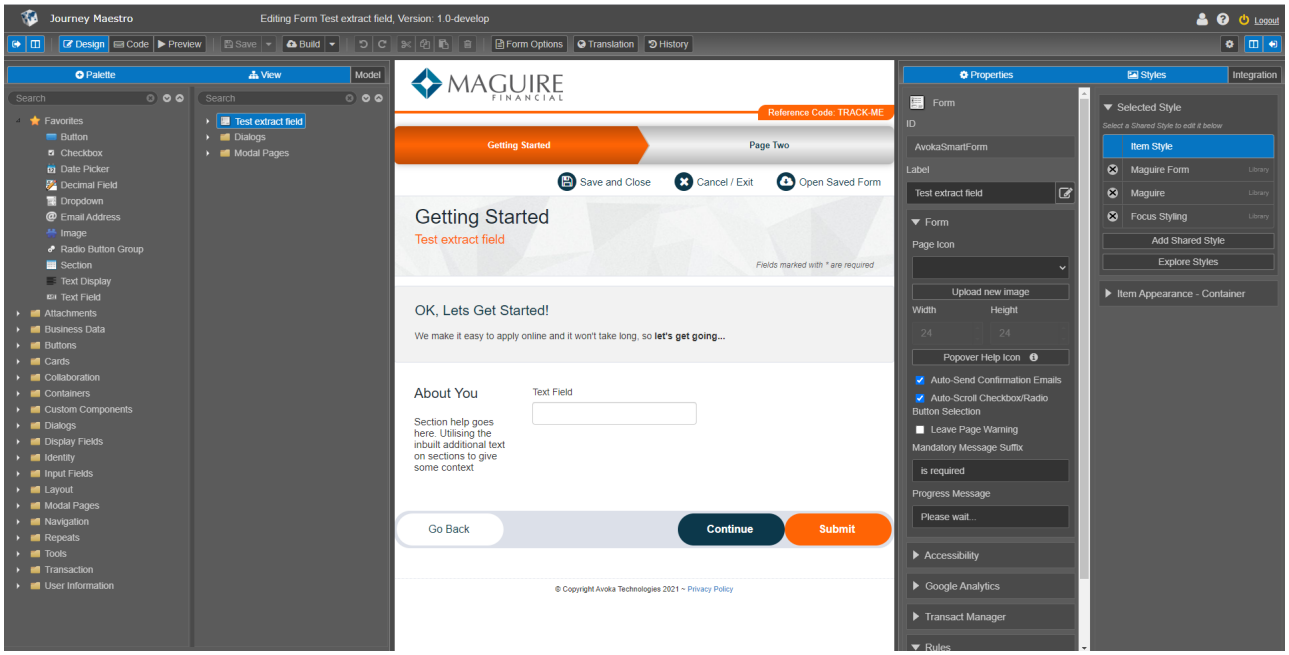
Component



For more information, see Design Mode Toolbar.

The Toolbar provides the following functionality, which you need to efficiently work with Maestro's forms, components, and templates:

-  shows or hides the left panel and  shows or hides the right panel.
-  displays two panes side by side or clear it to show only one pane. This is beneficial for larger screens as you can adjust the size of the panes according to the screen width.



If you have a larger screen, the four-tab view can be extremely powerful. On the left side, this allows you to drag and drop components from the Palette pane directly onto the form. Many seasoned Maestro developers find this a quicker and easier way to add components. You can also double-click the component from Palette for even faster additions. On the right side, this allows you to view the Properties and Styles panes together, which can also make editing a little faster.

- Design switches to design view in the editor, that is where a form is displayed when you open it from the Maestro Dashboard
- Code shows the code view in the editor
- Preview checks how a form will look when it's built
- Save updates a form's content
- Build a form version
- Publish a component or publish a template
- Undo and redo actions, as well as cut, copy and paste of a selected components
- Delete a selected component
- Form Options shows you additional form configurations

- Component Options shows you additional component configurations while working with a component
- Brand allows you to select a brand you want to modify while working with a template
- Translation applies multiple languages to a form
- History views the history of the asset and undo recent changes made to a form
- Settings Menu applies various configuration options to a form
- Close closes an open form

Left Panel

The left panel of the editor includes the following panes:

- Palette Pane displays all available components. All listed components are available to be added to a Maestro asset.
- View Pane provides a hierarchical structure of the form, which is particularly useful with larger forms, where finding components can be difficult. This pane replaces the Structure pane.
- Model Pane displays data models and is used to manage and share domain models. | 18.05 This feature was introduced in 18.05.
- History Pane shows a history of every action that has been performed in the current editing session. | 17.10 This feature was removed in 17.10 Use the History button instead.
- Editor pane shows an editable preview of component's properties on the right side of the window.

All panes in the left panel have the following controls:

Icon	Label	Description
	Search Input	
	Expand All	
	Collapse All	

Right Panel

The right panel of the editor includes the following panes:

- Properties pane displays the properties and business rules of a selected component. The properties vary depending on a type of the selected component. Some properties can contain rich text content which you can modify using the built-in rich text editor.
- Styles pane provides options to customize the look and feel of a component. It's recommended using a template to create and modify form's styles.

- Integration pane allows you to control the underlying data representation of a selected component. This pane replaces the Data pane. | 17.10 This feature was introduced in 17.10.

Views

The Maestro editor comes with the following views, also called modes, which help you create various Maestro assets:

- Design view is the default view, which is displayed when you open a new asset, and it is where you work with the open form, template, and component.
- Code view allows you to view, create and modify all business rules.
- Preview mode shows how a form will look when it is built. This is a working preview of the form that allows you to check the added components and some functionality without having to build a form.

Help

The Maestro editor provides you with the same help options as the Maestro Dashboard, plus the following addition option:

- Show Form Details to view more information about an open form, template or receipt.

Next, learn about the Properties pane.

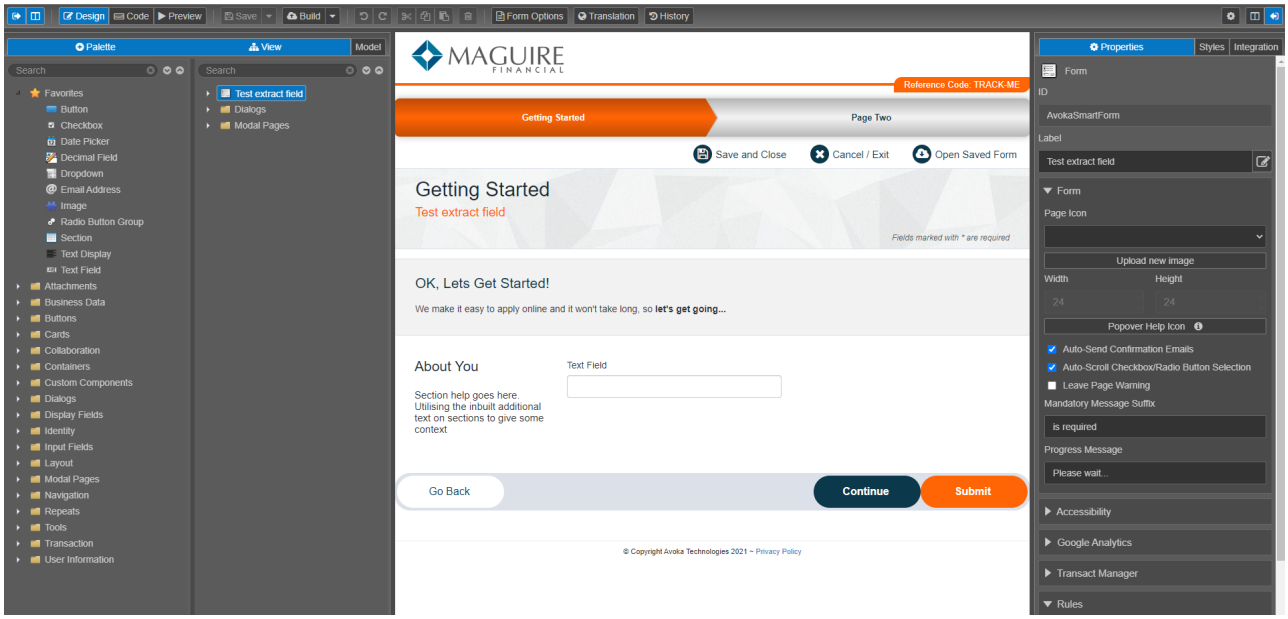
Design Mode

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

The Maestro editor comes with the Design Mode functionality, which allows you to create Maestro forms. This is the primary interface for creating and configuring designs.

To view a form in the Design Mode:

1. Open a form or a template in the Maestro editor and click Design.

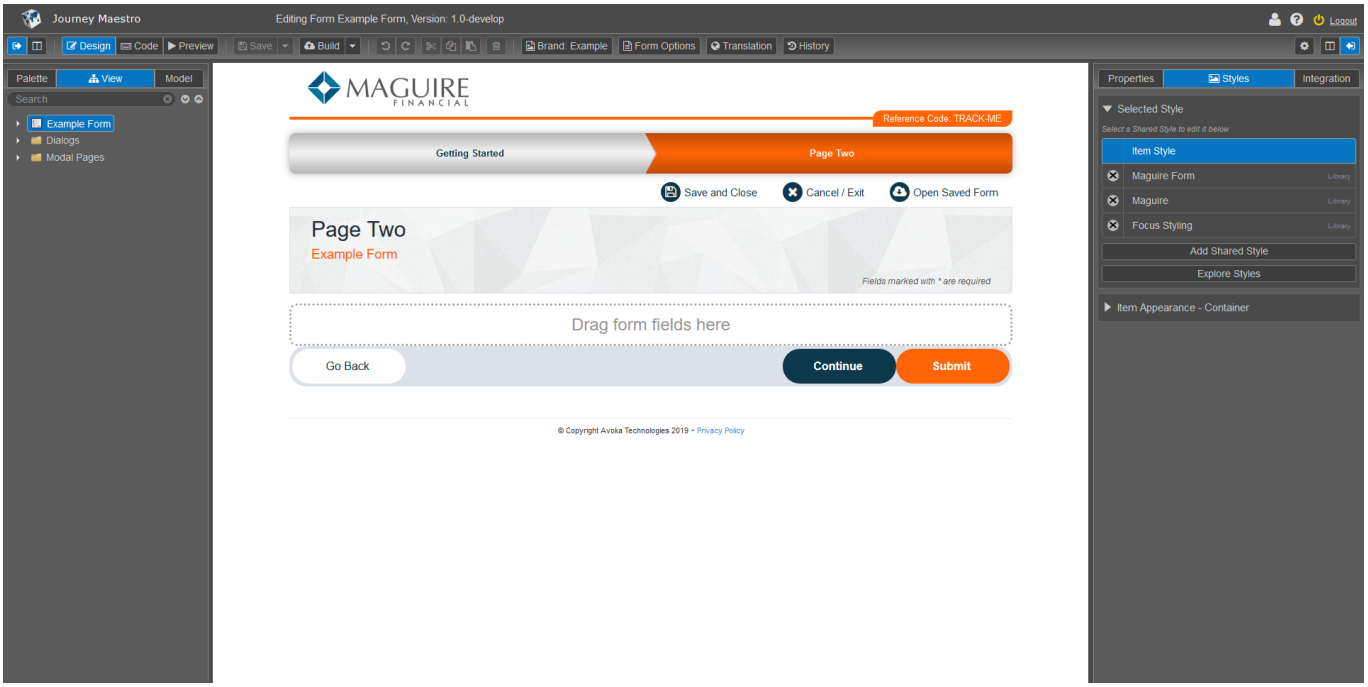


> NOTE
>
> When you open a design in Maestro, it will open in Design Mode by default.

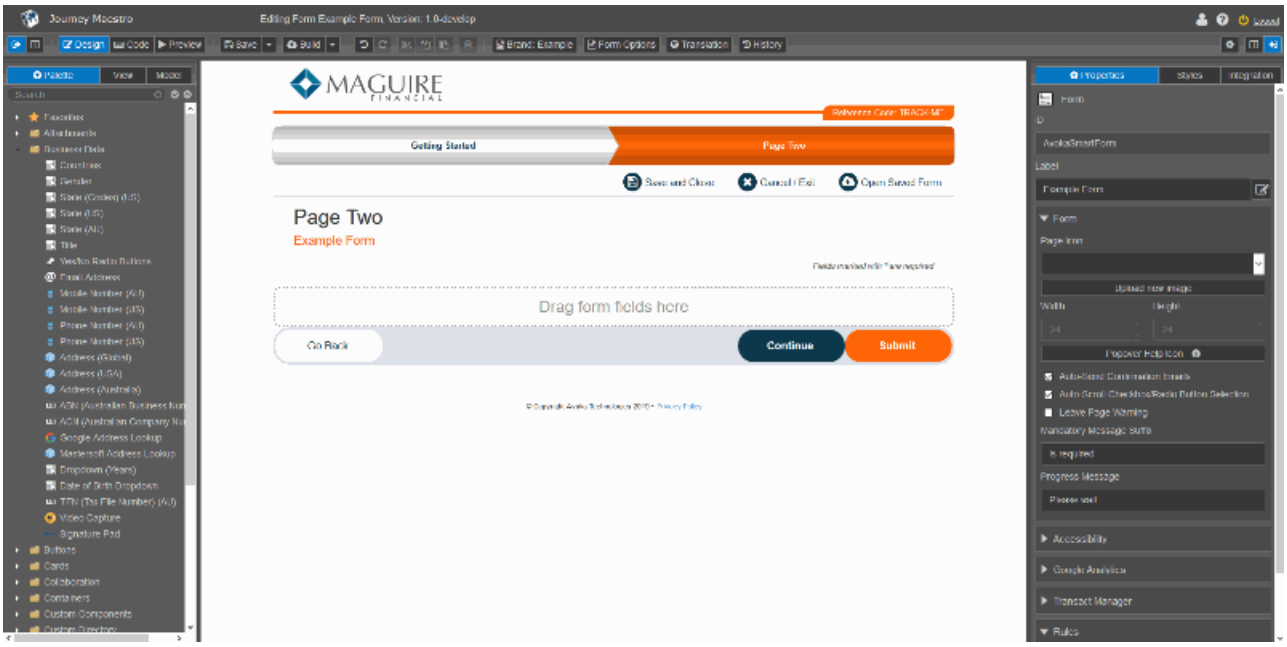
2. Drag and drop them from the Palette pane. Indicators will appear to inform you where the component will be inserted.
3. All of Maestro's designs are based on Bootstrap's twelve-column grid, which means multiple components can sit within the same row.
4. You may, however, have to adjust the component's column widths via the Properties pane, under the **Layout** heading.

> NOTE
>
> You cannot select multiple components at once within the wireframe.

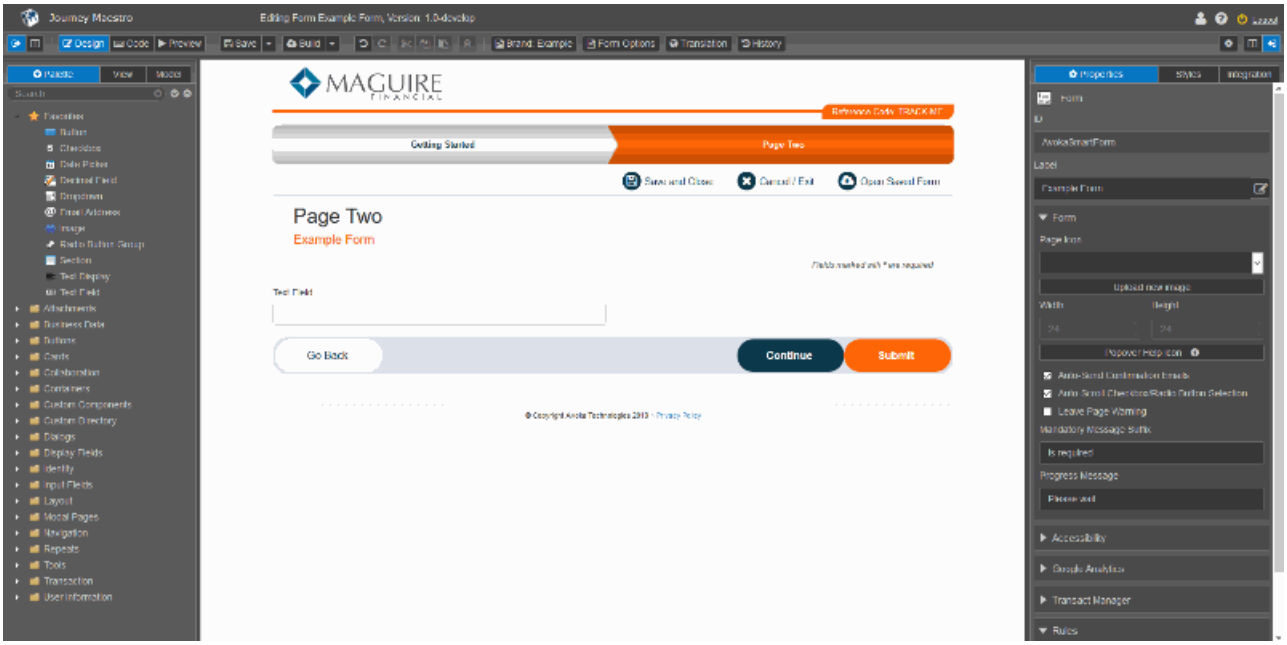
Here is a quick demonstration how you can design a form by dragging and dropping different components within the wireframe. Expand each animated image to watch it.



Drag and drop them from the Palette pane.



Add another component within the same row.



Next, learn about the rich text editor.

Rich Text Editor

MaestroThe UI design product. | Form Builder | 22.04 This feature was updated in 22.04.

The Maestro editor comes with the rich text editor allowing you to directly apply custom HTML, styles, images and Font Awesome[1], as well as use AngularJS[2] expressions in some component's, such as a label, to derive the text from other fields in the form.

To edit text in the rich text editor:

1. Open a form or a template in the Maestro editor and click the Design tab.
2. Select a component and switch to the Properties tab.
3. Click the property with a pencil icon, for example the label, to open the rich text editor.

   NOTE

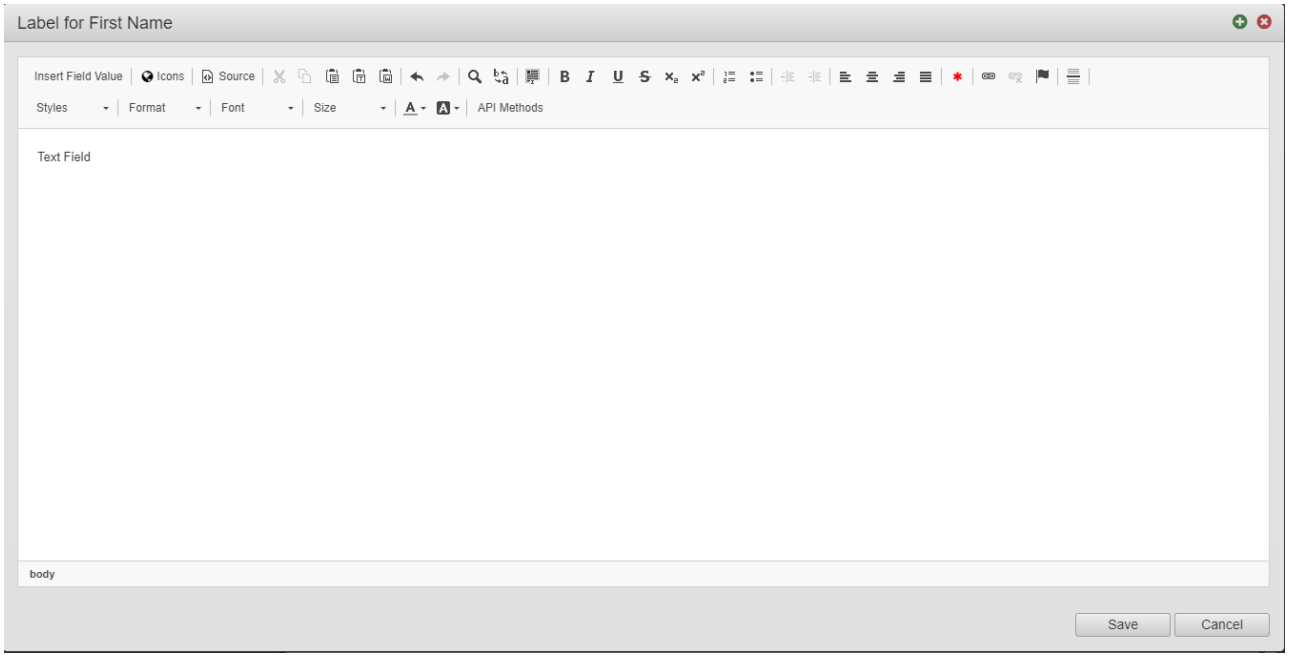
   The component's property must be enabled for the rich text content. Rich text is enabled for all labels, but you can also define component properties as rich text, for example section headings and the help text (popover) to name a few.

4. Edit the text in the editor using rich formatting as required.

[1]Font Awesome is a font and icon toolkit based on CSS and Less.

[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/



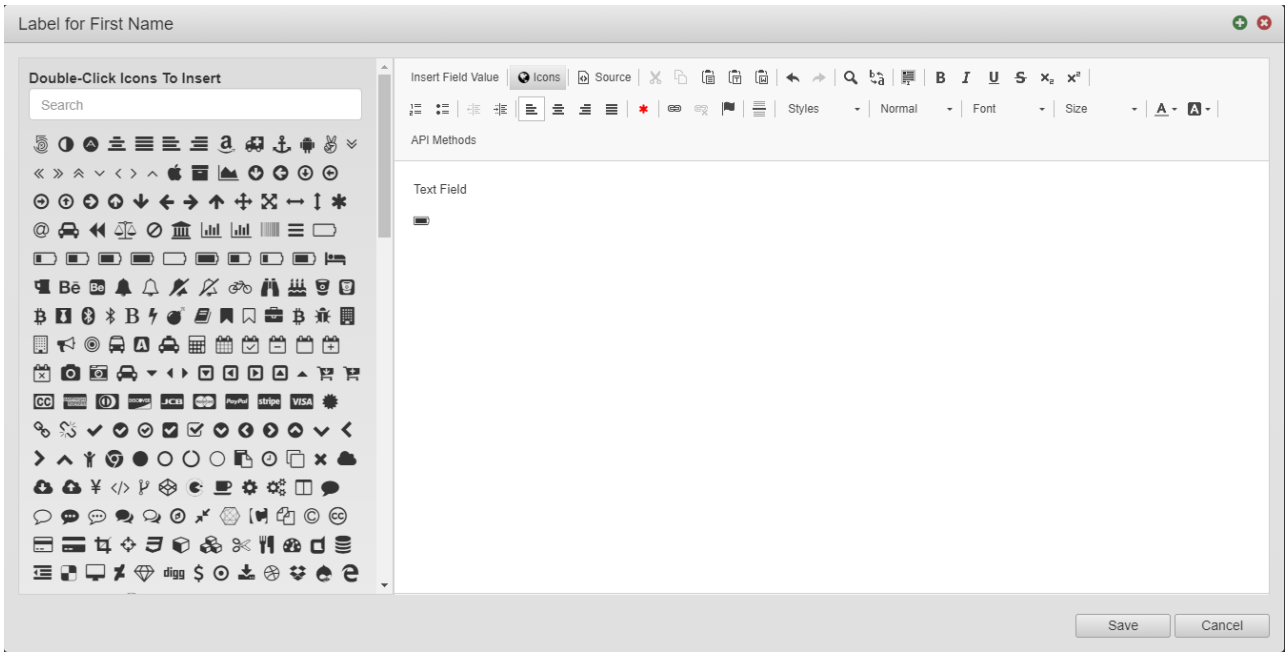
The editor provides a larger workspace for editing the properties and it has the following options:

- Insert Field Value
- Source
- Paste as Plain Text
- Paste from Word
- Find
- Replace
- Bold
- Italic
- Insert / Remove Numbered List
- Insert / Remove Bulleted List
- Decrease Indent
- Increase Indent
- Align Left
- Align Center
- Align Right
- Justify
- Mandatory Marker
- Link
- Unlink
- Format

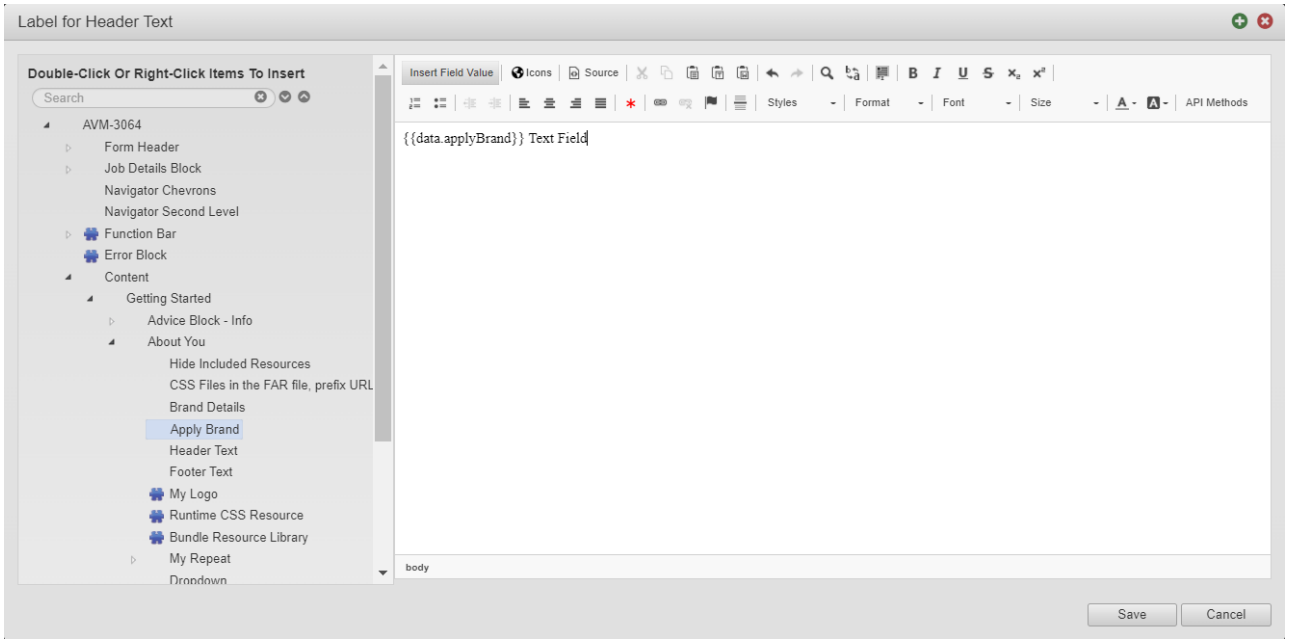
- Font
- Size
- Text Color
- Background Color
- API Methods

5. Select the Icons tab to choose an icon using the Icon Picker from the list of available icons and double-click it to insert it into the property. | 21.05 This feature was introduced in 21.05.

> NOTE
>
> You can also insert Font Awesome icons into a component using rich text properties using the HTML code, for example, `<i class="fa fa-camera-retro"></i>`



6. Select the Insert Field Value tab and double-click a property from the tree to reference its values in your property. This makes your property value change dynamically. This tree view displays the structure of the design, similar to the **View** pane. Click API Methods to open the JavaScript API Reference documentation for all available functions that you can use in your component. For more information, see dynamic labels.



7. Select the Source tab to view and edit the rich text content as a source code. You may need to use this feature to create a content including special characters, such as `&`. For example, if you want to have a label text as R & D, you should type it as `R & D` HTML source.

   NOTE

   You can't use any of rich text formatting in this mode.

8. Click Save to save your changes.

Next, learn about Properties pane.

Icon Picker

MaestroThe UI design product. | Form Builder | 22.04 This feature was updated in 22.04.

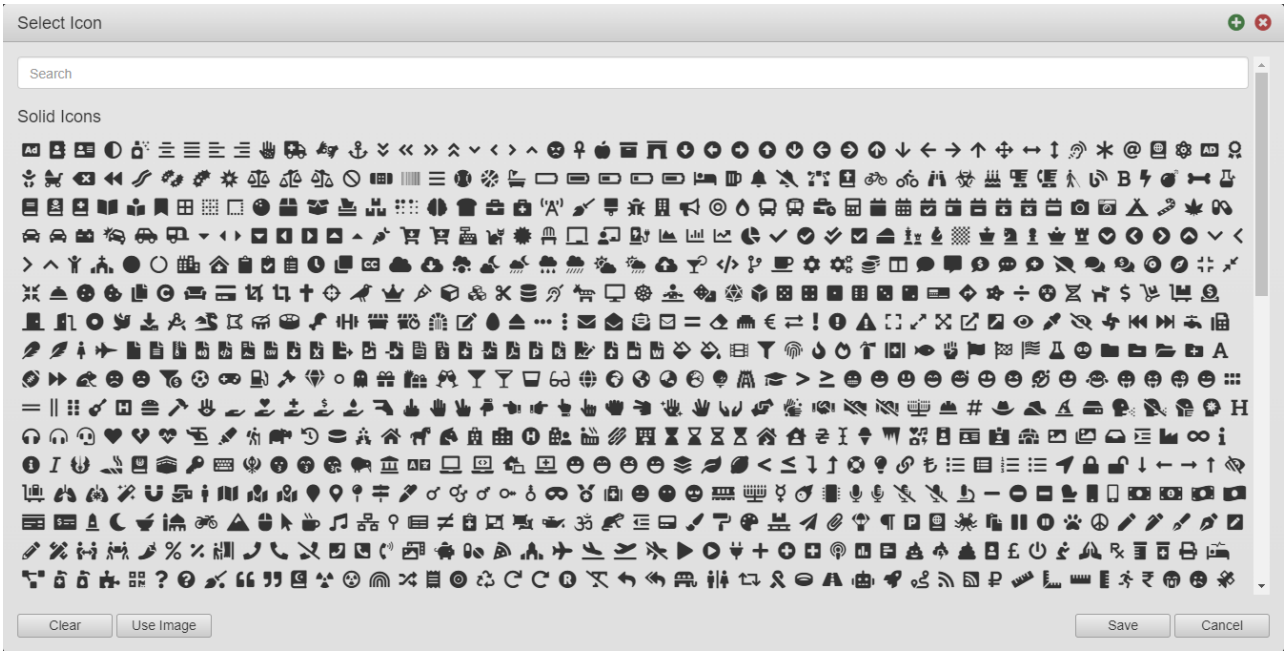
The Maestro editor comes with the Icon Picker allowing you to choose an icon from the pre-defined collection of icons, which is shipped with each release of Maestro. We use web font icons. For more information, see Font Awesome documentation.

NOTE

You can also use the rich text editor to set an icon for a component.

To select an icon using the Icon Picker:

1. Open a form or a template in the Maestro editor and click the Design tab.
2. Select a component and switch to the Properties tab.
3. Click the item's property, which allows you to specify an icon, to open the Icon Picker. For example, the Collapsed Icon property of the Section (Avalon) component opens up the Icon Picker.



The icons are grouped by the following categories:

- Solid Icons: They are faster to recognize unless their cues are subtle and not noticeable enough.
- Regular Icons:

- Brand Icons: They are specific to various company brands. For more information, see Font Awesome documentation.

4. Type a full or partial name of an icon you are looking for in the Search bar to filter the icon list.
5. Select an icon you need from the list or filtered list and click Save to inset the icon into the property.
6. Click Clear to undo your selection or click Cancel to discard your changes.
7. Click Use Image to see the instructions on how to use an image in the property. These instructions are explained below.

Use Image

In some cases, you may need to decide whether to use icons or images. There are the following differences you may want to consider while making such a decision:

- Icons are included in the core Maestro libraries by default
- Icons have simple HTML markup, so using a different font family to display the icon - the font family only needs to be loaded once and the entire icon set is available in the form
- Images need to be provided separately and loaded into the page individually

To use an image in a property:

1. Open a form or a template in the Maestro editor and click the Design tab.
2. Select a component and switch to the Styles tab.
3. Select a styling target, for example, Container.
4. Expand Items Appearance - Container
5. Expand Background and select Image from the dropdown list.
6. Choose an available image from the dropdown list or click Upload new image to choose your background image.

   NOTE

   Images you see in the dropdown list are these inside one of the libraries available to the project - either core, or any other organization or project-level libraries that have been added to your project.

7. Style the image as required.
8. Click Save to save your changes.

Next, learn about Properties pane.

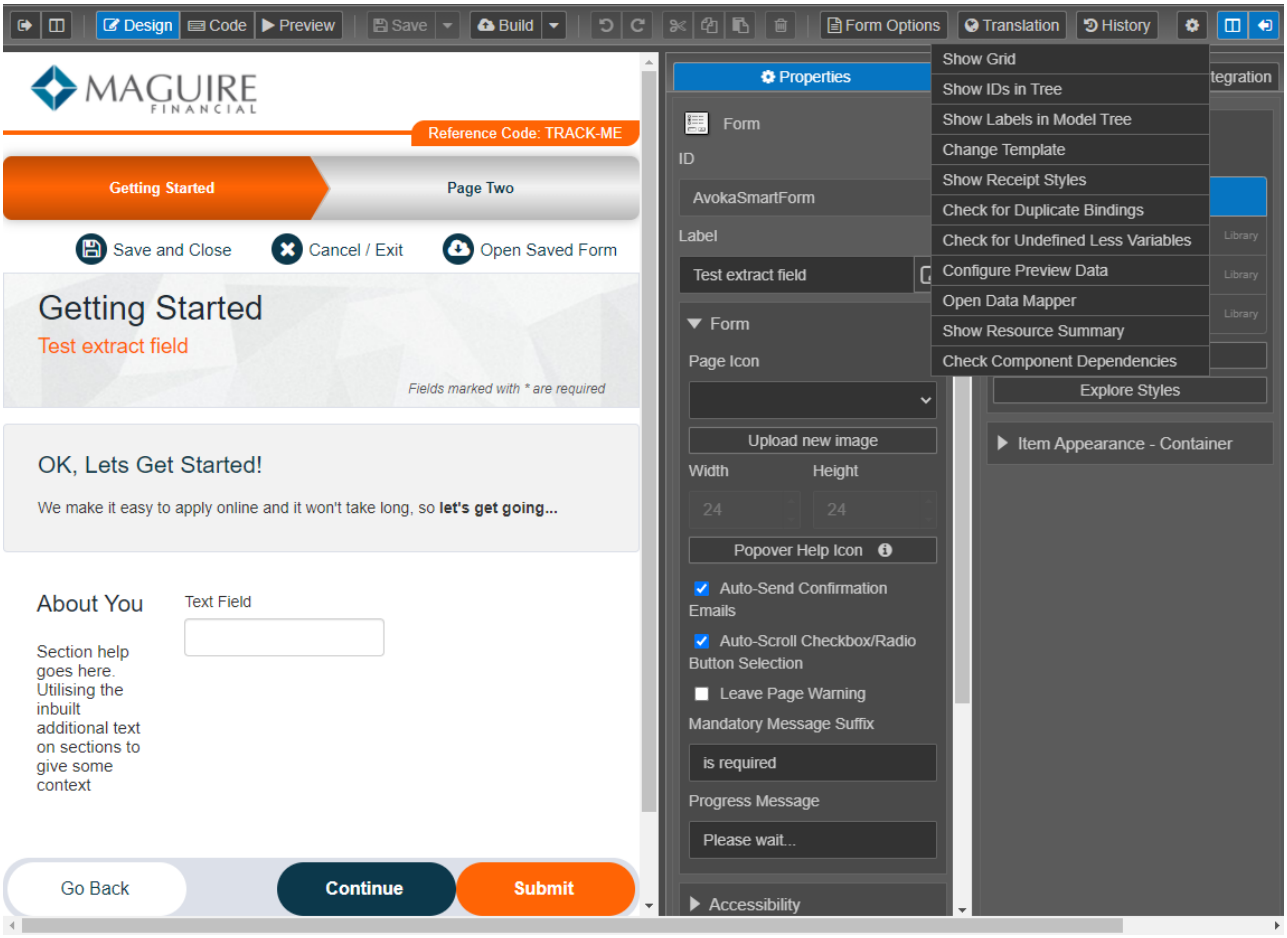
Settings Menu

MaestroThe UI design product. | Form Builder | 18.05 This feature was updated in 18.05.

The Maestro editor allows you to access and perform various actions on a form quickly using the Settings Menu functionality,

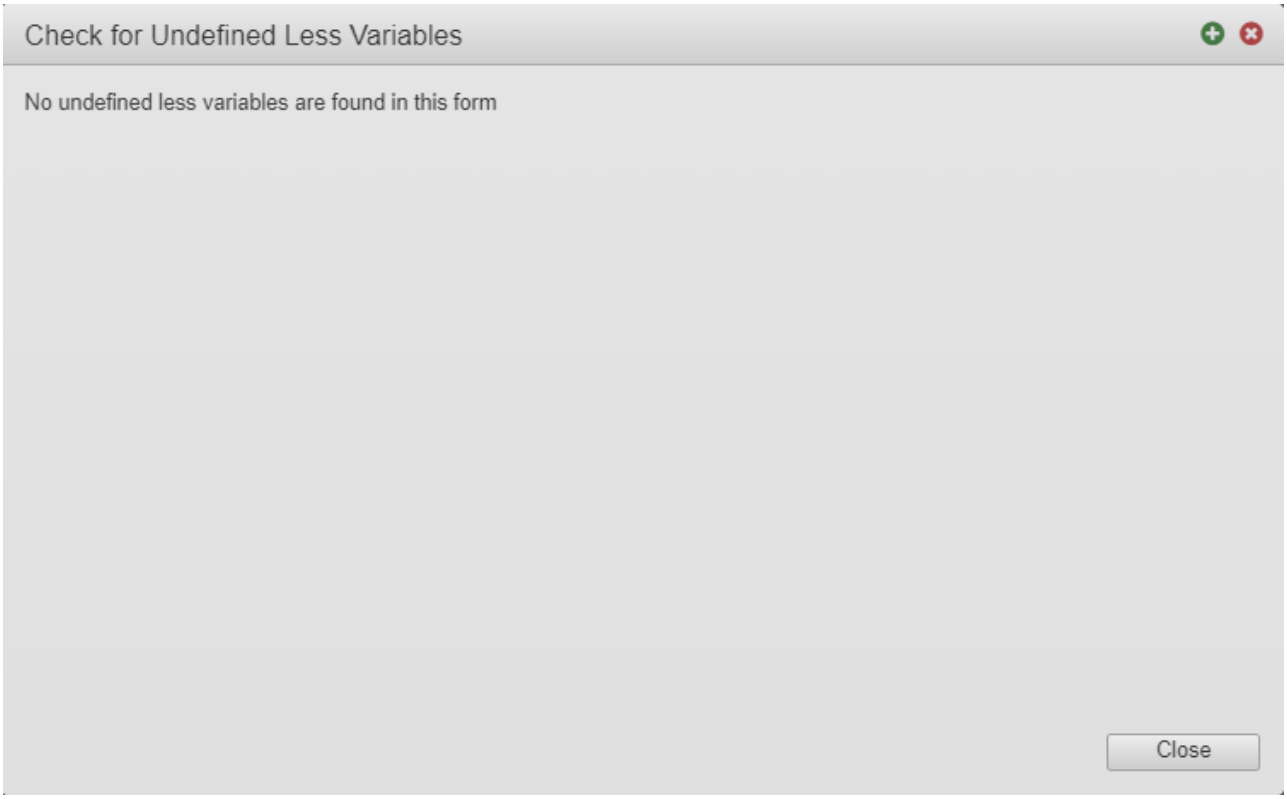
To open Settings Menu:

1. Open a form in the Maestro editor and click Settings Menu .



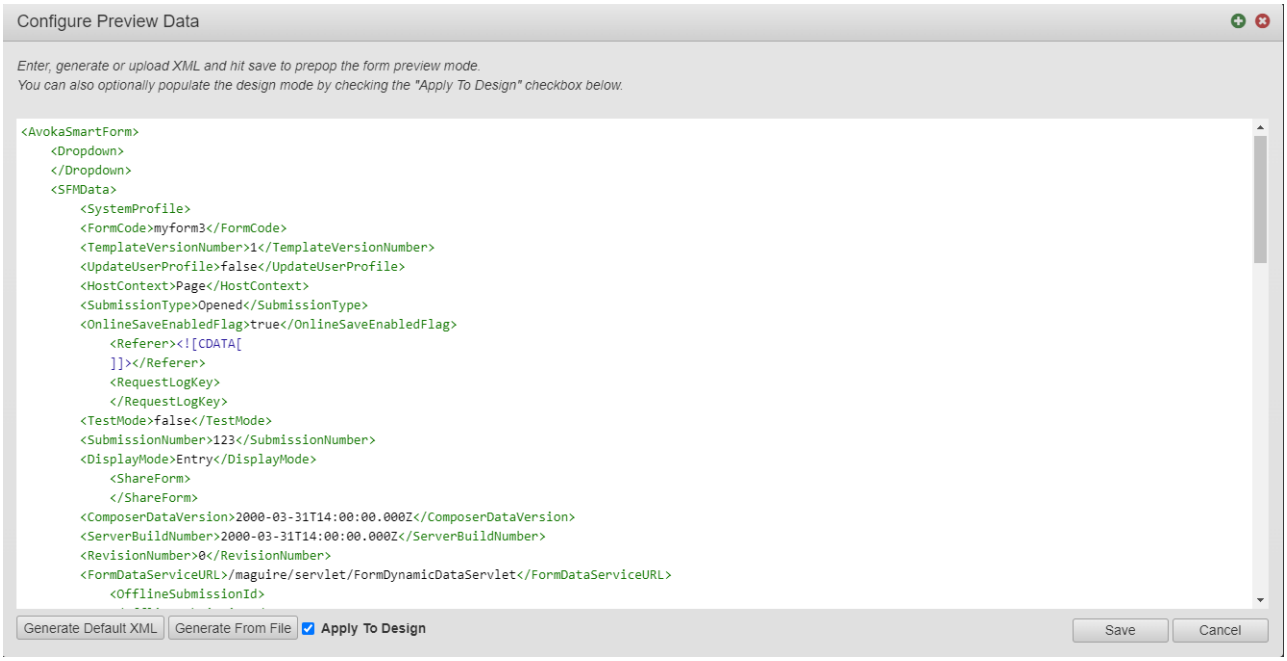
2. Select Show Grid to show a grid of horizontal and vertical dashed lines over a form to help you align components to a form layout. Select Hide Grid to hide the supplemental lines.
3. Select Show IDs in Tree to show components' IDs in the Palette pane. Select Show Labels in Tree to show components' labels.
4. Select Show Labels in Model Tree to show components' labels in the Model pane. Select Show IDs in Tree to show components' labels.
5. Select Change Template to change a template that a form uses.

6. Select Show Receipt Styles to show a form in the Maestro editor as it will be shown on a rendered receipt, for example, a PDF document. Select Hide Receipt Styles to show a form as it is normally displayed in the Maestro editor.
7. Select Check for Duplicate Bindings to check a form duplicate bindings.
8. Select Check for Undefined Less Variables to run a check for all undefined Less[1] variables in shared styles within the form or template and the referenced styles so you can resolve them.

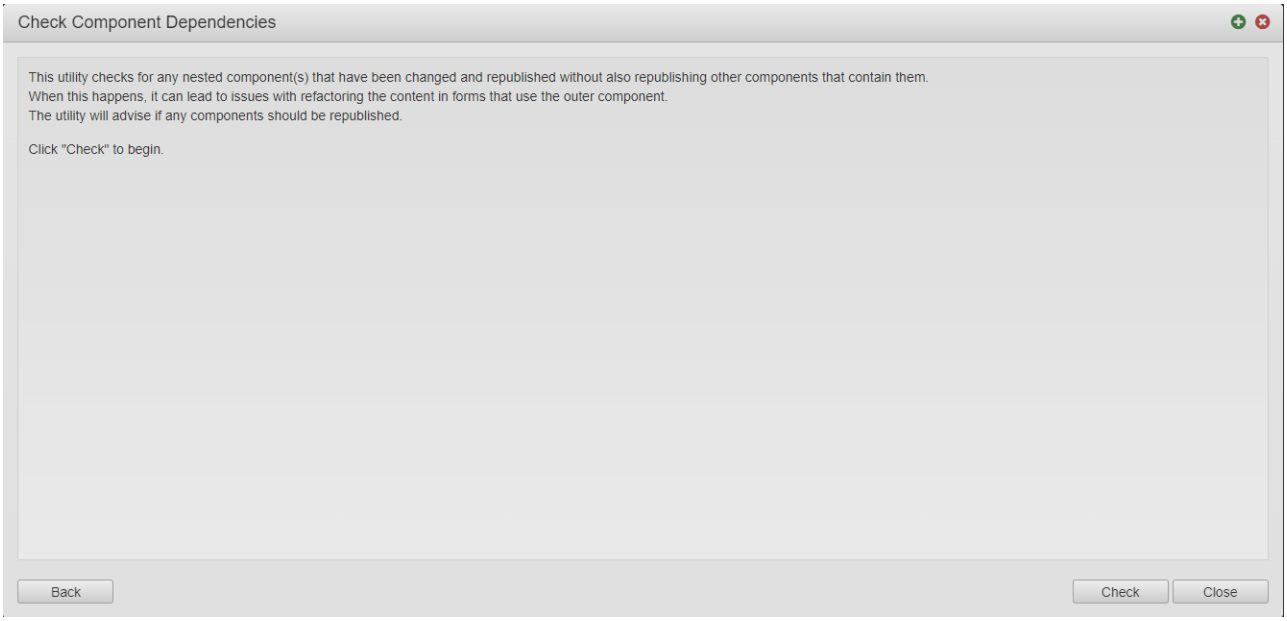


9. Select Configure Preview Data to pre-populate form data for the Preview mode. Click Generate Default XML or Generate from File to create form data. Select the Apply To Design checkbox to apply this data to form design and click Save.

[1]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.



10. Select Open Data Mapper to map form fields to submission data using the Data Mapper functionality.
11. Select Show Resource Summary to see what resources are included in this form and how much space each resource takes.
12. Select Check Component Dependencies and click Check to run a check for any nested components that have been changed and re-published without also re-publishing the components that contain them. When this happens, it can lead to issues with re-factoring the content in forms that use the other components.



Next, learn how to check "Design History" on the next page.

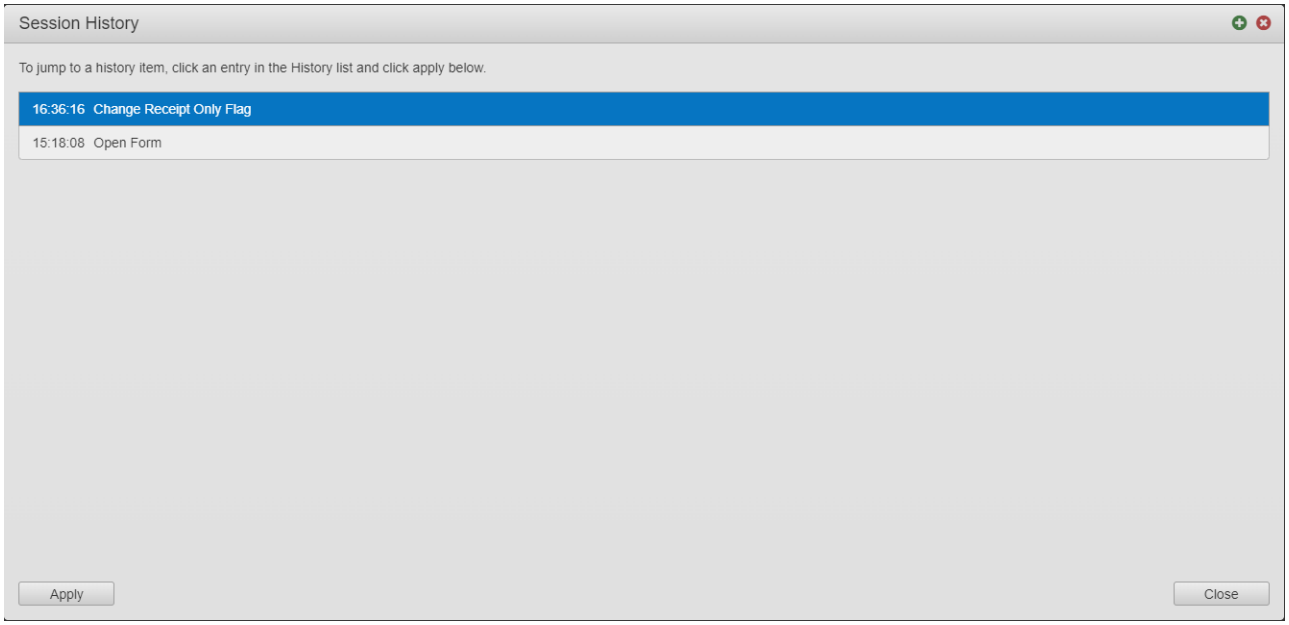
Design History

MaestroThe UI design product. | Form Builder | 17.10 This feature was updated in 17.10.

The Maestro editor allows you to check and undo changes made to a design, such as a form, a template, or a component, since it was opened in the editor.

To view, undo or revert the most recent design actions:

1. Open a form in the Maestro editor and click History.



2. Select an item from the session history list and click Apply to revert the design to an earlier state, thereby undoing one or more changes.

If you revert the design to an earlier state, the changes you've undone are not immediately lost. They will remain in the **Session History** dialog until new changes are made or until you exit the current design. Therefore, if you undo changes by mistake, you may be able to reapply those changes. If you're planning on making large changes to a design, use versions to keep those changes separate from existing work.

> NOTE
>
> Prior to version 17.10, the Maestro editor used the History pane, inside the **Right Panel**, instead of the History button.

Next, learn about the Maestro Palette Pane.

Form Resource Summary

MaestroThe UI design product. | Form Builder | 18.11 This feature was introduced in 18.11.

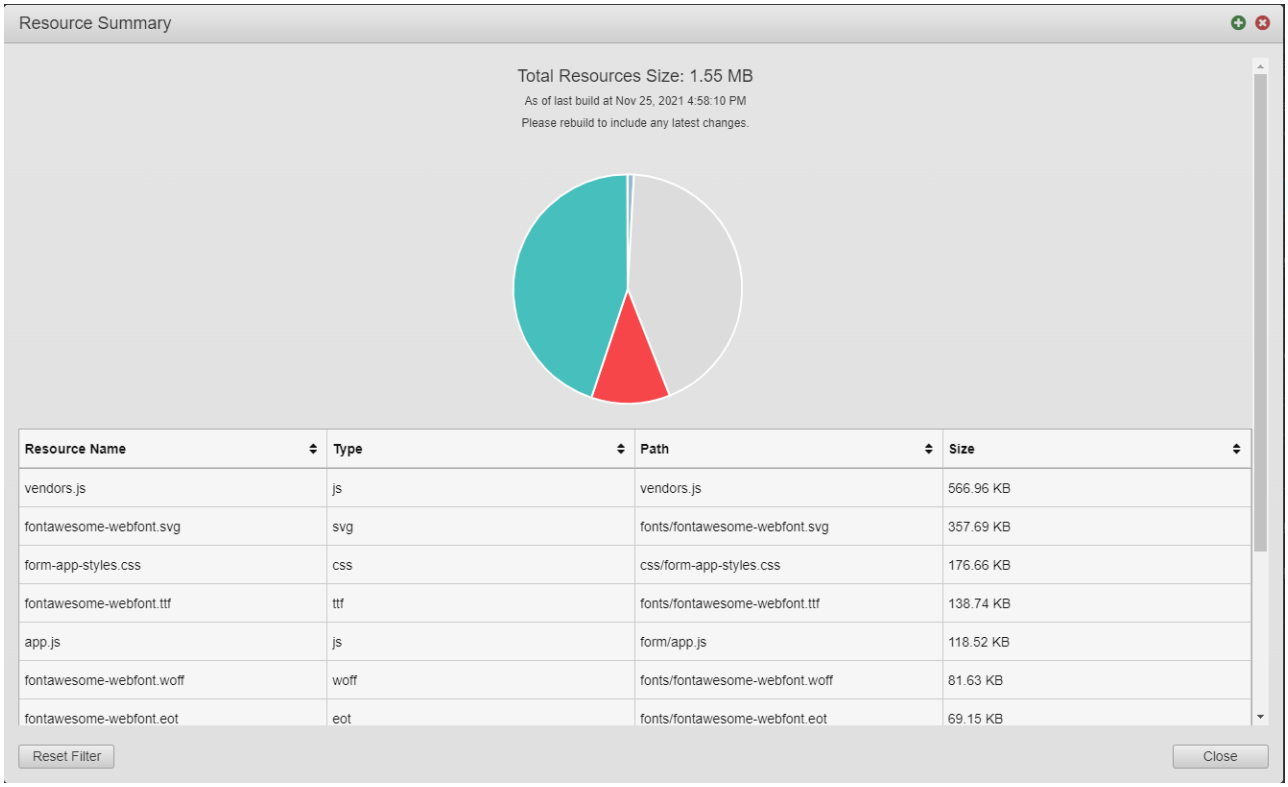
The Maestro editor comes with the Show Resource Summary functionality that generates the resource summary report for a form. This allows you to view what resources, such as HTML, image, CSS and JavaScript files, are used in the form and resources' sizes. You can use this information to troubleshoot form loading performance, or make a decision whether form resources should be externalized out of the form.

NOTE

The Resource Summary is only available once you have built a form. You can't get report summary for a template.

To show form resource summary:

1. Open a form in the Maestro editor.
2. Select Settings Menu > Show Resource Summary.



NOTE

Hover over any section of the Pie Chart to see the specific storage size of the resource.

3. Click black arrows on each column of the table to sort the summary table, for example, to sort all resources in alphabetical order.
4. Click a segment of the Pie Chart to filter the table by showing only selected resource type.
5. Click Reset Filter to clear the filter you made by clicking on the Pie Chart.
6. Click Close.

NOTE

Check other Settings Menu options.

Next, learn about the Palette pane.

Palette Pane

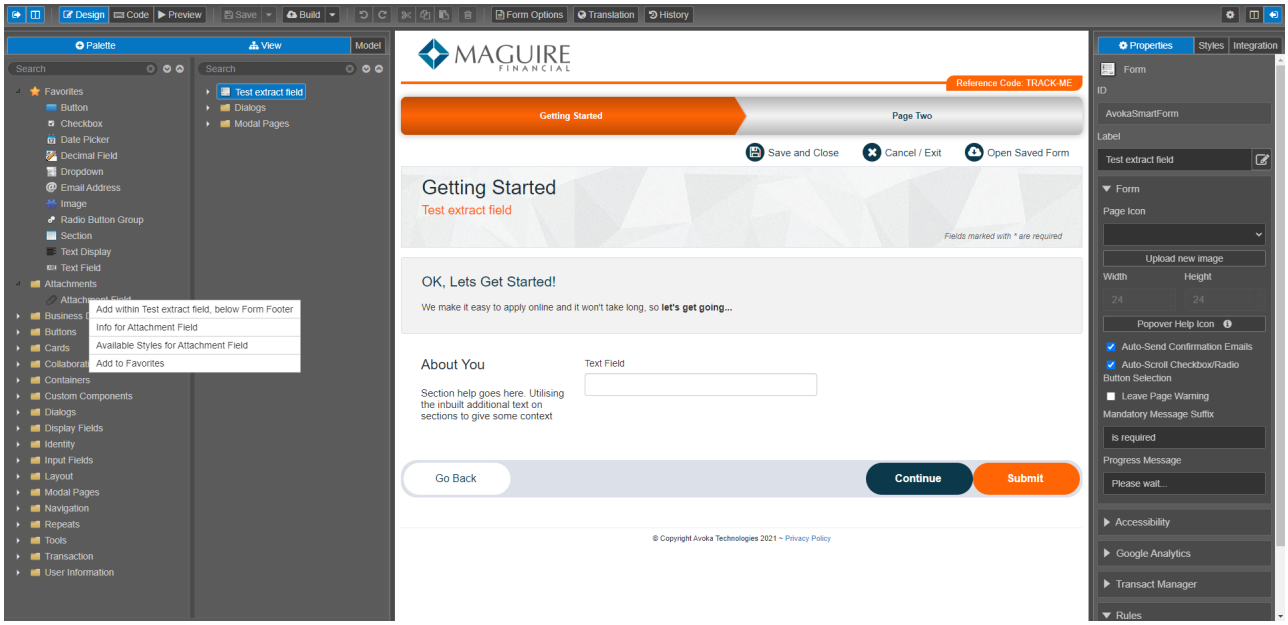
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Maestro editor comes with the Palette pane that shows all available components you can use to build Maestro forms for different customer journeys. The components are Native, Custom Native, Shared Exchange components and they are listed by categories to help you find them easily. The components available to a design depends on the libraries available to that design's project and organization. The receipt design types has access to fewer components, as some components don't make sense to include in flat, uneditable documents.

> NOTE
>
> The Palette pane gets its name from its similarity to an artist's palette that contains a range of distinct colors that are used to create masterful pieces of art.

To view a form in the Palette pane:

1. Open a form or a template in the Maestro editor and click the Palette tab.



2. Find a component you need. Components are grouped by categories according to their functionality. You may need to use the arrow to expand a category to see all available components. Alternatively, type a part of component's name in the search box. Use the down and up arrows to expand and collapse the selected categories.

NOTE

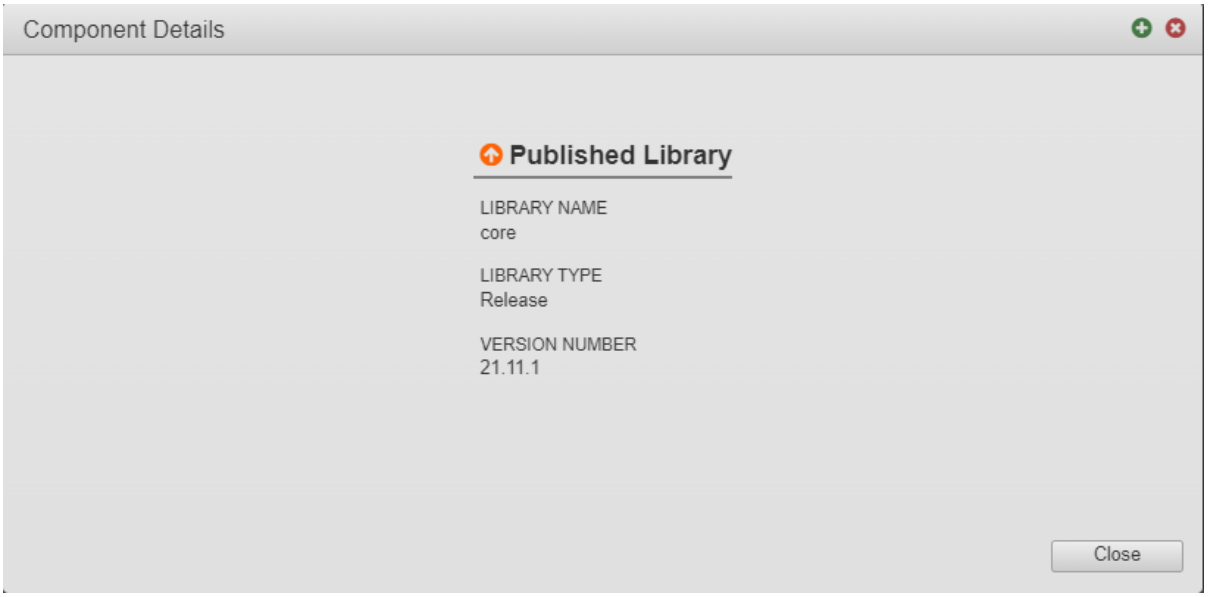
The list of displayed components may be different for different customers, or even different projects. This is because the Maestro editor might have been customized.

3. Right-click the component and select one of the following options:

NOTE

Some of these options are available when right-clicking a component in the View pane.

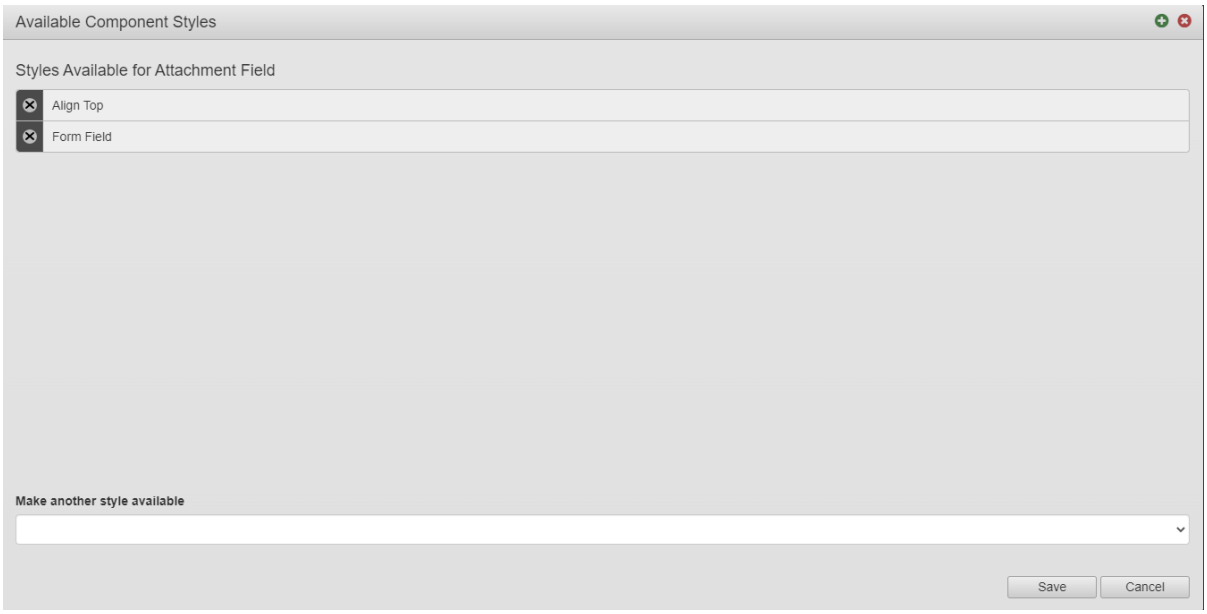
- Add Below [COMPONENT NAME] to add the selected component into the wire-frame, below whatever component is selected in the **View** pane.
- Add Within [COMPONENT NAME], Below [COMPONENT NAME] to add the selected component into the wireframe as the last child of whatever component is selected in the **View** pane.
- Info for [COMPONENT NAME] to view details, such as library name, library type and library number, about the library the component is published to.



NOTE

You can see additional details about components and their properties when you start using them in your design.

- Available Styles for [COMPONENT NAME] to view and modify the list of shared styles that the selected component can access to.



You can also do one of the following:

  - Click the X against a style to remove it from the list.
  - Select a new style from the Make another style available dropdown list to add a style and click Save.
- Add to Favorites to add the regularly used component to the **Favorites** category for easy access.
- Remove from Favorites to remove the selected component from the **Favorites** category.
- Exclude from Palette to prevent the component from appearing in the **Palette** pane. This option is only available while editing a template. A Template Designer can control which components are available to a Form Builder by excluding them from or including them in the Palette.
- Include in Palette to add the component back to the Palette pane, if it has been previously excluded, see the option above.

4. Click Save to update the component's changes in the form or the template.

Each component has a unique icon that helps you easily identify it. Depending on come component configuration, layout orientation and level of editability to you, an icon can change its background or have additional elements added to it. The list below summarizes these changes:

-  (a dot before and after an icon): a component is laid side-by-side with the previous component

Next, learn about the view pane.

View Pane

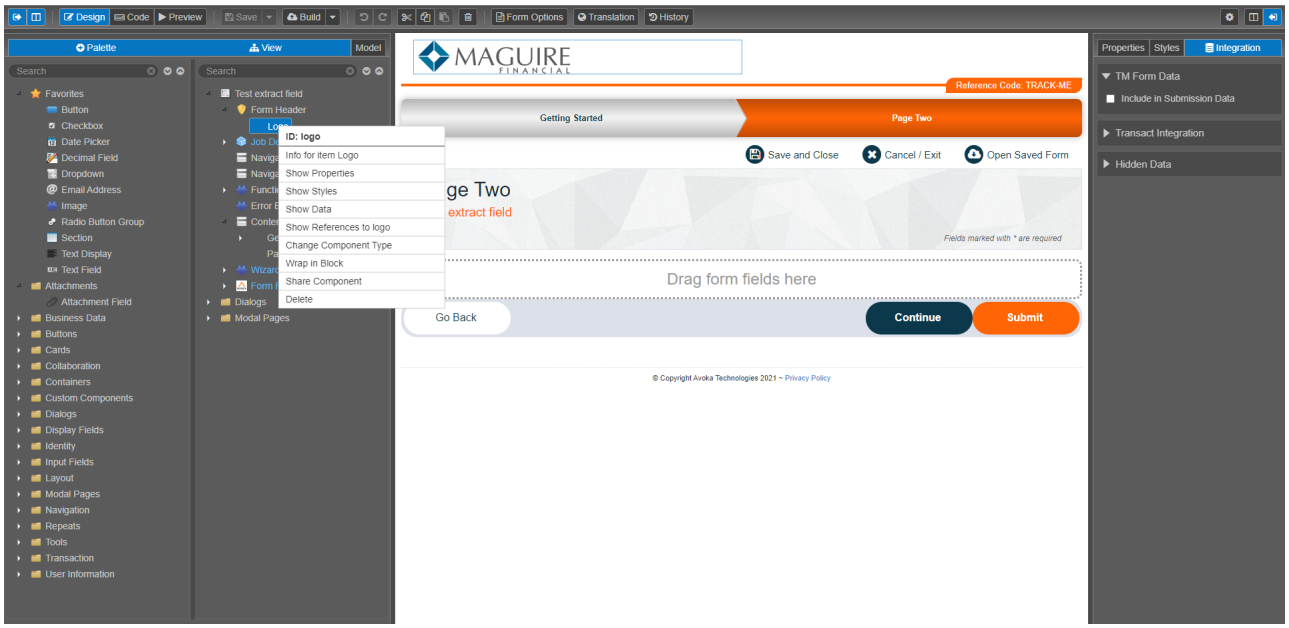
MaestroThe UI design product. | Form Builder | 18.05 This feature was introduced in 18.05.

The Maestro editor provides the View pane to visualize a selected component in a hierarchical list of the components used in the current design. This list can be a more efficient way of navigating the structure of a design than working within the Wireframe. The hierarchy you see is determined by the form's template, so even if a Form Builder does not have permission to make changes to some components, the View pane always displays the dialogs and modal pages. This is handy in navigating through large forms.

> NOTE
>
> The Structure pane was replaced with the View pane. | 18.05 This feature was introduced in 18.05.

To view a form in the View pane:

1. Open a form or a template in the Maestro editor and click the View tab.



2. Expand a component's hierarchy to see its child components. Select each component to see where it is positioned in your design.
3. Move components from one location to another by dragging and dropping them within the hierarchy.

NOTE

Some component types must be contained within other component types. For example, a Page can't exist underneath a Dropdown. The Editor will prevent you from moving or creating components under the incorrect parent.

4. Select multiple components by using the keyboard keys SHIFT or CTRL.
5. Right-click a component and select one of the following options:
   - Info for Item [COMPONENT NAME] to view details about a library the component is published to and the component's properties.
   - Show Properties to open the Properties pane for the selected component.
   - Show Styles to open the Styles pane for the selected component.
   - Show Data to open the Integration pane for the selected component.
   - Show References to [COMPONENT ID] to view references to the selected component, such as references inside rules or labels.
   - Change Component Type to change the selected component to a different component type.
   - Wrap in Block to create a block component and insert the selected component into the block.
   - Set as extension point to add an extension point to the selected component.

     NOTE

     This option is shown for templates only.

   - Share Component to publish the selected component
   - Apply Shared Styles to every [COMPONENT NAME] to apply a customized template's component style to all forms using this template.

     NOTE

     This option is shown for templates only.

   - Apply Properties to every [COMPONENT NAME] to apply a customized template's component property to all forms using this template.

> NOTE
>
> This option is shown for templates only.

> NOTE
>
> This option only doesn't apply to id, label, rules or layout.

- Delete to remove the selected component.
- Put on Previous Row to move the selected component to the end of the previous row.
- Remove Component Property Overrides to remove the property override.

> NOTE
>
> This option doesn't apply to rules, styles and the property's exclude flag.

> NOTE
>
> You can remove component property overrides using the Component and Property Source dialog.

When editing a template, additional options are available in this menu:

- Set as extension point adds an extension point to the selected component.

Some options are only available if you right-click specific components:

- Add Page Group
- Add Page
- Add Icon Button

If you right-click the **Dialogs** directory, a context menu with an **Add Dialog** option appears.

If you right-click the **Modal Pages** directory, a context menu with an **Add Modal Page** option appears.

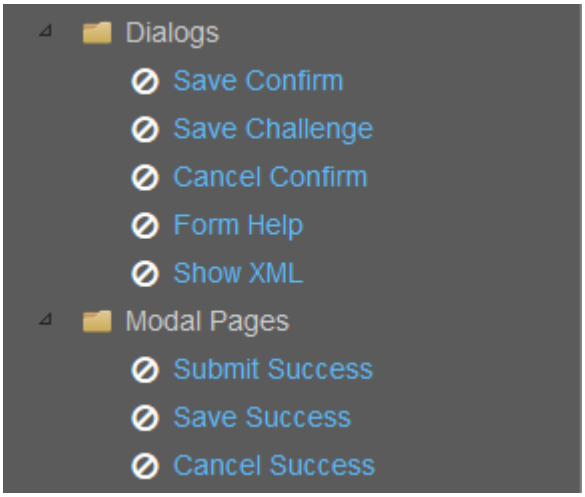
6. Click Save to update the component's changes in the form or the template.

Main View pane functionality is described in details below.

Structure

At a minimum, all designs have a root node, which is the parent to most other components in a design. In many designs, this root node is a Form component, but if you're working on a form, template, or receipt, the root node depends on the extension points configured in the parent template.

Most design types also have a **Dialogs** and a **Modal Pages** directory. These directories contain Dialog and Modal Page components that sit outside the normal flow of the form and are therefore located outside of the root node.

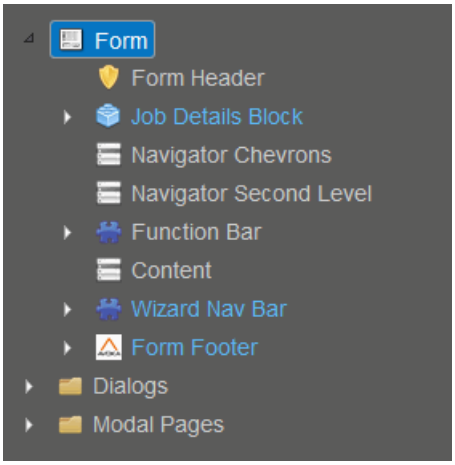


This table explains which design types have a **Dialogs** and/or a **Modal Pages** directory:

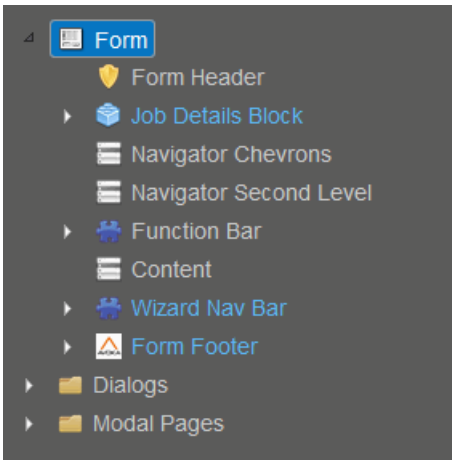
Design Type	Dialogs	Modal Pages
Forms		
Components		
Templates		
Receipts		

Sorting

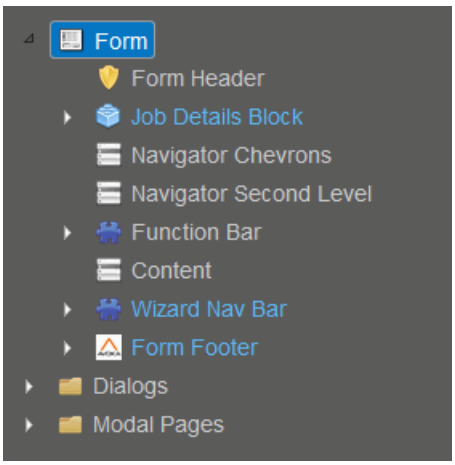
To arrange components within the hierarchy, drag and drop them.



To select multiple, non-sequential components, hold the CTRL key.



To select multiple, sequential components, hold the SHIFT key.



NOTE

Some component types can only be exist within certain component types. For example, the Page component cannot exist within a Dropdown component.

Icons

All components have an icon associated with them. These icons are not necessarily unique, but they do make it easier to grasp the structure of a design.

Additional icons are used within the **View** pane to communicate certain behavior:

Icon	Label	Description
	Unlocked	An extension point has been set on this component. This icon will only appear while editing a template.
	Line-Through	This component exists on the same row as the component above it.
	Locked	An extension point is preventing you from configuring this component.

Next, learn about the Model pane.

View Component and Property Source

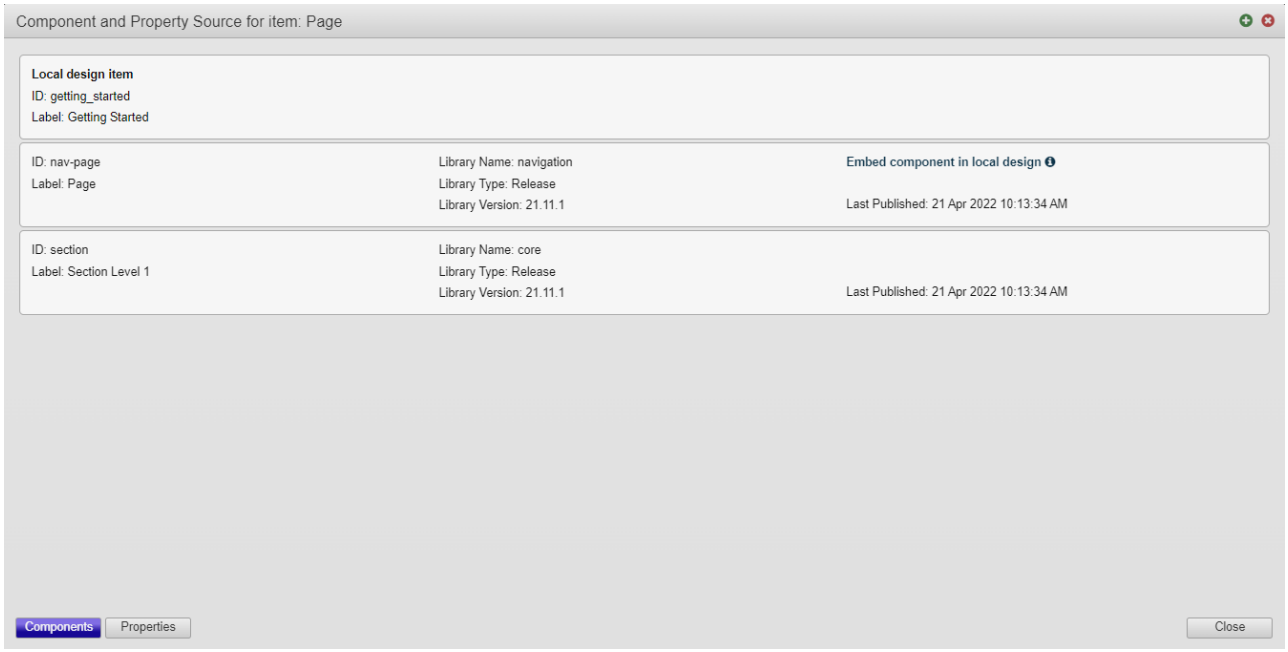
MaestroThe UI design product. | Form Builder | 19.11 This feature was updated in 19.11.

The Maestro editor allows you to view details about a library where a component is published to and the component's properties. This is handy to understand the component inheritance and properties' source, especially when you create or use shared components.

To view details about component and its properties' source:

1. Open a form or a template in the Maestro editor and click the View tab.

1. Right-click a component in the Wireframe or in the View pane and select Info for Item [COMPONENT NAME].
2. Select Components to see a library or a list of libraries where the selected component is published to.



In this example, we have a local item called getting_started that is a Page component, which is inherited from the nav-page item that is the Page component published in the navigation library. This component in turn is inherited from the section item that is the Section component.

Click Embed components in local design to add a component to your design. This will keep all the properties, rules and styles from the component the same as if you had added and set these properties in the form. This can be used on components shared from a form

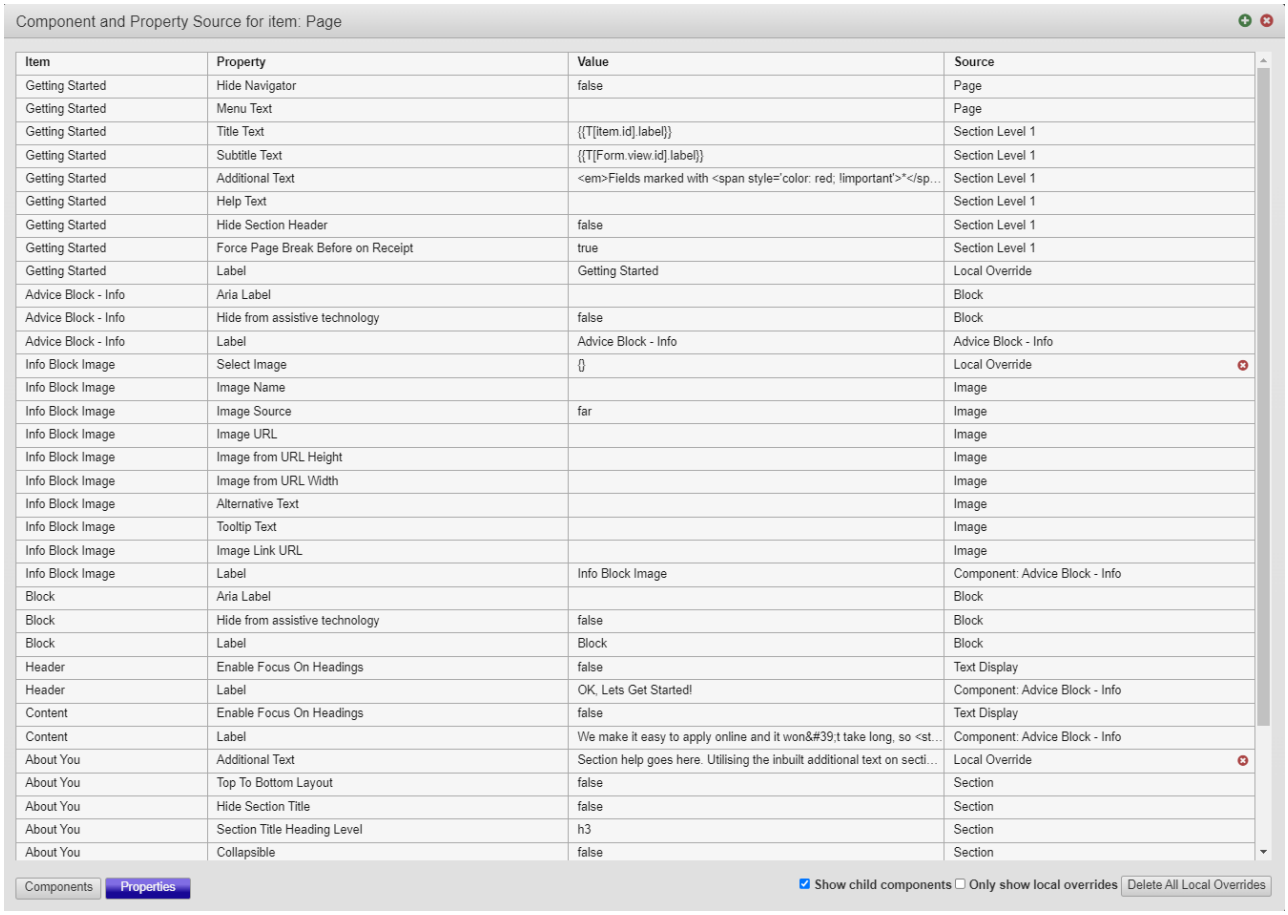
or built in the Maestro editor. If the option is not present, then this is a native widget which cannot be embedded as they would not be able to function correctly. One use for this is to add a block to your form but need to re-arrange the order of the included fields or add new ones then embedding removes this restriction and you can move the fields around.

3. Select Properties to view the list of properties used in the selected component, their values and the source.



In this example, you can see that the Getting Started components has 9 properties, with 2 properties coming from the Page components, 6 properties are from the Section Level 1 component and the Label property is a local override.

4. Select the Show child components checkbox to list the child components.



NOTE

Use View pane to see the child components in the hierarchy.

5. Select the Only show local override checkbox to list only local override. Local overrides are changes to properties and rules in this design from the original widgets. This can be used in components and templates so local overrides will take precedence over changes to the original component.



6. Click Delete All Local Override to remove all local override or click the red cross icon against each property to remove the local override. When you do this, you will restore the properties and rules to the same values as if you dropped in the component from the palette.
7. Click Close.

Next, learn about the Model pane.

Model Pane

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

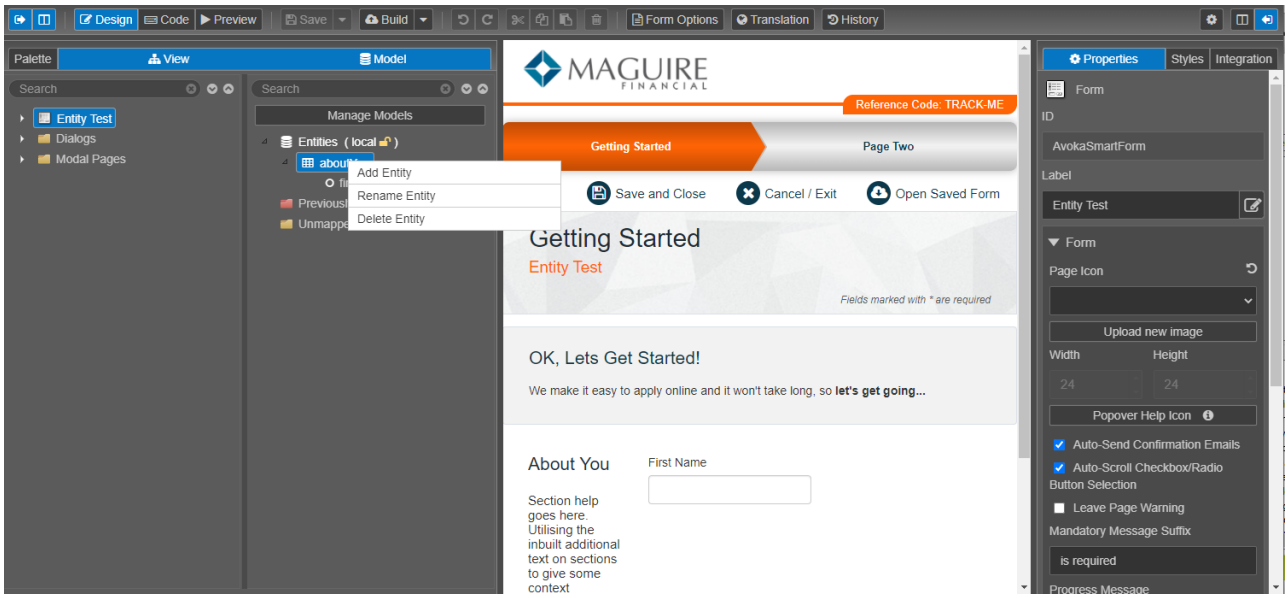
The Maestro editor comes with the Model pane that shows the hierarchical logical structure of a form, consisting of form's components, as their corresponding entities that make up a domain model. The domain model allows you to manage entities and their properties easily.

To view a form in the Model pane:

1. Open a form in the Maestro editor and select the Model tab to see the data elements and their properties that are bound to components, identified by component IDs.

   NOTE

   The Model tab was called the Entity tab in Maestro prior to 18.05.

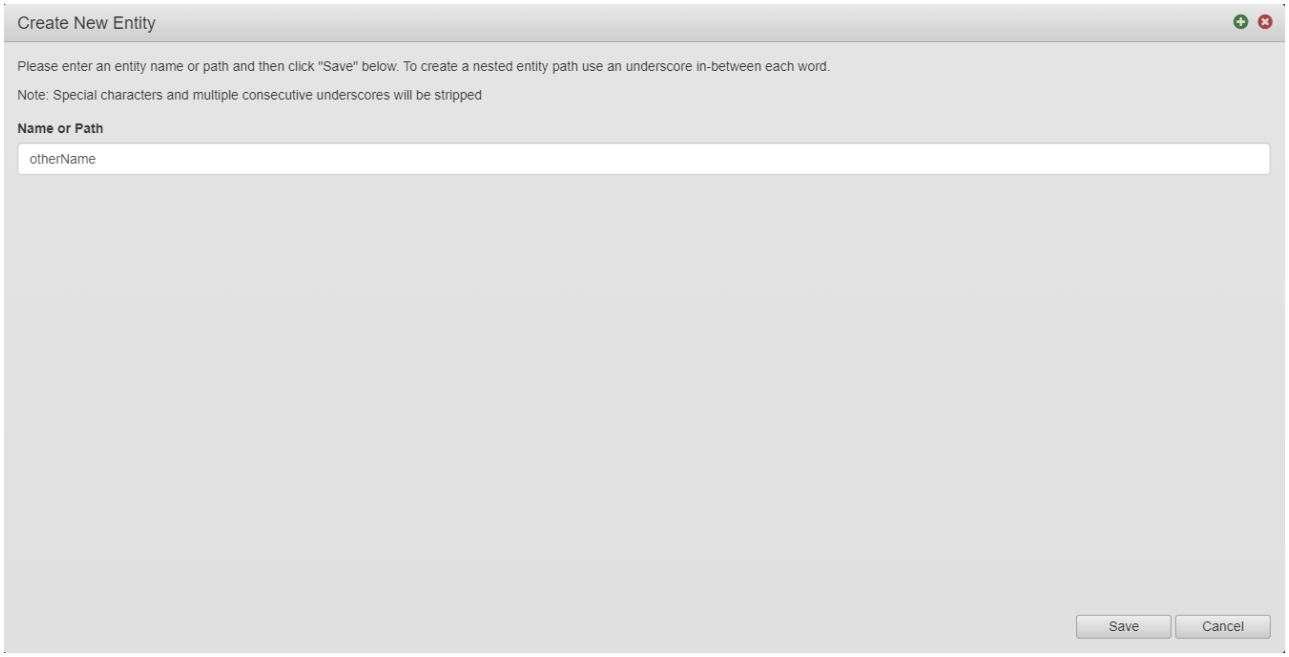


   The Mode pane is broken down into the following sections:

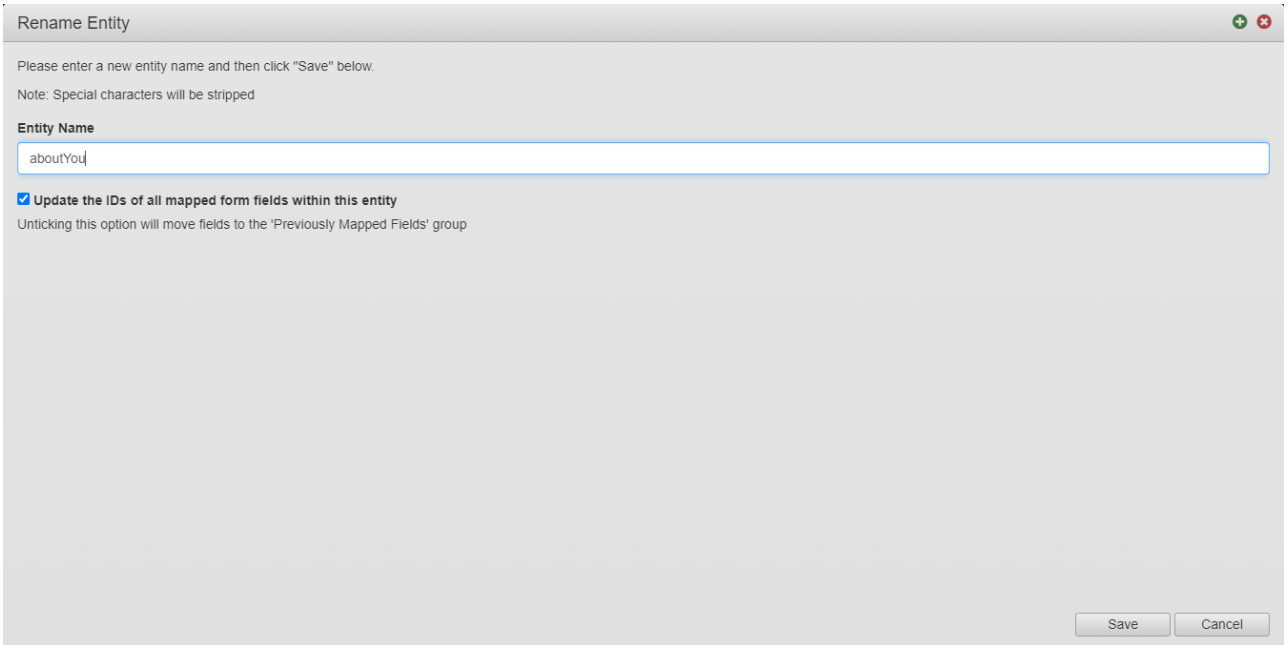
   - The Entities item displays the entities and properties that are defined in the selected form. The name local means that the schema is defined in this form and the open lock implies you can edit the data model. When a domain model is shared, the lock icon changes from open to locked. Any new entities created in a shared (locked) domain model will not be mapped or shared with other forms and libraries.
   - The Previously Mapped Fields item displays components that have previously been mapped to entities but are not mapped in the current domain model.

- The Unmapped Fields item displays components that are not associated with an entity.

2. Type a entity name in the Search field to look for the entity, which is handy when you have many components in the tree.
3. Expand the Entities section to see any entities configured for this form. If there are no entities in the list, right-click the Entities section and click Add Entity to create one.

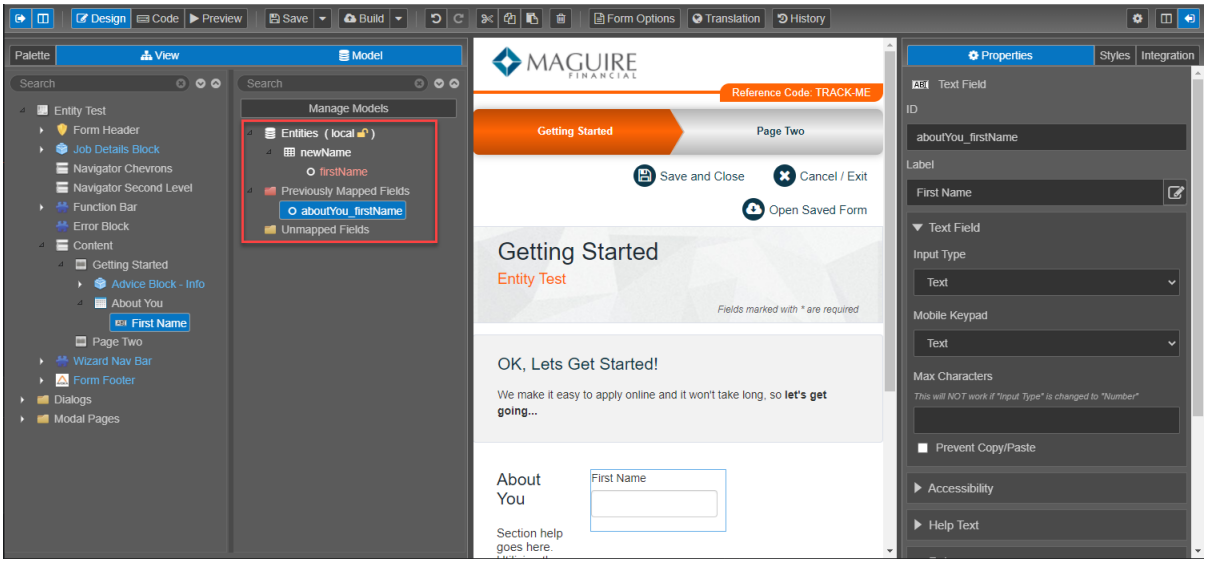


4. Right-click a selected entity and choose Add Entity to create a new entity.
5. If an entity has a property, right-click the property and select Rename Property to rename it or Delete Property to remove it.
6. Right-click a selected entity and choose Rename Entity to rename an entity. | 21.05 This feature was introduced in 21.05.



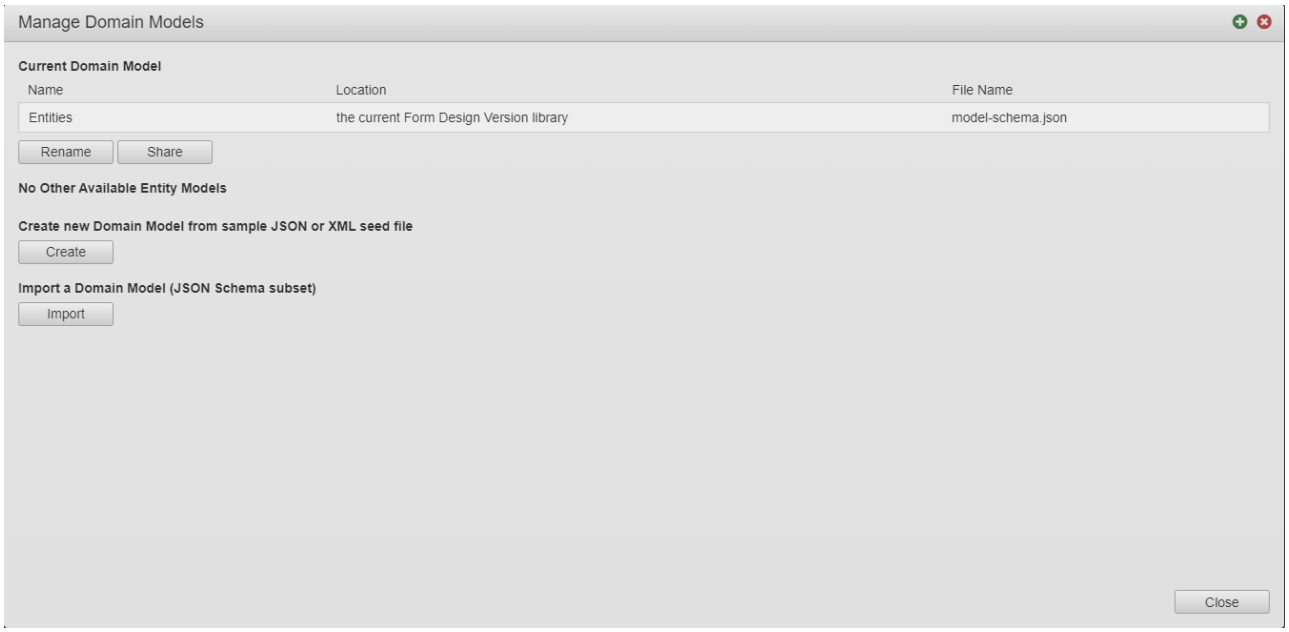
- Provide a new entity name in the Entity Name field.
- Select the Update the IDs of all mapped fields within this entity checkbox to automatically change the ID of the mapped form field (if there is one). If you select this option, the First Name field, as shown above, automatically has it's ID changed according to the new entity name.

  If you clear this option, then the entity name is changed, but the ID of the First Name field is not changed and the field appears in the Previously Mapped Fields section in the entity tree, as shown below.

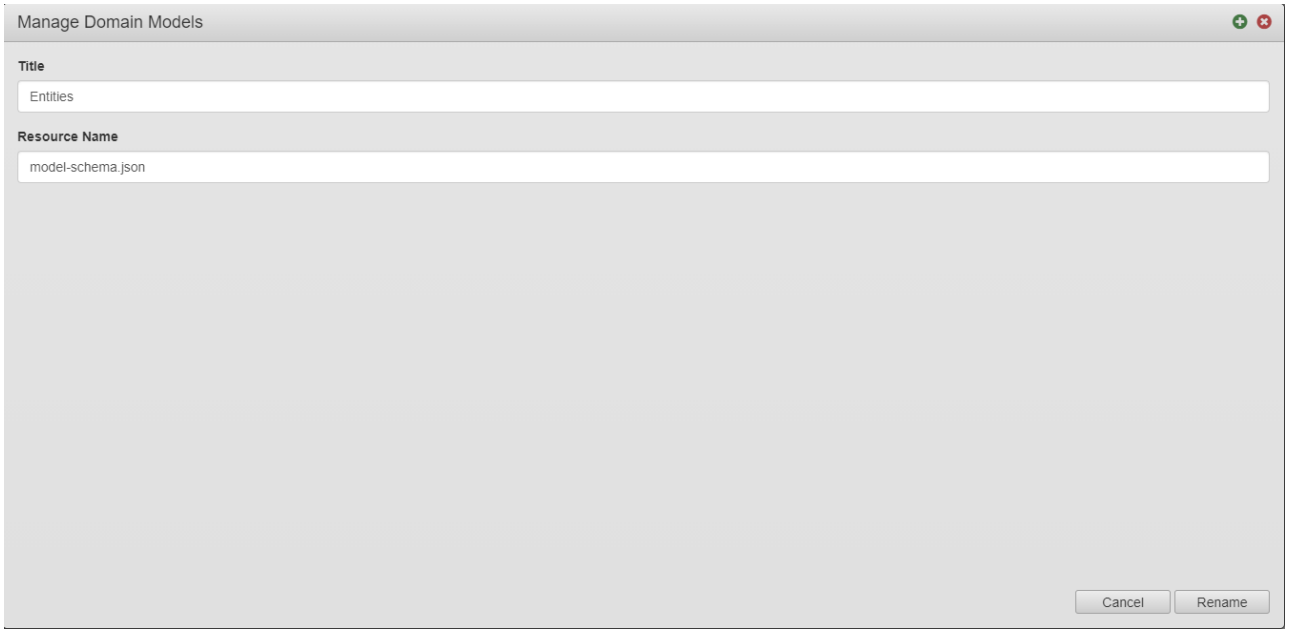


- Click Save.

7. Right-click a selected entity and choose Delete Entity to remove an entity.
8. Click Manage Models to modify the form's model.



9. Click Rename to change the name of the model.



   - Enter a new name in the Title field.
   - Provide a new unique JSON file name in the Resource Name field. Default is `model-schema.json`.
   - Click Rename to update the model.
10. Click Share to share this form's model.



11. Click Create to create a new domain model for your form.
12. Create Import to import an existing domain model as a JSON or XML file.

Next, learn about the Properties pane.

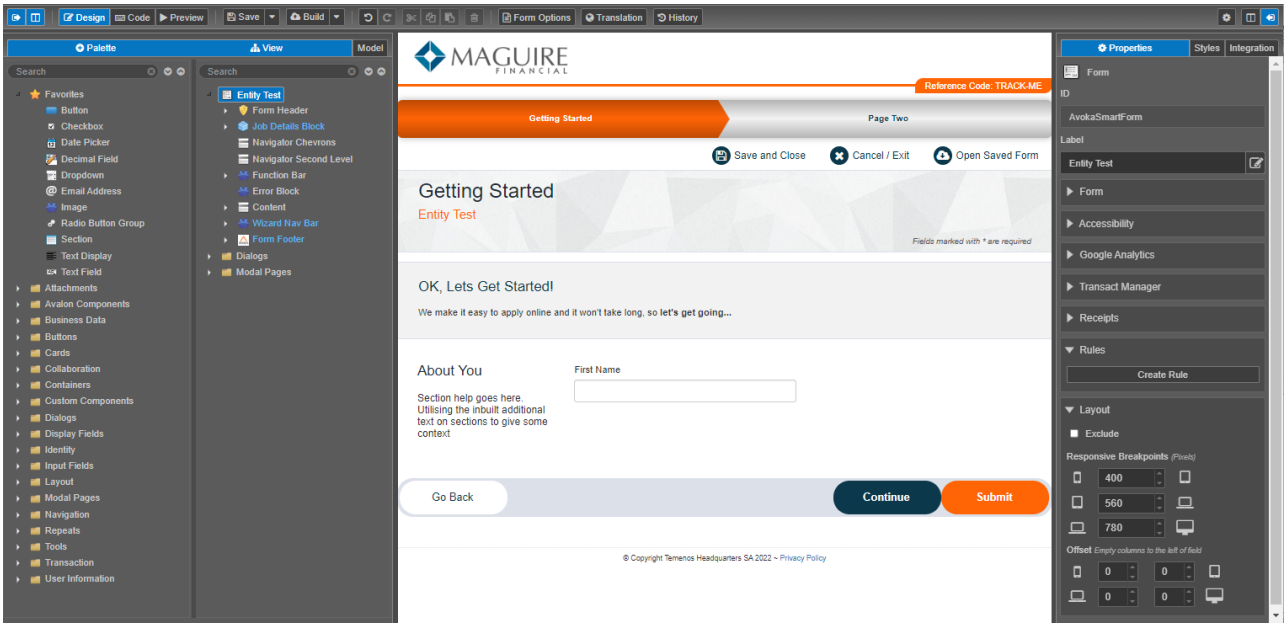
Properties Pane

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

The Maestro editor comes with the Properties pane that allows you to check and modify properties of a selected component. This is where you adjust a component's behavior by updating its properties. Different components have different sets of properties, so the Properties pane dynamically shows them as you select each component.

To view and edit component's properties in the Property pane:

1. Open a form or a template in the Maestro editor.
2. Select a component and click the Properties tab.



   Every component has its own set of unique properties, which you can check here. However, all components have the following common properties:

   - ID
   - Label
   - Rules
   - Layout
3. Find a property you need. Properties are grouped by related settings so you may need to use the arrow to expand the group to see all available options.

4. Right-click the property field and select one of the following options:

   NOTE

   These options are only available when you edit a template.

   - Add to Brand Options to add the property to a brand.
   - Remove From Brand Options to remove the property from a brand.

     NOTE

     You can only see this option if the property has been added to the brand.

   - Add to Form Options to add the property to Form Options.
   - Remove from Form Options to remove the property from Form Options.

     NOTE

     You can only see this option if the property has been added to Form Options.

   You can also see the additional information about a selected component:

   - Item - for example, Item: Dropdown
   - Property - for example, Property: Add Blank Value
5. Expand the Layout section to arrange components on the form according to your device's or browser's screen sizes. For more information, see component layout.
   - Select the Exclude checkbox to exclude the selected component from the XML as well as from the published form so it won't be present in the HTML form.
   - Select the Hide Label to hide or show a component label. Most components should have a label, but sometimes you may want to hide the label from an applicant on a form.
   - Select the Label Placement Left checkbox to change label location to the left. For the sake of usability, labels should generally be placed above input fields. This is how Maestro's components are configured by default. You can, however, change the placement of a label to the left of an input field. This is typically done on receipts as a way of saving space. For more information, see component labels.
6. Click Save to update the component's changes in the form or the template.

NOTE

You have more control over a component's appearance via the Styles pane.

Next, learn about the Styles pane.

Styles Pane

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Maestro editor comes with the Style pane that allows you to modify the appearance of a component. Although a template used in a Maestro form enforces the majority of the styles, you can use this pane to add minimal modifications to the visual look and feel of forms and components.

Form Builders should not use the Styles pane often, if ever. In most cases, Template Designers manage all styling in the template as it can get complicated. You should always speak to the Template Designer before making any styling changes.

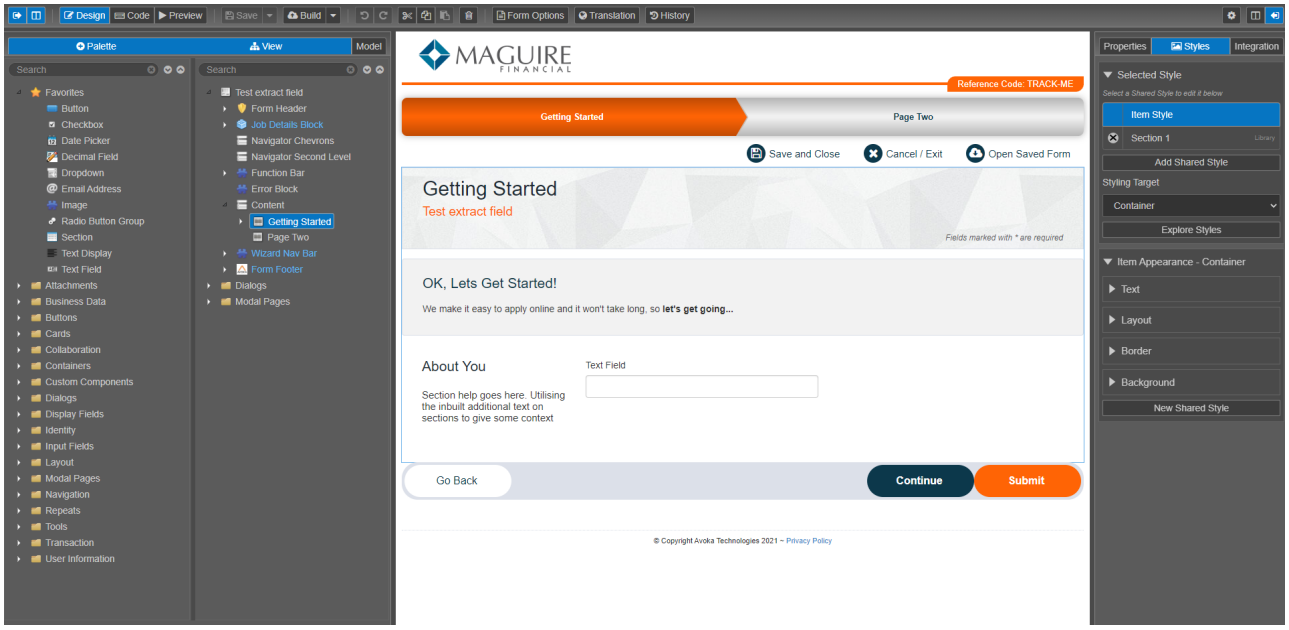
For example, you can apply multiple shared styles and item styles to a particular component. What happens if one of these styles defines the label to be red, and another defines it to be blue? What color will be chosen?

The answer is that the styles will be applied in the order that they are defined in the master stylesheet that is maintained by Maestro. Any style setting that is applied after a previous one will overwrite the previous one. You can't control what order the various styles will appear, except that the item style will always override any shared style settings. Maestro comes with the Explore Styles functionality that allows you quickly find out what the effective style of a component is and in which order CSS is arranged.

Styling uses CSS and LESS, so you should be fairly familiar with the concepts to create effective and reusable styles. For more information, see https://www.w3.org/Style/CSS/learning.

To view and edit component's styles in the Style pane:

1. Open a design in the Maestro editor and select the Style tab.

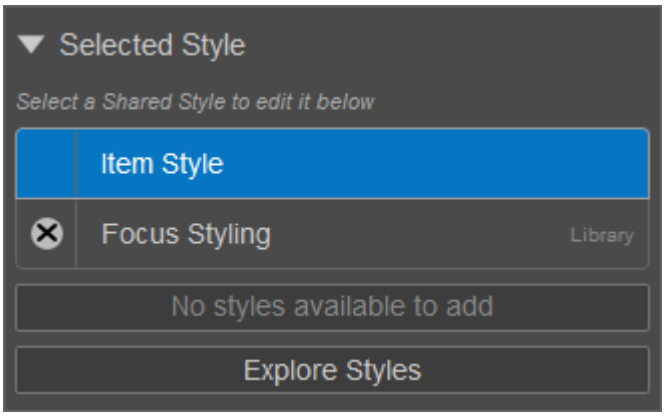


2. Find a style you need to modify. Styles are grouped by related settings so you may need to use the arrow to expand the group to see all available options.
3. Click Save to update the component's changes in the form or the template.

Main Style pane functionality is described in details below.

Selected Style

A list of styles applied to the component appears beneath the **Selected Style** heading.



The first style in this list - **Item Style** - is unique to the selected component and can only be modified via the GUI (using the controls under the **Item Appearance - Container** heading).

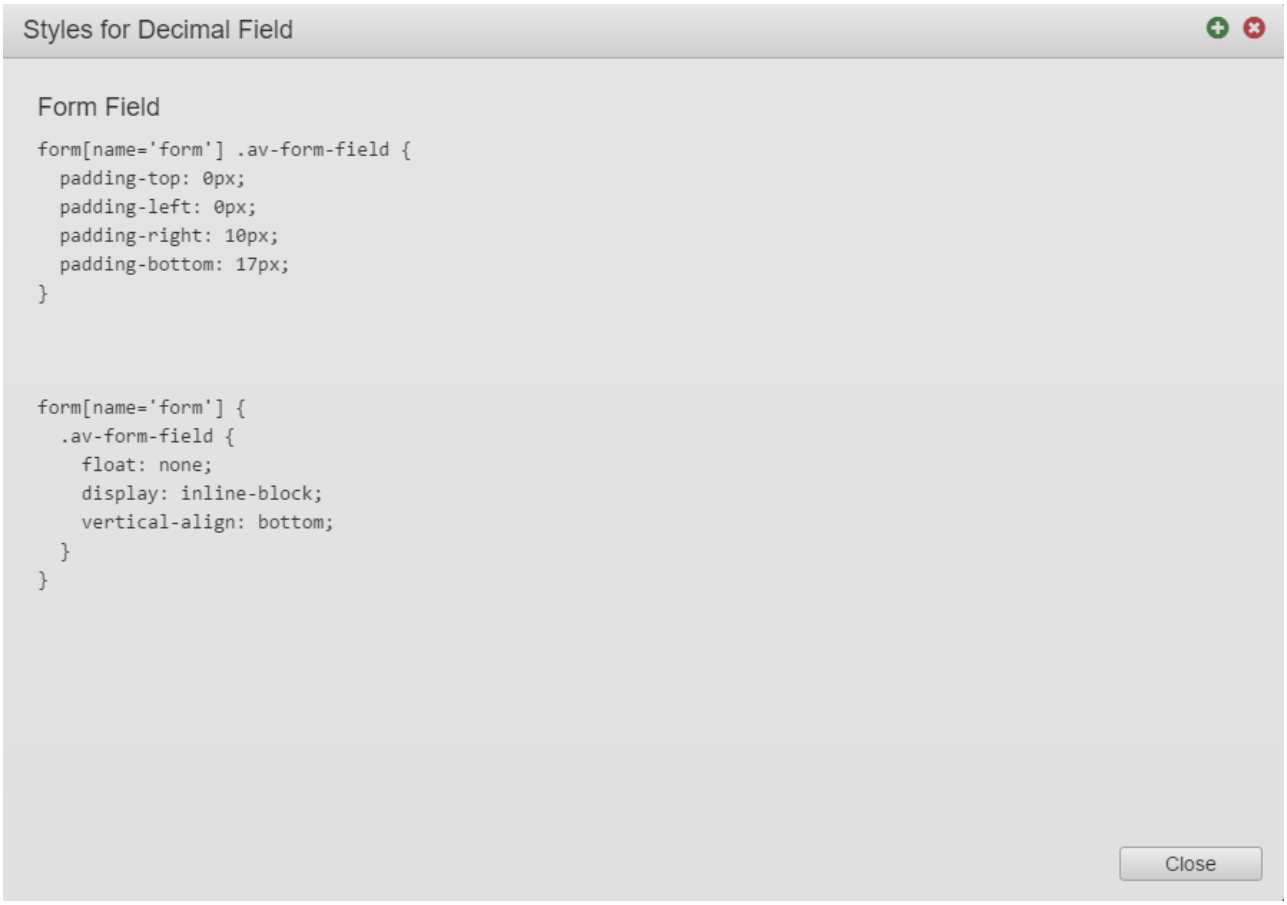
All other styles that appear in this list are shared styles. These are styles that can be used by multiple components and edited as CSS[1] or via the GUI. You always, however, have more control when editing the CSS file.

Beneath the list of styles, the following options are available:

- Add Shared Style - Add an existing shared style to the component. The styles in this list depend on the available component styles.
- Explore Styles - View the CSS and LESS[2] for all of the component's styles, which is the effective CSS for the component. This read-only view is especially useful for Template Designers to quickly check what styles are applied to the selected component.

[1]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.

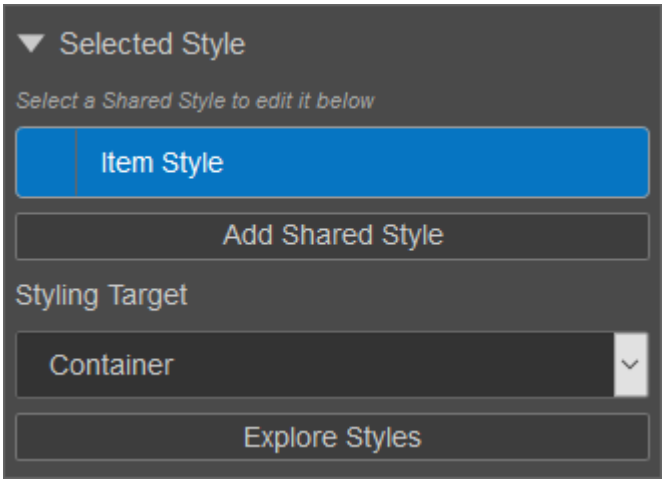
[2]LESS is a dynamic preprocessor style sheet language that can be compiled into Cascading Style Sheets (CSS) and run on the client side or server side.



- Apply to Receipt Only - If enabled, the selected style will only be applied to the component when it's rendered within a receipt.

Styling Target

When a component has multiple styling targets, an additional **Styling Target** property is shown:



This option allows you to adjust the style of an individual part of a component, such as the label of a Text Field component or the input of a Dropdown.

Even the simplest components in Maestro have a reasonable level of complexity. For example, the simple Text Field component consists of three separate parts.

- Label - which is simply a piece of text
- Input - which is the area into which you can type
- Container - which simply groups all elements into one

These separate parts are known as Targets, and you can style each of these separately.

Once you have selected one of these Styling Targets, you can use the Style settings to specify the Styles for each target separately.

For example, the following example shows a Text Field where the Border Style has been set for different targets.

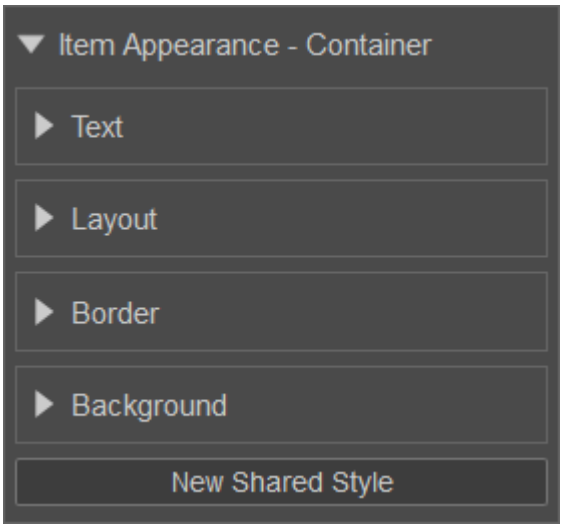
- Container - Border color set to Blue
- Label - Border color set to Red
- Input - Border color set to Yellow



Different types of components will have different targets, and you may need to experiment to understand which target affects which part of the component. Using the border settings, as shown above, can help to identify the different parts of the component. Alternately, you can also look at a form containing that component type in your browser and use the browser "Inspect" function to understand the structure of the component.

Normally, when you click on a component and start modifying its style, you are changing the Item Style, which is the styling for just that one item on the form. This is fine when you want to change the styling of just that one field. However, if you want to create a whole lot of components that all look the same, and you want these to be used consistently, then you need to create a shared style.

Item Appearance - Container



By default, the section that appears beneath the **Selected Style** section is called **Item Appearance - Container**. The first part of this heading refers to the currently selected style -- by default, **Item Style** -- and the second part of the heading refers to the styling target -- by default, the **Container** target.

If you select a different style from the **Selected Style** list, or select a different option from the **Styling Target** dropdown, the name of this section will change.

In addition to this, certain buttons will appear at the bottom of this section when certain conditions are met:

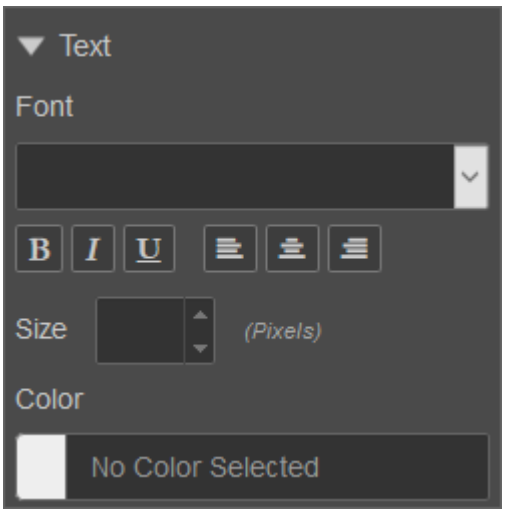
Label	Description
New Shared Style	Appears when an Item Style is selected.
Edit Custom CSS	Appears when a Shared Style is selected. For more information, see edit custom CSS.
Save to Shared Style	Appears when the selected style has been modified via the GUI. This is often the case for forms that use a template with one or multiple brands.
Clear Item Styles	Appears when the selected style has been modified via the GUI.

The Style Editor has many of the standard CSS properties without needing to understand the details of CSS. It allows you to specify styles using the following four options:

- Text. Allows you to specify attributes of text, including the Font or Typeface, bold, italics and underline orientation (left, center or right), and size and color.
- Layout. Allows you to specify the layout of the element within its bounding rectangle. This includes internal padding, and minimum and maximum heights and widths.
- Border. Allows you to specify the border (if any), style of line, width, and color, and the radius of the corner.
- Background. Allows you to specify the background color, either as a flat color or a blend, or to specify an image. It also allows you to specify the alpha transparency.

These options are always visible.

Text



Under the **Text** heading, the following properties are available for adjusting the text in the selected style:

Label	Description
Font	
Size	
Color	

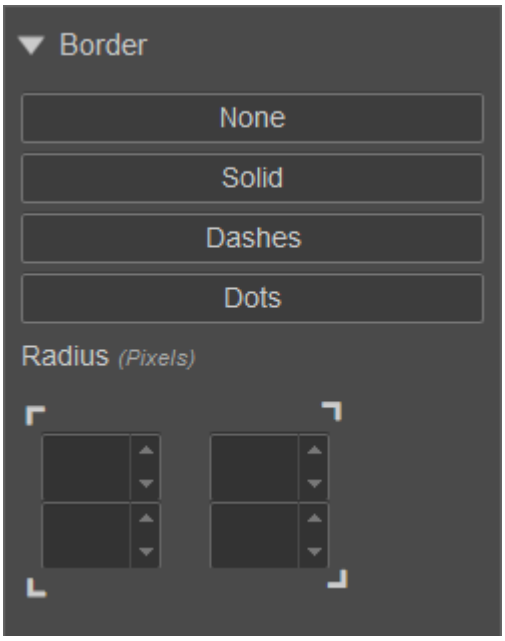
Layout



Under the **Text** heading, the following properties are available for adjusting the layout of the selected style:

Label	Description
Padding	
Min Height	
Max Height	
Min Width	
Max Width	
Vertical Alignment	

Border



Under the **Text** heading, the following properties are available for adjusting the border of the selected style:

Label	Description
Type	
Radius	

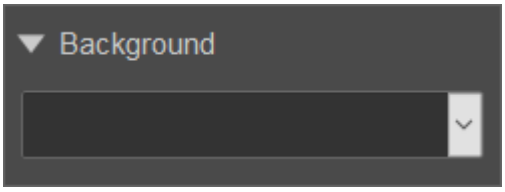
Background

Under the **Background** heading, there's a dropdown with the following options:

- (EMPTY)
- None
- Solid
- Left to Right
- Top to Bottom
- Image

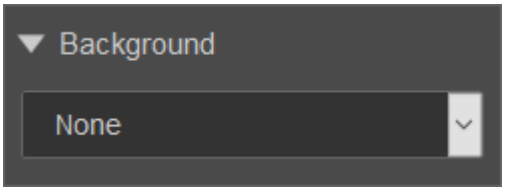
Depending on which option is selected, different properties will become available that affect the background of the selected style.

By default, the dropdown is empty:



In this state, a background will not be applied to the selected style. If a background has been applied by a different style though, it will continue to appear.

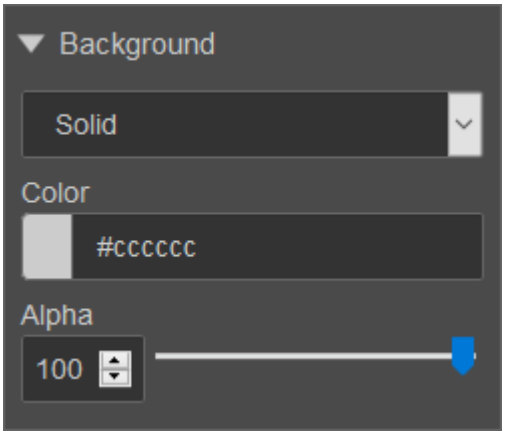
If you select **None**, any existing background on the selected style will be removed.



If you select **Solid**, a solid background color will be applied to the selected style.

The color is defined by two properties:

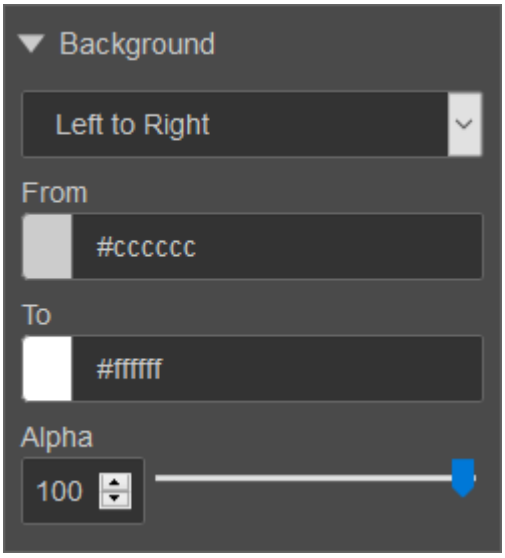
Label	Description	Default Value
Color		#cccccc
Alpha		100



If you select **Left to Right**, a left-to-right gradient background will be applied to the selected style.

The gradient is defined by three properties:

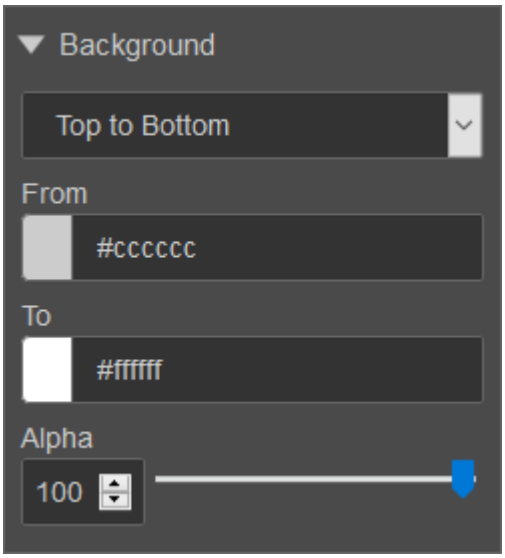
Label	Description	Default Value
From		#cccccc
To		#ffffff
Alpha		100



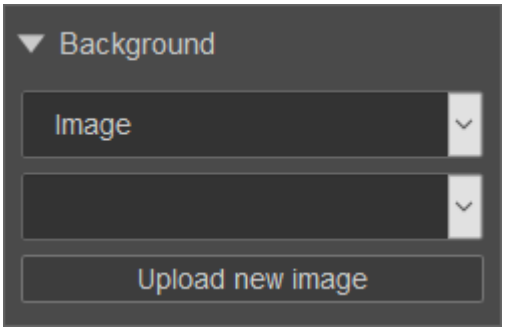
If you select **Top to Bottom**, a top-to-bottom gradient background will be applied to the selected style.

The gradient is defined by three properties:

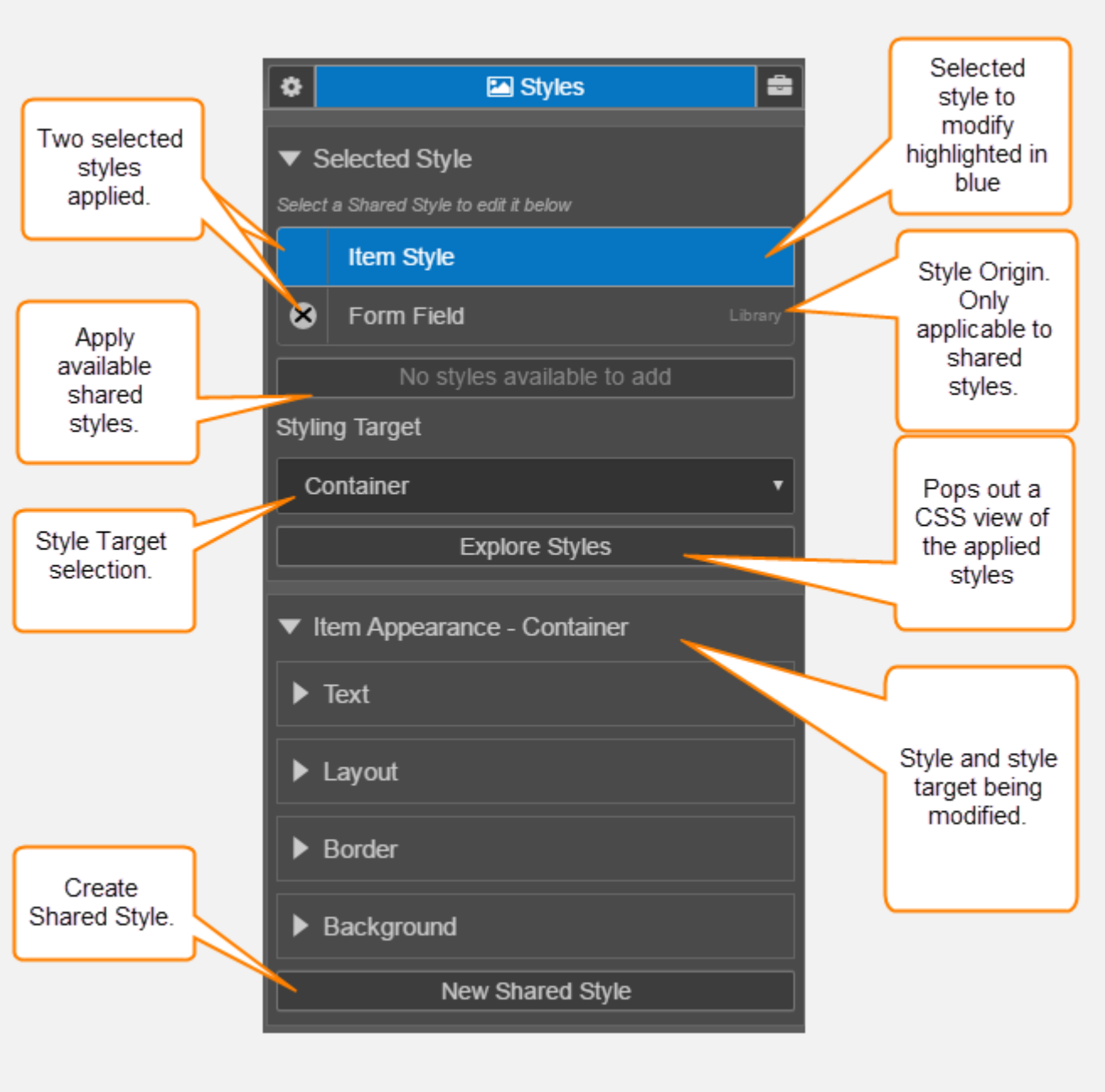
Label	Description	Default Value
From		#cccccc
To		#ffffff
Alpha		100



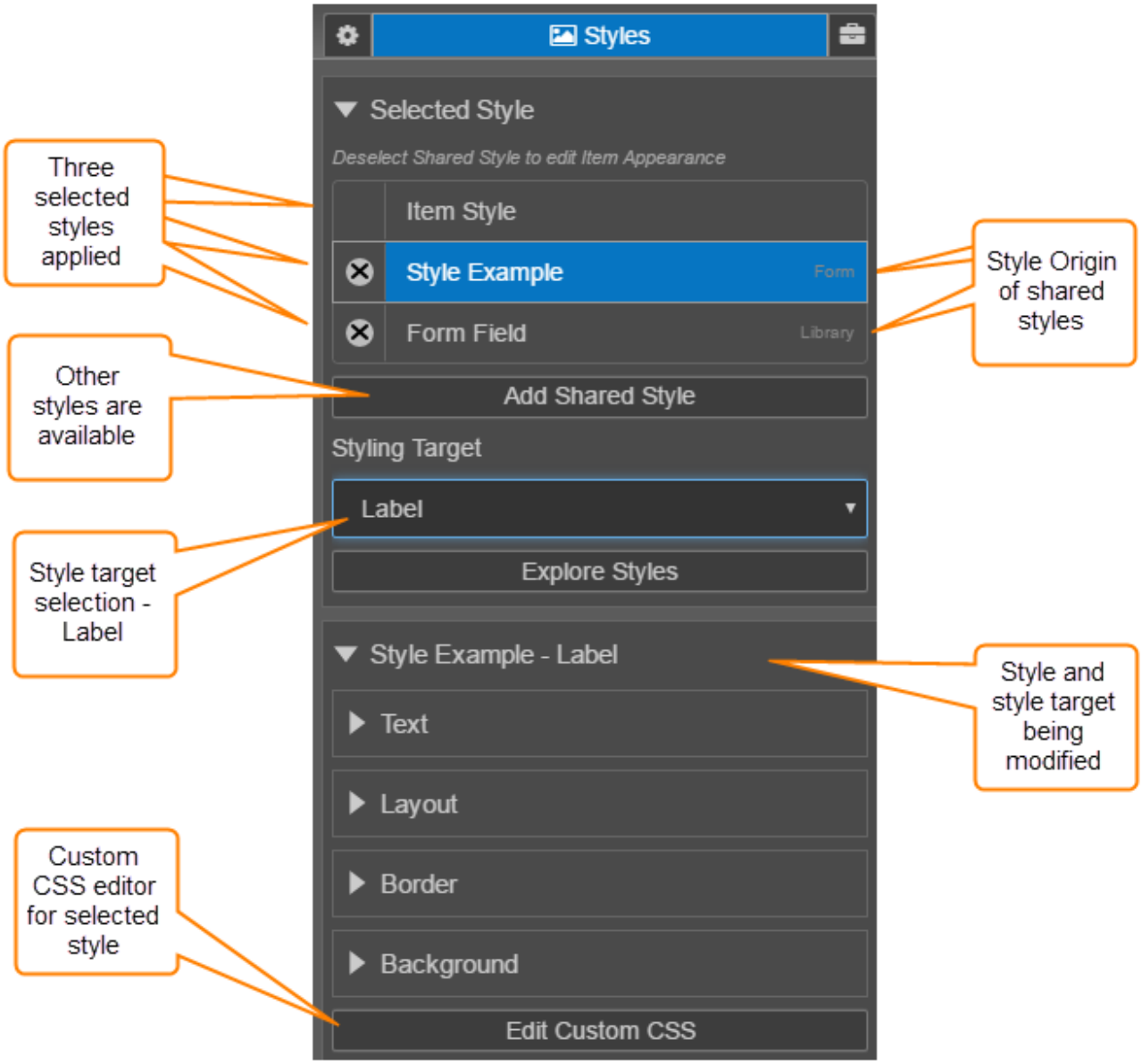
If you select **Image**, you'll be able to choose a background image from a list or upload a new image.



The example below shows the difference between components with the individual and shared styles applied:



A component with the default Item Style modified.



A component with the Shared Style modified.

Next, learn about the Integration pane.

Edit Custom CSS

MaestroThe UI design product. | Form Builder Platform Developer | 19.05 This feature was updated in 19.05.

The Maestro editor allows you to edit a shared style of a selected component in the style editor using CSS[1] and LESS[2] syntax. This is the most efficient way to update a style of a template or a form, so Template Designers and Form Builders are encouraged to use this style editor. The style editor features syntax highlighting, variable insertion and auto-indentation to help you work with CSS code.

NOTE

Any changes you make using the style editor will create a local override for the selected style, as opposed to updating the shared style directly. For more information, see Styles pane.

To edit custom CSS of a shared style:

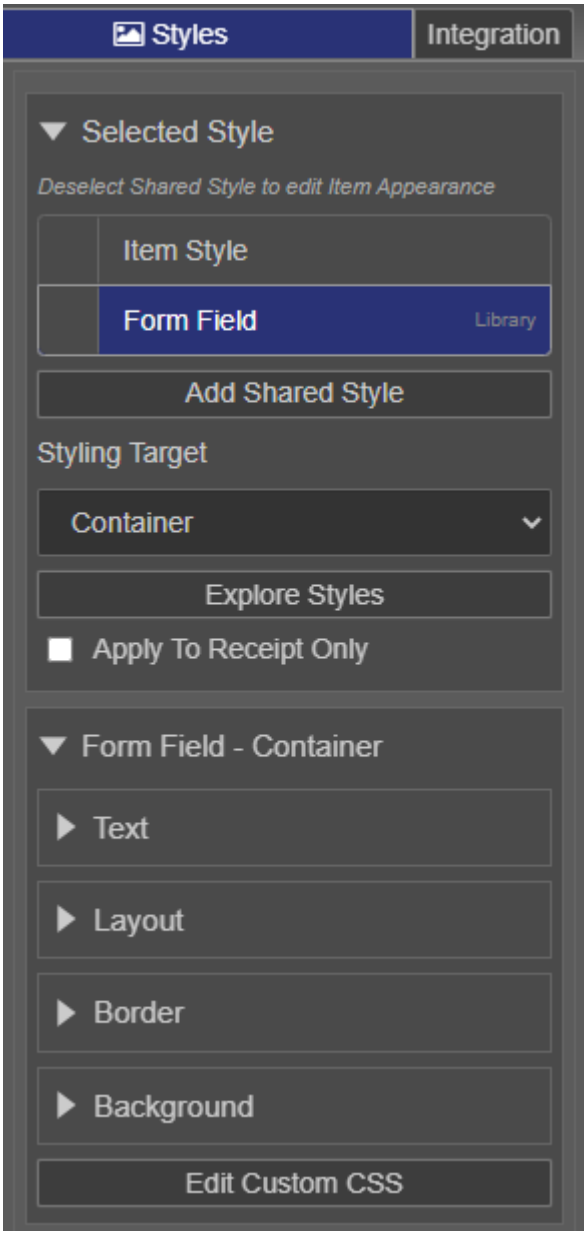
1. Open a design in the Maestro editor.
2. Select a component and click the Styles tab to view available shared styles. For more information, see the Styles pane.
3. Select a shared style under the **Selected Style** heading and click **Edit Custom CSS**.

   NOTE

   The **Edit Custom CSS** button is not available if Item Style is selected.

[1]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.

[2]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. Edit a shared style in the text area of the style editor using both the CSS and LESS syntax to benefit from features such as nested selectors, variables, and mixins. You can also double-click an item in the Navigator bar on the left side of the editor to automatically insert a variable into the CSS, for example, `color: @av-slide-menu-item-active-color`. | 19.05 This feature was introduced in 19.05.



5. Click **Save** to persist those changes.
6. Click Preview to verify the design's changes you've made. You can also view a new style override in the Resources tab.

Next, learn about brand theme options.

Integration Pane

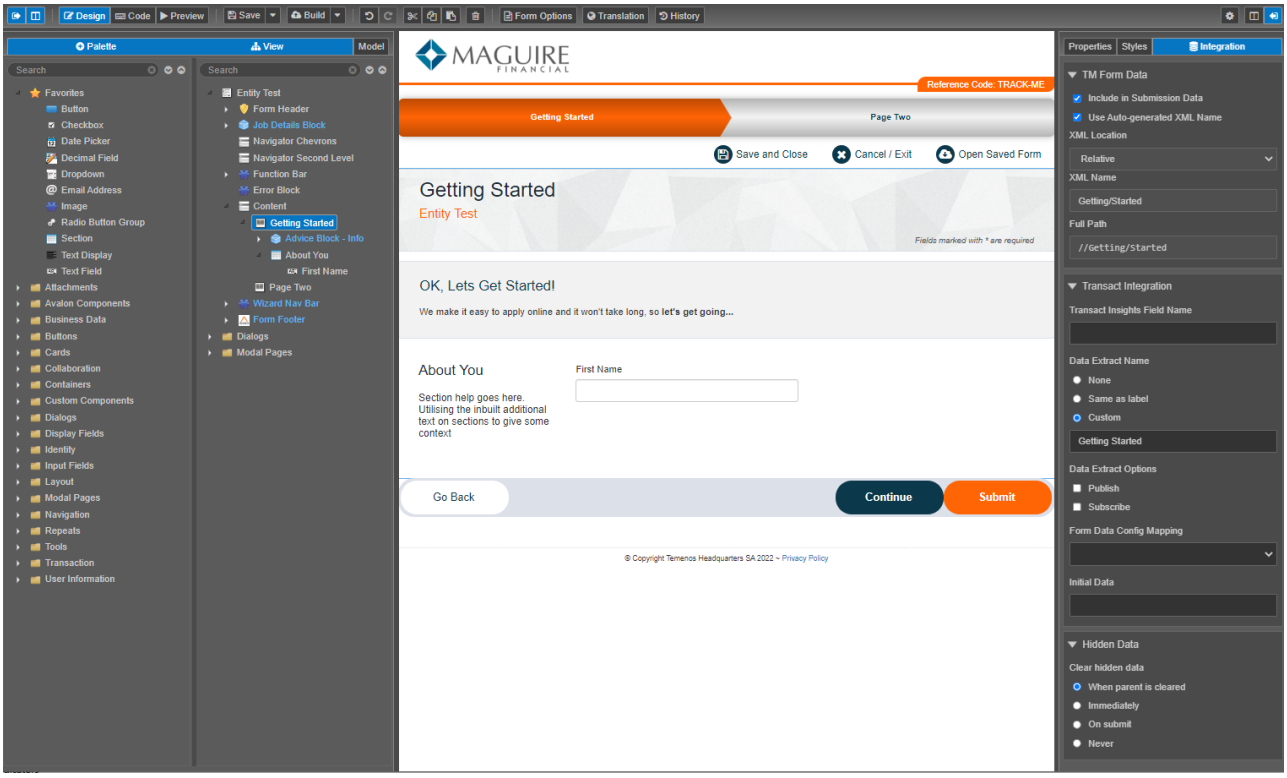
MaestroThe UI design product. | Form Builder | 18.05 This feature was updated in 18.05.

The Maestro editor comes with the Integration pane that allows you to configure how a form can be integrated with other various modules of the Temenos Journey Manager, mainly Journey Manager.

> NOTE
>
> Prior to Maestro 17.10, the **Integration** pane was known as the **Data** pane, which included Field Reference ID that is now called ID and resides in the Properties tab.

To configure a form or a template in the Integration pane:

1. Open a form or a template in the Maestro editor and select the Integration tab to see the integration options.



The Integration pane is broken down into the following sections:

- TM Form Data
- Transact Integration
- Hidden Data

TM Form Data

This section allows you to configure the data that is sent to Journey Manager. The configuration options vary depending on the data structure of a selected component. For more information, see form XML data payload.

2. Select the Include in Submission Data checkbox to include the component's data in the submission XML.

   NOTE

   Clear the Include in Submission Data checkbox to adjust the relative positions of the component and all its children but keep them in the submission data.

3. Select the Use Auto-generated XML Name or the Use Entity XML Path checkbox to generate the XML Location, XML Name and Full Path properties based on the ID of the component, for example, `//AboutYou/FirstName`.

   NOTE

   You can't see this option unless the Include in Submission Data checkbox is selected.

   Clear the Use Auto-generated XML Name or the Use Entity XML Path checkbox to edit the XML Location, XML Name and Full Path properties manually,

   NOTE

   If the selected component is associated with an entity, for example, `aboutYou_firstName`, the Use Entity XML Path option is shown instead of the Include in Submission Data option.

4. Select an XML location of the component from the XML Location dropdown list, which can be one of the following:

NOTE

You can't see this option unless the Include in Submission Data checkbox is selected.
You can't edit this option unless the Use Auto-generated XML Name or the Use Entity XML Path checkbox is selected.

- Relative - when the component is dragged between sections and blocks, it assumes a hierarchy relative to its location and thus break the data model enforcement.
- Absolute - to guarantee the location of the XML data upon submission of the form, you must ensure that in all cases, other than inside repeating elements, where the component is not associated with an entity, that the XML location is set as Absolute.
- System Data - to place the data element in the SFMData/SystemProfile section of the XML.
- Not Stored - to remove the component and all its children from the XML. This is mainly useful for container elements.

NOTE

If you want to readjust relative positions of the component and all its children without removing them from the submission XML, clear the Include in Submission Data instead.
If you want to exclude the component from both the submission XML and the form HTML layout, select the Exclude checkbox in the Properties tab.

5. Define a custom XML name for the component in the XML Name field, which can be a single level name, for example, `FirstName`, or you can create additional structure in the XML using a two-level name, such as `personalDetails.FirstNam`e or `personalDetails/FirstName`.

NOTE

You can't see this option unless the Include in Submission Data checkbox is selected.
You can't edit this option unless the Use Auto-generated XML Name or the Use Entity XML Path checkbox is selected.
You can't edit this option if the Not Stored location is selected.

6. View the complete XML path for the component in the Full Path field. This field is not editable.

   NOTE

   You can't see this option unless the Include in Submission Data checkbox is selected.

Transact Integration

This section allows you to configure how default values are defined for a component and how to set up submission data extract mappings to be used in Journey Manager.

7. Provide a field name in the Transact Insights Field Name field that is used for integration with Journey Analytics. If there are multiple fields in a form with the same label, for example, First Name, and the Insights Field Name is not set, Journey Analytics will combine the data for all fields with the same label.

   NOTE

   By default, Journey Analytics captures data through the field label, if the Transact Insights Field Name field is not filled out with a unique identifier.

8. Select a name of the submission data extract field in Journey Manager from the Data Extract Name, which can be one of the following:
   - None - to exclude the component in the data extract. This is the default option.
   - Same as label - to include the selected component in the data extract and it uses the component's label name as the extract name, for example Save and Close, in the Manager.
   - Custom - select this option to include the selected component in the data extract and provides a text field for you to enter a component name to be used in Manager. Enter a component name, for example, Save and Close.

   For more information, see submissions data extract.

9. Select one of the following options from the Data Extract Options section:
   - Searchable - the whole data extract value can be used for as search criteria in the Manager. | 18.05 This feature was removed in 18.05

- Publish - is used by Collaboration Jobs in conjunction with the Subscribe option. This is an alternative mechanism to share data between tasks in a Form Bundle - Job Step.
- Subscribe - it consumes the publish event using the published value. This is an alternative mechanism to share data between tasks in a Form Bundle - Job Ste.

10. Select a system value that is used in Journey Manager processes from the Form Data Config Mapping dropdown list, which can be one of the following:
    - Contact Email
    - Contact Phone
    - Save Challenge

    For more information, see create save challenge.

11. Specify a default value to be displayed when the form is loaded in the Initial Data field.

    NOTE

    You can't see this option unless the Include in Submission Data checkbox is selected.
    You can't see this option if the selected component can't hold data.

Hidden Data

This section allows you to configure the behavior of form data in an item when it is hidden by a visibility rule. The clear hidden data property only checks for visibility on the parent level of the component. It does not check for visibility on the children level of the component if this level is made visible. For the hidden data property to function correctly, the current block which contains the component in the form must be hidden.

12. Select one of the following options from the Clear hidden data section to defines when (or if) the data entered into a component is cleared when the component is hidden:
    - When parent is cleared -use the parent's clearing rule
    - Immediately -as soon as the component is hidden
    - On submit - when the form is submitted
    - Never - don't discard the data regardless of the component's visibility. This is the default option.
13. Click Save to update the changes in the form or the template.

Next, learn about the Code View.

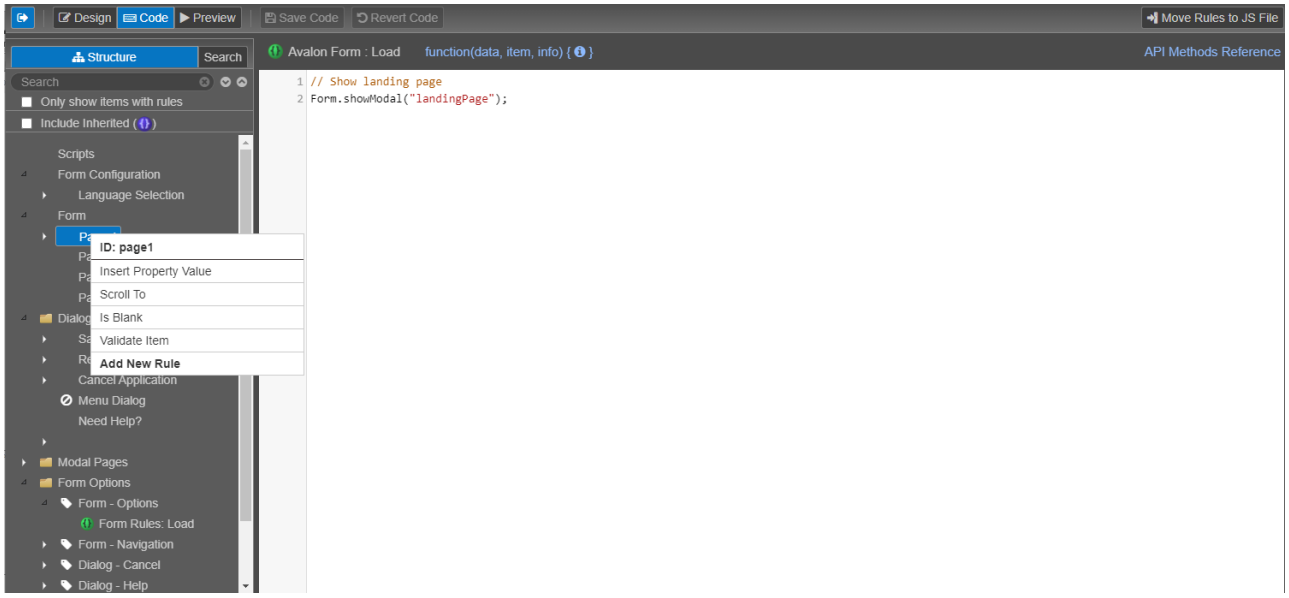
Code View

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

The Maestro editor comes with the Code View functionality, which allows you to view, create, remove and modify business rules that a Maestro form may require.

To view business rules associated to form's components in the Code View pane:

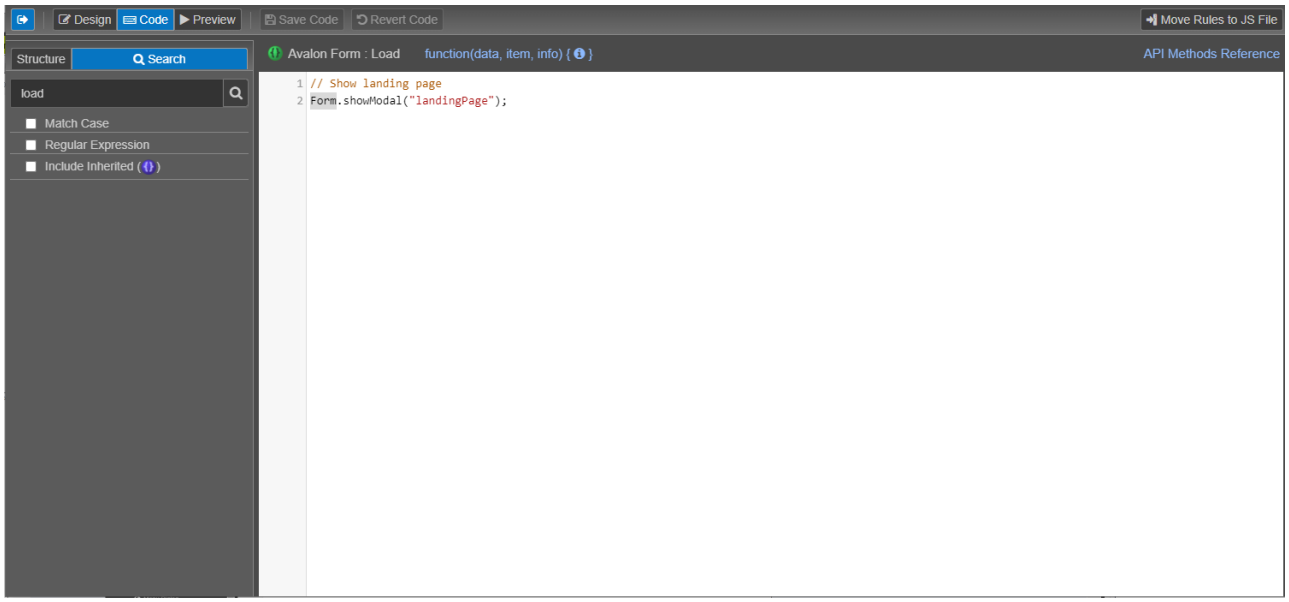
1. Open a form in the Maestro editor and click Code.



2. Select the Structure tab to view the components and associated rules in a tree view.
3. Type a component label or rule type in the Search field to look for a rule that was configured for the selected form. It does not search through the actual code, it only looks at the content visible in the design tree.
4. Select the Only show items with rules checkbox to see components with rules only. This significantly reduces the number of components to looking through.
5. Select the Include Inherited checkbox to see rules that are inherited based on the components default options or rules from a template.
6. Open the Form Options category to see more context from the category name and allows you to easily find the rule in the Form Options dialog as well. | 21.05 This feature was introduced in 21.05.
7. Select a rule in the tree to view the source code of the rule in the editor so you can edit it directly or create new rules by right-clicking any component in the displaying pane. You can also edit the rule directly in the code editor, or right-click the component to use any of the code assist options to edit the selected rule. Select Add New Rule to create a new rule for

a selected component and choose a rule you want to create in the Create Rule window. For more information, see add a new rule in Code View.

8. Right-click a selected rule and select Delete Rule to remove it.
9. Click Move Rules to JS File to move the selected rules to an external JavaScript file for use in external source code management tools, for example Git. For more information, see move rules to JS file..
10. Select the Search tab to search based on script content. This allows you to filter based on the content within the code, so it doesn't look at the component label or rule name, but rather at the code within the rule. The search results show all methods that match the search criteria, including lines within the method that match the search criteria.



11. Select the Match Case checkbox to filter the search results to only show those that match exactly according to the case structure used in the original search.
12. Select the Regular Expression checkbox to filter the search results to only show regular expressions.
13. Select the Include Inherited checkbox to filter the search results to include any inheritance that exists in the rules.

You can use the following resources from the Code View to create rules:

- API Methods Reference - it contains information on API methods published for use in Maestro business rule functions.
- Maestro Rule Functions - these rule functions can be created in the Code View or in the rule editor window accessed from the Rules section of the Properties pane.

Next, learn how to add a new rule.

Add a New Rule in Code View

MaestroThe UI design product. | Form Builder | 17.10 This feature was updated in 17.10.

To add a new business rule to a Maestro form while in the Maestro Code View, follow these steps:

1. Right-click a component.
2. Select Add New Rule.
3. Select a rule you want to create in the Create Rule window. This is the same window displayed when you click the Create Rule button on the Properties pane of the Maestro Editor.
4. Edit your new JavaScript code in the editor. You can type the code directly in the editor, or right-click the component to use the code assist options.



5. Click Save.

For more information on additional Code View resources, see additional Code View resources.

Move Rules to a JavaScript File

MaestroThe UI design product. | Form Builder | 17.10 This feature was introduced in 17.10.

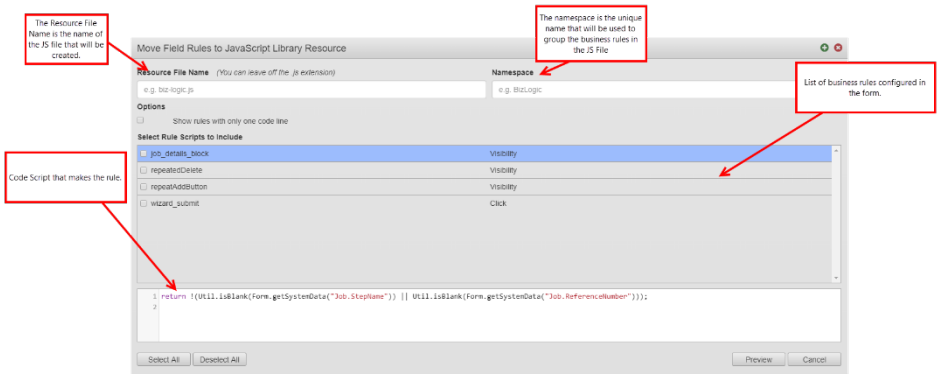
Traditionally, Maestro and other Transact tools relied on built-in code editors to create, edit, and manage Maestro business rules. However, with the move to Source Code Management (SCM) tools into the Transact platform, Maestro has introduced the Move Rules to JS (JavaScript) File feature, which allows you to move all selected business rules from a Maestro form to an external JavaScript file. This file can be later referenced in any Maestro form.

You should use the Move Rules to JS File feature to create a resource that holds all required JavaScript rules for a Maestro form. This file can then be edited in any SCM tool before being referenced into a Maestro form by using the JavaScript library component from the Maestro Palette.

The Move Rules to JS File feature should be used when you are looking to extend the functionality of your form by using Source Code Management (SCM) tools such as Git. Or if you want to consolidate all the rules for your form into one place to make editing easier as your form grows and matures.

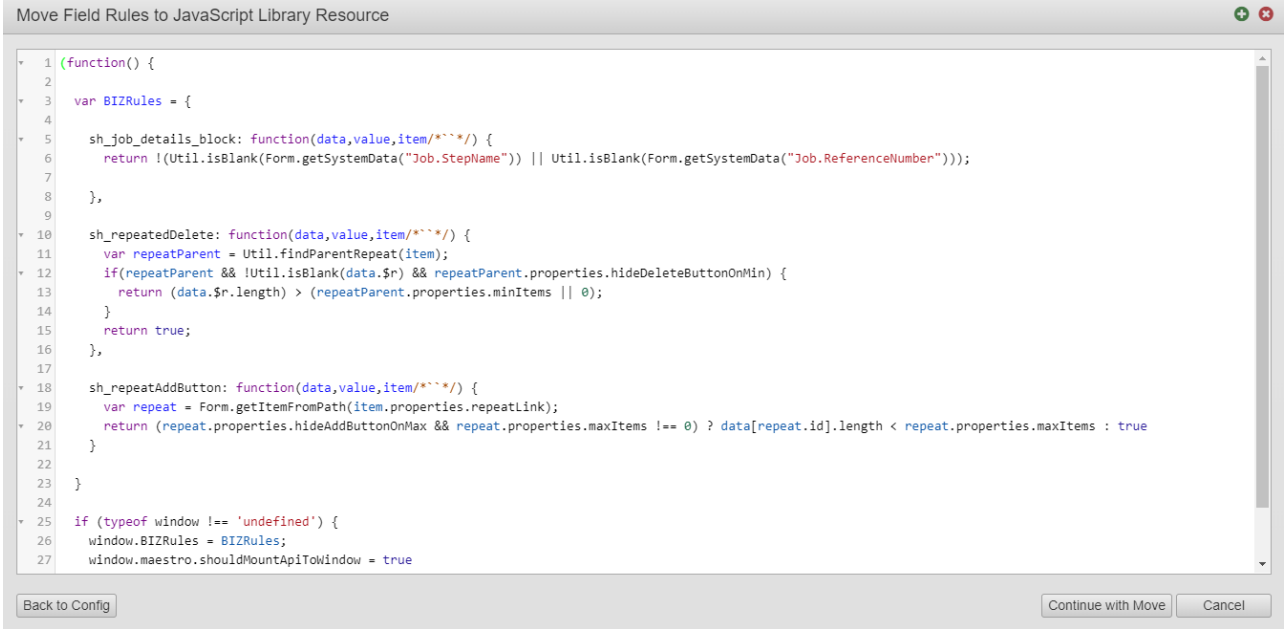
To move all configured business rules of a form to an external JavaScript file, follow these steps:

1. Open a form that has multiple business rules configured.
2. Click Code.
3. Click Move Rules to JS File.

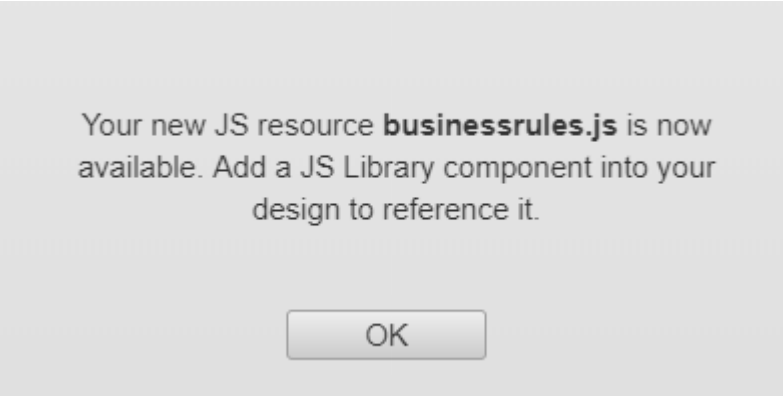


4. This dialog displays all the business rules that have been configured for the selected form. Each rule is identified and if selected, the script that creates the rule is shown as a JavaScript in the code box below the list of the rules. If your form's rule has only 1 line of code, a checkbox is presented that you can select to display these one-line rules.
5. Provide a Resource File Name for the JavaScript file. This is the name of the JavaScript file that is created when you move the selected rules to.

6. Provide a Namespace for the JavaScript File, which is the unique name that is used to group the rules in the JS file.
7. Select the rules that you want to move to a JavaScript File. Click Select All to select all the available rules.
8. Click Preview to display a preview of the JavaScript file. The screenshot below displays an example of a preview for a JS file.



9. Click Continue with Move. If you want to return to the previous dialog and select or clear other rules, click Back to Config. It creates the JS resource and displays the following dialog.



10. Click OK.

Once the Maestro rules have been moved to the JS file, you can then reference the created JS file directly in the form that you just used to create the JS file (using a JavaScript Library com-

ponent), and then download the JS file and edit it using a SCM tool before uploading the JS file back into Maestro.

Next, learn how to reference a JavaScript File into a Form.

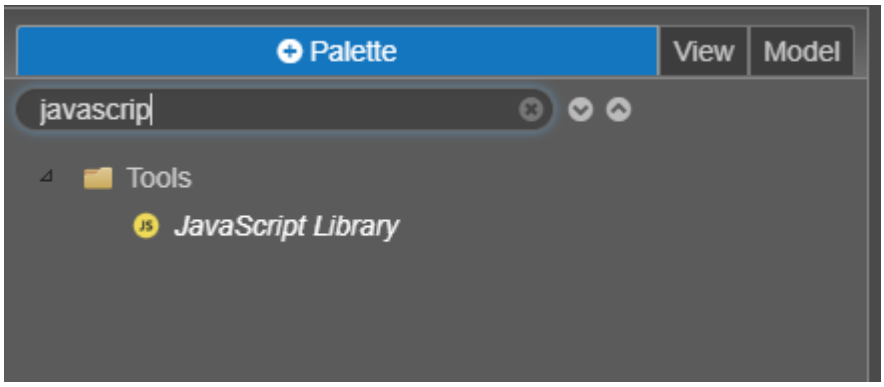
Reference a JavaScript File into a Form

MaestroThe UI design product. | Form Builder | 17.10 This feature was introduced in 17.10.

After you have moved the business rules out of a Maestro form to a JavaScript[1] file using the Code View, it is time to reference this JavaScript file back into the form.

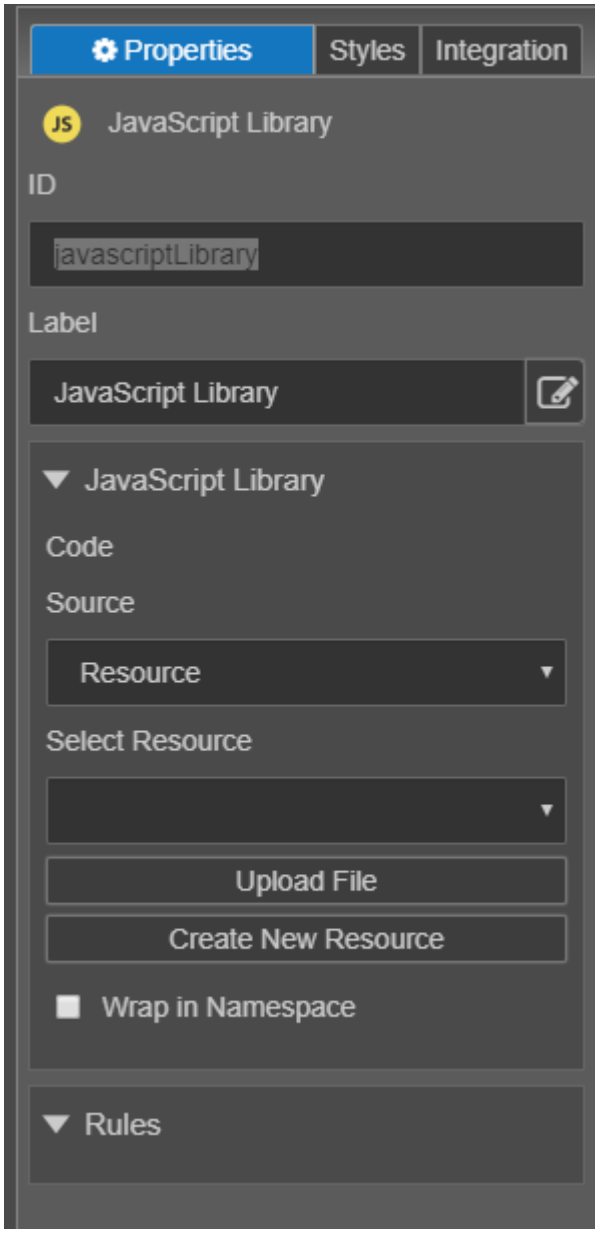
To reference the JavaScript file in a form:

1. Open the form that you moved the JavaScript rules from.
2. Use the Maestro Palette and select the JavaScript Library component.

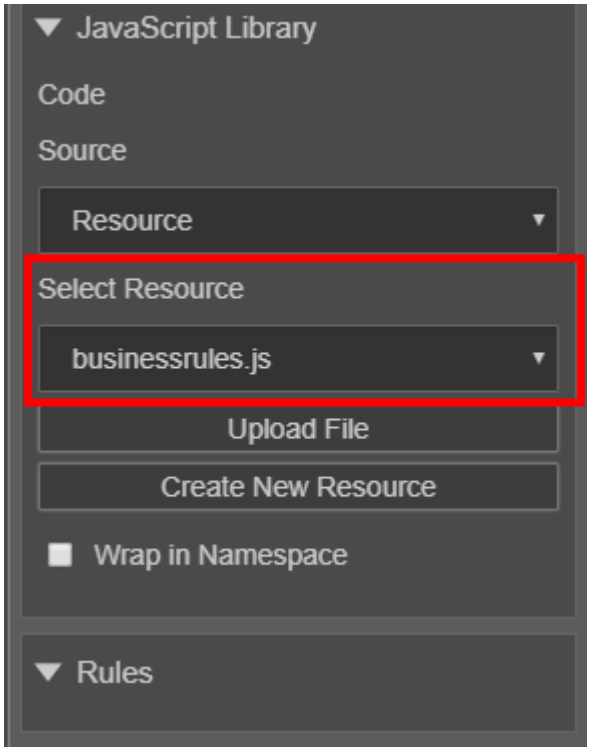


3. Drag and drop the JavaScript Library component onto the form.
   Dropping the JavaScript Library component onto the form will display the JavaScript library properties.

[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.



4. Select the Source of the JavaScript file. For a JS File that is created using the Move Rules to JavaScript File feature, the source will always be, Resource (which is selected by default).
5. Select the Resource from the Resource dropdown menu. This dropdown list will identify each of the JavaScript files and resources that have been created for the form.



6. Save the form to apply the business rules from the JavaScript file to the form.

Once you have referenced a JavaScript file in a form, you can then download the JavaScript file to edit it outside of Maestro before uploading it back into Maestro.

Next, learn how to download a JavaScript file.

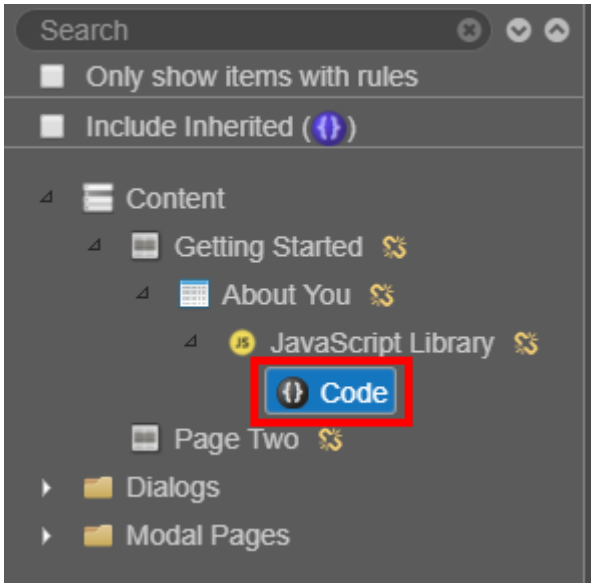
Download and Upload a JavaScript File

MaestroThe UI design product. | Form Builder | 17.10 This feature was introduced in 17.10.

Once the rules of a Maestro form have been moved to a JavaScript file and referenced in a form, you have the option of downloading the JavaScript file and editing it in a tool outside of Maestro. After editing the form outside of Maestro (in an IDE), you can then upload the JavaScript file back into Maestro before referencing it again in the Maestro form.

To download a JavaScript file:

1. Open a form in Code View.
2. Expand the tree to display the JavaScript library and select Code to display the JavaScript source code.



3. Click Download to download the JavaScript file on your computer.



4. Open the JavaScript file in any IDE[1] or text editor and edit the business rules as required.
5. Click Upload to upload the updated JavaScript file.



Next, learn about Maestro Preview Mode.

[1] Integrated Development Environment.

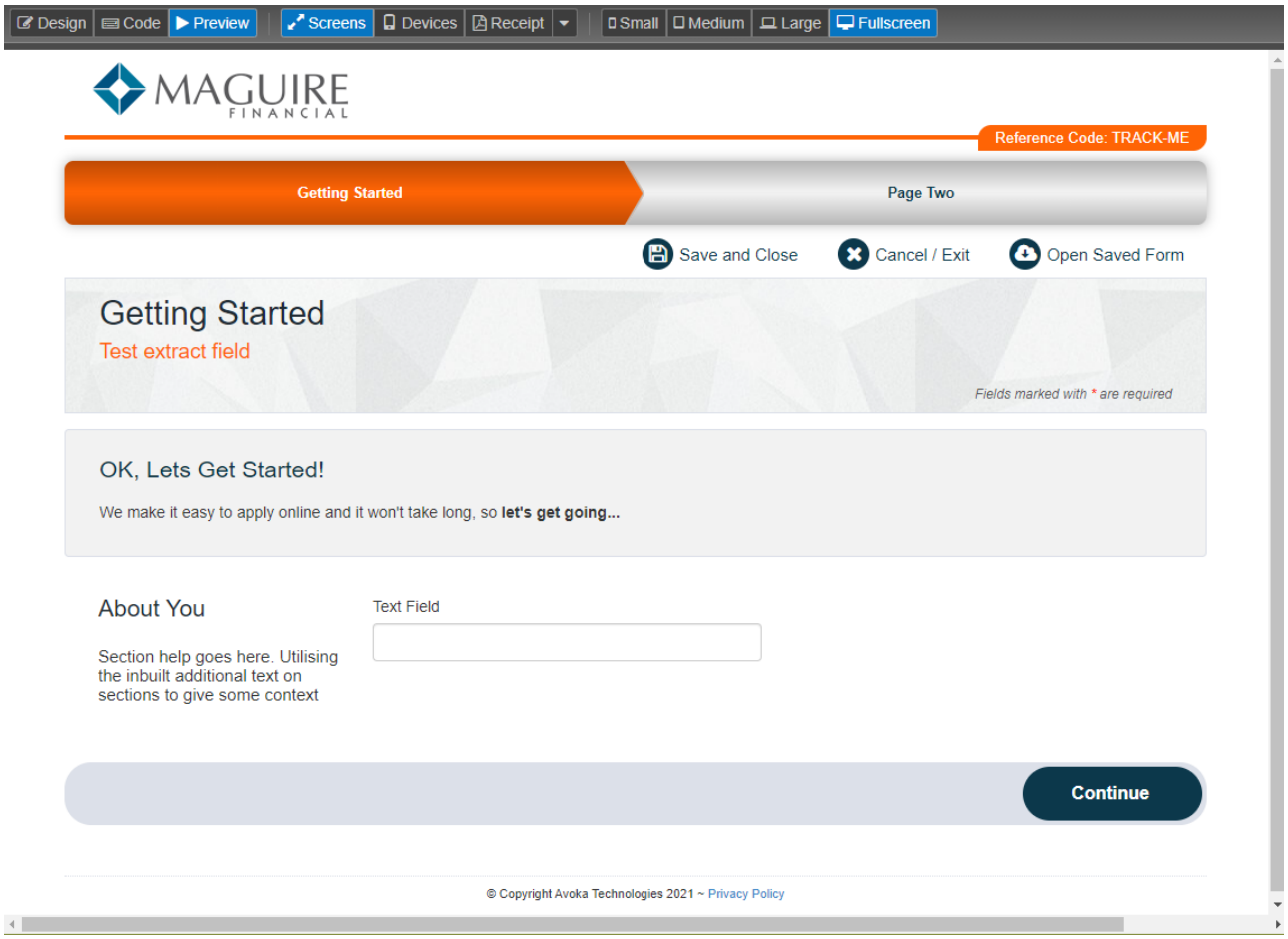
Preview Mode

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Maestro editor comes with the preview feature that allows you to quickly render a Maestro form so you can see how it will look when it's built and published, as well as try out some of form's functionality. This is a handy feature for Form Builders who need to render forms often to validate changes they make.

To open a form in the preview mode:

1. Open a form in the Maestro editor and click Preview in the toolbar.



2. Click Screens and select a screen type from the dropdown list to see how the form will look and behave on a selected screen size. The screen type can be one of the following:
   - Small
   - Medium
   - Large
   - Fullscreen

3. Click Devices and select a device from the dropdown list to preview the form on that device. The device can be one of the following:
    - Apple iPad
    - Apple iPhone 4
    - Apple iPhone 6
    - Apple iPhone 6s
    - Apple iPhone 6 Plus
    - Google Nexus 7
    - Samsung Galaxy S7

    NOTE

    This does not actually run the form on the device, but rather shows how the form will render.

    Click Rotate to toggle the orientation of the form between portrait and landscape.

4. Select a receipt type from the Receipt dropdown list, which can be one of the following:
    - PDF Receipt
    - HTML Receipt: An HTML receipt allows you to interactive with form fields in preview which is very useful for debugging forms.
5. Click Design to exit the preview mode.

Next, learn about Maestro's organizations.

Build a TM Form Version

MaestroThe UI design product. | Form Builder | 22.10 This feature was updated in 22.10.

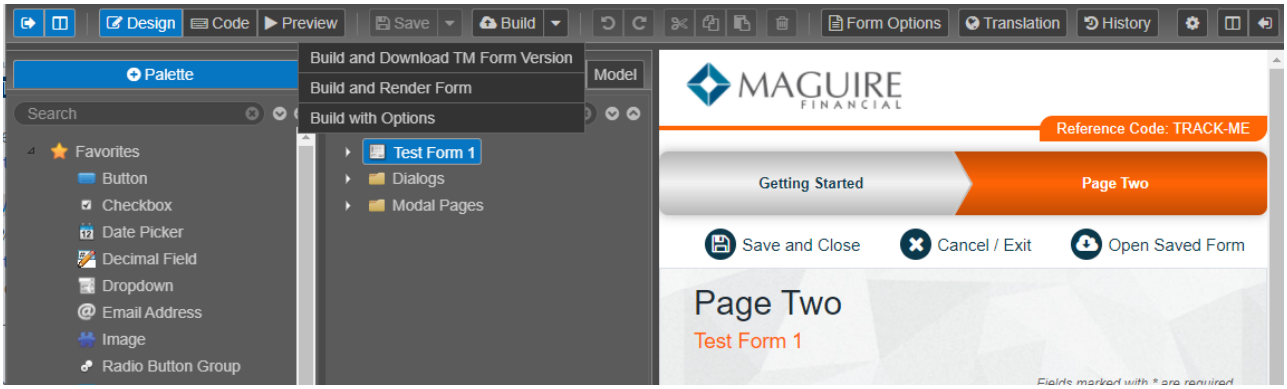
Maestro allows you to build a form as a TM Form Version ZIP file, which includes a FAR file and all related form's assets, so you can easily deploy it in a form space in Journey Manager. You develop and build the form in the same tool - the Maestro editor, which is handy and saves you time.

If the form containing multiple brands, you can build the form using one or many brands available to the form.

If you don't need to deploy a TM Form Version ZIP file right away, you can build and review it first, and import it into Manager later , when you need to. Usually, you import the TM Form Version ZIP file to Test and UAT environments, then you promote a form to a production environment by exporting and importing it using Manager's functionality.

To build the TM Form Version of a Maestro form:

1. Open a form in the Maestro editor and select one of the following options from the Build dropdown menu:



2. Click Build to build a TM Form Version. This doesn't download the TM Form Version or display the rendered form, which is useful when you are re-building and updating a TM Form Version that has already been imported into Journey Manager.
3. Click Build and Download TM Form Version to build a TM Form Version and then immediately download the ZIP file needed to import the Form Version into Manager.
4. Click Build and Render Form to build a TM Form Version and render the form in a new tab. You can download the TM Form Version later from the Management Dashboard. This option is handy when you need to view or test a rendered version of the form.

5. Click Build with Options to apply the build and brand options before building the TM Form Version.



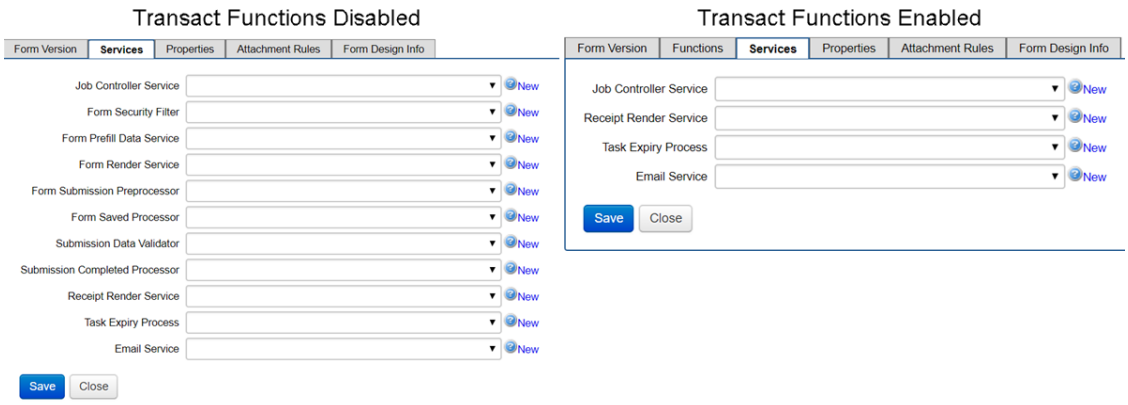
Enter a new TM Form Version name in the TM Form Name field, in case you want to change it from the default name, which is the existing name of the form.

Specify Build Options to build the form according to your requirements:

- Select the Use server side Transact Functions (requires TM 17.10 +) checkbox to use Transact functions to call services in Manager. | 17.10 This feature was introduced in 17.10.

  It's important to notice that enabling Transact functions causes the following:

  1. It limits the number of available services you can configure for your form version, as shown below:



2. It removes a number of `SystemProfile` properties, such as `<TestMode>-false</TestMode>`, from the form and from the submission XML data.

However, enabling Transact functions allows you to configure Transact functions, which were not available before that.

> NOTE
>
> You can see whether the Transact function are available in a form when you configure a form version in Manager.

> NOTE
>
> Transact functions require Journey Manager 17.10 and higher.

- Select the Minify Code checkbox to reduce the size of the form and related JavaScript[1] resources without changing its functionality so they can be sent out to a client browser more efficiently.

[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.

NOTE

Code minification[1] functionality has been enhanced in the following Maestro versions:

- Prior to 18.05 - JavaScript resources are concatenated into the `app.js` and then minified, but not transpiled[2].
- In 18.05 - JavaScript resources are concatenated into the `app.js`, then minified and transpiled. The transpiling support to the rules, the runtime framework and any widget scripts so that users could use ES6[3] constructs in their rules while maintaining IE 11 support as it is still a supported browser. External JavaScript files are the responsibility of the form developer to maintain for minification and IE 11 compatibility.
- Since 19.05 - JavaScript resources are added as they are as separate resources and provided to the form as-is. External JavaScript files that are loaded with just a URL are not processes at all other than to add the script tag that adds them to the form.
- Since 22.10 - a reference to Internet Explorer 11 was removed from this option.

NOTE

JavaScript resources are the object dropped on to a form and a user has selected a JavaScript resource that is in the form or other available library. External JavaScript files are added through the JavaScript object and setting the URL property.

---

[1]Minification is the process of removing all unnecessary characters from the source code of interpreted programming languages or markup languages without changing its functionality.
[2]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.
[3]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.

NOTE

Reducing the code makes it difficult to read it, so you have to consider deselecting the Minify Code checkbox when debugging the form.

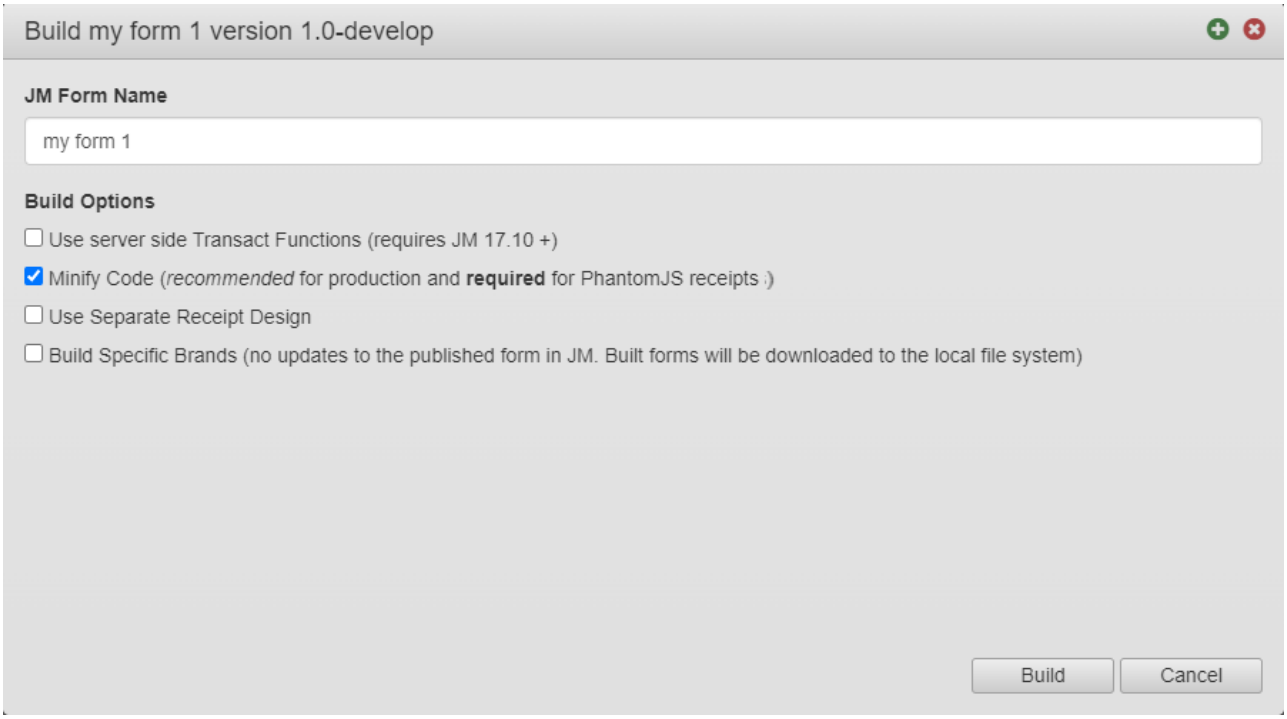
- Select the Use Separate Receipt Design checkbox to use a separate receipt, if a receipt design is available in the form, so all built receipt assets are pulled into the FAR file. This FAR is different as it effectively has two complete Maestro designs - a form and a receipt. There is an XML file in the FAR that tells Manager that the form uses a specific HTML file for the interactive form and a different HTML for the receipt.

  NOTE

  When Maestro builds the form, it doesn't rebuild the receipt form. It simply takes the latest version of the receipt that has been built and checks whether the last update time on the receipt is after the last built time on the receipt or not. If the receipt has been changed since its last built, Maestro shows a warning that the receipt needs to be republished.

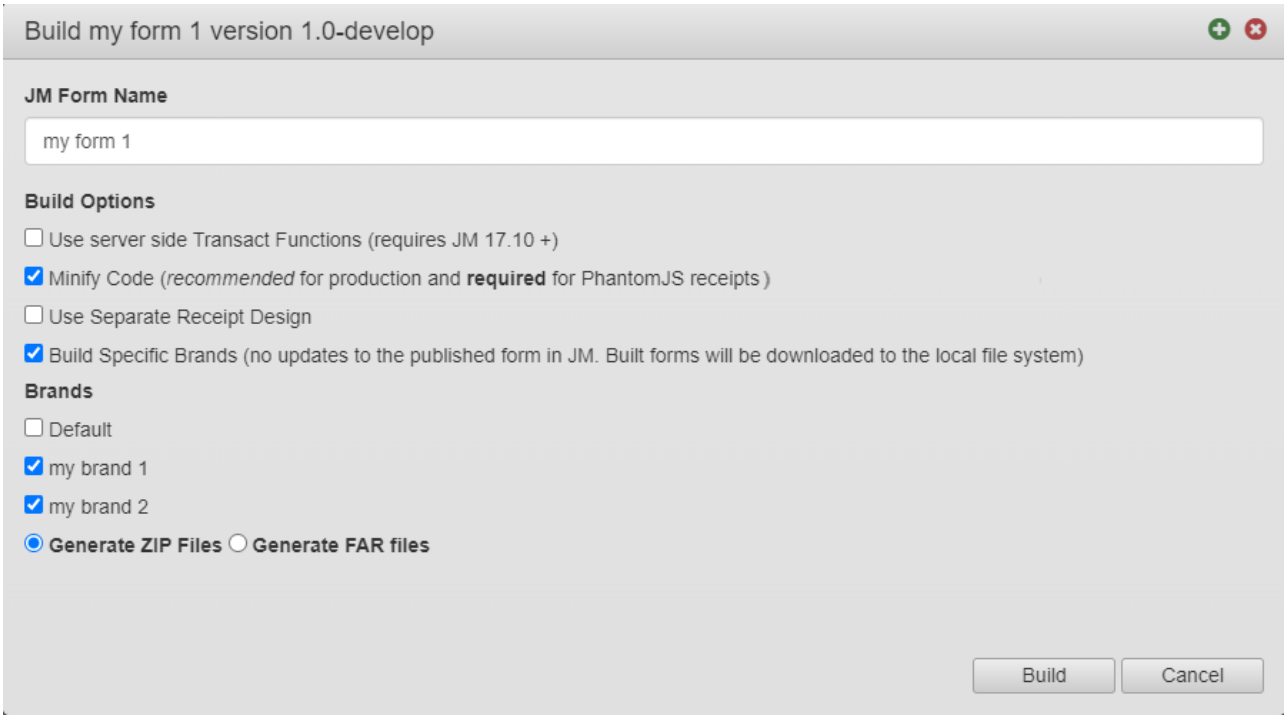
- Select the Remove Automation Framework checkbox to remove the MAF code, which reduces the download size of the form and prevent misuse of the API in production forms. The Maestro Automation Framework (MAF) provides an API which can be used by other tools to automate and investigate the state of a Maestro form. Once a form is ready for production, MAF is not required.

If the form containing multiple brands, Maestro gives you additional options to choose which brand, or all of them, to use to generate a TM Form Version.



Select the Build Specific Brands (no updates to the published form in JM. Built forms will be downloaded to the local file system) checkbox to see the the list of brands available to the form.

> NOTE
>
> You can't select the Build Specific Brands (no updates to the published form in JM. Built forms will be downloaded to the local file system) checkbox if the form has only one brand.

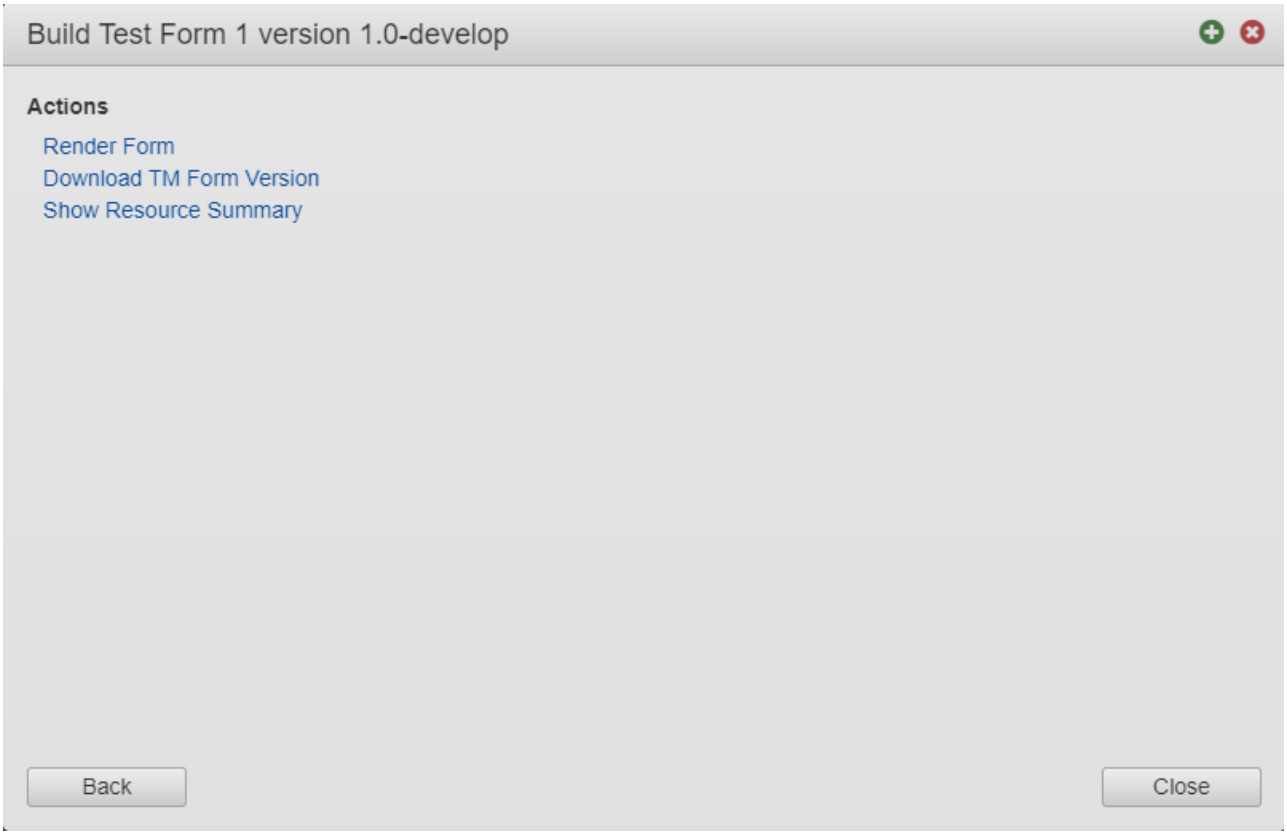


Specify Brands to build the form using specific brand or brands:

- Select one or many brands from the available list of brands defined in your template. The Default brand is always available.
- Select Generate ZIP Files to bundle the form and all relevant resources in a ZIP file, which you can deploy in various Manager environments.
- Select Generate FAR files to bundle the form and all relevant resources in a FAR[1] file, which you can deploy in various Manager environments.

Click Build to build the form. Check the build result and perform some actions as described below.

[1]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.



- Click Render Form to render the form in a new tab. It's similar to the Preview Mode.
- Click Download TM Form Version to download the ZIP file on your local PC.
- Click Show Resource Summary to see resources associated with this form and included in this ZIP file.
- Click Back to go to the previous screen.

Once a TM Form Version is built, you can download it at any time from the Maestro Management Dashboard using the Download JM Form Version option.

When building the TM Form Version, you may encounter some common errors, which you can resolve yourself.

Next, learn how to update TM Form Version.

Update a TM Form Version

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

You may want to update a Maestro form after it has been deployed to Journey Manager or migrate it to a new Journey Manager environment. This is to accommodate customer feedback or follow continuous improvement processes within your company.

To update a TM Form Version:

1. Open a form in the Maestro editor and edit it as required.
2. Build a TM Form Version.
3. Log in to the Manager server where you originally uploaded the TM Form Version ZIP file.
4. Select Forms > Forms and find the form that you want to update, which is the form that you just updated in Maestro.
5. Open the form, select a form version and click Update Design from Maestro to automatically update it form the Maestro server.

NOTE FOR ON-PREMISE AND AMAZON VIRTUAL PRIVATE CLOUD (VPC) MANAGER INSTALLATIONS

If the VPC is not managed by Temenos, in order for the updating process to function correctly, the Manager environment needs to send an outbound request over HTTPS to a Maestro host (for example, `maestro.avoka.com.au`). For this reason, it is recommended that before you attempt any update of a form version using Maestro, you first check your firewall rules and ensure that they allow this outbound request.

Next, learn about the SCM tool.

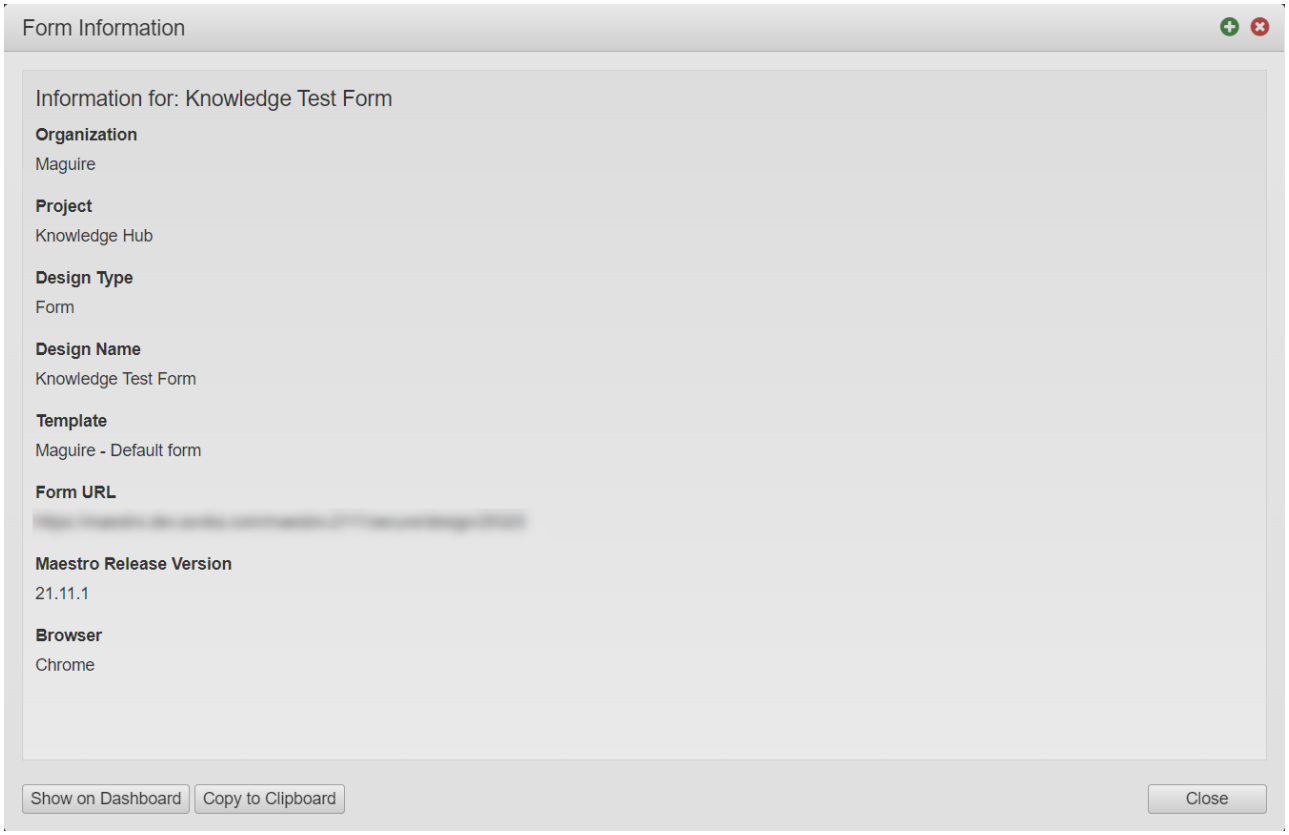
Show Form Details

MaestroThe UI design product. | Form Builder ▪ Platform Developer | All versions This feature is related to all versions.

The Maestro editor allows you to view some additional information about an open design, such as, a form, template, receipt or component.

To view a design details:

1. Open a design in the Maestro editor and click Help.
2. Select **Show Form Details** from the dropdown list to view the following design's information:



This information can be useful for troubleshooting purposes, especially when speaking with customer support.

- Organization - The organization that the design belongs to.
- Project - The project that the design belongs to.
- Design Type - The design type of the design. Possible options include Form, Component, Template, or Receipt.

- Design Name - The name of the design.
- Template - The template the design is based upon.
- Form URL - The absolute URL of the design.
- Maestro Release Version - The version number for the current instance of Maestro.
- Browser - The name of your web browser.

3. Click Show on Dashboard to locate this design in the Maestro Dashboard.
4. Click **Copy to Clipboard** to copy all of the information from the dialog to the clipboard and then use the `CTRL` + `V` shortcut (or `CMD` + `V` on macOS) to paste it somewhere else, such as, in a support ticket.
5. Click Close.

Next, learn about the Design mode.

Maestro Organizations Overview

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

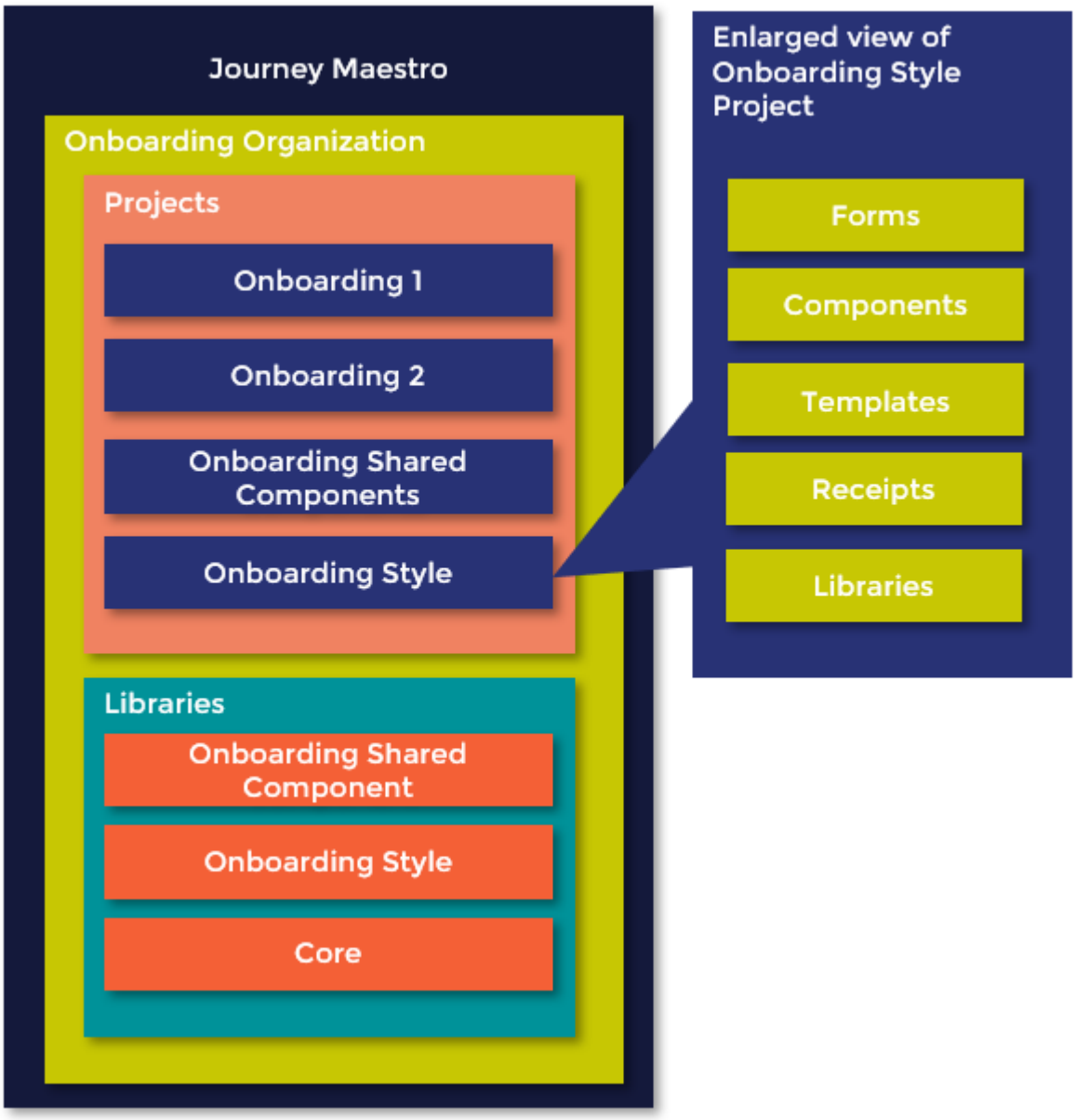
Journey Maestro uses the concept of organizations that allows you to structure your application form solutions to implement various customer requirements. An organization is the place to manage application form resources, such styles, images, and components, that your design uses. These resources fall under the following categories and are created and managed by different user roles:

- Shared components - business components shared across different forms, such as an address, contact details, owned by the Form Builder with responsibility for standardization
- Styles - corporate branding and template shared components, owned by the Template Designer
- Forms - various components, owned by the Form Builder

Organizations are the foundation of a three-tier structure including projects and forms. Each tier has an important role to play in building robust application forms. The tiers are:

- Organization - contains projects and organization level libraries
- Project - contains forms, components, templates and project level libraries
- Form - contains components to captured user data and pass it to Manager

The tires and their stricture are illustrated in the diagram below:



In this example, you can see that a customer[1] has 1 organization - the Onboarding organization - containing 4 projects and 3 libraries. It is important to note the 2 projects and 2 libraries - the Onboarding Shared Component and Onboarding Style - are solely for building and maintaining shared components and styles, which are used across all projects in the Onboarding organization. Thus, the Onboarding 1 and 2 projects contain only business specific implementation.

[1]A customer is a person who uses Temenos Journey Manager to create, host and manage application forms. Other names for a customer is a client.

Organization Structure

It's important to understand and create the correct organization structure for an efficient way to package application form resources that provides a clear separation of responsibility between Template Designers and Form Builders. This also enables forms to re-use the corporate styling without unnecessarily inheriting form specific components.

There are three types of projects:

- **Form Project** - These projects contain the forms under development but do not contain any styles or templates. Any shared components in these projects are form specific only, and not intended to be shared across projects. Over time, it is typical for there to be multiple form projects grouped by business line, form type, or geography.
- **Shared Components Project** - There should be only one shared components project per organization, which only contains shared business components to be shared across projects. This should not include template components such as header, footer, etc. Note that shared business components that are first created from a form will be added to the form project component folder by default. It is therefore necessary to move these to the shared component project by exporting and importing. Do not forget to delete from the form project.
- **Style Project** - There should be only one style project per organization, which contains the templates, images, all shared styles, and components needed by the template (e.g. header, footer, navigator). This includes shared styles needed for shared business components.

There are two types of organization libraries:

- **Shared Components Library** - This organization library contains all published shared business components. Only resources from the shared components project are published to this library.
- **Style Library** - This organization library contains all published style resources. Only resources from the style project are published to this library.

NOTE

The Shared Components Project and Style Project should have no project libraries. Their resources are always published to the organization libraries - Shared Components Library and Style Library respectfully.

NOTE

The Form Projects may have project libraries for any form specific shared components created.

Here are some recommended naming conventions for project and library names:

- Name the style project Style, unless the corporate style has a specific name.
- Name the shared component project Shared Components to highlight that this should only contain shared components.
- Prefix the style and shared component projects with the organization name or acronym so that exports are easily identified with the organization.
- Form projects and libraries should be named based on the grouping; for example, Home Loan, Customer Onboarding, US Wealth Management, etc.
- The libraries for these projects should have the same name, ending with the suffix Library.

Organizations must be created and configured in Journey Manager before you can access and use them in Maestro. Manager uses the similar concept of organizations. However, their organizations are more back-end oriented and have extensive configuration to provide a wide range of functions, such as user data storage and integration with 3rd party systems to name a few.

Usually, when a customer starts a new project, for example, an Onboarding application form, a new organization is created for this purpose either by the customer or the Cloud Services team. Also, you need to assign correct permissions to user accounts who will create, modify and delete projects, forms and other associated resources.

We recommend setting up your organization first, then creating a new template, and libraries, before you begin development forms, following the following structure.

After all this sorted out, you must create a project to begin with application form development. We recommend creating two additional projects - shared components and styles - to keep the common, reusable and shared components and styles separate from the business specific implementations. The diagram above shows these additional projects as well as two libraries containing these project artifacts. The libraries are at the organization's level meaning that every project within this organization, for example, Onboarding 1, can automatically use all resources from these libraries.

Next, learn how to view organizations.

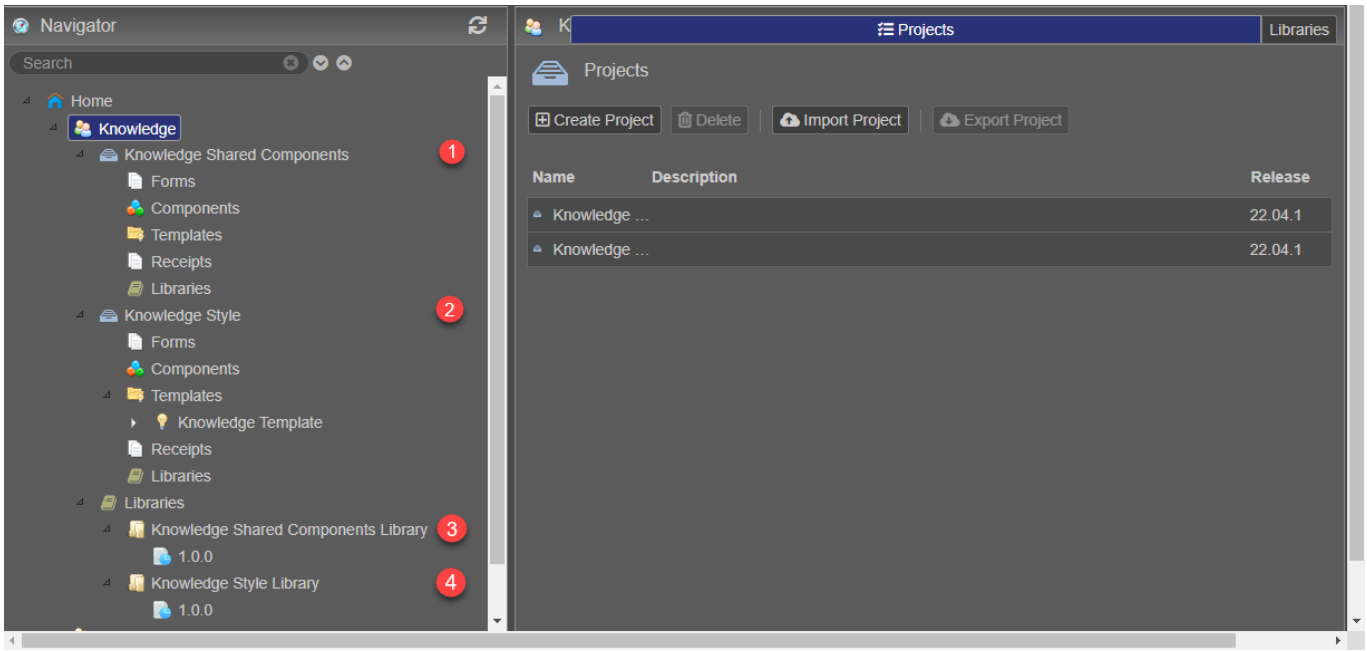
Organization Set-up

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

For a new organization or a new project in the existing organization, you must ensure that the organization is set up correctly. You have to do it only once. This step is very important as it provides many benefits in the long run. The set-up is all about creating the number of projects that constitute the structure within the organization:

- Shared Components Project - this project is used for creating and maintaining shared components
- Style Project - this project is used for template styles and all shared components' styles
- Shared Components Library - this organization-level library is used for all artefacts developed in the Shared Components project
- Style Library - this organization-level library is used for all artefacts developed in the Style project

The recommended organization structure is shown below, where the numbers correspond to the step-by-step instruction described in this topic:



This structure enables you to build and maintain application forms efficiently. Maestro doesn't enforce this structure, so it is up to you to implement the best practice and we strongly recommend doing so. For more information about organization structure and resource management, see Maestro Organizations overview.

To set up the organization structure:

Step 1: Create the Shared Components Project

1. Open your organization.
2. Create a project and name it as Your Organization Name followed by Shared Components, for example, Knowledge Shared Components.
3. Open the new project folder and select Libraries to list the default libraries, along with a custom project library.
4. Select the custom project library, for example, Knowledge Shared Components Library and click Delete.

> NOTE
>
> You do not need the project library for the Shared Components project because you will be publishing the project's artefacts to an organization library. This is a very important step as it prevents a lot of errors and confusion in regards to published shared components.

Step 2: Create the Style Project

1. Open your organization.
2. Create a project and name it as Your Organization Name followed by Style, for example, Knowledge Style.
3. Open the new project folder and select Libraries to list the default libraries, along with a custom project library.
4. Select the custom project library, for example, Knowledge Style and click Delete.

> NOTE
>
> You do not need the project library for the Style project because you will be publishing the project's artefacts to an organization library. This is a very important step as it prevents a lot of errors and confusion in regards to published shared styles.

Step 3: Create the Shared Components Library

1. Open your organization.
2. Create a library within the organization.

   NOTE

   This is the organization level library, not the project level library.

3. Name the library as Your Organization Name followed by Shared Components Library, for example, Knowledge Shared Components Library.

Step 4: Create the Style Library

1. Open your organization.
2. Create a library within the organization.

   NOTE

   This is the organization level library, not the project level library.

3. Name the library as Your Organization Name followed by Style Library, for example, Knowledge Style Library.

Now, your organization has been set up correctly. We recommend spending some time on creating templates, shared styles and components before you start creating the form projects. Forms projects are used for creating and maintaining business specific application forms and resources, which are not meant to be shared. So, if you are ready to move to the form development, go ahead and create form projects as required.

VIDEO

For more information, check the Projects Best Practices video.

Next, learn how to view organizations.

View Organizations

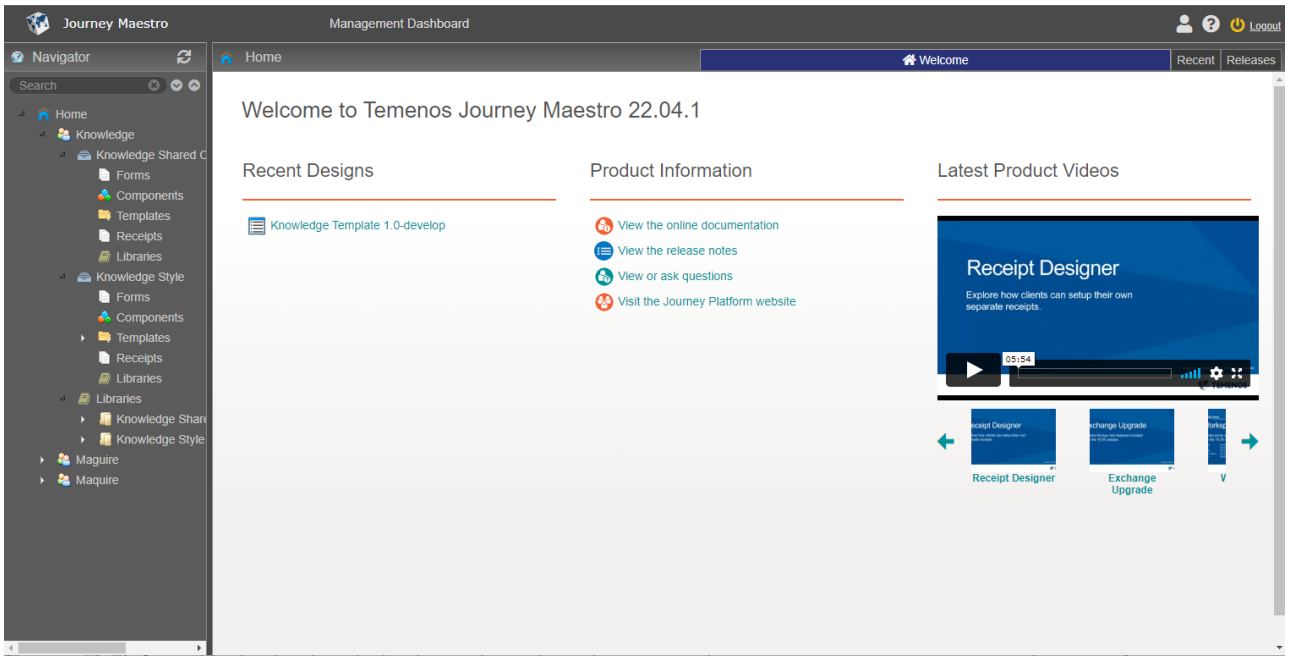
MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Maestro allows you to view the list of available organizations, so you can easily navigate to projects and their designs, such as forms, receipts, templates and components, and resources.

> NOTE
>
> Organizations are created and configured in Journey Manager so you can't change their configuration in Maestro.

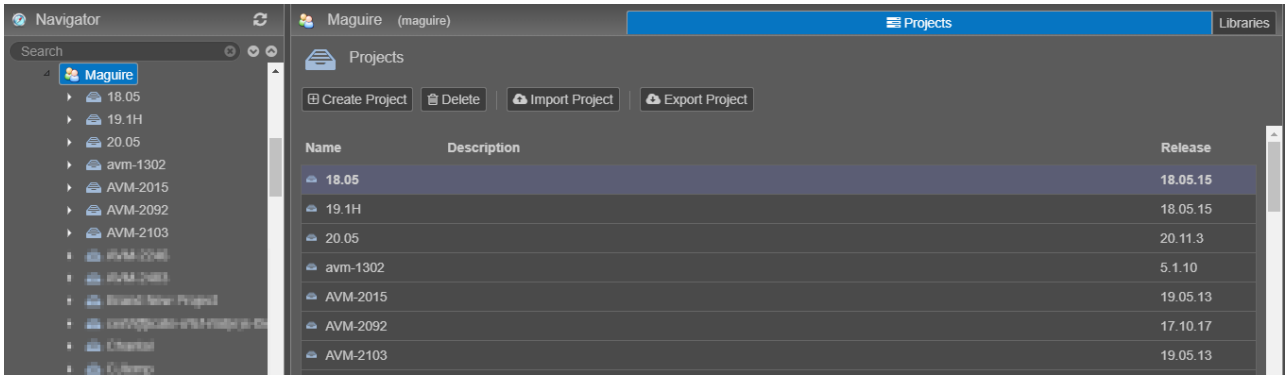
To view organizations in Maestro:

1. Log in to Maestro to access the dashboard.
2. Select Home to view the list of organizations available to you.



> NOTE
>
> Check roles and permissions assigned to your user account if you don't see organizations you require.

3. Select an organization to view and manage its assets and resources.
4. Select the Projects tab to view and manage its projects.



The projects are listed in the order they were created, with latest being on top. The list includes the following information:

- Project type that can be Standard  or SCM  . For more information, see create a project.
- Project name
- Project description
- Release version

5. Click Create Project to create a new project within this organization.
6. Click Delete to delete the selected projects.

> NOTE
>
> Deleting a project will also delete all assets and resources related to the project.

> NOTE
>
> Use `Shift` or `Ctrl` key to select multiple projects. | 21.05 This feature was introduced in 21.05.

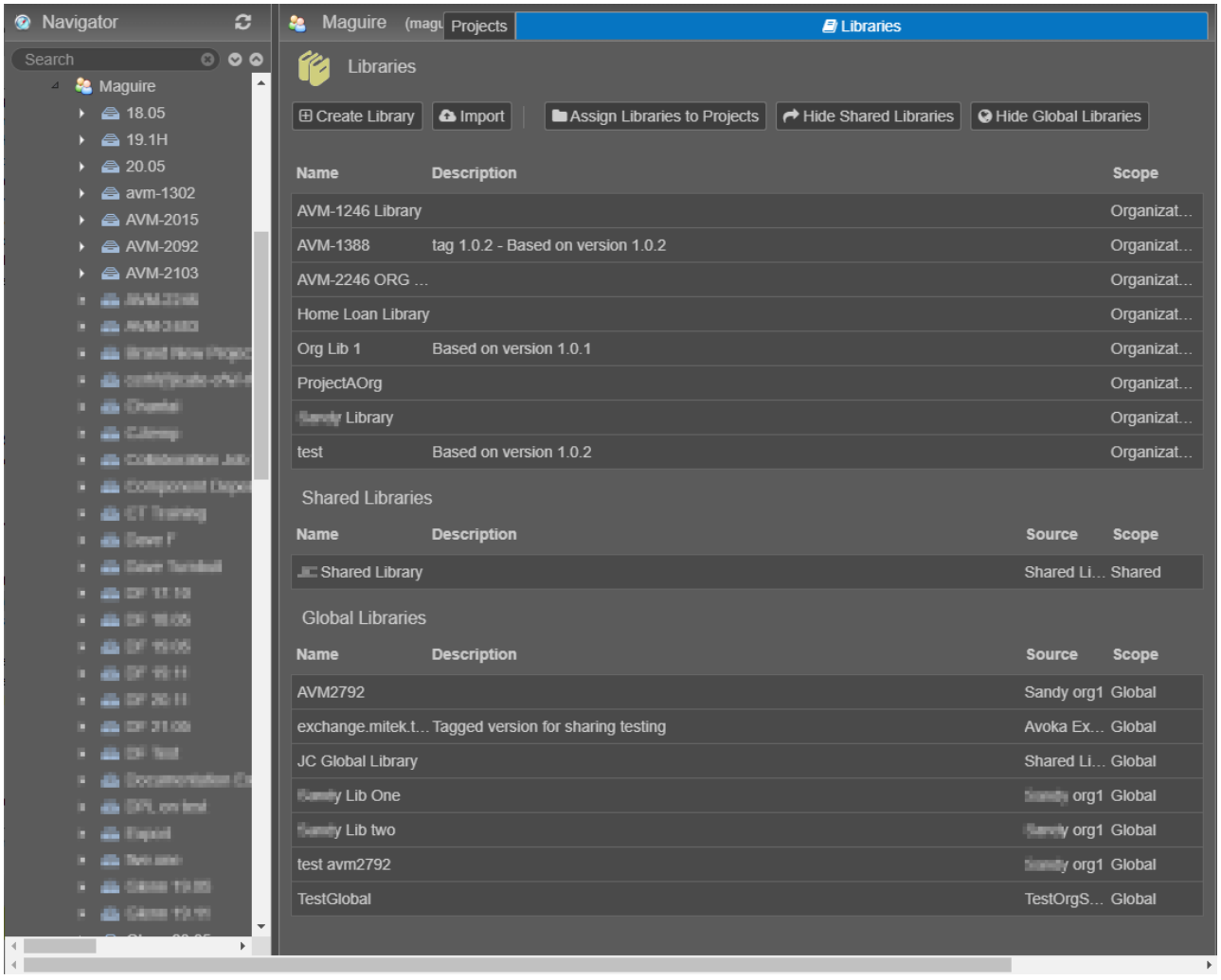
7. Click Import Project to import a project that was exported from other organization or even other Maestro environments.

> NOTE
>
> Use this functionality to move projects between organizations and environments, such as test and production, or perform a rollback or backup.

8. Click Export Project to export the selected projects so you can import them later.

> NOTE
>
> Use this functionality to move projects between organizations and environments, such as test and production, or perform a rollback or backup.

> NOTE
>
> Use `Shift` or `Ctrl` key to select multiple projects. | 21.05 This feature was introduced in 21.05.

9. Double-click a selected project to view its designs and associated resources and libraries.
10. Select the Libraries tab to view and manage libraries available to this organization.
11. 

The libraries are grouped by several categories, such as organization, shared, and global libraries. The library list includes the following information:

- Library name
- Library description
- Source - provides the additional information to distinguish the origin of a library.
- Scope - the scope can be: Organization, Shared, Global

> NOTE
>
> Libraries stored at the organization level will be available to all projects within the organization.

12. Click Create Library to create a new organization library. Organizations can have as many libraries as needed.
13. Click Import to import libraries from other projects to the organization level.

> NOTE
>
> Use this functionality to move libraries between organizations and environments, such as test and production, or perform a rollback or backup.

14. Click Assign Libraries to Projects to assign a selected library to projects. For more information, see shared libraries.
15. Click Hide Shared Libraries when you don't want to display a list of shared libraries, which is useful when this list is long and therefore clutters the view. Click Show Shared Libraries to display the list of shared libraries, after they were hidden.
16. Click Hide Global Libraries when you don't want to display a list of global libraries, which is useful when this list is long and therefore clutters the view. Click Show Global Libraries to display the list of global libraries, after they were hidden.

Next, learn how to create a project.

Projects Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro defines a project as an entity that contains forms, components, templates, receipts and project level libraries, along with all relevant resources, such as styles, images, fonts, and scripts. A projects belongs to an organization. Each organization can have many projects.

A project is the second tier in the organization structure, which enables you to configure forms and resources on the project level, and override the organization level resources, for example, a style.

All project level assets are stored in one of the following folders within the project:

- Forms - contains all available forms in the project. It is the development area for forms, from here you can create new forms or edit existing forms. Once forms are ready to be deployed, the form version needs to be built and imported into Journey Manager.
- Components - contains all editable components in this project. The components folder is the development area for components. It is from the components folder that you can create new components, share components or edit existing components. Components are not available to forms until they are published to a library.
- Templates - contains all editable templates in this project. The templates folder is the development area for templates. It is from the templates folder that the Template Designer can create new templates or edit existing templates. Templates are not available to forms until they are published to a library.
- Receipts - contains all editable receipts in this project.

You can edit assets within these project folders without affecting the published versions of these assets.

The Maestro Dashboard allows you create projects, check project configuration, as well as view project's designs. You can also perform the following actions:

- Rename a project
- Update description for a project
- Select the default template for a project
- Enable automatic release upgrades for a project
- Export a project
- Import a project

If you are about to create the first business project in a new organization, you must set up the organization correctly beforehand. This ensures that all next projects your create will make use of the existing organization's resources. This promotes reuse of shared styles and components as well as other benefits. After that, what you need to do is simply create a new form project as per business requirements.

Next, learn how to create a project.

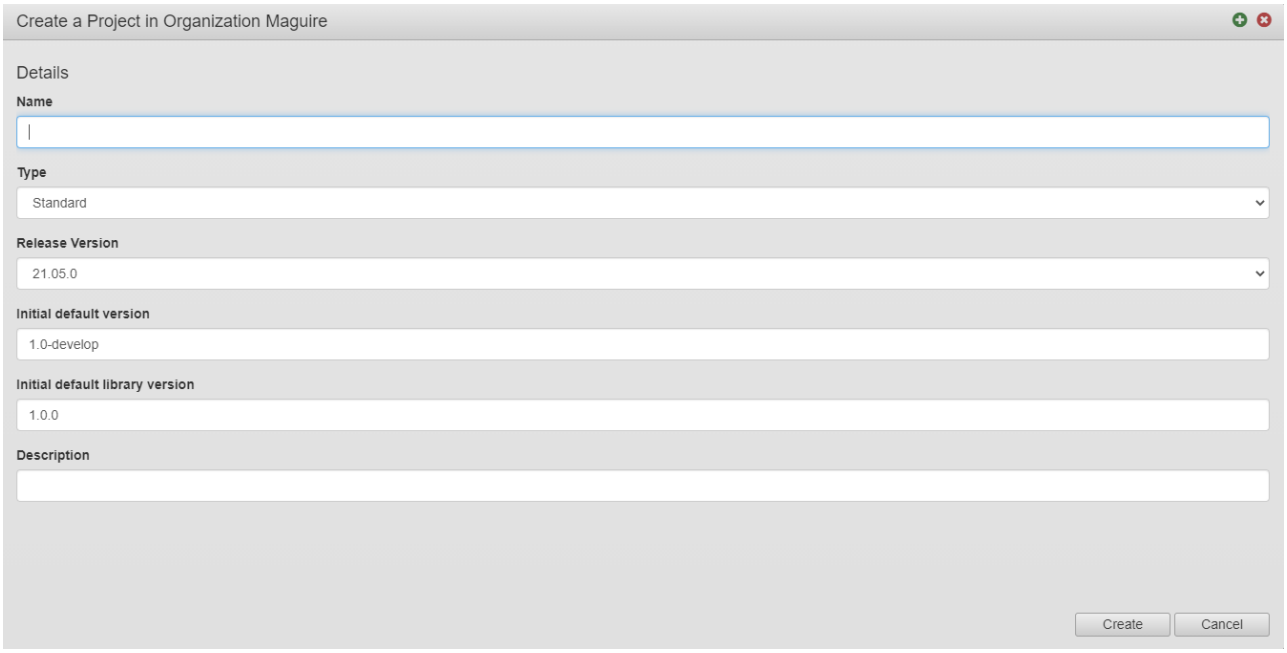
Create a Project

MaestroThe UI design product. | Form Builder | 17.10 This feature was updated in 17.10.

Journey Maestro allows you to create projects within a organization so you can build a complex structure to manage application forms efficiently.

To create a project:

1. Open a project in the Navigator pane and select an organization, where you want to create a new project.
2. Click Create Project.



3. Enter a project's name in the Name field, which should be unique across the organization. We recommend naming a form project based on the type of forms being produced. For example, Commercial Loans.
4. Select a project type from the Type dropdown list. | 17.10 This feature was introduced in 17.10.
   - Standard - a source code of the project is managed by Maestro
   - SCM - a source code of the project is managed externally using various SCM, such as Git[1]. Use this project type if collaborative work on a project is required.

[1]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.

NOTE

The SCM project type doesn't have any impact on the content of forms. It only affects the versioning options shown in the Management Dashboard.

NOTE

If you are a Form Builder, you need to have the SCM permissions assigned to you.

5. Provide a release version in the Release Version field. By default, the release version is automatically set to the latest minor release of Maestro, but you can select an earlier release version if required.
6. Enter an initial default version and its suffix in the Initial Default Version field, which is used as the initial version for all assets created within this project. If you enter incorrect version value, the Create button will be disabled. This field is optional. For more information, see semantic versioning and tagging.
7. Provide an initial default library version value in the Initial Default Library Version field. When you create a project, the default project library is set up as well. The initial default library version define a version number for all new libraries. This makes it easier to roll back to the previous library version or set up and make changes to the new versions without affecting an organization and projects using older versions.
8. Type a meaningful description for the project in the Description field.
9. Click Create.

The new project will appear under the selected organization and will include default forms, components, and libraries, with the initial library version as you have selected above, in this example, 1.0.0.

VIDEO

For more information, check the Create a New Project video.

Next, learn how to upgrade a project.

Upgrade a Project

MaestroThe UI design product. | Form Builder | 22.10 This feature was introduced in 22.10.

When a Journey Maestro environment is upgraded, you can choose whether projects continue using the original software release or have to be upgraded to a desired release, for example, the latest Maestro release. By default, all projects remain with their original version until the project's release version is manually changed in the Project Details.

We strongly recommend using the latest Maestro version as it provides you with the following benefits:

- Projects and forms are built and deployed using the latest libraries, including core libraries
- Forms are free of latest vulnerabilities and other defects related to unpatched libraries
- Form developers use the latest Maestro editor's features that benefits their productivity

When you open a form, component or template, Maestro checks what release version a project is using. The project version determines what version of Maestro and libraries are available to designs.

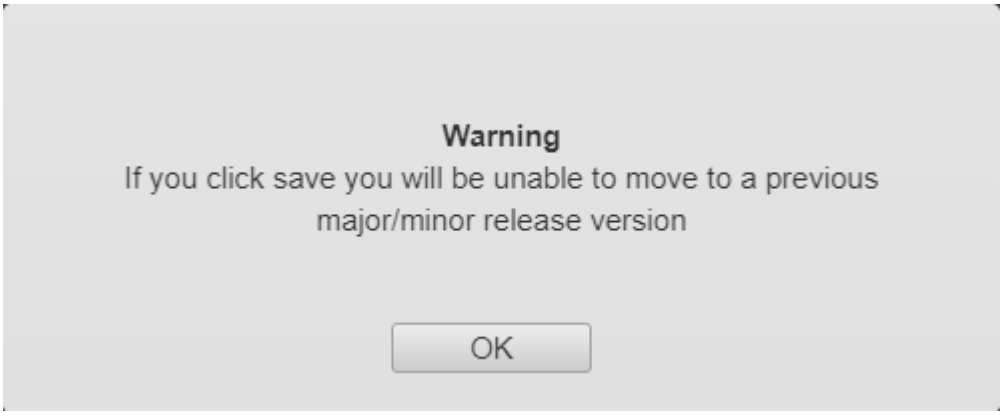
When the product is upgraded to a later version, Maestro does not force you to upgrade the version, thus providing seamless version compatibility. This allows companies to continue working on the version they were using prior to the upgrade, ensuring there is no change in behavior to their forms.

By default, all Maestro projects are automatically updated with each minor release of Maestro. When creating a new project, the Automatic Release Upgrades dropdown is set to Enabled by default. However, this feature only applies to minor releases. If Maestro is upgraded with a major release, projects always need to be manually upgraded, as descibed below. Also, if you've disabled this feature, you'll need to update your projects manually after each Maestro upgrade.

To update a project manually:

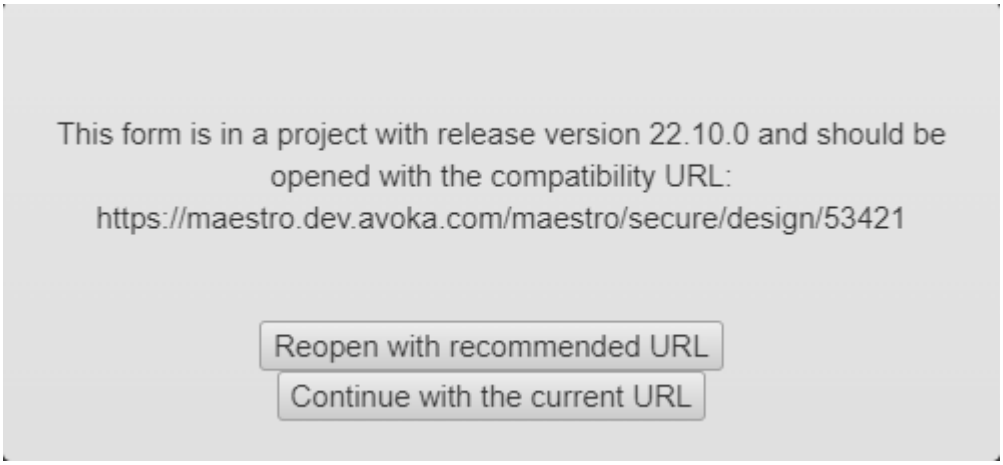
1. Open a project in the Navigator pane and select the Project Details tab.
2. Select a required version of Maestro from the Release Version dropdown list, for example, 22.04.0.

3. Click OK to confirm it.



> WARNING
>
> Once you have upgraded a project, you can't return to its previous version. Therefore, we strongly recommend backing up the project using Maestro's export functionality before upgrading the project. This allows you to restore the project using Maestro's import.

4. Click Save.
5. Open a form to check it works with the new version of Maestro. and review the available options to open the form:



   - Click Reopen with recommended URL to open the form using the new URL.
   - Click Continue with the current URL to open the form using the current, prio to the upgrade, URL.
6. Rebuild and test each of the forms and artifacts in the project.

Next, learn how to export a project.

Export a Project

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro allows you to export projects, so you can easily move or copy them into different organizations or environments.

NOTE

When you export a project, the export includes all the assets and resources, such as original templates, components, styling, and so on, contained in that project. Therefore, we recommend speaking to the Template Designer before exporting projects as they are responsible for the management of the templates and styles within the projects.

To export a project:

1. Open an organization in the Navigator pane and select the project.
2. Click Export Project.

NOTE

Also check how to export a Maestro asset.

Next, learn how to import a project.

Import a Project

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

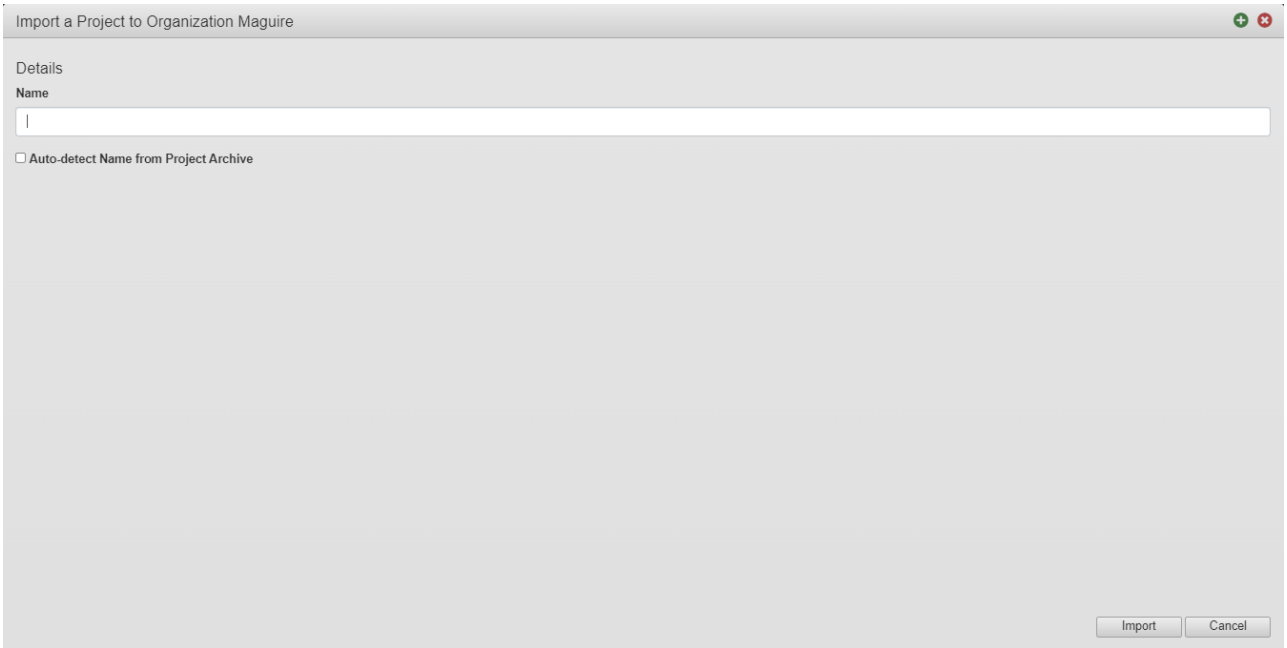
Journey Maestro allows you to import projects, which you can use to move or copy a project to a different organization.

NOTE

We recommend speaking to the Template Designer before importing projects as they are responsible for the management of the templates and styles within the projects.

To import a project:

1. Open an organization in the Navigator pane where you want to import the project to.
2. Click Import Project..



3. Enter a new project name in the Name field or select the Auto-detect Name from Project Archive checkbox to populate the name from the exported project as the archive ZIP file
4. Click Import.
5. Choose the exported project as the archive ZIP file and click Open.

NOTE

Also check how to import a Maestro asset.

Next, learn how to view project details.

View Project Details

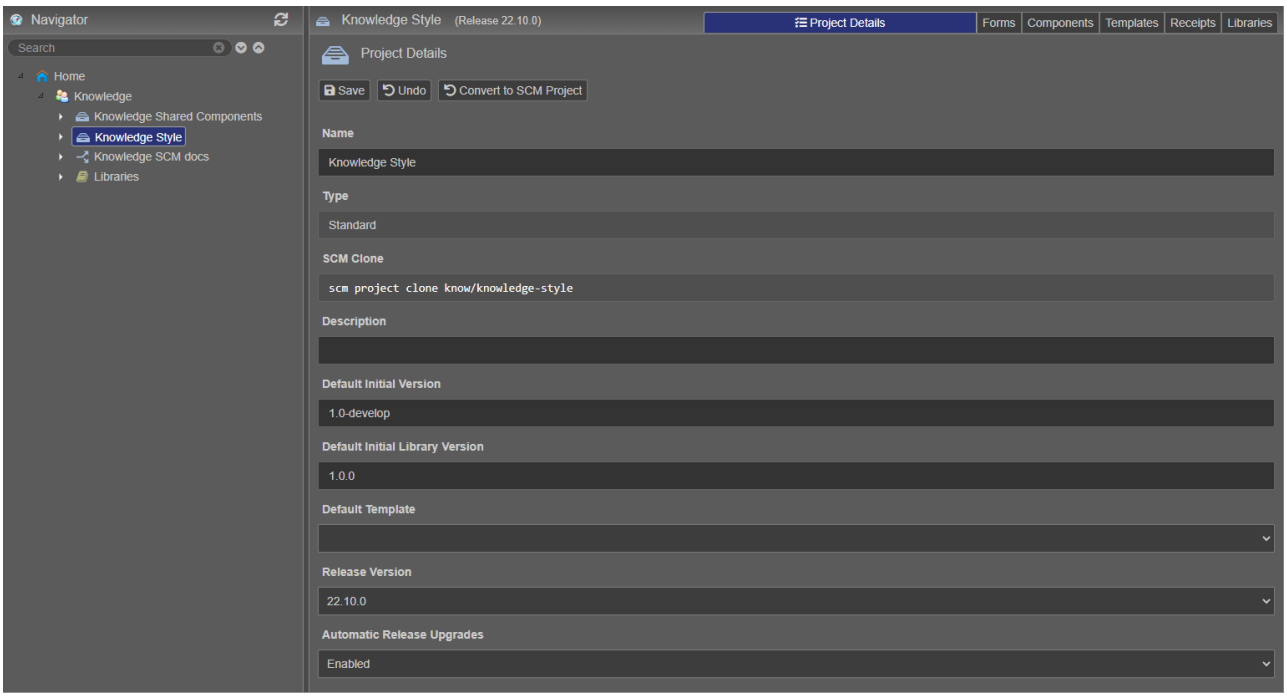
MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Journey Maestro allows you to view project details as well as update some of its configuration. You may need to do this after the project is created. For example, you may want upgrade your project to the latest Maestro's release version so you can start using the latest product's features.

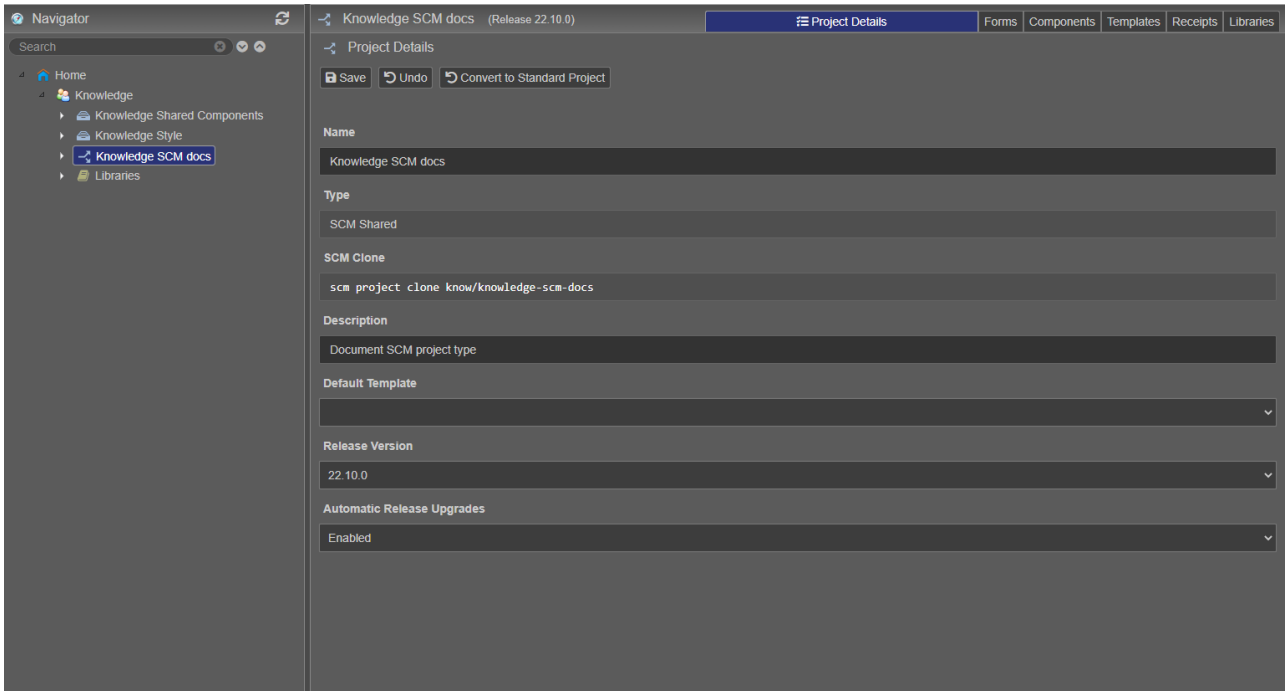
To view the project details:

1. Open a project in the Navigator pane and select the Project Details tab to view project configuration. The configuration is slightly different for the Standard and SCM projects as shown below:

   Standard Project



   SCM Project



2. Check the current Maestro version used by the project. In this example, it is Release 22.10.0.
3. Edit a project name in the Name field.
4. Check a project type in the Type field, which can be one of the following:
   - Standard - a source code of the project is managed by Maestro
   - SCM - a source code of the project is managed externally using various SCM, such as Git[1]. Use this project type if collaborative work on a project is required.

> NOTE
>
> The SCM project type doesn't have any impact on the content of forms. It only affects the versioning options shown in the Management Dashboard.

> NOTE
>
> If you are a Form Builder, you need to have the SCM permissions assigned to you.

[1]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.

5. Copy a command from the SCM Clone field and copy it to your SCM tool to clone the project, for example, `scm project clone know/knowledge-style`.
6. Provide a meaningful project description in the Description field.
7. Select a version from the Default Initial Version dropdown list, which is used as a base version when new forms and form versions are created. You can change this value at any time, but this only impacts new forms and assets within this project. Version numbers of existing forms are unaffected.
8. Select a template from the Default Template dropdown list, for example, Avalon Form Template.
9. Select a Maestro version from the Release Version dropdown list to upgrade the project manually. You have to do it when the Automatic Release Upgrade setting is set to Disabled, see below, or then Maestro is upgraded to a major version.
10. Select one of the following options from the Automatic Release Upgrade dropdown list:
    - Enabled - to upgrade the project automatically, meaning whenever Maestro software is upgraded to a new, minor release, the project is automatically upgraded as well, for example, from 21.10.2 to 21.10.3. However, this feature only applies to minor releases. If Maestro is upgraded with a major release, projects always need to be manually upgraded. This is the default option and we recommend using it.
    - Disabled - to upgrade the project manually.
11. Click Save to update the changes.
12. Click Undo to discard the changes.
13. Click Convert to SCM Project to convert the project from the Standard type to the SCM type so you are able to manage its source code using your Source Code Management tools.
14. Click Convert to Standard Project to convert the project from the SCM type to the Standard type so you can use Maestro functionality to manage versioning.

Next, learn how to view project's assets and resources

View Project Assets and Resources

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Journey Maestro allows you to view and manage projects and their assets, such as forms, components, templates, and receipts, and resources, such as libraries.

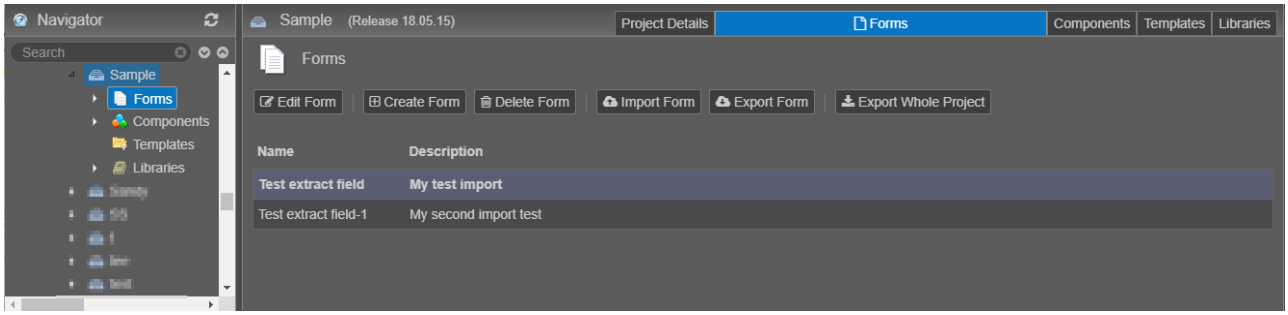
NOTE

Forms, components, templates, and receipts are all Design type categories that you can access from a project.

Forms

To view the list of forms in an project:

1. Open a project in the Navigator pane.
2. Select the Forms tab to view and manage a list of forms belonging to the project.



   The list includes the following information:

   - Name - a form name
   - Description
3. Click Edit Form to edit a selected form.
4. Click Create Form to create a new form.
5. Click Delete Form to delete selected forms.

   NOTE

   Use `Shift` or `Ctrl` key to select multiple forms. | 21.05 This feature was introduced in 21.05.

6. Click Import Form to import a form.

7. Click Export Form to export selected forms.

   NOTE

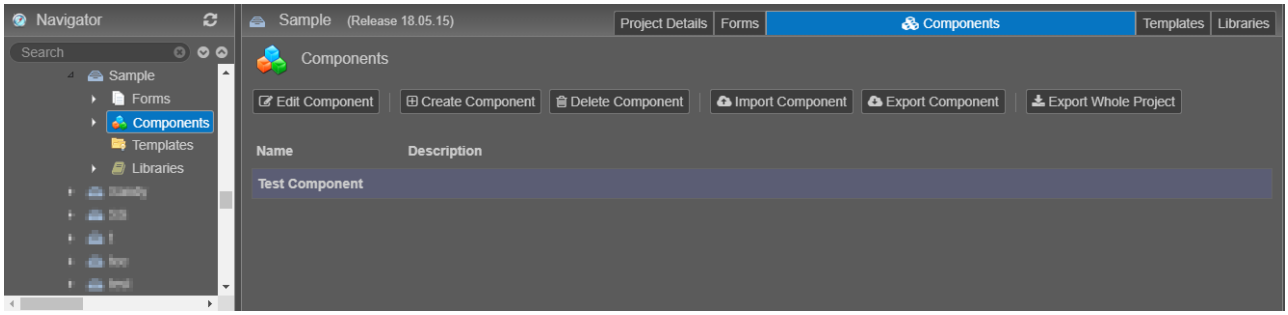
   Use `Shift` or `Ctrl` key to select multiple forms. | 21.05 This feature was introduced in 21.05.

8. Click Export Whole Project to export the project including all assets and resources.
9. Double-click a selected form to view it.

Components

To view the list of components in an project:

1. Open a project in the Navigator pane.
2. Select the Components tab to view and manage a list of components belonging to the project.



   The list includes the following information:

   - Name - a component name
   - Description

3. Click Edit Component to edit a selected component.
4. Click Create Component to create a new component.
5. Click Delete Component to delete selected components.

   NOTE

   Use `Shift` or `Ctrl` key to select multiple components. | 21.05 This feature was introduced in 21.05.

6. Click Import Component to import a component.

7. Click Export Component to export selected components.

   NOTE

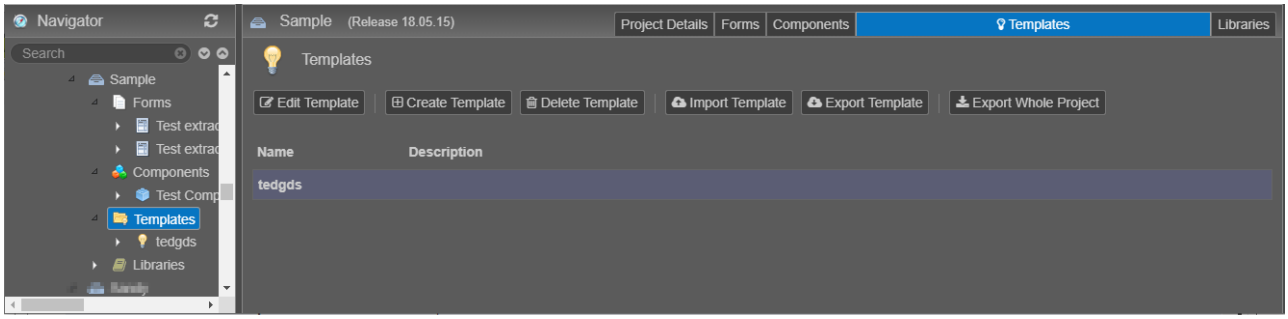
   Use `Shift` or `Ctrl` key to select multiple components. | 21.05 This feature was introduced in 21.05.

8. Click Export Whole Project to export the project including all assets and resources.
9. Double-click a selected component to view it.

Templates

To view the list of templates in an project:

1. Open a project in the Navigator pane.
2. Select the Templates tab to view and manage a list of templates belonging to the project.



   The list includes the following information:

   - Name - a template name
   - Description

3. Click Edit Template to edit a selected template.
4. Click Create Template to create a new template.
5. Click Delete Template to delete selected templates.

   NOTE

   Use `Shift` or `Ctrl` key to select multiple templates. | 21.05 This feature was introduced in 21.05.

6. Click Import Template to import a template.

7. Click Export Template to export selected templates.

   NOTE

   Use `Shift` or `Ctrl` key to select multiple templates. | 21.05 This feature was introduced in 21.05.

8. Click Export Whole Project to export the project including all assets and resources.
9. Double-click a selected template to view it.

NOTE

To check project's details and configuration, click Project Details.

Next, learn how to view project's libraries.

Maestromo Libraries Overview

MaestroThe UI design product. | Form Builder | 18.05 This feature was updated in 18.05.

Journey Maestro uses the concept of libraries, where a library is a collection of resources that can be made available to Maestro's forms, components, and templates. Resources can be:

- Images
- Styles
- Fonts
- Scripts
- Components
- Templates
- Languages
- Shared Translations
- Domain Models

Maestro defines resources as stand-alone objects contained by assets like shared components and templates. Resources are identified by their name and by their values, if there are multiple resources with the same name. The version of the resource that is 'live' and shared across forms and templates is the last version to be published to a library (taking into account precedence rules).

You can create libraries for organizations or projects and add (publish) resources to these libraries. This allows anyone with access to these organizations or projects to share and reuse the resources. Maestro comes with the editor that is easy to use when you want to view and manage resources in a library.

The text-based resources, such as styles and scripts, can contain either Windows (`CRLF` or `\r\n`) or UNIX (`LF` or `\n`) new line characters. Importing a resource with one line ending format over the other won't cause any issues.

> NOTE
>
> A resource must be published to a library to make it available to forms, components, or templates that use that library.

We encourage Template Designers and Form Builders to use libraries to avoid resource duplication, streamline resource reuse, and promote changes to multiple forms in a single action thereby enforcing a consistent user experience.

To emphasize the importance of libraries, consider the following use cases.

Use Case 1: Template Designer

As a Template Designer, you want to ensure that corporate branding is consistent across a project or organization. If the resources that implement your brand are used in many different places, it's easy for those resources to diverge as branding guidelines evolve.

To solve this problem, a Template Designer can create a library named "Corporate Brand", for example, containing resources that implement the corporate brand, and make that library available to the organization. This library acts as a single source of truth for corporate branding. All templates, forms, and components within that organization use the resources in the "Corporate Brand" library, thereby ensuring compliance with branding guidelines. If the guidelines change, the Template Designer need only publish the changes to the library for them to be reflected throughout the organization.

Use Case 2: Form Builder

As a Form Builder, you want to ensure that form elements are laid out correctly, behave correctly across multiple device types, and accurately validate a range of data types. If an organization has a number of forms that ask for the customer's address, for example, it makes sense for the form fields that ask for the address to always look and behave the same way. If they don't, the inconsistency can lead to a frustrating user experience.

To solve this problem, a Form Builder can create a range of components that behave as desired and save those components to a common library. This library can be attached to an organization or project, and if those components ever need to be updated, the changes only have to be made once in the library rather than everywhere the components are used.

Next, learn how to view libraries.

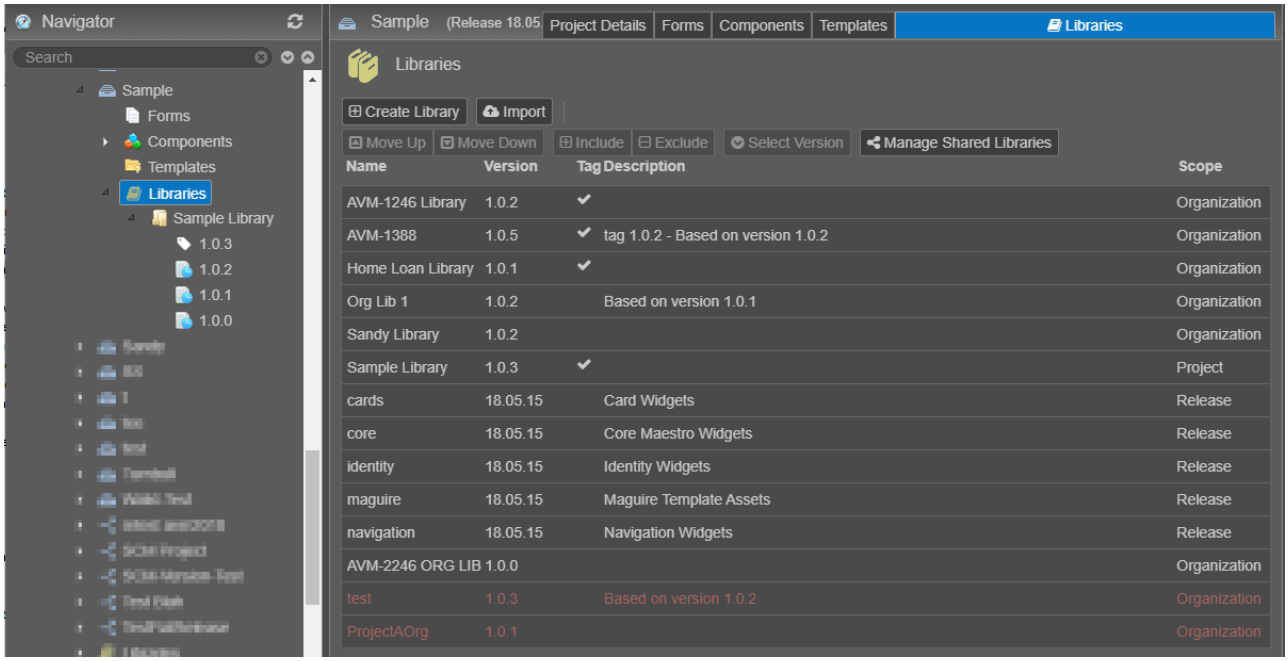
View Libraries

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Journey Maestro allows you to view and manage libraries that you can use to build your forms.

To view and manage libraries:

1. Open your project in the Navigator pane and click Libraries to see the list of libraries available in your project. You can also click your project and select the Library tab.



   The list of libraries is ordered based on library precedence and includes the following information, which helps you use libraries correctly:

   - Library name -the name is not unique across different organizations
   - Library version
   - Whether a library is tagged or not
   - Library scope
2. Click Create Library to create a new library.
3. Click Import to import a library.

4. Click Move Up to move the selected library up in the desired order of precedence.

   NOTE

   Use drag and drop to rearrange the order of libraries. | 20.11 This feature was introduced in 20.11.

5. Click Move Down to move the selected library down in the desired order of precedence.

   NOTE

   Use drag and drop to rearrange the order of libraries. | 20.11 This feature was introduced in 20.11.

6. Click Include to include selected libraries to your project and forms. You have to do it to reinstate excluded libraries, see below. The included libraries remain at the bottom of the list, so you may need to move it up to give it the desired precedence.

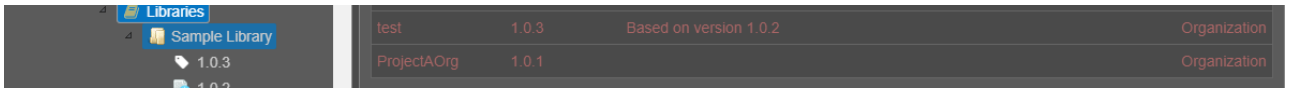
   NOTE

   Use `Shift` or `Ctrl` key to select multiple libraries. | 21.05 This feature was introduced in 21.05.

7. Click Exclude to make selected libraries unavailable to your project and forms. By default, all organization libraries are available to all projects in the organization. However, for some projects, you may not want some libraries to be available and, thus, need to exclude them. The excluded libraries are moved to the bottom of the library list and displayed with a red font to indicate that the library and its resources are not available to the project.

   NOTE

   Use `Shift` or `Ctrl` key to select multiple libraries. | 21.05 This feature was introduced in 21.05.



8. Click Select Version to chose a library version to use in your project, for example, change from 1.0.1 to 1.0.2.

9. Click Manage Shared Libraries to manage selected libraries.

> NOTE
>
> Use `Shift` or `Ctrl` key to select multiple libraries. | 21.05 This feature was introduced in 21.05.

Next, learn how to view library versions.

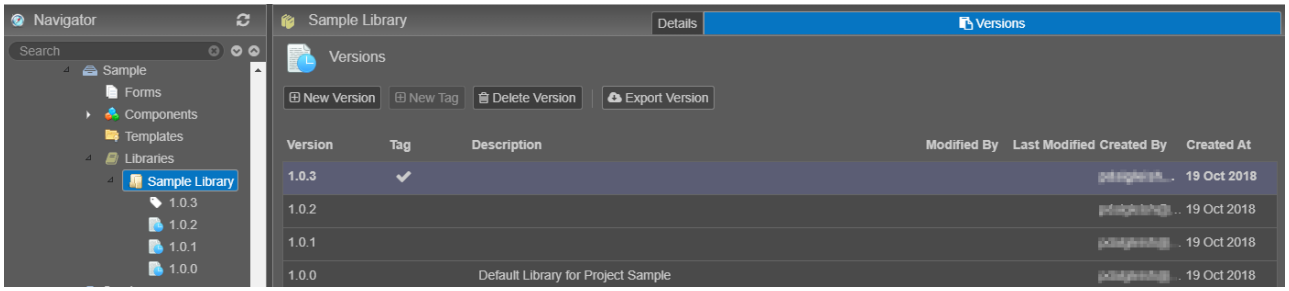
View Library Versions

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Journey Maestro allows you to view and manage library versions that you can use to build your forms.

To view the list of library versions:

1. Open your project in the Navigator pane and click Libraries to view the list of libraries available in your project.
2. Select a project library in the Libraries folder to view available versions of this library.



   The libraries are listed in ascending order based on library version. The list includes the following information, which helps you use libraries correctly:

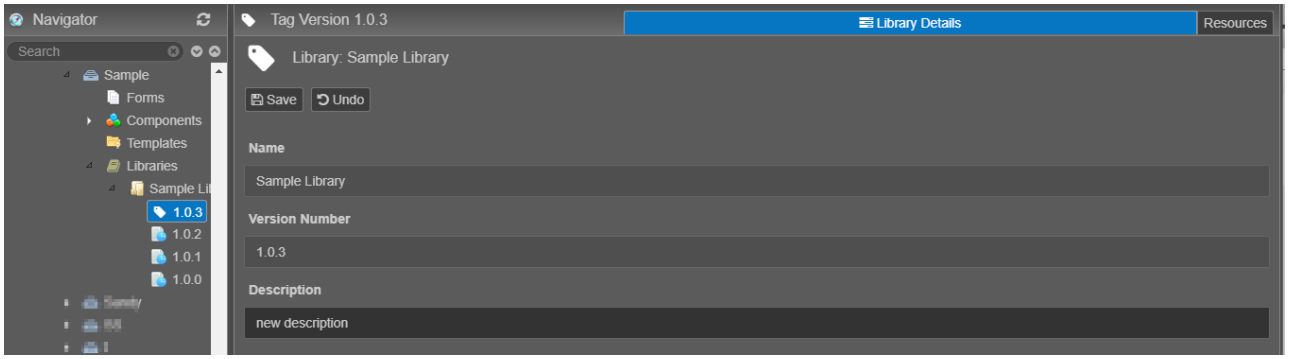
   - Library version
   - Whether a library version is tagged or not
   - Library version description
   - Who modified and created a library version
   - When a library version was created
3. Click New Version to create a new version of a selected library.
4. Click New Tag to tag a selected library version.
5. Click Delete Version to delete the selected libraries.

   > NOTE
   >
   > Use `Shift` or `Ctrl` key to select multiple libraries. | 21.05 This feature was introduced in 21.05.

6. Click Export Version to export the selected libraries.

> NOTE
>
> Use `Shift` or `Ctrl` key to select multiple libraries. | 21.05 This feature was introduced in 21.05.

7. Double-click a selected library version to view the library's details where you can update the **Description** field. Click Save to update the changes.



> NOTE
>
> You can't update the name and version number of the library.

8. Select the Resources tab to see the list of resources published into the library.

As you can see, there are a few things you can do to manage libraries. Here's more information on how to use organization libraries.

Next, learn about library precedence.

Library Precedence

MaestroThe UI design product. | Form Builder | 20.11 This feature was updated in 20.11.

When you select a project's Libraries folder, you'll see the list of libraries available to that project. By default, the libraries will be sorted from newest (most recently added to the project) to oldest, with the release libraries at the bottom and user-created libraries at the top. If a new library becomes available to the project, it will be added to the top of the list. This is significant because the order of this list can affect the way your forms look and behave.

Name	Version	Description	Scope
Credit Card Library	1.0.0	Default Library for Project Credit Card	Project
Training Styles Library	1.0.0	A library to publish Training styles.	Organization
exchange.mitek.tiden	1.0.0	Mitek Tiden	Organization
Avoka Library	1.0.0		Organization
cards	18.05.3	Card Widgets	Release
core	18.05.3	Core Maestro Widgets	Release
identity	18.05.3	Identity Widgets	Release
maguire	18.05.3	Maguire Template Assets	Release
navigation	18.05.3	Navigation Widgets	Release

When a form needs a resource, it will look for it by name in the project's libraries and use the first one it finds with the specified name.

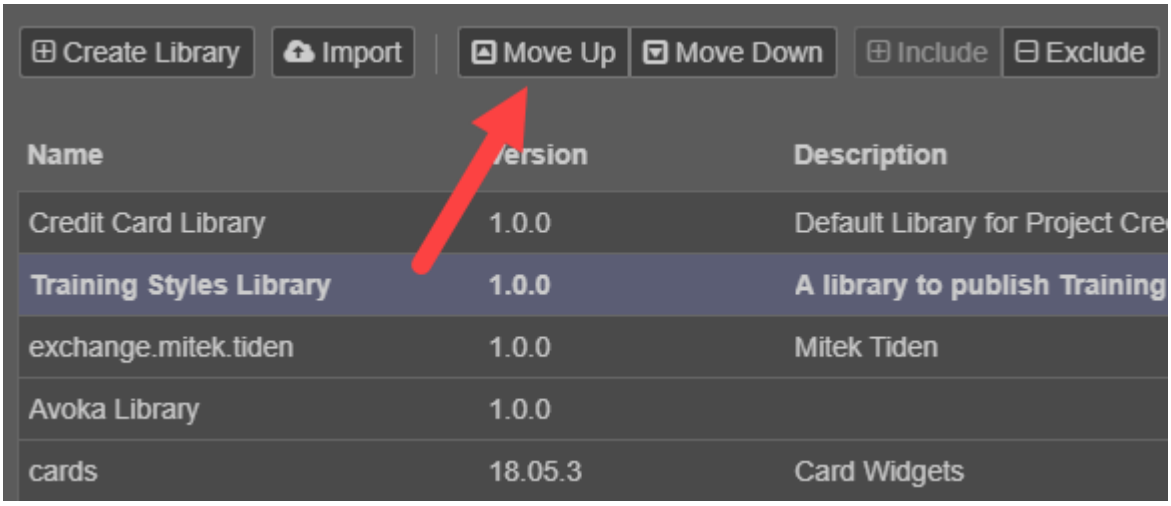
Here's an example:

Imagine there's a style in one of the default libraries that you want to override. If you create your own library with a rule that overrides the style, the rule you've defined will take precedence over the original one. But what happens if you create a second library that also overrides the same rule? Which rule will be used by the form?

The rule inside the most recently created library will be used because libraries are given precedence based on their order in the library list and the most recently created library will be at the top of the list.

In other words, **libraries higher in the library list are considered more important than libraries lower in the list**. If you want the rule from the second library to not take precedence over the first library, you would need to move it further down the list.

To control the precedence of libraries, select a library and use the Move Up and Move Down buttons to adjust its position in the list.



NOTE

The order of the release libraries should not be changed as this may lead to unexpected results.

For similar reasons, it is strongly recommended that you do not change any resources at the form level. Form level resource changes override all library resources, so any form resource change can lead to an inconsistent user experience.

You can publish a shared component in to two different libraries. In this case, the version of the component in the library with higher precedence will be used. This is the library higher up in the list of libraries for your project.

Next, learn about library scope.

Library Scopes

MaestroThe UI design product. | Form Builder | 20.11 This feature was updated in 20.11.

Each Maestro library has a scope that determines which forms have access to that library's resources. Maestro supports the following scopes:

- Release -the libraries are locked down and tied to a specific release. These libraries are global libraries available across the whole Maestro environment.
- Organization - the libraries are tied to an organization and available to its projects.
- Project - the libraries are tired to one project only.
- Shared organization- the organization libraries that can be shared with other organizations. They can even be shared across all organizations in your environment making these libraries global. | 20.11 This feature was introduced in 20.11.

To view a library's scope, navigate to a Maestro project's Libraries tab and look at the Scope column:

Name	Version	Description	Scope
Credit Card Library	1.0.0	Default Library for Project Credit Card	Project
Training Styles Library	1.0.0	A library to publish Training styles.	Organization
exchange.mitek.tiden	1.0.0	Mitek Tiden	Organization
Avoka Library	1.0.0		Organization
cards	18.05.3	Card Widgets	Release
core	18.05.3	Core Maestro Widgets	Release
identity	18.05.3	Identity Widgets	Release
maguire	18.05.3	Maguire Template Assets	Release
navigation	18.05.3	Navigation Widgets	Release

NOTE

If you navigate to the Libraries tab of an organization, you will only see the organization-level libraries.

A **release library** is a built-in library for a specific Maestro release that is always available to your Maestro projects. The resources available in a release library are also backwards-compatible with earlier Maestro releases. Release libraries include:

- Cards
- Core
- Identity
- Maguire
- Navigation

These libraries contain the standard Maestro resources that need to be accessible to every form. They cannot be removed or modified and, because of library precedence, should not be moved out of their original order.

An **organization library** is a user-created library that is attached to an organization. Any projects and forms within the organization will have access to the resources in these libraries. By default, there are no organization libraries.

A **project library** is a library attached to a project. Any forms within a project can access the resources in it's project libraries. When a project is created, a project library is created for it automatically. You can create additional project libraries as you need them.

> NOTE
>
> A project library is visible only to the project in which it was created. Project libraries are never visible under an organization's Libraries folder.

There is no limit to the number of organization and project libraries you can have. However, there are some important considerations that may help you to decide how many libraries you need and what scope they should have:

1. What is the best way to group your assets and resources into separate libraries?
2. Should a library be attached to an organization or a project?
3. Will having too many (or too few) libraries affect their manageability?

Next, learn about shared libraries.

Shared Organization Library

MaestroThe UI design product. | Form Builder | 20.11 This feature was introduced in 20.11.

Journey Maestro allows you to create and use libraries containing forms resources, such as components, within a single or across multiple Maestro environments. Each library has a scope or a library level that determines which forms have access to that library's resources. Maestro always has 3 library levels - release, organization, and project libraries.

Maestro version 20.11 has introduced a concept of shared organization libraries, which are centralized organization libraries that can be shared across other organizations. They can even be shared across all organizations in an environment making these libraries global. The shared library is then used by projects within these organizations. From the client's source project, a shared library is similar to a release library with a fine grained control over where this library is shared and used. One of the main advantages of a shared organization library is that you can develop a new library in your own specific organization, make some changes to it and push the changes to other organizations and projects where the library is used.

Shared organization libraries can be used as:

Shared

One client that has multiple organizations wants to share assets across one or more of those organizations. Your company has different business units that each use a different Maestro organization. You are developing and maintaining common components that can be used by all business units.

Global

A site administrator wants to share assets globally across all organizations. You run a Maestro multi-tenanted environment and want to provide your clients with common templates that are centrally maintained.

Shared organization libraries provide the following functionality:

- Share a tagged organization library with one or several other organizations
- Share a tagged organization library globally
- Allow organizations to manage the use of shared libraries in their projects
- Allow projects to manage the use of shared libraries in their project library list
- Restrict removal of libraries that are in use

- Add new permission levels to share and un-share libraries. If you have access to an organization that the library has been shared with, then you can access the library when the form editor loads.

NOTE

When you can use the offline build tool, which supports non SCM projects, with shared and global organization libraries, a client will download them and use them within the build. Thus, sharing libraries needs to be carefully considered, so people don't get libraries that shouldn't have been shared with them.

How to Share Organization Libraries

You can use a separate organization where you develop libraries you want to share. Then, you tag them and share as required.

NOTE

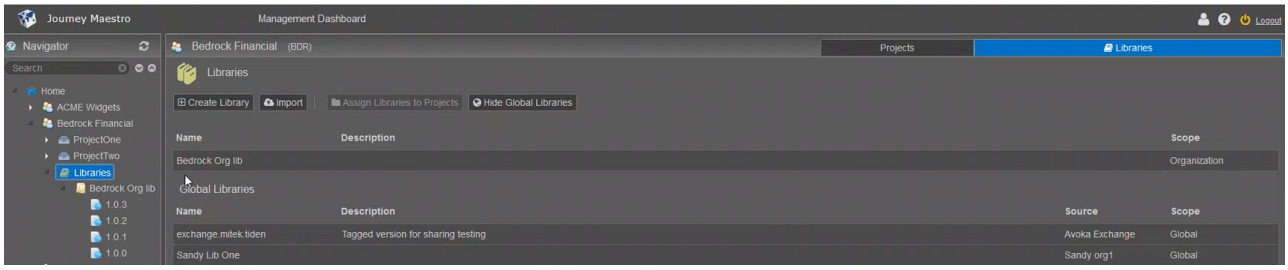
You can share only tagged read-only organization libraries.

NOTE

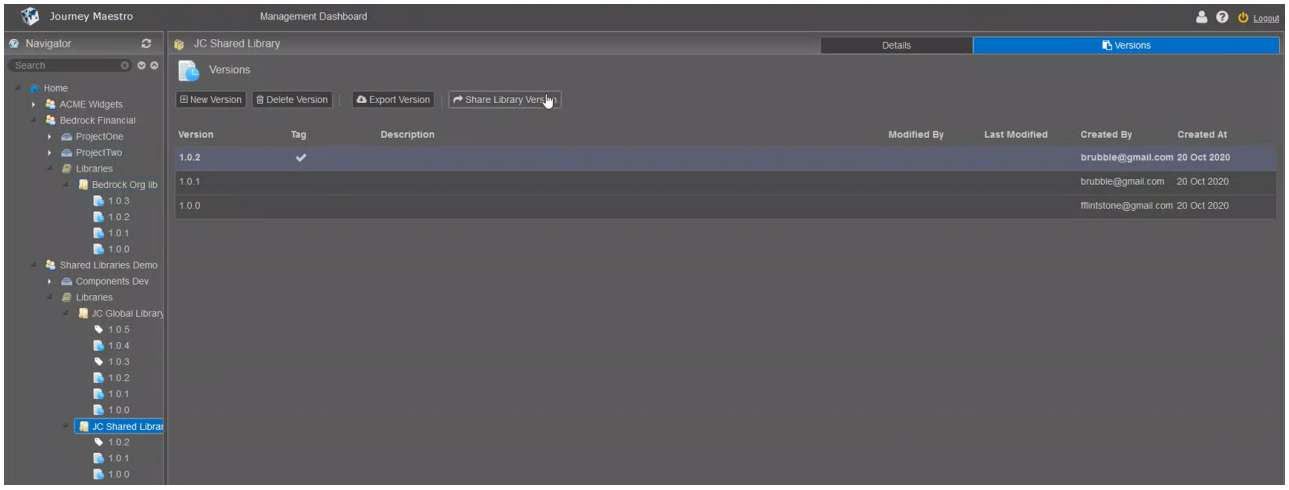
You can't share a library with the source client.

To share a library:

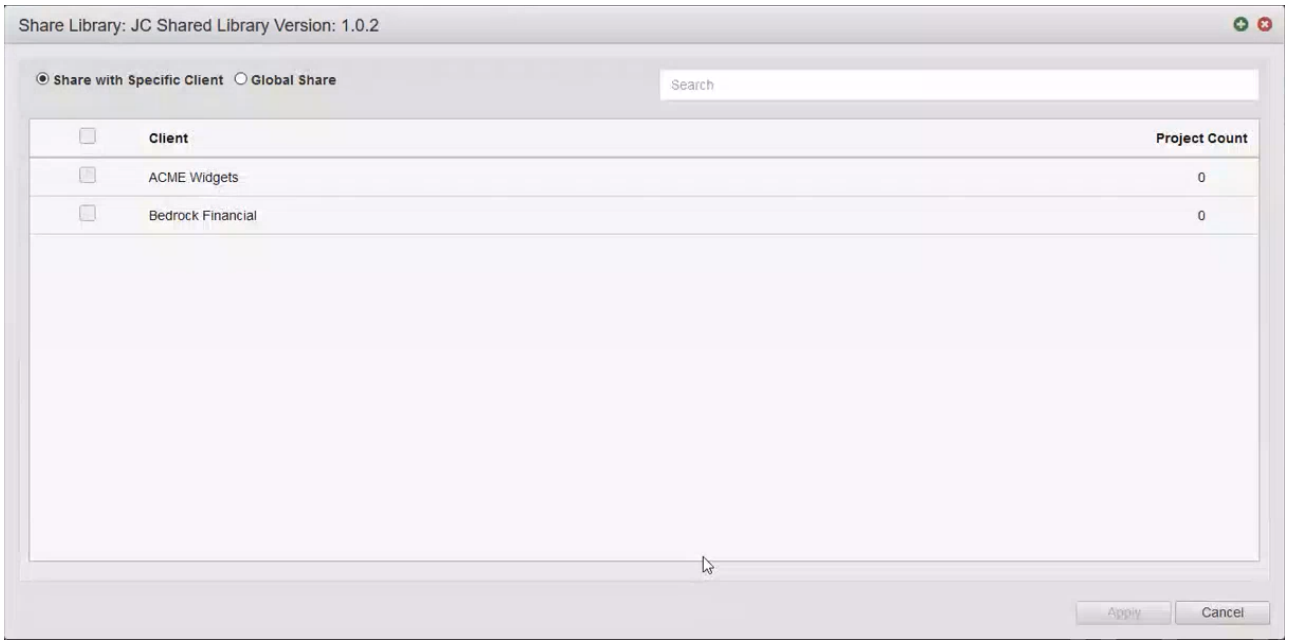
1. Open an organization, in this example it is Shared Libraries Demo, where you develop libraries you want to share.
2. Select a library. This can be a library of components, for example, the second-components.
3. Open an organization you want to share this library with, for example, Bedrock Financial.
4. Select the organization's libraries to view all libraries used by this organization. In this example, there are 2 global libraries but there are no shared libraries in the list.



5. Open the Shared Libraries Demo > Libraries and select your library to share, for example, JC Shared Libraries.
6. Select a version of a library you want to share with the Bedrock Financial organization, for example, version 1.0.2.
7. Tag the library, so it becomes read-only making it more secure to share.



8. Click Share Library Version to specify how and with whom to share the library.

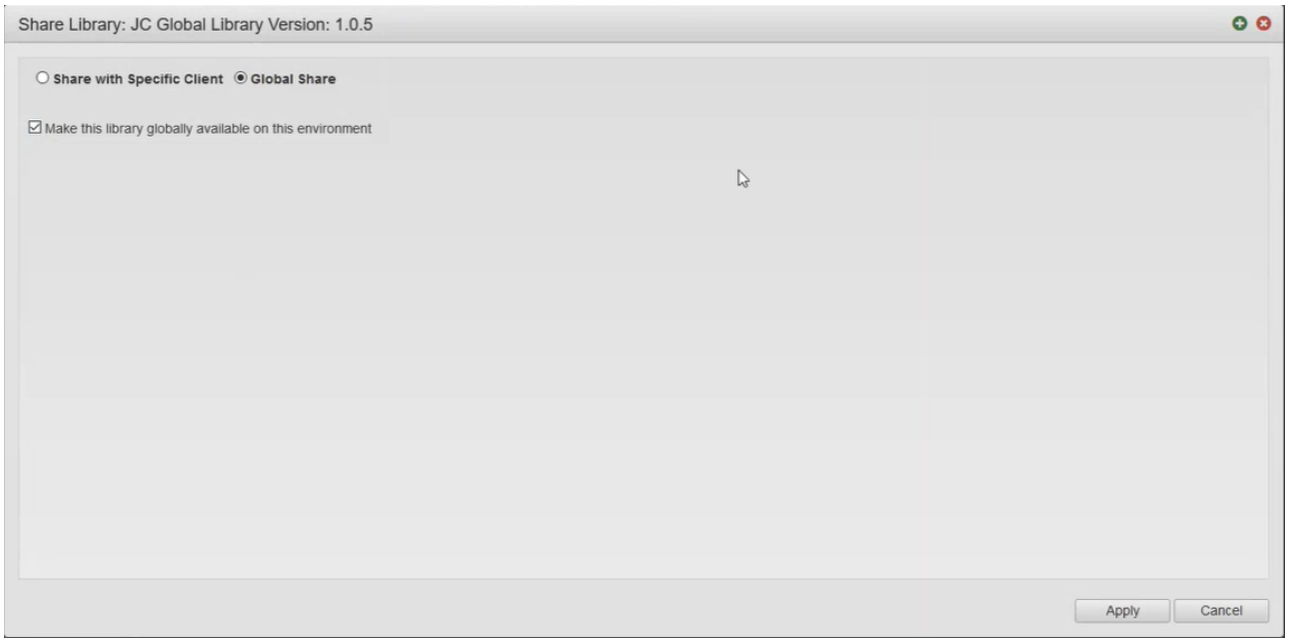


9. Select the Share with Specific Client option to share with specific organizations.
10. Select one, several, or all organizations from the Client column to share the library with. Use the search bar to look for organizations you want to share the library with, for example, Bedrock Financial.



You can see the number of projects that use this shared library in the Project Count column. In this case, there is only 1 project that uses it.

11. Alternatively, select the Global Share option and select the Make this library globally available on this environment checkbox to share it with all organizations globally.

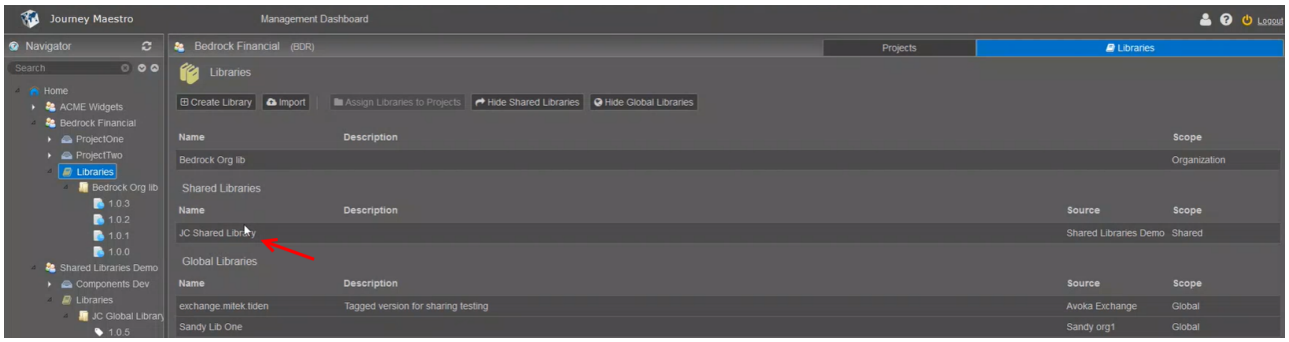


NOTE
You need to have the Library Share Global permission to be able to use the Global Share option.

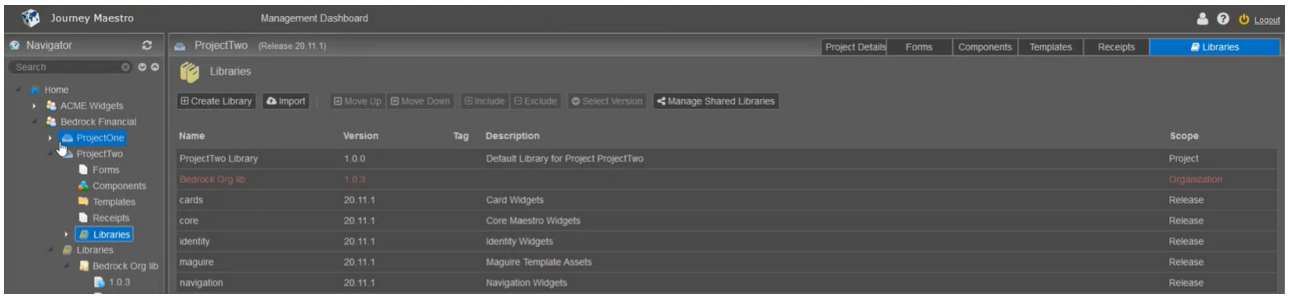
12. Click Apply to share the library with the selected organizations.

> NOTE
>
> You still need to assign the shared library to a project or projects under the selected organizations.

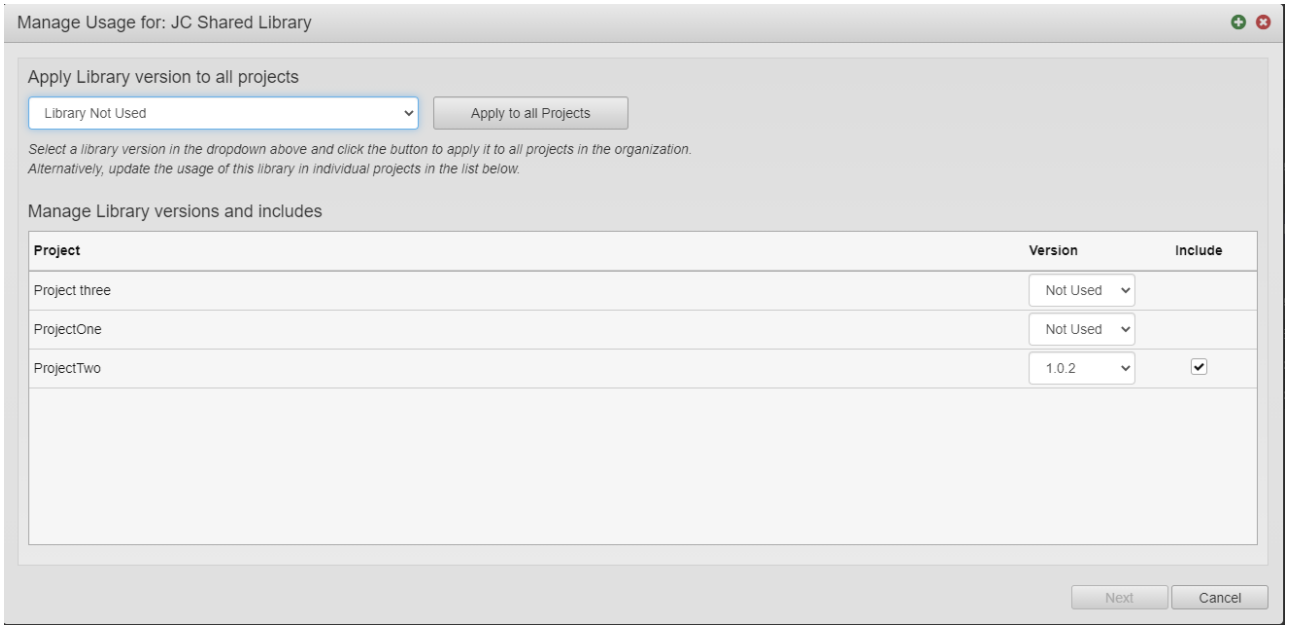
13. Go back to the Bedrock Financial organization and select Libraries to check the new shared library is added to the list of available libraries.



This means the JC Shared Library can be used in this organization, but it still needs to be assigned to projects. You can open a project, for example, ProjectTwo, and select Libraries to see that this shared library is not in there yet.



14. Click Assign Library to Projects to choose projects to assign the library to.

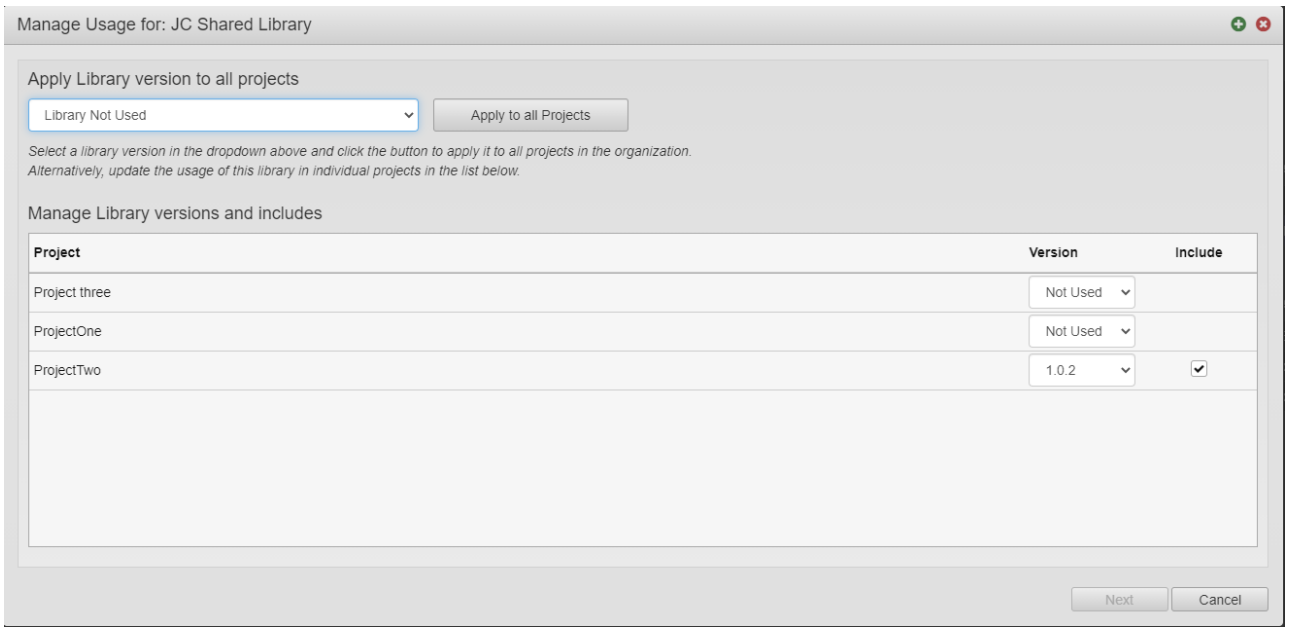


15. Select a library from the dropdown list, for example, JC Shared Library 1.0.2.

> NOTE
>
> Select Library Not Used to un-assign the library.

16. Select a library version from the Version dropdown list for a project you want to share it with. For example, select version 1.0.2 for ProjectTwo.

    If you want to assign the library to all projects in the list, click Apply to all Projects.

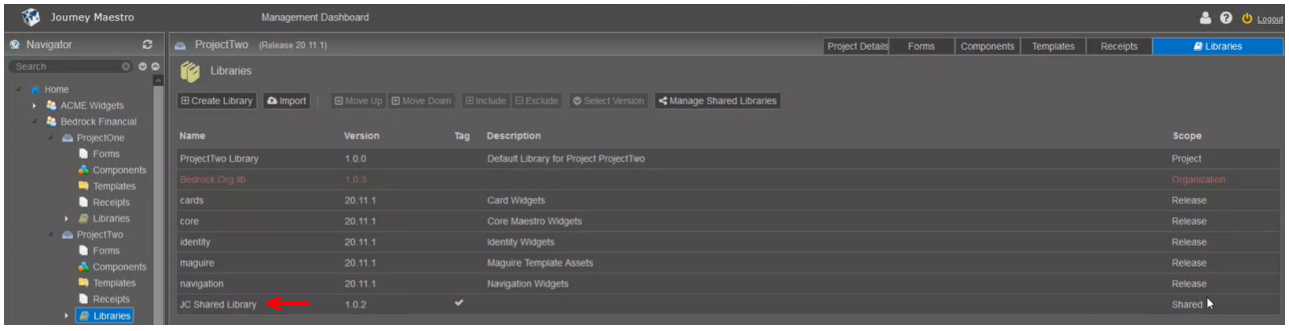


17. Select the Include checkbox to include this library.
18. Click Next to review the changes and confirm the sharing options.

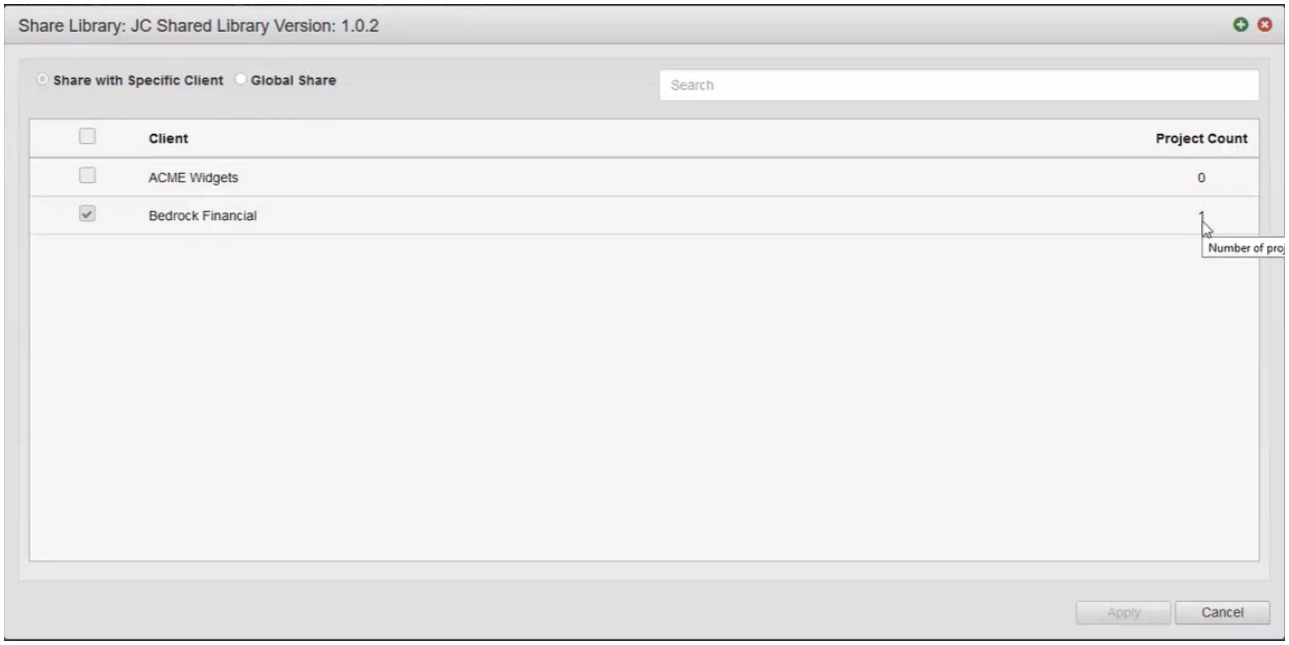


> NOTE
>
> The changed libraries are shown in blue.

19. Click Apply to assign the library to the project. Check the shared library is added to the list of available libraries of this project.



20. Drag and drop the JC Shared Library to change its order in the list of libraries. Alternatively, you can scroll through the list using the Move Up and Move Down buttons.
21. Open the Shared Libraries Demo > Libraries > JC Shared Libraries and click Share Library Version to see in how many projects it is used.



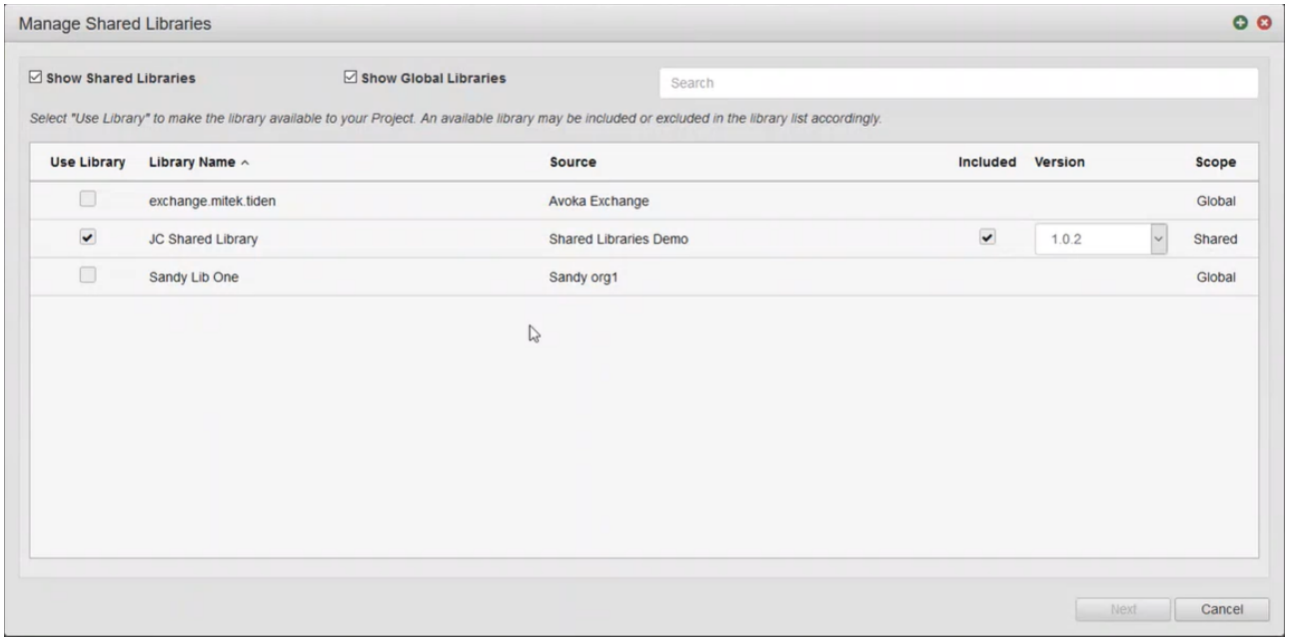
Now, you have a centralized shared library that is shared across the Bedrock Financial organization and it's applied to a project library, so you can use the library in your forms.

> NOTE
>
> You can't remove the shared library from the organization by clearing the checkbox if there are projects within this organization that use it. You need to remove it from the projects first.

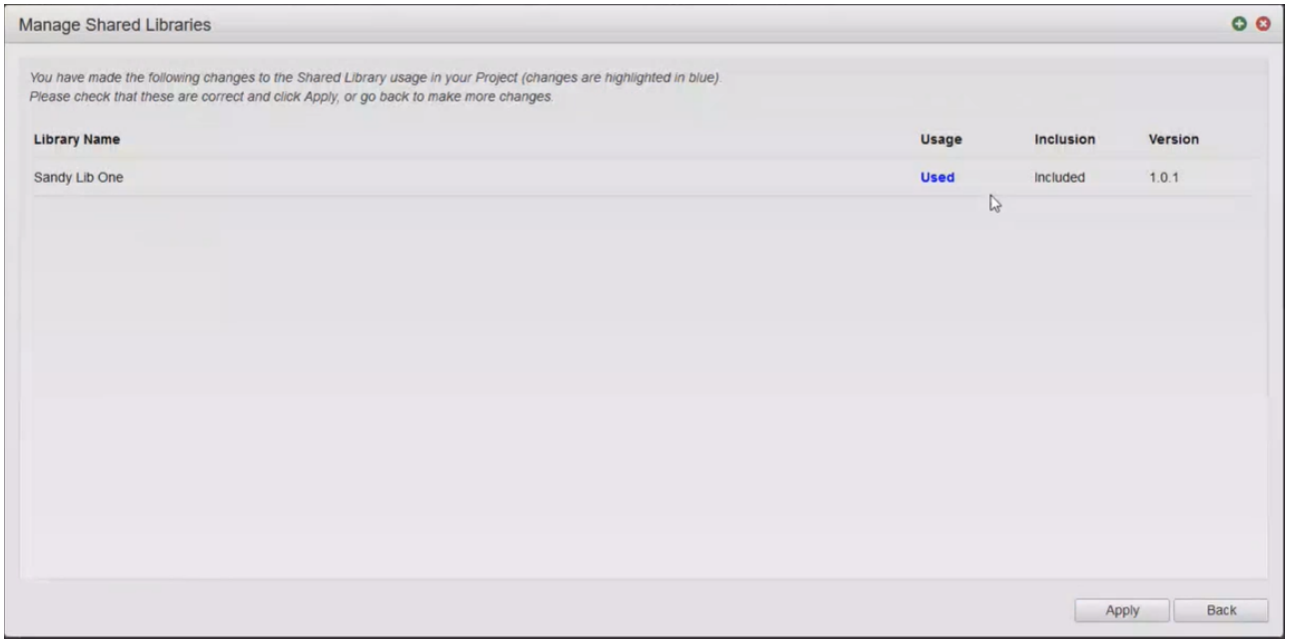
Manage Shared Organization Libraries

To manage shared libraries with individual projects:

1. Open an organization.
2. Select a project and click Libraries.
3. Click Manage Shared Libraries to manage shared libraries from the project.



4. Select the Show Shared Libraries and Show Global Libraries checkboxes to filter the list of shared libraries.
5. Select the Use Library checkbox to make use of a shared library.
6. Check the name of the library in the Library Name column. Different shared libraries can have the same names, so check the Source column to see where the library is coming from.
7. Select or clear the Included checkbox to include or exclude a shared library.
8. Select a library version from the Version dropdown list.
9. Check the scope of a shared library in the Scope column, which can be:
    - Global - Any new organizations created after that will inherit this shared (global) library. You can check the scope of the lib is set to global to confirm this lib is globally shared.
    - Shared
10. Click Next to review and confirm the changes you have done to the library configuration.



11. Click Apply.
12. Check the shared library configuration changes in your project.

Exclude Shared Organization Libraries

To exclude the shared library:

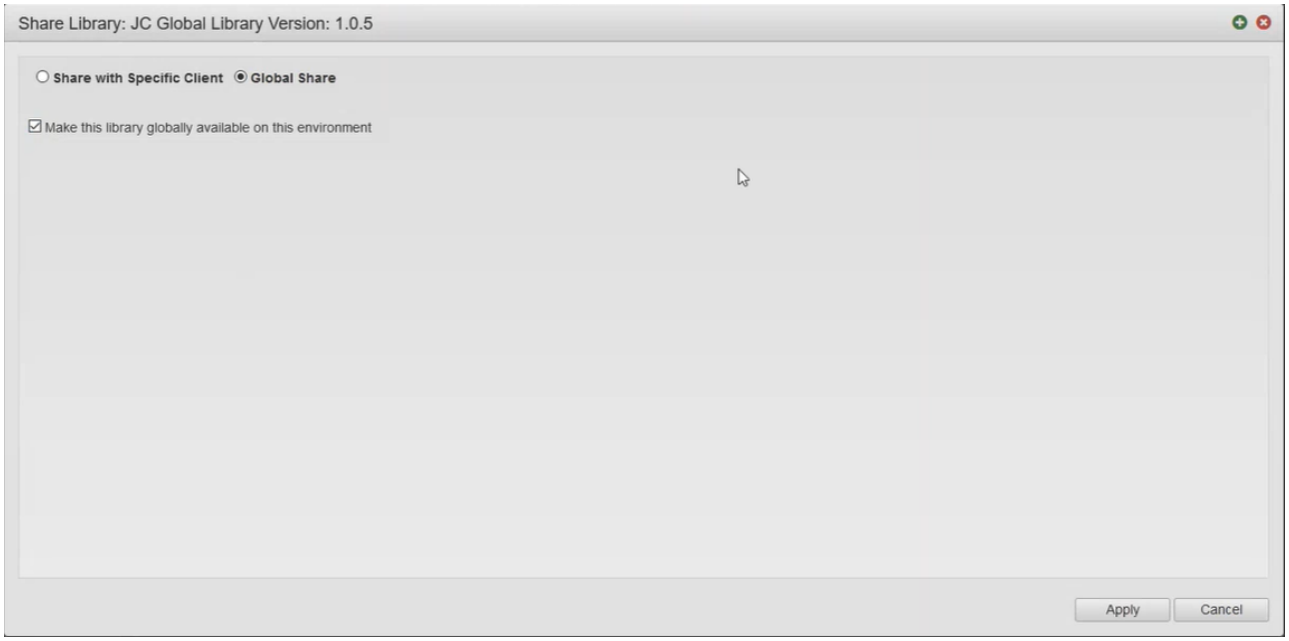
1. Open an organization
2. Select a project and click Libraries.
3. Select a shared library and click Exclude. Th.e excluded library doesn't move down the library list, as it used to do prior to this release.

Un-share Global Organization Libraries

As discussed above, you can share organization libraries across all organizations, which makes them global libraries. Sometimes you need to un-share them. Un-sharing when it is shared with specific clients is about unticking the checkboxes in the list, libraries that are included in projects will be read-only until no more projects have them included, if they are excluded then they can be unshared. However, you can't un-share a library with specific clients that you don't have access to.

To stop using the global organization library:

1. Open an organization.
2. Select a project and click Libraries.
3. Click Share Library Version.



4. Select the Global Share option.
5. Clear the Make this library globally available on this environment checkbox to remove shared library from all organizations.
6. Click Apply.

NOTE

You can't un-share the global library if there are projects that use it. You need to remove it from the projects first.

Add Sharing Permissions

You need to have the following permissions set up in Manager to be able to share libraries:

- Library Share
- Library Share Global

NOTE

When you add sharing permissions to users on multi-tenanted Maestro environments, make sure users can't and don't share content between competitor organizations as it may cause leaks or conflicts of intellectual property.

To add the permissions:

1. Log onto Journey Manager.
2. Assign the following permissions to a role you use to develop or administer Maestro forms.
    - Library Share - most developers can have this permission.
    - Library Share Global - only admins, for example, the Maestro Global Lib Admin role, should have this permission, because incorrectly shared library can cause leak of intellectual property.

Next, learn about library versions.

Export a Library

MaestroThe UI design product. | Form Builder | 18.05 This feature was updated in 18.05.

Journey Maestro allows you to export libraries, so you can easily move or copy them into different organizations or environments, as well as create a back-up of libraries. However, library resources are exported as published versions, which means that the resources will not be editable when imported. To edit them, you will need the access to the original library and its resources.

To export a library:

1. Open an organization or project in the Navigator and select a library in the Libraries folder.
2. Select the library version you want to export and click Export Version. The library will be downloaded as a ZIP file.

   NOTE

   The library ZIP file name starts with `maestro-org-lib` or `maestro-project-lib`, reflecting the library scope, and includes the library name, and export date.

Next, learn how to import a library.

Import a Library

MaestroThe UI design product. | Form Builder | 18.05 This feature was updated in 18.05.

Journey Maestro allows you to import libraries, so you can easily move or copy them into different organizations or environments, as well as create a back-up of libraries. However, library resources are exported as published versions, which means that the resources will not be editable when imported. To edit them, you will need the access to the original library and its resources.

> NOTE
>
> If a project library contains resources that need to be used across multiple projects, don't import that library into multiple projects as this will create unnecessary duplication. Instead, move the library to an organization-level library. Any resources in the organization library will become available to all projects within the organization.

To import a library:

1. Open an organization or a project in the Navigator.
2. Select a Libraries folder and click Import.
3. Select an exported library as the archive ZIP file and click Open.

Next, learn how to publish resources to another library.

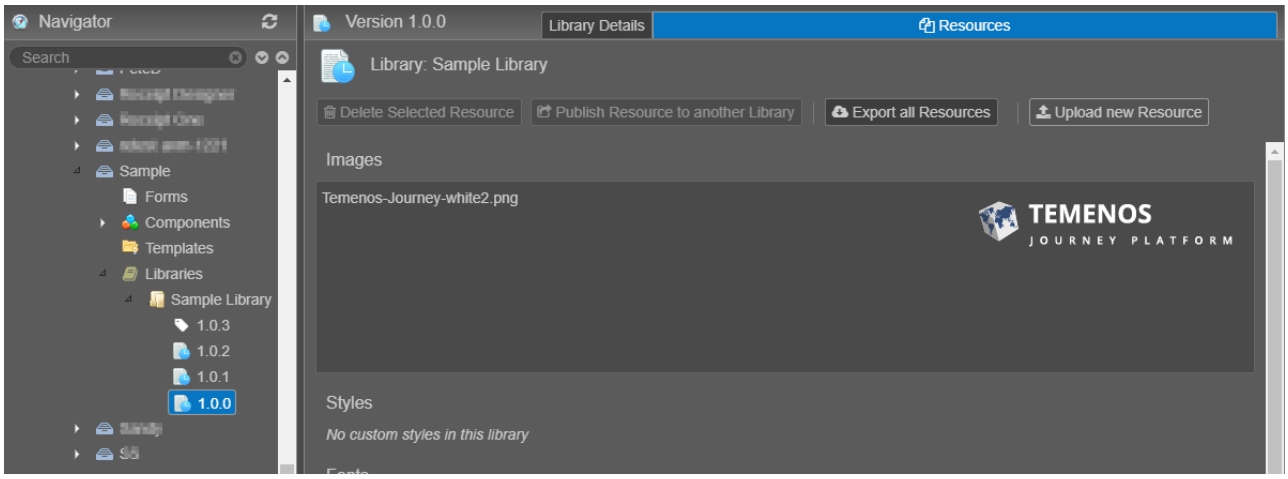
Publish Resources To Another Library

MaestroThe UI design product. | Form Builder | 18.05 This feature was introduced in 18.05.

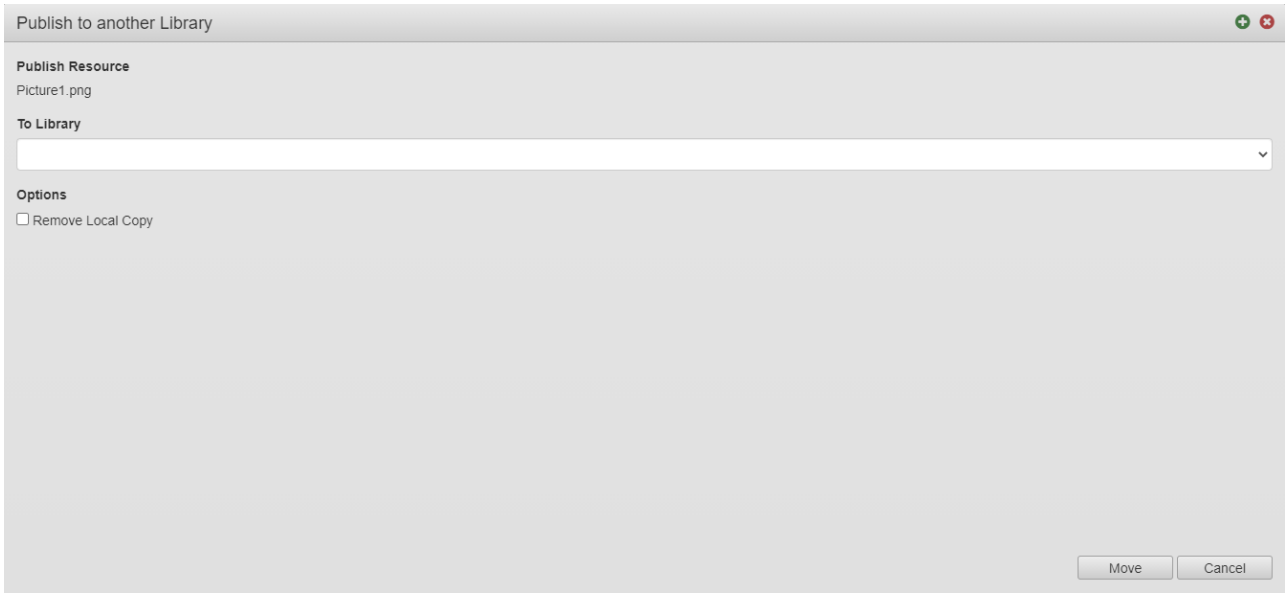
Journey Maestro allows you to publish a resource into a library version. You may want to do this to re-publish already published resource to a different library version, for example, to promote a resource from a project library to an organization library, or to share a project resource in other projects.

To publish a resource to another library version:

1. Open a resource in Maestro.



2. Select the resource and click Publish Resource to Another Library.



3. Select the library to where you want to publish the resource from the To Library dropdown list.
4. Select the Remove Local Copy checkbox to remove the resource from the current library. This ensures that the resource is available only in the destination library, assuming it was previously available only in the current library.

   NOTE

   We recommend selecting Remove Local Copy to prevent resource duplication in multiple libraries.

5. Click Move.

Next, learn about organization libraries.

How to Use Organization Libraries

MaestroThe UI design product. | Form Builder

Maestro allows you to create and update libraries. If you've never used a library before, or if you're not sure how they fit into the Maestro workflow, follow along to see them in action. This guide assumes you understand the basics of libraries; if you don't, you should read the Maestro Libraries Overview first, then come back to this topic.

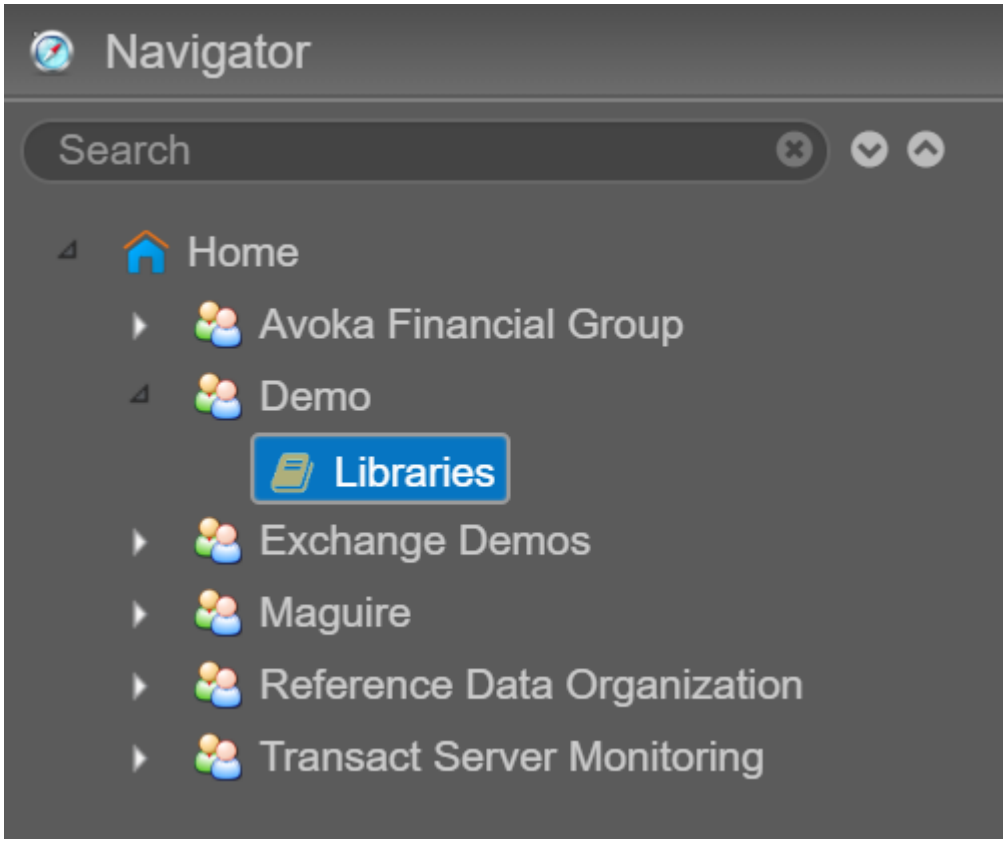
Step 1: Create a Library

You can create a library at the organization or project level. If a library is created for an organization, its resources will be available to every project and form within that organization. If a library is created for a project, its resources will only be available to the forms within that project.

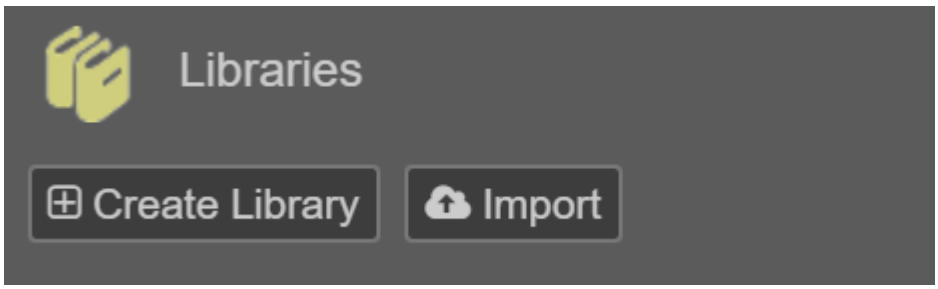
In the following exercise, we're going to create an organization library that makes a component available to all of the projects and forms within that organization. If you were creating a project library instead, you would need to create a project first then create the project library; otherwise, the process is quite similar.

To create a library:

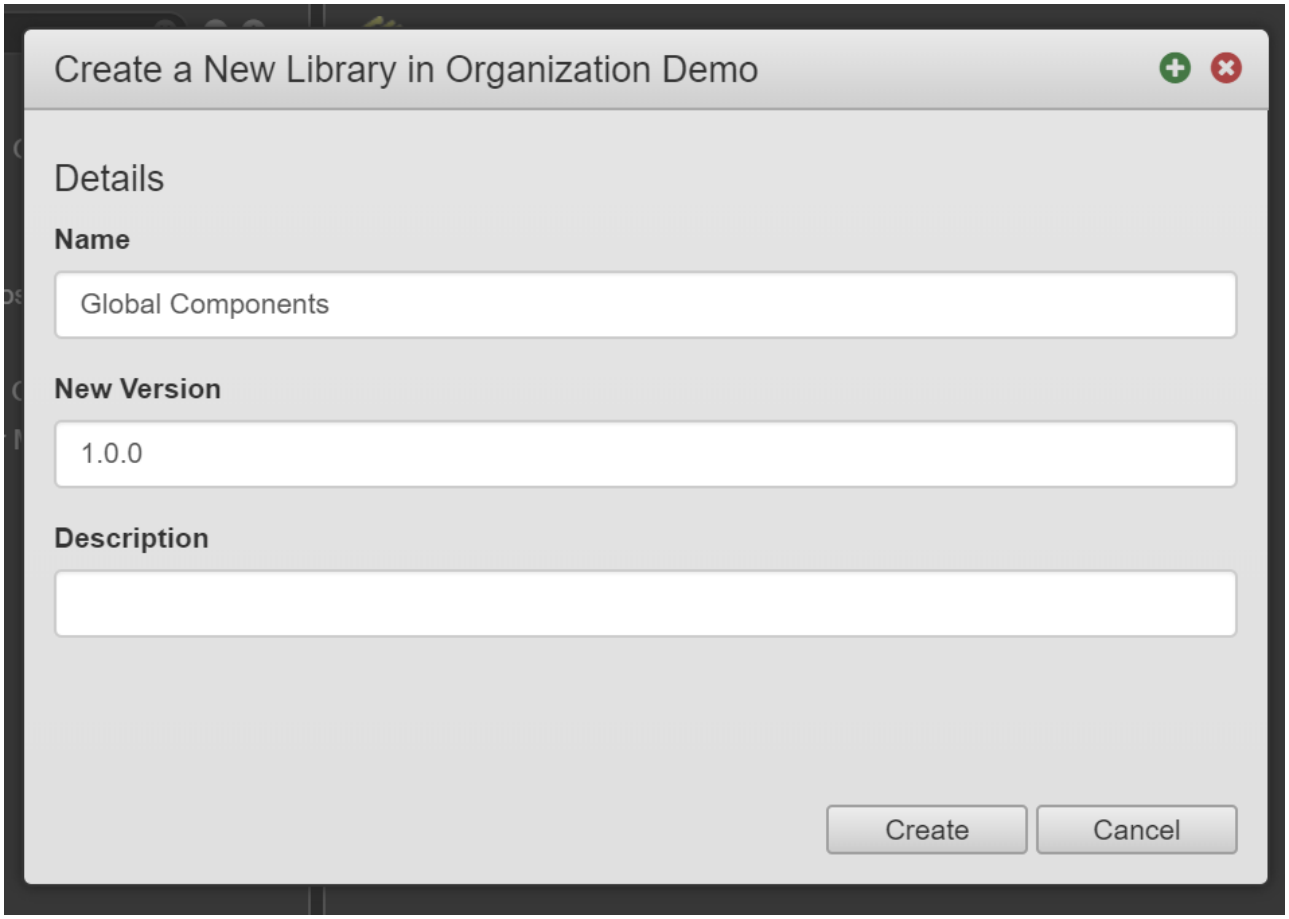
1. Login to Journey Maestro.
2. Expand an organization in the **Navigator** pane and open the **Libraries** item. (If there are no organizations that you can use, you'll need to create one first.) We'll use the same organization throughout this exercise.



3. Click **Create Library**.

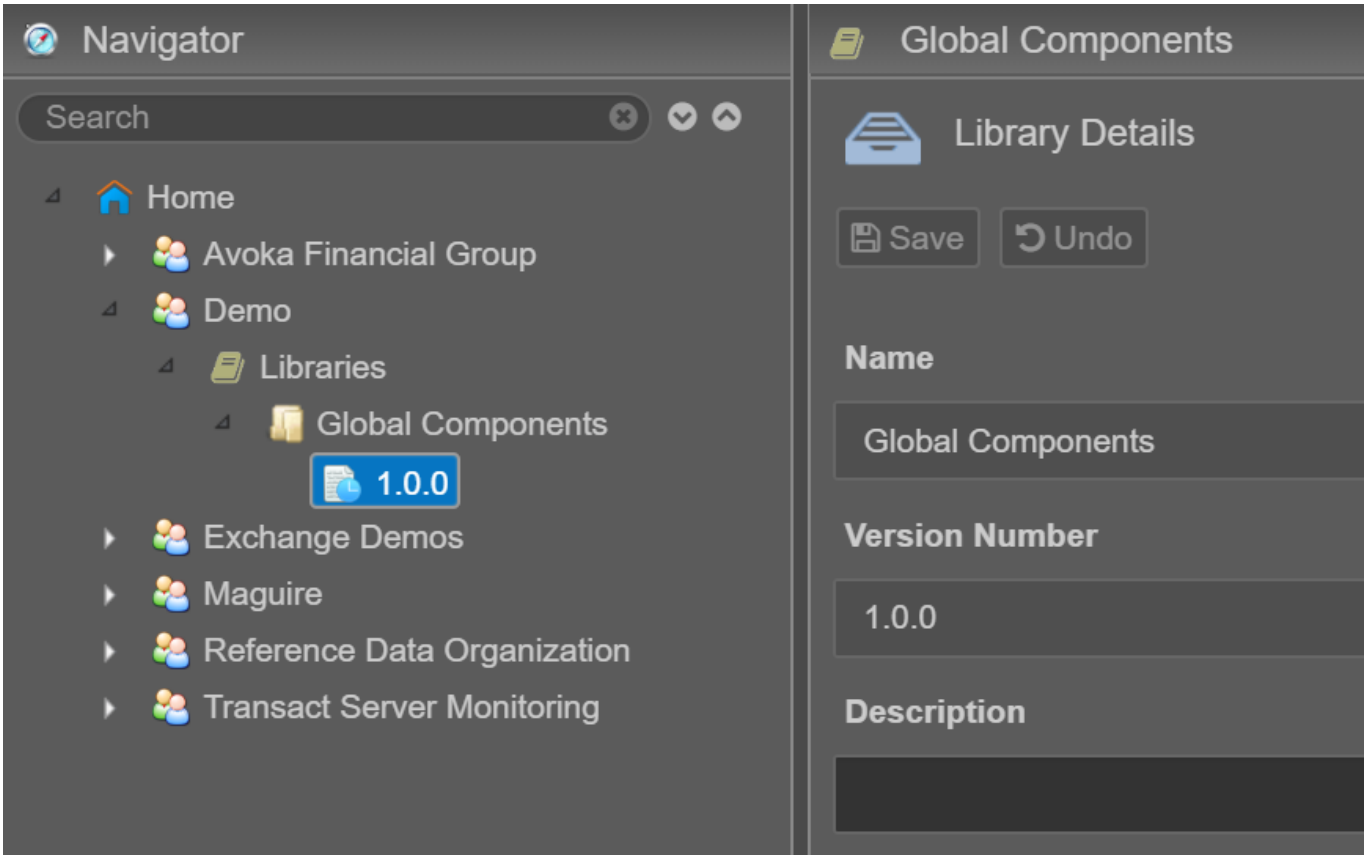


4. In the **Create a New Library** modal window, enter a **Name** for the library and click **Create**. You can ignore the other fields for the time being.



5. After creating the library, you'll be taken to its **Library Details** page. Here you can update the library's**Description**, or switch to the **Resources** tab to view the resources published to this library.

The new library has a version number of 1.0.0 which was entered on the **Create a New Library** modal. You can see the library's **Version Number** on the **Library Details** page, as well as in the Navigator pane under the library.



The library version number is significant because:

- All libraries are automatically versioned. You don't have to use this functionality, but versioning does help in managing form development.
- All libraries must have at least one version. If you delete the last version of a library, you'll be forced to delete the library itself.

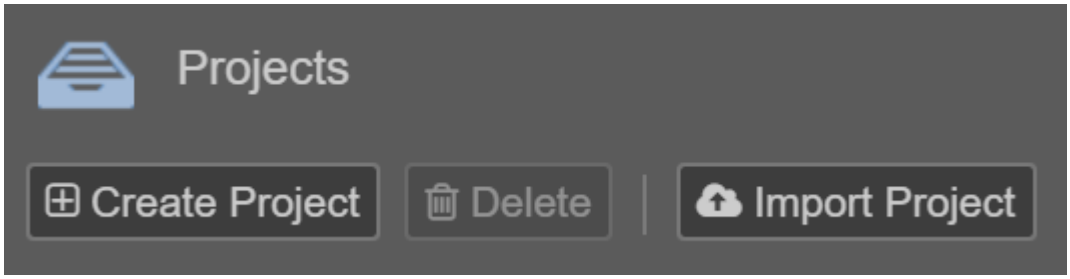
Step 2: Create a Project

We've created a library but there's nothing in it. We'll create a project and work on the resources for our library in that project. Then we'll publish the resources to the library in a later step.

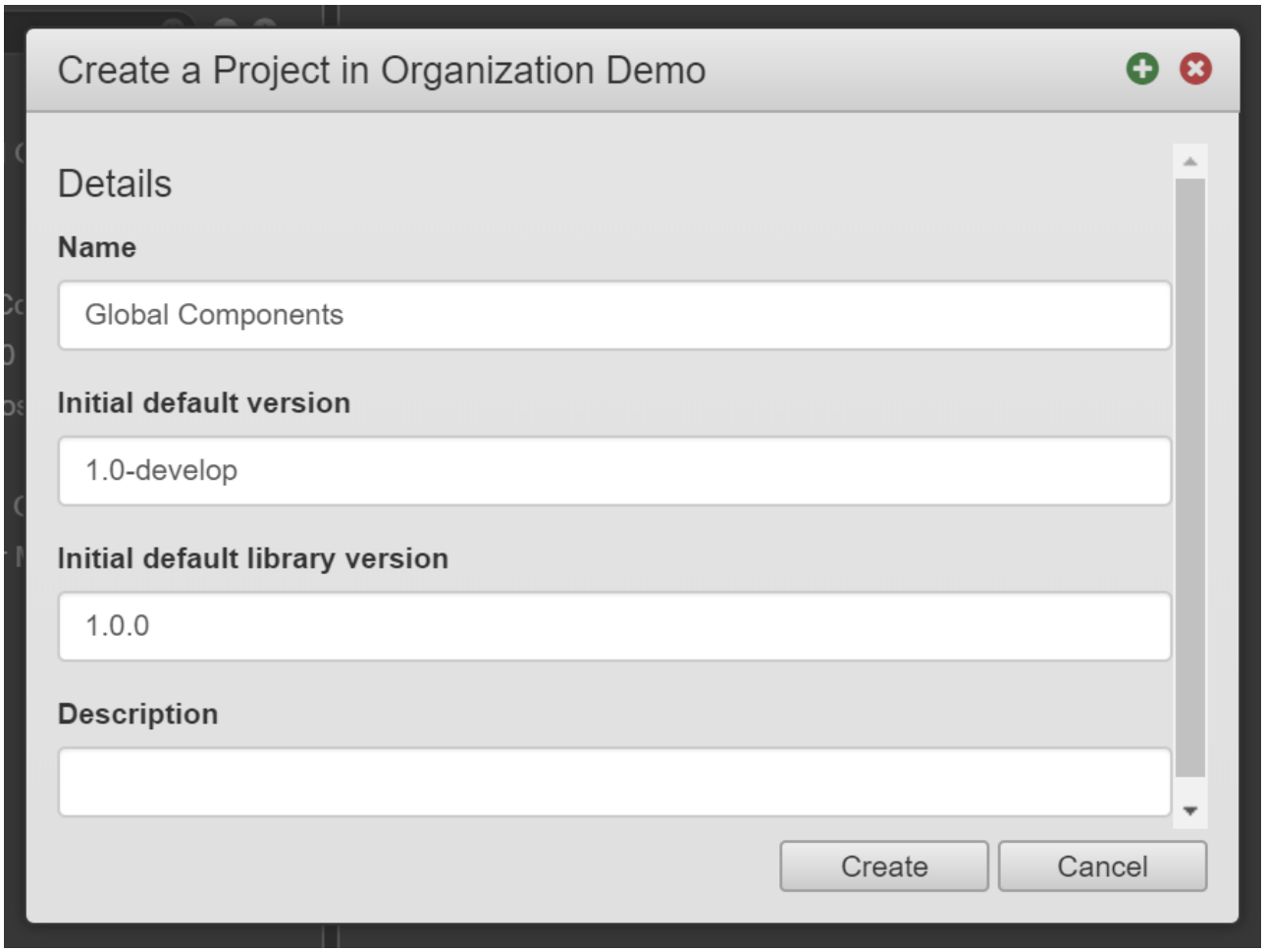
To be clear, we don't have to create a new project for every library we create. We could create a single project and work on all of our library resources in that project. However, this is not recommended as it becomes difficult to keep track of where the resources are stored. Creating one project per library is best practice, with the advantage of being a very scalable approach.

To create a project for our library:

1. Select your organization from the **Navigator** pane.
2. Switch to the **Projects** tab and click **Create Project**.



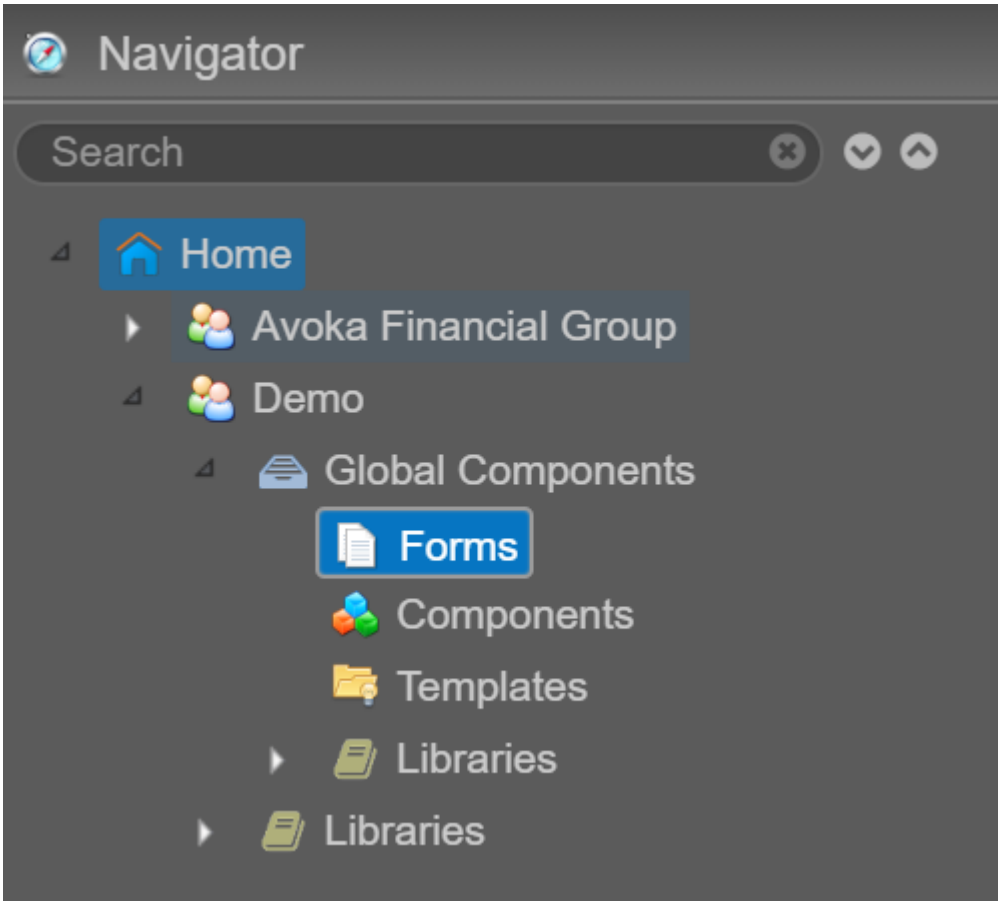
3. In the **Create a Project** modal, enter a **Name** for the project. It's a good idea to make this name similar or identical to the name of the library, so the relationship between the project and the library is apparent. The other fields can be ignored for the time being.
4. Click **Create**.



On the **Navigator** pane, we can see four folders in the project we just created:

- Forms
- Components
- Templates
- Libraries

These four folders are included in every project by default.



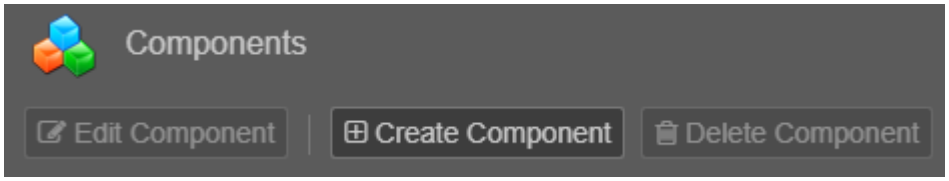
For the purpose of the current exercise, the **Components** folder is the only one that interests us now because a) forms can't be published to a library, b) we don't want to publish resources to our project's library, and c) creating a template is beyond the scope of this topic.

Step 3: Create a Component

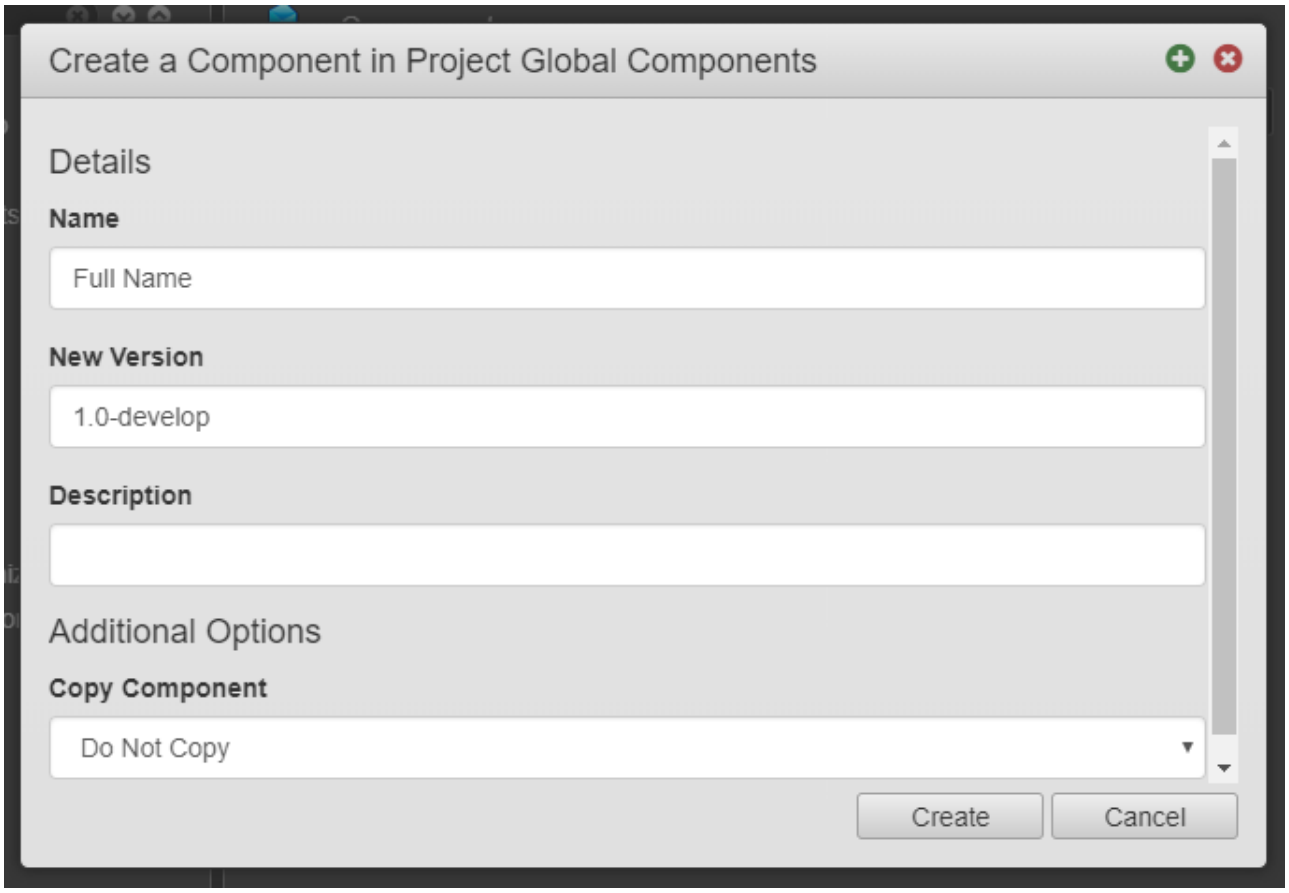
In this section, we're going to create a component. This component will become available to all of the projects in our organization.

To create a component:

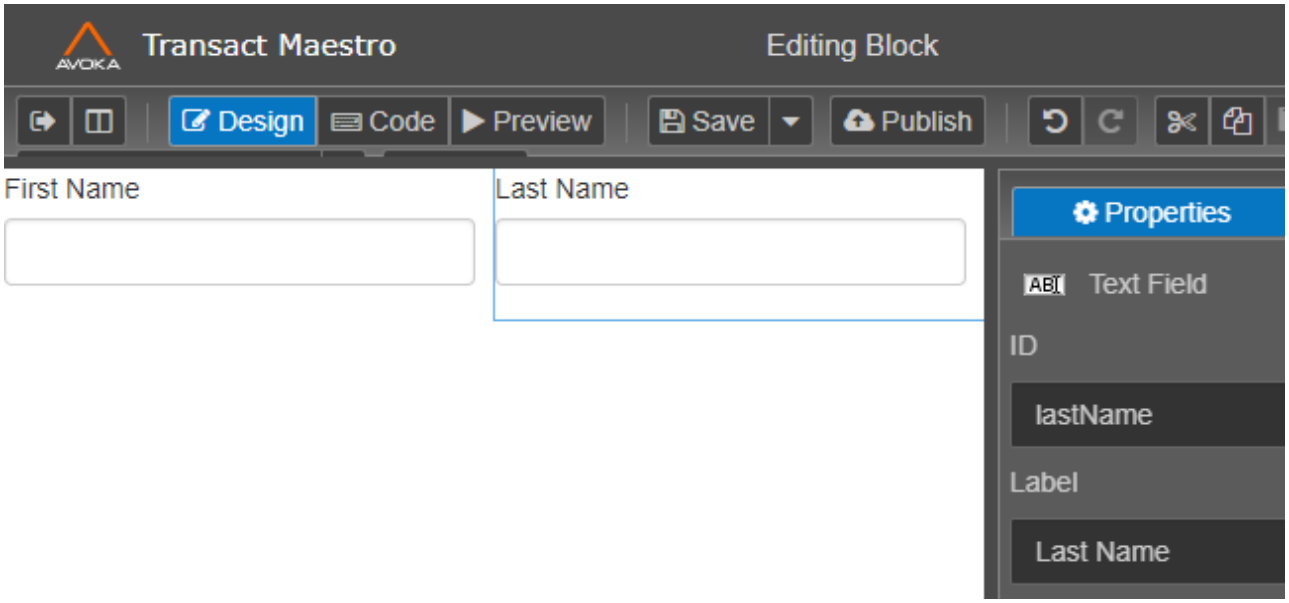
1. From the **Navigator** pane, select the **Components** folder under our project.
2. When the **Components** tab loads, click **Create Component**.



3. Enter a **Name** for our component. To follow along with this example, call it "Full Name".
4. Click **Create**.



5. The **Maestro Editor** opens a new tab for the new component. After the component tab opens, drag two **Text Field** components into the main work area. Set the **ID** of the first text field to "firstName" and the **ID** of the second text field to "lastName".



6. Click **Save**.

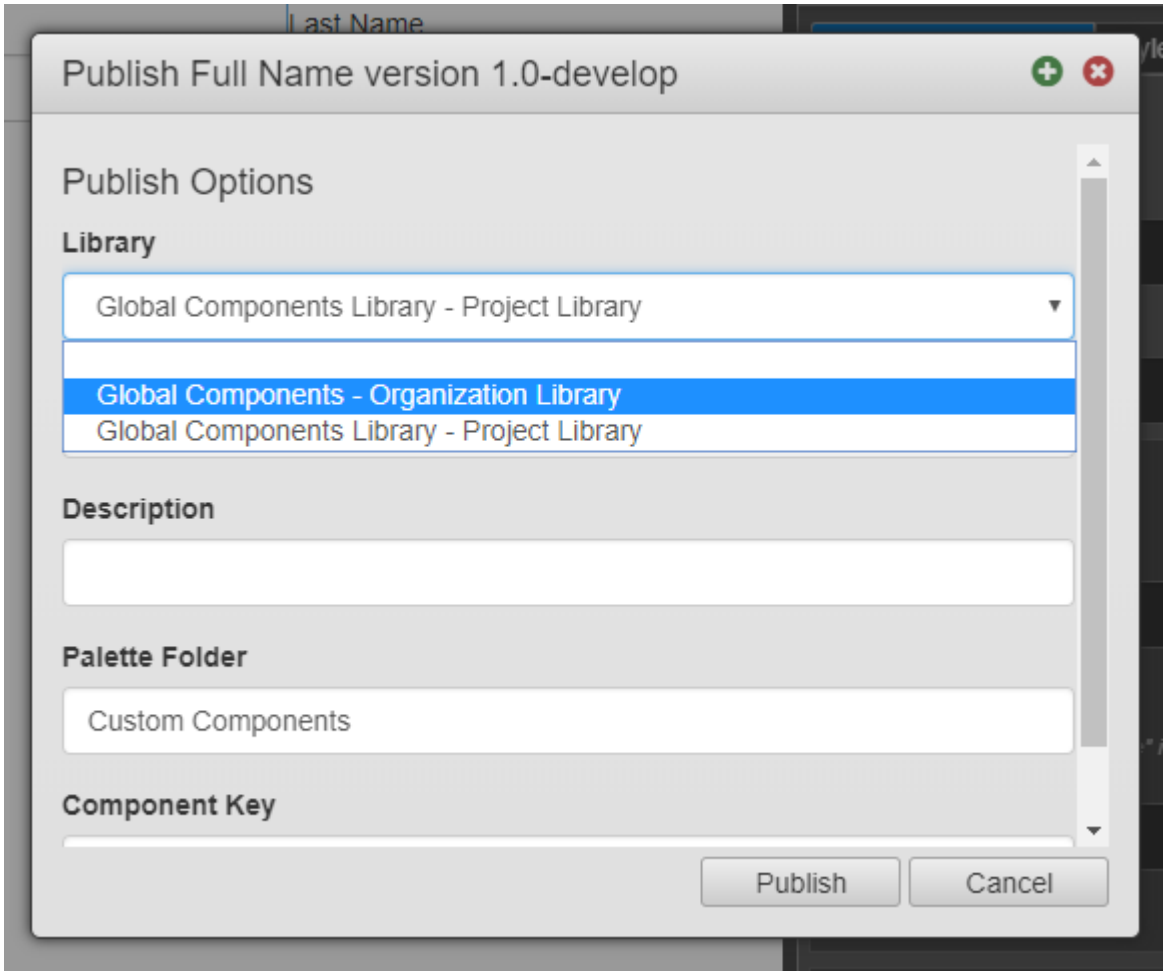
Step 4: Publish a Component

For our component to become available via the library we created earlier, we need to publish it to the library.

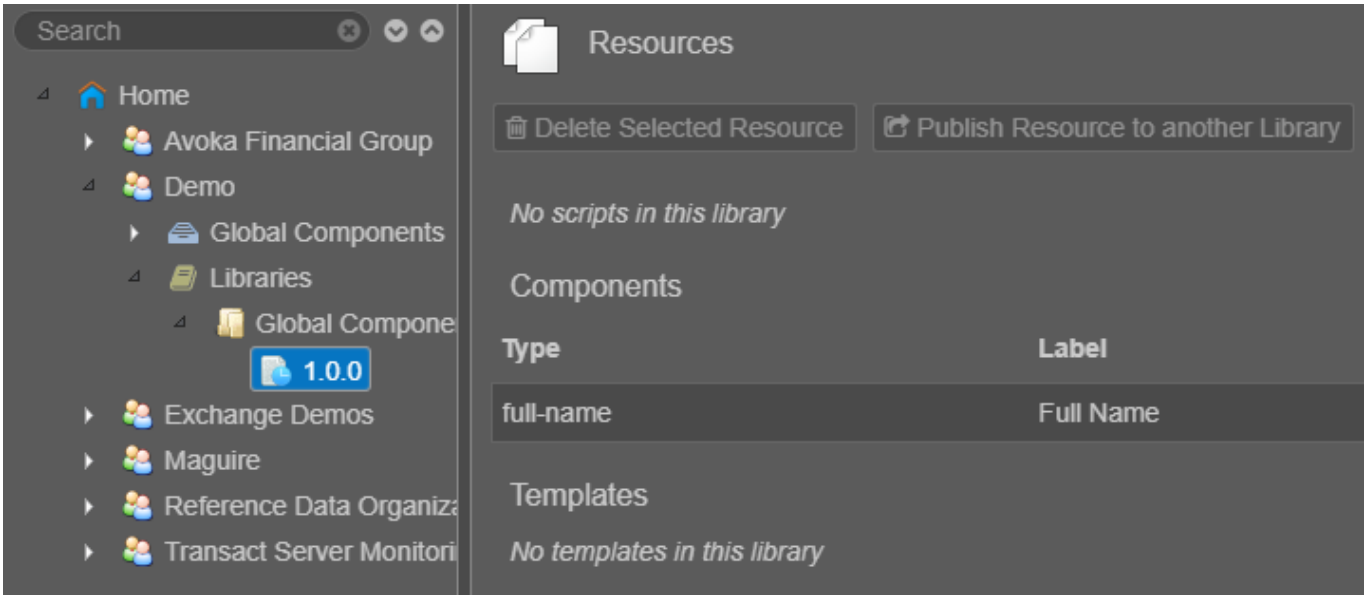
To publish a component:

1. With the component still open in the **Maestro Editor**, click **Publish**.
2. When the **Publish** modal appears, select the organization library from the **Library** drop-down. Note that each library in the drop down list is identified as being either an organization or project library.

3. Click **Publish**.



Select your organization library version in the **Navigator** pane and switch to the **Resources** tab, and you'll see your component included in the library's Component resources, meaning it is now available to be used in any project in your organization.



If the component doesn't appear, verify that you published the component to the organization library, not the project library. You may also need to refresh the page.

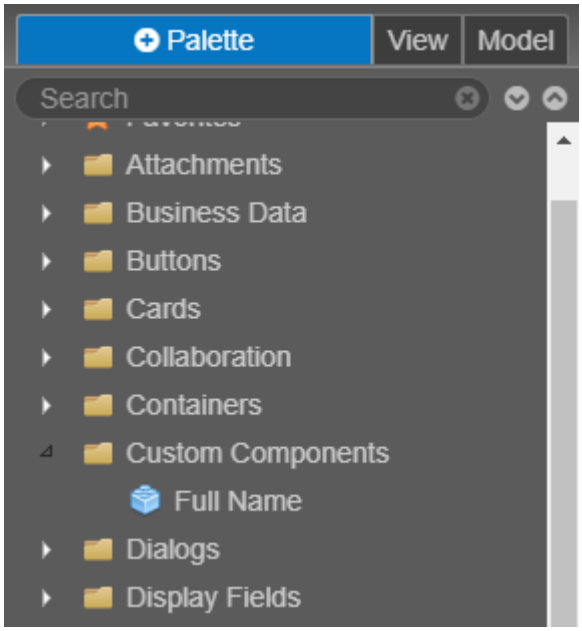
Step 5: Use a Published Component

Using a published component is similar to using any other component. The only difference is that components published to a library are not available everywhere. They're only available within the organization or project they're published to.

To see this in action, follow these steps:

1. Create a new project within your organization.
2. Select the Forms folder in your new project and click **Create Form**.
3. Enter a **Name** for your new form and select a **Template**, then click Create.
4. Your new form will open in the **Maestro Editor** in a new tab.
5. Expand the **Custom Components** folder inside the **Palette** pane. You'll see that the "Full

Name" component is available for use in our form.



The Advantage of This Approach

Let's say you need to change the "Full Name" component, perhaps to add a new feature. You can return to the project where that component exists and make whatever changes you want without worrying about breaking the component. This is because changes are only reflected throughout the organization's forms when the component is re-published. Until then, you can develop components with the peace of mind that the changes you're making are private and will not impact Form Builder's who are using your component.

When you've finished the changes to your component, you just need to re-publish it and every instance of that component is automatically updated. This means the component has a single source of truth that you can evolve as requirements change.

Next, learn about shared libraries.

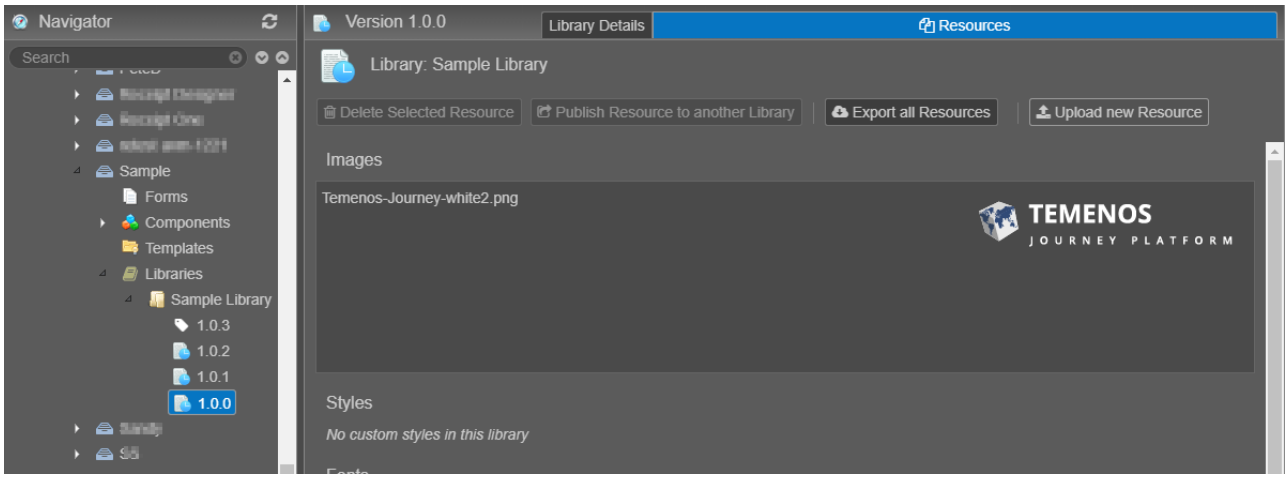
View Resources

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Journey Maestro allows you to view the list of resources published to an asset, such as a form, component, template, and library.

To view the list of resources in an asset:

1. Open a project in the Navigator pane and select an asset from the list of assets versions, for example, a library version.
2. Double-click an asset version you wish to view resources of.
3. Select the Resources tab to view a list of resources grouped by resource type, as shown below.



The resources are grouped as follows:

- Images - a list of images
- Styles - a list of styles identified by a Key and a Label
- Fonts - a list of available fonts, such as TTF, EOT, SVG, or WOFF. For more information, see Web Font.
- Scripts - JavaScripts available to forms
- Components - a list of components
- Templates - a list of available templates that have been published into a selected library
- Languages - available languages
- Shared Translations - form's translations
- Domain Models - a list of JSON files describing data models, such as `model-schem-`

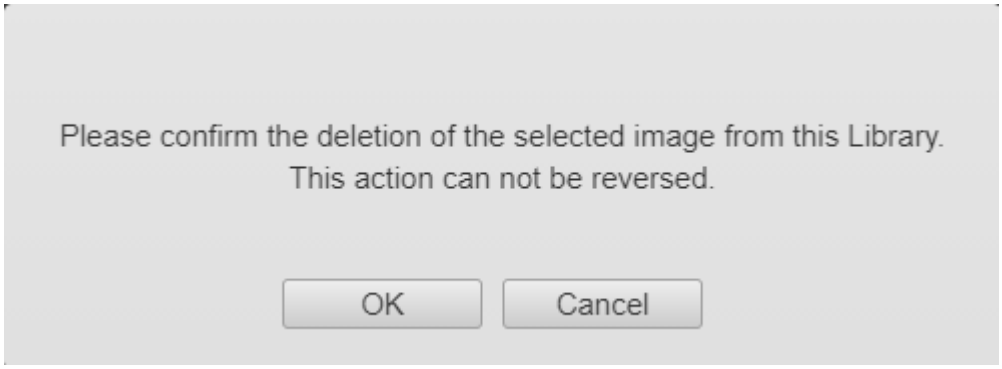
`a.json`

- Runtime CSS Resources - CSS and LESS

4. Click Delete Selected Resource to remove selected resources from the asset.

   NOTE

   Use `Shift` or `Ctrl` key to select multiple resources. | 21.05 This feature was introduced in 21.05.

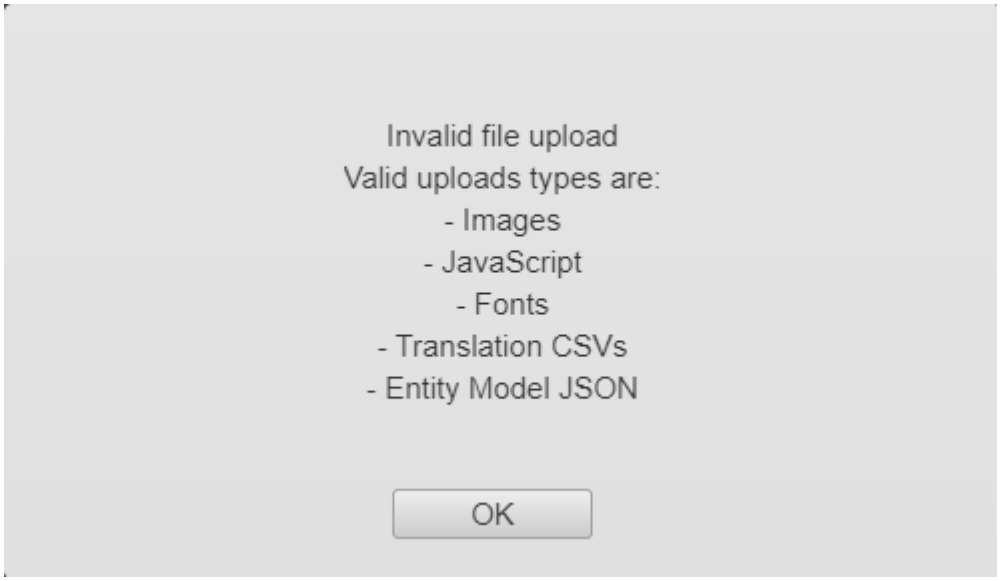
5. Click Publish Resource to another Library to publish selected resources into a library.

   NOTE

   Use `Shift` or `Ctrl` key to select multiple resources. | 21.05 This feature was introduced in 21.05.

6. Click Export all Resources to export all resources in a ZIP file, which you can then import into another form or a project.
7. Click Upload new Resource to add a new resource to the asset. | 21.05 This feature was introduced in 21.05. This is a fast and easy way to publish a resource, as opposed to the traditional way of adding a resource by uploading a new resource in the form when selecting the resource or alternatively using the form's export and import functionality.

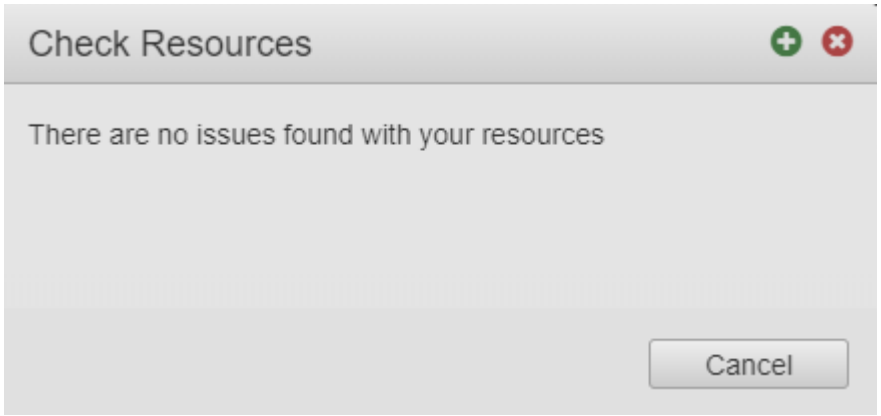
   A resource must be of the several allowed types, otherwise a following error message is shown.



NOTE

Use the Web Font component to add an SVG[1] font as a resource, otherwise it will be added as an image.

8. Click Check All Resources to validate resources. To ensure resources you add to a library are correct and won't cause any run time errors when a form is rendered, you can validate the resources within a library. During this process, Maestro runs a set of validation scripts to ensure that all resources, for example, styles and scripts, have valid structure and syntax. If validation is successful, the following message is displayed.

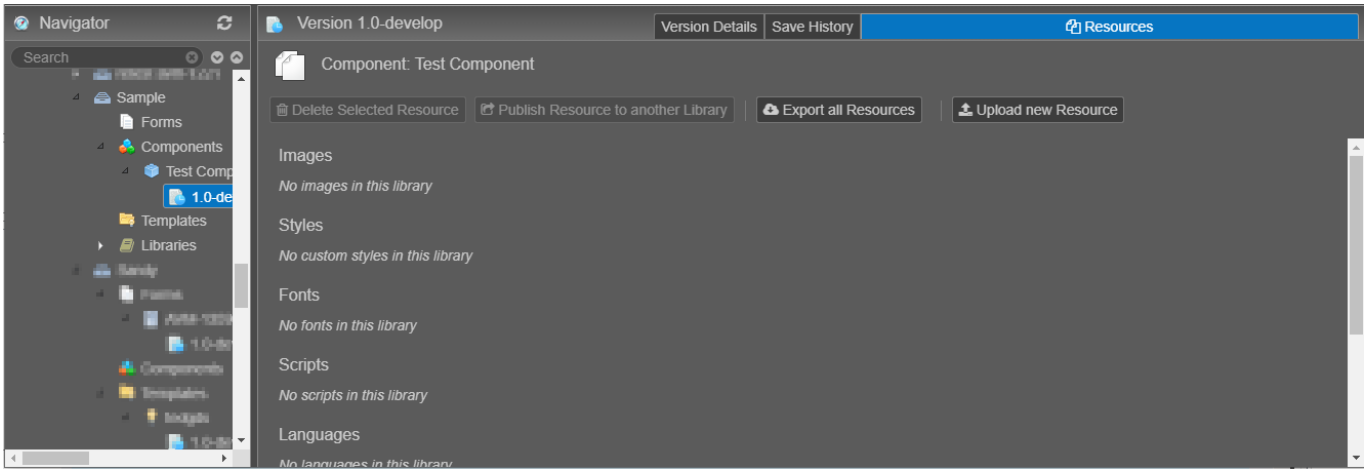


[1]An SVG font is a new version of the OpenType format, with SVG standing for Scalable Vector Graphics. The SVG glyph format allows the characters to be displayed in multiple colors and different transparencies, and some may even be animated.

NOTE

You can't open a release library, so if you don't have any user-created libraries, create one with the Create Library button.

You use the same steps above to view resources of forms, components, and templates.



Next, learn about library versions.

Load External JavaScript Libraries

MaestroThe UI design product. | Form Builder

You can use one of the following ways to include or load external JavaScript libraries in a Maestro form:

- The JavaScript Library widget in the Maestro palette, which allows the developer to upload a library in Maestro. Note that these libraries are loaded on Form Load and the source is included in the app.js file.
- The Page Tracking feature of a form configuration in Manager.
- Lazy loading the libraries using jQuery.

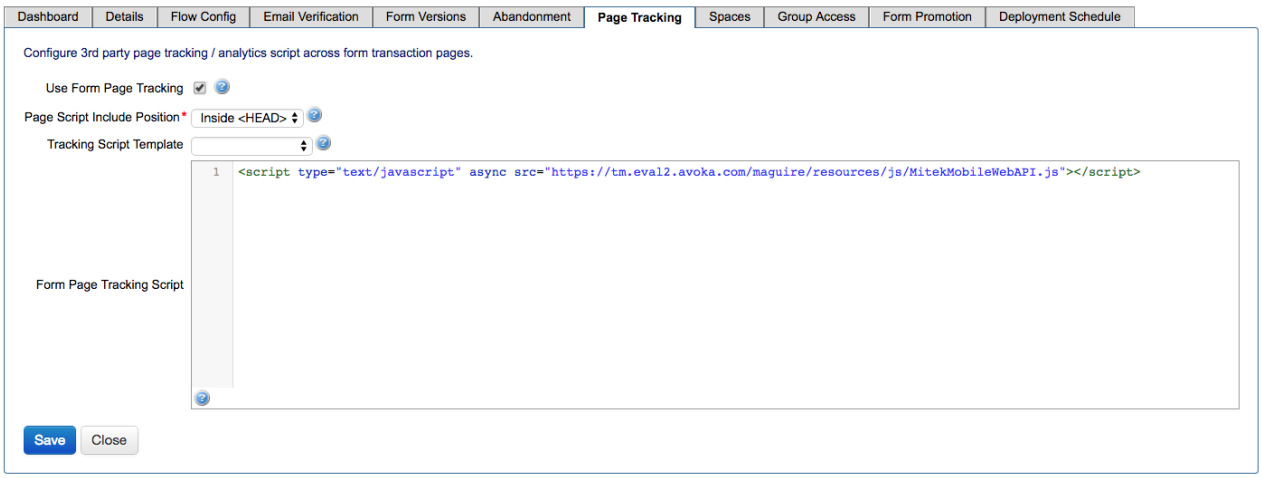
The last two approaches are commonly used.

Use Page Tracking

Although the feature is called Page Tracking, you can also use it to add custom JavaScript and HTML content into the application.

To add a JavaScript:

1. Navigate to the Page Tracking panel for your form configuration.
2. Toggle Use Form Page Tracking to ON.
3. Set the Page Script Include Position to Inside <HEAD>.
4. Add the JavaScript library reference, as shown below:



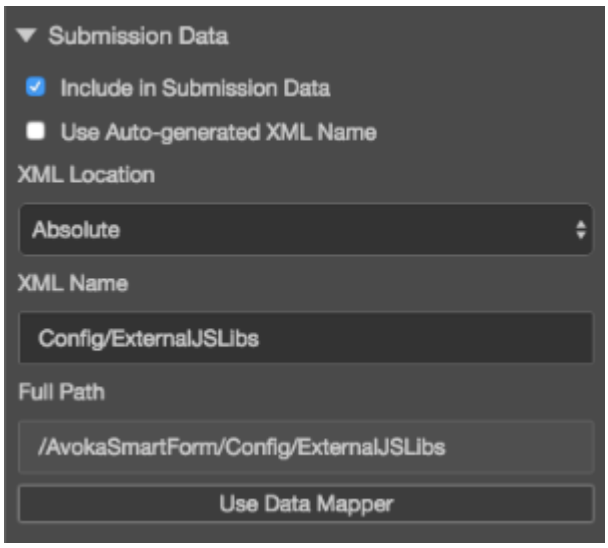
The async attribute can be used to allow asynchronous running of the script. This technique generally allows the Maestro form to load and render faster. Some JavaScript libraries cannot be used this way.

Load Libraries with jQuery

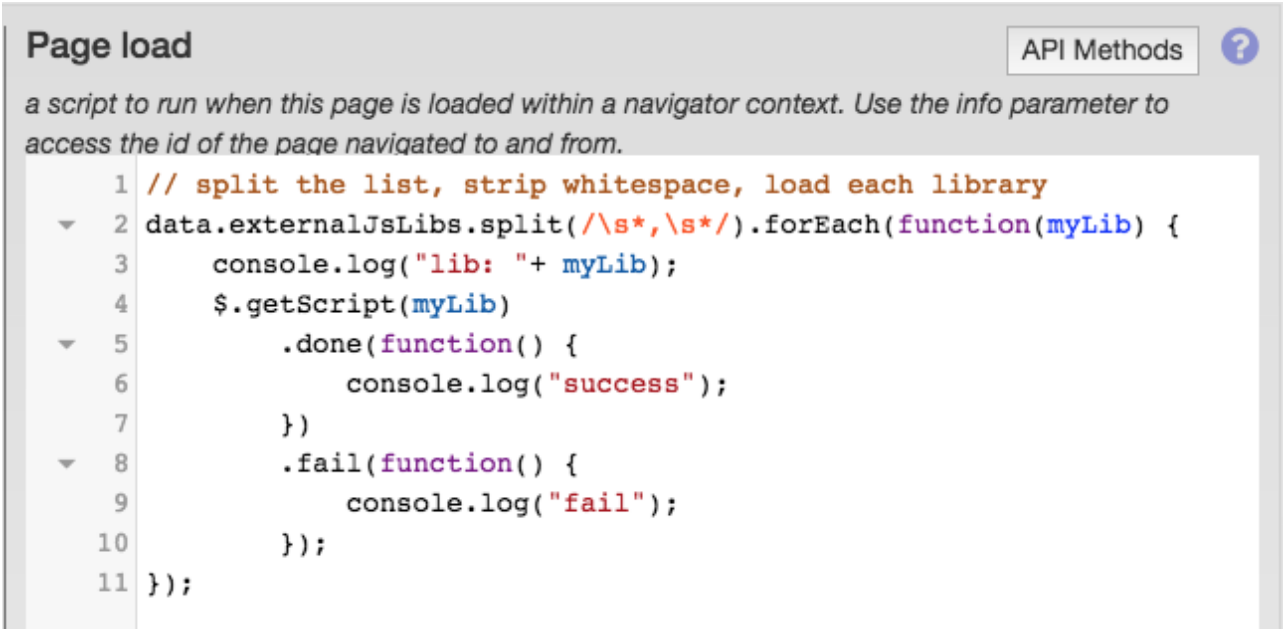
One benefit of loading JavaScript libraries with jQuery, is that the developer has complete control over when the library is loaded. Specifically, the library can be loaded when it is required. Libraries that are related to a feature that is not used in the form never need to be loaded. This obviously decreases the size of the Maestro application footprint.

Another benefit of using this approach is that the path and names of the Javascript libraries can be environment-specific. Many supporting libraries have different production and non-production URLs.

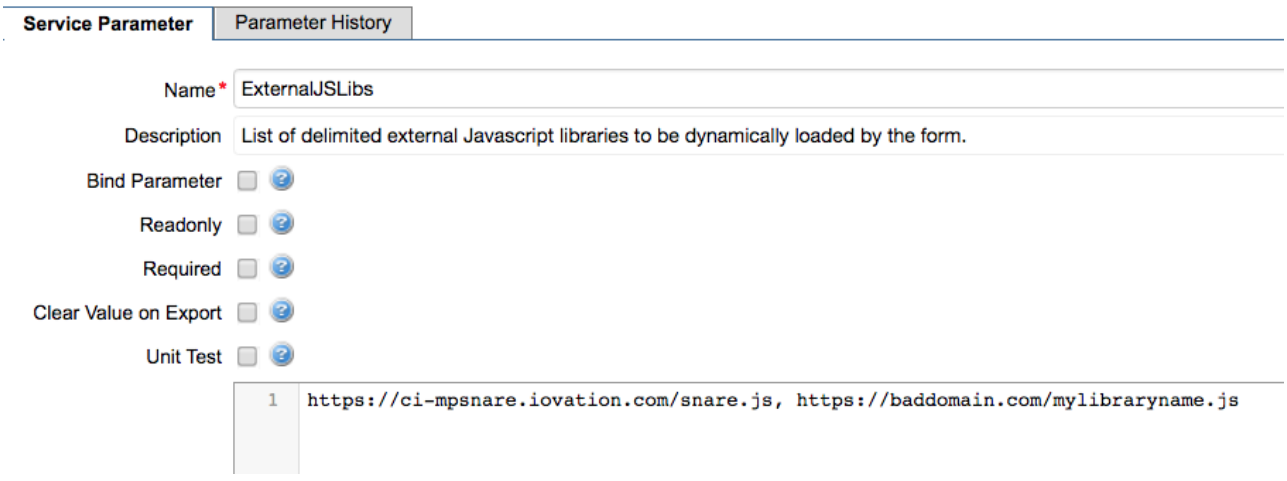
1. Add a data field into the page content of your Maestro form.
2. Go to the Data tab and toggle Include in Submission Data to ON.
3. Set the XML Location and the XML Name, as shown below:



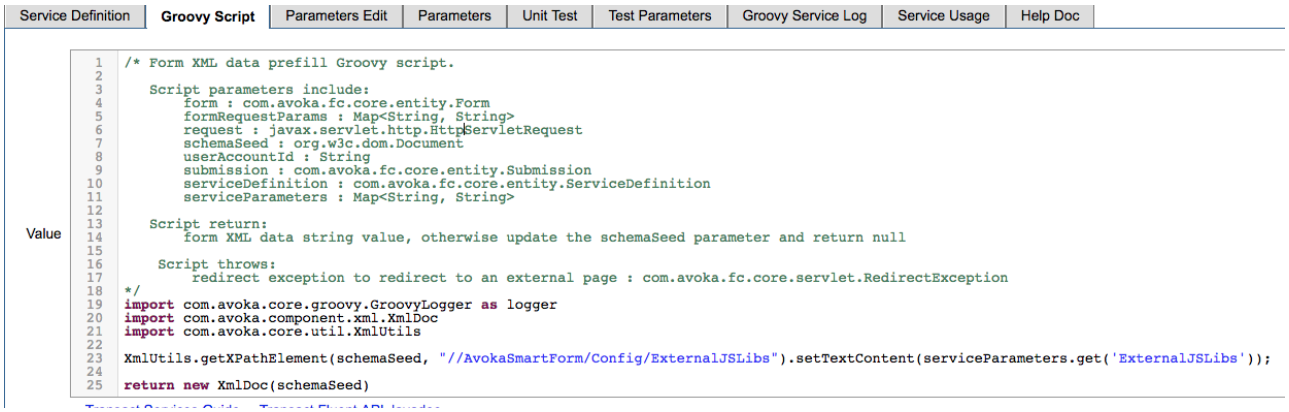
4. Create a rule to dynamically load external JavaScript libraries, for example a Page load rule as shown below:



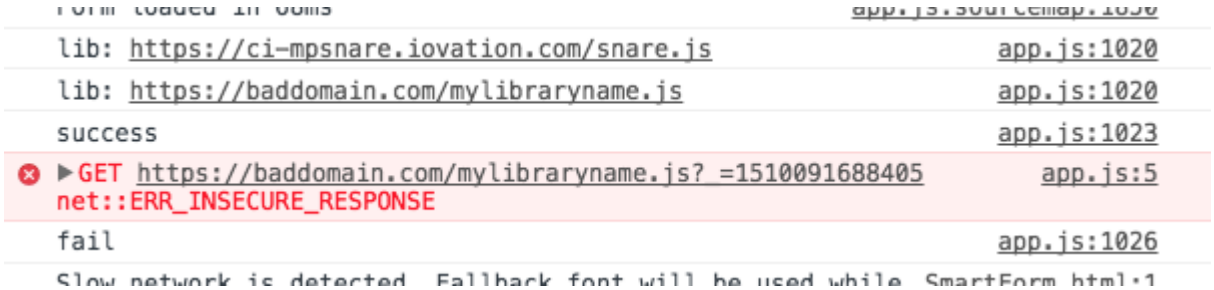
5. Note that loading libraries with jQuery provides success and failure handlers. A success handler can be used to invoke a function in the loaded script. Failure can be used to notify the user.
6. Log in to Transact Manager and create a Form Prefill service.
7. Create a service parameter - ExternalJSLibs, set its Type to String, and provide a delimited list of JavaScript resources, as shown below:



8. Write a Groovy script that reads this service parameter and writes it into the Form XML location identified above.



9. Save the script. Associate this Prefill Service with the Maestro form configuration in Transact Manager.
10. Render the form. Inspect the Console and the Network for the libraries loaded with Page Tracking and jQuery. Note the success and fail messages in the console. One library loaded; the other did not.



Next, learn about library scopes.

Library Versions

MaestroThe UI design product. | Form Builder | 18.05 This feature was introduced in 18.05.

Journey Maestro allows you to create multiple versions of each library, so you can achieve the following:

- Standardize versioning of Maestro's design artifacts.
- Roll back to previous versions of a library.
- Make changes to a library version without affecting organizations or projects that rely on an older version of the library.

Library Version Numbers

Each new library version has a unique version number. The recommended versioning approach uses version numbers in the format MAJOR.MINOR.PATCH. For example, the first version of a library might have a version number of 1.0.0.

You may have a preferred or mandated scheme that defines how version numbers are allocated. In addition such a scheme, consider the following when you need to select a new library version number:

- If you create a new library version based on a tagged version, it may be appropriate to increment the first (major) part of the version number by one. For example, a new version based on tagged version 1.0.2 might become 2.0.0.
- By default, when you create a new library version, the third (patch) part of the version number will increment by one.
- A version number can include a suffix that contains numbers and letters. Adding a suffix may be useful to indicate non-obvious but important information.

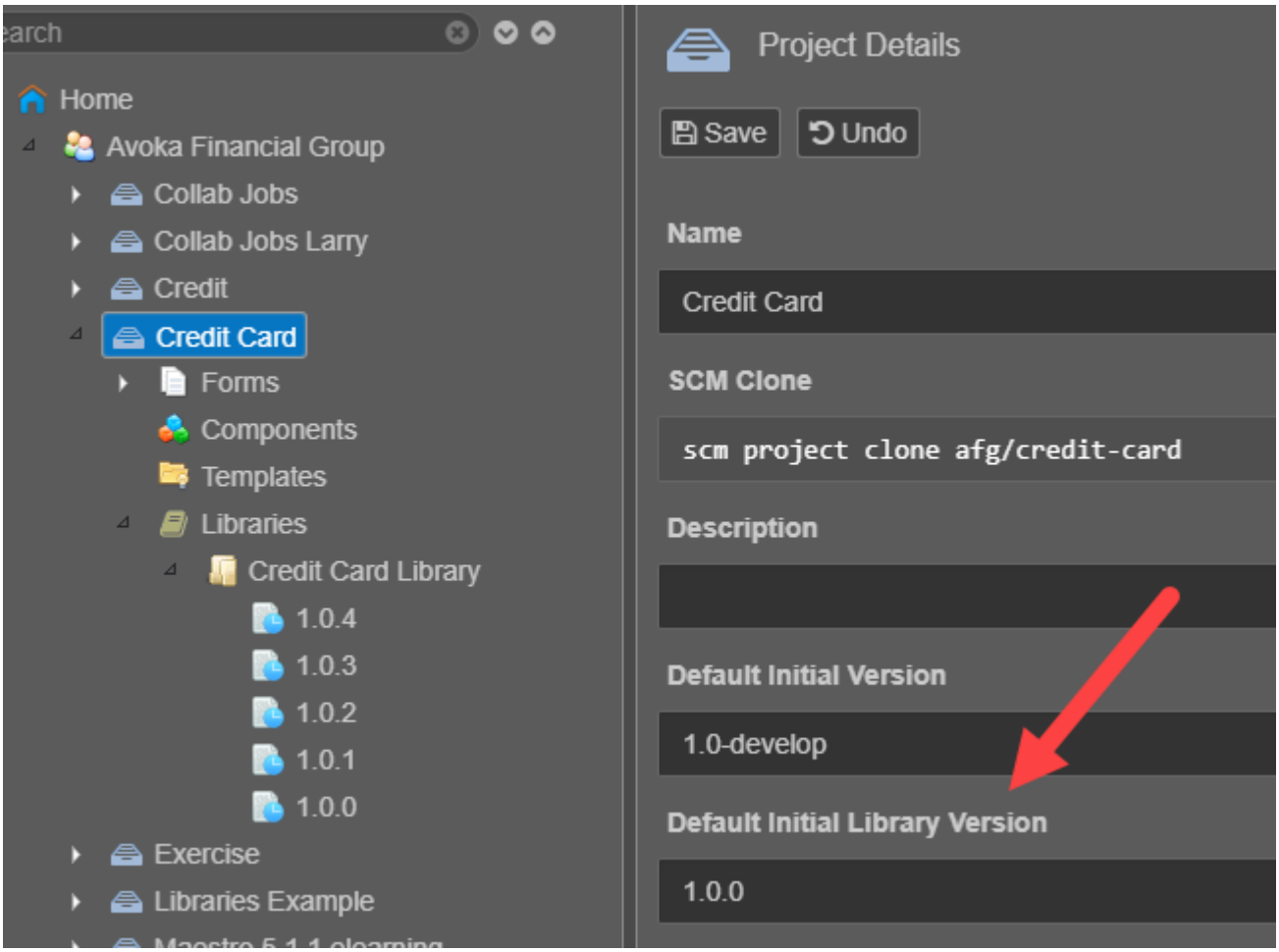
Viewing Library Versions

To view all versions of a library, select a library from the Navigator pane. The versions will be sorted based on their version number, in descending order meaning that more recent library versions appear at the top of the list. For example, version 1.0.2 of a library appears higher in the list than version 1.0.1.

Version	Description	Modified By	Last Modified	Created By	Created At
1.0.4				sjones	13 Dec 2018
1.0.3				sjones	13 Dec 2018
1.0.2				sjones	13 Dec 2018
1.0.1				sjones	13 Dec 2018
1.0.0	Default Library for Project Credit Card			sjones	8 Oct 2018

Default Initial Library Version

When a library is created, a default version number is assigned to it. All libraries created for a project start at the same version, The default version number assigned to a library is the Default Initial Library Version which is defined in the Project Details and has the default value "1.0.0".



To change the Default Initial Library Version:

1. Select a project from the Navigator and open the Project Details tab.
2. Modify the value in the Default Initial Library Version field.
3. Click Save.

After making this change, all libraries created for this project will start with this newly defined version number.

> NOTE
>
> The recommended value for the Default Initial Library Version is "1.0.0".

Next, learn how to create a library version.

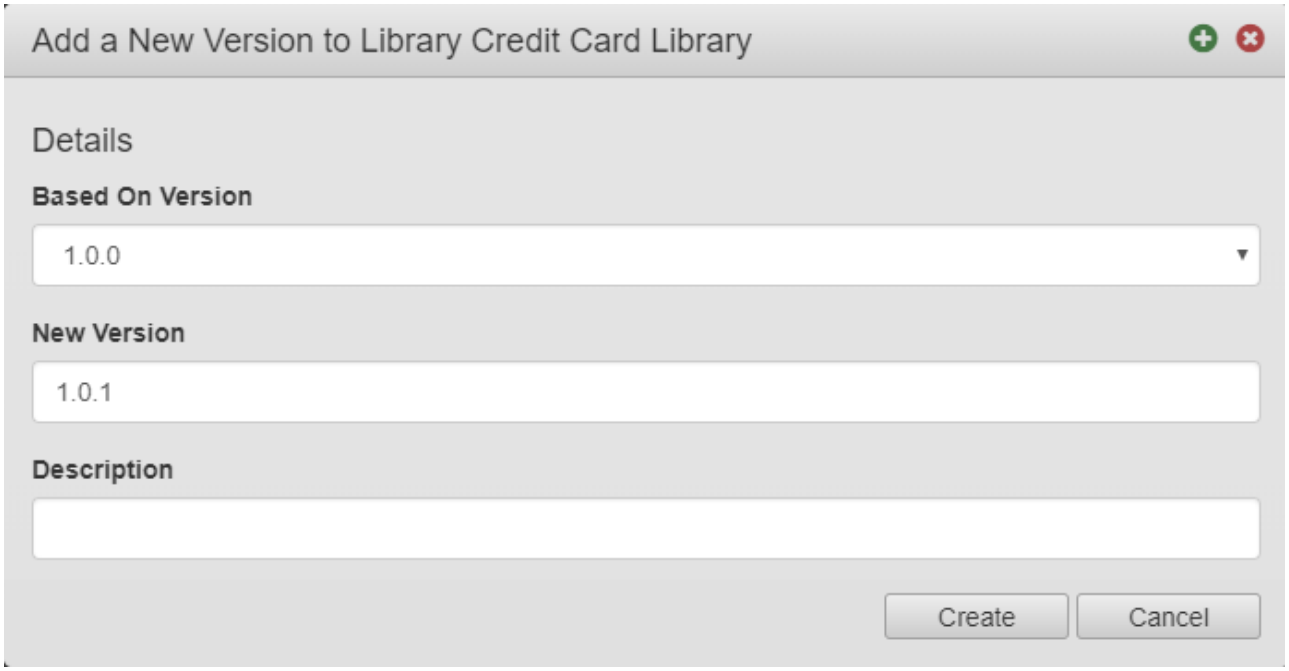
Create a Library Version

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro allows you to create new versions of a library so you can safely change and use them in your projects without any risks of negatively impacting other projects that relay on the same base library. Therefore, every Maestro library should be versioned. Usually, the default version number is `1.0.0`, but it can depend on the Default Initial Library Version.

To create a new library version:

1. Navigate to the Libraries folder to view all libraries for an organization or project.
2. Select a library you want to create a new version for.
3. Click New Version, which displays the Add a New Version to Library modal window.



4. Select a version number from the Based on Version dropdown list, which is used as the basis for the new version. The default value of this field is the most recent version. For example, if you select version `1.0.1`, the assets and resources corresponding to version `1.0.1` will be copied into the new library.
5. Provide a new library version number or name in the New Version field if you are not happy with the default version number, which is an incremented number of the previous version number.
6. Enter a library description in the Description field to make it easier to distinguish between

different versions of the same library.

7. Click Create. Check the new library appears in the organization.

Next, learn how to delete a library version.

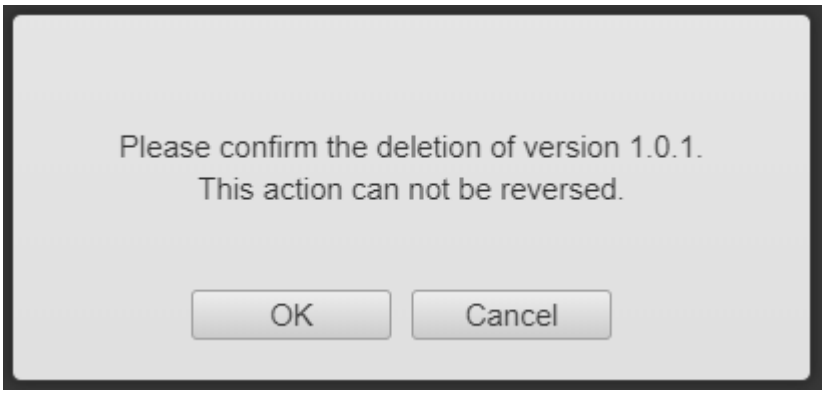
Delete a Library Version

MaestroThe UI design product. | Form Builder | 18.05 This feature was introduced in 18.05.

Library versions can be deleted at both the organization and project level. Note, however, that only one library version can be deleted at a time.

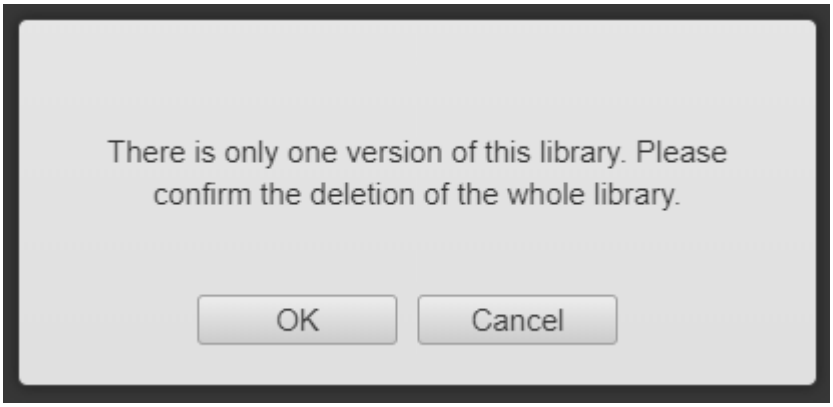
To delete a library version:

1. Select the Libraries folder for a project or organization.
2. Select a library then select the version of that library that you want to delete from the library list, and click Delete Version.
3. What happens next depends on how many library versions there are.
   - If there are multiple library versions, a message is displayed asking you to confirm that you want to delete the selected library version:

     Click OK to confirm the deletion (or Cancel to keep the library version).

   - If there is only one version of a library, a message is displayed asking you to confirm the deletion of the entire library. This is because a library must have at least one version.

Click OK to confirm deletion of the whole library (or Cancel to keep the library version).

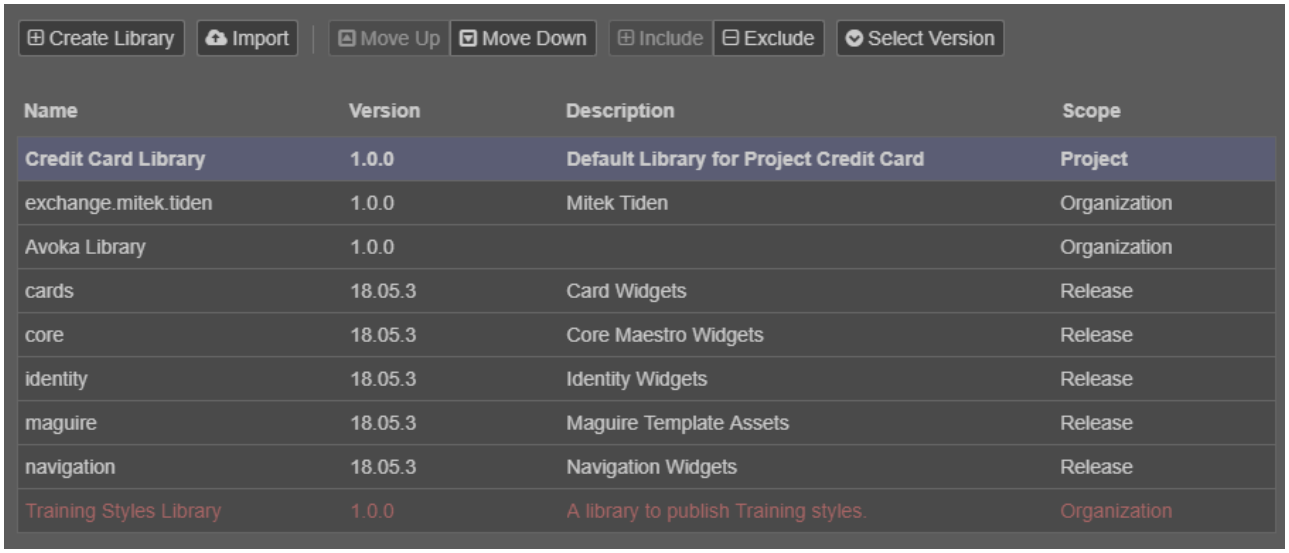
Map a Library Version to a Project

MaestroThe UI design product. | Form Builder | 18.05 This feature was introduced in 18.05.

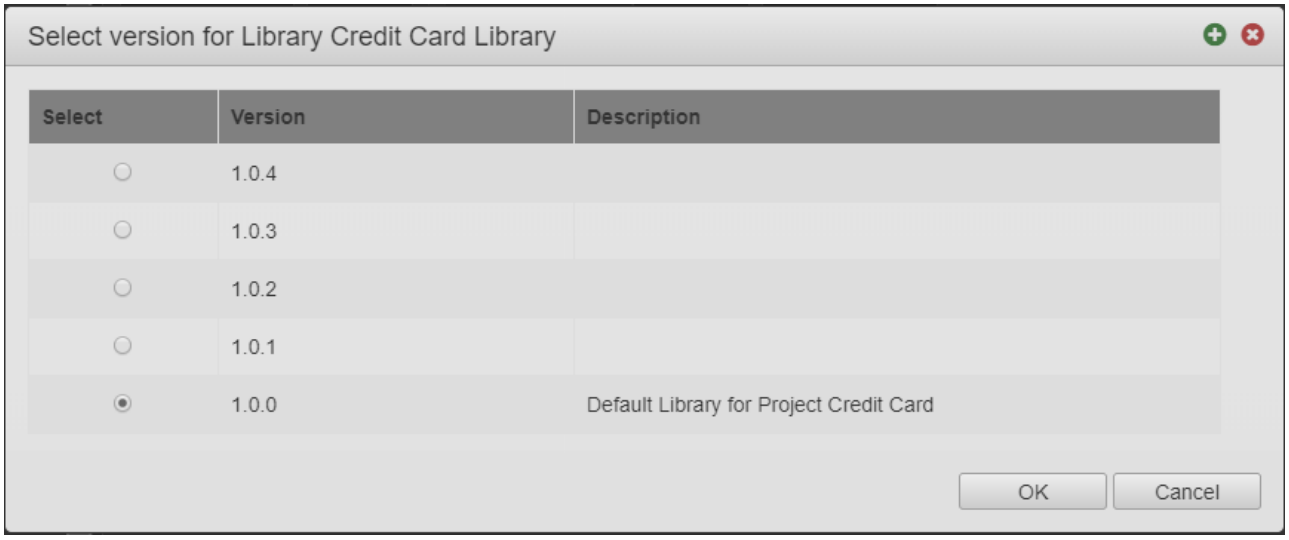
Journey Maestro allows you to specify which version of a library you want to use in a project. When you have multiple versions of the same library, you might not necessarily want the most recent version of the library to be used in the project, so you can map a desired library version to your project.

To map a library version to a project:

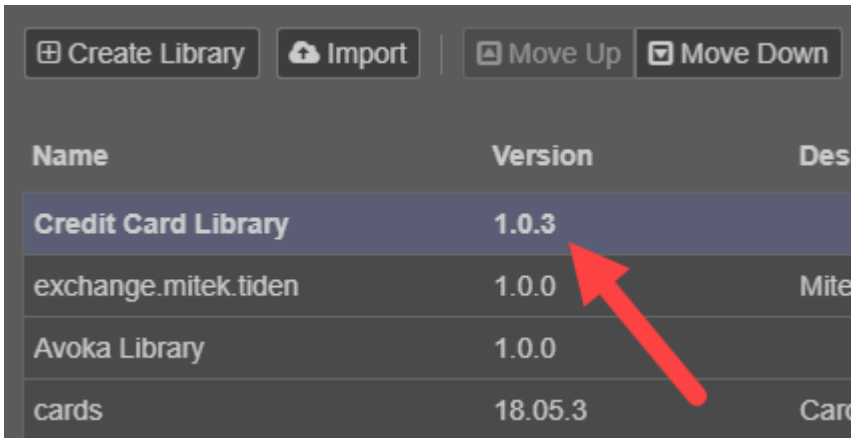
1. Open a project and select the Libraries folder.
2. Select the library that you want to map a version of to the selected project and click Select Version.



3. Select a library version to be mapped to the project from the Select Version for Library modal window.



4. Click OK. The selected version number now appears against the library in the project's library list, as shown below.



Next, learn about how to create a new library version based on a tagged library.

Create a Tagged Library

MaestroThe UI design product. | Form Builder | 18.11 This feature was introduced in 18.11.

As a general good development practice, you will want to have known good quality versions of all libraries that Form Builders and others depend on to do their work. Having good quality library versions provides these users with a level of confidence that any issues they encounter are likely to be caused by what they are doing, not by the dependent libraries they rely on. Dependent library versions, therefore, need to be immutable, and tagging a library version satisfies this requirement.

Maestro allows you to create and use tagged libraries.

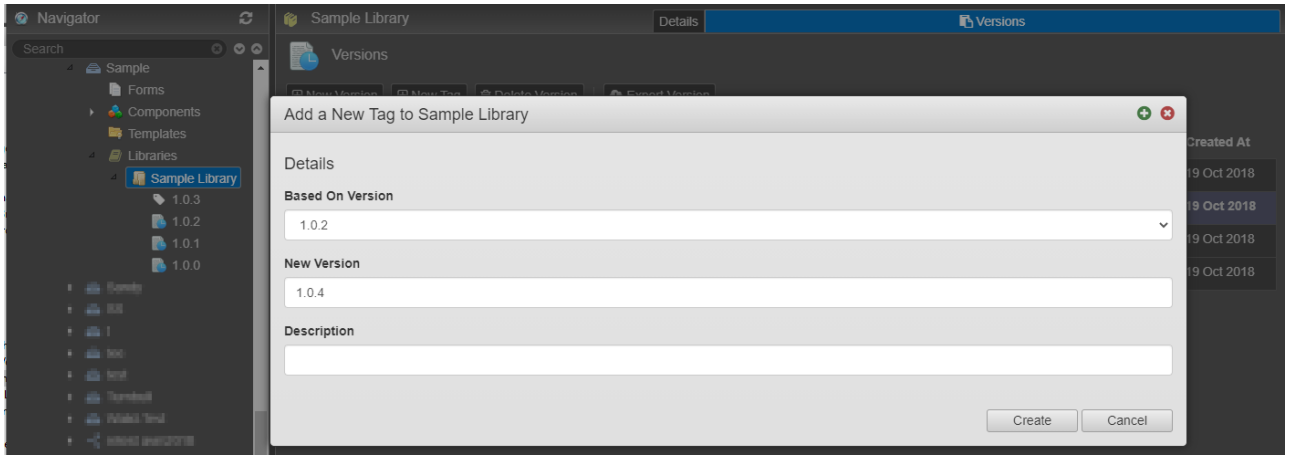
A tagged library is a read-only library version that prevents any changes from being saved to the library. Tagged library versions provide an effective way of keeping a clean, final record of a library. Once a tagged library is created, it is still possible to create new library versions based on the tagged version.

There are a few things you need to know when you work with tagged libraries:

- You can't publish assets to a tagged library.
- Only Administrator can delete a tagged library.
- If you create a library based on a tagged library version, the new library is editable because it isn't a tagged library version.
- You can export, view, and use tagged libraries as the library version in your project.

To create a tagged library:

1. View the available library versions in the Navigator pane.
2. Select a library version you want to tag and click New Tag. The Add a New Tag modal window appears.

3. 

4. Select the version of the library you want the tagged version to be based on from the Based On Version field.
5. Enter the New Version for the tagged version, and optional Description. For information about library version numbers, see Library Versions Overview.
6. Click Create. Check the tagged version of library is created and there is a check against this version in the Tag column. Also, if you open the tagged library version, you can see a tag icon at the top of the form next to the library name.

Once you have created a tagged version of a library, you may need to create a new version of the library based on this tagged version.

To create a library version based on a tagged library version:

1. Navigate to the Libraries folder within a project.
2. Select a library to view the existing tagged library versions, that is a check against this version in the Tag column.
3. Select a tagged version and click New Version.
4. Complete the Add a New Version modal.
5. From the Based On Version drop-down list, select a library version that you want the new version to be based on.
6. In the New Version field, enter a new library versions number, for example, `2.0.0- develop`.
7. Enter a description into the Description field to help organize the different versions of the library.
8. Click Create.

Next, learn about library precedence.

Maestro Forms

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A Maestro form is a web document designed to capture user's data.

NOTE

In this documentation, we use the following terms interchangeably: form, application form, application, user application and customer application.

You create a form in Journey Maestro using various components and libraries. A form design (look and feel) is usually based on a template or a brand that significantly simplifies your life as a Ford Builder. A Template Designer creates a template, which helps you build forms following a specific design. First, you create a form for individual purposes and then extend it and add extra functionality to it by creating new form versions of the original form. Maestro forms are built on the AngularJS[1] framework.

When the Maestro form is ready, you build it as a Form Archive (FAR file), which is a self contained ZIP file with all generated resources in it. Now, you can publish (deploy) it in various Journey Manager environments to make the form available to form users and applicants.

As forms require various types of data, including user data, dynamically, they interact with Manager. This means you have to configure Maestro forms in Journey Manager before they can be used.

The Maestro editor is hosted on a Manager instance and the version the project uses dictates which version of the editor the form opens in. This allows you to edit your form in the same version you have used to edit your forms to help manage changes to your forms so making a simple text change will not get new features from newer versions. For more information, see Maestro features.

[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/

NOTE

A Maestro's form is represented differently in Manager. For more information, see Manager's form overview.

Forms are tools for capturing data from applicants (form users). They typically contain page groups, pages, sections, and fields to guide applicants through the data capture process. A Form moves through various events of the Customer Journey Experience, usually configured by a Business Analyst (BA) or Subject Matter Expert (SME) based on requirements gathered. We refer to the BA or SME as the Form Designer.

A Transaction Experience is a sequence of events that includes Prefill, Data Capture, Save and Resume, Attachments, Payments, Assignments, Notifications, and Collaboration between parties.

Some of these events have an impact on the form design, while others are configured in Manager.

Organizations frequently have high-level concerns before beginning form development, including:

- Branding
- Integration
- Accessibility
- Security

These concerns are translated into action during a Form's development that can include:

- Styling
- Validation
- Data Structure
- Usability

Maestro provides the tools to address these concerns using the UX principles embodied in the software.

Although the same interface is used to create templates and forms, a Form Builder will likely have less control over the appearance of a form than a Template Designer has over a template. This is because Template Designers typically define extension points, which prevent Form Builders from introducing inconsistencies across an organization's forms.

When a form is published, all of its assets, such as templates, images, style, components, and so on, are bundled into a single-page AngularJS application and made available via a form space.

When creating a form, a Form Builder will:

1. Choose what template the form should be based on
2. Apply a brand to the form (if more than one brand exists)
3. Add page groups, pages, sections, and fields to the form
4. Create validation rules and other business logic
5. Integrate with third-party services via pre-built Exchange packages
6. Configure what data will be collected via Journey Analytics

Next, learn about a form architecture.

Maestro Form Architecture

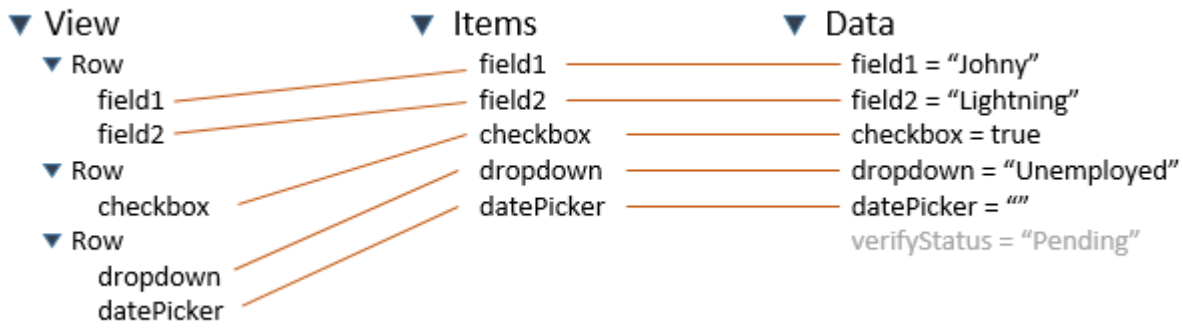
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A Maestro form is a web document, also known as a web application, designed to capture user's data and perform some synchronous or asynchronous actions. In order to do so, each form is built based on the certain architecture. When you create a form in the Maestro Editor, it applies these architectural principles for you, so you have to understand them to use the tool efficiently.

The architecture of a form consists of the following main elements:

- Form View - a hierarchical structure that defines the visual layout of the form.
- Form Items - each component, such as a Field or a Button, or container, such as a Block or a Section, within the form structure is considered an item. The Form Items is a list containing every item in the form.
- Form Data - a map of key / value pairs. Most data elements are linked to a form item, such as a Field, but data elements can also exist autonomously.

These 3 elements are all closely related. Form items appear in the Form View and have Data elements associated with them, which is show below:



Let's examine each of these elements in details.

Form View

The form view is a hierarchical structure that consists primarily of horizontal rows, each containing one or more components. The hierarchical structure represents the layout of the form and all components within it.



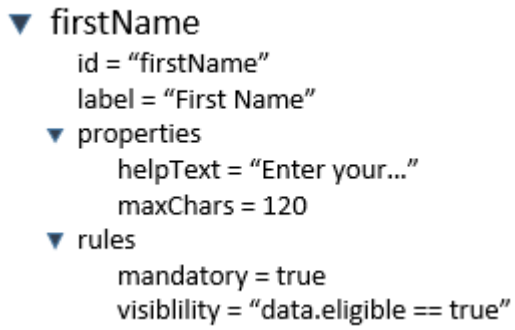
Being hierarchical, rows in the view can be containers for lists of more rows in a recursive manner.

Form Items

Each form item in Maestro is uniquely identified by a field Id. Maestro generates these field Ids based on the name of the field but also allows you to override the generated value and manually specify a field Id so long as they are unique in the context of the form.

Having guaranteed unique field Ids for all items within a form allows us to have a flat list (array) representation containing all items in the form, making access to these items much simpler than having to navigate a view hierarchy to find them.

Items can contain UI elements, but also have properties and rules associated with them. By way of demonstration, consider a mandatory first name field that has a maximum length of 120 characters and a visibility rule:



Form Data

All data in the form is stored in a single object called the Form Data object. Like form items, each form data element must have an id that is unique in the context of the form. If a data element

exists with the same Id as a form item, that form item will read and write its data to that data element. Data elements can exist without being bound to a form item.

Depicted below is 4 data elements, 3 of which are bound to fields (items) and one that is unbound (verifyStatus). The middleName data element is blank suggesting that nothing has been entered into the middleName field.



Data elements that are not bound to form items (verifyStatus in this example) will not be Saved/Submitted with the form data and so only exist for the duration of the session.

> NOTE
>
> The Data Field component type can be used to bind data elements without having to display them in the visual layout so that they get persisted with the form data.

Next, learn how to open a form.

Create a Form

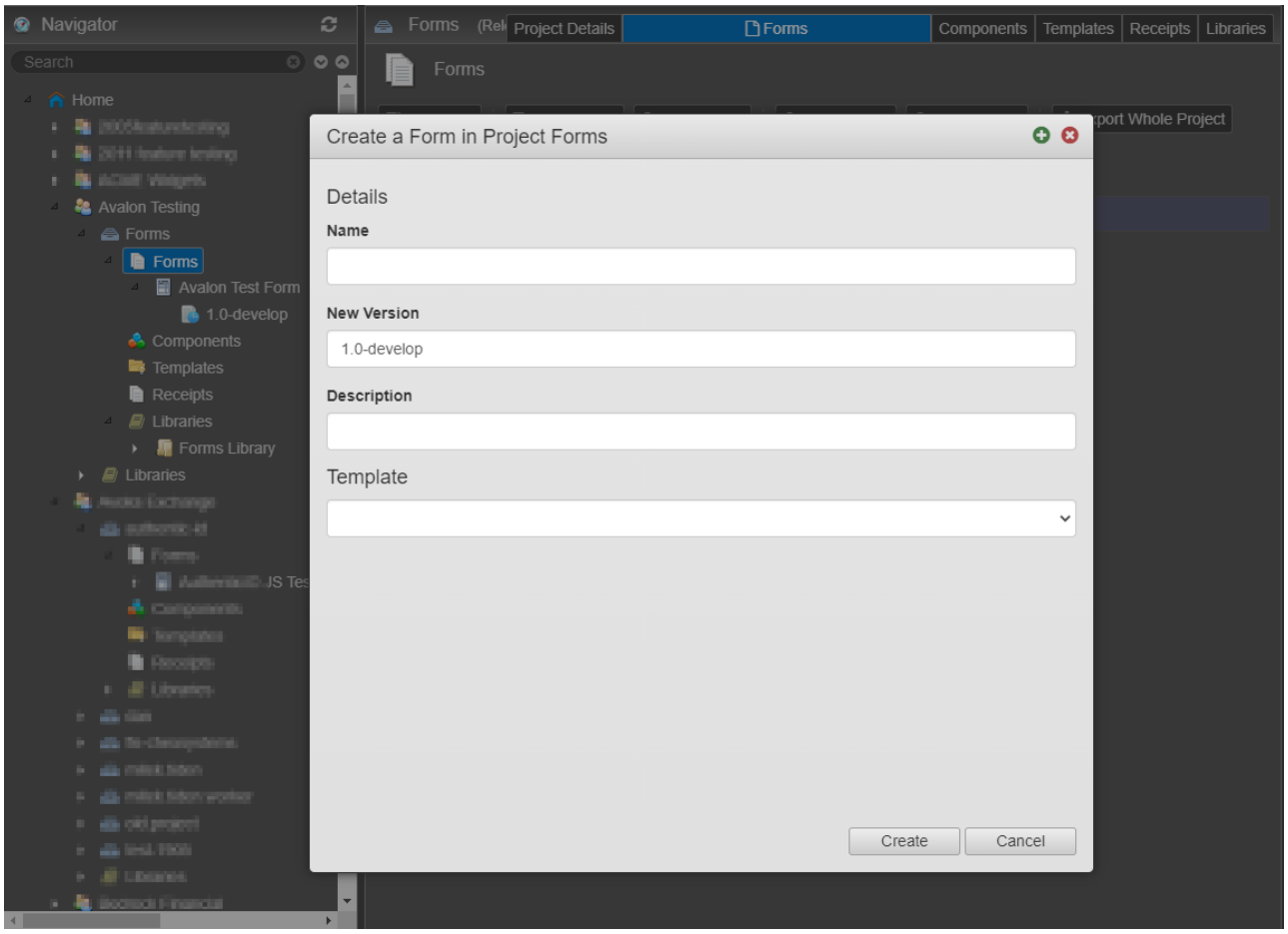
MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Journey Maestro allows you to create new forms to implement various customer journeys in your applications. This is the first step in creating an onboarding user journey.

> NOTE
>
> As most forms are based on templates, we recommend checking how to create a template as well.

To create a new form:

1. Open a project from the Management Dashboard.
2. Select Forms and click Create Form.



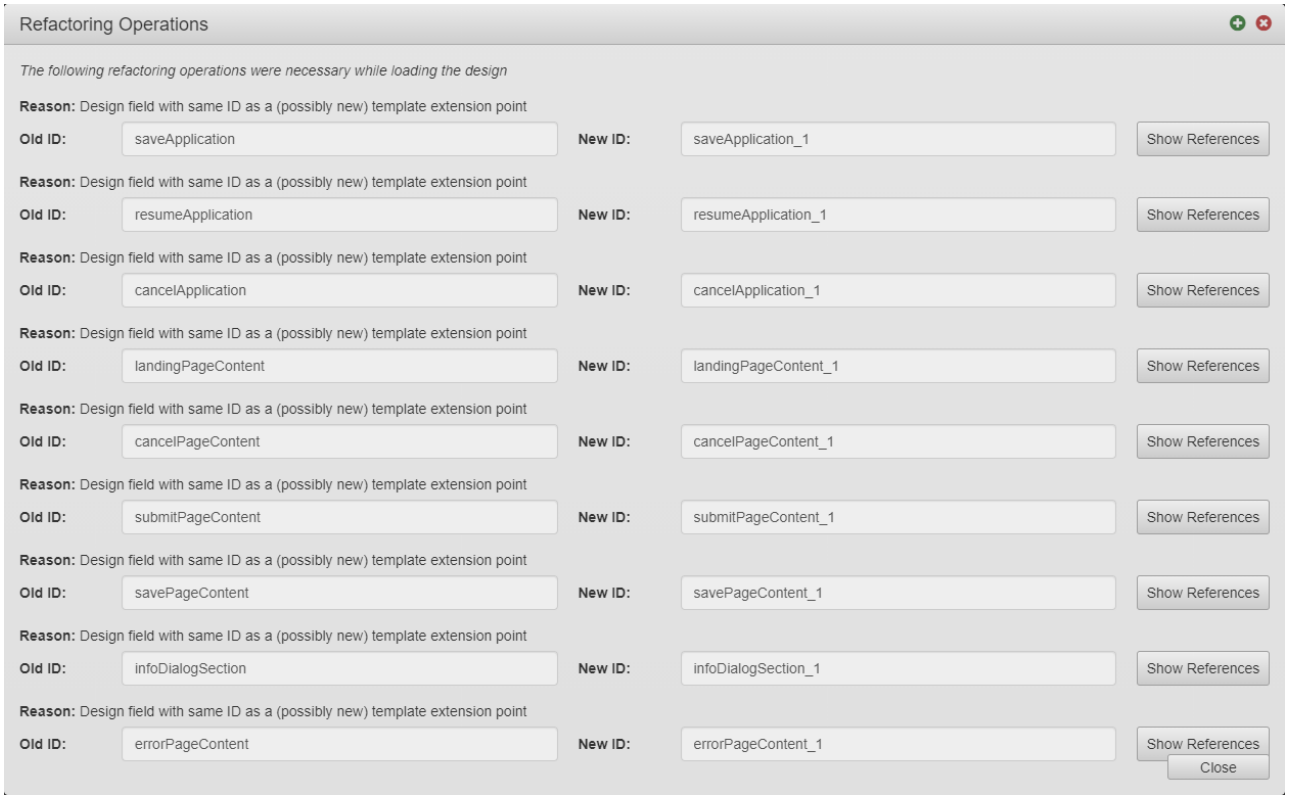
3. Enter a form name in the Name field.

4. Change a system generated version in the New Version field to adjust it to your requirements. Maestro uses semantic versioning.
5. Provide a meaningful description in the Description field so it's clear for others what this form is for and what it does.
6. Select a template from the Template dropdown list. Maestro comes with the following build-in templates:
    - Avalon Form Template | 21.05 This feature was introduced in 21.05.
    - Avalon Receipt Template | 21.05 This feature was introduced in 21.05.
    - Core - Empty Form
    - Core - Empty Receipt
    - Maquire - Default Form
    - Maquire - Default Receipt
    - Maquire - Empty Form
    - Maquire - Empty Receipt

    NOTE

    You should always use a custom template created for your business or project. You should never use the built-in templates to create forms. A Template Designer should use these built-in templates to create new templates for your specific purposes. If you can not find a custom template available in the dropdown list, speak to your Template Designer before creating your form.

7. Click Create.
8. Check a list of refactoring operations Maestro has performed to create a new form and click Close.



Maestro opens the newly created form in the Maestro editor in a new tab in your browser.

> NOTE
>
> If the form does not open after you click Create, check your browser's pop-up blocker settings. When you first open a form or other assets in Maestro, you need to enable the pop-up by selecting a relevant option, for example, the Always allow pop-ups from. Form more information, see help documentation on your browser.

Next, learn how to add components to a form.

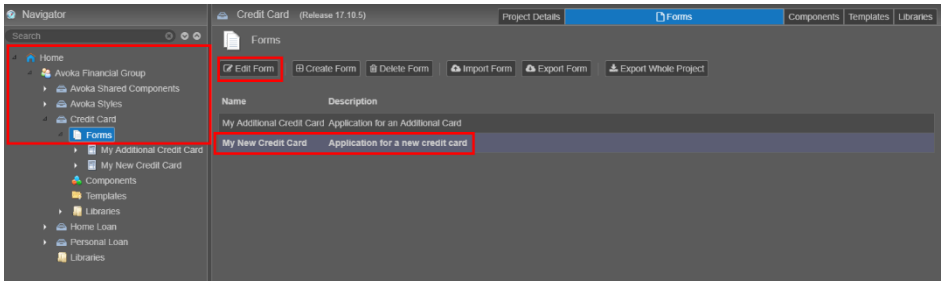
Open a Form

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

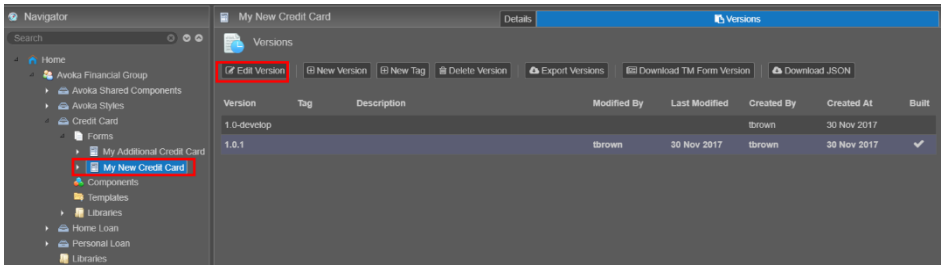
While browsing forms in the Maestro Dashboard, you may want to open a Maestro form in the Maestro editor to check it or make some changes to it.

To open a form:

1. Log in to Maestro to access the dashboard.
2. Go to the Navigator pane.
3. Select Home > Organization > Project > Forms.
4. Select a form from the Forms tab and then click Edit Form.



Alternatively, select a form version and then click Edit Version.



5. Edit the form as required.

Next, learn how to save a form.

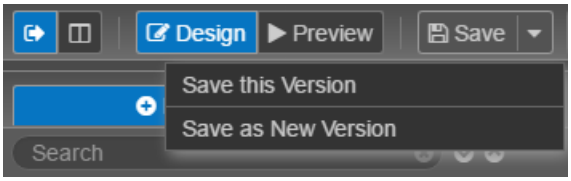
Save a Form

MaestroThe UI design product. | Form Builder | 17.10 This feature was updated in 17.10.

Maestro doesn't automatically save a form while you are working on its content, so you must click Save regularly to persist your changes to the form.

To save a form:

1. Open a form in the Maestro Editor.
2. Edit the form as required.
3. Click Save to save the changes.

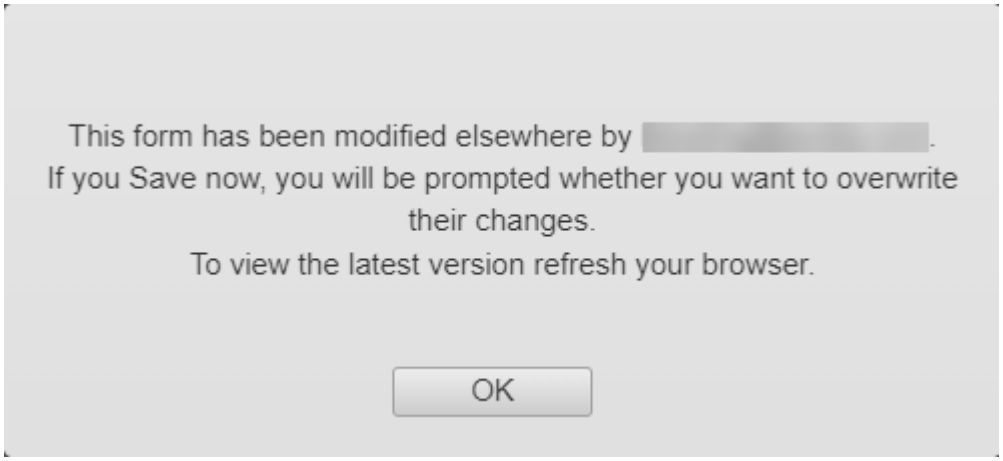


> NOTE
>
> The save button becomes enabled only if there have been changes to a form since it was last saved.

4. Select one of the following save options:
   - Save this Version - saves the changes to the currently open form version and it works in the same way as clicking the Save button.
   - Save as New Version - saves the changes to a new form version. You must enter a new version number and a description in the Save as New Version window.
5. Click Close to close the tab with the form.

Maestro comes with the save form notification mechanism. | 17.10 This feature was introduced in 17.10.

If you have saved a form while other Maestro editor users have the form open or have been working on it, they will see the following warning message:



Should any conflict arise while saving a form, the message displays the user name who has saved the form, so you can reach out to that user to discuss which changes should be saved or rolled back.

> NOTE
>
> When a user saves a form, Manager marks a form as `userSavedFlag = TRUE`, so this form submissions won't be abandoned.

Next, learn how to close a form.

Close a Form

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

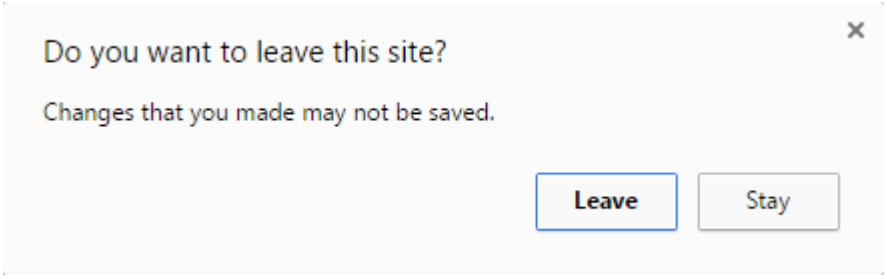
Maestro opens a form for editing in a new browser tab while the Management Dashboard remains open in its original tab. When you are finished editing a form, you can close the relevant tab and return to the Maestro Management Dashboard.

To close the form:

1. Close a browser tab you are editing the form in.



2. Review a prompt to save your changes, which is displayed if you close the tab without saving.



3. Click Leave to cancel any changes. This will close the form without saving.
4. Click Stay to save your changes and then click Save on the toolbar at the top of the form window.

Next, learn how to export a form.

Export Maestro Forms

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to import and export forms, so you can reuse them across different projects, organizations or environments. This saves you time and effort, as you don't need to create a new form from scratch. This process is also commonly used to migrate forms from one environment to another. You can use this functionality to:

- Rollback one or several forms
- Re-create one or several forms in another Maestro
- Promote one or several forms to other environments

To export a form:

1. Open a project in the Navigator pane where you want to export a form from and click Forms to view the list of forms created in your project.
2. Select the form you wish to export from the list and click Export Form. Maestro exports the form into a form archive ZIP file, which will be downloaded to your machine.

In some cases, you may need to export the whole project.

To export a whole project:

1. Open a project in the Navigator pane where you want to export a form from and click Forms to view the list of forms created in your project.
2. Select the form you wish to export from the list and click Export Whole Project to export the project including all assets and resources. Maestro exports the project into a form archive ZIP file, which will be downloaded to your machine.

Maestro lets you export a form version individually. You can later import the exported form version using the import form functionality described below.

Next, learn how to import forms.

Import Maestro Forms

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to export and import forms, so you can reuse them across different projects, organizations or environments. This saves you time and effort, as you don't need to create a new form from scratch. This process is also commonly used to migrate forms from one environment to another. You can use this functionality to:

- Rollback one or several forms
- Re-create one or several forms in another Maestro
- Promote one or several forms to other environments

To import a form:

1. Open a project in the Navigator pane where you want to import a form to and click Forms to view the list of forms created in your project.
2. Click Import Form and choose a form archive ZIP file.

   NOTE

   You can import a single form version, which was exported using the Export Versions functionality, only if there is no existing form version in a project.

   NOTE

   If you have imported a form to the same organization and the project, the form name will have the same name as the original but with an added suffix, for example, "-1", to differentiate it.

   In case of an error during the import, you can choose one of the following options to proceed:

   - Click Create New to create a new copy of this form.
   - Click Update Existing to update an existing form with the same name.
3. Select the imported form in the Navigator pane and click Details to update the Name and Description of this form.

Next, learn how to migrate a form.

Migrate Maestro Forms

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to migrate and copy forms across different projects, organizations or environments. You can use this functionality to:

- Rollback one or several forms
- Re-create one or several forms in another Maestro
- Promote one or several forms to other environments

Migrating a form is merely using the Maestro export and import functionality, which is explained below.

To migrate a form:

1. Export a form
2. Import a form

Be aware that when you move a form, the resources used within that form will remain in the original project. If you move the form on its own, you may notice the appearance does not match the original and that there are resources missing. This would include the template used to create the form, any shared components, and styles.

To ensure the form displays as originally designed, you need to move the assets used to the new project along with the form. This can be done by moving the individual assets to the relevant project. Another option is to move the library, which contains the published assets and resources. You can move the library to a new project or an organization library where all projects have access to the published assets and resources.

NOTE

It is recommended that you speak to the Template Designer before importing and exporting templates, styles, and libraries.

Forms can also been exported and imported in Manager. For more information, see migrate forms in Manager.

Next, learn how to test forms.

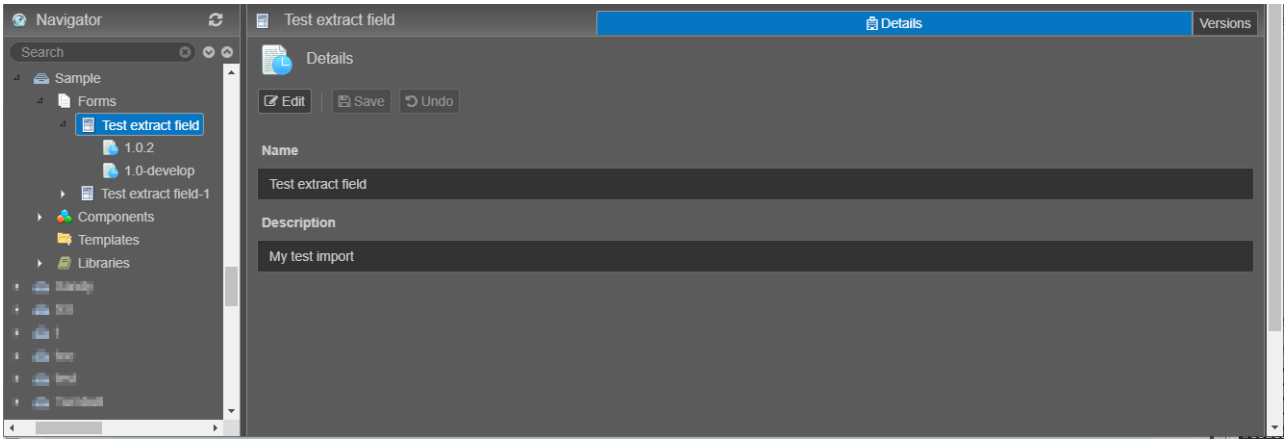
Change a Maestro Form Description

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro allows you to change a description of a form, which you may need to do after you have created a form. You can also change a form version's description at any time after the form version is created.

To change a form's description:

1. Open Maestro and locate a form to see its current description.
2. Select the Details tab.



3. Update a form's description in the Description field.
4. Click Save to update the changes.

Next, learn how to open a Maestro form.

Test Maestro Forms

MaestroThe UI design product. | All versions This feature is related to all versions.

Testing is an essential, yet often under-appreciated, part of the form development process. A well-tested form demonstrates an attention to detail that reflects well on an organization. An untested (or poorly tested) form is likely to provide a frustrating and revenue-reducing customer experience.

To test Maestro forms, organizations are encouraged to create a testing specification. A specification is a list of expectations the organization has about how their forms should look and behave. For example, the organization may expect some fields to be mandatory, certain information to be available, and back-end integrations to operate in a particular way.

Test Checklist

There are no universal expectations that can be applied to every form. There are, however, some expectations that will apply to most forms and can be used as a starting point for more specific expectations. These are listed below.

Functional

- When supplying valid data, you can complete the form from start to finish. (If you can't complete this test, the rest of the tests will be too granular to be useful. Ensure this test passes before continuing.)
- There are no errors or warnings in the JavaScript Console.
- Fields marked as mandatory are *actually* mandatory.
- Fields reject invalid data.
- Add and Remove buttons for repeating block components work as expected.
- You can enter data in an unusual order (from bottom to top, randomly, etc).
- After submitting a form, the Submission XML contains the data entered in the form.

Visual

- Fields are aligned correctly.
- Fields are arranged in a logical order (from left to right, top to bottom).
- Long labels or text don't cause wrapping or alignment issues.
- Dropdown lists are wide enough to fit the content they contain.
- Different types of fields align correctly when placed in the same row.
- Mandatory fields are identified as mandatory.

- Optional fields are identified as optional.
- Fields look consistent between pages.
- Spelling, punctuation, and grammar are correct.
- Error messages are relevant and easily understood.

Receipt

- Fields are shown or hidden (where appropriate).
- Receipt is flat and un-editable.

Accessibility

- You can navigate through a form using only the keyboard.
- Using the keyboard navigates the fields in a logical order.
- Images have `alt` attributes.

> NOTE
>
> For more information on accessibility, refer to Accessibility for Journey Maestro Form Developers.

Dynamic Data

- Dynamic data lookups retrieve requested data.
- Third-party integrations (such as identity verification) operate as expected.
- When integrations don't work, manual alternatives are available.
- Data is prefilled.
- Attachments can be uploaded.
- Forms can be signed digitally.

Devices & Browsers

- Layout responds to different view-port sizes.
- Fields are touch-friendly.
- All tests pass on primary browsers and devices.

Next, learn how to debug Maestro forms with Chrome DevTools.

Debug Maestro Forms with Chrome DevTools

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

If a Maestro form isn't behaving as expected, it might be related to one of the business rules you've created using the Maestro script editor. Tracking down problems in business rules can be tricky; this topic explains some useful techniques for finding and fixing problems using Google's Chrome DevTools.

NOTE

This topic only scratches the surface of what Chrome DevTools can do. To learn more, read Google's official documentation.

Prerequisites

This topic assumes you have experience with HTML, CSS, and JavaScript.

Maestro runs on the AngularJS JavaScript framework, so AngularJS experience will be useful if you'll be debugging on a regular basis, but is not required for this guide. To learn about AngularJS, read the official tutorial.

We recommend Google Chrome for debugging Maestro forms. It has a wide range of debugging tools and it's what we'll be using in this topic.

Building a Form

Before a Maestro form can be debugged, it needs to be built with some specific build options and then rendered.

NOTE

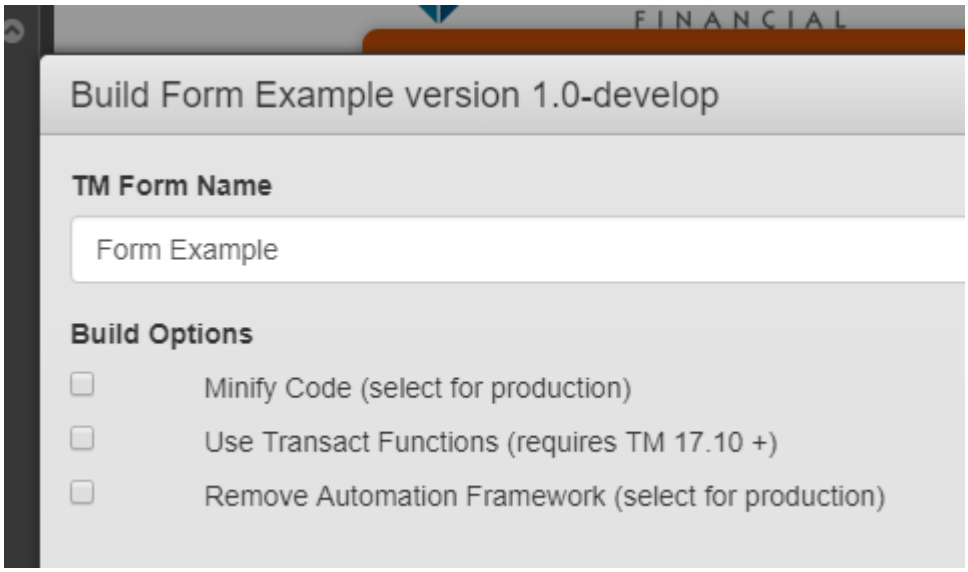
You cannot debug a Maestro form via the Design, Code, or Preview tabs.

To prepare a Maestro form for debugging:

1. In the Maestro Editor, click Build > Build with Options to display the Build modal.
2. Ensure the Minify Code checkbox is cleared. Selecting this option minifies the JavaScript in the rendered form, which compresses the code by removing unnecessary characters

such as white space, new lines, and comments. Minification is great for performance but interferes with debugging.

3. Ensure the Remove Automation Framework checkbox is cleared. Selecting this option removes the Maestro Automation Framework (MAF), which is used to test Maestro forms with tools like Selenium. However, the MAF also has some features useful for debugging, so it's best if we don't remove it until the form is ready to go live.



4. Click Build.
5. Click Render Form.

The form will open in a new browser tab. If a new browser tab doesn't open, check if your browser is blocking popups and allow them from the Maestro URL.

Whenever you make a change to the form, you'll need to rebuild it. If you're doing this a lot, you can streamline the process by using some keyboard shortcuts: `Control+P` (`Command+P` on Mac) to open the Build modal and `Control+Enter` (`Command+Enter`) to build the form.

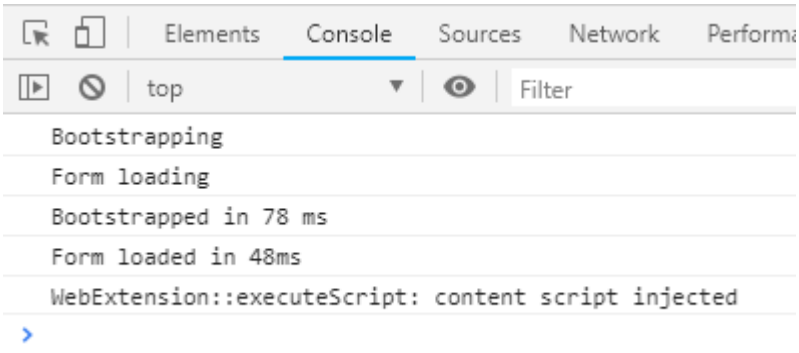
Opening Chrome DevTools

Chrome DevTools is a set of web developer tools built into the Google Chrome browser. These tools provide access to a range of features for diving into the look, functionality, and performance of web applications.

To open Chrome DevTools:

1. Select the browser tab that contains the rendered Maestro form.
2. Press `F12` or `Control+Shift+I`.

The DevTools pane will appear on the right or bottom edge of the browser window.



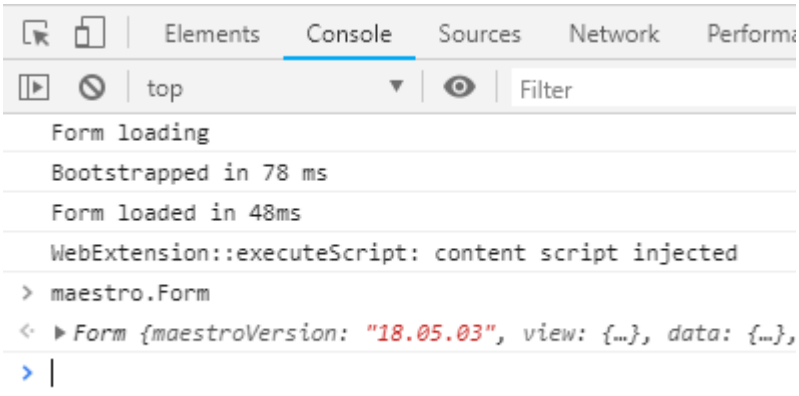
Inspecting the Form Object

When you load a Maestro form in your browser, a Form object is initialized that contains the data required to render the form. It's worth understanding the Form object so you know where to look when something goes wrong.

You can inspect the Form object in Chrome DevTools. To do this, open the Console tab and enter:

```
maestro.Form
```

This statement is available to us through the Maestro Automation Framework that we chose not to remove when building the form. When we execute this statement, the Form object is returned:



You can also access this data via jQuery:

```
$("Form").scope().Form
```

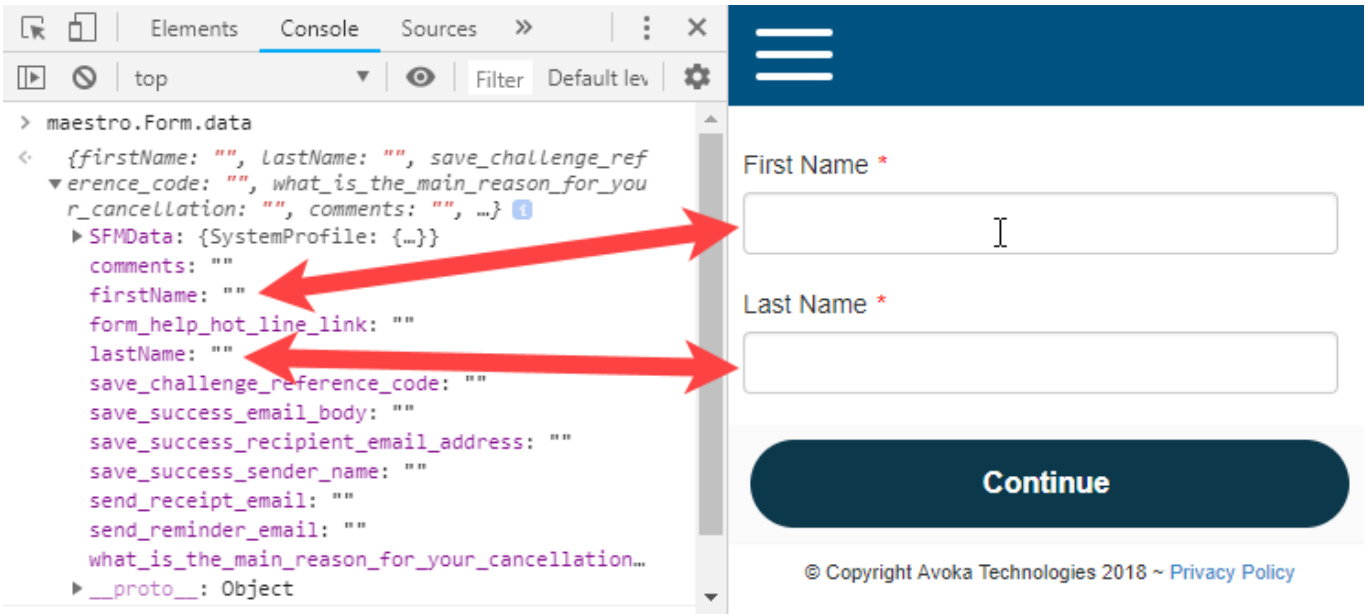
If you expand the Form object, you'll see that it's quite large. You don't, however, need to understand every part of this object. It's just useful to have a broad understanding of how this object describes the structure of a Maestro form.

There are three objects within the Form object that are worth exploring: `data`, `items`, and `view`.

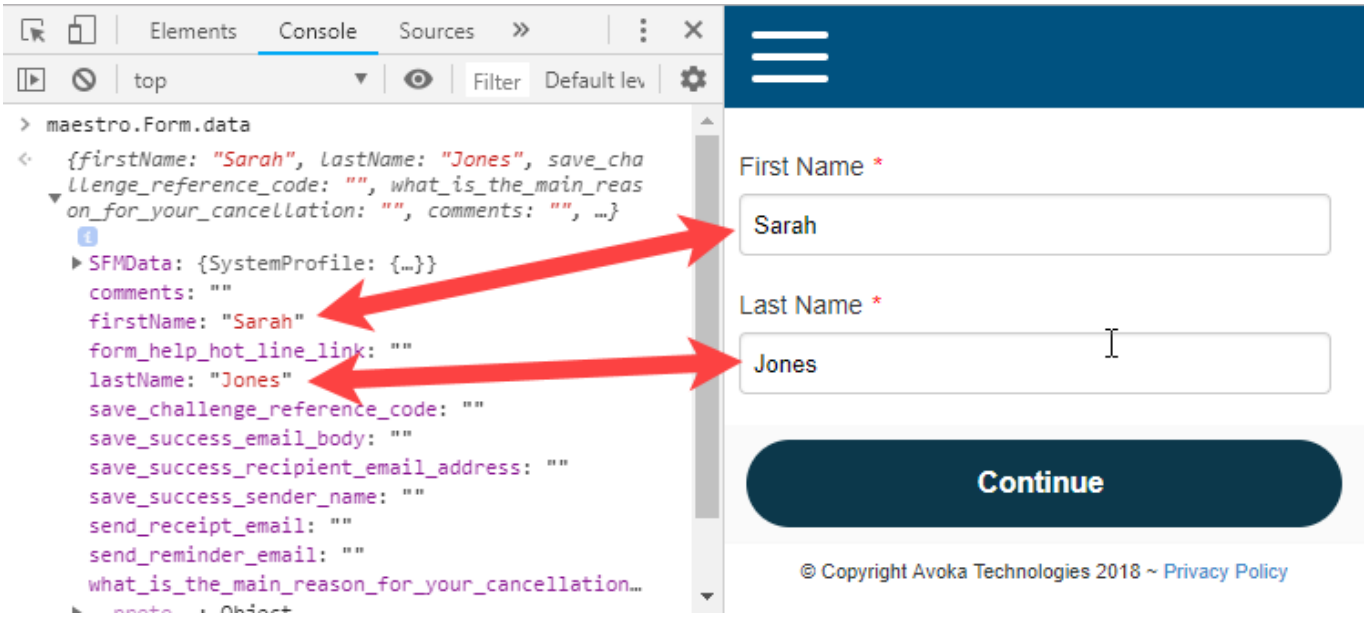
data

The `data` object describes the current state of the form's data. When values are entered in to the fields of a form, the `data` object is updated to reflect the new values.

For example, if you have an empty form, there won't be any values in the data object's properties:



But if you enter some values into the form, maestro.Form.data will contain those values:



items

The `items` object describes the components (such as fields and buttons) and containers (such as blocks and sections) within a form. Each component and container is considered an item.

For example, this is what the `item` object for the "First Name" text field looks like:

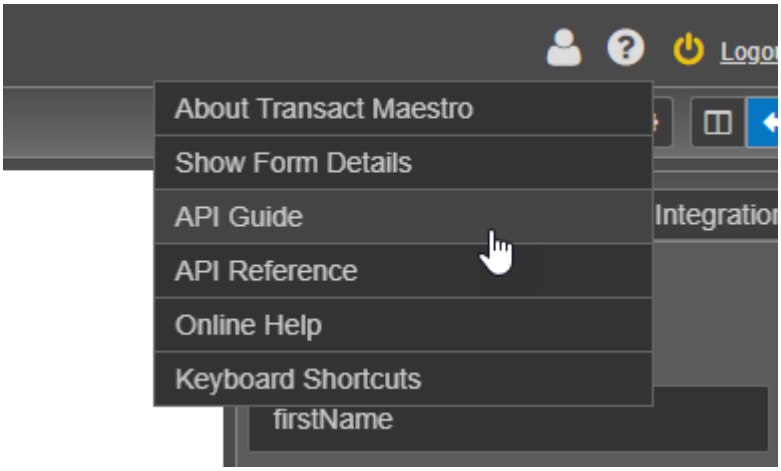
```
firstName: {
       $$parent: {span: {…}, offset: {…}, label: "About You", rows: Array(2), styles: Array(1),
…}
       $fieldSequence: 19
       category: "Input Fields"
       exclude: false
       getData: ƒ ()
       hasCustomId: true
       htmlLabel: "First Name"
       icon: "services/formresources/160/widgets/text-input/text-input.png"
       id: "firstName"
       isDisabled: ƒ ()
       label: "First Name"
       mandatory: true
       mandatoryMsg: "First Name is required"
       offset: {xs: 0, sm: 0, md: 0, lg: 0}
       path: "FirstName"
       properties: {inputType: "text", maxLength: "", preventCopyPaste: false, placeholder: "",
helpText: ""}
       setData: ƒ (value)
       span: {md: 12, lg: 12, xs: 12, sm: 12}
       styles: ["av-form-field"]
       type: "text-input"
}
```

Each item in the `items` object is identified by a unique ID. In the above example, "firstName" is the unique ID. Because of this, the items object doesn't need to nest items within a hierarchy. This makes it easier to access specific items.

view

The `view` object describes the design of the form. It includes a range of data, including the form's responsive breakpoints, the available brands, the current brand, the selected template, and the structure of rows and columns that define the layout of the form.

You can find more information about these objects in the Maestro API Guide. To view the API Guide, Open Maestro, click the Help button and select API Guide.



Debugging JavaScript Rules

All business rules in Maestro, from validation to calculation, are implemented in JavaScript. As a result, a straightforward way to debug these rules is by using `console.log` statements in the script editor:

```
console.log("Hello world...");
```

To distinguish between different types of output, you can also use the warn and error methods:

```
console.warn("This message will have a yellow background");
console.error("This message will have a red background");
```

Below are some useful snippets for outputting relevant data inside a business rule:

```
// Inside a business rule, "value" refers to the data attached to the current field:
console.log(value);

// The "data" object, explained above, can be output from inside a business rule:
console.log(data)

// You can access any field value via the data object by referring to its ID:
console.log(data.lastName)

// The "item" object, explained above, can be output from inside a business rule:
console.log(item)

// Some components, such as the Page Controller component, output extra information via
the "info" object:
console.log(info)
```

While `console.log` statements are easy to use, you'll need to rebuild and reload the form every time you make a change. This makes it a fairly slow method of debugging.

For more productive debugging, breakpoints are a better option.

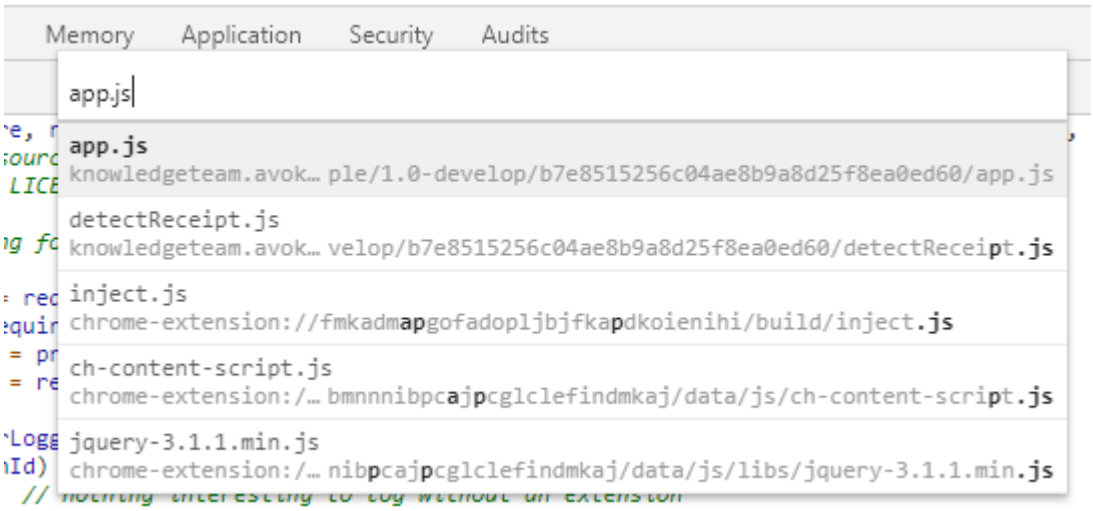
Using Breakpoints

Breakpoints allow you to define points in your JavaScript should pause its execution. This allows you to figure out exactly where your code is faulty without having to write a bunch of `console.log` statements or continually rebuild and refresh the form.

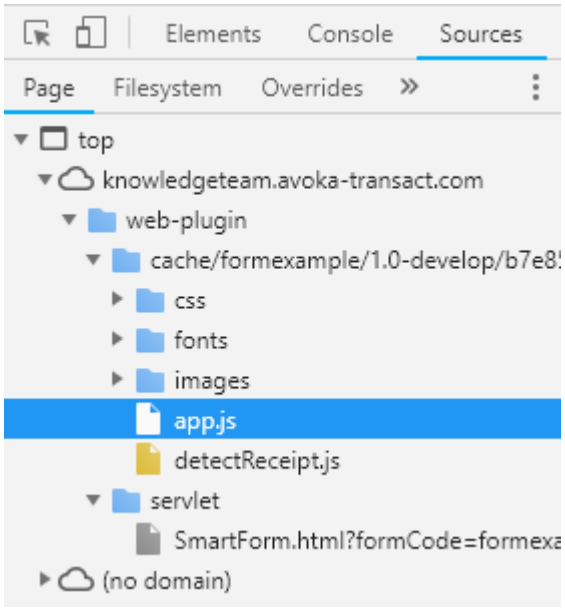
To use breakpoints, we need to locate the app.js file of our rendered Maestro form. This file contains the JavaScript code for our form's business rules.

To find the app.js file:

1. Open a rendered Maestro form.
2. In the DevTools pane, switch to the Sources tab.
3. Press `Control+O` (`Command+O` on a Mac), type "app.js", and click the first result:



Alternatively, you can navigate through the Page hierarchy on the left and double-click the app.js file:

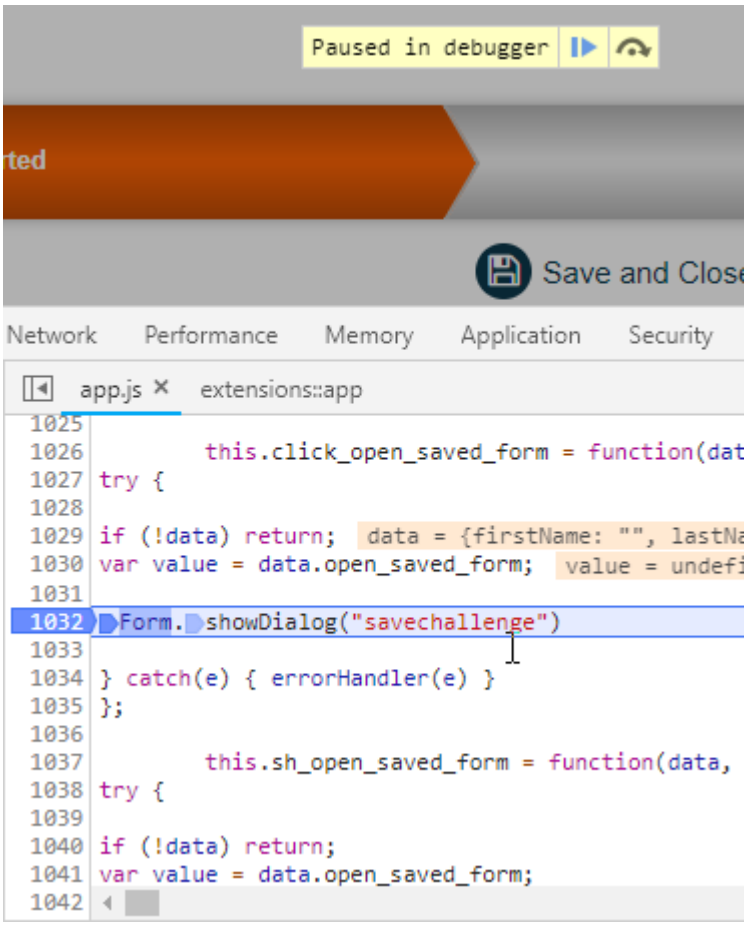


Then, to set a breakpoint, use the find feature in the DevTools pane (`Control+F` on Windows, `Command+F` on Mac) to find the following line of code:

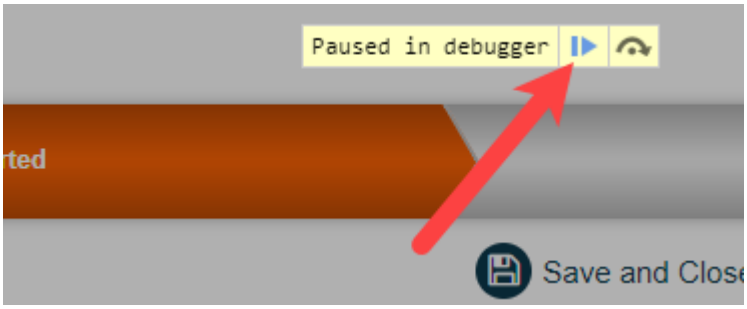
```
Form.showDialog("savechallenge")
```

This code is responsible for opening the modal that allows users to re-open a previously saved form.

If you click Open Saved Form, you'll notice that the modal doesn't appear. Instead, the line of code where the breakpoint is set will be highlighted, indicating that the execution of the JavaScript has paused on that line. Also, a "Paused in Debugger" message is displayed.



To continue executing the script, thereby allowing the modal to open, click the Play button:

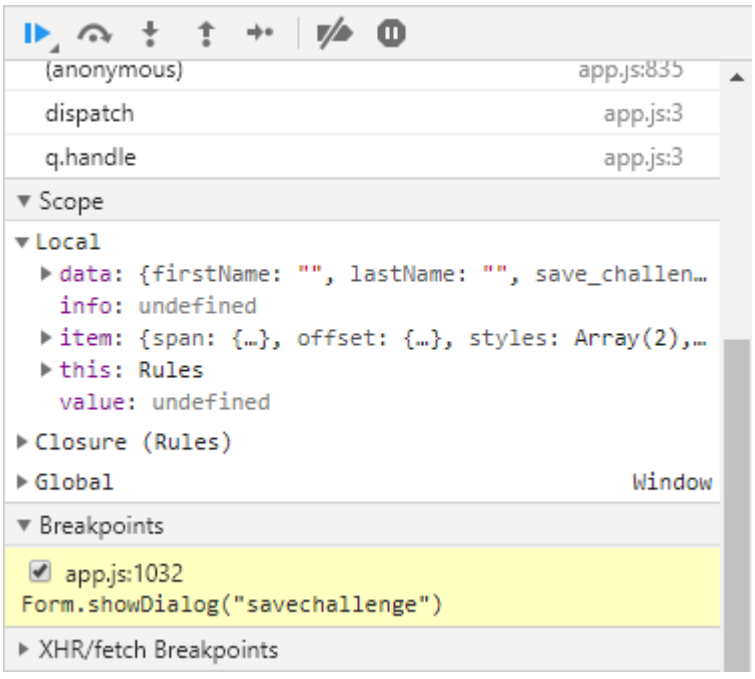


If you're trying to debug more complicated logic, you can set multiple breakpoints:



When you continue after each breakpoint, execution of the script will stop at the next breakpoint, allowing you to move step-by-step through a script.

While moving between breakpoints, pay attention to the Scope section. This will show the current state of the form's data, which may provide insight into why something isn't behaving as expected.



Rule Types and JavaScript Function Naming

As demonstrated above, it's not difficult to find the app.js file. In an actual form though, the file may contain hundreds of rules, which can make it difficult to find and debug the code for a

specific rule.

One solution is to use the `Control+F` shortcut to search for a statement from one of the rules. For example, if you know there is a `console.log` statement in the rule, you can search for "console.log". But if you have a lot of rules, those rules may contain similar statements, making this approach less convenient.

Fortunately, every rule in Maestro has a corresponding JavaScript function that follows a strict naming convention. The functions are named using a rule type code and the ID of the field, with an underscore between them:

```
<rule-type-code>_<field-id>
```

Below are some example function names:

Function Name	Rule
click_verifyIdentity	A click rule for the verifyIdentity button.
eq_totalIncome	A calculation rule on the totalIncome currency field.
load_AvokaSmartForm	A load rule on the form.
ok_emailAddress	A validation rule on the emailAddress field.
sh_previousAddress	A visibility rule on the previousAddress block.

These function names are guaranteed to be unique, so they're a handy way to find specific rules in the app.js file.

Rule Type Codes

Here are the valid rule type codes:

Rule Type Code	Description
chok	Short for "change OK". This is the code for a validate after change rule that augments the standard validation rule to provide support for dynamic data validations. The result of this rule can contain the error message (like the standard validation rule) or a promise.
dc	An acronym for "dynamic class". This rule can be used to dynamically apply a CSS class to a field.
eq	Short for "equals". This is the code for a calculation rule. The result of this rule is the cal-

	culated value of the field. Note that calculation rules are not executed if the field they belong to is not visible.
md	Short for "mandatory". This is the code for a mandatory if rule that controls the required state of an item. The result of this rule is a boolean value where `true` = mandatory, `false` = optional.
ok	This is the code for a validation rule that controls the error state of the field. The result of this rule is a blank string for valid values or an error message string for invalid values.
sh	Short for "show". This is the code for a visibility rule that controls whether the field is presented on screen. This rule returns a boolean value where `true` = visible and `false` = read-only.
us	Short for "usable". This is the code for an editability rule that controls the read-only status of a field. This rule returns a boolean value where `true` = editable and `false` = read-only.

Action Rules

Action rules are triggered at various points in the lifecycle of a form.

Rule Type Code	Description
blur	Executed when the field loses focus.
change	Executed when the value of a field changes.
click	Executed when a clickable UI item is clicked.
focus	Executed when a field acquires focus.
load	Executed when the form is loaded.
presubmit	Executed prior to the form submit event.
postsubmit	Executed after the form submit event.

Dynamic Data Rules

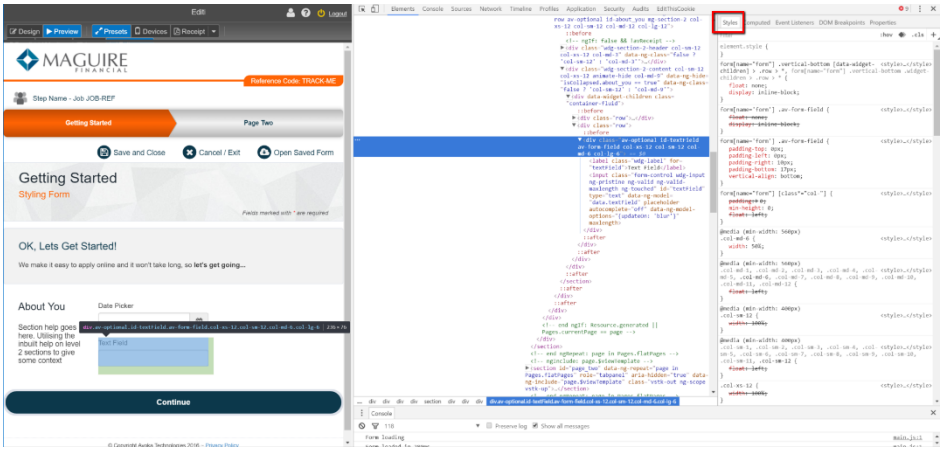
Dynamic Data rules are specific to the Dynamic Data Button component and are triggered upon the successful or unsuccessful completion of a dynamic data call.

Rule Type Code	Description
`onFailure`	Executed when a dynamic data call fails.
`onSuccess`	Executed when a dynamic data call succeeds.

Debugging Form Styles

Maestro forms and templates use CSS extensively. CSS styling can become complicated for big form application. In particular, if two different CSS statements provide conflicting instructions about how to style a particular item, it can be very confusing to debug why your changes do not appear to be working.

To debug styling problems, we recommend that you use the native debugging features built into your browser of choice, rather than attempting to understand this entirely within Maestro. Simply go into Preview mode, and then select F12, or the appropriate menu or key for your browser and operating system, to launch the debugger. This is shown below:



You need to have a moderate to advanced understanding of CSS to debug styles.

When there are conflicts, the usual CSS rules apply. For example:

- If two identical instructions are processed, the one that is later in the CSS definition will override the earlier one
- The more specific the CSS selector, the higher it's precedence
- Precedence can be fine-tuned by adding the `!Important` declaration

Next, learn about advanced debugging of Maestro forms.

Advanced Debugging of Maestro Forms

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Occasionally, while developing Maestro forms, you may see a behavior that doesn't seem right and you can't figure out why. This usually occurs in relation to rules that you've created using the script editor. It's very important to understand the form architecture before you delve into form debugging, as there are certain assumptions of how the 3 form elements are used in interact within the form.

You can learn how to do the basic debugging of Maestro forms, but sometimes you need to apply more advanced debugging techniques that you can use to find out what is going on in your script and in the form data, especially debugging CSS and styling issues are not covered.

Recommended Skill Set

Debugging Maestro forms in the browser is a technical task and requires some understanding of web application development in general, specifically the web standards of HTML, CSS and JavaScript.

Additionally, forms produced by Maestro run on AngularJS, which is a JavaScript based open-source front-end web application framework backed by Google. There is a very strong community supporting this technology and while it is not required that you are familiar with AngularJS, it would be a benefit for you to have some basic knowledge of the framework. We would recommend you review the tutorials to develop your understanding of this technology:

Debugging Tools

Before getting started, ensure you have appropriate tools for debugging JavaScript in the browser. These include:

Modern Browser

Most modern browsers come with good developer tools these days. We prefer to use Google Chrome, but Firefox and Microsoft Edge can also be used. This article assumes you are using Google Chrome, which we recommend.

Developer Tools Enabled

All these browsers come with developer tools out of the box and allow you to install plug-ins and extensions to provide additional capabilities. There are a number of ways to enable these developer tools, but universally, hitting the F12 key while viewing a page will bring up the built-in developer tools.

After that, you can run various commands in the console to print form's data. For example, you can use `console.log(maestro.Form.getDataXml(true))` to list the `/SFMData/SystemProfile` content as an XML file, as shown below:

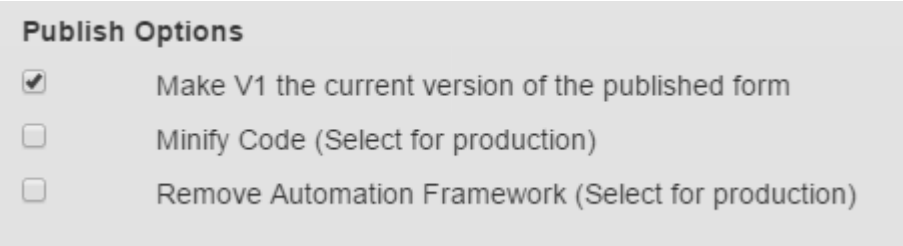
```
<AvokaSmartForm>
	<AboutYou>
		<FirstName>
		</FirstName>
	</AboutYou>
	<SFMData>
		<SystemProfile>
		<FormCode>Test 1</FormCode>
		<RequestLogKey>12345</RequestLogKey>
		<DisplayMode>Entry</DisplayMode>
		<ServerBuildNumber>22.4.0</ServerBuildNumber>
		<RevisionNumber>0</RevisionNumber>
		<FormDataServiceURL>https:/[/YOUR-SERVER-HOST-NAME]/web-plu-
gin/servlet/FormDynamicDataServlet</FormDataServiceURL>
		<TrackingCode>ABCD1234</TrackingCode>
			<AbandonmentReason>
			</AbandonmentReason>
			<AbandonmentComments>
			</AbandonmentComments>
		</SystemProfile>
	</SFMData>
</AvokaSmartForm>
```

AngularJS Browser Plugin

Maestro forms are built on the AngularJS framework and plugins are available to provide specific support for AngularJS web applications. In Chrome, we use AngularJS Batarang, Firefox also has AngularJS plugins that may provide the same capability.

Publishing Forms for Debug

When you publish a Maestro form there are a number of options available to you.



Some of these are very relevant to debugging, these are:

Publish Options	Description
Minify Code	Deselect this option. Code minification is the process of compressing code to reduce the size and therefore load time, but does not affect the operation of the code. The process removes unnecessary characters from the code,white space, new line, comments etc... The result of minification is a single line of code that is very long and difficult to read and debug so ensure that this option is deselected for debugging.
Remove Automation Framework	Deselect this option. The automation framework built into Maestro forms facilitates UI driven automated tests such as those that would be run from test tools like Selenium. This framework also contains some features that will assist in debugging JavaScript so it is handy to keep the automation framework for debugging.

Activating a Debug Session

To start a debugging session you should:

- Load the published Maestro form in a modern browser of your choice
- You must access the published version of the form, DO NOT try and debug a form in Preview mode in the Maestro designer.
- If you are intending to debug JavaScript (e.g. Maestro rules) then ensure you have deselected the Minify Code and Remove Automation Framework publish options as described above.

- Once the published form is loaded in the browser you can activate the browser developer tools by hitting the F12 key.

Shortcuts for Republishing Forms

While debugging forms you may need to republish your form with changes multiple times and you need to refresh the page with your published form on it each time. The following keyboard shortcuts are available for you to use to streamline this process:

- CTRL-P: When used from the Maestro Design page will bring up the Publish dialog. (Command-P on Mac)
- CTRL-ENTER: Will trigger the publish function. Remember to deselect the Minify Code publish option first.
- CTRL-TAB: This will switch you to the last active browser tab which in most cases will be the published form.
- CTRL-R: Will refresh the currently active browser window.

Refreshing the page with the republished form on it should leave the browser developer tools active so you should not need to reactivate them each time.

Inspecting the Form Object

In the Maestro framework, the Form object contains everything else you might need to access for debugging including the View, Items and Data (see Maestro Form Architecture), so to access any of these elements you need to scope the Form object. There are a number of ways to do this from the Console tab in the browser developer tools:

- Using JQuery you can scope the form object in this manner:

```
$("Form").scope().Form
```

- If you have included the Automation Framework (see Publishing Options) you can short cut this access by using the maestro root element included with that framework:

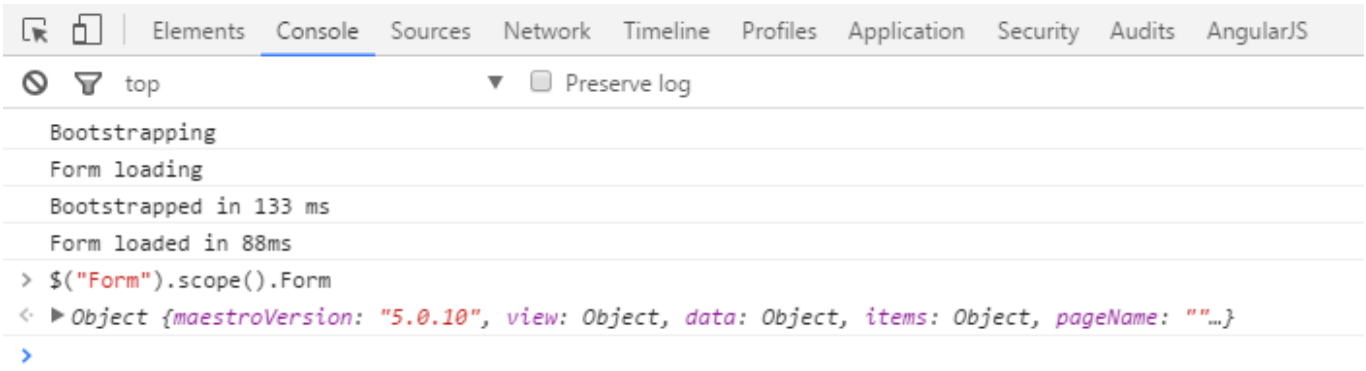
```
maestro.Form
```

- If you have AngularJS Batarang (or similar) installed, they may also allow you to access the Form object by way of the $scope object (You need to select an element in the Ele-

ments tab first - e.g. body or form):

```
$scope.Form
```

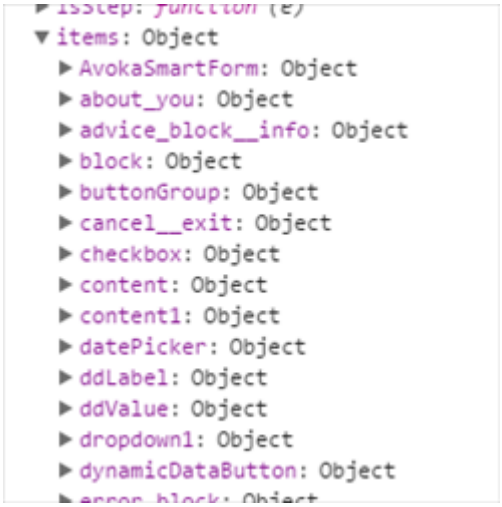
Enter one of these directives into the Command prompt in the Console tab and press Enter to scope the Form object:



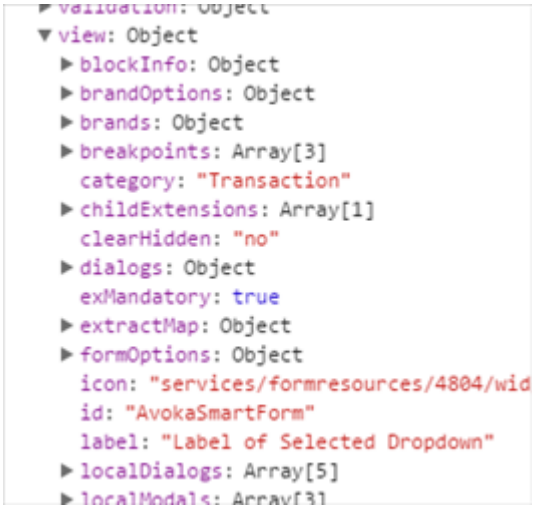
Opening up this object you will see a long list of sub elements but among those you will find the 3 key elements to the Maestro Form Architecture, the Form Data object,



The Form Items,



And the Form View.



Of course, if you have a JavaScript debugging session active (as described below) you will have access to the Form Data object in the function scope and can readily inspect its contents.

Logging an Item to the Console

There is a handy Util function to generate a printable representation of an item and log it to the console as follows:

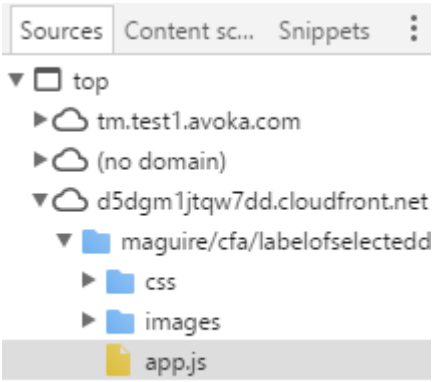
```
$scope.Util.logItem($scope.Form.items.personalDetailsBlock)

"personalDetailsBlock
┌ firstName
└ middleNames
- lastName
- dateOfBirth
 ┌ day
 | month
 └ year
 — date
 — daysData
- emailAddress
- mobileNumber
"
```

Locating a JavaScript Rule for Debug

All rules in Maestro are implemented as JavaScript so to locate the logic implemented in a Maestro rule you must find the JavaScript code that implements that rule. You can do this as follows:

1. In the developer tools window select the tab that lists the page source files (in Chrome this is titled Sources).
2. Open the app.js file by either using CTRL-O (CMD-O on Mac) and typing in the name of the file, or by searching in for it the file sources hierarchy then double-click to open it (the location of this file will vary depending on your installation and the existence of a CDN in your environment, but it should only be a couple levels deep).



3. In the app.js file, search for and locate the JavaScript function that contains your rule in the app.js file. There are several ways to do this depending on the capabilities of the browser dev tools, but most commonly you will search by a code snippet or by the function name:
4. To find the function using a code snippet, select a snippet of code from your rule script by clicking in the app.js file and using CTRL-F to activate the find feature. You may need to review your rule script in the Maestro designer and select a snippet that is likely to be unique. For example, below I have searched for the snippet Form.getItemFromPath ('data.buttonGroup') and located the function eq_ddLabel that contains my rule script:

```
447
448         this.eq_ddLabel = function(data, item, info) {
449 if (!data) return;
450 var value = data.ddLabel;
451
452 var item = Form.getItemFromPath('data.buttonGroup');
453 var selectionObj = Util.find(item.properties.options, "value", data.buttonGroup);
454
455 return (selectionObj && selectionObj.label) || "";
456 };
457
```

5. To search by JavaScript function name you will need to know the naming conventions for these functions (see the section on JavaScript Function Naming below). If you know the function name you can either search for the function name as a snippet (as above) or if your dev tools support it you can use the JavaScript Member search capabilities - in Chrome this can be activated using CTRL-SHIFT-O:



6. To debug your rule, set a break-point on one of the lines before the code you want to step through by clicking in the left column area and confirming the break-point indicator:

```
448         this.eq_ddLabel = function(data, item, info) {
449 if (!data) return;
450 var value = data.ddLabel;
```

7. Now you are ready to debug the rule. You can return to your form window and perform the action that will trigger the rule to execute and step through your rule script using the standard JavaScript debug functions.

A word of caution against modifying JavaScript in-line in the browser debug tools. Results of this sort of change are often unreliable. While it is less convenient to republish a form each time you want to change a rule - this is the recommended approach.

Rule Types and JavaScript Function Naming

Every rule that you put into your Maestro form has a corresponding JavaScript function in the `app.js` file. The name of this function is generated based on a standard naming convention that consists of a rule type code and the field ID with an underscore separator, for example:

```
<rule-type-code>_<field-id>
```

The following codes represent the types of rules used by the Maestro widgets:

Rule Type Code	Description
`sh`	Short for show, this is the code for a visibility rule that controls whether the field is presented on screen. The result of this rule should be a boolean value where true = visible, false = read-only.
`us`	Short for usable, this is the code for an editability rule that controls the read-only status of a field. The result of this rule should be a boolean value where true = editable, false = read-only.
`md`	Short for mandatory, this is the code for a mandatory if rule that controls the required state of an item. The result of this rule should be a boolean value where true = mandatory, false = optional.
`ok`	Interpreted as OK, this is the code for a validation rule that controls the error state of the field. The result of this rule should be a string that is either blank (for valid values) or contains the error message (for invalid values).
`chok`	Interpreted as Change OK, this is the code for a validate after change rule that augments the standard validation rule to provide support for dynamic data validations. The result of this rule can either contain the error message (like the standard validation rule) or a promise.
`eq`	Short for equals, this is the code for a calculation rule. The result of this rule should be the calculated value of the field. Warning: In Maestro, calculation rules are not triggered if the field they belong to is

	not visible.
`click`	A click rule is an action rule triggered by clicking on a clickable UI item.
`blur`	A blur rule is an action rule triggered when the focus leaves a field.
`focus`	A focus rule is an action rule triggered when a field acquires the focus in the browser window.
`change`	A change rule is an action rule triggered when the value of a field changes.
`dc`	An acronym for dynamic class, this rule can be used to apply a CSS class to a field dynamically based on logic.
`load`	An action rule that is triggered at the time the form is loaded.
`presubmit`	An action rule that is triggered prior to form submit.
`postsubmit`	An action rule that is triggered after the form submit event.
`onSuccess`	This rule type is specific to the Dynamic Data Button and is an action rule that is executed upon successful completion of the dynamic data function.
`onFailure`	This rule type is specific to the Dynamic Data Button and is an action rule that is executed upon failure from a dynamic data call.

NOTE

Widget developers can create additional rule types for their widgets when needed.

Examples

Function Name	Rule
`sh_previousAddress`	A visibility rule on the `previousAddress` block
`eq_totalIncome`	A calculation rule on the `totalIncome` currency field
`ok_emailAddress`	A validation rule on the `emailAddress` field
`click_verifyIdentity`	A click rule on the `verifyIdentity` button
`load_AvokaSmartForm`	A load rule on the form

Next, learn about UI test automation.

UI Test Automation Overview

MaestroThe UI design product. | Form Builder | 22.04 This feature was updated in 22.04.

Many customers choose Journey Maestro because of the speed to market advantages the platform offers. Underpinning speed to market is a successful DevOps[1] strategy and the key to DevOps is a comprehensive regression test suite.

Maestro provides the compelling set of features that enables UI testers to dramatically accelerate the creation and maintenance of UI test suites for forms as well as solving one the most pressing issues of web-testing - non-deterministic test[2] results.

Before we delve into details, we recommend becoming familiar with a WebDriver based test automation framework and advanced Maestro debugging techniques.

Selenium & WebDriver

Talking about UI test automation, we assume you use a WebDriver derived UI automation framework. We also provide all examples for Selenium[3] in Java[4]. Although that may not be your company's preferred UI testing framework, the concepts are valid for any WebDriver derived framework.

WebDriver is a standard maintained by the Worldwide Web Consortium (W3C), which defines it as follows:

---

[1]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.

[2]A test is non-deterministic when it passes sometimes and fails sometimes, without any noticeable change in the code, tests, or environment. Such tests fail, then you re-run them and they pass. Test failures for such tests are seemingly random.

[3]The Selenium testing software tool is used to automate tests across browsers for web applications. It's used to ensure high-quality web applications – whether they are responsive, progressive, or regular. Selenium is an open-source tool.

[4]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.

- WebDriver is a remote control interface that enables introspection and control of user agents. It provides a platform- and language-neutral wire protocol as a way for out-of-process programs to remotely instruct the behavior of web browsers.
- Provided is a set of interfaces to discover and manipulate DOM elements in web documents and to control the behavior of a user agent. It is primarily intended to allow web authors to write tests that automate a user agent from a separate controlling process, but may also be used in such a way as to allow in-browser scripts to control a – possibly separate – browser.

In short, it allows for the remote control of a browser application, you write tests that 'drive' the web application.

It is beyond the scope of this document to explain the intricacies of testing, but a short overview of the key concepts is appropriate.

UI Testing Key Concepts

To control an element in the browser, you must locate it in the browser's DOM[1]. Since the original Selenium framework, on which WebDriver is derived, was written in JavaScript and ran in the browser, locators closely match familiar `ID`, `CSS` and `XPath` based selectors. For example, all of the following Java code snippets locate an element and then click on it:

- `driver.findElement(By.id( "element-id" )).click();`
- `driver.findElement(By.cssSelector( "#element-id" )).click();`
- `driver.findElement(By.xpath( "//*[@id='element-id'" )).click();`

Each call follows the same structure:

- `driver` - is the WebDriver, an object that provides the means to connect to, introspect, and control a browser session.
- `findElement` - is a method that finds the first element using the given `By` locator or throws if no element can be located matching the locator. The return value is a WebElement that provides methods to query and interact with the remote DOM object.
- `By` - is a class that provides the means to lookup an element in the DOM by a specific locator scheme. As noted, conceptually, these should be familiar with anyone who uses CSS Selectors. In each example we select by the element's ID but, obviously, this isn't how

[1]The Document Object Model is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree.

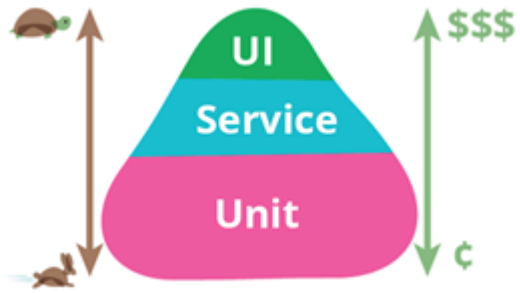
`cssSelector` and `xPath` would typically be used. Note that `By.name`, `By.tagName`, `By.linkText`, and `By.partialTagName` are also supported, but we do not need to use them.

- `click()` - is a method on the WebElement that clicks on the element if it is visible and has a height and width of greater than zero. (Selenium is a test framework that tries to behave as a human would: a DOM element may exist but not be visible.)

To create a test, every element in the DOM must be identified and then code written that introspects and interacts with these elements, asserting that the correct behaviour is observed.

Issues in UI Testing

Martin Fowler popularized the idea of the Test Pyramid that was first espoused by Mike Cohn as a way of thinking about different kinds of automated tests. This is illustrated below.



Martin Fowler's Test Pyramid. For more information, see https://-martinfowler.com/bliki/TestPyramid.html

This neatly captures the fact that UI testing is costly and slow. From the code examples above, one reason why this is true should be obvious: identifying elements in the DOM is time consuming and fragile. During development, element types change, element IDs change, element structure in the DOM changes, and CSS changes. Changes introduce non-deterministic results into test suites.

To quote Fowler - "Most importantly such tests are very brittle. An enhancement to the system can easily end up breaking lots of such tests, which then have to be re-recorded. You can reduce this problem by abandoning record-playback tools, but that makes the tests harder to write. Even with good practices on writing them, end-to-end tests are more prone to non-determinism problems, which can undermine trust in them. In short, tests that run end-to-end through the UI are: brittle, expensive to write, and time consuming to run"

Non-deterministic[1] failures are the pernicious false negatives – for example, tests that exercise a web based GUI and fail, but fail because of a tiny CSS change.

So much for the bad news. The good news is that using Maestro, most of these problems can be solved:

- Locator instability can be eliminated
- Non-deterministic run time failures can be moved to test build time (with a statically typed language)
- Timing issues can be reduced

Next, learn how to perform UI test automation of Maestro forms.

[1]A test is non-deterministic when it passes sometimes and fails sometimes, without any noticeable change in the code, tests, or environment. Such tests fail, then you re-run them and they pass. Test failures for such tests are seemingly random.

UI Test Automation of Maestro Forms

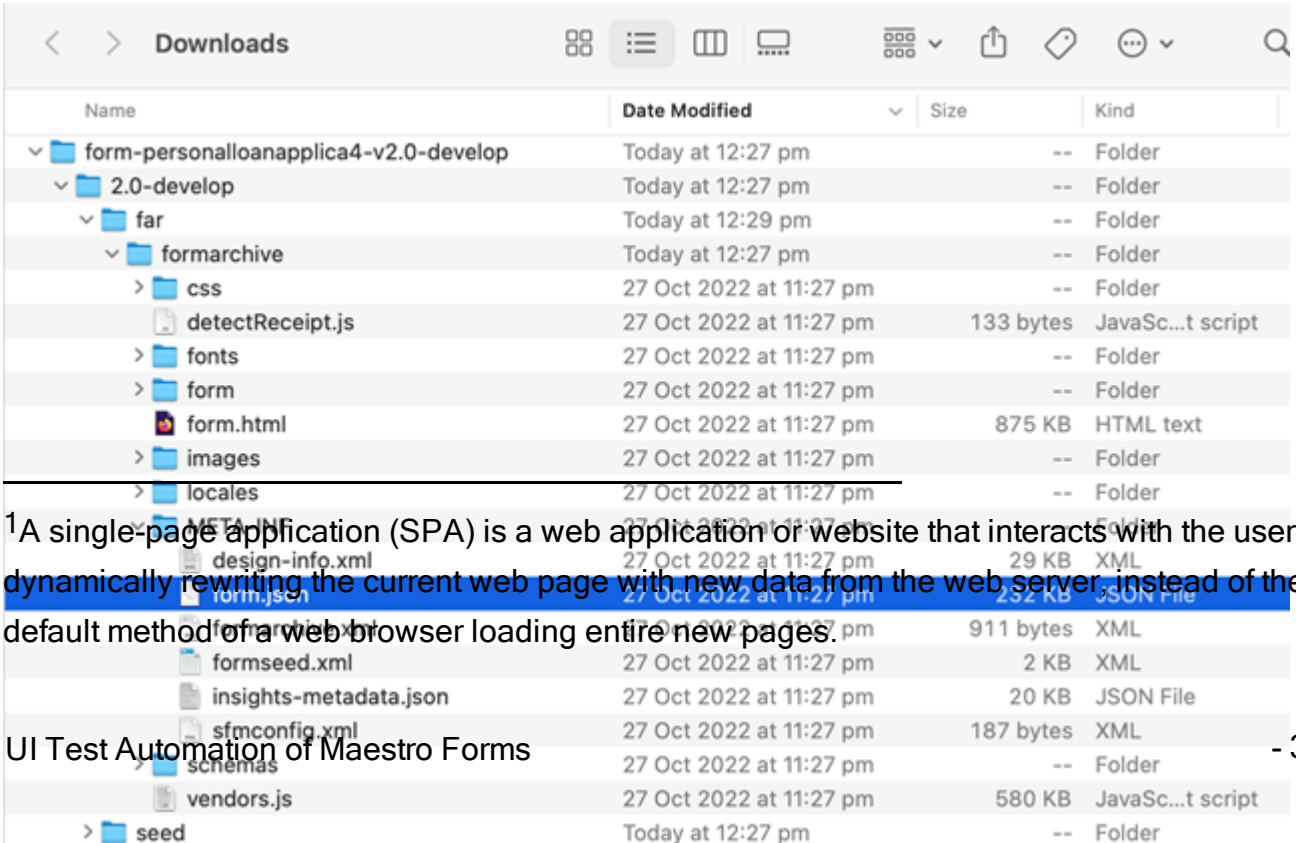
MaestroThe UI design product. | Form Builder | 22.04 This feature was updated in 22.04.

As we explained in UI test automation overview, UI testing of application forms can be a daunting task. However, Journey Maestro can help you with that. Let's look at how you can achieve this.

Maestro is a design tool that generates single page web applications[1]. These application are Maestro forms, which are packaged during a build process. How it does this is key to solving the test automation puzzle:

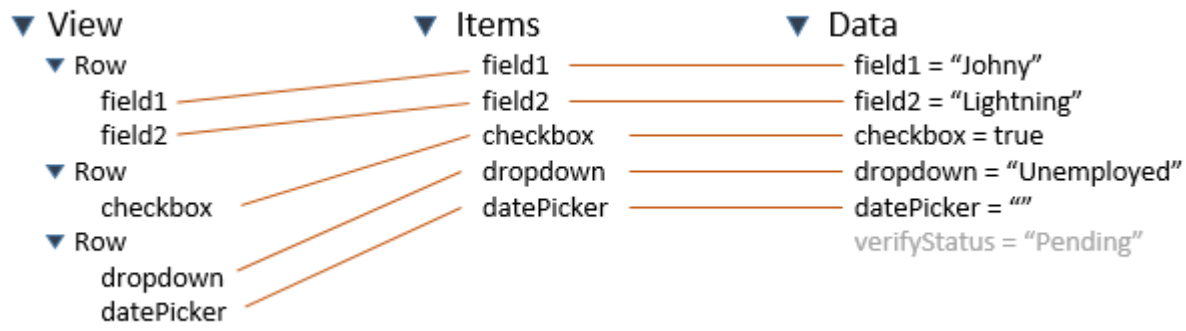
- The Maestro build process generates an artefact containing a machine-readable definition of a form
- Maestro enforces certain constraints on a form developer, ensuring that every element has a unique identifier
- Every form element always generates the same HTML structure. The structure varies according to the element type, but the pattern is always the same.
- Form elements with validation rules generate elements that can be reliably linked back to the parent input element
- There is a simple mapping between form element identities and both the runtime representation of form data (JSON) and the submission format (XML)

The machine readable file is called `form.json` and it's packaged in the `META-INF` directory of the Form Archive (FAR) build artefact, which is itself packages within a ZIP file. This is illustrated below.



[1]A single-page application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages.

The `form.json` provides the Form View definition, as shown below:



This structure is unsterilized into a browser DOM:



Maestro Form.json

Look inside the `form.json` and you will see a structure that represents the rows and items that define the UI elements. However, this data exists in a complex structure, most of which is superfluous for our testing needs.

Three objects are of importance:

- rows - contains the form UI page definitions, which are often deeply recursive - there are rows withing rows that represent the hierarchical structure of the `<div>` elements in the rendered form. The UI elements themselves are all leaf nodes and we will exploit this fact when extracting the definitions.
- dialogs - contain dialog definitions, each of which is represented as a named object, each of which has a rows property which conforms to the structure discussed, above.
- modals - contains modal definitions and has an identical structure to the dialogs object.

Writing code that parses this structure is challenging. Therefore, in our framework, we sidestepped this issue and used JsonPath. JsonPath expressions are analogous to XPath

expressions, allowing the user to query complex structures using simple expressions. For more information, see https://goessner.net/articles/JsonPath/index.html#e2 and https://-github.com/json-path/JsonPath.

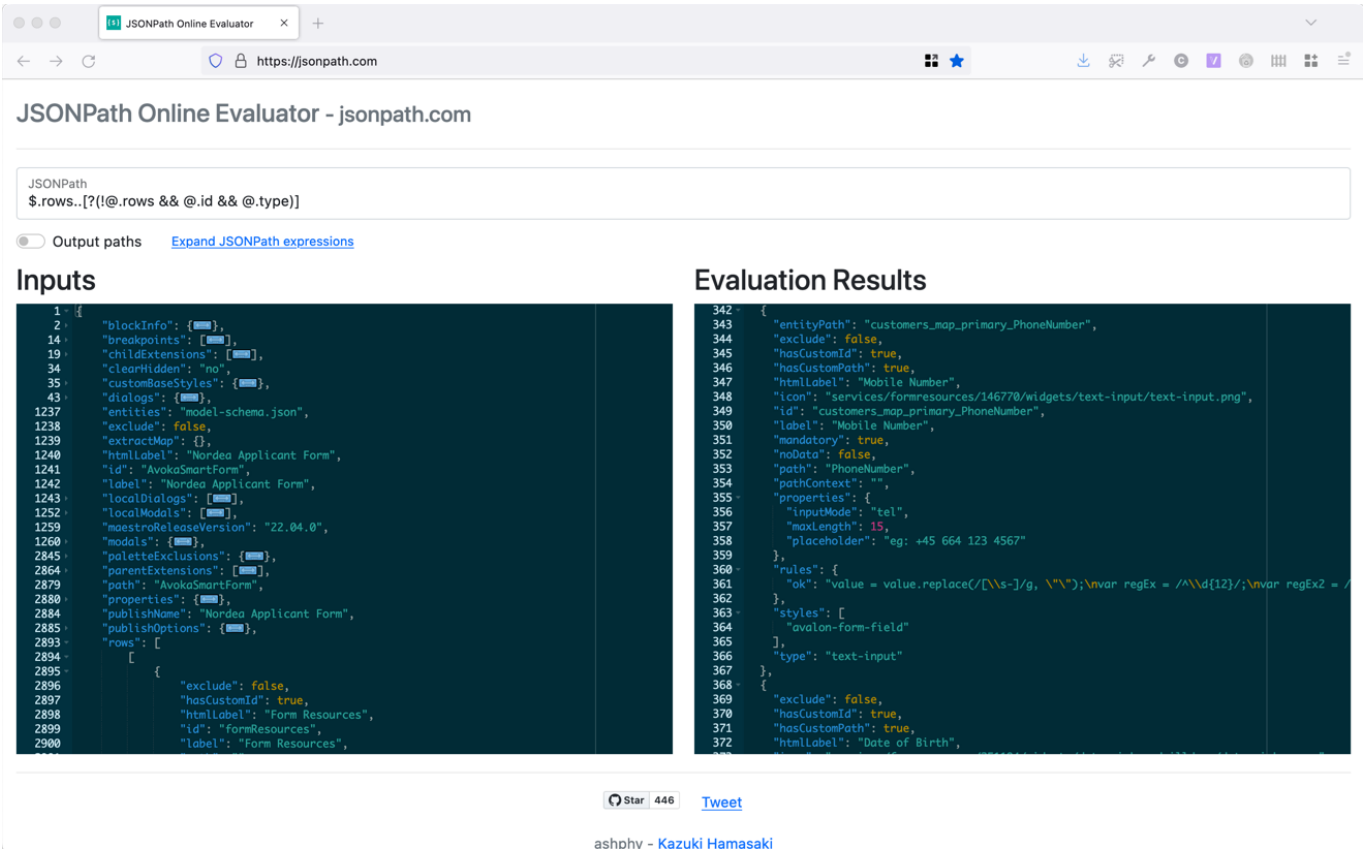
Three expressions are needed to get you started, one for each of rows, dialogs and modals:

- `$.rows..[?(!@.rows && @.id && @.type)]` - This recursively searches all children of the root rows object, filtering children that do not have a rows element themselves, but do have both an id and a type property.
- `$.dialogs..[?(!@.rows && @.id && @.type)]` - Same as above but start the search at a different root location in the JSON structure.
- `$.modals..[?(!@.rows && @.id && @.type)]` - Same as above but start the search at a different root location in the JSON structure.

The result will be returned as an array of Form Item definitions.

> NOTE
>
> We use item and element interchangeably.

When testing these expressions, we found the JSONPath Online Evaluator to be invaluable. This tool is easy to use. What you need to do is to paste a `form.json` file on the left side of the main window, as shown below, and collapse all of the nodes except rows. We used the JSONPath expression to return form elements, namely, leaf elements under the rows object that have an id and a type.



Form Item Definition

The form item definition, shown in an image above, is replicated below verbatim. We scrolled the display to show a complete item and we will use this to explain the structure and how it maps to a Selenium object locator.

```
{
  "entityPath": "customers_map_primary_PhoneNumber",
  "exclude": false,
  "hasCustomId": true,
  "hasCustomPath": true,
  "htmlLabel": "Mobile Number",
  "icon": "services/formresources/146770/widgets/text-input/text-input.png",
  "id": "customers_map_primary_PhoneNumber",
  "label": "Mobile Number",
  "mandatory": true,
  "noData": false,
  "path": "PhoneNumber",
  "pathContext": "",
  "properties": {
    "inputMode": "tel",
    "maxLength": 15,
    "placeholder": "eg: +45 664 123 4567"
  },
  "rules": {
    "ok": "value = value.replace(/[\\s-]/g, \"\");\nvar regEx = /^\\d{12}/;\nvar regEx2 =
/^\\d{11}/; \nvar regEx3 = /^\\d{10}/; \nvar number =  value.replace(/[\\s-+]/g,
\"\");\nif(value.indexOf(\"+\") == 0 && (number.length >= 10 || number.length <= 12) &&
(regEx.test(number) || regEx2.test(number) || regEx3.test(number))) {\n\tdata[item.id] =
value;\n\treturn true;\n} else {\n  if (data.selectLanguage == \"de\") {\n    return
\"Bitte geben Sie Ihre \"+item.label+\" im Format +439999999999 ein.\";\n  } else {\n\tre-
turn \"Please enter your \"+item.label+\" in the format +439999999999.\";\n  }\n}"
  },
  "styles": [
    "avalon-form-field"
  ],
  "type": "text-input"
}
```


The structure of a form definition is fairly self-explanatory, if we ignore the rules element, and consists of the following elements:

- `id` - is the unique identity for the element.
- `type` - is the Maestro type and, as we shall see, this type plus the ID is the key to test generation.
- `entityPath` - is the entity path into the Maestro data model in the browser.

- `hasCustomId` - is a Boolean indicating whether the system or the user has generated the Id (and is therefore useful in linting applications).
- `mandatory` - is a Boolean indicating whether this item must be provided if the element is visible.
- `properties` - provides a type specific container for additional properties. In the example above, the mobile keyboard type is set to telephone, the maximum number of characters that can be input is set to 15 and the placeholder for text used by a screen reader is set.

Item Definitions and Selenium Locators

We're now in a position to bring together all of the threads in this discussion and explain how the item definition is used to generate Selenium identifiers.

Maestro generates HTML code, the code varies according to the item type, but is always predictable. Using the example of a text-field we see the element rendered in a form with the DOM Inspector showing:



We can see that the element id is exactly that from the item definition in the `form.json`. Because all text-field items are treated the same way by the Maestro compiler, we can safely and reliably generate locators:

Framework	Example code
Selenium Java	`By.id("customers_map_primary_PhoneNumber");`
Selenium C#	`By.id("customers_map_primary_PhoneNumber");`
Selenium JavaScript	`By.id("customers_map_primary_PhoneNumber")`
Protractor	`by.id("#customers_map_primary_PhoneNumber")`
Cypress	`cy.get("#customers_map_primary_PhoneNumber")`
Galen	`phoneNumber id customers_map_primary_PhoneNumber`

By generate, we mean that a piece of code reads the form.json definition and writes the Locator code itself by iterating over the array of items extracted from the form.json as explained earlier.

A simplistic implementation of a method that generates Java code for the text-field type is:

```
/**
 * Build a By statement for any of the Text Field types
 *
 * @param maestroElementDefinition The element definition from the form.jsor
 * @return Java source for a By selector that will locate this element
 */
private static String builtTextFieldSelector(final JSONObject maestroElementDefinition) {
  final String TEXT_FIELD_TEMPLATE = """
      // Maestro type: %s
      static final By %s = By.id(\"%s\");
      """;

  final String id = maestroElementDefinition.getAsString("id");
  final String type = maestroElementDefinition.getAsString("type");
  final String by = String.format(TEXT_FIELD_TEMPLATE, type, id, id);
  return by;
}
```

Given the JSON object used earlier as an argument, this would return:

```
// Maestro type: text-field
static final By customers_map_primary_PhoneNumber = By.id("customers_map_primary_PhoneNum-
ber");
```

Sadly, different Maestro element types have different code generation strategies. Testers must also deal with Repeats, which are blocks containing Form elements that can be repeated an arbitrary number of times. Finally, all we have used is the id and type properties; we have ignored the rich source of metadata about the element including in the property object. We will deal with each of these issues later in this documentation.

Statically Typed Definitions

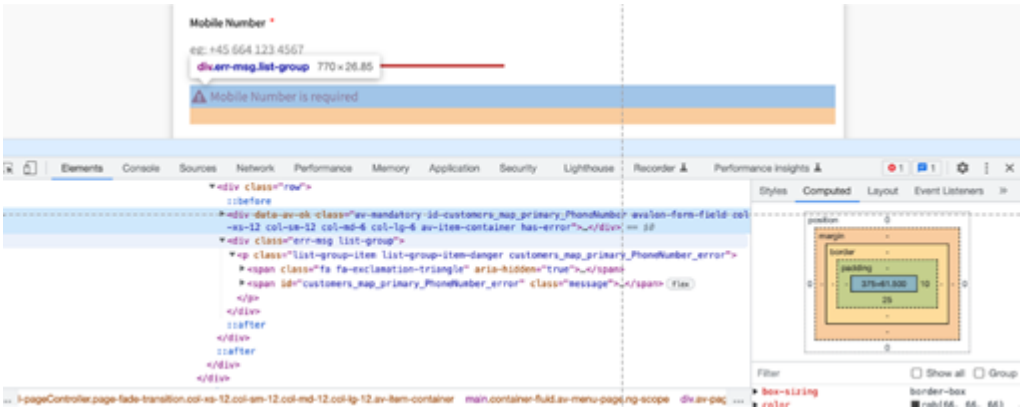
In the generated code example above, we used the property name as the member name and we made the field `static final`. This brings us to why a statically typed language is so valuable. If a developer changes the id of an element, then the test that relies on this element name will fail with a compilation error, immediately on regeneration of the locators.

It is hard to overstate how important this is to tester productivity. By generating code in a statically typed language, we eliminate the major cause of non-deterministic runtime test failures.

However, as noted, we have not yet used all the metadata available to us to produce a truly strongly typed model. To do this, we would have to write a Maestro aware framework on top of Selenium.

Field Validation

Maestro provides validation rules and, where these fail, an error message is displayed.



Once again, the same rule always applies for the generation of these messages. The id of the `<span>` containing the error message is created from `{id}_error`. Where `{id}` is the id of the Maestro element to which this message is bound, in this case it is `customers_map_primary_PhoneNumber`.

In testing terms, this grants us a major advantage - by checking whether the error block is displayed, we have a reliable means of testing whether the value input is valid. These selectors can be generated from our JSON.

Data Model Lookup

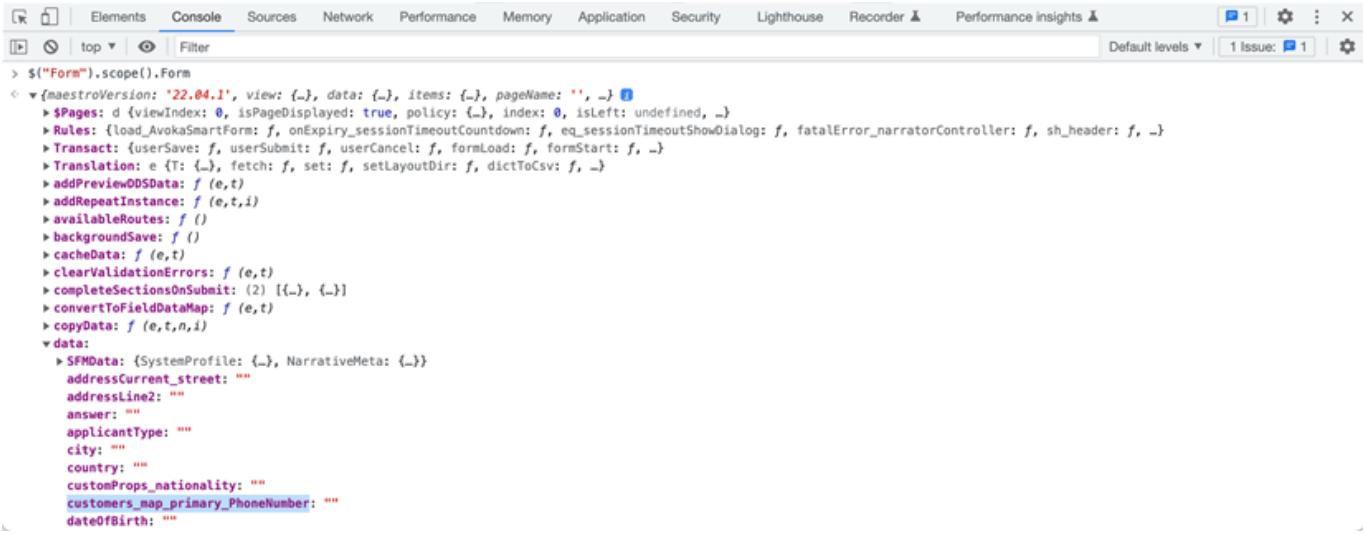
Thus far, we have concerned ourselves with the DOM elements for our form. In Maestro, each item has an associated entry in the form data model. In our item definition we have:

```
"entityPath": "customers_map_primary_PhoneNumber"
```

This key can be used to get or set the data value bound to the element. Maestro exposes the following API to access this data:

```
$("Form").scope().Form
```

You can check it in your browser's console, as shown below. We have expanded the return value to show the data entry and highlighted the phone number field.



A test that exploit this by using the Selenium driver's `executeScript` method of the `JavaScriptExecutor` class:

```
((JavaScriptExecutor)driver).executeScript(
  "return $('Form').scope().Form.data.customers_map_primary_PhoneNumber" );
```

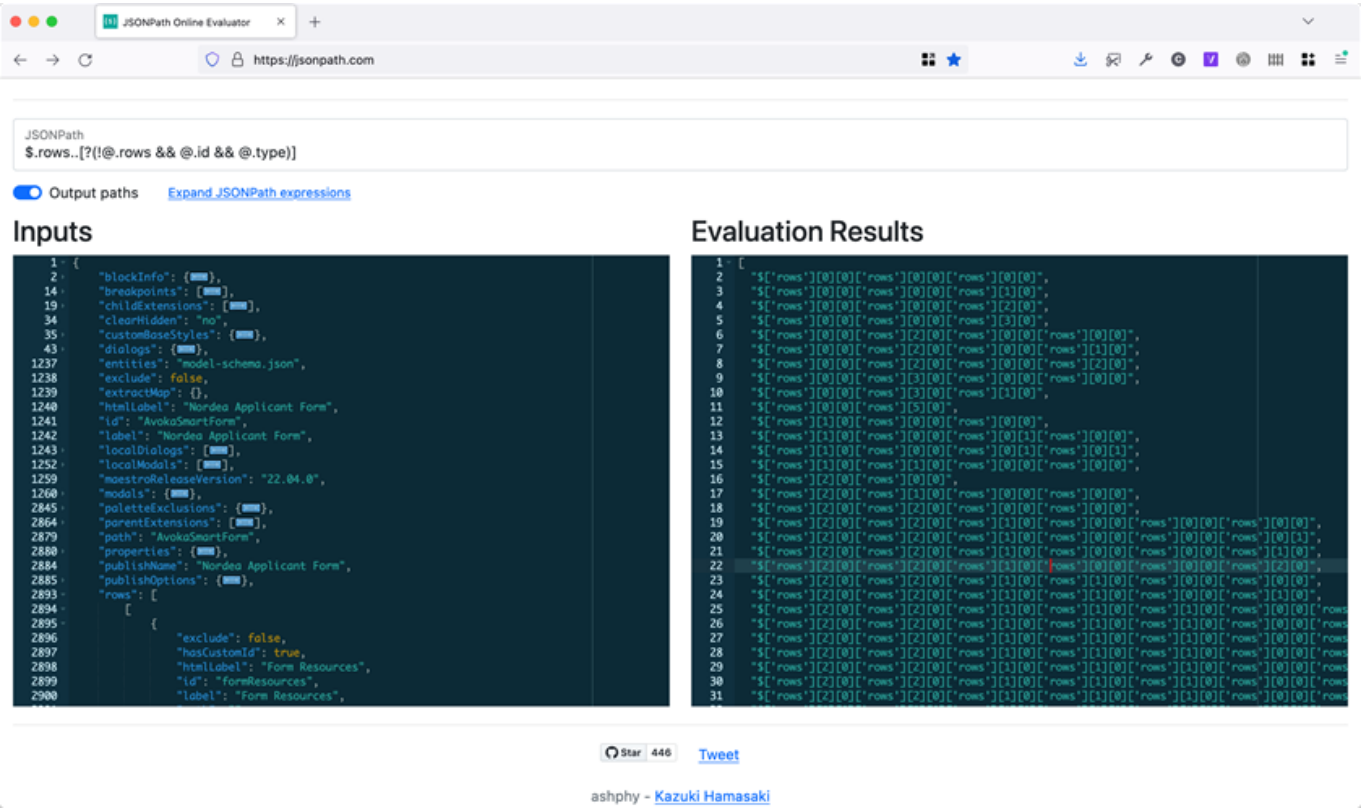
You can see that an automated test can not only set and validate elements via the UI, it can also verify that the underlying data model is updated as expected for the element type, which is a vital step requirement when dealing with elements like Social Security Numbers that require masking.

Page Object Model Generation

A Page Object Model is a collection of the locators or higher-level elements, grouped logically by page. However, looking back, you can see that the item's JSON definition doesn't contain any information identifying the parent page / dialog / modal.

To solve this problem, we need to resort to some coding so that an item is aware of where it sits in the hierarchy. As an aside, the runtime form engine provides this feature, decorating each item with a parent property. This is one of the few things done at runtime and so we must do this by hand in our code generation framework.

Once again, JSONPath can assist us. In this case, you use the Output Paths feature of the tool, which outputs the path to a JSON object matched by the query expression. This is illustrated below:



You use this information as follows:

```
gather the list of JSON Path elements for the page
gather the list of JSON elements for the page
loop for each target JSON element
  get the matching JSON Path element for the target JSON element from the JSON Path list
  loop for each path element in the JSON Path
    remove the last path element from the JSON Path
    lookup the JSON element from this JSON Path
    if the type of this JSON Path is page, then break
  continue
  add the page name as a property on the JSON element
end loop
```

NOTE

You can implement this pseudo code in your preferred programming language.

Once you have the page information, you can output a true Page Object model.

Conclusion

You could rightfully ask - "Why doesn't Temenos Provide a Testing Tool that does this for us?"

We asked our customers "What UI test framework and language do you use?" There were almost as many answers as we have customers.

For this reason, we've decided to publish this guidance for how to exploit the data we provide. We're also looking at open sourcing our own internal Proof of Concept (POC) that we use to demonstrate the techniques we discuss here.

Next, learn how to "Debug Maestro Forms with Chrome DevTools" on page 272.

Improve Page Load Time of Maestro Forms

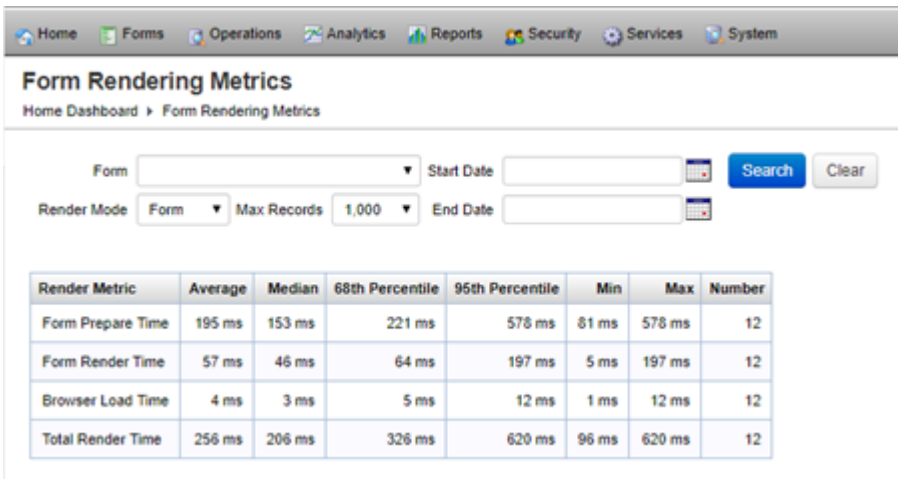
MaestroThe UI design product. | Form Builder ▪ Template Designer | 18.11 This feature was updated in 18.11.

Studies have shown that page load time has a significant impact on conversion rates. In short, the faster your forms load, the less likely they will suffer from high rates of abandonment. As a result, the performance of your forms has a direct impact on your organization's bottom-line.

In the topic, we're going to explore a few simple methods for improving page load times by analyzing your forms in their current state and by enabling or tweaking features available to you right now.

Check Form Rendering Metrics

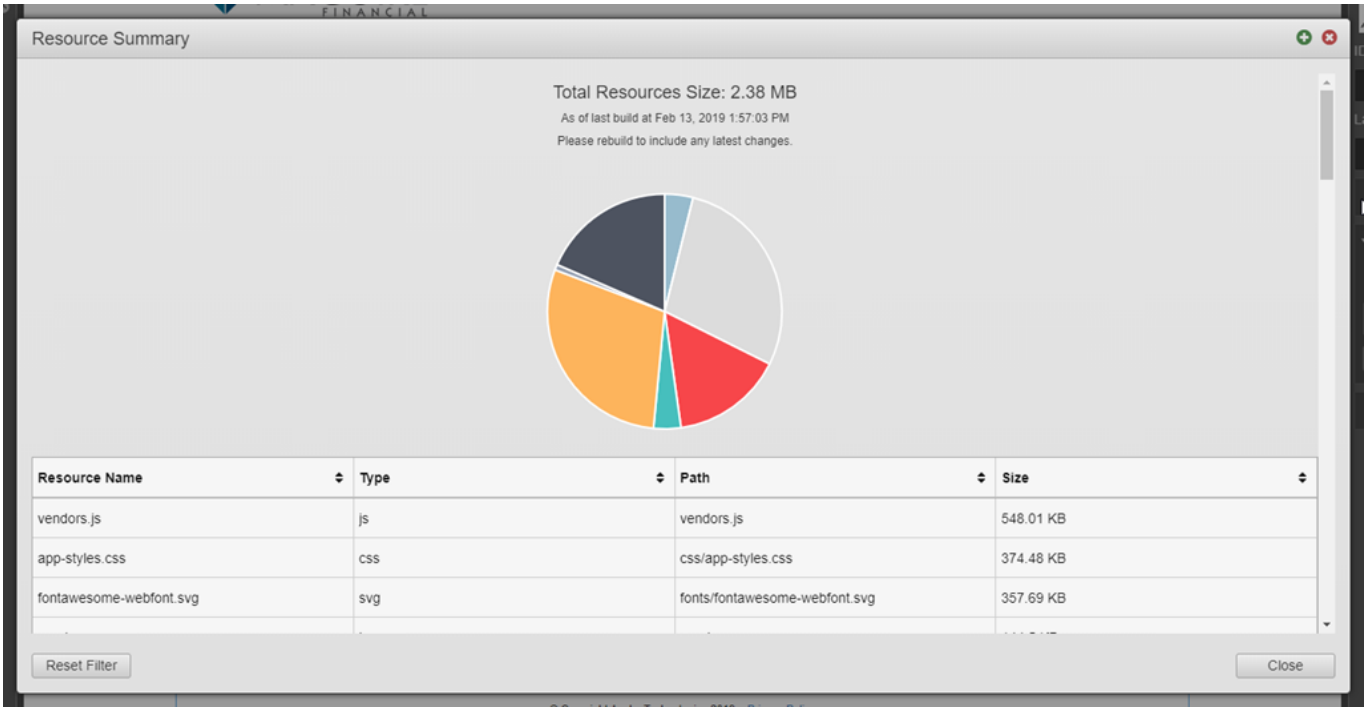
If you already have forms in development or production environments, Manager provides you with form rendering metrics to see how forms are performing.



This should help you ping point the forms that have the slowest load times, so you can do something about it.

Check Form Resource Summary

Now, when you know which forms are loaded slowest, you can open each form in Maestro to check what resources it uses. This is a quick way to identify which parts of a form might be causing performance issues.



You can sort by the Size column to identify the largest resources.

Analyze Page Load Time

There's a number of tools available for analyzing the page load time of web pages. These are useful for understanding how different users might experience your forms.

Here's a few of the most popular options:

- GTMetrix
- PageSpeed Insights
- Pingdom
- Web Page Test

Each tool offers different metrics and ways of displaying those metrics, so it's worth running your forms through all of them to gain the most insight into their performance. In particular, look for large resources, unnecessary resources, resources that could be combined into a single HTTP request, and resources that aren't loading.

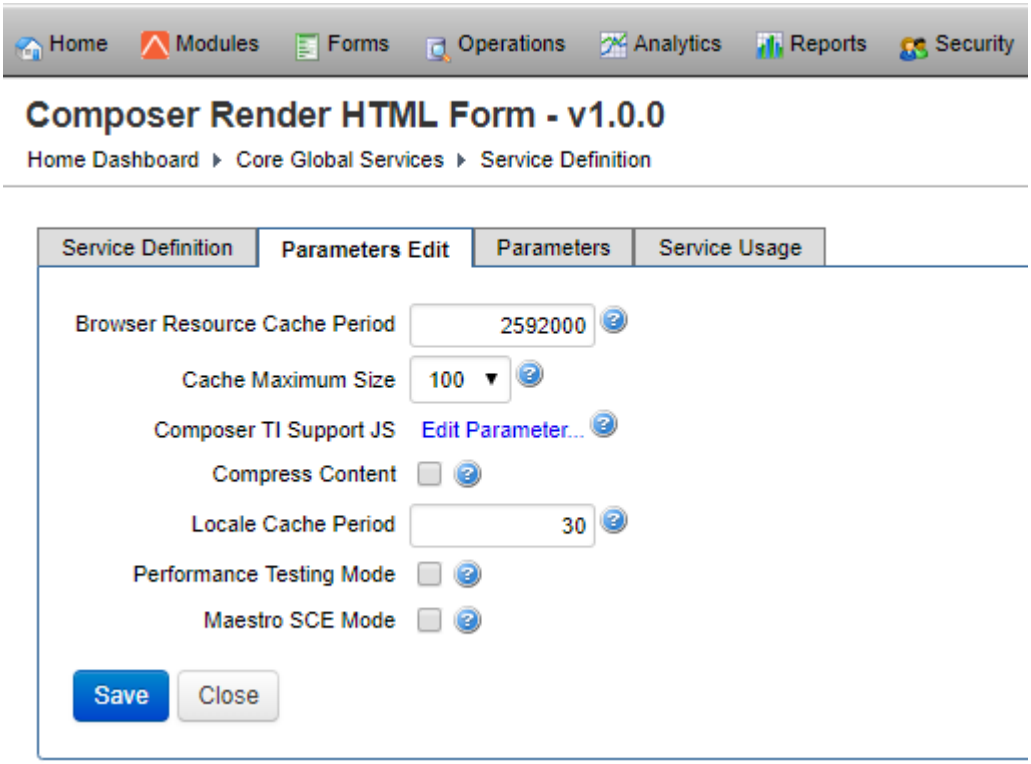
Enable Static Content Externalization

Manager 18.11 introduced a feature known as Static Content Externalization. When enabled, this feature serves most of a form's content via content distribution networks. As a result, content

is delivered to the end-user in parallel and from servers closer to where they live. In some cases, enabling this feature has immediately t page load times in half.

Enable GZIP Compression

Another option worth enabling is the Compress Content option, which uses GZIP compression to reduce the size of HTML, CSS, and JavaScript before it's delivered to users. This compression happens server-side, which means it does place more load on the server, but the significant reduction in page load time makes it a worthwhile trade-off.



Load Reference Data via Dynamic Data Calls

Forms often rely on a significant amount of reference data, such as a list of countries or states. Depending on the type of form you're creating, you might be able to load some of this data when the form loads, but consider the following scenario:

1. A user arrives at the form.
2. They select their country from a dropdown.
3. A different dropdown is populated with states from the user's country.

In this situation, not only would all of the countries need to be loaded as the form renders, but also all of the states for all of the countries. If there are other fields that rely on reference data, such as cities, a very large amount of data will need to be loaded upfront.

To reduce the impact of this, use dynamic data calls to load the relevant subsets of reference data after the form loads. You could, for instance, load a list of a country's states after the user has selected the country they live in. The time it takes for this additional HTTP request to complete will be significantly shorter than loading all of the data upfront.

Use Correct Image Formats

Image formats are designed for different purposes. If you use a format for the correct purpose, image sizes will be as small as possible while retaining their quality. If you use an incorrect format, the end-user will wonder why low-quality images are taking so long to render.

Here's a quick reference to understand when certain formats are appropriate:

Format	When To Use
JPEG	Images with large color palettes, such as photos.
PNG	Web graphics with a small color palette or transparency.
GIF	Animated images.

Other formats are available, such as Google's WebP format, but the formats in the above table are the easiest to implement without having to consider browser compatibility.

Compress Images

To further reduce the file size of images, use a compression tool such as ImageOptim.

Image compression tools work by removing unnecessary meta-data embedded in image files and running clever algorithms that imperceptibly lower the quality of images while drastically reducing their file sizes.

NOTE

You can use ImageOptim via the command line, allowing you to integrate image optimization into your development workflow.

Perform Load Testing

Analyzing the page load time of a form is an excellent way to establish a baseline of your form's performance, but it doesn't necessarily indicate how your form will stand-up under real-world use. Once a form (and the server on which it's hosted) is placed under the load of simultaneous users, its performance can degrade.

To prepare for this, consider using a load-testing service:

- Eggplant
- Load Impact
- Loader

These services direct a large amount of traffic to a server in a small amount of time, allowing you to determine how your form will perform under heavy use. This makes it easier to discover bottlenecks that need to be accounted for before putting a form in front of customers.

What Else?

It's rare for any web page to load as fast as it can possibly load. There is almost always something else that can be done to save a few extra milliseconds here and there. With this in mind, a priority should be to monitor the performance of your forms using the tools listed in this topic. As the tools evolve, technologies improve, and best practices shift, this ongoing awareness will pay dividends.

Next, learn about the Maestro form accessibility.

Maestro Form Accessibility

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 19.11 This feature was updated in 19.11.

Journey Maestro provides some basic, component-level accessibility automatically. For instance, linking a component's label text with its input field. However, a Maestro Template Designer and Form Builder need to perform the following additional tasks to achieve maximum form's accessibility:

- Use Sequential Page Navigation
- Use Page and Section Headers
- Mark-up Lists
- Add Alternate Text to Images
- Do Not Use Background Images
- Use Fieldsets
- Do Not Use Label Placement Left
- Check Color Contrast
- Do Not Use Placeholder Text
- Do Not Replace Component Labels with Text
- Do Not Disable Anything
- Use Aria Label

NOTE

We recommend configuring as much accessibility features in a template as possible.

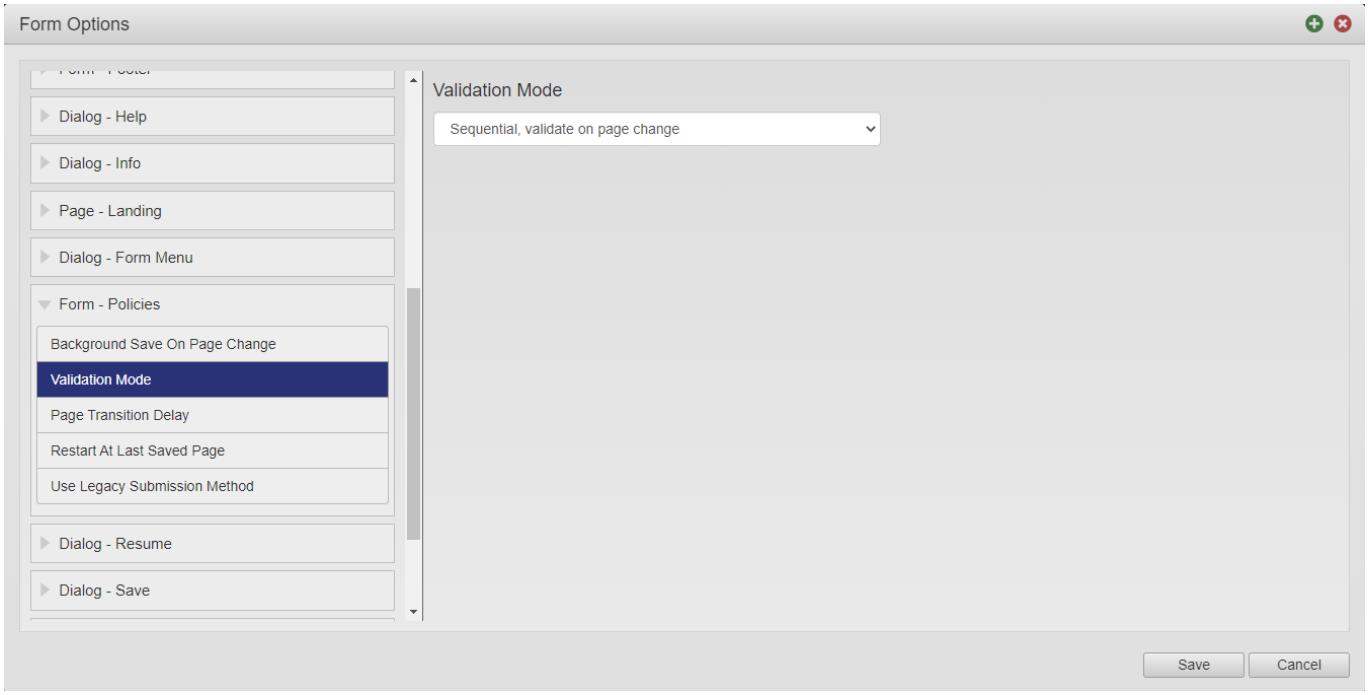
Let's look at each accessibility configuration in details.

Use Sequential Page Navigation

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 doesn't perform full form validation until they attempt to submit it from the final page. This can cause a potential problem that will be difficult to correct for multi-page forms.

To configure Sequential Page Navigation, select Form Options > Form - Policies > Validation Mode.



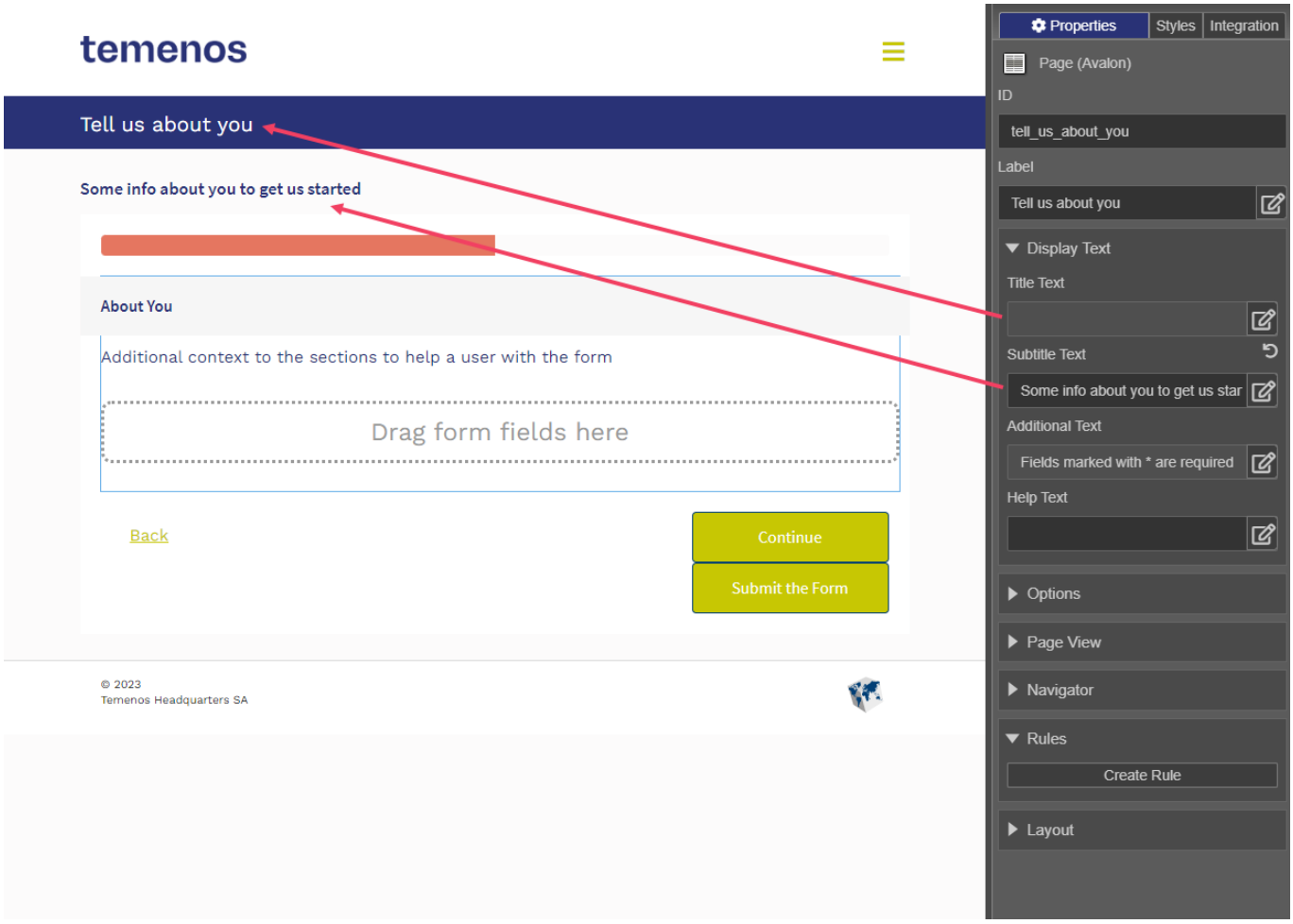
NOTE

In this example, we use the Avalon template. The Maguire template has a slightly different form option's categories.

Use Page and Section Headers

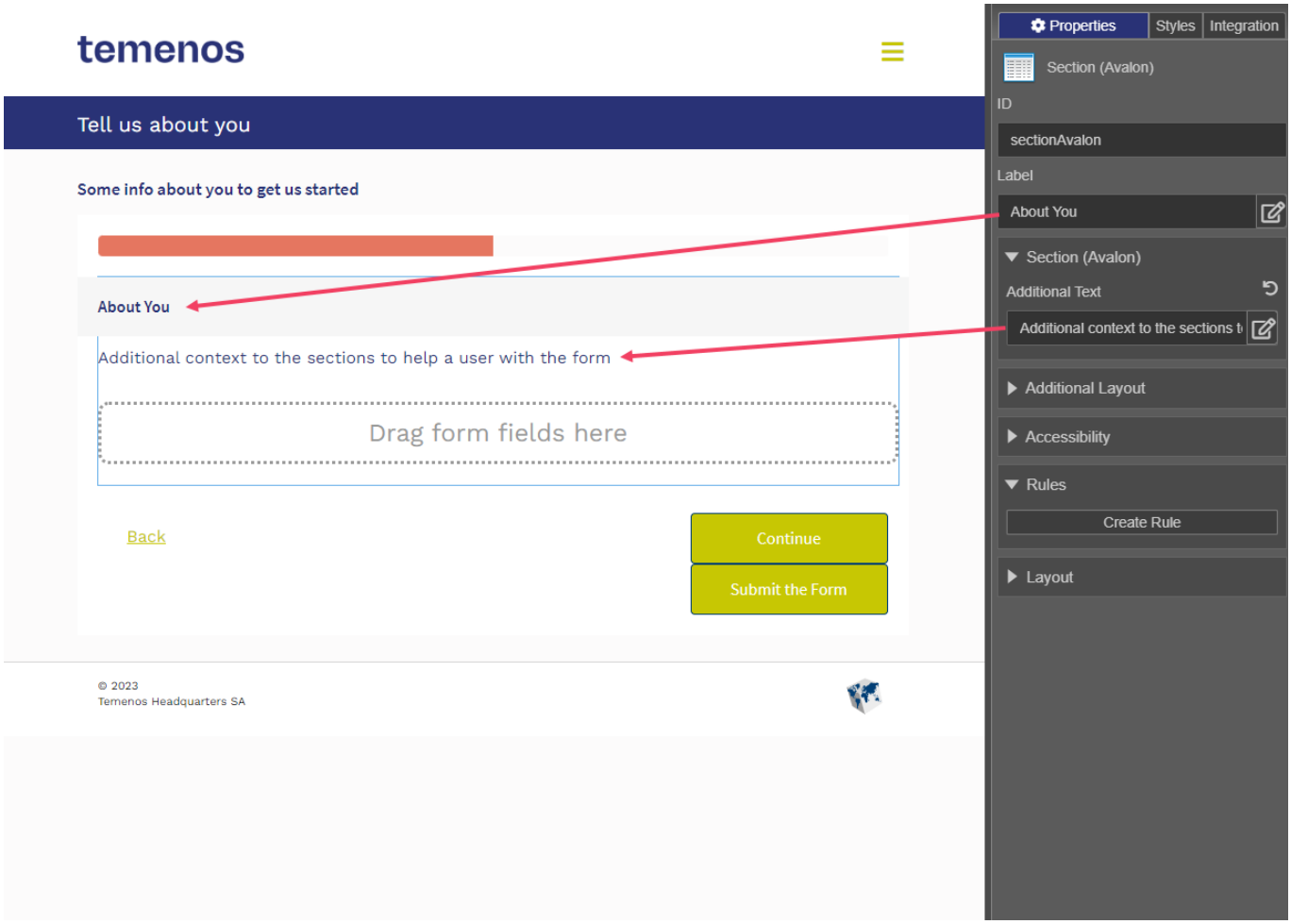
Correctly marked-up headings are provided automatically by the page and section headers at levels 1, 2 and 3. Use page and section headers and do not manually create them with text display fields.

To configure page's headers, select Page > Properties > Display Text and use the Title Text and Subtitle Text fields.



In this example, the Tell us about you will be rendered as the `<h1>` element. The Subtitle Text will be rendered as the `<h1>` element as well.

To configure section's headers, select Section > Properties > Label. Also, use the Additional Text field to specify any extra content for a form user.



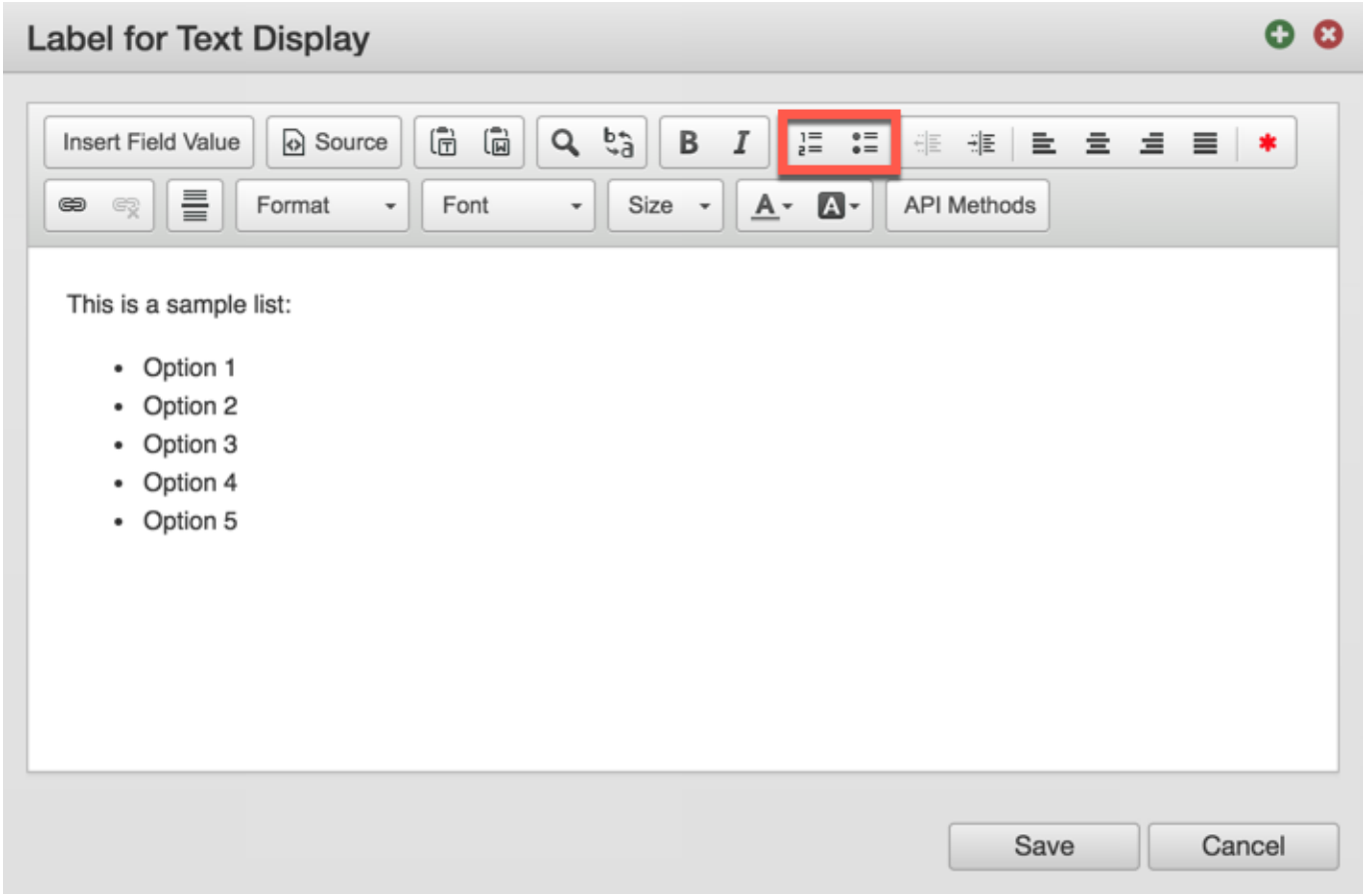
In this example, the About You label will be rendered as the `<h3>` element, whilst the Additional Text will be rendered as the `<div>` element.

Mark-up Lists

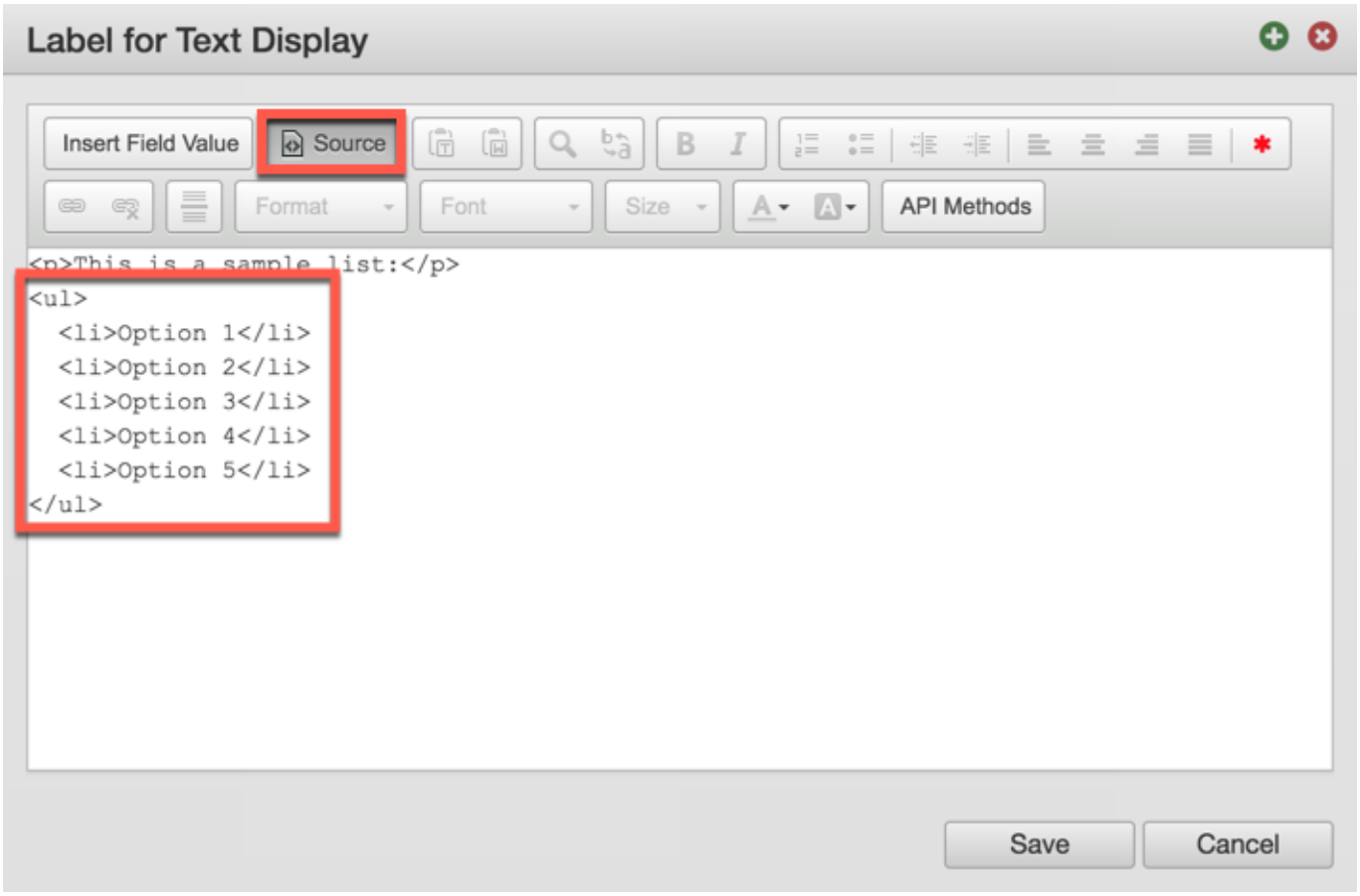
When creating lists, it is important to ensure that the correct HTML mark-up is used. When copying and pasting content from external sources (e.g. Word Documents) the mark-up is often lost and replaced by characters representing the numbers or bullets in the list. Whilst visually this may look OK, the lack of correct HTML mark-up will prevent screen readers from interpreting the list as a list.

Using the built in rich text editor you can add lists using the ordered or un-ordered list buttons shown below:

Select Text Display > Label to configure it.

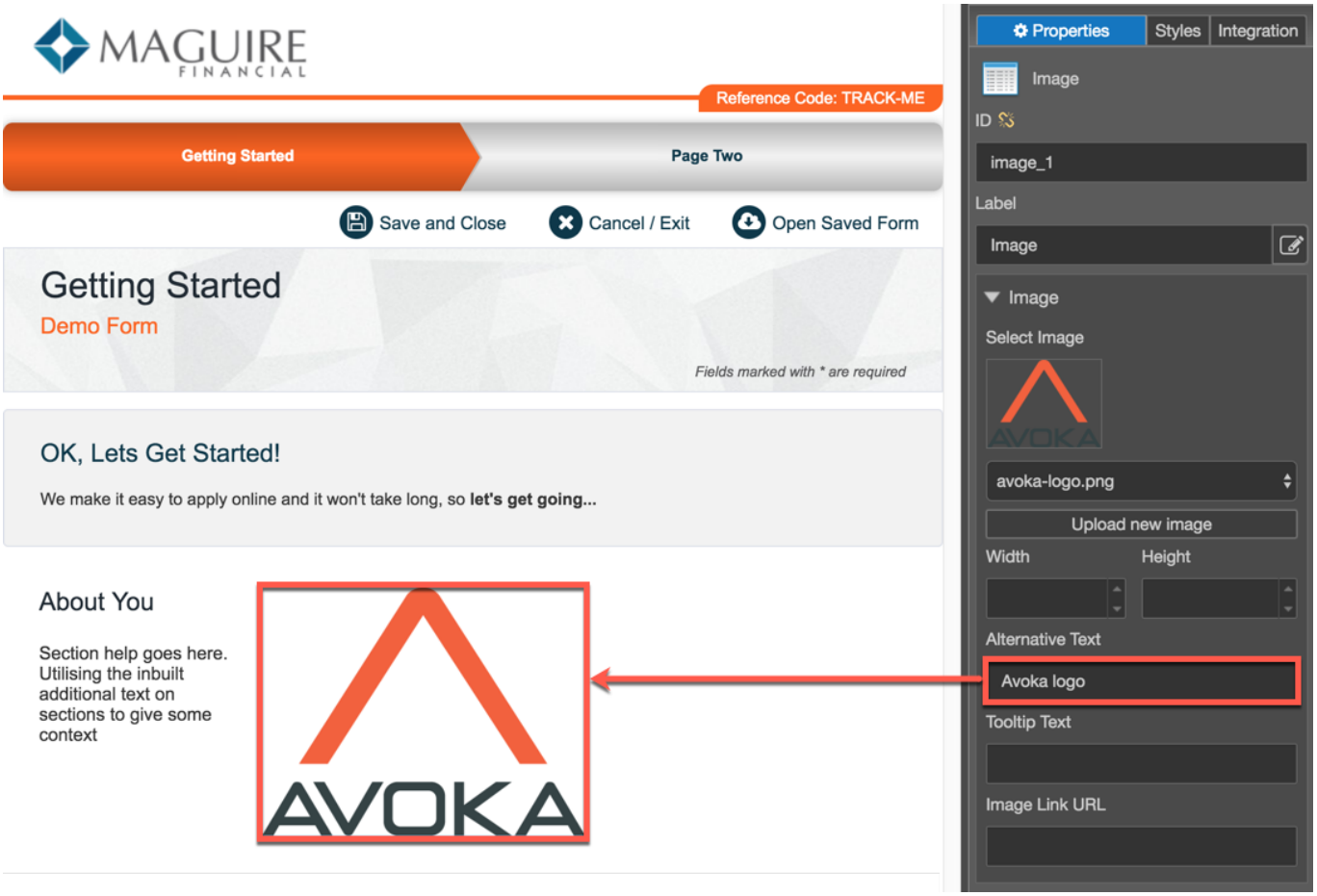


If copying and pasting content containing lists, review the source and verify that correct HTML mark-up has been used:



Add Alternate Text to Images

Select Image > Properties > Image > Alternative Text to configure it.



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

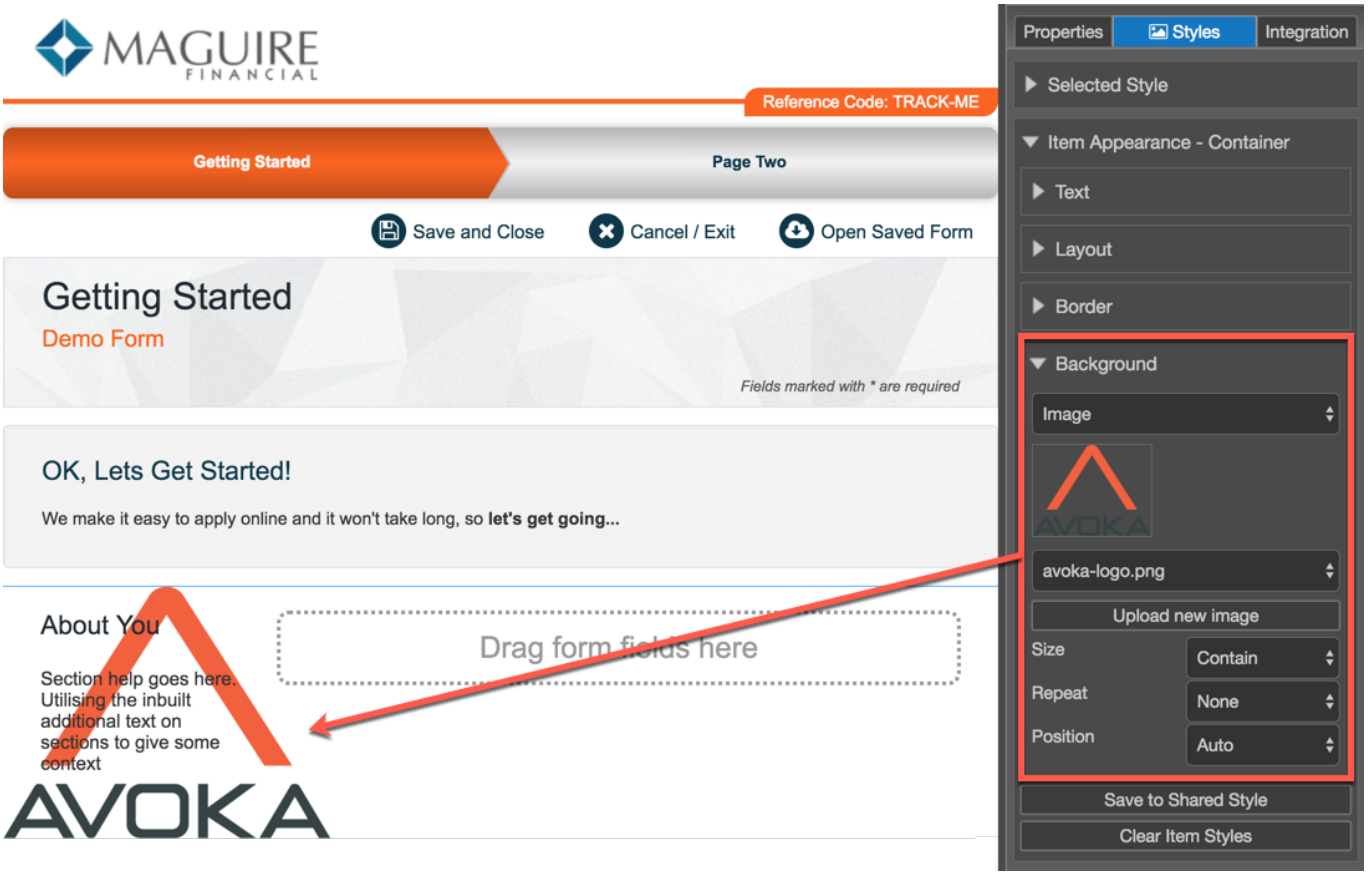
Decorative images do not convey information to the user and are typically used to make the form more visually attractive by adding visual components such as separators and textured backgrounds. An image is also considered decorative if it conveys information that is also fully provided by surrounding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

Informative

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.

Do Not Use Background Images

Select Component > Styles > Background to configure it.

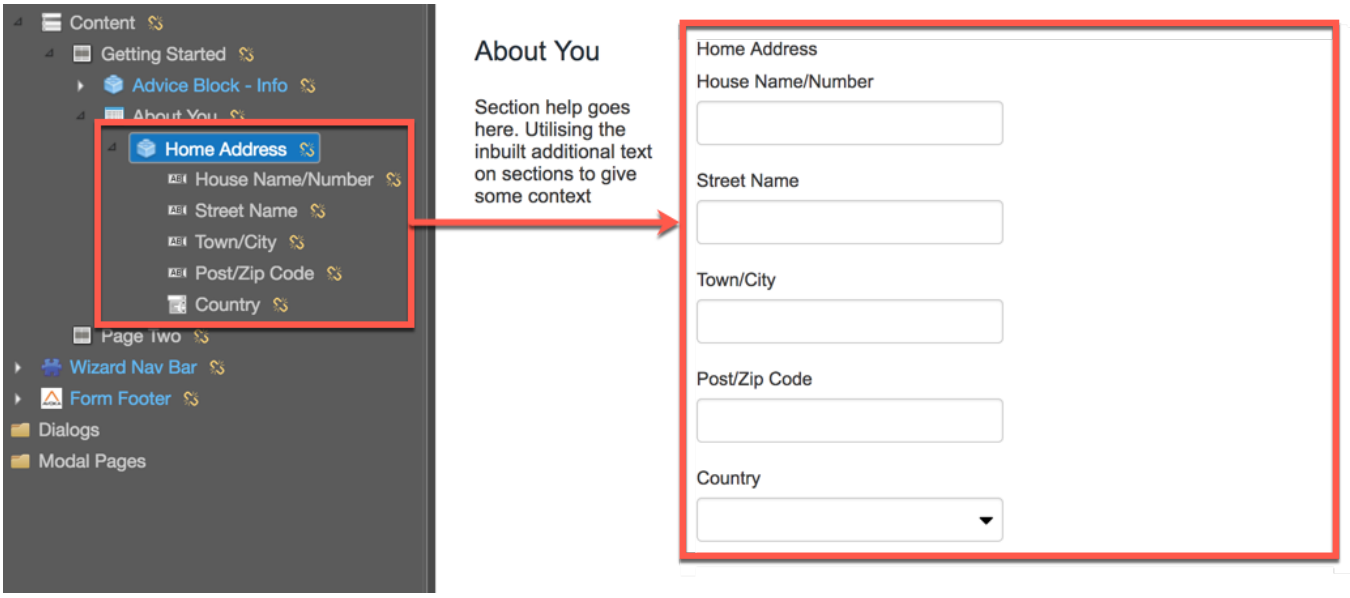


Informative images should not be used as a background image as it is not possible to provide them with alternate text and some browsers remove background images when put in high contrast mode. It is fine to use decorative images as a background image.

Use Fieldsets

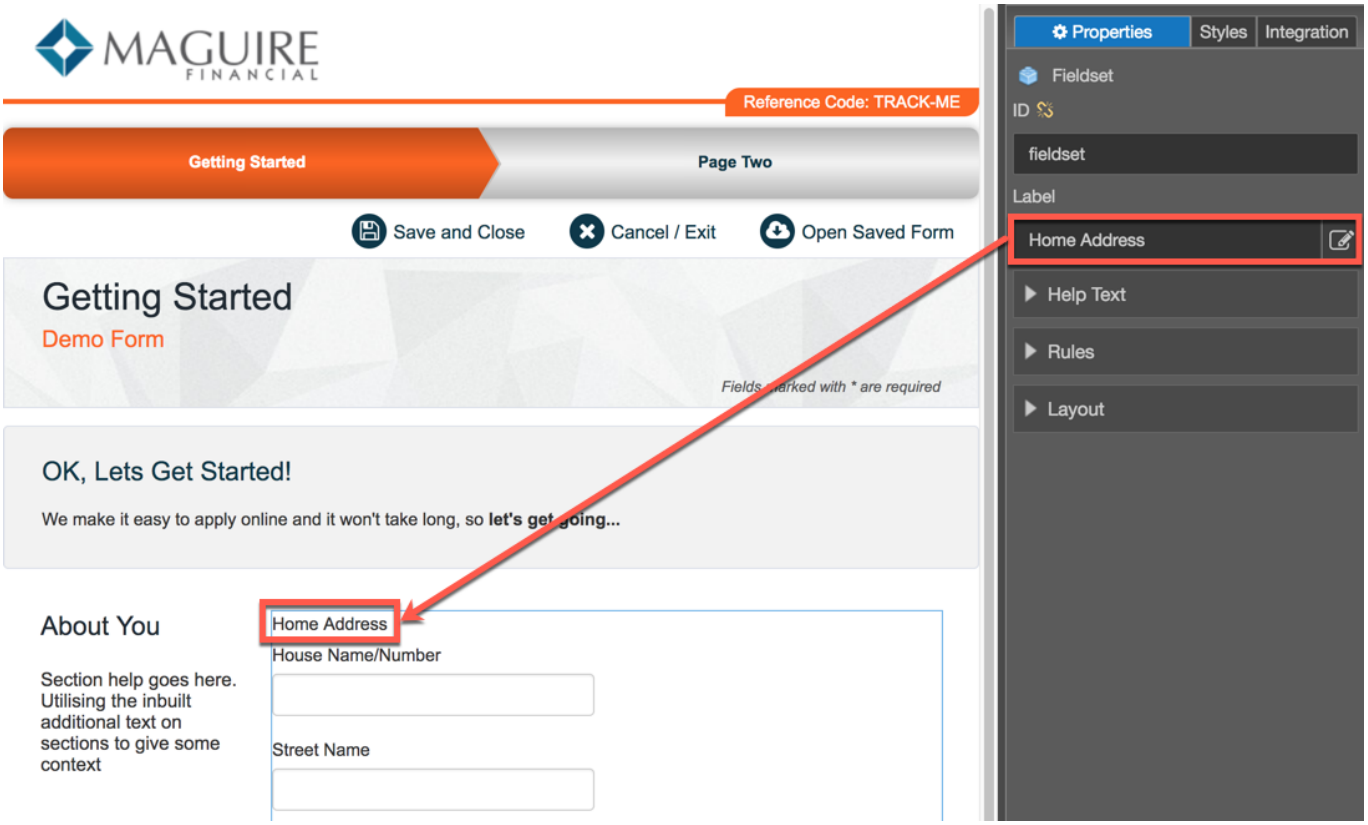
Fields that are not stand-alone and only make sense in the context of a group of fields should be contained within a Fieldset component. For instance, when using:

- Multiple Checkbox components, such as contact preferences
- Multiple components, such as telephone region code and number, home address and so on

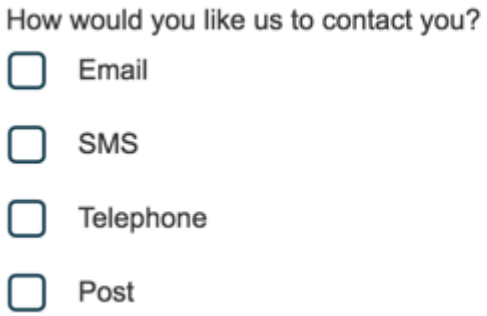


A meaningful description of the group should be provided in the Fieldset's label (legend text), which is displayed on-screen as a heading:

Select Fieldset > Properties > Label to configure it.



The fieldset's legend text will be used by assistive technologies to inform the user of the context for fields within the fieldset. For instance, tabbing to a checkbox button whilst a screen-reader is running might inform the user that they are currently focused on a checkbox button with caption Email for the fieldset How would you like us to contact you?:

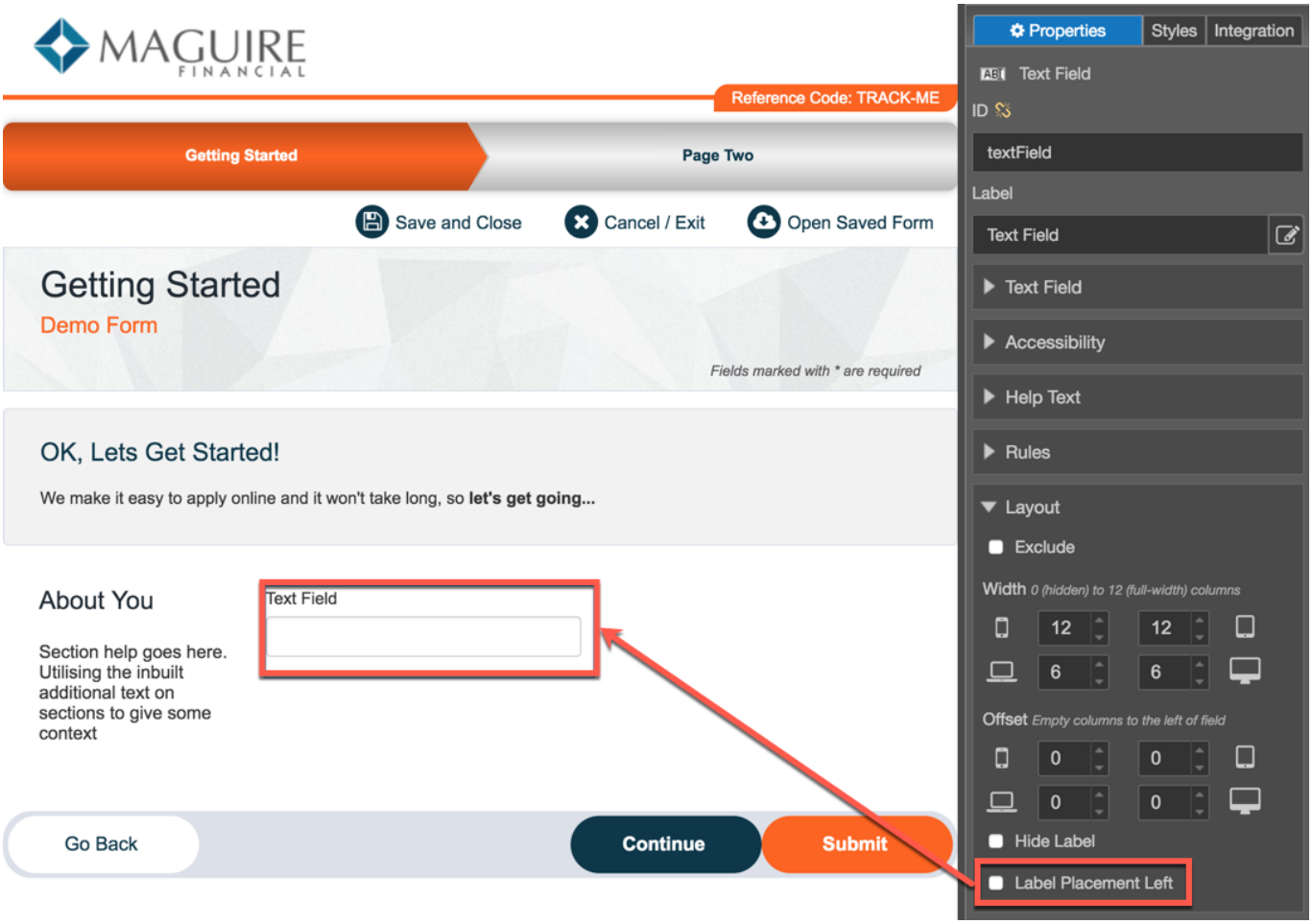


Without the fieldset, the user would only be informed that they are focused on a checkbox button with caption Email which is clearly insufficient. To avoid repetition, the majority of assistive technology will only inform the user of the fieldset legend for the first field focused within the group, regardless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

It should be noted that when using the Radio Button Group component that there is no need to use a fieldset as this component includes one already.

Do Not Use Label Placement Left

Select Field > Properties > Layout > Label Placement Left to configure it.



Label placement above the field (Maestro default) 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 label without having to move the cursor too far and reduces the chances of finding the wrong field 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 label placement results in additional, unnecessary movement as the sighted user's eye jumps around between labels and inputs that are less aligned.

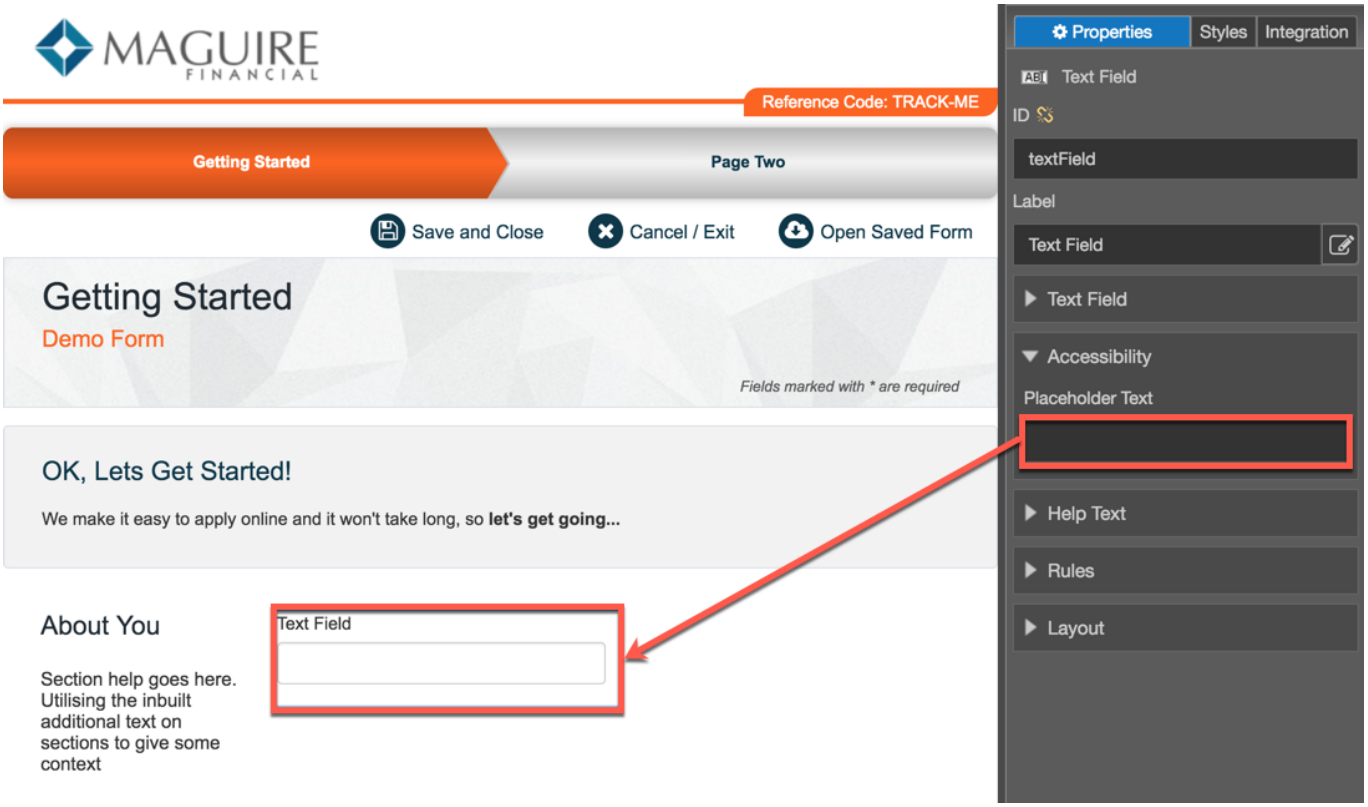
Check Color Contrast

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 TPGi's Color Contrast Analyser.

Do Not Use Placeholder Text

Select Field > Properties > Accessibility > Placeholder Text to configure it.



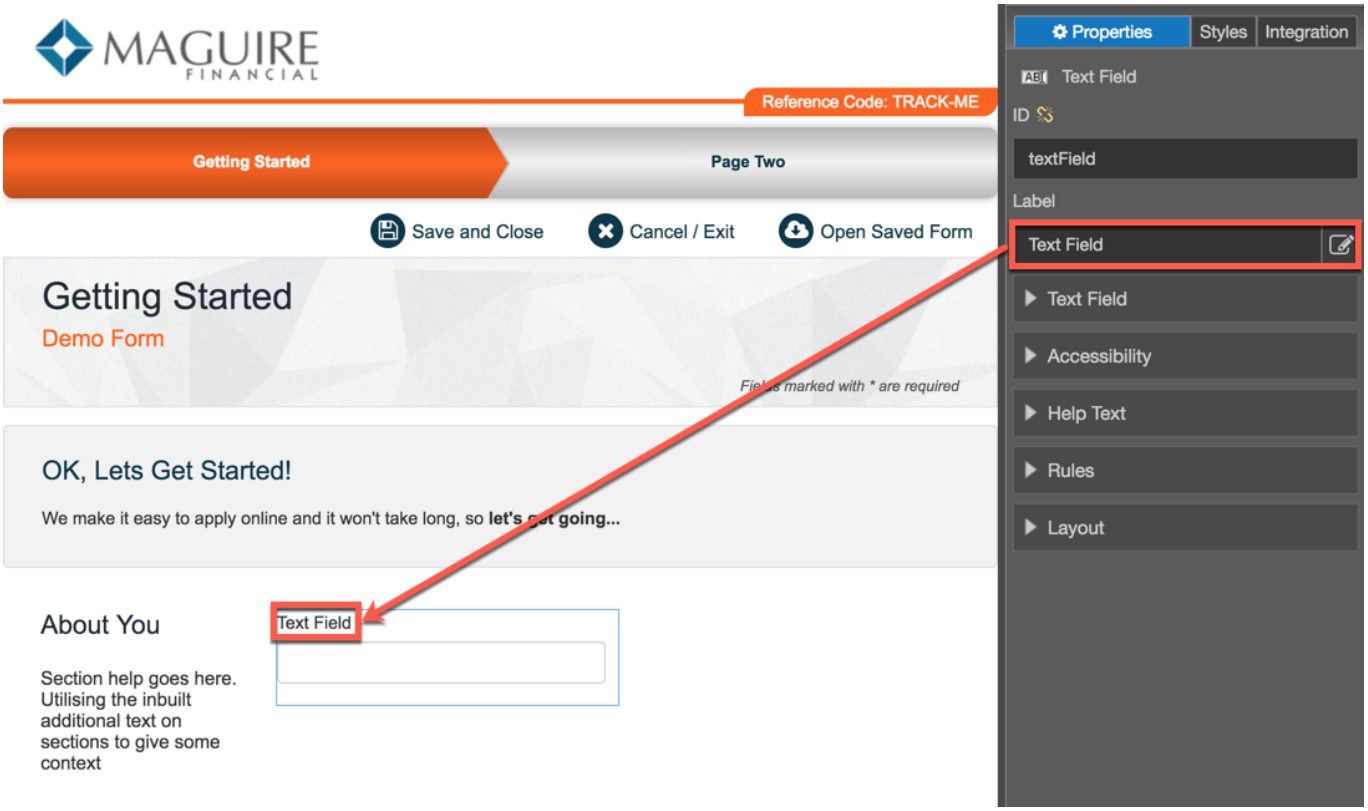
Placeholder text is the light grey text that is displayed in a field until data is entered at which point it is removed.

Placeholder text is not accessibility compliant due to the lack of consistent support in assistive technologies. Some assistive technologies 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 technologies do the correct thing in only notifying the user of the placeholder text when it is visible on-screen.

Do Not Replace Component Labels with Text

Select Component > Properties > Label to configure it.



It can be tempting to hide the built-in field label and create a replacement with a separate text display component in order to achieve layouts such as a label 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 label sets focus to its field.

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

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 users 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 assistive technologies will have no knowledge of the existence of the continue button at all which is likely to lead to confusion of what action is expected of them in order 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.

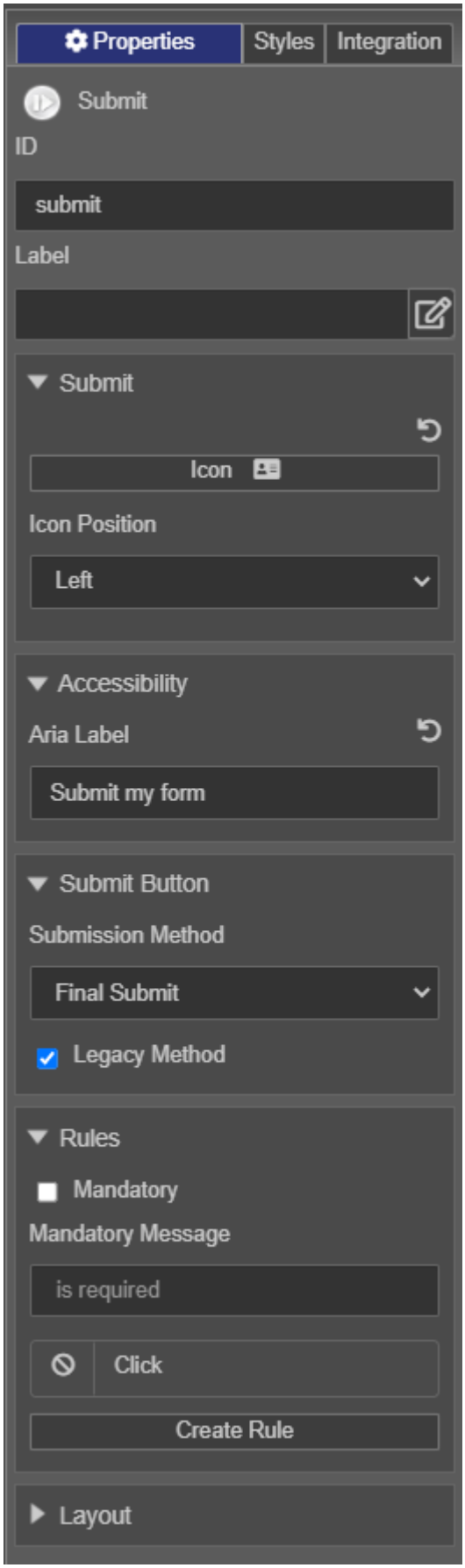
The other common alternative approach to disabling components is to hide them.

Use Aria Label

The `aria-label` HTML attribute allows you to provide an accessible name for an element so users of assistive technology will be able to recognize it. You should use `aria-label` if the element doesn't have a visible name you can reference, for example, `aria-label="Submit my form"`. For more information, see ARIA (Accessible Rich Internet Applications) documentation and ARIA specification.

Most of Maestro's components, for example, Icon Button, Second Level Navigator, and Submit, have the Aria Label property, which you can use to add text description.

To configure `aria-label`, select Component > Properties > Accessibility >Aria Label:


Properties
Styles
Integration
Submit
ID
submit
Label
Submit
Icon
Icon Position
Left
Accessibility
Aria Label
Submit my form
Submit Button
Submission Method
Final Submit
Legacy Method
Rules
Mandatory
Mandatory Message
is required
Click
Create Rule
Layout

In this example, we use the Submit component with an empty label and an icon to show on a Submit button. After you publish the form, the `aria-label="Submit my form"` property is added to the `<button>` element so assistive technology, such as a screen reader, will interpret it accordingly.

Next, learn about rule templates and rule helpers in a form.

Rule Templates and Rule Helpers in a Form

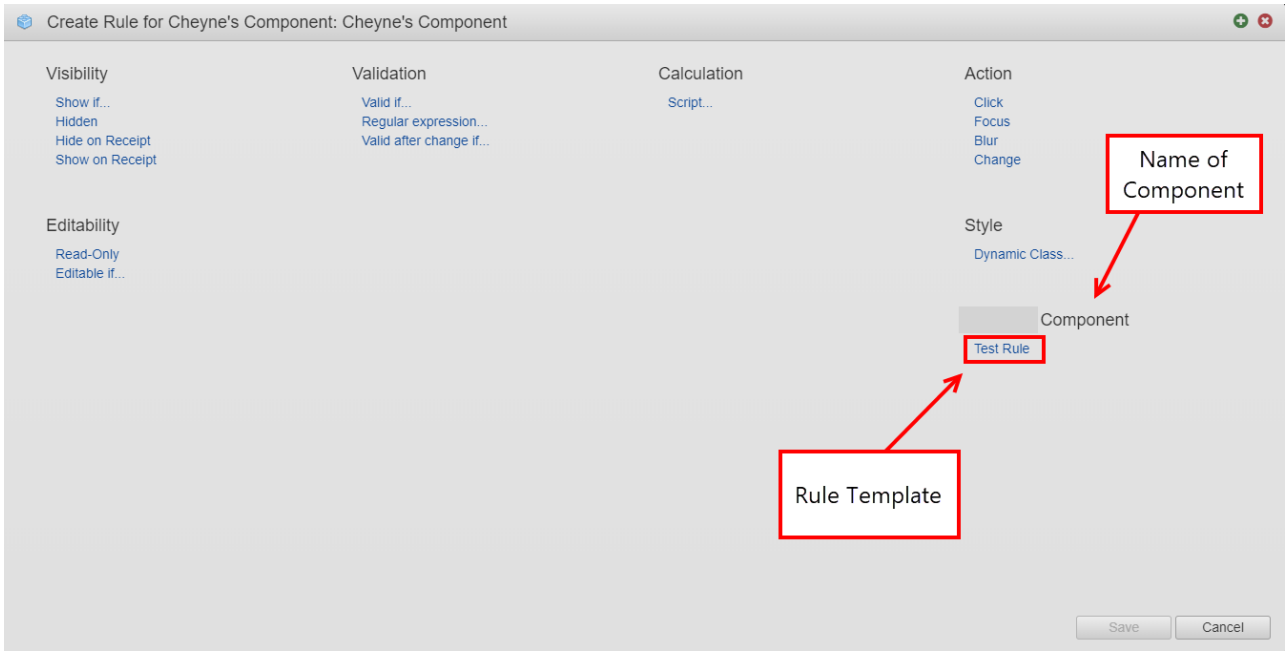
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to configure a component with the Rule Templates and a Rule Helpers component options, which can be then used in a form.

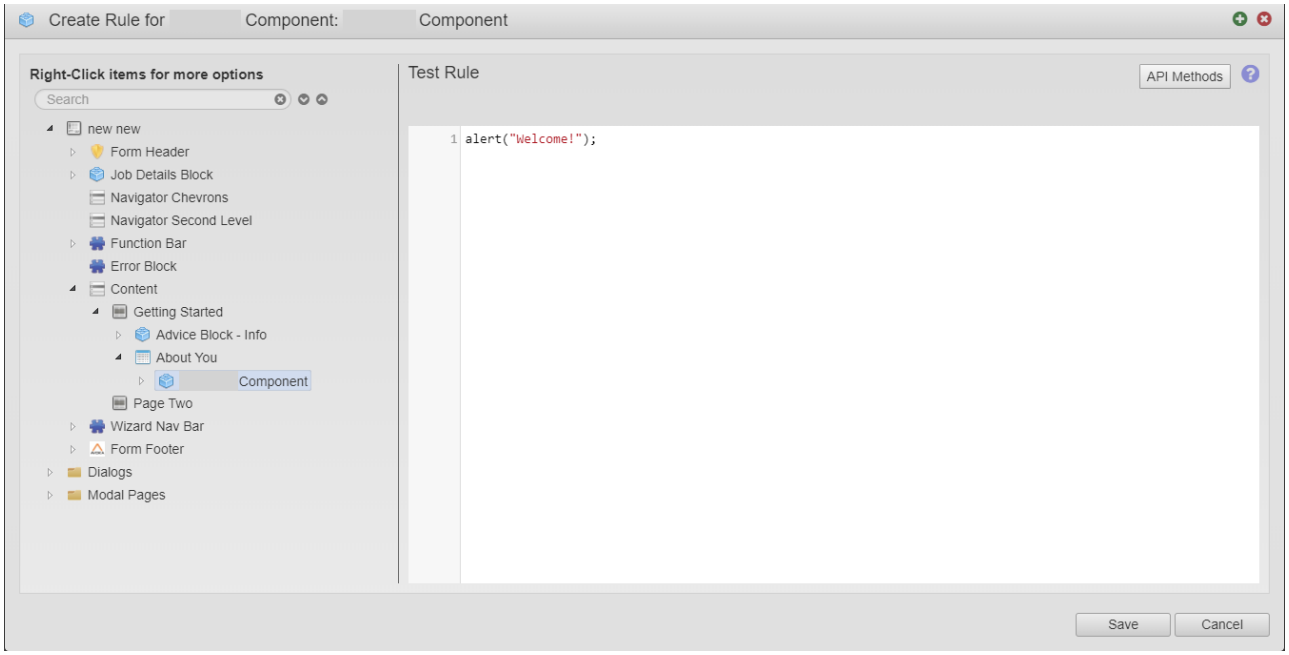
Once you have configured a component with the Rule Templates and a Rule Helpers component options, you can use it in a Journey Maestro form.

To configured a component with the Rule Templates and a Rule Helpers component options:

1. Open a project in your organization.
2. Select a form that has a component with a rule helper and rule template configured.
3. Click Edit Form.
4. Add this component to the form.
5. Select this component.
6. Click the Properties tab and click Create Rule. It displays the Create Rule Dialog. if the rule template has been configured correctly, it should be a selected rule option.



7. Select Rule Template under the name of the customized component, for example Test Rule.
8. Edit a rule as a JavaScript, for example, show a welcome message when the rule is triggered from other component.



9. Click Save.
10. Add another component to the form that executes this rule. For example, you may add a button with a Click rule that executes a JavaScript when the button is clicked: `Form.fireRule("testRule", item, data)`

Next, learn about component labels.

Form Options

MaestroThe UI design product. | Form Builder | 21.11 This feature was updated in 21.11.

Maestro gives you flexibility to configure component's properties differently for each form in your template using the Form Options functionality.

NOTE

Compare this to brand's Item Properties where you can set component's properties differently for each brand.

When it comes to using Form Options, the best practice is that a Template Designer creates form options with some default values in a template and group them logically under various categories, so a Form Builder can easily find them. The Form Builder updates form options' values to implement required form functionality. You can think about a form option as a shortcut to a particular property of a component. The property name of a component doesn't change while you create the form option, so you can always find out what that component is.

Let's illustrate it with an example. Imagine you need to remove the navigator from forms that only contain one or two pages. Since the navigator is configured in the template by the Template Designer, you are not able to customize its appearance or properties, but you are able to hide it from the form.

When the Template Designer locks down a form by setting extension points at the template level, it is likely that these extension points will still give the Form Builder access to basic form option configuration, such as hiding the navigator or save button, so the Form Builder can select these options in the Form Options window.

Another benefits of using Form Options is that it allows a single form option to set the properties on multiple template's components at the same time. For example, the Template Designer can use the Logo form option to configure the company logo, which appears in many places in a form and all modal pages, in one place, instead of creating multiple options and asking the Form Builder to set all of them to the same image. Using the form option allows to link all these places to a single option, so changing it once updates all of the image properties through the form that display the company logo.

Here are a few more notes on working with Form Options:

- If you create a form option in the template, the form will have the ability to override the template, which can lead to inconsistencies between forms
- If you remove a form option, this may or may not remove the property override from the form. Ideally you would want it to remove the override from the form, but the JSON may remain in the form and this required some manual updates.
- You can not and should not add form options to components that exist within extension points - the form developer will already have a local copy of these components and will be able to make all the changes they need.

The Form Options interface allows you to:

- Add properties
- Configure properties
- Remove properties

Next, learn how to add a form option.

Add a Form Option

MaestroThe UI design product. | Template Designer | 21.11 This feature was updated in 21.11.

Maestro allows you to create and add new form options to a template. Usually, a Template Designer creates form options with some default values in a template and group them logically under various categories, so a Form Builder can easily find them. Then, when a form option is not needed any longer, a Template Designer can delete it.

When you create a new form option in a template, Maestro adds it to a template JSON file.

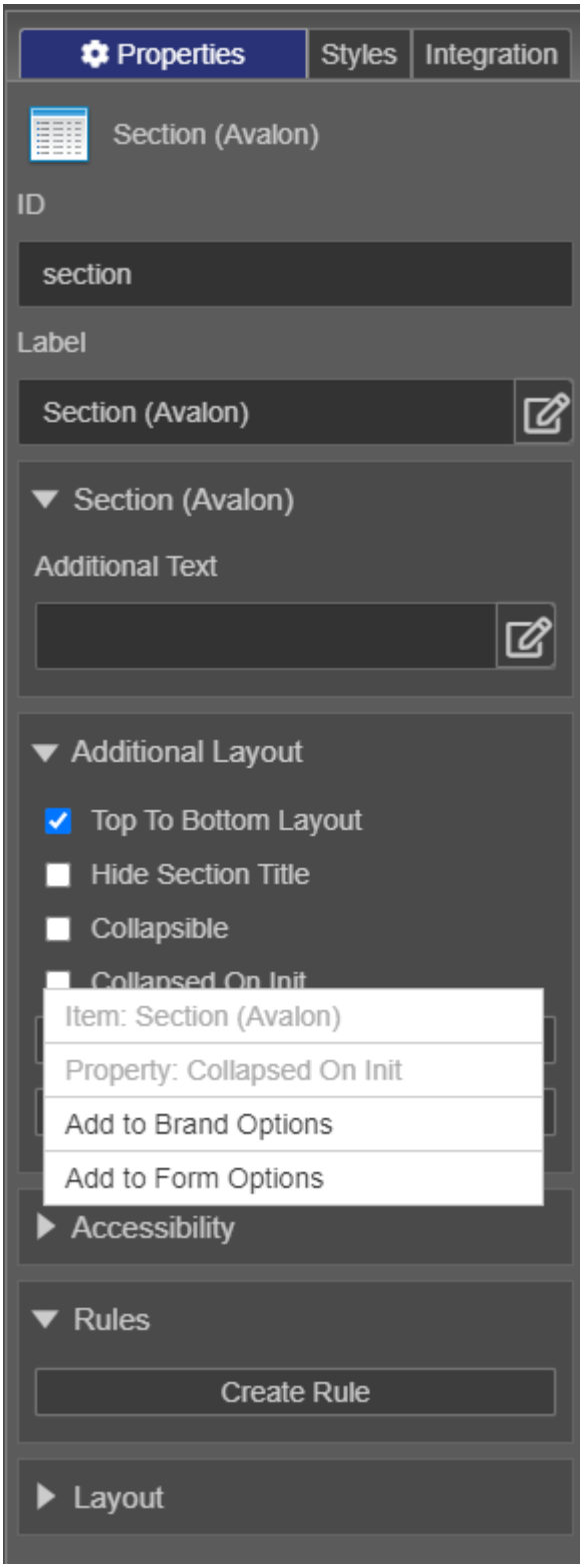
NOTE

Compare this to brand's Item Properties where you can set component's properties differently for each brand.

Add Properties

To add a component's property to Form Options in a template:

1. Open a template in the Maestro editor.
2. Select a component in the Design mode and click the Properties tab.
3. Right click on a non-rules property you want to add, for example, Label or Hide Section Header, and select Add to Form Options.



For rules properties, right click on the Create Rule button and select Add to Form Options.

NOTE

You also have a choice of adding a component's property to brands. However, if you've already added the property to Form Options, you can't add it to brands. You must remove it from Form Options first.

4. Select how you want to add a property from the Add as a radio button group:



   - New Option to add as a new property. You can use this option in most cases.
   - Linked Item to Existing Option to add as linked to an existing property and then select an existing property from the Select Existing Option dropdown list. You can use this option if the property you are adding needs to be linked to another property that has already been added, for example, an image component used to show the logo in the form might have the Image property linked to other instances of the image component used to show the logo on modal pages. This allows you to create 1 form option which will update the property of multiple components.
5. Select a category to use for the property from the Category radio button group:
   - New Category to create a new category for the new property and type a category name in the field.
   - Existing Category to add a new property to the already existing category and then select an existing category from the dropdown list.

NOTE

Different templates may have different categories to group form options.

NOTE

Adding categories help you group and navigate properties, especially, when there are many of them.

6. Type a name for the property in the Label field
7. Click Save. The property is added to Form Options, where you can edit it.
8. Click Save in the Maestro toolbar to save the template.
9. Click Publish to make the template changes to be available to forms that use it.

NOTE

The template changes may not be available to a forms depending on where the extension points are in the template - anything that's inside the extension point (that can be changed in the form) can't be updated by the template

Remove Properties

If a property has already been added as a form option, then you'll have the option to remove it. The default Avalon and Maguire templates have form option created for you, so you can only remove them, not to add them.

NOTE

When you delete a form option, Maestro removes it from the template JSON file.

To remove a property from Form Options:

1. Open a template in the Maestro editor.
2. Select a component in the Design mode and click the Properties tab.
3. Right click on Label and select Remove from Form Options. The property is removed from Form Options in the template.

Next, learn how to configure a form option.

Configure a Form Option

MaestroThe UI design product. | Form Builder | 21.11 This feature was updated in 21.11.

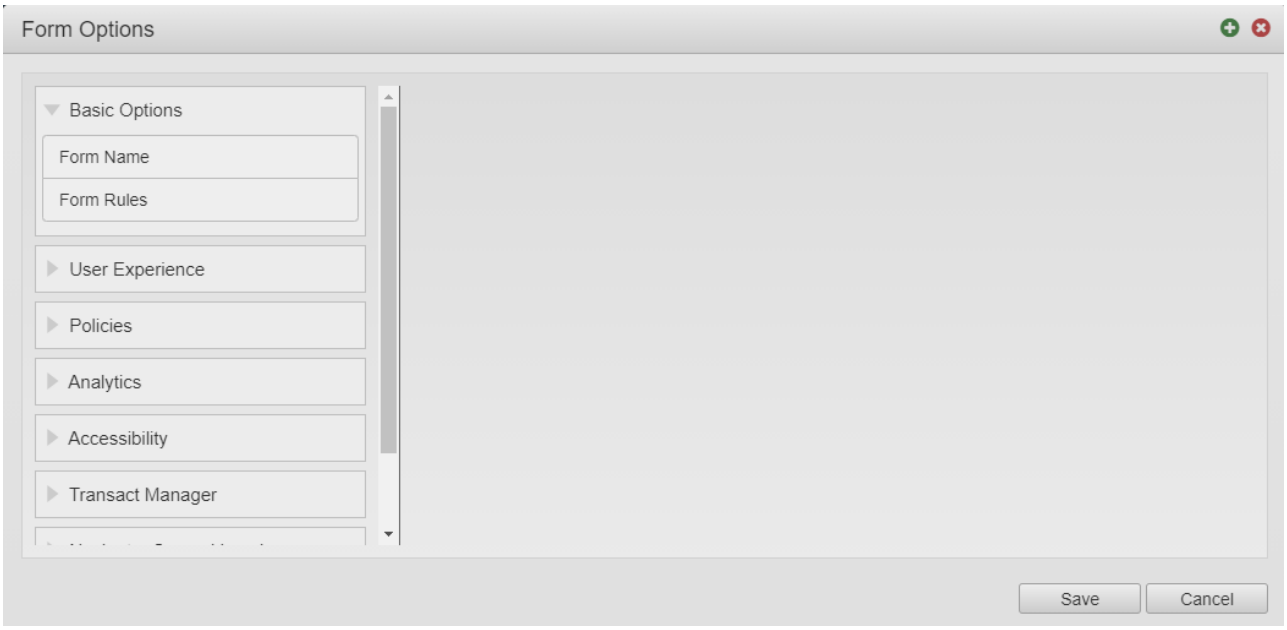
Maestro allows you to configure form options in a form. Usually, after a Template Designer created form options with some default values in a template, a Form Builder can updates their values to implement required form functionality. When you change form option's property in your form, Maestro directly updates its form JSON file with the component's property value, which overrides the template's form option.

Configure Properties

To configure a form using the Form Options:

1. Open the form in the Maestro editor and select Design.
2. Click Form Options to view all available form options to your form. Different templates may have different categories that a Template Designer used to group form options, which is illustrated with the Maguire Template and Avalon template below:

   Maguire Template



   Avalon Template



NOTE

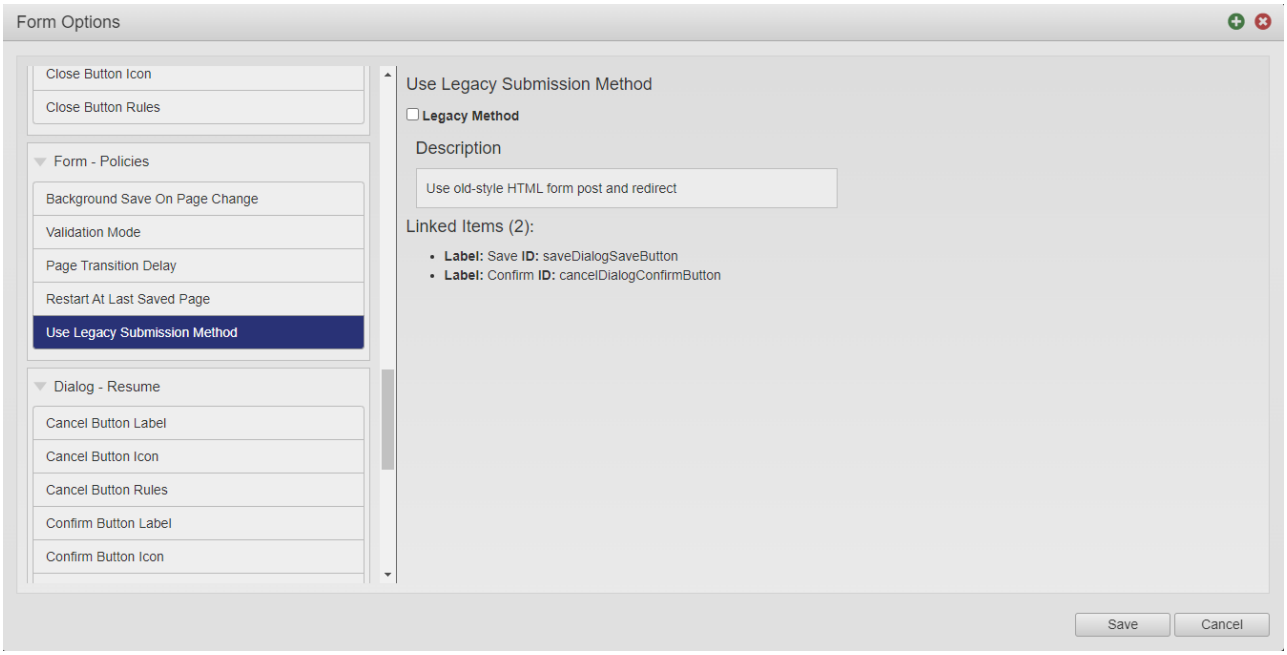
The list of categories you can see also depends on your access rights, which are set by the Template Designer in the template.

3. Expand the relevant category and select the option you want to view and update. The most common form options are:
   - Form Options > Form Name
   - Form Options > Form Rules
   - User Experience
   - Form - Policies
   - Analytics
   - Accessibility
   - Transact Manager
   - Navigation
   - Navigator Second Level
   - Wizard Navigation Bar

NOTE

If a form option that you require is not available, speak to your Template Designer to add it to your template.

4. Update the property as required. Check if the form option is used in other components, which will be listed in the Linked Items sections, as shown below:



   In this example, the Use Legacy Submission Method form option is used by 2 components identified by:

   - Components' label names - Save ID and Confirm ID
   - Components' IDs - `saveDialogSaveButton` and `cancelDialogConfirmButton`

5. Click Save to update the configuration.

NOTE

We recommend building and rendering a form to see how your changes are displayed to form users.

Next, learn how to change a form name.

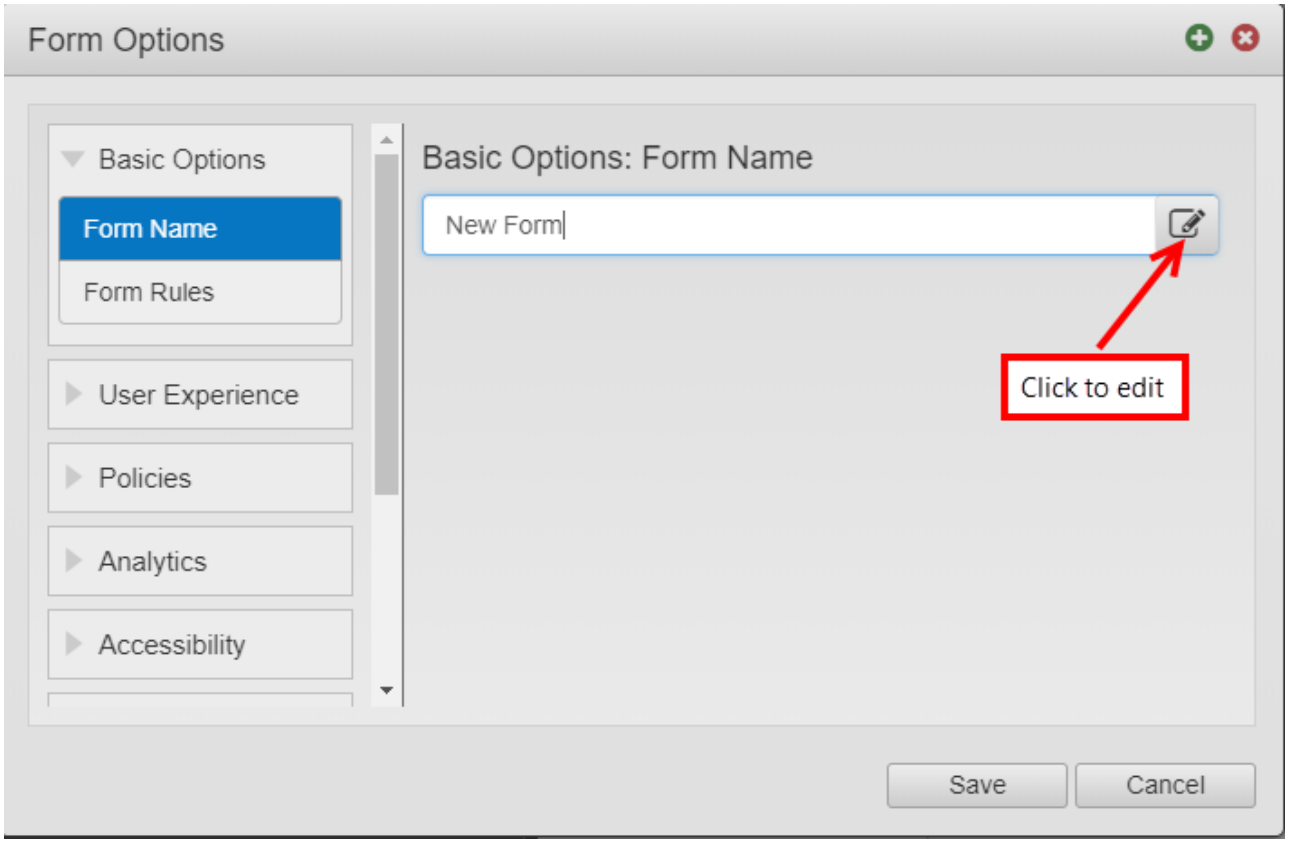
Form Name

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

From the form options window, you can configure a form name for a Maestro form. The Form Name basic option is used to edit a name of a form, which is a part of the form metadata[1]. However, the form name only impacts areas on the form that reference the form name metadata.

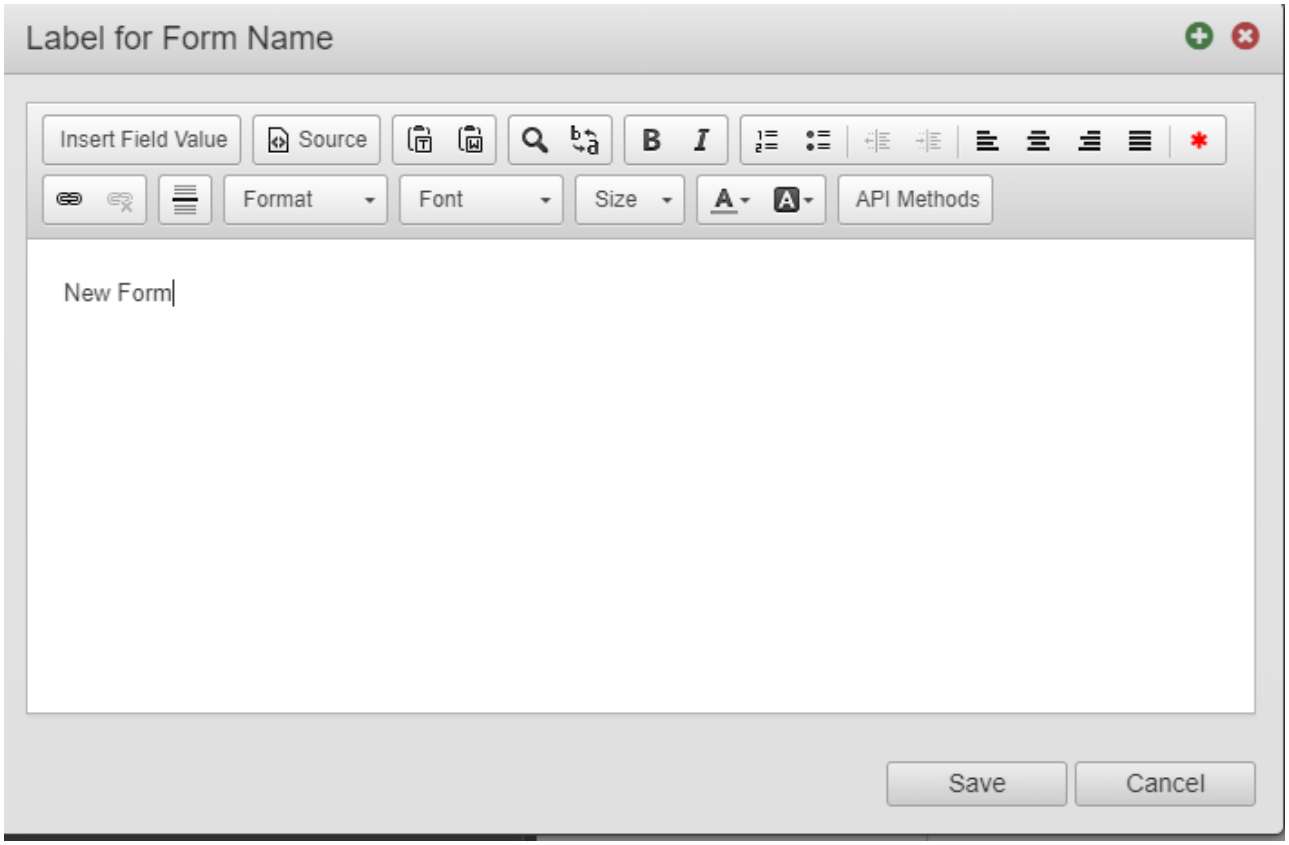
To change a form name from the form options window

1. Open a Maestro form and select Form Options.
2. Select Form Name.

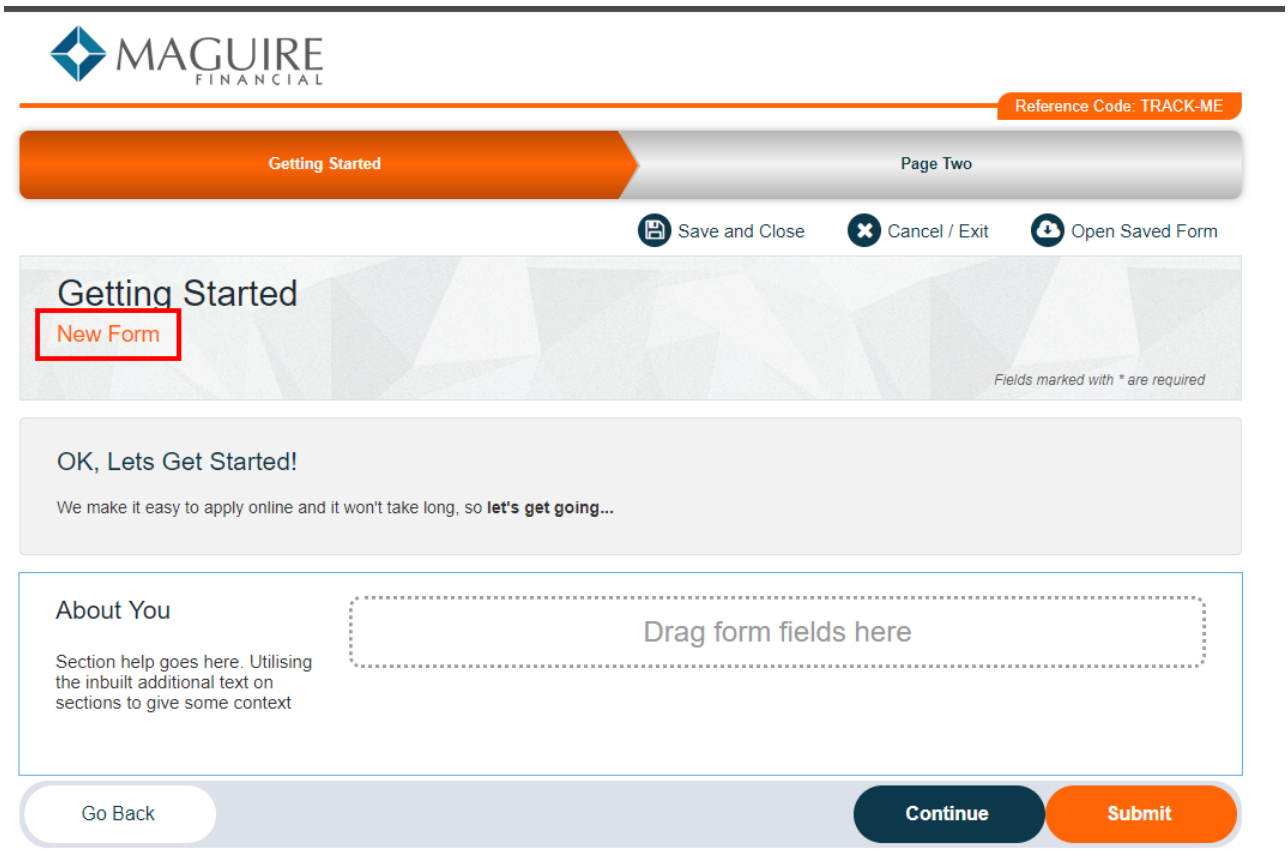


[1]Metadata is data that describes other data. Meta is a prefix that in most information technology usages means "an underlying definition or description." Metadata summarizes basic information about data, which can make finding and working with particular instances of data easier. For example, author, date created and date modified and file size are examples of very basic document metadata. Having the abilty to filter through that metadata makes it much easier for someone to locate a specific document.

3. Click Edit.



4. Edit the name of the form and click Save.
5. Click Build and Render Form to see how this rule is displayed when the form is built and rendered. In the default template the form name displays near the top of the form. This display may be different in your form depending on the template used.



Changing the form name in the Form Options doesn't change the form name displayed in the Maestro Dashboard. Likewise, editing the form name in the dashboard doesn't change the form name displayed and referenced in the form.

To configure the form name in the Maestro Dashboard:

1. Open a form in the Maestro Dashboard and click Details.
2. Edit the form name and description and click Save.

Next, learn how to create a form rule.

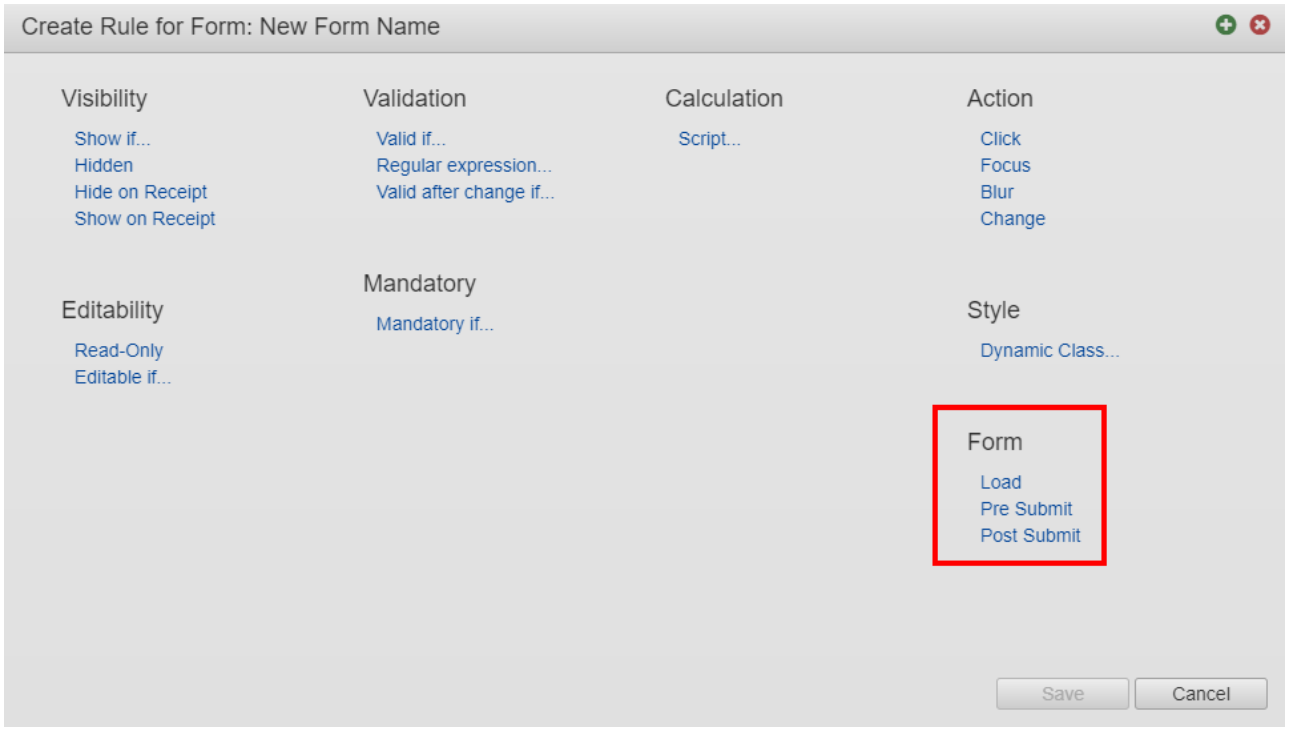
Form Rules

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to create business rules for a Maestro form using Form Options.

To create a business rule using the Form Options:

1. Open a Maestro form and click Form Options.
2. Select Basic Options > Form Rules and click Create Rule.



3. Select a rule from the available options:
   - Load – this rule is invoked when a form is loaded by a browser.
   - Pre Submit - this rule is invoked before the submit action is fired.
   - Post Submit - this rule is invoked after the submit action is fired.
4. Edit the JavaScript that implements the rule.
5. Click Save.

Next, learn how to configure user experience.

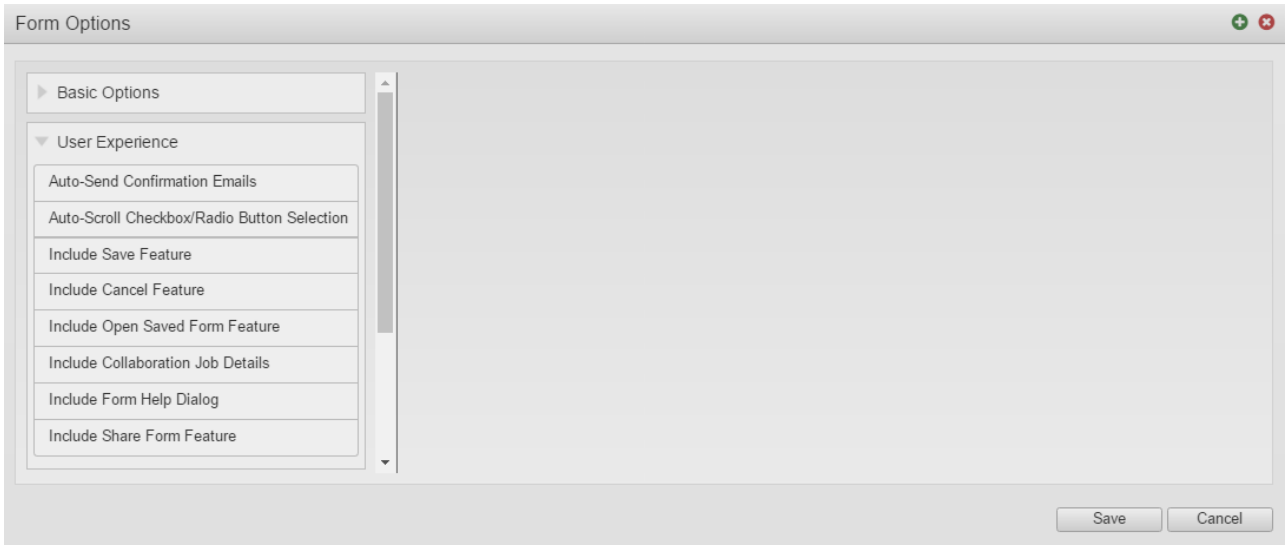
User Experience

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to configure User Experience using Form Options.

To configure User Experience using the Form Options:

1. Open a Maestro form and select Form Options.
2. Select User Experience.



3. Select Auto-Send Confirmation Emails to automatically send emails to the user when the form is submitted and/or saved.
4. Select Auto-Scroll Checkbox/Radio Button Selection to enable the auto-scroll feature for checkboxes and radio button groups. If the user selects a checkbox or radio button group the page will auto-scroll so that the selected component is in the middle of the screen.
5. Select Include Save Feature to allow the user to save the form. The Linked Items list displays the areas within the form that are related to the save feature. Items may include Dialogs, Modal Pages or buttons.
6. Select Include Cancel Feature to enable the user to cancel the form without saving or submitting it. The Linked Items list shows features related to the cancel.
7. Select Include Open Saved Form Feature to enable the user to open a saved form. The Linked Items list shows features related to the resume.
8. Select Include Collaboration Job Details to include collaboration job details.
9. Select Include Form Help Dialog to include the form help dialog.
10. Select Include Share Form Feature to include share form feature.
11. Click Save.

Next, learn how to configure form's policies.

Policies

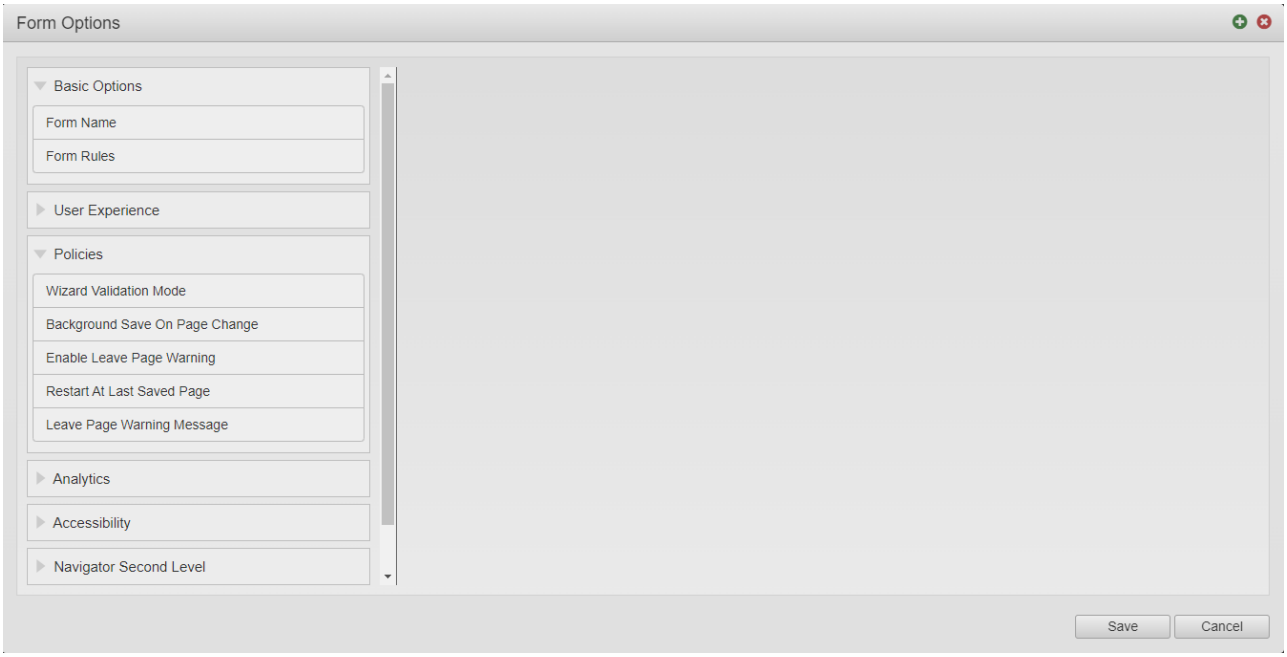
MaestroThe UI design product. | Form Builder | 21.11 This feature was updated in 21.11.

Maestro allows you to configure various form's policies using Form Options.

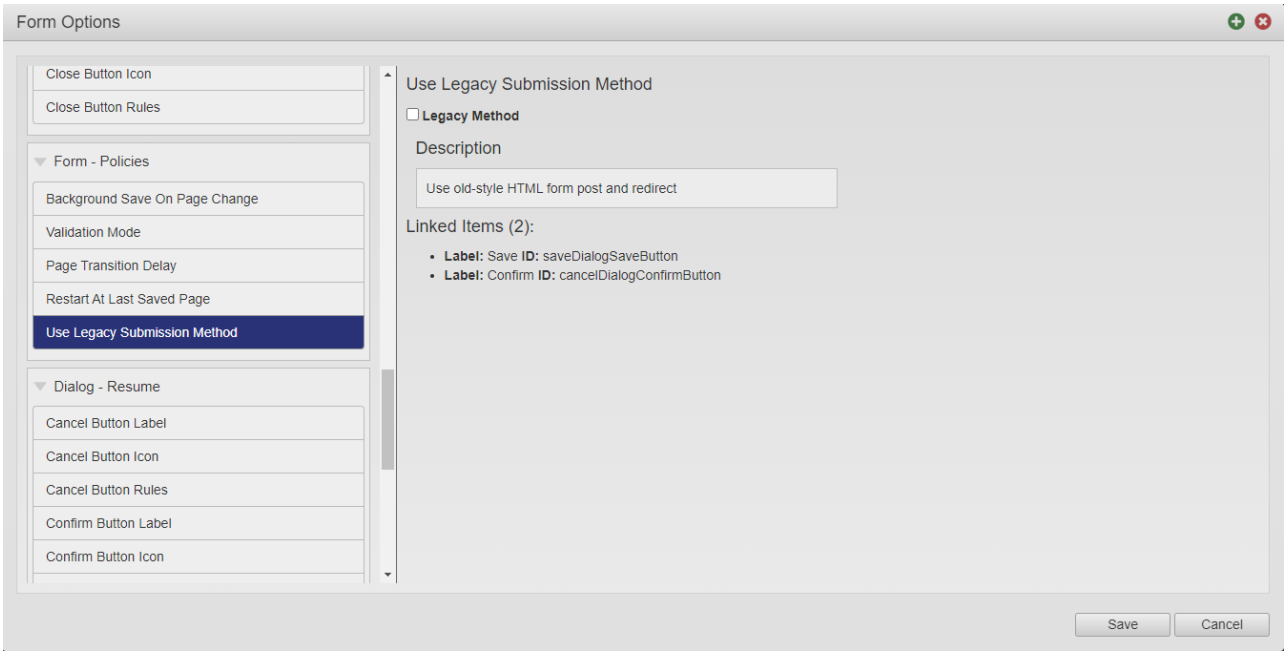
To configure form's policies:

1. Open a Maestro form and select Form Options.
2. Expand Policies or Form - Policies, as shown below for 2 most popular templates we provide:

Maguire Template



Avalon Template



3. Select Validation Mode to configure a page validation mode, which can be:
    - Unconstrained, validate on submit
    - Sequential, validate on page change
4. Select Page Transition Delay to update the default value for page transition delay.
5. Select Background Save On Page Change to enable a background form save on page change.
6. Select Enable Leave Page Warning to show a confirm dialog when a form user tries to leave a page.
7. Select Restart At Last Saved Page to make a form return to a page it was on after save and resume.
8. Select Leave Page Warning Message to update the default message for the leave page warning defined with the Enable Leave Page Warning setting.
9. Select Use Legacy Submission Method to use old-style HTML form post and redirect.
10. Click Save.

Next, learn how to enable analytics.

Analytics

MaestroThe UI design product. | Form Builder | 21.11 This feature was updated in 21.11.

Maestro allows you to add Analytics properties to a Maestro form using Form Options. You usually do this after you have updated the form's standard page tracking configuration in Manager.

Maestro has limited support for GTM[1], so it only checks for `window.gtag` or `window.ga` presence in the Page Controller and a Modal Page (when it is visible). For more information, see how to add Google Tag Manager in a Maestro form. However, Maestro is fully integrated with the Journey Analytics module where it send all analytics events t.

To add Analytics properties using the Form Options:

1. Open a Maestro form and select Form Options.
2. Select Analytics > Google Analytics Tracking ID and enter the Google Analytics Tracking ID. | 21.11 This feature was removed in 21.11
3. Select Analytics > Google Analytics Base URL and enter the Google Analytics Base URL, which is used to tag an event in GMT within a Page Controller and a Modal Page. The example of the GA script is shown below:

```
// Google Analytics - Page Tracking can add GA to the global window scope
if (!!window.gtag) {
	gtag('event', 'page_view', {
		page_title: newPage.label,
		page_path: baseUrl + newPage.id
	});
} else if (!!window.ga) {
	ga("set", {
		page: baseUrl + newPage.id,
		title: newPage.label
	});
	ga("send", "pageview");
}
```

Next, learn how to configure form accessibility.

---

[1]Google Tag Manager (GTM) 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.

Accessibility

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to configure form's accessibility using Form Options. The correct design and configuration of form's accessibility is crucial to ensure that more users have equal access to the form.

To configure accessibility using the Form Options:

1. Open a Maestro form and select Form Options.
2. Select Accessibility.
3. Select Enable Mobile Zooming to enable pinch-to-zoom on mobile devices. This is not normally needed with good responsive design, but some accessibility reviews require it.
4. Click Save.

NOTE

Each component has its own accessibility configuration with various set of options depending on a type of component. See the list of components to find a component you need and check what accessibility options it provides.

Next, learn how to configure Navigator Second Level.

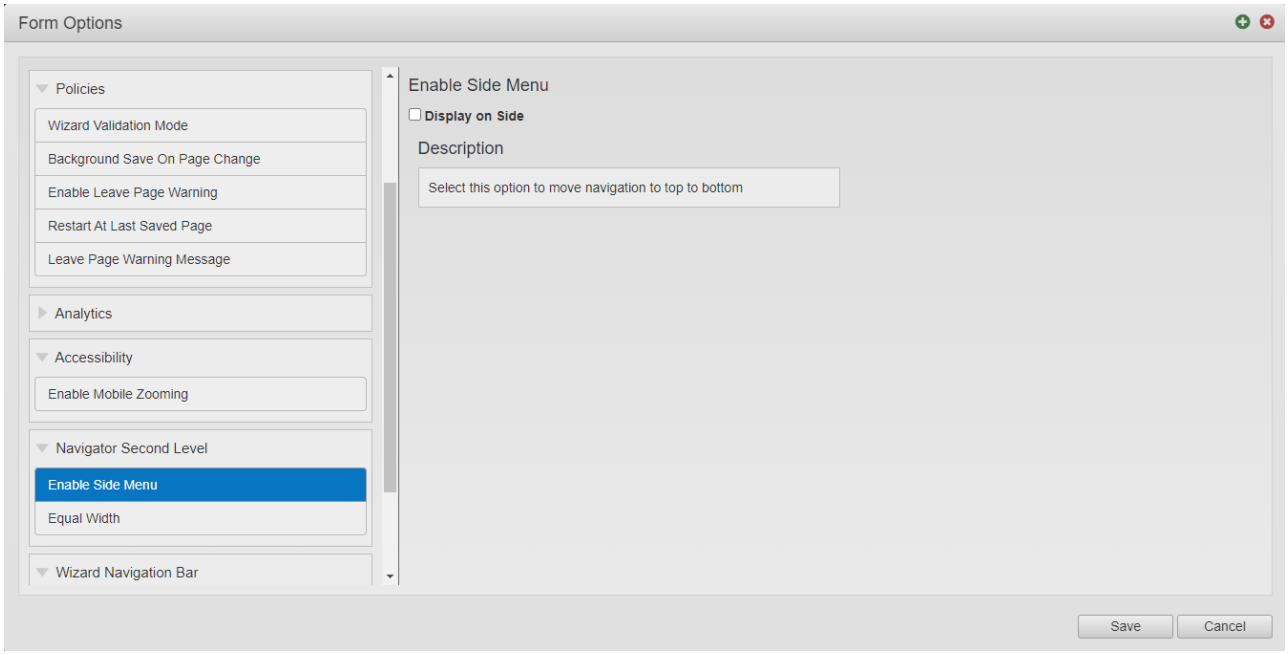
Navigator Second Level

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to configure Navigator Second Level in a Maestro form using Form Options.

To configure Navigator Second Level using the Form Options:

1. Open a Maestro form and select Form Options.
2. Select Navigator Second Level.



3. Select Enable Side Menu and select the Display on Side checkbox to move navigation to top to bottom.
4. Select Equal Width and select the Equal Width checkbox to ensure equal item width.
5. Click Save.

Next, learn how to configure Wizard Navigation Bar.

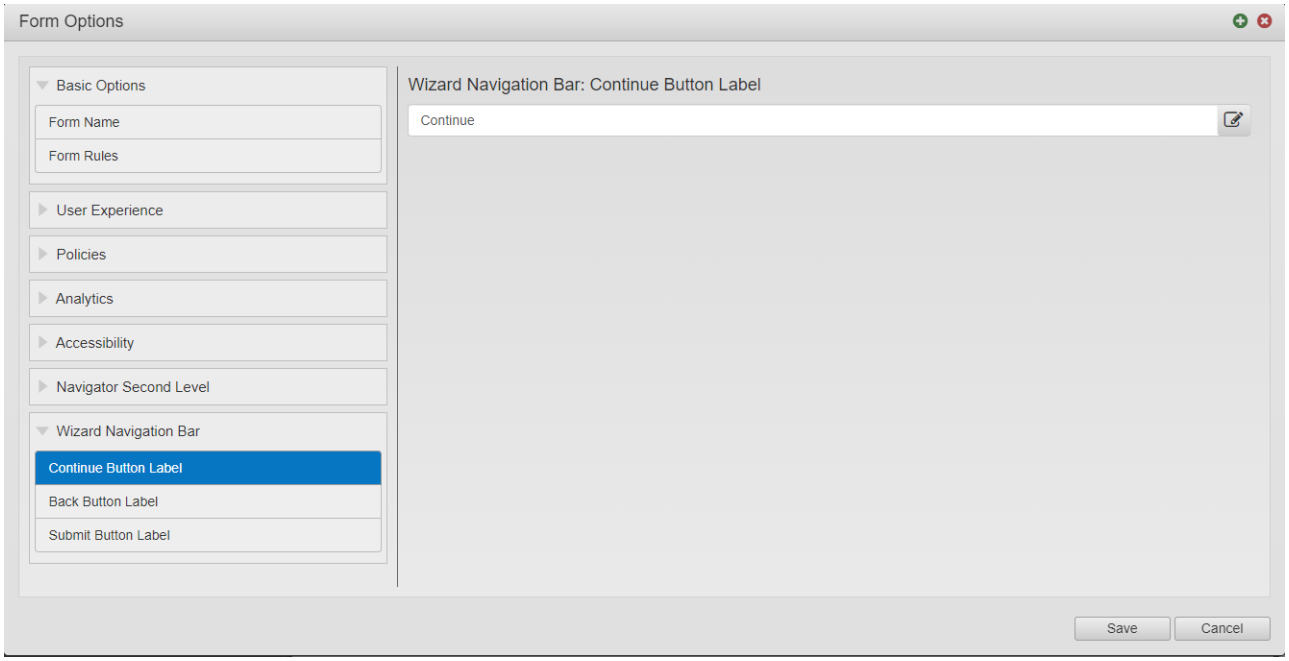
Wizard Navigation Bar

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to configure a Navigation Wizard Bar component for a Maestro form using Form Options.

To configure Wizard Navigation Bar using the Form Options:

1. Open a Maestro form and select Form Options.
2. Select Wizard Navigation Bar.



3. Select Continue Button Label to edit the text displayed on the Continue button.
4. Select Back Button Label to edit the text displayed on the Back button.
5. Select Submit Button Label to edit the text displayed on the Submit button.
6. Click Save.

Next, learn about all Form Options.

Form Background Save

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro has the background save functionality that automatically saves a form to a Journey Manager server in the background, without a user having to explicitly click the Save button. The background save option must be enabled for the form to be saved each time the user moves from one page to another.

Background save allows a user to return to a form even if something goes wrong while they are entering data. For example, the user is completing a 10-page form and they get to page 8 and all of a sudden, their computer locks up and closes the browser. With background save enabled, this data can be recovered using the form reference code, so the user can return to page 8 of the form. Though the user may lose the data of the page they were currently on when the computer locked up, redoing one page of data is much better than having to redo all 7 of the previous pages.

If the user did not record their reference code, our support team can provide you with a form link.

You can configure the background save using:

- Form Options
- Content Container

You can also configure a background save intervals for forms.

Next, learn how to enable form background saves.

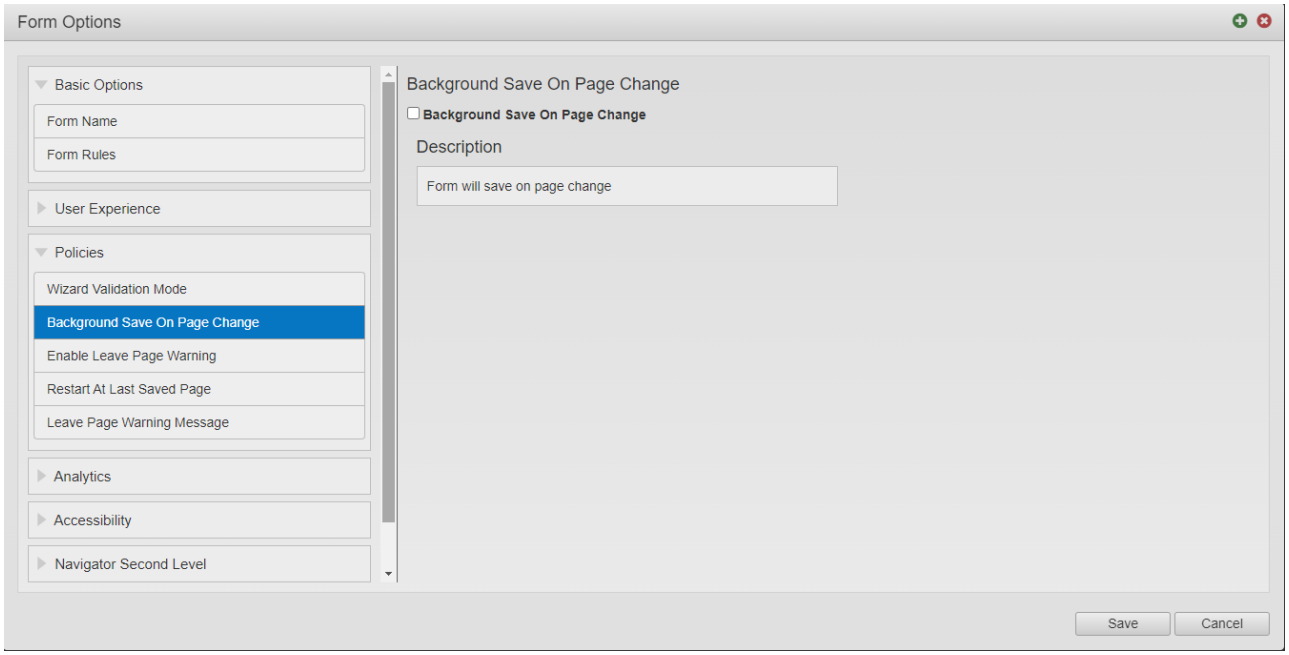
Enable Form Background Save from Form Options

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to enable the background save feature using Form Options.

To enable a form background save using Form Options:

1. Open a Maestro form and click Form Options. and select Policies.
2. Click Background Save on Page Change.



3. Select the Background Save on Page Change checkbox to enable the background save.
4. Check the message in the Description field.
5. Click Save.

NOTE

Click Build and Render Form to see how this rule is displayed when the form is built and rendered. The form is now saved each time the user changes to a different page. This feature is exposed through Form Options in the standard Maguire template, although your Template Designer may or may not have exposed this capability to your forms.

Next, learn how to enable form background save from content container.

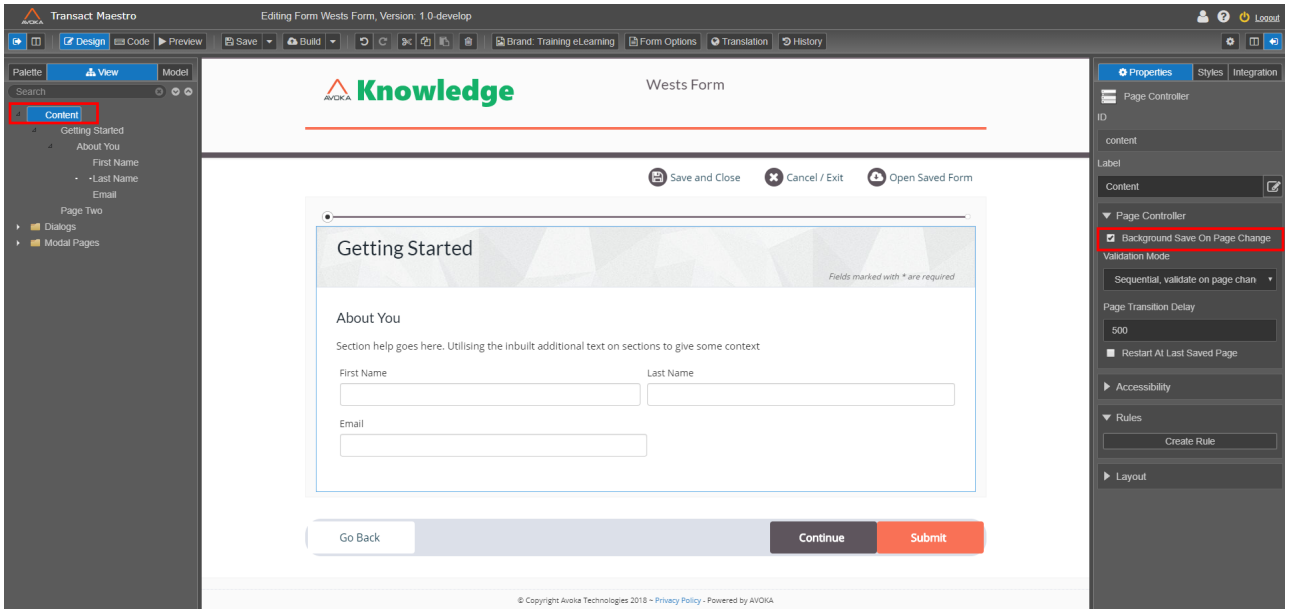
Enable Form Background Save from Content Container

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A Maestro form can have the background save feature enabled from the Content Container.

To enable a form background save from the Content Container:

1. Open a Maestro form.
2. Select the Content container directory where all form pages are located.



3. Click the Properties tab.
4. Click Options.
5. Select the Background Save on Page Change checkbox.
6. Click Save.
7. Click Build and Render Form to see how this rule is displayed when the form is built and rendered. The form is now saved each time the user changes to a different page. This feature is exposed through Form Options in the standard Maguire template, although your Template Designer may or may not have exposed this capability to your forms.

You can also enable the form background save feature from the form options window.

Next, learn about ?

Configure Form Background Save Interval

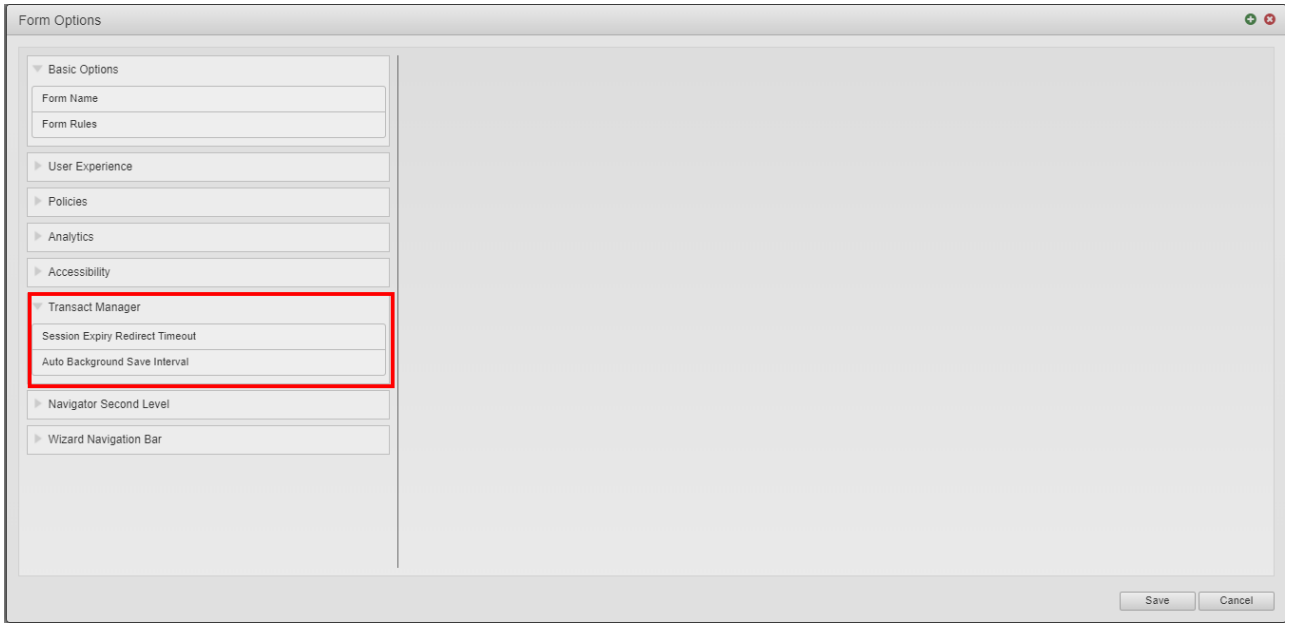
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro allows you to configure time intervals, in seconds, when a form is automatically saved in the background. Configuring background save intervals means that the form is saved at these defined intervals, for example every 40 seconds. This can be extremely useful when you are building large Maestro forms.

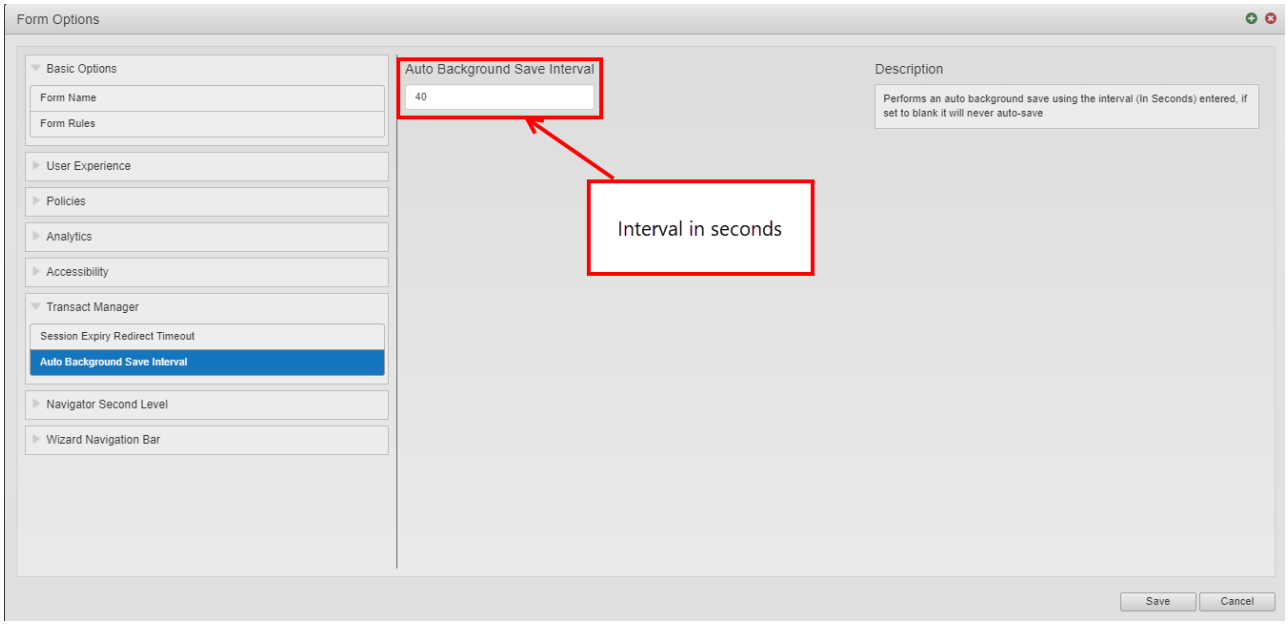
Once background save has been enabled for a Maestro form, a save interval can be configured.

To configure form background save intervals using Form Options:

1. Open a Maestro form and click Form Options and select
2. Select Transact Manager



3. Click Auto Background Save Interval.



4. Enter an interval duration in seconds, for example 40. Alternatively, use the arrow buttons in the textbox.
5. Click Save.
6. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Next, learn about ?

Form Save on Page Change vs Timed Save

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Fluent Function model uses triggers to initiate functions or services during form processing. These functions or services can be bound to these triggers for each form version. One of these triggers, Form Update, is ambiguous because it can be initiated by either:

- A page change within the form.
- A timed background save.

In most cases, this ambiguity is not a problem. However, because the former is triggered by the user, with validation taking place on the page data, and the latter is passive, with no validation, some applications need to distinguish between these events. For example, pushing the XML form data to a 3rd-party application, for example, Salesforce, might work fine at a page change, but might contain invalid data for a timed save. To address this problem you can use the solution described below.

Make a Place to Store the New Metadata

In the Form XML Data, add an element `/AvokaSmartForm/Meta/AutoSaveType`.

If you are using Value Objects to map the Form Data to a Java object, add a String variable `autoSaveType` to the Meta value object class.

Update the Metadata Before Calling the Service

In Maestro, in Form Options / Policies / Background Save On Page Change, disable the Background Save on Page Change option .

In Maestro, replace the functionality disabled in step 1 with the following: on each page, add a "Page change" rule that executes code similar to the following:

```
/* Let the save function know that this background save is due to a
   page change, to distinguish it from one triggered by a timed background save. */
data.autoSaveType = "PageChange";

var promise = Form.backgroundSave();

/* Settle the promise before resetting the state. */
promise.then(
    function(result) {
        console.debug('Form Background Save Result: ' + result);
    }).catch(function (error) {
        console.log('Form Background Save Error: ' + error);
    }).finally(function () {
        /* Reset the state. */
        data.autoSaveType = "";
    })
```

Update the Service to Use the New Metadata

The Groovy code below is contrived and for illustration purposes only. It assumes that the XML Form Data element `/AvokaSmartForm/Meta` has been unmarshalled into the `appMeta` value object.

```
boolean isFormUpdateForPageChange(FuncParam param, Meta appMeta) {
    logger.debug("Trigger: " + param.trigger)
    if (FuncParam.TRIGGER_FORM_UPDATE.equals(param.trigger)) {
        logger.debug("meta.AutoSaveType: " + appMeta.autoSaveType)
        return "PageChange".equals(appMeta.autoSaveType)
    }
    return false
}
```

Next, learn about the content container.

Form Pages

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Forms, built in Journey Maestro, can contain many components and therefore can become rather long. Navigating long forms can become daunting experience that can cause issues for end-users. To address these issues, Journey Maestro allows you to use page and section components in forms to improve the form user journey by grouping fields into logical areas. You can add a page to a form as frequently as you add new areas to it to provide users with concise data collection tasks.

If you use a template with a navigator, most templates do, then adding a page will automatically add a new item in the navigator menu.

Add a Page to a Form

When creating or editing a form in Maestro, you may want to extend the form functionality by adding multiple pages.

To add a page:

1. Open a form in the Design mode.
2. Select the Page component from the Palette and drag and add it to the form.
3. Add new components to the page.

NOTE

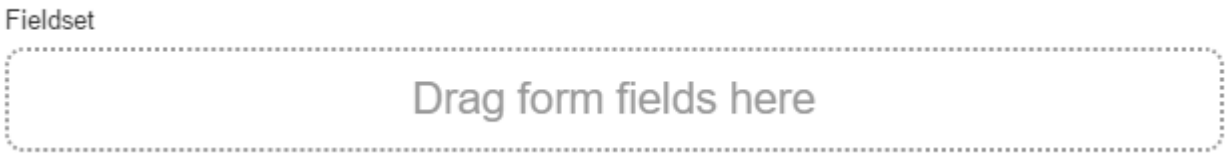
Alternatively, you can right-click the Contents item and select Add Page from the menu before dragging the page onto the form.

Form Sections and Fieldsets

If you want to further break up form content in a page, you can use section components. A section includes a heading and optional explanatory text as shown below.



You can break up your fields even further by grouping them into fieldsets. This is the recommended approach of grouping several related fields together.



As well as using sections and fieldsets, you can also configure a form to use two-level navigation.

> NOTE
>
> For pages to operate correctly, they must be inserted under the Content container in the hierarchy. Dropping the page component onto the form design may result in incorrect placement.

Form Two-Level Navigation

To improve the user experience of large forms, you can add two-level navigation to a form, which is a form sub-menu.

To create two-level navigation:

1. Open a Maestro form.
2. Select the Page Group component from the Palette. Make ensure that all pages are located within the Page Group, otherwise, it may be confusing for end-users. Page Groups will appear on the first level navigator, and pages within the Page Groups will appear on the second level navigator.
3. Add it to the Content.
4. Drag a new or existing Page to the Page Group.



NOTE

You can also use pages and sections to improve the user journey on long forms.

Form Versions Overview

MaestroThe UI design product. | Form Builder Platform Developer | 17.10 This feature was updated in 17.10.

Journey Maestro allows you to save a unique copy of a form by using form versions. You may want to create a new form version each time the TM Form Version is built, or when you are making significant changes to a form.

Creating a new form version at these points may save you the hassle of fixing your form should something go wrong. For example, say you're adding a feature or changing the behavior of existing form items but while making these changes something goes wrong. Form versions allow you to easily return to a previous working version of the form.

Form versions can be deleted, exported and compared in Maestro.

since 17.10, the Temenos Journey Manager platform follows open source semantic versioning conventions. Following this convention, all new form version names are recommended to start at version 1.0-develop until they reach production. In versions of Maestro prior to 17.10, form versions are incremented by whole numbers (for example version 1, version 2, version 3, etc.).

When you create a new form, the initial form version depends on the Default Initial Version specified in the Project Details tab of a project. You can change the Default Initial Version at any time by switching to the Project Details tab and changing its value. However, this only impacts new forms and assets within this project; version numbers of existing forms are unaffected.

> NOTE
>
> Form versions are sorted and displayed in descending order based on major, minor and patch release numbers.

Semantic versioning provides the following benefits:

- Standardize versioning of Maestro design artifacts which in turn makes management of different versions easier.
- Align more closely with Manager's versioning scheme.
- Assist Form Builders who are developing forms concurrently.
- Lock down a form version (using tags).

When working on any form, it is recommended that you do all your work on version 1.0-develop (the initial form version) until the form is ready for production. When a form is ready for production, it is recommended that you create a tagged version of the form.

NOTE

When you create a SCM project, Maestro disables the form version feature, so you must create and later maintain form versions using a Git[1] version control.

NOTE

If you are using Maestro 17.10.x with a Journey Manager 5.x environment, and your form version number is longer than 10 digits (e.g. "1.0-develop"), you will get an error when you attempt to import the Manager 17.10.x form to the Manager 5.x environment. In this case, it is recommended that you shorten the default suffix when creating new form versions in Maestro 17.10.x.

Next, learn how Manager handles form versions.

[1]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.

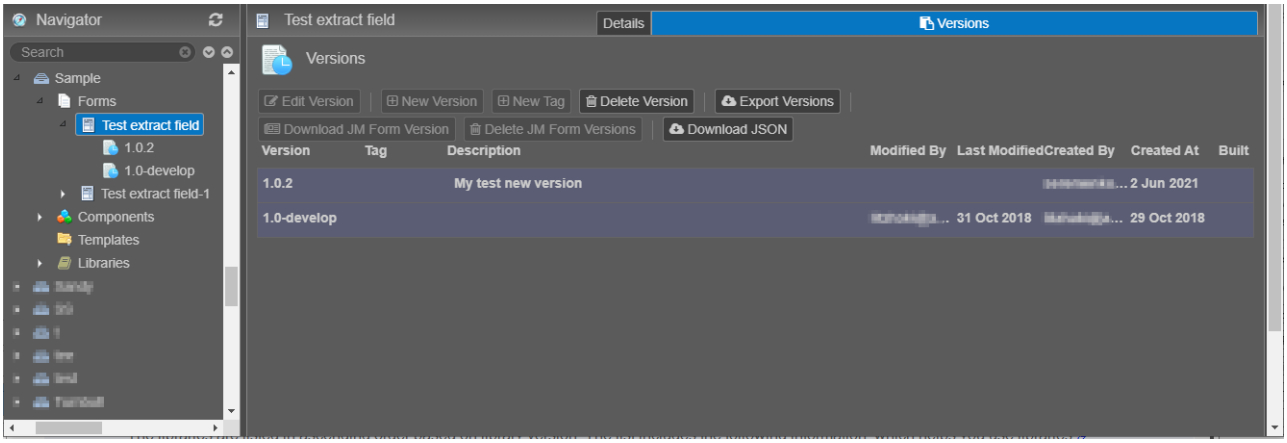
View Form Versions

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 21.05 This feature was updated in 21.05.

Journey Maestro allows you to view and manage form versions.

To view the list of form versions:

1. Open your project in the Navigator pane and click Forms to view the list of forms created in your project.
2. Select a form in the Forms folder to view available versions of this form.



The forms are listed in ascending order by form versions. The list includes the following information, which helps you use forms correctly:

- Version - a form version
- Tag - whether a form version is tagged or not
- Description a form version description
- Modified by - who modified a form version
- Last Modified - when a form version was modified
- Created By - who created a form version
- Created At - when a form version was created
- Built - this flag is set when there is a corresponding built form where the form developer has edited a form version and clicked Build in the editor and there is a Manager Form Version. In this case, if you use the Delete JM Form Versions functionality, described below, it will delete the built form and deleting the form design version will also delete the built version as well. If the Built checkbox is selected, the

Download JM Form Version button is available, so you can use it. Otherwise, you need to open the form version in Maestro and build the TM Form Version of the form.

3. Click Edit Version to open a selected form version in the Maestro editor for editing.
4. Click New Version to create a new version of a selected form.
5. Click New Tag to tag a selected form version.
6. Click Delete Version to delete the selected form version.

   NOTE

   Use `Shift` or `Ctrl` key to select multiple form versions. | 21.05 This feature was introduced in 21.05.

7. Click Export Versions to export the selected form versions.

   NOTE

   Use `Shift` or `Ctrl` key to select multiple form versions. | 21.05 This feature was introduced in 21.05.

8. Click Download JM Form Version to download a form version deployed in Manager. This button is only available for selection once the form version has been built at least once. | 21.05 This feature was introduced in 21.05.
9. Click Delete JM Form Versions to delete a built form version from a Manager instance where it is deployed but leave the form design in Maestro.

   NOTE

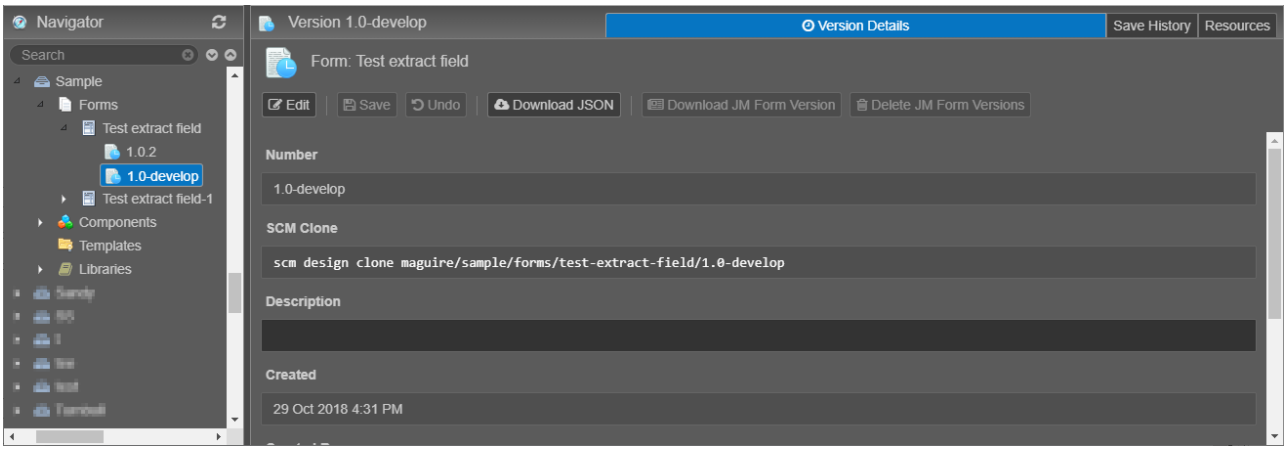
   Use `Shift` or `Ctrl` key to select multiple form versions. | 21.05 This feature was introduced in 21.05.

   You can use this functionality to solve the following project issue. Each form you build needs an organization wide unique name in Manager. Now, if you want to use the same form name in a new project, you must delete the previously published forms that use that name, for example, My Application Form, and start using the other form to publish to the same name. If you delete a project, form design or form design version, it will also delete the published form, otherwise the published form is "orphaned". If you have access to the Manager dashboard, you can do it manually from there. However, this features allows you to do it straight from the Maestro dashboard.

10. Click Render JM Form Version to open a form version, that has previously been built and deployed in the host Manager, in a new tab of your browser. | 21.05 This feature was introduced in 21.05.
11. Click Download JSON to download JSON files of the selected form versions.

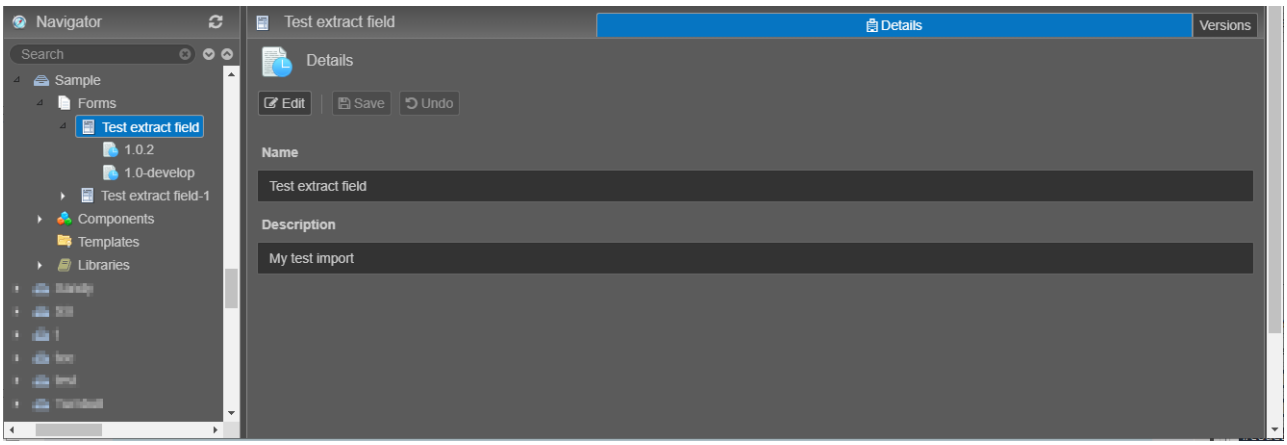
> NOTE
>
> Use `Shift` or `Ctrl` key to select multiple form versions. | 21.05 This feature was introduced in 21.05.

12. Double-click a selected form version to view the form version's details where you can update the **Description** field. Click Save to update the changes.



> NOTE
>
> You can't update the name and version number of the form.

13. Select the Details tab to see the form details where you can change the form's name and description in the corresponding Name and **Description** fields.



Click Save to persist the changes after you are done.

Next, learn how to export form versions.

Form Version Number

MaestroThe UI design product. | Form Builder Platform Developer | 17.10 This feature was updated in 17.10.

A form version can contain numbers and periods (.) and can only be up to three levels. Form version numbers are based on major, minor and patch releases. For example, if a form version number is 1.1.3, it means that 1 is major, 1 is minor, and 3 is patch number. If a version number includes less than three levels, Maestro will add the missing levels as "0". For example, if "1" is entered, the new form version will become "1.0.0" when you navigate away from the field.

since 17.10, the Temenos Journey Manager platform follows open source semantic versioning conventions. Following this convention, all new form version names are recommended to start at version 1.0-develop until they reach production. In versions of Maestro prior to 17.10, form versions are incremented by whole numbers (for example version 1, version 2, version 3, etc.).

VIDEO

For more information, check the Introduction to Semantic Versioning video.

Next, learn how to compare form versions.

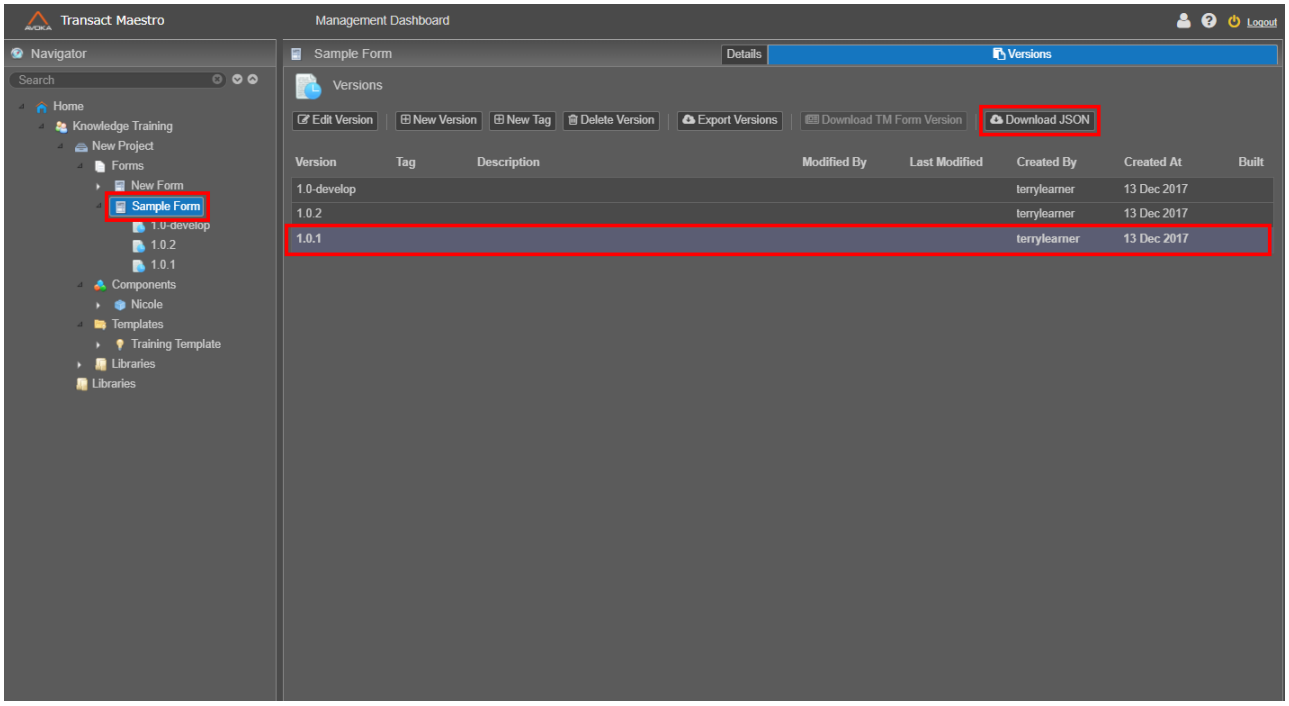
Compare Form Versions

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 18.05 This feature was introduced in 18.05.

If you have multiple versions of a form, you can download the JSON[1] file of each version, and use an external tool to compare the differences between the downloaded form versions. There are two methods of downloading the JSON file of a form version.

Method 1

1. Select the form from the Navigator pane.
2. Select a form version.
3. Click Download JSON.



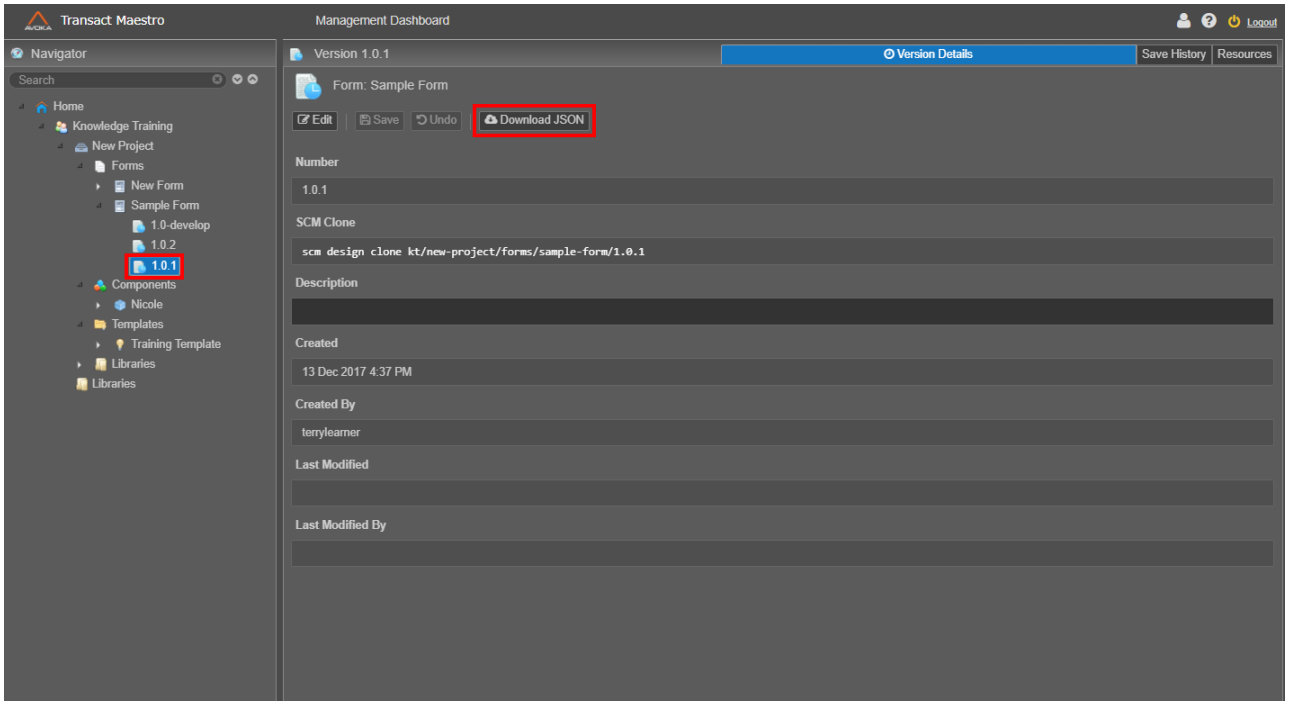
4. Select a different version from the list and download its JSON file.
5. Using an external tool, you can then compare the differences between the two downloaded JSON files.

---

[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.

Method 2

1. Select a form version from the Navigator pane
2. Click Download JSON.



3. Select a different version from the list and download its JSON file.
4. Using an external tool, you can then compare the differences between the two downloaded JSON files.

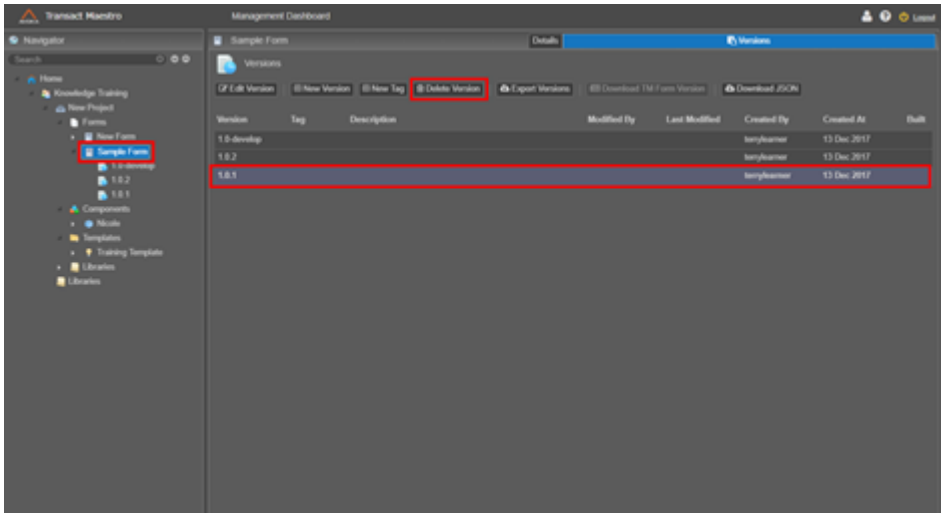
Delete Maestro Form Version

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 17.10 This feature was updated in 17.10.

You can delete Maestro form versions if necessary.

To delete a form version:

1. Select the form from the Navigator pane.
2. Select the form version to delete.
3. Click Delete Version.
4. Click OK to confirm the deletion of the selected form version.



Next, learn how to delete form versions in Manager.

Export Form Versions

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 17.10 This feature was updated in 17.10.

Maestro allows you to export form versions. You can use this functionality to:

- Migrate a single form version to a Manager production environment, as either a new or updated form version.
- Back up form versions.

To export a current form version:

1. Select a form.
2. Click Export Current Form Version to export the current form version.

To export any other (not current) form version:

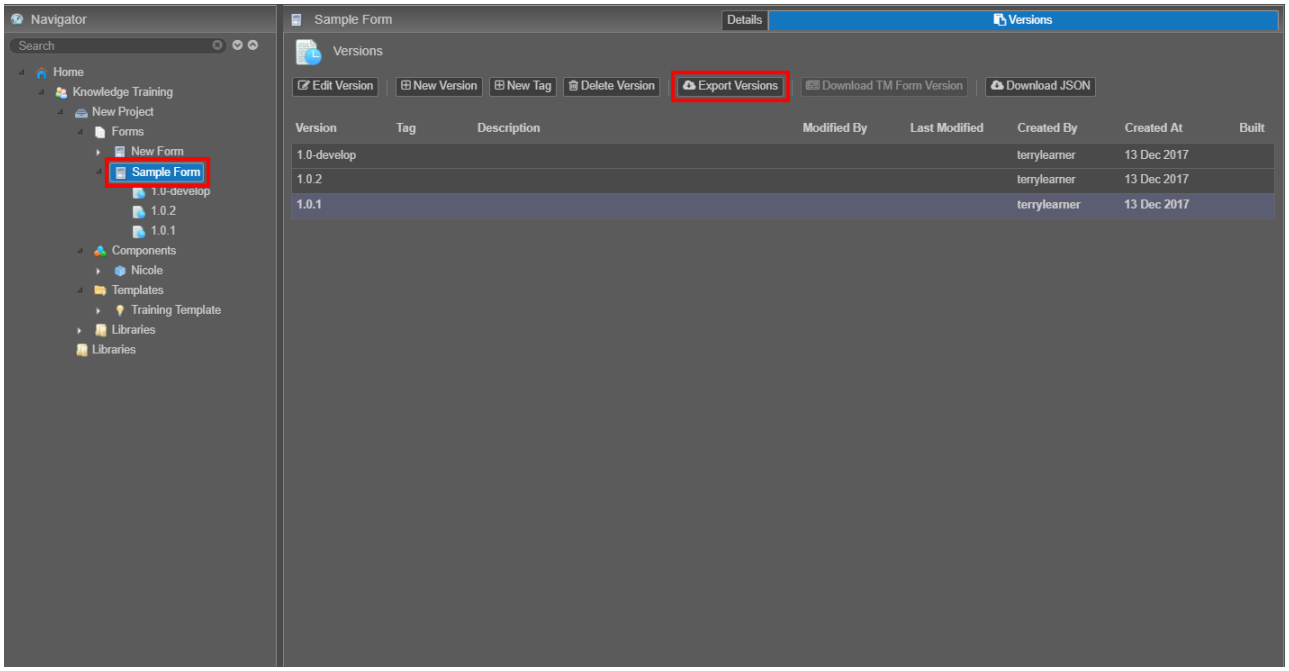
1. Select the form version.
2. Click Export .
3. Click OK to confirm your decision.

Maestro also allows you to export all versions, or selected versions of a form.

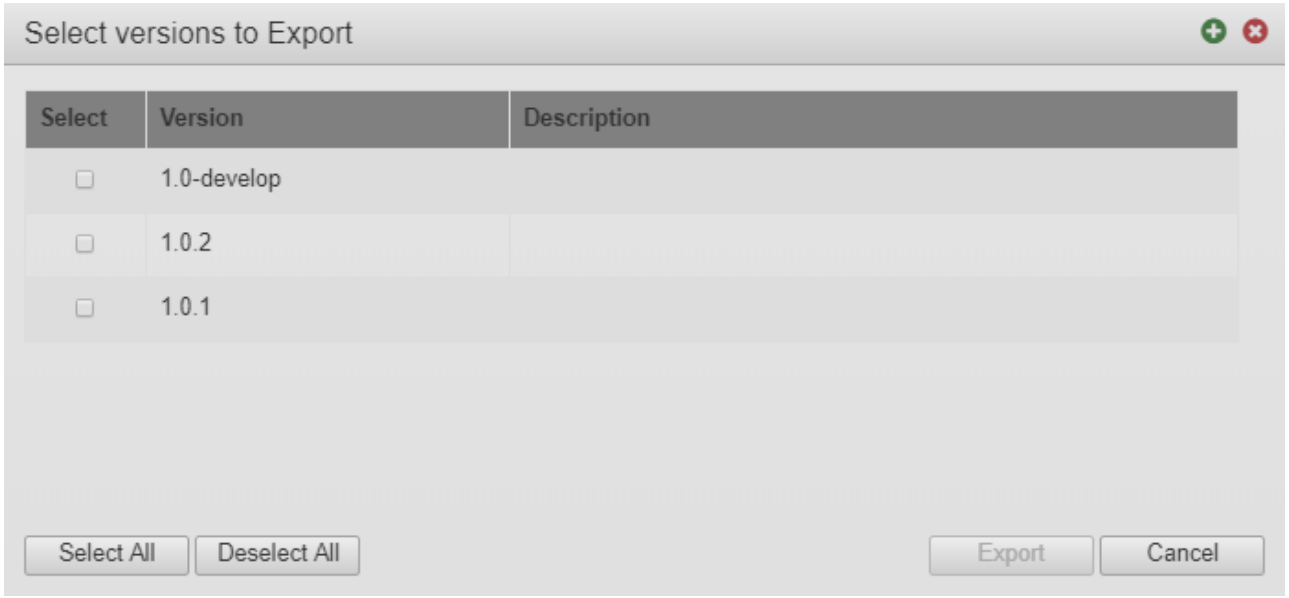
since 17.10, the Temenos Journey Manager platform follows open source semantic versioning conventions. Following this convention, all new form version names are recommended to start at version 1.0-develop until they reach production. In versions of Maestro prior to 17.10, form versions are incremented by whole numbers (for example version 1, version 2, version 3, etc.).

To export a single or multiple form versions of the same form:

1. Select the form from the Navigator pane.
2. Click Export Versions.



3. Select the versions you want to export. You can select one or more versions to include in the export. You can also use the Select All option to select all versions of the form.



4. Click Export. The selected form versions will be packaged into a single zip file and the zip file will be downloaded to your computer.

Next, learn how to import a form.

Nominate the Current Form Version

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 17.10 This feature was updated in 17.10.

The current form version is the form version that is rendered when the form is requested by the user. When there is only one form version, that version is considered the current form version.

Nominating the current form version can be done through the Manager's Form Dashboard tab.

To nominate the current Form Version of your form, select your form and then:

1. Select the Form Versions tab.

   The current Form Version is the one that is listed with a tick in Current Version column of the table.

   To nominate a different form version to be the current form version:

2. Click **Make Current link** next to the newly nominated current form version.

   This may also be done in the same way from the Form Dashboard tab in the Form Versions section.

NOTE

When a form has been configured for A/B Testing, it is now in the control of the A/B Testing regime instead of the current version. Only on A/B Testing completion, will the current version be enforced.

Open an Earlier Version of a Maestro Form

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 17.10 This feature was updated in 17.10.

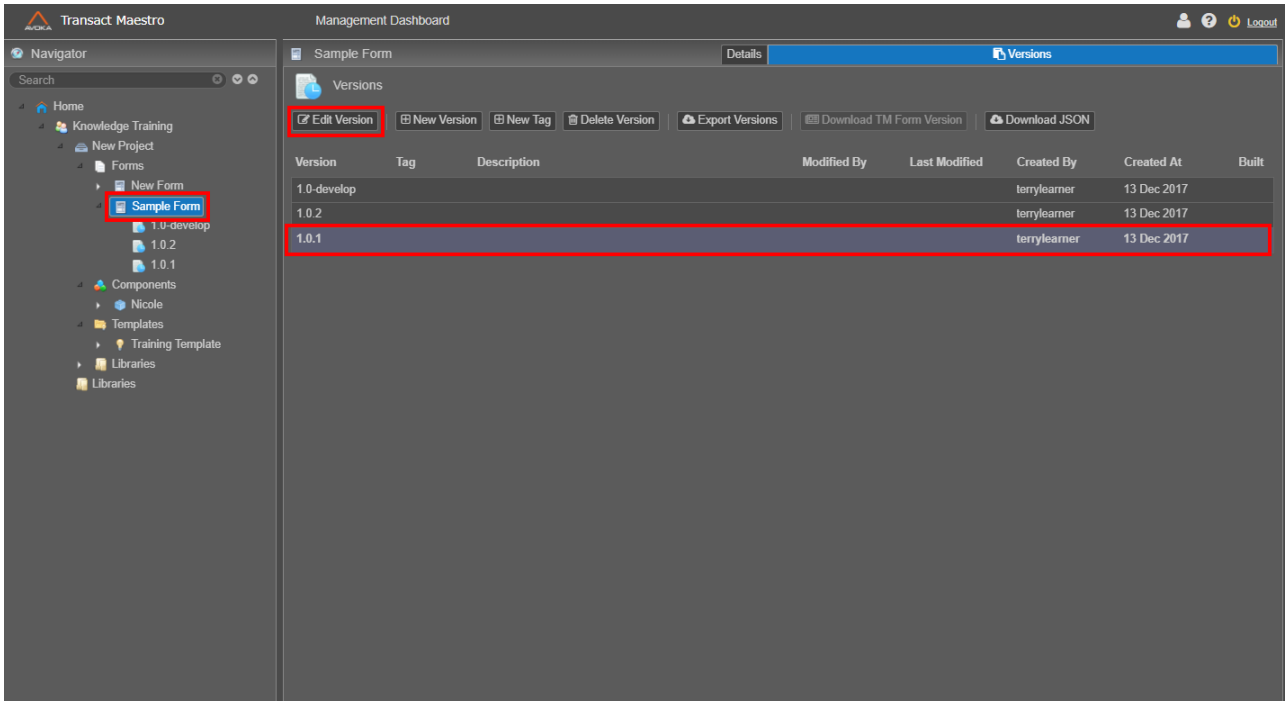
When you open a form version from the Maestro dashboard, the latest version of the form will be opened. Though this is normally what you want to do, sometimes you may want to open a previous version of the form. After a previous version of the form has been selected and edited, you can choose to set it as the current version of the form.

since 17.10, the Temenos Journey Manager platform follows open source semantic versioning conventions. Following this convention, all new form version names are recommended to start at version 1.0-develop until they reach production. In versions of Maestro prior to 17.10, form versions are incremented by whole numbers (for example version 1, version 2, version 3, etc.).

There are two methods available to open an earlier version of a form in Maestro.

Method 1

1. Select a form from the Navigator pane.
2. Select a form version.
3. Click Edit Version.



Method 2

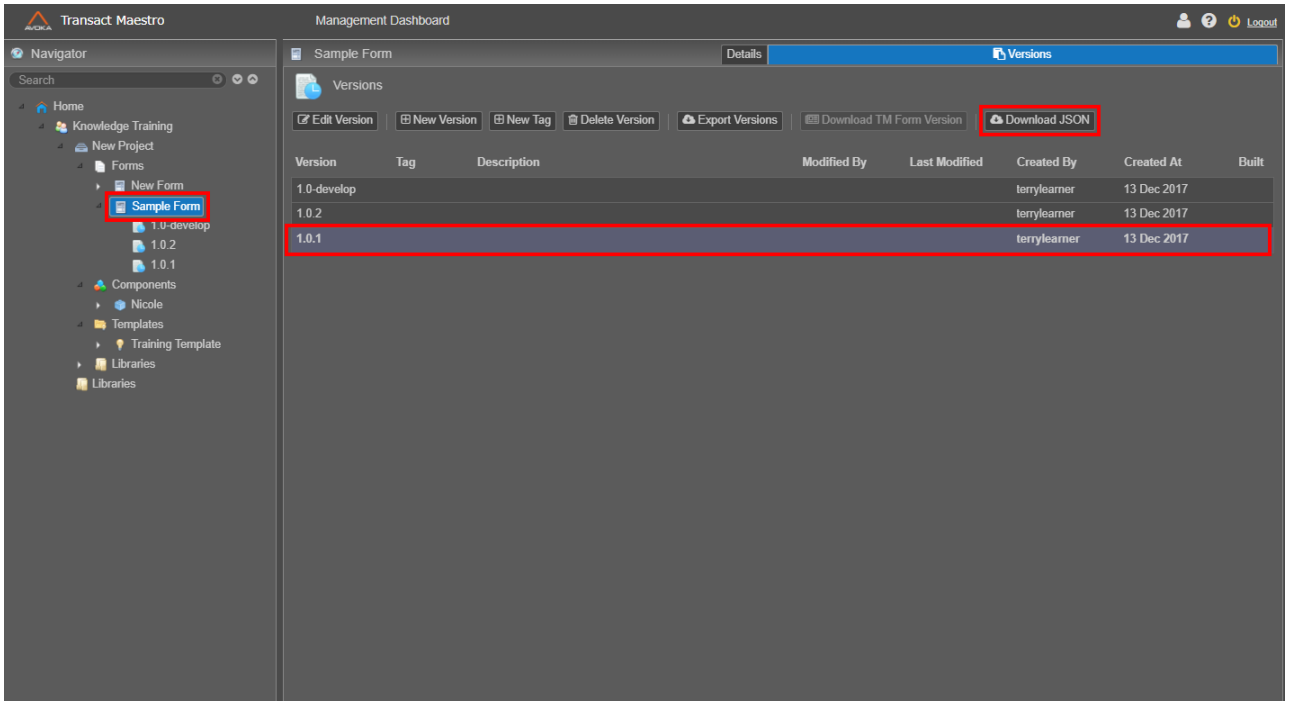
Expand the form in the Navigator pane, select the form version that you want to open, and click Edit.

Compare Form Versions

If you have multiple versions of a form, you can download the JSON[1] file of each version, and use an external tool to compare the differences between the downloaded form versions. There are two methods of downloading the JSON file of a form version.

Method 1

1. Select the form from the Navigator pane.
2. Select a form version.
3. Click Download JSON.



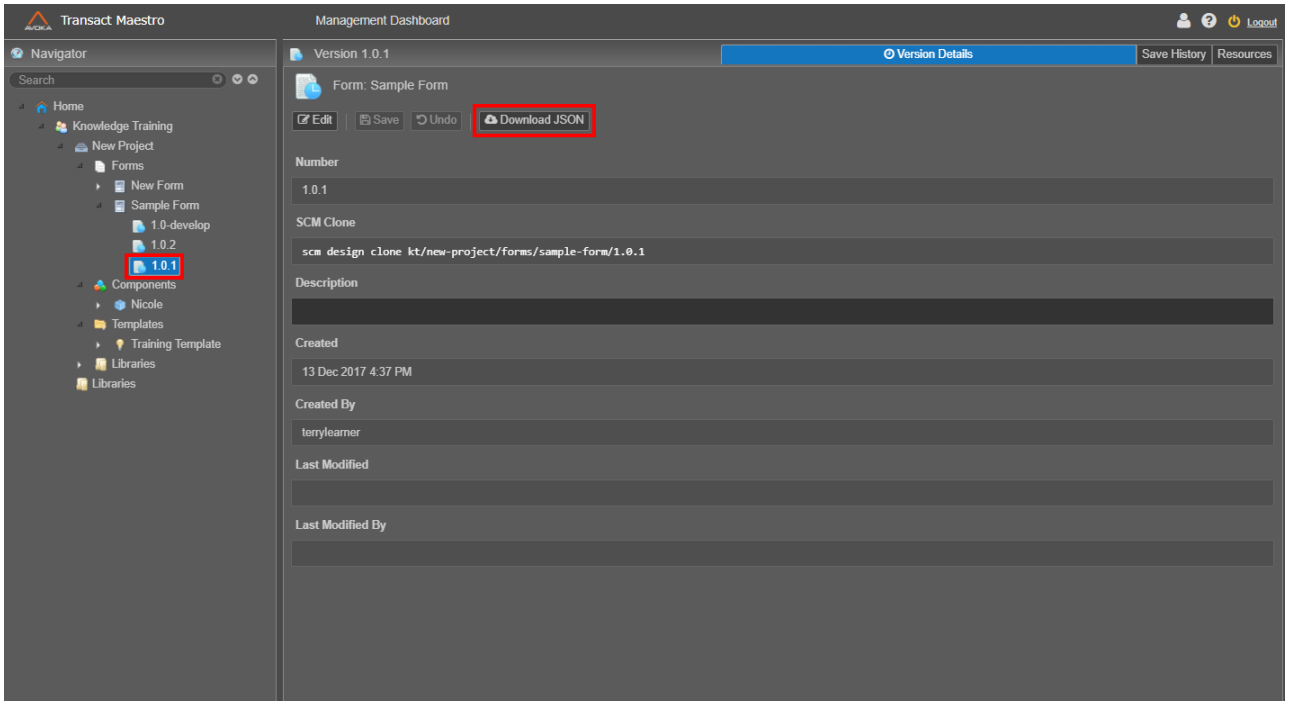
4. Select a different version from the list and download its JSON file.
5. Using an external tool, you can then compare the differences between the two downloaded JSON files.

---

[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.

Method 2

1. Select a form version from the Navigator pane
2. Click Download JSON.



3. Select a different version from the list and download its JSON file.
4. Using an external tool, you can then compare the differences between the two downloaded JSON files.

Tagged (locked version) Form Version Overview

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 17.10 This feature was introduced in 17.10.

A tagged version of a form is a read-only form version that prevents any changes from being saved to the form. A tagged version is an effective way of keeping a clean, final record of a form. It is recommended that you create a tagged version each time a form is deployed to an environment. For instance, while you are developing a form, you should be using the "1.0-develop" form version; once this version of the form is ready for production and deployment to an environment, it is time to create a tagged version of the form. Once a new tagged version of a form has been created, it is possible to create new form versions based on a tagged version.

since 17.10, the Temenos Journey Manager platform follows open source semantic versioning conventions. Following this convention, all new form version names are recommended to start at version 1.0-develop until they reach production. In versions of Maestro prior to 17.10, form versions are incremented by whole numbers (for example version 1, version 2, version 3, etc.).

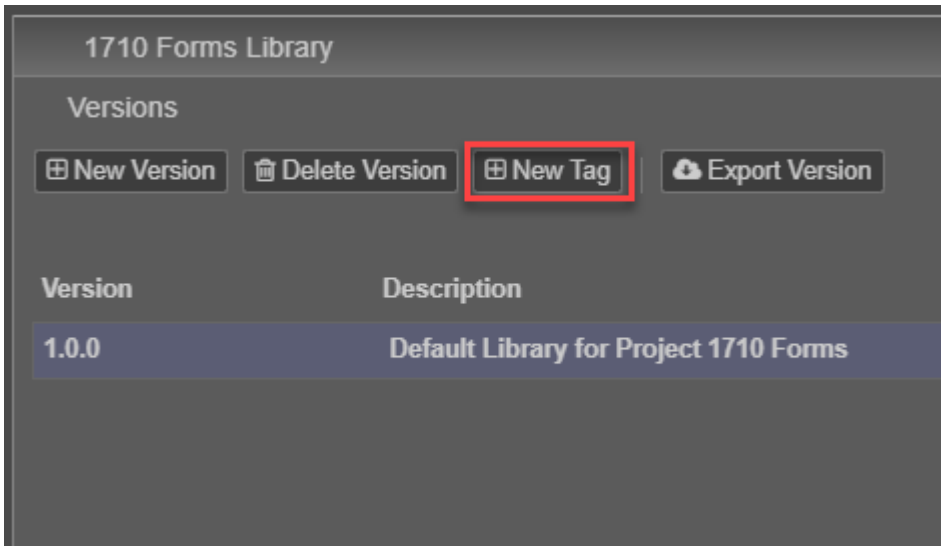
Create A Tagged Version of a Form

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 17.10 This feature was introduced in 17.10.

Maestro allows you to tag a form version.

To create a tagged version of a form:

1. Select the form from the Navigator on the Dashboard.
2. Click New Tag.



3. Complete the options in the New Tag window.
4. Select which version of the form you want the tagged version to be based on from the Based On Version dropdown list.
5. Enter a version number for the tagged version in the New Version field.
6. Click Create.

From the Versions list, you can see a check mark in the column next to the tagged version. If you open the tagged version in the Maestro Editor, you can see the read-only text at the top of the form. In the read-only version, you notice that you are not able to add new components to the form, and there is no option to save any changes you make.

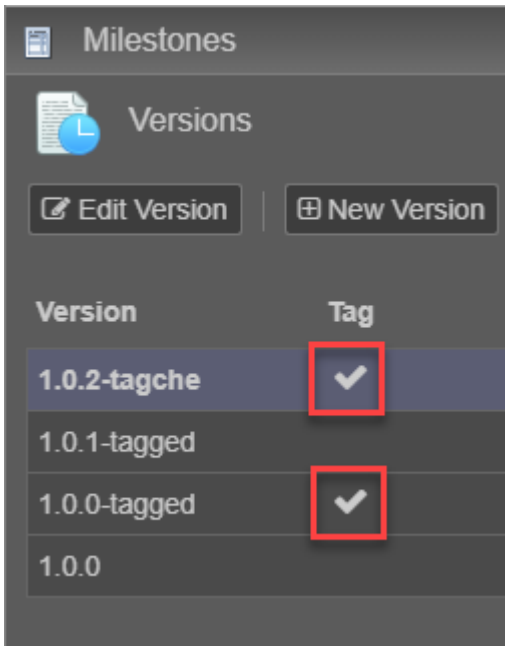
Once you have created a tagged version of a form, you may need to create a new version of the form based on the tagged version.

Create a New Form Version based on a Tagged Version

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 17.10 This feature was introduced in 17.10.

To create a new form version based on a tagged version:

1. Navigate to the Forms folder within the project.
2. Select a form to view the existing versions (you can also expand a form from the Navigator pane to see all form versions). If the form contains any tagged versions, each tagged version will be identified by a check mark.



3. Select a Tagged Version (in the above screenshot, the tagged version "1.0.2-tagche" has been selected).
4. Click New Version.
5. Complete the Add a New Version dialog.
6. Select a previous form version that you want the new version based on from the Based On Version dropdown list. By selecting the tagged version "1.0.2-tagche" before clicking New Version, this will be the form version that the new form version will be based on.



7. Enter the version number you want to use in the New Version field. You can also include a suffix that contains numbers and letters. When creating a new version, the third (the patch) part of the version number will be incremented by one to the previous version of the form. Since you are creating a new version of the form based on a tagged version, it may be appropriate to increment the first (major) part of the version number by one. Ffor example, version 1.0.2-tagche will become 2.0.0- develop.



8. Enter a description text to help organize the different versions of a form in the Description field.
9. Click Create.

The new form version will be created, and its details displayed in the Maestro Dashboard. You will also see the new version listed under the form in the Navigator panel. Even though the new form version is based on a tagged version,you can edit it because it wasn't created as a tagged version. You can edit the new form version in the usual ways; for example, select the new form version in the Navigator and click Edit to open it in the Maestro editor.

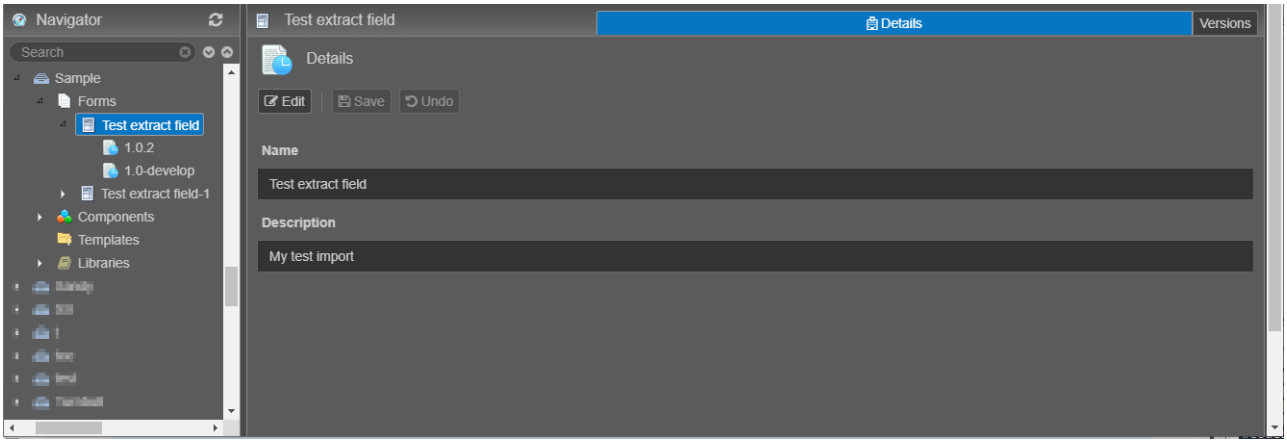
Change a Maestro Form Version Description

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro allows you to change a description of a form version, which you may need to do after you have created a new form version. You can also change a form's description at any time after the form is created.

To change a form version's description:

1. Open Maestro and locate a form version to see its current description.
2. Click the version directly in the Navigator pane. Or, select the form in the Navigator pane and then double-click the form version in the Version pane.



3. Update a form version's description in the Description field.
4. Click Save to update the changes.

Next, learn how to open a Maestro form.

How Applicants Experience a Maestro Form

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro forms are highly configurable, which means how applicants experience a form will depend on the organization's requirements. Even so, certain patterns and best practices do find their way into many Maestro forms. This topic provides a high-level overview of how applicants typically experience a Maestro form.

1. Load a form

When an applicant loads a form, what they see is determined by the Template Designer and Form Builder. They might be immediately presented with form fields to complete, or if the applicant needs certain things to complete the form (such as identification or proof of income), they might be presented with a landing page that describes these requirements.

Forms are published as single-page AngularJS applications, which means the entire forms are loaded up-front. This impacts the initial load time, but the user won't have to wait between page transitions once the form is loaded.

> NOTE
>
> To improve the initial load time of a form, enable Static Content Externalization.

2. Data is prefilled

Prefilling is the process of automatically inserting applicant data into a form, such as their name or email address, thereby reducing the amount of effort required by the customer to complete the form.

For example, if the applicant is authenticated, many of their details can be pulled in via a HTTP request. Even if an applicant is not authenticated though, details such as their latitude and longitude can be retrieved (with their permission). This is possible with the `Util.getGeoLocation()` method:

```
Util.getGeoLocation().then(function(result) {
       console.log("Latitude: " + result.latitude);
       console.log("Longitude: " + result.longitude);
});
```

Prefilling streamlines the user experience and can significantly reduce the abandonment rate of a form. To learn about the different ways to implement it, refer to Form Prefill Options.

3. Enter data into fields

The applicant will spend most of their time in this part of the process, entering data into fields and ensuring they meet whatever requirements the Form Builder has configured. Typically, applicants can expect to see:

- A range of different field types, such as text fields, date pickers, and dropdowns.
- Errors when invalid data is entered into a field (or when required data is missing).
- Fields spread across multiple pages and organized into logical sections.

When more advanced functionality is required, users may also interact with third-party services. Many of these services can be integrated with Maestro by using pre-built Exchange packages.

4. Save and resume

When a form is opened, a tracking code is created. This code allows applicants to complete forms across multiple sessions, which is useful for long forms or forms that have complex requirements. The typical implementation is for applicants to answer a security question when saving the form. This answer can then be paired with the tracking code to re-open the form at a later time.

To learn how to implement a save and resume feature, refer to Create a Save Challenge.

5. Submit the form

When an applicant submits a Maestro form, they are usually presented with a page that confirms their data has been submitted. This page is often implemented as a Modal Page component, rather than a regular Page component. This prevents the final page from automatically appearing in the navigation menu. If further steps are required by the user, or if some form of approval is required before the applicant can proceed to the next step, this page is an ideal place to explain to the applicant the next step of the process.

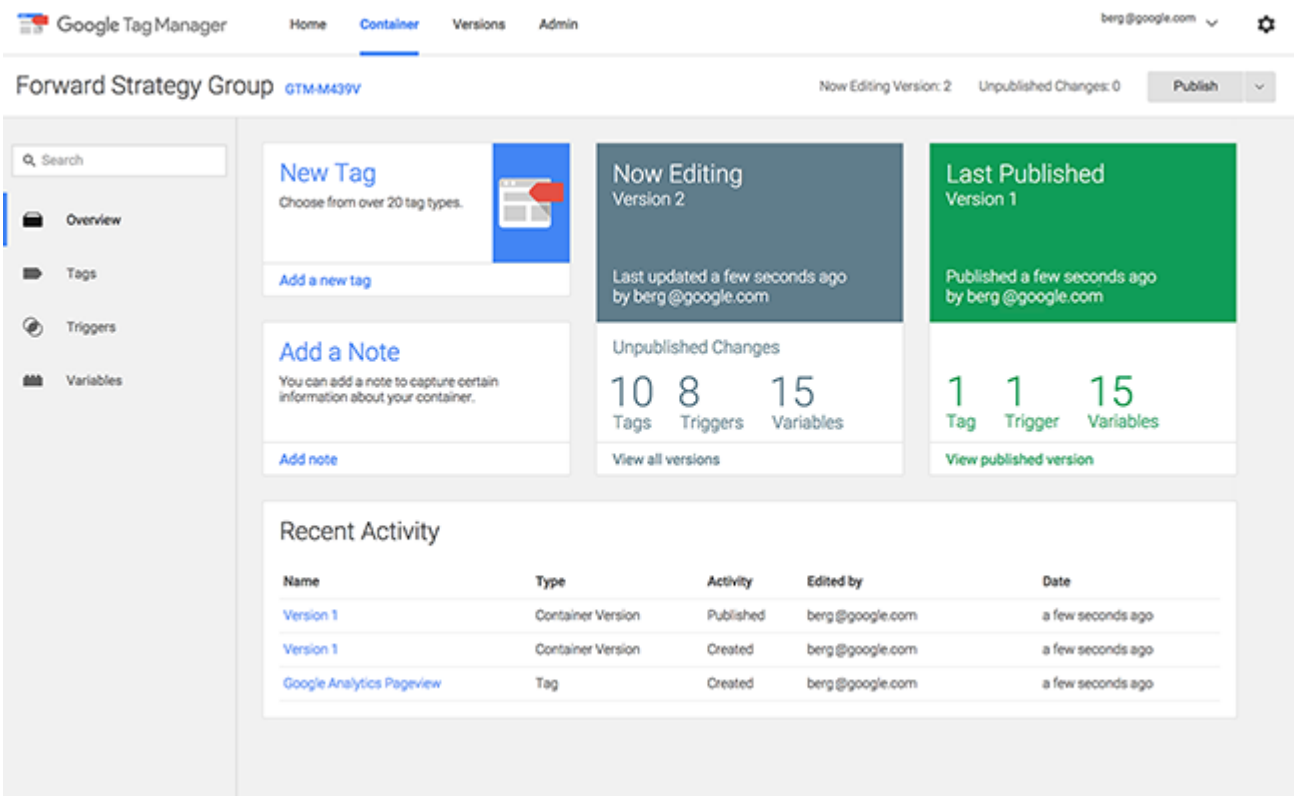
Upon submission, Maestro sends the form data to Manager. Once there, a wide range of actions can be performed using the data. For example, Manager can send a confirmation email to the form user about their submission, generate a PDF receipt, or trigger a collaboration job. Other functionality can be implemented via services or Groovy scripts. Some functionality is enabled by default, such as logging and purging of personally identifiable information (PII).

Add Google Tag Manager to a Maestro Form

MaestroThe UI design product. | Form Builder Template Designer | All versions This feature is related to all versions.

When you create an onboarding web application forms using Maestro, a Maestro form will have a single-page architecture, that is all the content exists in a single rendered HTML page, but the user navigation flow is controlled through hide and show logic of different sections, also known as logical form pages.

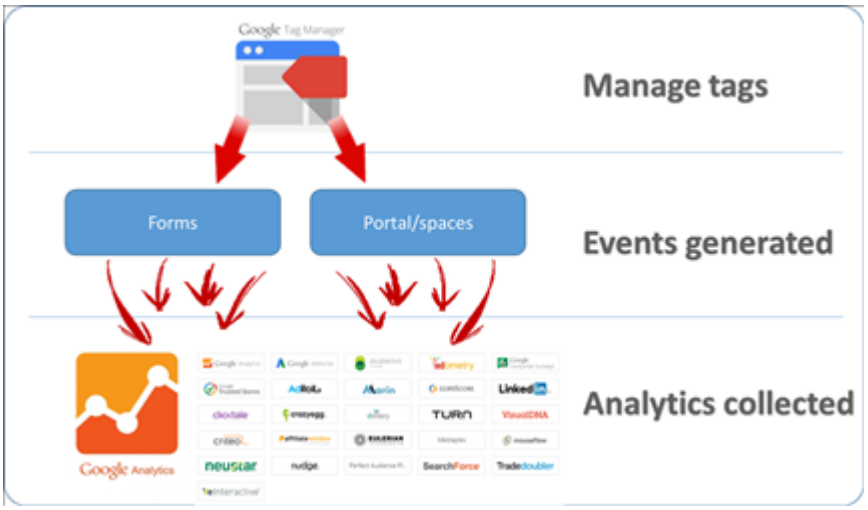
Google Tag Manager (GTM) lets you launch new tags easily so you can have good control over your digital marketing and analytics software.



Whilst providing a better user experience, any standard Google Tag Manager (GTM)[1] configurations that utilize page views to track web-site traffic will stop tracking the user movement once the form is opened meaning none of the in-form navigation will be tracked.

[1]Google Tag Manager (GTM) 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 web-site or mobile app.

To go around this constrain, you must correctly configure GTM in a Maestro form that enables virtual navigation events to be tracked for user navigation, which is described below. Additionally, you need to understand what events will be created in GTM as a user navigates through a live Maestro form, and how they can be captured in tags to generate analytics data.



NOTE

There is a pre-built GTM container for the Google Analytics framework that can be used as an example, or even in a production environment, if you decide to go with Google Analytics. Alternatively, we recommend using our Journey Analytics product, which is fully integrated with Maestro forms.

To enable a Maestro form to send virtual navigation events, you need to:

1. Import the GTM widget library into Maestro. You do it once per each Organization.
2. Add the GTM widget to the relevant form in Maestro. You do it once per form.
3. Connect the form to GTM in Manager. You do it once per form per Manager instance.

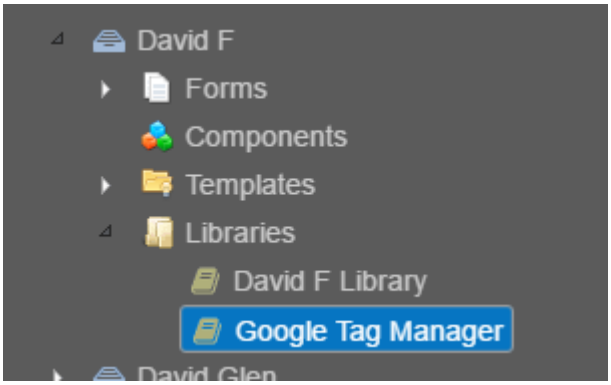
Let's look at each step in details.

Import the Widget Library into Maestro

To import the widget:

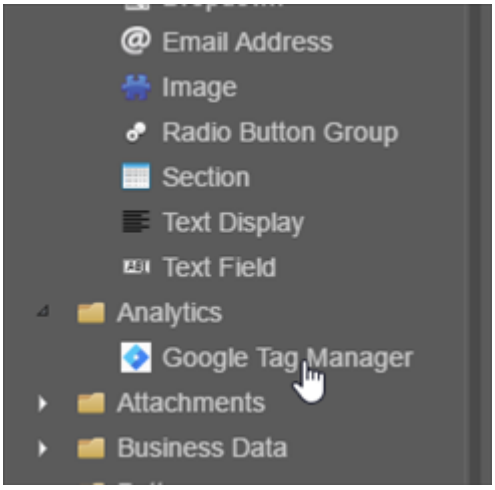
1. Log into Maestro.
2. Import the GTM `googleTagManager.zip` library into Maestro.

3. Check that Google Tag Manager entry appeared in the Libraries folder, as shown below.

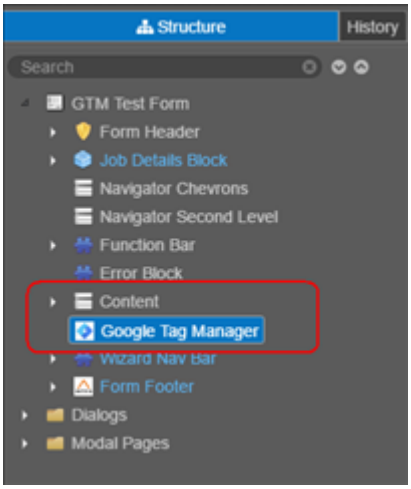


Add the GTM Widget to a Form

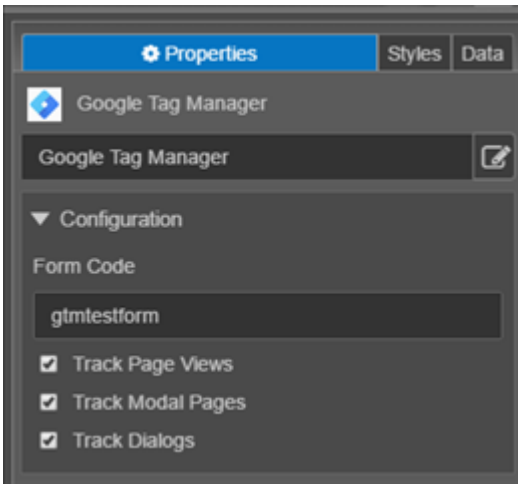
1. Create a new form or open an already existing form.
2. In the palette on the left-side of the screen, locate the Analytics folder, and the Google Tag Manager widget inside.



3. Drag this Google Tag Manager widget to the top level of the form in the structure panel. It is recommended to place it immediately after the Content element in the structure panel.



4. With the Google Tag Manager element selected in the Structure panel, expand the Configuration menu in the Properties panel.



5. Provide an unique form identifier in the Form Code field, which is passed to GTM to identify this form as separate to any other forms using the same GTM container. Alternatively, you may leave this field blank, so the default `form-code` assigned by Manager will be used.
6. Select the Track Page Views checkbox so section-level navigation will be sent to GTM as page change events. This is selected by default.
7. Select the Track Modal Pages checkbox so any modal pages, such as Submission confirmation page, Saved confirmation page or any custom modal windows, will be sent to GTM as page change events. This is selected by default.
8. Select the Track Dialogs checkbox so the opening of a dialog, such as Save challenge dialog, Are you sure dialogs, or any other custom dialogs, will be sent to GTM as a page change event. This is selected by default.
9. Click Save and then build and publish the form.

Configure Manager to Enable Analytics

As mentioned before, a Maestro form is built using a single-page architecture, where all of the content added to the form in the Maestro tool is rendered in a single HTML page. However, the actual form content only forms part of the user-experience picture.

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

Manager maintains global versions of these pages that are used by all forms by default, so it becomes important to be able to identify which form the page has been opened for. For the most part this can be achieved by using the formCode URL request parameter that is required to open the form, however for some of the post-form pages, such as Submission Complete, Attachments, Save Confirmation, this parameter is not provided. In these cases GTM has been setup to look for a form code within the HTML of the page that's been rendered, by finding a hidden element with the id of `submission_formCode`. The pages do not automatically include this hidden element, so you need to add it manually.

To add the HTML element to a form:

1. Log into Manager.
2. Select System > Form Spaces.
3. Find the form space that contains the form you would like to add GTM capability to, and click on the name of the space.
4. Find the page to be updated, and click Edit.
5. Select the Pages tab.

   > NOTE
   >
   > You need the Portal Edit permission to be able to edit the page.

6. Place the following HTML code inside the page content. The location is not critical:

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

NOTE

Do not put this code inside any conditional logic that has the markup such as `#if ($session.emailed)`.

7. Click Save.
8. Repeat the same updates to the all pages that need to be tracked.

Connect the Form to GTM in Manager

You need to configure a form in Manager with the correct GTM container configuration, including a JavaScript.

To add the GTM container script to the form:

1. Select Forms -> Forms and locate a form you want to track.
2. Click Edit and select the Page Tracking tab to enable page tracking.
3. Select the Use Form Page Tracking checkbox.
4. Select the Inside HEAD in the Page Script Include Position dropdown list. Keep the Tracking Script Template blank.
5. Paste the GTM script in the Form Page Tracking Script text field to enable tracking of the form.

   NOTE

   You can find the GTM script from your Google Tag Manager account by selecting Admin > Install Google Tag Manager. For more information, see how to set up and install Tag Manager.

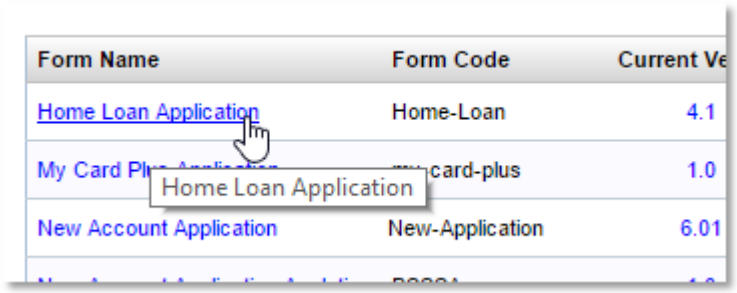
6. Click Save to update the changes.

Additional Forms Pages

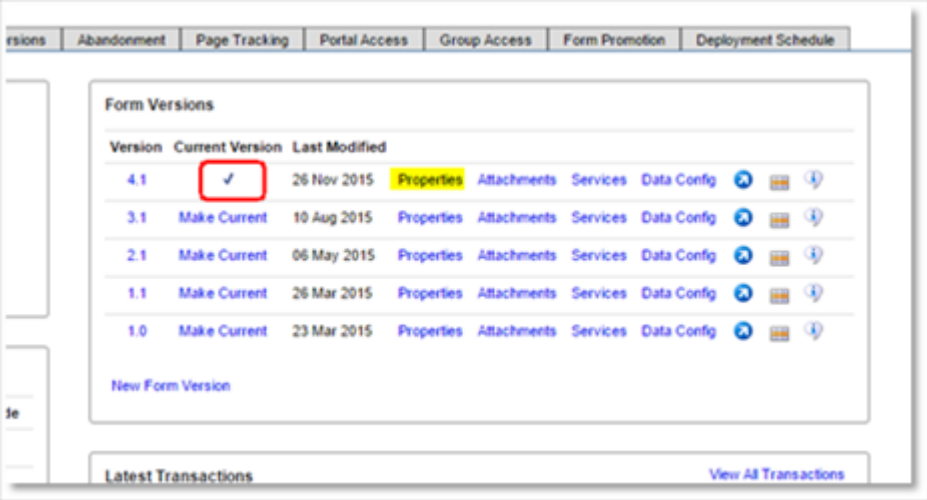
Manager has the ability for a form to maintain its own version of these pages - so that different content may be displayed to the user for a particular form. However, this setup is optional, and if not configured, the default pages referred above are used.

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

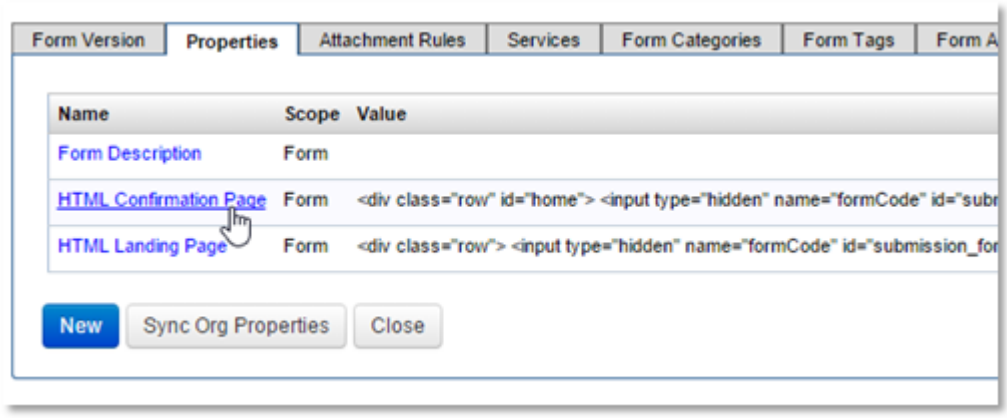
1. Select Forms -> Forms and Locate the relevant form and click on the name entry.



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



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



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

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

6. Click Save.

Content Security Policy (CSP) Settings

GTM, by its very nature, relies on the use of third party JavaScripts, so you must adjust the Content Security Policy (CSP)[1] at the organization level in Manager. This needs to be done for every organization that contains forms using GTM.

To configure CSP:

1. Open your organization security configuration.
2. Select the Override System Default checkbox.
3. Add the following policy values in the Organization CSP Header field:
   - `'unsafe-eval'`
   - `'unsafe-inline'`
   - `tagmanager.google.com`
   - `www.googletagmanager.com`
   - `www.google-analytics.com` - it is required if you use Google Analytics with GTM

   For example, CSP Header might look like this: `script-src 'self' 'unsafe-inline' 'unsafe-eval' www.googletagmanager.com tagmanager.google.com; object-src 'none'`.
4. Click Save.

This configuration will allow the form to connect to GTM. However, depending on the GTM implementation, there may also be additional CSP adjustments. For example, if the GTM container is

---

[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/

to send data to Google Analytics, then the host name of the GA server (www.google-analytics.com) will be also required. Similarly, if we use any other server-send data (like for GA), we will need to add host name of that particular server.

Configure a GTM account

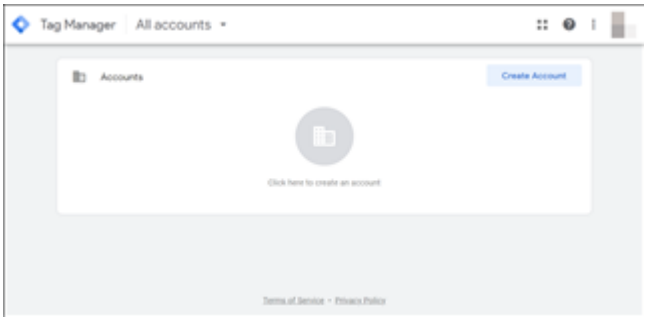
Once you have done all the steps above, the form will begin sending events to GTM when users open and navigate it. At this point, the GTM administrator needs to create their own container to use these events and send information to their chosen analytics engine. For more information, see which events are created, and what other relevant information is set in the data layer for these events.

Alternatively, Temenos has created a GTM container that is pre-configured to listen for these events, and act accordingly. By default, this container sends a page-view to Google Analytics, but could easily be adapted for other analytics engines.

You need a GTM account to use Google Analytics and Google account is mandatory to use GTM

To create a GTM account:

1. Go to https://tagmanager.google.com/
2. Click Create Account if you do not already have a Google account, otherwise, log in to your Google account, so you can access the GTM home page.
3. Click Create Account to create a Tag Manager account.



For more information, watch https://youtu.be/P4suvDuj0kI.

Import the Maestro GTM Container

The Maestro GTM Container is designed to run in isolation – the tags pre-configured within it will only produce page views for Maestro forms and their related pages. For this reason, it is safe to

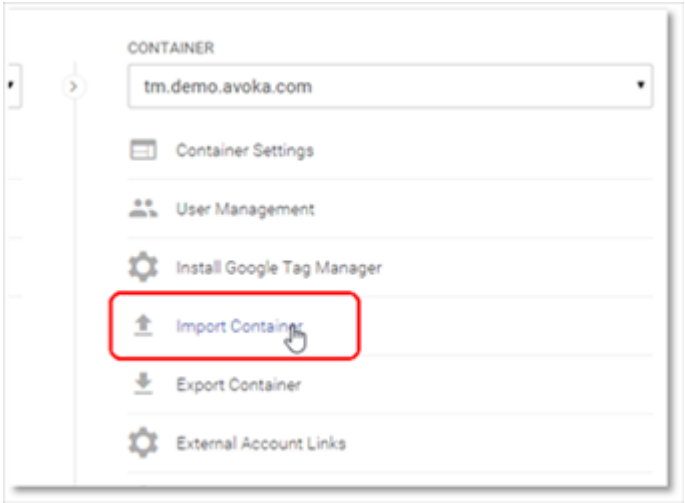
use either option for importing the container into GTM – either merge it into an existing container, or create a new one.

If you are choosing to merge it into an existing container, care should be taken that existing tags and triggers do not adversely affect the Maestro form pages. Additionally, if new tags are being created for Maestro forms, correct use of triggers should be maintained to ensure that the tags only fire in the correct places – not in other pages that may be connecting to the same GTM container.

> NOTE
>
> It is also recommended that if the GTM container for Composer forms has been or is still being used, then the Maestro container must be installed separately.

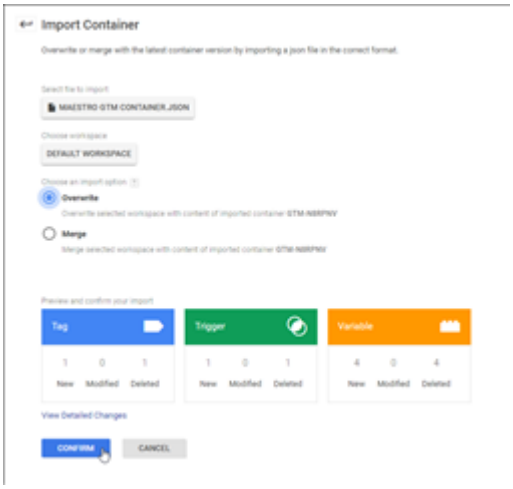
To import the GTM container:

1. Log in to GTM
2. Select the Accounts tab and choose a container to import into. You may need to create a new container if required.
3. Select the Admin tab and click Import Container in the Container section.



4. Under Select File to Import, select Choose Container File and navigate to the Maestro container file on your PC. You can also use the example of Maestro GTM container file.

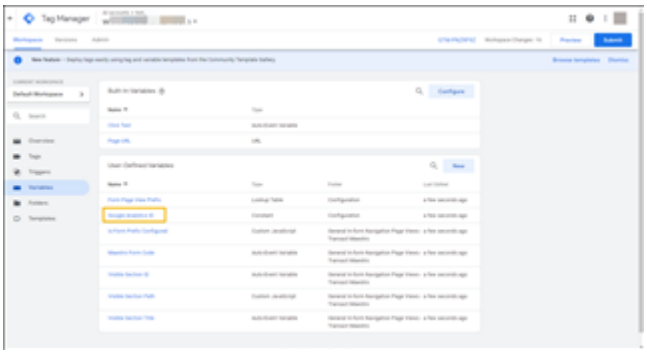
5. Select whether to use an existing Workspace or create a new one.



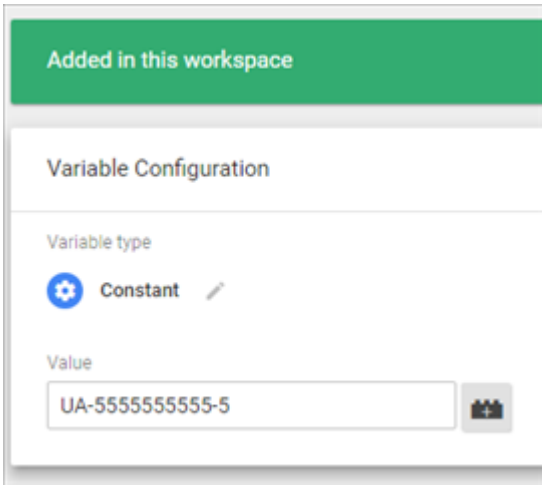
6. Select to either overwrite the existing container (for new containers only) or merge it.
7. Click Confirm.

Configure GTM to connect to Google Analytics

1. Select Container -> Variables > Google Analytics Id as a variable.



2. Enter the Google Analytics Tracking Id in the Value field.



3. Verify that page views are being sent to the correct Google Analytics account.

Specify the Virtual Page View URL Location

You also need to configure a meaningful URL location for each form page views that the down-stream analytics engine (GA) can use to correctly associate the views with a business process, as well as implementing Goals, Funnels, and so on.

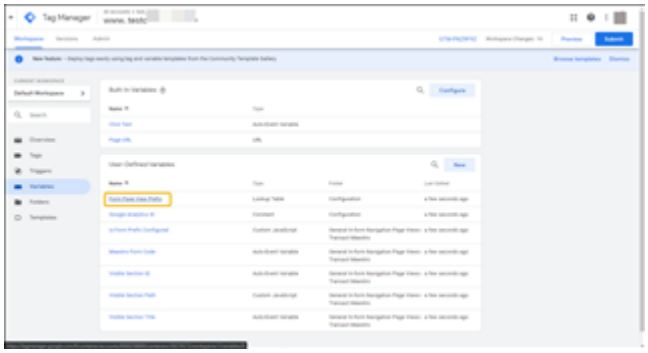
For example, if the form is a credit card application, then when the user navigates to the About You section of the form, rather than a page-view in GA on the location `/aboutyou/`, it could be more informative for it to be something like this: `/cards/credit/application/aboutyou/`.

For this reason, a look-up table has been provided in our pre-designed Maestro GTM that links the form code to a user-defined URL.

> NOTE
>
> It's important that this configuration is correct, otherwise, the form will never send page-views to Google Analytics.

To configure the virtual page view URL:

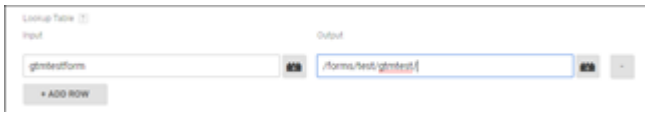
1. Select Workspace > Variables > Form Page View Prefix.



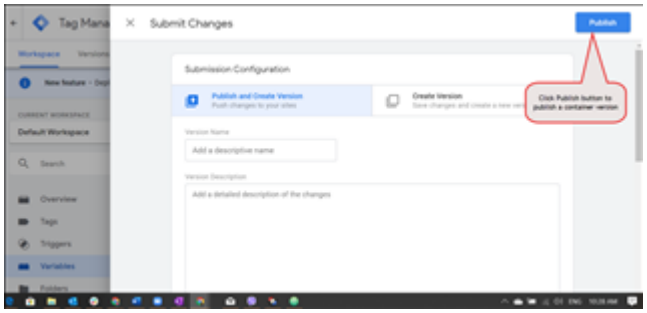
2. Click inside the Variable Configuration section to allow editing, and add a row to the Lookup Table.



3. Type the form code In the Input field.
4. Enter the page view prefix in the Output field. This prefix will be used for all Page Views generated by this form, and is a unique path that will identify the form in the Analytics engine.

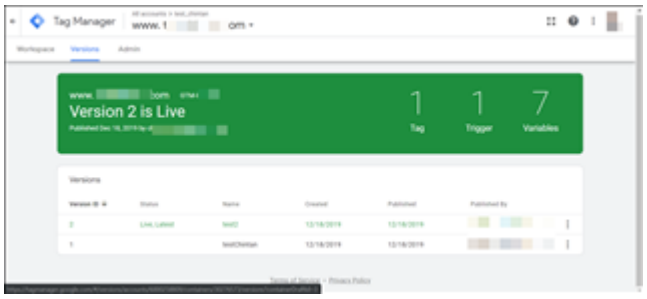


5. Click Save to save the variable.
6. Click Submit to apply the changes after all changes have been made.
7. Click Publish to publish the updated version of container. You also might need to create and publish a new version of a container.

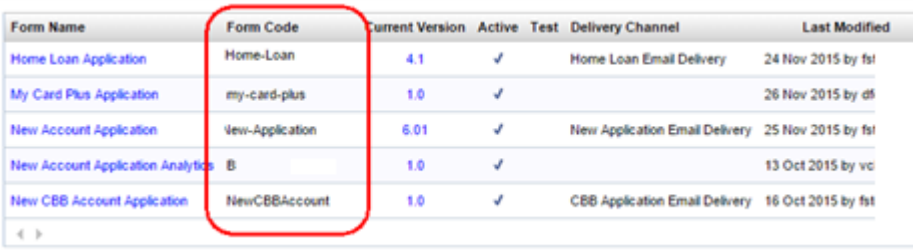


Once published, traffic on that form will generate the new page views, which will be sent to Google Analytics.

8. Select the Versions tab to see all published containers and all associated traffic from forms.



9. Log in to Manager and select Forms -> Forms to see the form codes for all published forms in that environment:



Now, you have completed all necessary GTM related configuration.

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

Navigate and Locate Items in GTM

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

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

- Specific Page Views - Maestro - contains Variables, Triggers and Tags for the page views that are to be generated for all pages outside of the actual Maestro form - such as confirmation pages, attachments pages and landing pages.

In-form Page Views

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

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

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

Other Page Views

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

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

- Submission Complete Page Opened (Trigger)
  - Page View Trigger
  - Fires on DOM Ready
  - Checks that the Page URL contains `confirmation.htm` and that a form prefix has been configured for this form - which automatically ensures the page is a TM form confirmation page, and that the form has been set to use Google Analytics

- Submission Complete Page View (Tag)
  - Google Analytics (Universal) tag type
  - Uses the configured form prefix with a hard-coded page name - i.e. `{{Form Page View Prefix}}submission`
  - Fires on the Submission Complete Page Opened trigger

Next, learn how to Autocomplete Fields for Dynamic Data Services.

Autocomplete Fields for Dynamic Data Services

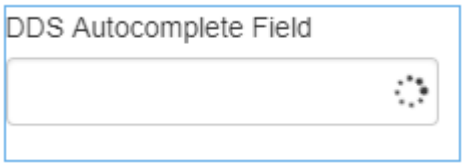
MaestroThe UI design product. | Form Builder Template Designer | All versions This feature is related to all versions.

The Maestro Autocomplete Field Component is very flexible, and with that can come complexity. Especially as the Component is now compatible with the newer Transact Functions. We'll show here how to connect Autocomplete Fields to Dynamic Data Services.

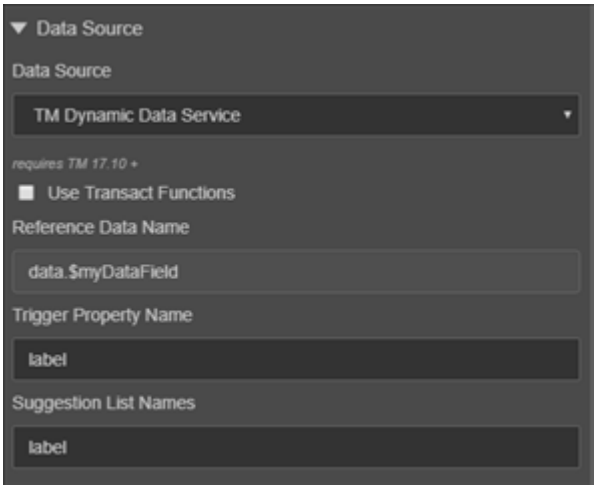
Maestro Component Properties

The Autocomplete Field Properties that we'll cover are separated into Data Source, and DDS Configuration. Second DDS Configuration will be reviewed later.

Obviously, you have to add an Autocomplete Field Component to your Form first.



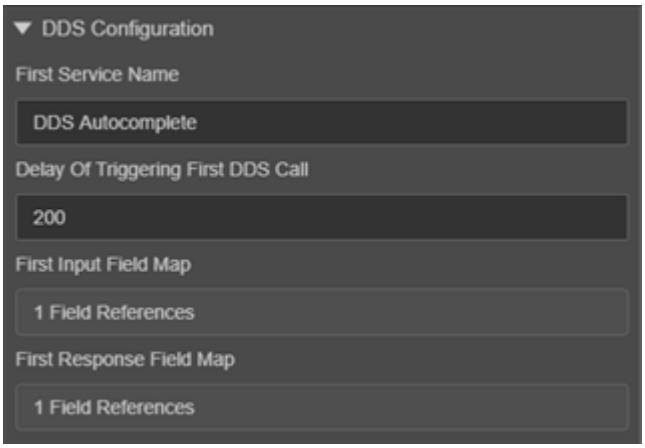
Then expand the Data Source panel on the Component Properties, shown below.



Set the Data Source dropdown to TM Dynamic Data Service, and then for this example set both the Trigger Property Name and the Suggestion List Names to "label". These names are used inside the Transact Dynamic Data Service.

Note: Do not check the "Use Transact Functions" checkbox! That is not compatible with Dynamic Data Services. How to use Transact Functions with Autocomplete Fields is a separate article.

Next, expand the DDS Configuration panel.



The first field is the name of the Transact Dynamic Data Service to be called - type that in. It must match the service name exactly.

Next, click the First Input Field Map control. Type the Key, our sample uses "searchString".



Then click in the Field Reference field, and double-click the Autocomplete Field in the tree on the left side of the dialog. This auto-populates the field data reference - you can type this manually if you like.

The First Response Field Map is set similarly, using "label" as the key.



Note that the response is being mapped back into the Autocomplete Field. In a single-DDS setup, the Autocomplete Field provides input to the DDS, and accepts the selected output!

Save, Build, and Deploy the form to Transact Manager. You should be able to open the form now, but of course the Autocomplete Field's DDS needs to be created.

For this article, a DDS was created that reads a list of animals from a Service Parameter, returning the ones that match what has been typed so far. The source is:

```
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import groovy.transform.TypeChecked
import groovy.json.JsonBuilder
import javax.servlet.http.*

@TypeChecked
class FluentDynamicData {

  // Injected at runtime
  public Logger logger

  /*
  * Perform Form Dynamic Data service call.
  *
  * returns: REST response text data
  */
  String invoke(SvcDef svcDef, Txn txn, HttpServletRequest request, User user) {

    String searchString = request.getParameter("searchString")

    logger.info "searchString == ${searchString}"

    String searchData = svcDef.paramsMap["searchData"];

    logger.info "searchData == ${searchData}"

    List searchList = searchData.tokenize(',')

    List baseResults = searchList.findAll { it.startsWith(searchString) }

    logger.info "baseResults == ${baseResults}"

    List results = []

    baseResults.each {
      results.push( ['label':it, 'data':it] )
    }

    logger.info results
```

```
    String data = new JsonBuilder(results).toString()

    return data
  }
}
```

The input into the service is the request parameter "searchString", which was mapped in the First Input Field Map on the DDS Configuration panel, shown above.

The output is the "results" object, which is a List containing maps with a "label" and a "data" elements. The "results" object is converted to JSON and sent back to the Form.

The use of "label" in the return from the service ties in to the use of "label" as the "Suggestion List Names" value in the Data Source panel shown above.

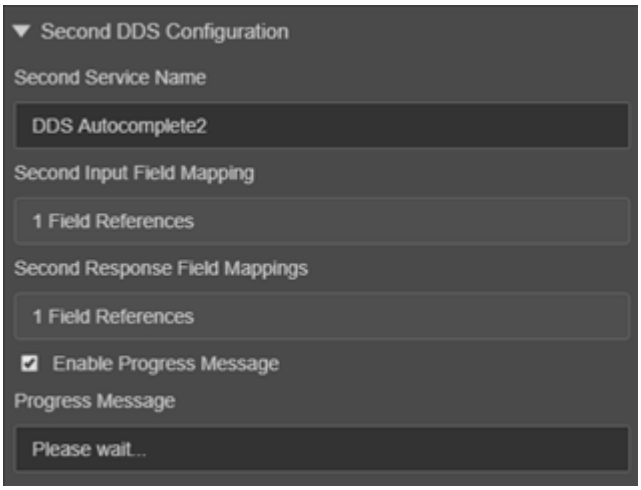
If you test the form now, it should populate with values from the service as you type.

Now, let's add the second DDS

The Autocomplete Field Component can be configured with a second DDS, which is called after the user makes a selection from the options and the output is delivered per the mappings configured. To use this functionality, add a Text Field to the Form.



Then open the Properties for the Autocomplete Field and update the Second DDS Configuration panel. Add a second service name, and use that name to create a new DDS on Transact Manager. Some sample code will be provided below.



The Second Input Field Mapping should be setup to look like the First Input Field Mapping.



The Second Response Field Mapping should be set to map the data keyed "label" to the new Text Field.

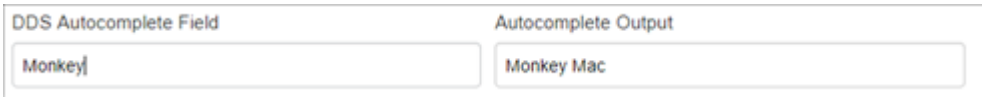


For this article, the second DDS, named "DDS Autocomplete2", uses another Service Parameter to find a name based on the first letter of the input "searchString". This creates a little alliteration to send back to the form.

```
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import groovy.transform.TypeChecked
import groovy.json.JsonBuilder
import javax.servlet.http.*

@TypeChecked
class FluentDynamicData {
  // Injected at runtime
  public Logger logger
  /*
  * Perform Form Dynamic Data service call.
  *
  * returns: REST response text data
  */
  String invoke(SvcDef svcDef, Txn txn, HttpServletRequest request, User user) {
  String searchString = request.getParameter("searchString")
   logger.info "searchString == ${searchString}"
   String searchData = svcDef.paramsMap["secondarySearchData"];
   logger.info "searchData == ${searchData}"
   List searchList = searchData.tokenize(',')
   List baseResults = searchList.findAll { it.startsWith(searchString[0]) }
   logger.info "baseResults == ${baseResults}"
   List results = []
   results.push( ["label":"${searchString} ${baseResults[0]}", "data":"${searchString}
${baseResults[0]}"] )
   logger.info results
   String data = new JsonBuilder(results).toString()
   return data
  }
}
```

Save and deploy everything, and the Autocomplete Field should repeatedly call the first DDS to find values that match the typed letters. When a returned value is selected, the second DDS is called and the return displayed in the Text Field.



The Service Parameter value used for `searchData` is:

```
Aard-
vark,Al-
batross,Al-
lig-
ator,Alpaca,Ant,Ar-
teat-
er,Ante-
lope,Ape,Ar-
madillo,B-
aboon,Badger-
,Bar-
racuda,Bat,Bear,Beaver-
,Bee,Bis-
on,Boar,But-
ter-
fly,Camel,Capy-
bara,Cari-
bou,Cas-
sowary,Cat,Cater-
pil-
lar,Cow,Chamois,Chee-
tah,Chick-
en,Chim-
pan-
zee,Chinchil-
la,Chough,Clam,Co-
bra,Cock-
roach,Cod,Cor-
mor-
ant,Coyote,Cr-
ab,Crane,Croc-
codile,Crow,Cur-
lew,Deer-
,Dino-
saur-
,Do-
g,Dog-
fish,Dol-
phin,Dor-
key,Dot-
```

```
terel,Dove,Dragor-
fly,Duck-
,Dugong,Dur-
lin,Eagle,Echid-
na,Eel,Eland,Elephant,Elk,Emu,Fal-
con-
,Fer-
ret,Finch,Fish,Flamin-
go,Fly,Fox,Frc-
g,Gaur,Gazelle,Ger-
bil,Gir-
affe,Gr-
at,Gnu,Goat,Goose,Goldfinch,Goldfish,Gorilla,Goshawk,Grasshopper,Grouse,Guanaco,Guinea
fowl,Guinea pig,Gull,Ham-
ster-
,Hare,Hawk,Hedge-
hog,Her-
on,Her-
ring,Hip-
popot-
amus,Hor-
net,Horse,Hum-
mingbird,Hyena,Ibex,Ibis,Jackal,Jaguar,Jay,Jellyfish,Kangaroo,Kingfisher,Koala,Komodo
Dragon,Kooka-
bur-
a,Kouprey,Ku-
du,Lap-
wing,Lark,Lemur,Lec-
pard,Lion,Llama,Lob-
ster-
,Lc-
cust,Lor-
is,Louse,Lyre-
bird,Mag-
pie,Mal-
lard,Manatee,Mandrill,Mantis,Marten,Meerkat,Mink,Mole,Mor-
goose,Mor-
key,Moose,Mouse,Mos-
quito,Mule,Nar-
whal,Newt,Nightir-
```

```
gale,Octopus,Okapi,C-
pos-
sum,Oryx,Ostrich,Ot-
ter-
,Owl,Ox,Oyster,Panther,Parrot,Partridge,Peafowl,Pelican,Penguin,Pheasant,Pig,Pigeon,Polar
Bear,Pony,Porcupine,Porpoise,Prairie Dog,Quail,Quelea,Quet-
zal,Rabbit,Raccoon,Rail,Ram,Rat,Raven,Red Panda,Reindeer-
,Rhinoceros,Rook,Salamander,Salmon,Sand Dollar,Sandpiper,Sardine,Scorpion,Sea Lion,Sea
Urch-
in,Seahorse,Seal,Shark,Sheep-
,Shrew,Skunk,S-
nail,Snake,S-
par-
row,Spider-
,Spoor-
bill,Squid,S-
quir-
rel,Starling,St-
ingray,Stink-
bug,St-
ork,Swal-
low,Swan,Ta-
pir,Tar-
sier,Termite,Tiger,Toad,Trout,Turkey,Turtle,Viper,Vulture,Wallaby,Walrus,Wasp,Water
Buffalo,Weasel,Whale,Wolf,Wolverine,Wombat,Woodcock,Woodpecker,Worm,Wren,Yak,Zebra
```

And the Service Parameter value used for `secondarySearchData` is:

```
Abby,Ace,Al-
lie,Ar-
gel,Ar-
nie,A-
pol-
lo,Archie,Ather-
a,Ava,Baby,Bailey,Bailey,Bandit,Baxter,Bear,Beau,Bella,Belle,Benji,Benny,Bentley,
Blue,Bo,Bonnie,Boomer,Brady,Brody,Bruce,Bruno,Brutus,Buddy,Buster,Cali,Callie,Cash,
Chance,Charlie,Charlie,Chase,Chester,Chewy,Chico,Chloe,Cleo,Coco,Coco,Cody,Cookie,
Cooper,Copper,Daisy,Dakota,Delilah,Dexter,Diesel,Dixie,Duke,Ella,Ellie,Emma,
Finn,Fiona,Frankie,George,Georgia,Gigi,Ginger,Gizmo,Grace,Gracie,Gunner,Gus,
Hank,Harley,Harley,Harper,Hazel,Heidi,Henry,Holly,Honey,Hunter,Ivy,Izzy,Jack,
Jackson,Jake,Jasmine,Jasper,Jax,Joey,Josie,King,Kobe,Koda,Kona,Lacey,Lady,Layla,
Leia,Leo,Lexi,Lilly,Lily,Loki,Lola,Louie,Lucky,Lucy,Luke,Lulu,Luna,Mac,Macy,Maddie,
Panut,Peanut,Pearl,Penelope,Penny,Pepper,Phoebe,Piper,Poppy,Prince,Princess,Ranger,
Remi,Rex,Riley,Riley,Rocco,Rocky,Romeo,Roscoe,Rosie,Roxie,Roxy,Ruby,Rudy,Rusty,
Sadie,Sam,Sammy,Samson,Sandy,Sasha,Scout,Scout,Shadow,Shadow,Shelby,Simba,Sophie,
Stella,Sugar,Tank,Teddy,Theo,Thor,Toby,Trixie,Tucker,Tyson,Willow,Winnie,Winston,
Xena,Zeke,Zeus,Ziggy,Zoe,Zoey
```

Autocomplete Fields for Transact Functions

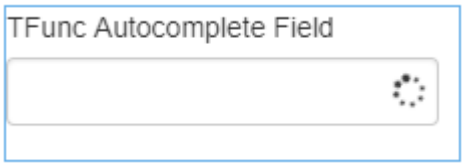
MaestroThe UI design product. | Form Builder Template Designer | All versions This feature is related to all versions.

The Maestro Autocomplete Field Component is very flexible, and with that can come complexity. Especially as the Component is now compatible with the newer Transact Functions. We'll show here how to connect Autocomplete Fields to a Form Function-triggered Transact Function.

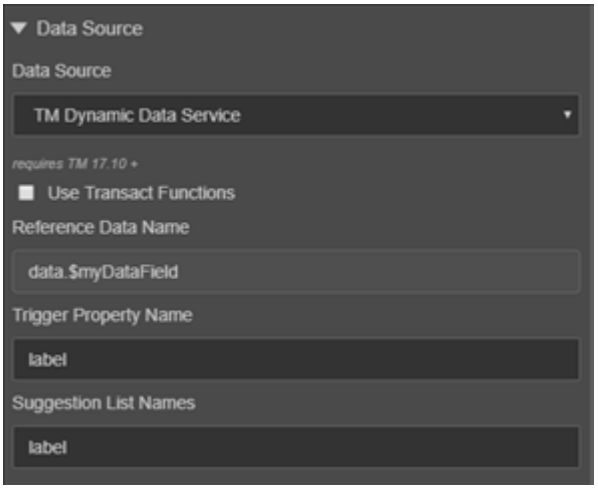
Maestro Component Properties

The Autocomplete Field Properties that we'll cover are separated into Data Source, and DDS Configuration. Second DDS Configuration will be reviewed later.

Obviously, you have to add an Autocomplete Field Component to your Form first!

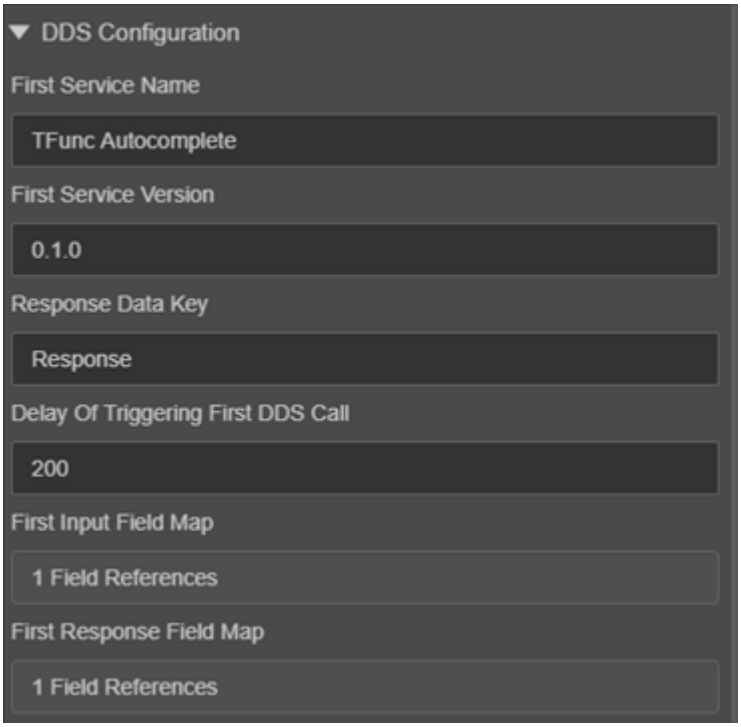


Then expand the Data Source panel on the Component Properties, shown below.



Set the Data Source dropdown to TM Dynamic Data Service. That seems counter-intuitive, but the other option, Reference Data, does not call any service. To make use of a Transact Function rather than a legacy Dynamic Data Service, check the box for "Use Transact Functions". For this example also set both the Trigger Property Name and the Suggestion List Names to "label". These names are used inside the Transact Dynamic Data Service.

Next, expand the DDS Configuration panel.



The first field is the name of the Transact Function to be called - type that in. It must match the service name exactly.

Similarly, type in the version of the service. The Response Data Key can be left as Response -- it is used in the Groovy code to identify the returned values in the FormFuncResult.

Next, click the First Input Field Map control. Type the Key, our sample uses "searchString".



Then click in the Field Reference field, and double-click the Autocomplete Field in the tree on the left side of the dialog. This auto-populates the field data reference - you can type this manually if you like.

The First Response Field Map is set similarly, using "label" as the key.



Note that the response is being mapped back into the Autocomplete Field. In a single-DDS setup, the Autocomplete Field provides input to the Transact Function, and accepts the selected output!

Save, Build, and Deploy the form to Manager. You should be able to open the form now, but of course the Autocomplete Field's Transact Function needs to be created.

For this article, a Transact Function was created that reads a list of animals from a Service Parameter, returning the ones that match what has been typed so far. The source is:

```
import com.avoka.tm.func.*
import com.avoka.tm.svc.*
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import groovy.transform.TypeChecked
import groovy.json.JsonBuilder
import javax.servlet.http.*

@TypeChecked
class FluentFunction {
  // Injected at runtime
  public Logger logger

  /*
  * Perform Fluent Function call.
  *
  * returns: FuncResult
  */
  FuncResult invoke(FuncParam param) {
    String searchString = param.params.get('searchString').toString()
    logger.info "searchString == ${searchString}"
    String searchData = param.svcDef.paramsMap["searchData"];
    logger.info "searchData == ${searchData}"
    List searchList = searchData.tokenize(',')
    List baseResults = searchList.findAll { it.startsWith(searchString) }
    logger.info "baseResults == ${baseResults}"
    List results = []
    baseResults.each {
      results.push( ['label':it, 'data':it] )
    }
    logger.info results
    String data = new JsonBuilder(results).toString()
    FormFuncResult result = new FormFuncResult()
    result.data["Response"] = results
    return result
  }
}
```

The input into the service is the parameter "searchString", which was mapped in the First Input Field Map on the DDS Configuration panel, shown above.

The output is the "results" object, which is a List containing maps with a "label" and a "data" elements. The "results" object is added to the FormFuncRetsult object and sent back to the Form.

The use of "label" in the return from the service ties in to the use of "label" as the "Suggestion List Names" value in the Data Source panel shown above.

Lastly, with Transact Functions, the Function and its Trigger must be added to the Form Version. Navigate Transact Manager to the deployed Form, and in the Version add this Transact Function with a Form Function Trigger and a Sequence of 1.



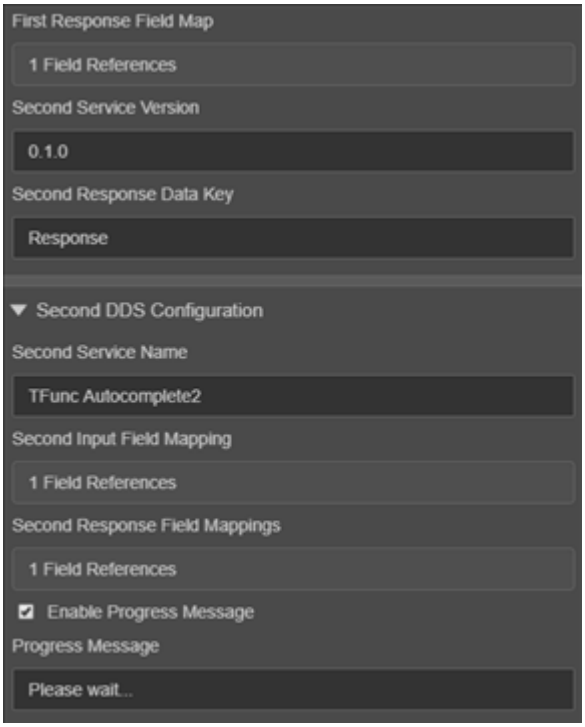
If you test the form now, it should populate with values from the service as you type.

Let's look how to add a second DDS

The Autocomplete Field Component can be configured with a second Transact Function, which is called after the user makes a selection from the options and the output is delivered per the mappings configured. To use this functionality, add a Text Field to the Form.



Then open the Properties for the Autocomplete Field and update the bottom of the DDS Configuration panel and Second DDS Configuration panel. Add a second service name, and use that name to create a new DDS on Transact Manager. Some sample code will be provided below for the second service.



Note that the DDS Configuration panel has some settings for the second service -- Second Service Version and Second Response Data Key. These two fields are NOT in the Second DDS Configuration Panel!

The Second Input Field Mapping should be setup to look like the First Input Field Mapping.



The Second Response Field Mapping should be set to map the data keyed "label" to the new Text Field.



For this article, the second Transact Function, named "TFunc Autocomplete2", uses another Service Parameter to find a name based on the first letter of the input "searchString". This creates a little alliteration to send back to the form.

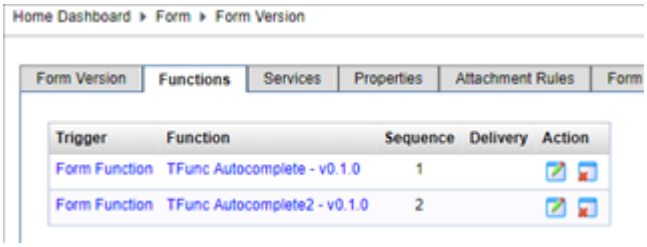
```
import com.avoka.tm.func.*
import com.avoka.tm.svc.*
import com.avoka.tm.util.*
import com.avoka.tm.vo.*
import groovy.transform.TypeChecked
import groovy.json.JsonBuilder
import javax.servlet.http.*

@TypeChecked
class FluentFunction {

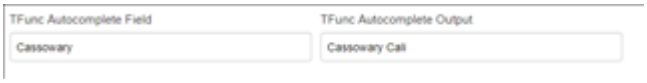
// Injected at runtime
public Logger logger

/*
* Perform Fluent Function call.
*
* returns: FuncResult
*/
FuncResult invoke(FuncParam param) {
String searchString = param.params.get('searchString').toString()
logger.info "searchString == ${searchString}"
String searchData = param.svcDef.paramsMap["secondarySearchData"];
logger.info "searchData == ${searchData}"
List searchList = searchData.tokenize(',')
List baseResults = searchList.findAll { it.startsWith(searchString[0]) }
logger.info "baseResults == ${baseResults}"
List myresults = []
String comboName = "${searchString} ${baseResults[0]}".toString()
myresults.push( ['label':comboName, 'data':comboName] )
FormFuncResult result = new FormFuncResult()
result.data["Response"] = myresults
logger.info result.data.Response
return result
}
}
```

Go back to the Form Version deployed on Transact, and add this new Transact Function to the Form Version. The trigger is Form Function again, and Sequence can be 2.



Save and deploy everything, and the Autocomplete Field should repeatedly call the first Transact Function to find values that match the typed letters. When a returned value is selected, the second Transact Function is called and the return displayed in the Text Field.



The Service Parameter value used for searchData is:

```
Aard-
vark,Al-
batross,Al-
lig-
ator,Alpaca,Ant,Ar-
teat-
er,Ante-
lope,Ape,Ar-
madillo,B-
aboon,Badger-
,Bar-
racuda,Bat,Bear,Beaver-
,Bee,Bis-
on,Boar,But-
ter-
fly,Camel,Capy-
bara,Cari-
bou,Cas-
sowary,Cat,Cater-
pil-
lar,Cow,Chamois,Chee-
tah,Chick-
en,Chim-
pan-
zee,Chinchil-
la,Chough,Clam,Co-
bra,Cock-
roach,Cod,Cor-
mor-
ant,Coyote,Cr-
ab,Crane,Croc-
codile,Crow,Cur-
lew,Deer-
,Dino-
saur-
,Do-
g,Dog-
fish,Dol-
phin,Dor-
key,Dot-
```

```
terel,Dove,Dragor-
fly,Duck-
,Dugong,Dur-
lin,Eagle,Echid-
na,Eel,Eland,Elephant,Elk,Emu,Fal-
con-
,Fer-
ret,Finch,Fish,Flamin-
go,Fly,Fox,Frc-
g,Gaur,Gazelle,Ger-
bil,Gir-
affe,Gr-
at,Gnu,Goat,Goose,Goldfinch,Goldfish,Gorilla,Goshawk,Grasshopper,Grouse,Guanaco,Guinea
fowl,Guinea pig,Gull,Ham-
ster-
,Hare,Hawk,Hedge-
hog,Her-
on,Her-
ring,Hip-
popot-
amus,Hor-
net,Horse,Hum-
mingbird,Hyena,Ibex,Ibis,Jackal,Jaguar,Jay,Jellyfish,Kangaroo,Kingfisher,Koala,Komodo
Dragon,Kooka-
bur-
a,Kouprey,Ku-
du,Lap-
wing,Lark,Lemur,Lec-
pard,Lion,Llama,Lob-
ster-
,Lc-
cust,Lor-
is,Louse,Lyre-
bird,Mag-
pie,Mal-
lard,Manatee,Mandrill,Mantis,Marten,Meerkat,Mink,Mole,Mor-
goose,Mor-
key,Moose,Mouse,Mos-
quito,Mule,Nar-
whal,Newt,Nightir-
```

```
gale,Octopus,Okapi,C-
pos-
sum,Oryx,Ostrich,Ot-
ter-
,Owl,Ox,Oyster,Panther,Parrot,Partridge,Peafowl,Pelican,Penguin,Pheasant,Pig,Pigeon,Polar
Bear,Pony,Porcupine,Porpoise,Prairie Dog,Quail,Quelea,Quet-
zal,Rabbit,Raccoon,Rail,Ram,Rat,Raven,Red Panda,Reindeer-
,Rhinoceros,Rook,Salamander,Salmon,Sand Dollar,Sandpiper,Sardine,Scorpion,Sea Lion,Sea
Urch-
in,Seahorse,Seal,Shark,Sheep-
,Shrew,Skunk,S-
nail,Snake,S-
par-
row,Spider-
,Spoor-
bill,Squid,S-
quir-
rel,Starling,St-
ingray,Stink-
bug,St-
ork,Swal-
low,Swan,Ta-
pir,Tar-
sier,Termite,Tiger,Toad,Trout,Turkey,Turtle,Viper,Vulture,Wallaby,Walrus,Wasp,Water
Buffalo,Weasel,Whale,Wolf,Wolverine,Wombat,Woodcock,Woodpecker,Worm,Wren,Yak,Zebra
```

And the Service Parameter value used for secondarySearchData is:

```
Abby,Ace,Al-
lie,Ar-
gel,Ar-
nie,A-
pol-
lo,Archie,Ather-
a,Ava,Baby,Bailey,Bailey,Bar-
dit,Bax-
ter-
,Bear,Beau-
,Bel-
la,Belle,Benji,Benny,Bent-
ley,Blue,Bo,Bor-
nie,Boomer-
,Brady,Brody,Bruce,Brur-
o,Bru-
tus,Buddy,Buster-
,Cali,Cal-
lie,Cash,Chance,Charlie,Charlie,Chase,Chester-
,Chewy,Chico,Chloe,Cleo,Cc-
co,Cc-
co,Cody,Cook-
ie,Coop-
er,Cop-
per-
,Daisy,Dakota,De-
lil-
ah,Dex-
ter-
,Dies-
el,Dix-
ie,Duke,El-
la,El-
lie,Em-
ma,Fir-
n,Fior-
a,Frankie,Ge-
orge,Ge-
orgi-
a,Gi-
```

```
gi,Ginger-
,Giz-
mo,Grace,Gra-
cie,Gur-
ner-
,Gus,Hank,Har-
ley,Har-
ley,Harper-
,Hazel,Heidi,Henry,Holly,Honey,Hur-
ter-
,Ivy,Izzy,Jack-
,Jack-
son,Jake,Jas-
mine,Jasper-
,Jax,Joey,Josie,King,Kobe,Koda,Kor-
a,Lacey,Lady,Lay-
la,Lei-
a,Leo,Lex-
i,Lilly,Lily,Loki,Lola,Louie,Luck-
y,Lucy,Luke,Lu-
lu,Lur-
a,Mac,Macy,Mad-
die,Mag-
gie,Mar-
ley,Mar-
ley,Mav-
erick-
,Max,May-
a,Mi-
a,Mil-
a,Mil-
lie,Milo,Mir-
nie,Mis-
sy,Mocha,Molly,Moose,Murphy,Nala,Nova,Od-
in,Olive,Oliv-
er,Ol-
lie,Oreo,Oreo,Os-
car,Ot-
is,Pais-
ley,Pea-
```

```
nut,Pea-
nut,Pearl,Penelope,Penny,Pep-
per-
,Phoe-
be,Piper-
,Poppy,Prince,Prin-
cess,Ranger-
,Remi,Rex,Ri-
ley,Ri-
ley,Roc-
co,Rock-
y,Romeo,Ro-
scoe,Rosie,Rox-
ie,Roxy,Ruby,Rudy,Rusty,Sadie,Sam,Sammy,Sam-
son,Sandy,Sasha,S-
cout,S-
cout,Shad-
ow,Shad-
ow,Shelby,Sim-
ba,Sophie,S-
tel-
la,Sugar,Tank,Teddy,Theo,Thor,Toby,Trix-
ie,Tucker,Tyson,Willow,Winnie,Winston,Xena,Zeke,Zeus,Ziggy,Zoe,Zoey
```

Deep clearing of Chrome browser cache

MaestroThe UI design product. | Form Builder ▪ Template Designer | All versions This feature is related to all versions.

Sometimes no matter how you try to clear it, Chrome's cache seems to just hang on for dear life.

This article will show you a method of deep clearing Chrome's cache that seems to work consistently, especially when dealing with new releases of Maestro.

Try this method when you have not had success with the common means of clearing Chrome's cache such as:

Menu item: More tools... Clear browsing data, then selecting 'the beginning of time' and at least 'Cached images and files'

Clicking refresh icon while holding `Ctrl` key down.

We will use Chrome's developer tools panel to get to the Network tab where we can see the list of files the browser is fetching and caching.

1. Navigate to the Maestro Management Dashboard:
2. Press `F12` to bring up the Developer Tools panel
3. Click on the Network tab
4. Check the Disable Cache checkbox, your screen should now look a bit like this:
5. Hold down the `Ctrl` key and click the refresh icon in the URL bar , then wait for the file downloads to complete
6. Scroll through the list of files that were downloaded and look for ones that have '(from disk cache)' in the Size column
7. Right click the filename in the Name column and select 'Clear browser cache':
8. Hold the Ctrl key down and click the Refresh icon again
9. Close Developer Tools panel by clicking the X on top right of the panel, or just press `F12`

Your persistent Chrome cache should now be cleared more thoroughly than before and you should be able to eliminate browser caching as the root cause of odd behaviour you might have been experiencing.

Maestro File Sizes

MaestroThe UI design product. | Form Builder Template Designer | All versions This feature is related to all versions.

Maestro file sizes can become quite large when the HTML file that is used to create the form and the assets that are used to extend the functionality of the form are mismanaged. The file size of a Maestro form can affect the efficiency of the form when it is published and deployed to an environment. Large file sizes can also lead to forms that are slow and unresponsive when being edited by form builders and developers.

There are several approaches to take to keep Maestro file sizes down. The list below identifies several approaches that are recommended to minimize the file size of a Maestro form.

- When publishing a form, ALWAYS select the Minify Code build option checkbox. Selecting this build option will reduce the size of your form without any extra work and will ensure that your form is efficient to transmit across different deployment environments. The screenshot below highlights the Minify Code build option checkbox.
- Never publish a form that still has comments in the code. Publishing a form with comments will increase the size of the Maestro HTML file and extend the time it takes the form to load each time you want to edit the form.
- Use the Repeating Block Template when multiple instances of the same component are required. For example, when asking for multiple family members names, use a single repeating block template rather than hard-coded text inputs for each family member. Using repeats can decrease the size of the HTML file by 50%.
- Where possible, use a common JavaScript Library to store Maestro business rules, DO NOT repeat the same lines of code when the same rule is used in multiple places in the form.
- Where possible, do not hardcode long dropdown list options, instead, use other means to populate long dropdown list options. For example, do not hardcode a list of countries.

Access to Form Reference Data

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 19.05 This feature was introduced in 19.05.

Many applications require some type of reference data that infrequently changes, such as lists of countries, vehicle manufacturers, models, years, titles, product types, and so on.

Maestro allows Form Designers to access reference data using the following techniques:

- JavaScript library
- Property prefill
- Dynamic Data service

JavaScript Library

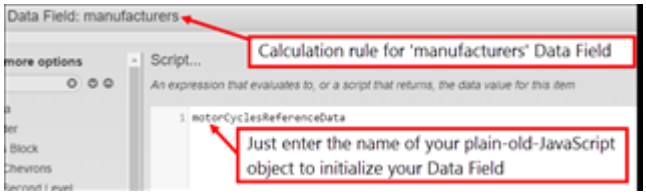
The Form Designer supplies JavaScript source, which is embedded directly into the form as a JavaScript Library component. Reference data under this scenario stays with the form and has no external dependencies, with the benefit that versioning of the data is managed by Maestro as part of the form content.

The reference data here is embedded in the form inside a JavaScript library. In the sample data below, we have used the property names `label` and `value` in our JSON object. These property names are the default values for displaying in Data Driven Dropdown components. By using these names, we can accept the default values for the dropdown's `Label Field` and `Value Field` properties. We can use any names we want in our JSON though, and simply change the dropdown properties to suit. To do this, follow these steps:

1. Create a JavaScript file that contains a variable holding your reference data as a JSON object. You can have as many variables as you want, but for this example, we will only use one, `motorCyclesReferenceData`.
2. Add a JavaScript Library component to your form, and upload your JavaScript file into it.



3. Add a Data Field component to your form to hold each variable your JavaScript file contains. We use the name 'manufacturers'
4. Add a Calculation rule to the manufacturers Data Field.



5. In the Calculation rule, put the name of the JavaScript variable and save the rule.
6. Reference your Data Field component as you want, in other parts of the form.
7. Add a Data Driven Dropdown to your form, calling it 'Manufacturer'.
8. In the Data Source property, double-click the 'manufacturers' Data Field.
9. Preview the form.
10. Your Manufacturer dropdown will now show values from your reference data.

Property Prefill

Reference data is stored and managed by Journey Manager and injected into the form's XML data at the time of the rendering of the form. There is a non-trivial setup process required in Manager, but one benefit is that reference data remains external to the form. For more information, see Prefill a Cascading Dropdown in Composer and Form Prefill for Anonymous Users. Be aware that prefilled data will be returned along with other data fields in the submission XML. For this reason, using prefill for reference data is not the ideal choice. For example, if your reference data is a list of country codes, you don't really what to have this data come back in the submission XML.

The reference data here is held in Manager as a Form Property. Before you can assign a value to a 'Property', you need to create a Property Type for it. To do this, follow these steps:

1. Log on to Maestro.
2. Create a form and add a Data Field, calling it 'ManufacturersPropertyData'.
3. On the Data tab, check 'Include in Submission Data', otherwise you will not be able to map property data into this field in the TM Property Mapping panel. Checking this checkbox also causes the data field to be sent back with the submission XML data, which we probably don't want to do for reference data. We can prevent this from happening with some code.
4. Add a Data Driven Dropdown, calling it 'ManufacturersPropertyDropdown'.
5. Change the Data Source to `data.$tempManufacturers`.
6. Add the following Form Load rule to the form:

```
data.$tempManufacturers = JSON.parse(data.manufacturerspropertydata);
delete data.manufacturerspropertydata;
delete data.prefill.manufacturerspropertydata;
```

7. This rule converts the JSON formated text value that has been injected into the ManufacturersPropertyData data field into a JSON object. Then, it saves the converted JSON object into a temporary data field `$tempManufacturers` - form fields with names beginning with '$', are not sent back in the submission XML data. It removes the manufacturerspropertydata field from the form data object, so it will not be sent back in the submission XML. It also removes the manufacturerspropertydata field from the form prefill data object, otherwise it will be merged back into the form data object before submission
8. Save and Publish your form to Manager
9. Log on to Manager.
10. Select Forms > Property Types and click New to create a new property type.
11. Name it as ReferenceData.
12. Select a scope as Client.
13. Select a data type as JSON.
14. Save the new property type.
15. Select Forms > Forms and find your form to open it in the Dashboard window.
16. Click the Properties link in the Form Versions section.
17. Click New to create a new property.
18. Name it as ReferenceData.
19. Choose ReferenceData from the Property Type dropdown list.

20. Paste your JSON data into the Value field, which you can copy from the `Motorcycles.json` file.

    NOTE

    If you try to create a new property of the same Property Type in your form, you cannot do that. A Property Type can only be given a value by one Property in the same form.

21. Click Save.
22. Re-open your form in the Dashboard window.
23. Click Data Config link in the Form Versions section and select the Property Prefill Mapping tab.
24. Click Edit.
25. Scroll the left pane down until you can see the ReferenceData property
26. From the right pane 'Target: Form Data XML', drag 'ManufacturersPropertyData' into the 'Form Data XPath' column and drop it beside 'ReferenceData'
27. Click Save
28. In some browsers, you may now be left with a blank screen. If so, press F5 to refresh the page.
29. Open the form and select the Dashboard tab.
30. Test the form by clicking the icon under 'Direct' in the Form URLs box
31. Your form should now show data in the dropdown

Dynamic Data Service

Reference data is stored and managed by running a custom Fluent Groovy Service in Manager and made available at run-time when the form decides to fetch it. An advantage of this method is the potential ability to dynamically filter the data. Reference data is still external to the form using this method, but it does require the creation of a service in Manager. A form builder will need to write code to persist the dataaccess the service and deal with the response.

The table below lists compares the features of each techniques.

Feature	JavaScript Library	Property Pre-fill	Dynamic Data Service
Works in Preview mode	Yes	No	No
Version control	Managed by Form versioning	None	Determined by service

Use When	form-specific datain-frequent data changesno sharing requiredstatic data requiredwork entirely in Maestro	sharing requiredstatic data requiredneed to override at org/form/user	filtering requiredsharing required
Manager content validation	N/A	Yes	Determined by code
Supports lazy-loading	No	No	Yes
Share-ability	None	Shareable	Shareable
Security	None	None	Determined by code
Re-publish required after change	Yes	No	No
No-code implementation	No	Only for simple data type	No
Impact on save/retrieved forms	No	May break	None
Impact on review task forms	None	May break	None
Good Points	easy to implementno TM action required-versioned with formno server calls needed	content managed by TMhierarchical org/form/user overridesno server calls needed	most flexiblelazy loadabilityfilterablesecurable
Filterable data source	No	No	Yes, if coded
Data variability	Static	Static	Dynamic, if coded
Data type	Object	String, can be converted to Object in code	Object
Data source	Embedded in JavaScript code	TM Properties	Determined by service

Data instance location	Global context	Form XML	Determined by code
Data associated with	Form	Organization/Form/User	Determined by service
Complexity of TM setup	Not required	Medium	Complex
Bad points	re-publish on change-scoped directly to formcannot be sharedstatic data only	complex TM setupmay break review/retrieved form-scode required for object conversioninitially, but contaminates form XMLstatic data only	higher coding requirementoverkill for simple cases

Each of the following guides details how to gain access to reference data using the techniques discussed above. In each case, we populate an initial Data Field with some sort of data (for this example we will use data from motorcycle manufacturers). At the end of the guides, we will demonstrate methods of using the manufacturer's data to build cascading data dropdowns. Building the cascading dropdowns with each method is exactly the same process, the only difference is how you initially obtain the reference data.

Here we will create a new Manager service which will return our JSON as text. Then we will call the new service from a form to populate a dropdown as before. We will call our new service from the form's Load event.

1. In Manager, choose All Services from the Services menu and click New
2. Choose Service Type 'Dynamic Data'
3. Choose Service Template 'Fluent Dynamic Data'
4. Enter Service Name 'Motorcycle Manufacturers'
5. Choose your Organization
6. Click Save
7. Select the Groovy Script tab
8. Replace all the existing JSON text between the pairs of triple single-quotes (''') with text from this Motorcycles.json file
9. Your complete code should now look like:



10. Click Save
11. In Maestro, create a new form
12. Drop a Data Field onto the form, call it 'ManufacturersDDS'
13. Drop a Data Driven Dropdown onto the form, call it 'Manufacturers From DDS'
14. Change its Data Source to data.manufacturersdds by double-clicking on ManufacturersDDS
15. Click the Form Options button in the Maestro control bar
16. Click Form Rules
17. Click Create Rule and click Load under 'Form'
18. Paste in this code:

```
DynamicData.call("Motorcycle Manufacturers", {}).then(function(response) {data.-
manufacturersdds = response;});
```

19. Click Save
20. Click Save again
21. Save and Publish the form
22. Render the form and you will see data from the service in the dropdown

Add Cascading Dropdowns

To complete your reference data implementation, you need to add cascading dropdowns. Here we will add additional dropdowns for Model and Year to the form. When the value selected in the manufacturer dropdown changes, the model will be re-populated with the models available from the chosen manufacturer. Similarly, when the model dropdown selection changes, the years dropdown will be re-populated.

Because we have continued to use the 'label' and 'value' properties throughout all levels of our JSON data, we don't need to change the 'Label Field' and 'Value Field' properties on any of the dropdowns.

The instructions here relate to the initial Data Field named 'manufacturers' and Dynamic Data Dropdown called 'manufacturer'. We created these components in the JavaScript library method. If you followed the JavaScript Library instructions and saved your form, re-use that form here so that your data fields and dropdown names match these instructions.

1. In Maestro, open your form created during the JavaScript Library method instructions
2. Check the Calculation rule on the 'manufacturers' Data Field... it should already contain 'motorCyclesReferenceData' from the JavaScript Library method instructions
3. Add 2 more Data Field components to your form, name them 'models' and 'years'
4. Add 2 more Dynamic Data Dropdown components, name them 'Model' and 'Year'
5. Change the Data Source properties of Model and Year dropdowns to the Data Fields 'models' and 'years'
6. Add a Change rule to the Manufacturer dropdown, with the code:

```
var selected = data.manufacturer;
data.manufacturers.some(function(element) {
    if (element.value === selected) {
        data.models = element.models;
        data.years = undefined;
        return true;
    }
});
return undefined;
```

7. The code above will execute when the Manufacturer dropdown selection is changed. The code just replaces the Data Fields 'models' with the child property 'models' from the selected manufacturer, and also sets 'years' to undefined.
8. At this stage, you could Publish and render your form and see the Model dropdown populates on a change of Manufacturer, but the Year dropdown is always empty. We will implement a Model Change rule to bring the Year dropdown into play.

9. Now add a Change rule to the Model dropdown, with the code:

```
data.years = data.models.filter(function(element) {
    return element.value === data.model
})[0].years;
```

10. The code above just sets the 'years' Data Field to the years property of the currently select model. It closely follows the Manufacturer change rule code but is implemented slightly differently to demonstrate there are different methods that can be used to achieve the same result. In this case, we use the JavaScript filter function to find the matching data from the models Data Field.
11. Save and Publish your form

Now when you render the form, you will see the cascading behavior between the Manufacturer, Model, and Year dropdowns.

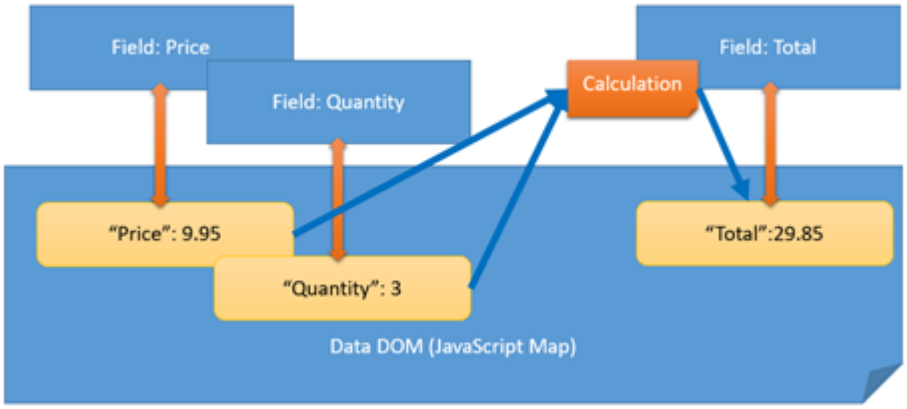
Next, learn about pre-submit validation.

Maestro Data Models

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Maestro data model is a powerful yet simple abstraction that allows you to use much of the Journey Maestro functionality when you are developing a form, so it is important to understand it.

When you type a piece of data into a component in Journey Maestro, the data is not only displayed in the component itself but is also copied to an internal data structure known as the data model, or the data DOM[1]. The component and its value are sometimes known as the view. These terms both derive from well-known programming pattern Model-View-Controller. In our case, the controller is any business rule that is defined in Maestro, as shown below.



Maestro uses a unique field ID to identify both the component and its value in the data model. Internally, the data model is implemented as a JavaScript map. You can think of the data model as being a spreadsheet with two columns: the ID of the component and its value. This is illustrated with a sample table.

Field ID	Value
"Price"	9.95
"Quantity"	3
"Total"	29.85

---

[1]The Document Object Model (DOM) is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document. The objects can be manipulated programmatically and any visible changes occurring as a result may then be reflected in the display of the document.

The component's value and the value in the data model are automatically synchronized in both directions. Therefore, when you are programming in Maestro, it is usually much easier to modify the data in the data model and let the automatic synchronization take care of updating the visible values in the components themselves. If you refer to the above diagram, you will see that the calculations refer to the values in the data model, rather than referring directly to the components themselves.

For example, when you are in the Script Editor, and double-click on the price component, what is inserted into your script is "data.price". This is not actually referring to the component itself, but rather referring directly to the component's value in the data model. The "data." is the clue that you are referring to the data model. The values in the data model can be both read and written, and the synchronization will take care of updating all the components in the view automatically.

To learn how to use Data Mapper in Journey Maestro, see data mapper.

It is important to ratify your data model upfront as much as possible before setting out - and do not assume that a model which represents a downstream integration is a good way to operate.

For example, if you are integrating with Vendor A, and Vendor A requires the following format of output for "pets":

```
<Classification>
   <PartA>
      <Pets>
         <Pet>
            <Name/>
            <Type/>
         <Pet>
      </Pets>
   </PartA>
<Classification/>
```

It may be tempting to include this structure as part of your form. This has the benefit of making the payload meet the requirements for integration with little work. However, the forms will suffer as small tweaks to the schema for output require republishing every form.

Consider a client data dictionary, as it has several benefits, including:

- Ownership - is in the client's estate, they have to agree on data definitions up front and it removes uncertainty.

- Integration – switching vendors in and out becomes less problematic as you have one working model and various translation layers.
- Certainty – over time, model ownership synthesizes to good working practice, extensibility, and surety of operation.

When working with complex data structures, you can't set complex JSON (arrays, nested structures) into a data field bound to the payload XML (saved data to Manager) and expect it being prefilled back on resume as a JSON object value. This is not supported in Maestro. Any value stored in an XML bound data field must be of a primitive type which can be easily serialized as a string (int, float, string, boolean). If you do store a complex JSON to a bound data field, Maestro will try to recreate the JSON structure as XML nodes with elements structure. It will be still saved to Manager as form XML but when resuming a form, Maestro won't recreate the JSON object from the XML data.

To go around this, you can choose one of the following recommendations:

- Try to use repeats which mimic your JSON structure using Maestro widgets and bindings for data storage and have unbound data field as the data source for your data drive implementing calculate rules from the data in the repeat.
- Store the JSON to a bound data field as a serialized JSON string via JSON.stringify() but you would need to decentralize it back from the bound data field into another one used as the data source for you data driven widget on form load.

The Maestro editor comes with the Integration pane so you can easily configure component's data extracts and integration with Manager.

Next, learn about the unified application data document model.

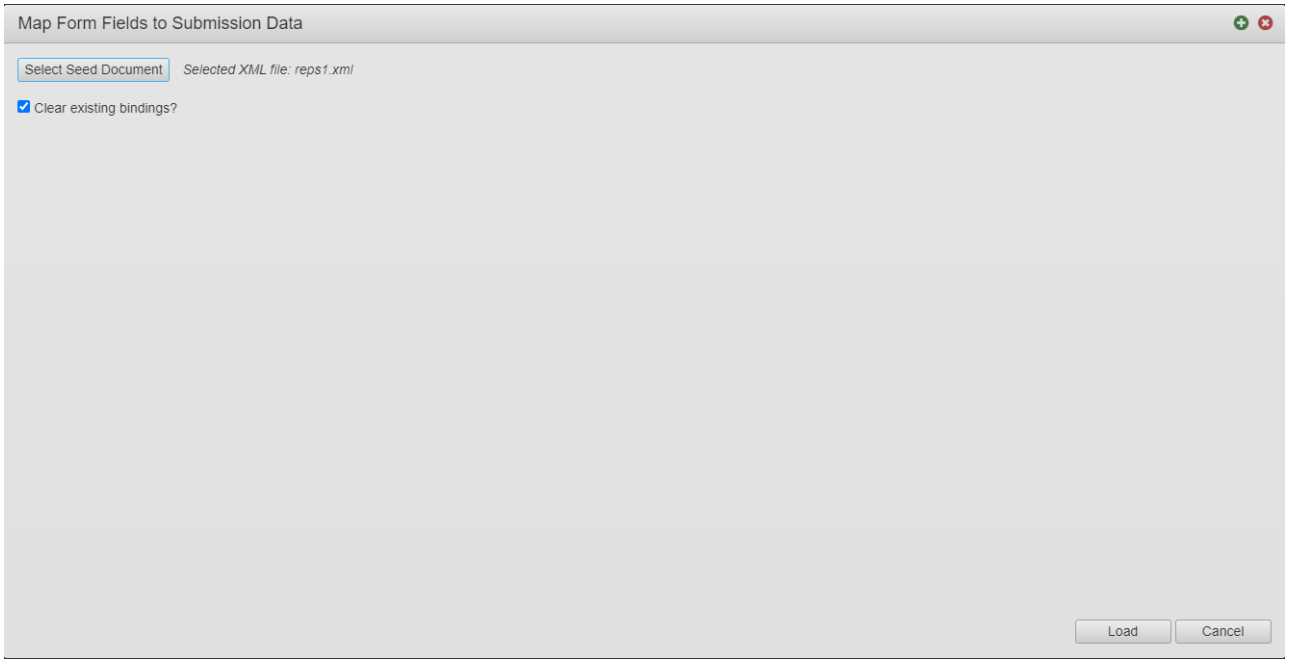
Data Mapper

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Maestro editor allows you to map an existing client schema, represented with an XML file, into a form. This is a quick and easy way to set up a form compatible the desired submission output. For more information, see form data XML.

To map form fields to submission data:

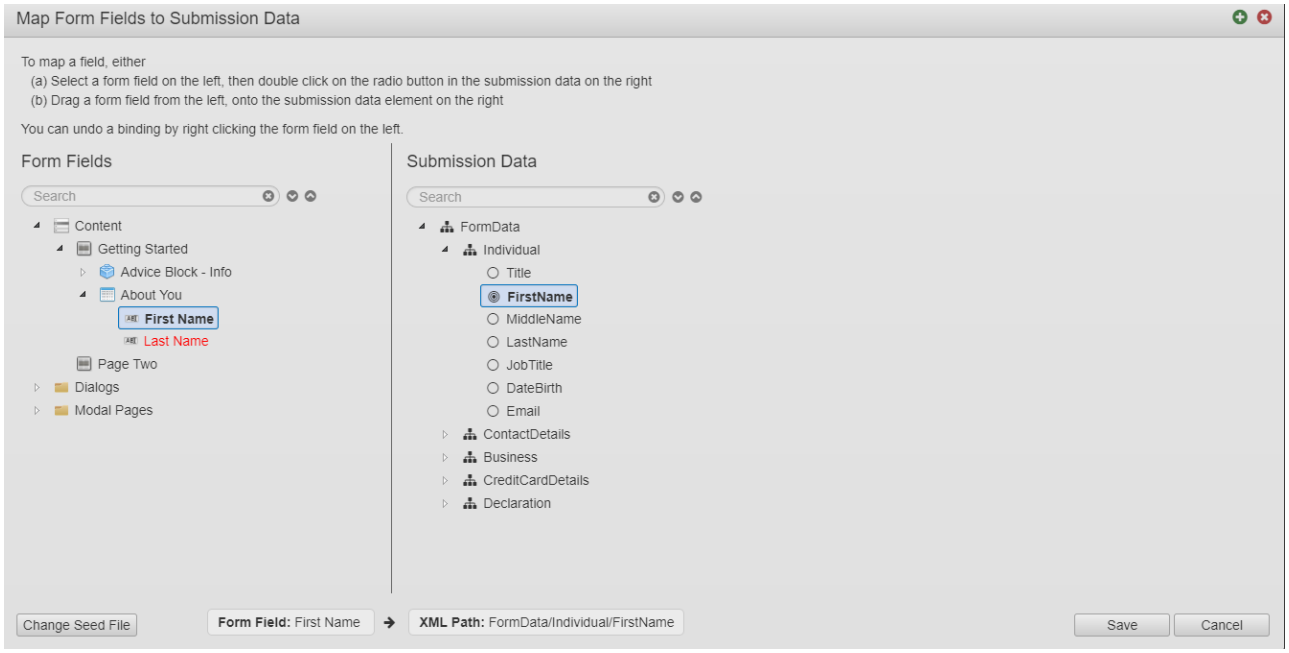
1. Open a form in the Maestro editor and click Settings Menu.
2. Select Open Data Mapper to map form fields to submission data.



3. Click Select Seed Document to choose an XML file to upload from your PC. A file content can be like one shown below:

```
<root>
	<firstName/>
	<lastName/>
</root>
```

4. Select the Clear existing bindings checkbox to clear data bindings.
5. Click Load to upload the file.



6. Map the components from the Form Fields column to the Submission Data column by dragging it across.
7. Click Save.
8. Click Build and Render Form to see how this mapping affects the form

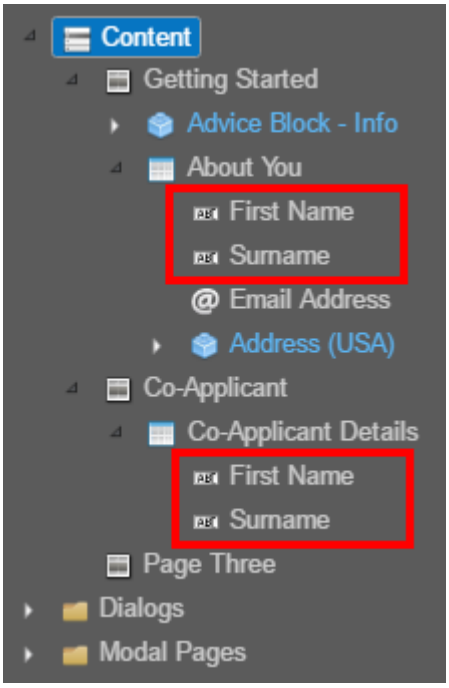
Next, learn about form XML data payload.

Form XML Data Payload

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Each Journey Maestro form has the XML Data Payload, which is the XML layer that represents the user's input data of the form and is used for external systems integration. This layer is created when a form is built. Most components are added to the XML layer by default, except for pages, sections, blocks, and data fields. For more information, see data model overview.

The default form XML data payload is a flat structure XML document, which only includes components, but not the hierarchy they belong to. This is illustrated by the following form with two fields, First Name and Surname, that appear in two different sections of the form.



By default, the XML tags created for each of these items are identical: FirstName and Surname present twice in the XML document, as shown below.

```
<AvokaSmartForm>
    <FirstName>
    </FirstName>
    <Surname>
    </Surname>
    <EmailAddress>
    </EmailAddress>
    <AddressUSA>
    </AddressUSA>
    <SFMData>
```

Only the first XML tag is created for First Name and for Surname, so this results in a duplicate bindings error.



Next, learn about component data binding.

Dynamic Data in Forms

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro forms are made up of HTML, CSS, and JavaScript, and are executed when the form is deployed in a web browser. You can think of Maestro forms as AngularJS[1] single page applications that run in a browser. Almost every form interacts with a form user who, at some stage, is required to enter some data into the form.

Data in a Maestro form representing form field content is formatted as XML. When a form is first rendered, an XML document is generated to contain the data collected by the form. This XML will eventually be populated or updated and saved when the form is submitted back to the Manager server.

Maestro forms include the concept of repeats to model real-world repeatable data, such as lists of previous addresses, lists of applicants and so on. Fields in a form can be wrapped inside repeating components, with various means of controlling the number of repeats dynamically.

A form generally needs access to some type of dynamic data, including personal data contact details, reference data lists of city names, and volatile data, such as current interest or exchange rates. This data is regarded as dynamic as its nature changes according to the information the form is designed to collect, and who is the form.

A well-designed Maestro form should make use of dynamic data that is external to the form itself. That is, you should not hard-code data into the form, but rather access it dynamically when the form is run. Keeping data outside of the form definition leads to a form that can dynamically adapt to changing circumstances. This adaptability can reduce the overall impact of any changes to the data content, or the business rules that govern the data.

A form still needs its supporting data, regardless of where it's kept, so either the form must be given all the data it needs, or the form must collect the data itself. Accordingly, dynamic data makes its way into a form via one of two methods:

---

[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/

- Prefill – data is given to the form by Manager, which writes values into the form's XML file during rendering.
- Dynamic Data Services – the form gets its own data when it is required by executing JavaScript code, which calls Manager dynamic data services.

Dynamic data can be used for several different purposes in a form, including:

- Presentation – to display information to a form user completing the form, for example, displaying terms and conditions that may vary according to circumstance.
- Calculation – to calculate real values based on volatile variables such as current interest rates, share prices, or currency exchange rates.
- Experience – to help a form user enjoy a better user experience by allowing them to pick from lists, or provide auto-completion of, for example, country or city names.
- Validation – to ensure forms contain data that is not only correctly formatted but conforms to business rules and data relationships.
- Decision-making – to determine the behavior of the form in response to user input, for example, directing users to additional pages for confirming eligibility or entitlement.

This document describes the Prefill and Dynamic Data Services methods of working with dynamic data.

Prefill Dynamic Data

Manager applies form prefills from all enabled prefill sources in the order listed in the table below.

Source	Managed By [1]	Mapped By [2]
XML Seed file (enable) [3]	Forms/Forms/form/Form Dashboard/Form Versions box/Data Config link/Form XML Data tab	Input XML Prefill Mapping
Organization properties	Forms/Organizations/organization/Properties tab	Property Prefill Mapping
Form properties	Forms/Forms/form/Form Dashboard/Form Versions box/Properties link	Property Prefill Mapping
URL request parameters	Form Dashboard/Form Versions/Data Config link/Request Param Prefill Mapping tab	Request Param Prefill Mapping
User profile properties	Security/User Accounts/User/User Profiles tab/profile name/User Properties tab	Property Prefill Mapping
Form Prefill Service	Forms/Forms/form/Form Dashboard/Form Versions box/Services link/Services tab/Form Prefill Data Service dropdown	Not required

(XML) Services/All Services/service/Groovy Script tab

NOTE

[1] - The Managed By column shows where to go in Manager to manage the prefill source content.
[2] - The Mapped By column shows where to apply mappings for that source. The destination for all prefill data is the form XML.
[3] - XML seed prefill is not enabled in a Form Space by default, it needs to be enabled on the Form Space >Properties Edit tab

Prefill data applied at higher rows, shown in the table, will be overwritten by enabled lower rows if there are multiple mappings from prefill sources to the same form field.
A Maestro form designer does not need to do anything extra or special to have a form's fields prefilled. It's just 'there' as though it was given an initial value by the form designer at design time. Manager does all the work to configure mappings and push values into the form XML without the form even being aware of what has occurred.

Form Prefill Service

Form field content in a Maestro form is formatted as XML. The initial XML of a form can be provided at render time by a custom Groovy service – a Form Prefill Service. The service to be used for a form is specified in the Form version Services tab. XML from the Form Prefill Service is directly pushed into the form XML, without the need to map values from source to destination. If the XML contains repeating elements, repeating components are dynamically created in the form.

Prefill Mapping

All other prefill sources require mapping to direct their values into the structure of the form XML. A mapping target form field is specified by an XPath expression. The source field of each mapping is specified differently according to the prefill source:

- Property – source is specified by property name.
- Input XML – source is specified by another XPath expression.
- Request Parameter – source is specified by URL request parameter name.

In the case of input XML mapping, repeating components in a form are automatically set to the number of repeated elements of the input XML, provided that the mapping is made between the parent items of the repeated content.

Property Prefill

Properties are defined at the following scopes, where each scope has slightly different rules regarding properties:

Property Types

Except for untyped form properties, all properties must be associated with a 'Property Type'. A property type defines the nature of a property but does not provide a value for one.
Property types are a combination of:

- A unique name within the Manager instance.
- A scope – Organization, Form or User.
- A data type - Boolean, HTML, Image, JSON, List, Long Text, Number or String.
- An optional organization – if not provided, the property type will be global and be available to all organizations that want to use it.

Organization Properties

An organization can define only one property of a given property type, but It is not necessary to define a property for each property type if one is not needed.

Form Properties

Each form property may be of a defined property type or may be un-typed. A form can define only one property of a given property type, but It is not necessary to define a property for each property type if one is not needed. An untyped form property must have a property name which is unique within the form, a data type - Boolean, HTML, Image, JSON, List, Long Text, Number or String, and a value.

User Profile Properties

A user profile property must be of a defined property type, or one of the pre-defined property names: Email, Family Name, Given Name, or Mobile.

Prefill Caveats

The list below identifies several of the caveats surrounding the use of form prefill.

- A form is only prefilled when it is first rendered.
- A form that is saved and retrieved is not prefilled again, even if property or seed values in Manager have changed since the initial render.
- You can prefill form fields only with string values.

- A field cannot be prefilled unless it's represented in the form XML – which means its property 'Include in submission data' must be ticked while designing the form in Maestro.

Dynamic Data Services

Unlike prefill, which happens like magic, a form must implement its own logic to fetch data from a dynamic data service. A form is in absolute control over if, when and how it calls dynamic data services. While it is executing, a Maestro form is always able to stay in communication with the server that rendered it, so it's quite easy for a form to ask for more data when it decides it needs it. Manager supports two types of dynamic data service, both of which are implemented in the Groovy language:

- Groovy Dynamic Data Services.
- Fluent Dynamic Data Services.

The difference between the two is in the way that the Groovy APIs are designed. Both choices have no impact on the way that a Maestro form calls them. 'Fluent' is a style of designing APIs. See towards the end of this document for a brief explanation. A Maestro form can call either service type, and in fact, has no knowledge of which type has been used to implement the Dynamic Data Services. When code in a Maestro form makes calls to dynamic data services, complex parameters in JSON format can be passed. Complex values may also be returned in the response, again typically in JSON format, although the exact format returned is up to the service author.
Dynamic data services are called using two different techniques:

- Dynamic Data Button – a Maestro component that can be configured to call any dynamic data service. It is a visual component you configure with point and click using the Maestro user interface but can be hidden and activated by code instead, if required.
- DynamicData.call – a Maestro API method that requires more effort to implement, and some more JavaScript programming knowledge to use. The Dynamic Data Button uses this API method to do its work behind the scenes.

Dynamic Data Button

This Maestro component provides all the capability a Maestro form needs to call dynamic data services. You configure it entirely within the Maestro user interface, where you can take advantage of input and output mapping dialogs to collect input parameters and distribute results. You can also write code in the On Success and On Failure rules that the component provides.

The Dynamic Data Button is a visual component, so it will be visible in the Maestro design mode, unless it is excluded, and will be visible at runtime unless it is hidden with a visibility rule.

DynamicData.call

This method calls a Manager dynamic data service, passing in any parameters you provide, and returns a Promise object from which you can obtain the response, so you need to know a little more about JavaScript programming to use it.

You will need to acquire the parameter values, marshal them into method parameters, call the API method, then deal with the returned Promise to retrieve the returned value when it is ready. You will then need to un-marshal the values from the response and distribute them to other components on your form. You will also need to write code to deal with errors. Unless you have a good reason, it's preferable in almost all cases to use Dynamic Data Buttons to call dynamic data services, because it reduces the overall amount of custom JavaScript code you need to write.

One case where you will need to use DynamicData.call is if the response returned from a service call contains nested properties, as the Dynamic Data Button can only map output properties at the top level of the response object. The following example show an JSON response with a nested object `bsb`, which is a child of `data` and not a child of the root, so it can't be output-mapped by a Dynamic Data Button.

Nested JSON object:

```
{
	"success": true,
	"data":{
		"bsb": "012-002",
		"bank": "ANZ"
	}
}
```

You will need to use DynamicData.call in this case, unless you can have the service changed to return a flat JSON object like:

Flat JSON object:

```
{
        "success": true,
        "bsb": "012-002",
        "bank": "ANZ"
}
```

What to fetch

A dynamic data service can return data from absolutely any source, including data held within Manager, as well as data from external sources, such as public or private web services. One source of data that can be easily accessed is organization property values. Complex JSON objects can be returned. from Is there supposed to be more here?

When to fetch dynamic data

From an efficiency aspect, it is preferable as much as possible to fetch data only once, when a form loads, in the Form Load Rule. Such data can then be retained for the lifetime of the form. However, not all data lends itself to this usage pattern.

Consider that when a new form loads, for example, nothing may be known about the person filling the form in, so you cannot yet ask for their personal data. It is important to consider that you may not want to load a list of all city names in the world until you at least know what country you need city names for. There are choices that need to be made about exactly when and how often to fetch dynamic data, which largely depend on the type of dynamic data required.

> NOTE
>
> Do not call DDS in a calculation rule.

Dynamic data can be loosely categorized as:

- Personal Data.
- Reference Data.
- Volatile Data.

How to test dynamic data

When developing a Maestro form, you can't call services for the following reasons:

1. It is likely that your form is located on one of the shared Maestro instances which will not have the service available. On the shared instances we do not install services as this

should be done on your organization Journey Manager instances.

2. Even if it was available on the same environment, previewing the form generates the form and injects it into an iFrame and only is running in your browser, it is not part of a transaction so Journey Manager would not acknowledge the service call it would not have a valid request key that Manager requires to handle a dynamic service call.

This means that you can't invoke `DynamicData.call()` in preview without adding preview data.

You can add the following script to your code that does something like:

```
if (Resource.preview) {
    Form.addPreviewDDSData("myService", {name: "Glenn", team: "Journey Maestro"})
}
```

Or

```
if (Resource.preview) {
    Form.addPreviewDDSData("myOtherService", function(params) {
        if (params.name) {
            return {message: "Hello "+params.name }
        } else {
            return {message: "Hello unnamed person"}
        }
    });
}
```

This way you can test your function calls while providing stub data as publishing the form and pushing it to your Journey Manager instance and then testing your form does have a longer round trip than using preview. For more information, see the API documentation for `Form.addPreviewDDSData(serviceName, dataOrHandler)`

Personal Data

Personal data is related to the individual(s) interacting with the form. By definition, you can only fetch personal data once you have identified the individual(s) concerned. This data could include:

- Existing policies or accounts.
- Employment history.
- Contact details.

In this case, personal data can be fetched only once enough identifying information has been captured to perform a data lookup. It should not be necessary to fetch that same data again during the form's lifetime.

Reference Data

Reference data typically consists of values or lists that don't change all that often, so you can probably safely obtain it when the form first loads and keep it for the entire form session. This type of data is typically used to allow applicants to pick from dropdown lists, to validate form content and to provide fixed values on forms, such as:

- Lists of countries.
- Vehicle models.
- Insurance or finance products.
- Phone numbers.

It is quite a common scenario to select an initial value from a reference data list of, say countries, and make further calls for more specific data such as city names. Similarly, selecting a vehicle manufacturer may require further calls for models, years, colors or related options.

Volatile Data

This is data that is constantly changing and should be called for as late as possible, just before you need use in a calculation, such as:

- Share prices.
- Exchange rates.
- Interest rates.
- Number of seats remaining for a flight.
- Account balances.

In this case, volatile data may need to be fetched more than once, immediately before use each time.

Accessing Prefill Data

It's useful to understand the data processing pipeline that Maestro goes through when populating a form (at least at a high level).

- The prefill XML (provided by prefill services in Transact Manager) is parsed and turned into an in-memory Javascript object graph with the same structure as the XML.

- An object graph called "data" is constructed, using the fields in the form to determine the structure of this object graph.
- The "data" object is initialized from the prefill object.
- The data object is further modified as calculation rules fire.

The "data" object is live, in that it is continually changing as the user interacts with the form, and calculation rules fire. The "prefill" object never changes. In some scenarios, it may be helpful to refer to the original prefill information.

We may want a display field to show the original value of a prefill value, and not change if that value is changed by the user. In this case, you can directly reference the prefill object like this: `{{Form.prefil.TestText}}`.

If you would like to see the prefill data object in your browser's debugger, type: `maestro.Form.prefill` at the console prompt.

Next, learn about component data binding.

Component Data Binding

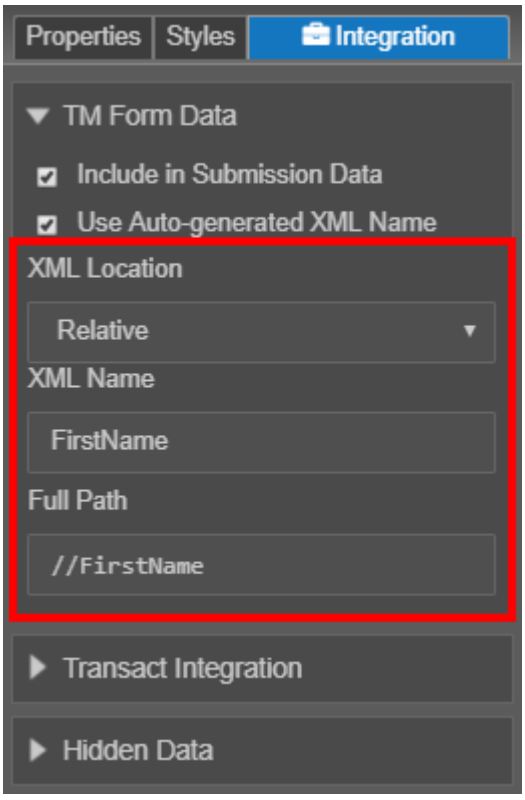
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Maestro editor allows you to link a component in a form to the XML data layer using the Integration pane. This process is known as data binding.

Relative Binding

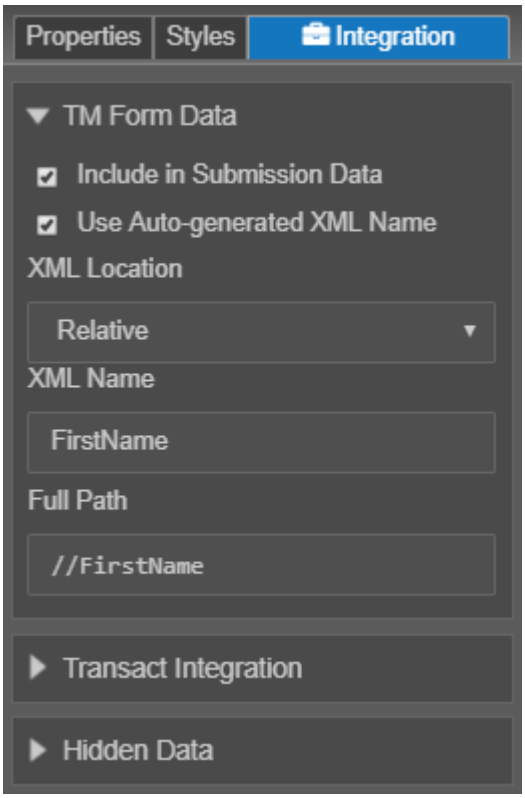
By default, most items (exceptions include pages, sections, blocks, and data fields) are bound under the root element in the XML layer, unless their parent container has been included in the submission data. This is known as Relative binding. If an item's hierarchy changes, the XML Name will automatically be updated to reflect this.

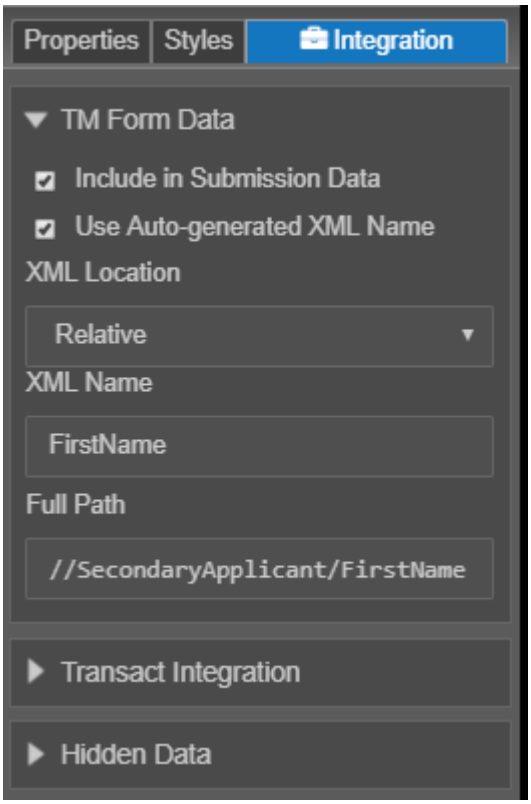
On the Integration panel, you will find the XML Location, XML Name, and Full Path. These options will only display if the component is not part of an entity.



Notice, the Full Path in the images below.

Parent not included in submission data Parent is included in submission data





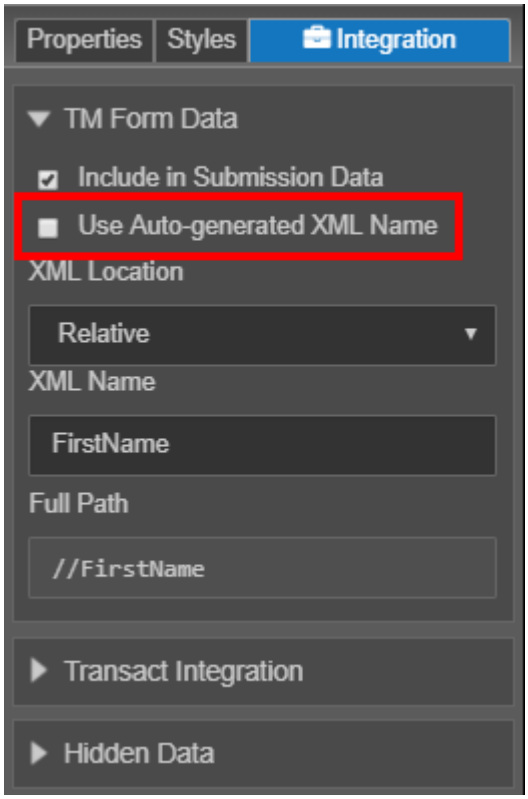
You can include components in the submission data, by selecting the component and then selecting "Include in Submission Data".

Absolute Binding

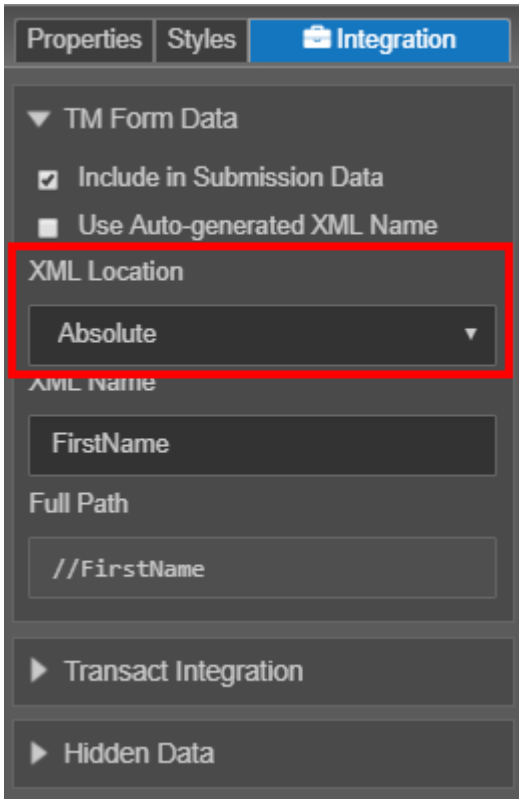
If you want to ensure that an item is always going to be bound to a particular XML tag, use "Absolute" binding.

To change to absolute binding, follow these steps:

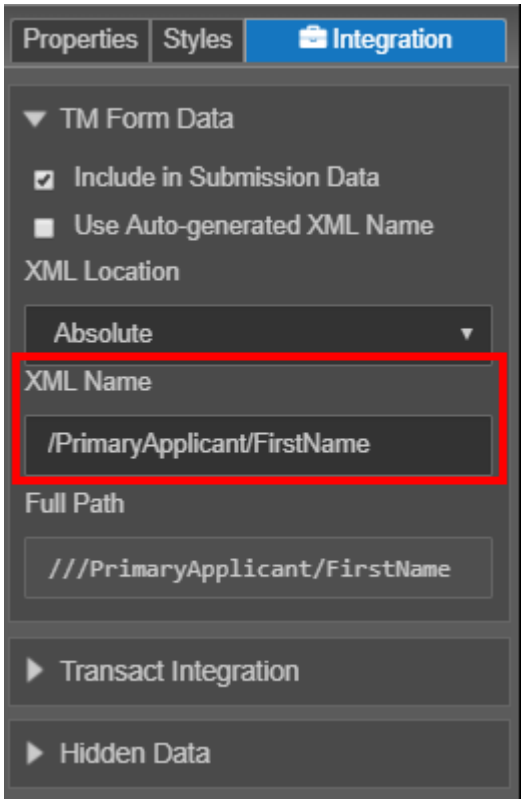
1. Select the component and switch to the Integration panel
2. Clear the Use Auto-generated XML Name checkbox to change the XML Location.



3. Select Absolute from the XML Location dropdown list.



4. Change the XML Name to include the component you want to bind the selected component to. This will update the Full Path.



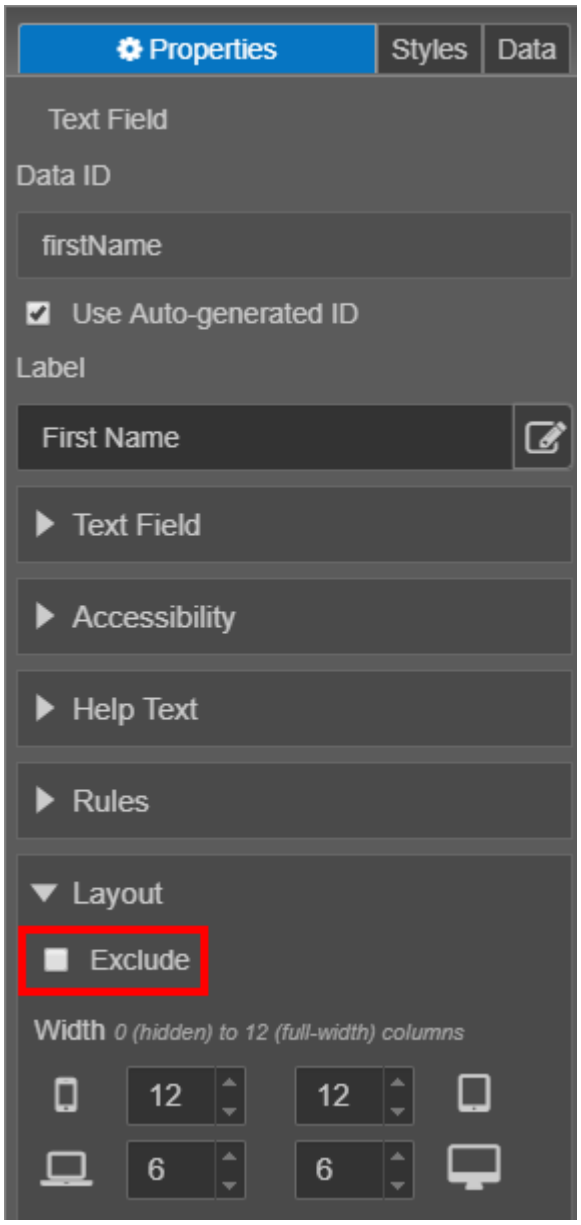
You should also use Absolute binding when you are binding to externally generated XML. It is safer to use Absolute binding to ensure the structure remains as you expect. For more information, see data model.

Exclude a Component

You can exclude a component from the form's XML if needed.

1. Select the component.
2. Switch to the Properties panel.
3. Expand Layout.

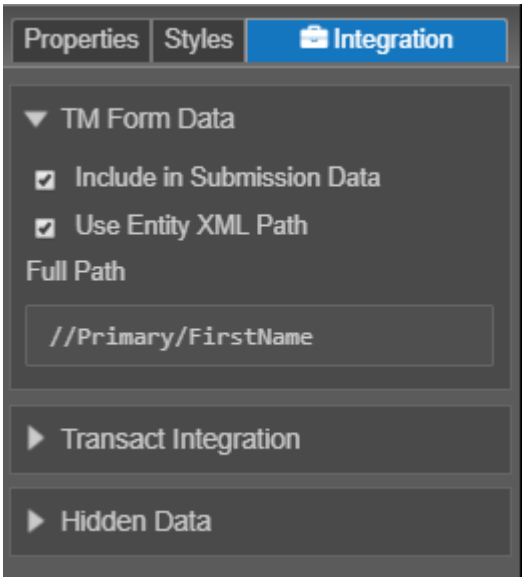
4. Select Exclude.



Components Using Entity XML Path

When a component ID contains an entity the Integration panel options will be different.

The XML Location and XML Name properties will not be available. The location will be based on the entity included in the ID property as displayed in the Full Path property. You can deselect Use Entity XML Path to enter a different Full Path.



We recommend checking Submission Data Extract

Next, learn about XML data payload.

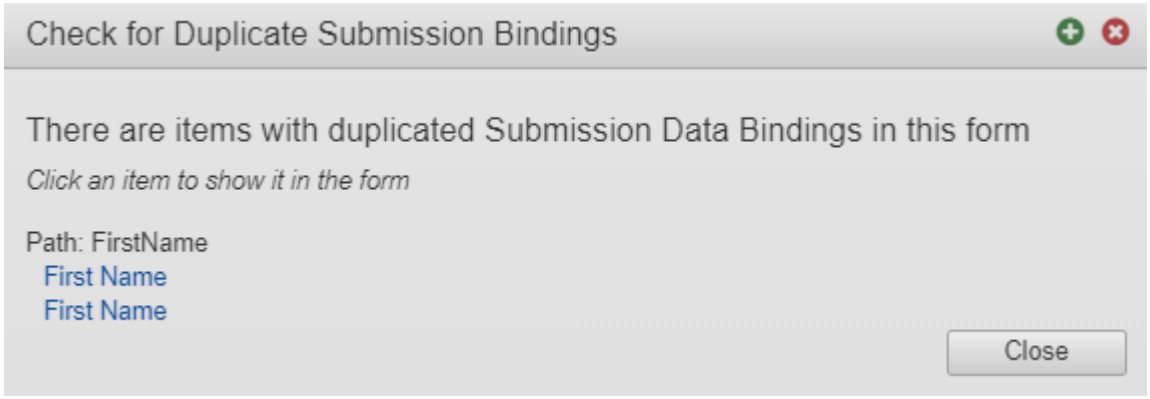
Check for Duplicate Data Bindings

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Maestro editor allows you to check for duplicate data bindings that may occur in a form you develop.

To check for duplicate data bindings:

1. Open a form in the Maestro editor and click Settings Menu.
2. Select Check for Duplicate Bindings.



3. Check the list of the components that contain a duplicate binding and resolve the issues.
4. Click Close.

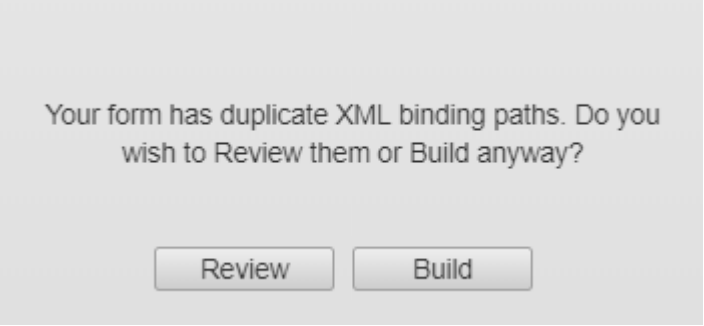
Next, learn how to resolve duplicate data bindings.

Resolve Duplicate Data Bindings

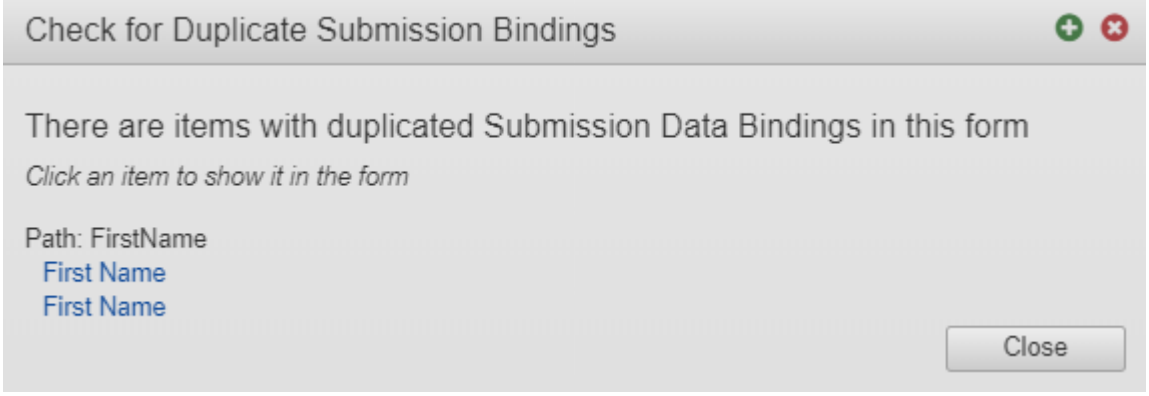
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Duplicate bindings occur when more than one component shares the same XML Name. This can be caused by using the Accept button when a duplicate ID occurs, or when the XML Name is manually updated by editing the XML Name from the Integration panel. If you enter the same XML Name as another component you will not receive an error message until you use the Check for Duplicate Binding Errors option or build the form version. For more information, see form data configurations.

When you add components to a form, Maestro will automatically add this component to the underlying XML layer. Parent containers are not automatically added so the default XML structure is flat. Therefore, you may inadvertently, or purposely, bind two components to the same location. The dialog shown below displays when Maestro detects that two components have the same XML binding location when you build the form version.



Click Review to view the components that contain the same ID.

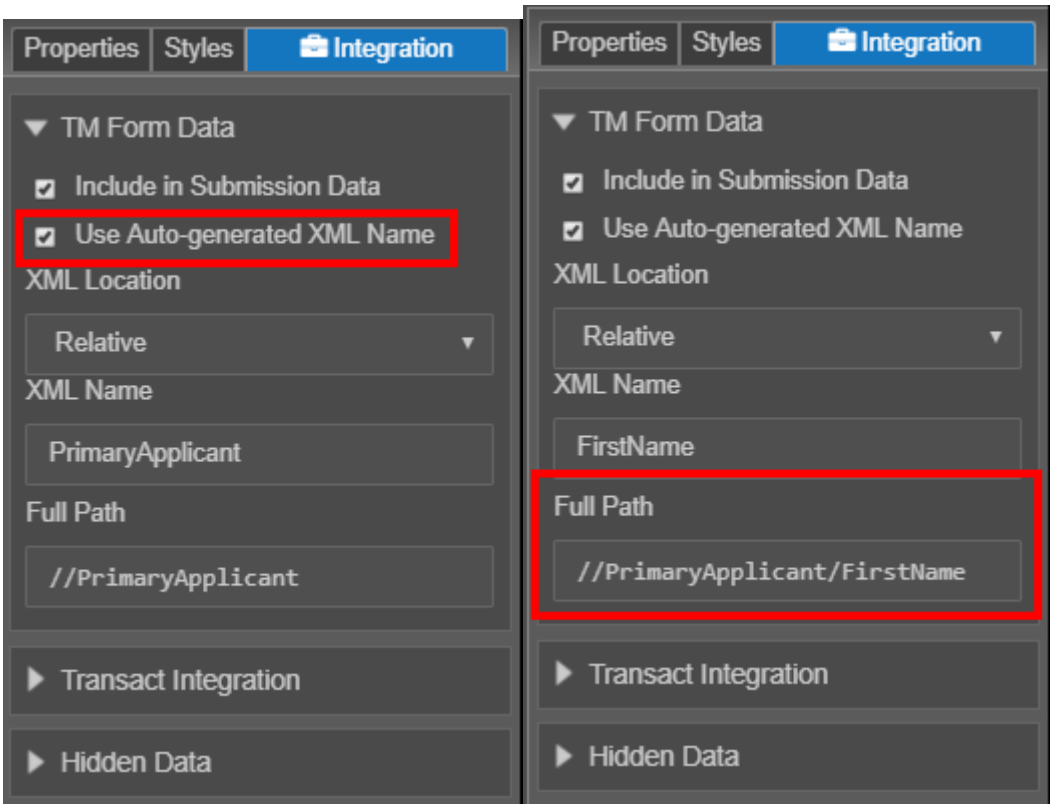


Click Build to continue building the form version. However, that can cause unexpected issues.

You can check for duplicate bindings at any time while developing the form.

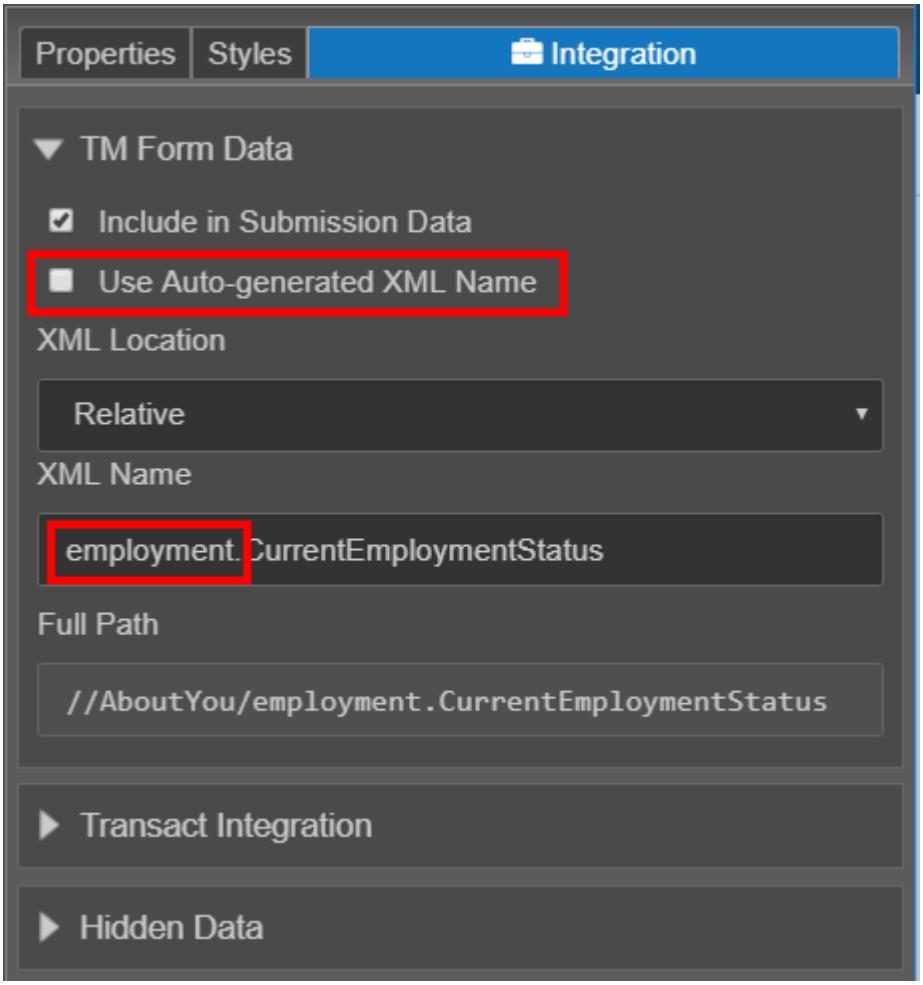
One way to resolve this issue, is to modify the component's parent's binding to ensure the components with the conflicting binding have different paths.

To achieve this, select the parent container of the component, select the Integration tab and select the "Include in Submission Data" option and ensure the XML Location option is set to Relative.



Updating the container component will update the Full Path of components within the container that are set to a Related XML Location.

Another option is to select the component and manually update its XML Name property to include an additional parent level.



You may use either a dot (.) as above or a backslash (/) to create the additional parent in the component's hierarchy.

User Mobile Device Detection

MaestroThe UI design product. | Form Builder Template Designer | All versions This feature is related to all versions.

When building adaptive UX in Maestro you may run into a requirement to present different content to users depending on whether they are on a mobile or desktop client. For example, it is unlikely that desktop users will utilize their camera to snap a photo of their driver's license so you may want to down-play this capability for desktop users and promote it for mobile users.

Unfortunately there is no standard way to determine if a user is on a mobile device and a Google search will uncover strong debate over the best way to achieve this. This example provides one simple option that searches the user agent string for a set of known mobile or tablet identifiers.

Let's look at the example of 'Is Mobile' Detected

The following JavaScript snippet demonstrates a simple example on how to detect if the user is on a mobile (or tablet) device. This will not give 100% coverage but should cover most of the commonly encountered devices.

NOTE

var isMobile = navigator.userAgent.match(/(iPhone|iPod|iPad|Android-|webOS|BlackBerry|IEMobile|Opera Mini)/i)

You could add this code directly into your rule scripts, or create a reusable JavaScript library that you can distribute among your teams.

For more fine-grained detects you may want to review the Modernizr library which provides a comprehensive analysis of the HTML, CSS and JavaScript features the user's browser has to offer https://modernizr.com/

To using Modernizr in Maestro:

1. Build your custom modernizr-custom.js file on their website and download it.
2. Add a JavaScript library to your form and upload your Modernizr JS file.
3. Now in your rule scripts you can use the Modernizr detects, for example, `if (Modernizr.touchevents)`

Components Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with a wide range of components, also known as widgets, that you use to build forms. Components are the modular building blocks that can be dragged and dropped into all types of designs, including forms, receipts, templates, and even other components. Some components are simple elements, such as the Text Field component, while others provide more complex functionality, such as the LinkedIn Auto-Fill component.

You can access all available components from the Palette pane of the Maestro editor where you can create, add, edit and delete them. Each component has a set of properties, rules and styles allowing you to configure it to your requirements. A component inherits some of properties, rules and stiles from a parent component as well as it has its own configuration set, which gives you extra flexibility in building highly reusable components and templates.

Maestro distinguishes between the following types of components:

- Core (Native) components
- Custom shared components
- Custom native components
- Exchange components

Now, let's look at each component type in detail.

Core (Native) Components

Core components, also known as Native components, are fundamental components and are shipped with Maestro software so you can use them out of the box. They are the low level components and include HTML, CSS and JS. Native components can be customized and combined to create shared components or added directly into forms, templates, and receipts.

Native components are the basic widgets that are supplied with Maestro. These include components for entering a single data value, such as data entry fields, checkboxes, radio buttons, and dropdown lists. There are also a number of more sophisticated components such as numeric and number fields, and more. Many components have at least two sub-parts, the label and the data entry area. Some also include an icon (such as a currency indicator), and most will allow for an optional help icon to display help text. A date field is arguably a more complex component, because it includes a pop-up calendar control - but is still really just a simple component.Under the covers, a simple component consists of an HTML definition, a Maestro

descriptor that tells Maestro what properties to display in the Properties tab, and optional JavaScript and CSS. However, as a Maestro user, you are shielded from the underlying complexity, and simply drag and drop the component into your form. These components are usually shipped in the product, although they can be built by anyone with HTML, CSS and JavaScript skills.

NOTE

We recommend getting familiar with the complete list of components so you know what is available to you while creating a form.

KEYBOARD SHORTCUT

Most of native component have a keyboard shortcut assigned to them.

Custom Shared Components

Custom shared components are the components that you can create using the Maestro editor to implement your business requirements. Usually, you use existing core components, create specific business rules for them, bundle them as a resource, and finally publish it. This makes the resource shared so you can use it across your forms. Shared components are not native components, so they have some limitations. Shared components are the components that users can build in the editor when they create a component that can use native components or other shared components.

In other words, shared components are combinations of existing components. You can, for instance, create a "Full Name" component that contains two Text Field components -- one for the first name and one for the last name. This component would be visible via the Palette pane, configurable via the Properties pane, and completely reusable -- just like any of the components included with Maestro.

Custom Native Components

Custom native components are the components that you or somebody else can create, package and import as a Maestro library. These components are an advanced type of components that are created outside of the Maestro editor Custom Native components should only be created when absolutely necessary. In most cases, a custom shared component will meet your business requirements and should be used prior to creating custom native components. Custom native

components are native components built outside the product team and are usually client specific, but could be, for example, Exchange components.

You can embed non-native components in your form, like address, you can add it and then right click on it and select embed and it will remove the component reference and add the properties, etc to the definition used in the form.

NOTE

As with all design types, components support versioning. To learn more, refer to Versions.

Maestro distinguishes two types of complex, composite components - Dialogs and Modal Pages - which are designed to perform specific actions:

- Dialogs: display to or request some action from a form user
- Modal Pages: represents a start or an end point where a form user must perform a task, an action, or make a decision

Next, learn how to add a component to a form.

Component Common Properties

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

A component can have many configurable properties, which you can adjust to control its behavior. Different components have different sets of properties, which you can check here. However, all components have the following common properties:

- Name
- XML Name
- ID
- Label
- Rules
- Layout

Let's take a look at each of these common properties.

Name

Each field also has a name. The name is usually the same as the caption. The field's name is displayed in the Structure tab, as well as at the top of the Properties tab. The name is usually the same as the label. The only time you might be aware of the name is when a field's label is very long. In this case, Maestro will give the field a truncated name. The example of Name is First Name.

XML Name

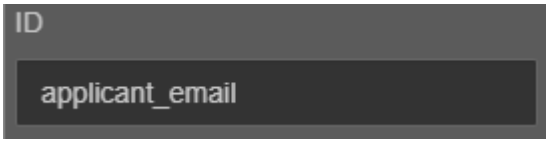
The XML name is used by Maestro to construct the XML payload for the form. The XML name will be used as the element name for this field in the XML structure. By default the XML Name is calculated from the Name. It has spaces removed, and will capitalize the individual words within the name. This naming convention more closely follows what many people use in XML Schemas. If you use the Data Mapper to bind fields to a seed file, then the XML Name will be updated accordingly. The example of XML Name is FirstName

ID

The ID is a unique identifier for the component. This is an internal name, which is used in calculations. Every widget also has an ID. By default the ID is calculated from the Name. It has spaces and other prohibited JavaScript characters replaced with underscores, and truncated.

You can override the default generated value and use your own, more meaningful name. The example of ID is `first_name`.

You can change the ID as required.

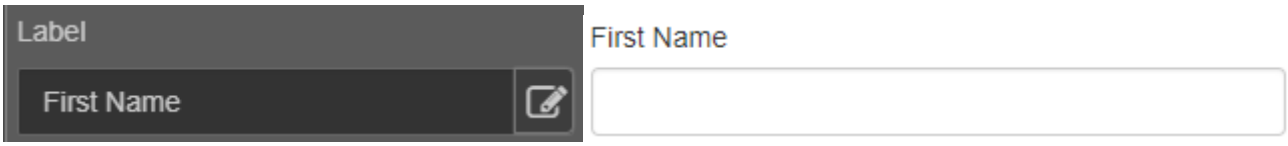


NOTE

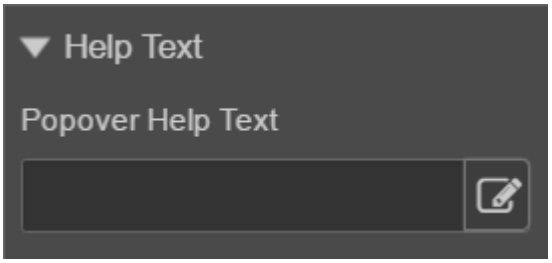
You are strongly recommended to override the default values of both the ID and XML Name of any fields which have a long or non-meaningful labels. This will assist greatly when you are writing calculations or working with the data.

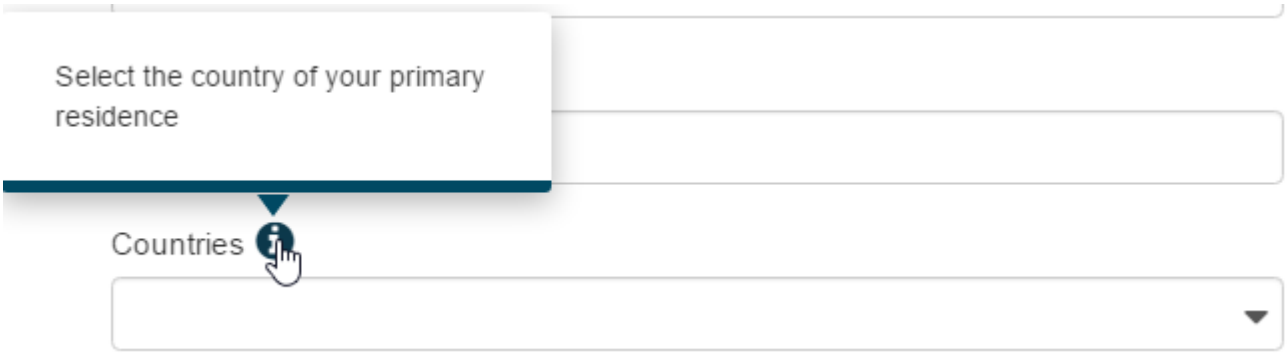
Label

The first property of every component is the Label. It is displayed to the user on the form, but you may choose to leave the label empty. The subsequent properties vary depending on what type of component is selected. The example of Label is First Name.



Help Text > Popover Help Text: When you enter text in the Popover Help Text, an information icon will display next to the component. The form user will be able to select the icon or roll their mouse over the icon to display the entered help text.

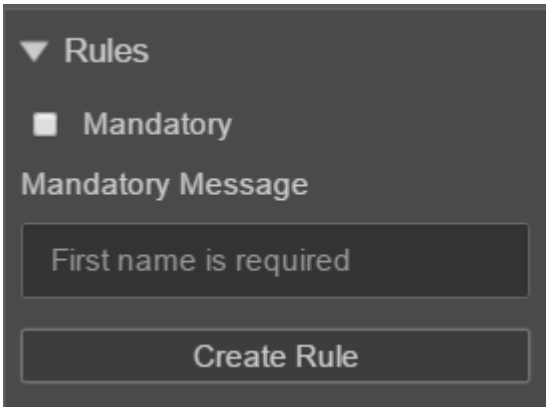




Some properties, such as help text, labels, and text display, can contain rich text. The Maestro editor comes with the rich test editor which you can use to create rich textual content including various styles and images. The editor has the Icon Picker functionality allowing you to choose an icon for the label. | 21.05 This feature was introduced in 21.05.

Rules

The Rules section is where you can mark components as mandatory and create additional rules such as visibility, validation, calculation, or editability.

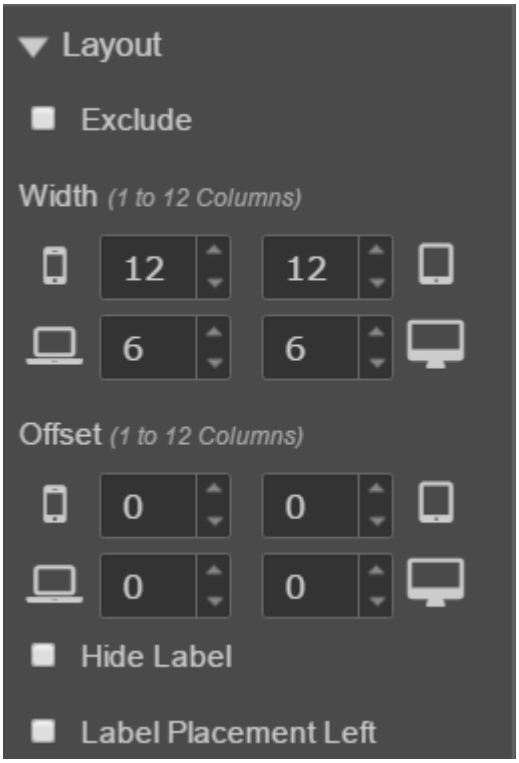


Under the **Rules** heading, properties are available for affecting the rules assigned to the component:

- Animate Visibility - this option will only be available if a Visibility rule exists on the component.
- Create Rule - open the Create Rule dialog to assign a rule to the component.

Layout

The Layout section is where you can change the position and width of a selected component.



Under the **Layout** heading, properties are available for affecting the appearance of the component:

- Exclude allows you to remove the component from the form, without deleting it. It removes the component from the form without removing it from the View pane. This is useful for disabling parts of a design without deleting them.
- Width allows you to set the width of the component based on the breakpoints, or screen size of the device used to view a form. It adjusts the number of columns available to a component.
- Offset allows you to set the horizontal position of the component based on the breakpoints of the device used to view a form. It adds empty columns to the left of the component.
- Hide Label allows you to hide the label of the component on the form.
- Label Placement Left moves the label to the left of the selected component. The default position of label components is above the selected component. Label Placement Left

The Properties pane of the Maestro editor allows you to view and edit a component's properties.

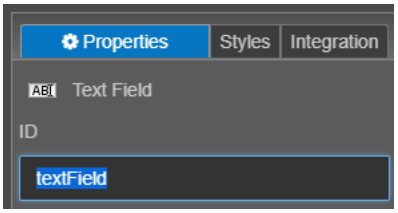
Next, learn about component form dependencies.

Component ID Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Every component on a form has a unique identifier, also known a component ID, which you can use to refer to the component. This ID is initialized when you create a component, but you can changed it at any time in the ID field on the Properties panel.

When you add a new component to a form, the ID property is assigned a default value and is highlighted automatically so that you can start typing into the field. The ID's default value is based on the component type; for example, the default ID for a Text Field is "textField".



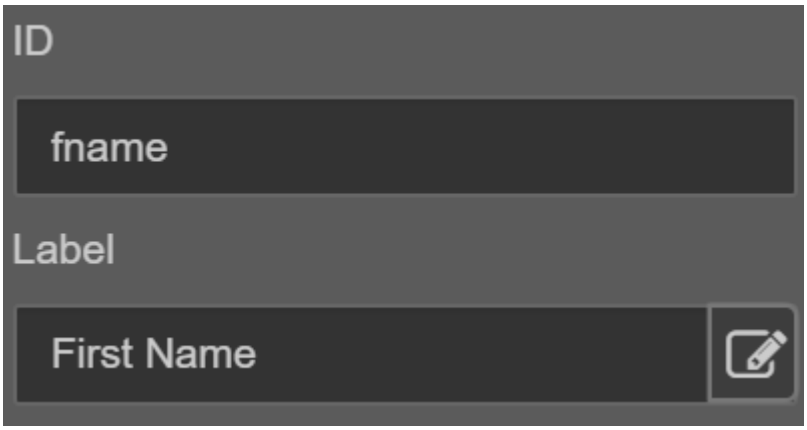
You can change the component ID from this default value at any time. Changes to ID are reflected in other properties.

- The first time you change the ID property, the Label property is updated based on the ID. The value for Label is the ID with spaces removed and converted to camel-case. For example, if you enter "firstName" as the ID of a new Text Field, Label is updated to display "First Name". Any subsequent change to ID does not update Label. However, you can change the Label property directly if needed.
- Every time you update the ID property, the XML Name and/or Full Path properties on the Integration tab are also updated to match the ID.

The screenshot below shows a partial view of the Properties pane with a new component added to a form. ID was initialized with the value "firstName", causing Label to be updated automatically with the value "First Name".



The screenshot below shows that Label remains as “First Name” when ID is updated to “fname”.



The ID property is also linked to the XML Name and Full Path properties. When ID changes, XML Name and/or Full Path are updated if Use Auto-generated XML Name or Use Entity XML Path on the Integration tab is selected. If these options are not selected, the XML Name and/or Full Path will not be updated automatically when ID changes but can be changed directly as required.

As you type a value into the component ID property field, Maestro checks whether there is an existing component with the same name and reports a duplicate ID if one is found. In this case, you have to change the component ID.

You can use entities in the component IDs.

Next, learn about component XML Name and Entity XML Path.

Component Identity Properties

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Every Journey Maestro component has the following properties that uniquely identifies it so the component can be refered to from within the form.

- ID – Every component has an ID. The ID is an internal name, which is used in calculations. The ID can also include an Entity which will allow you to group the structure of the related component. The ID is used to create the Label and the XML Name. The default value of the label is always the same as the name of the field type - but this is almost always immediately updated by the form designer. It is recommended that you always change the ID as soon as the new component is added to the form.The ID cannot include spaces and other prohibited JavaScript characters. If these characters are used, the ID will be truncated.Examples are: `firstName` and `PrimaryApplicant_firstName`.
- Label – The label is a human-readable name used to represent the component.The default value of the label is based on the ID entered. You can change the Label so that it is not related to the ID. The label usually appears above the component when displayed in the form. For a more complex element such as a header, the label may be just one of several pieces of text that are contained in the component.Examples are: First name and My Postal Address is the same as my Residential Address.
- Name – Each component also has a name. The name is usually the same as the caption. The field's name is displayed in the Structure tab, as well as at the top of the Properties tab. The name is usually the same as the label. The only time you might be aware of the name is when a component's label is very long. In this case, Maestro will give the component a truncated name. Examples are: First name and My Postal Address.
- XML Name – The XML Name is used by Maestro to construct the XML payload for the form. The XML Name will be used as the element name for this component in the XML structure.By default, the XML Name is calculated from the ID. It has spaces removed and will capitalize the individual words within the name. This naming convention more closely follows what many people use in XML Schemas. If you use the Data Mapper to bind fields to a seed file, then the XML Name will be updated accordingly. Examples are: FirstName and MyPostalAddressIsTheSameAs.

NOTE

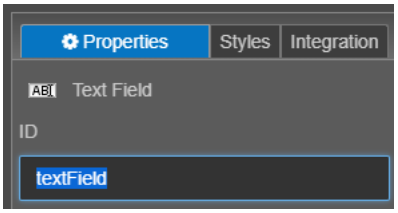
It’s recommended to override the default values of both the ID and XML Name of any component with long or non-meaningful labels. This assists greatly in writing calculations or working with the component data.

Change a Component ID

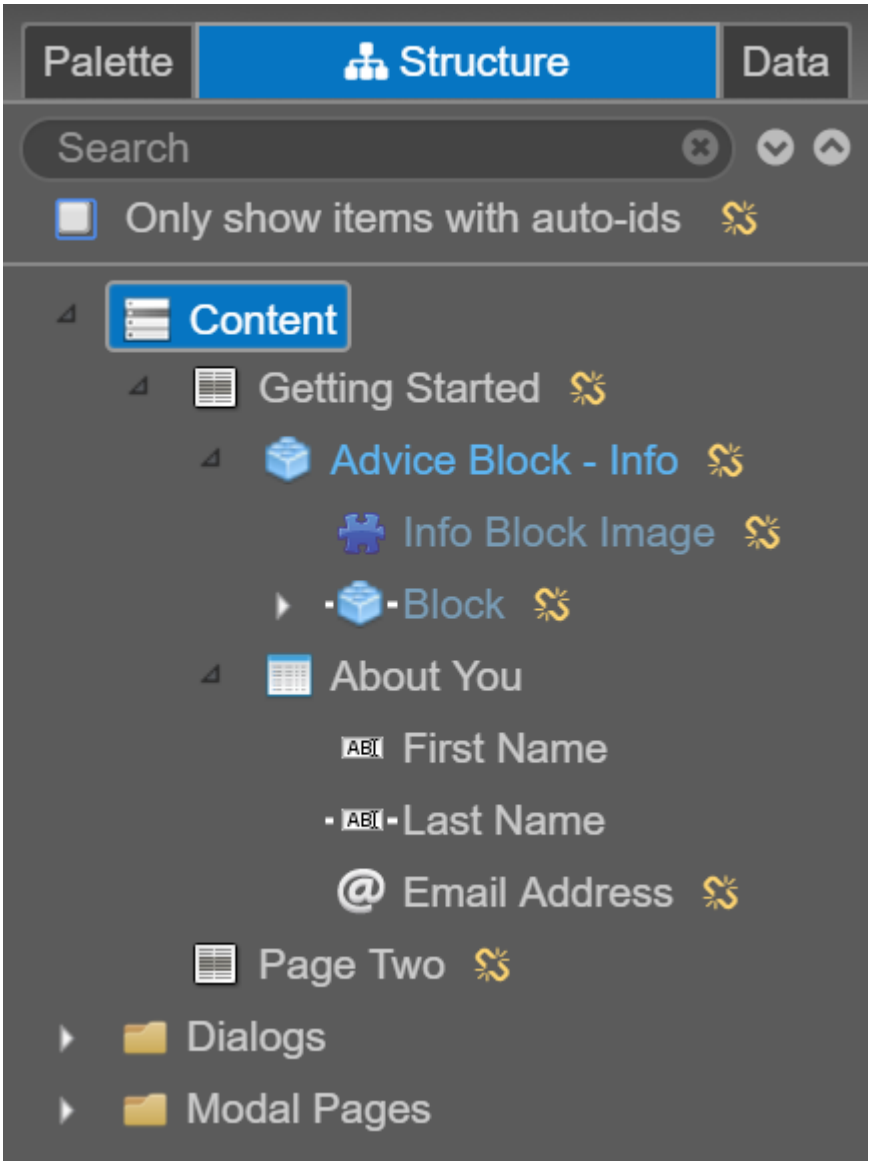
MaestroThe UI design product. | Form Builder | 17.10 This feature was updated in 17.10.

Every component in a form must have a unique identifier, also known a component ID, which you can use to refer to the component. This ID is initialized when you create a component, but you can change it at any time using the ID field in the Properties panel.

When you add a new component to a form, the ID property is assigned a default value and is highlighted automatically so that you can start typing into the field. The ID's default value is based on the component type. For example, the default ID for a Text Field is `textField`.



Maestro comes with the Auto-ID functionality that automatically generates a component ID and assigns it to the new component when you add it to a form. | 17.10 This feature was introduced in 17.10. On the Structure pane, a link icon appears next to each component that has an auto-matically generated ID, or Auto-ID.



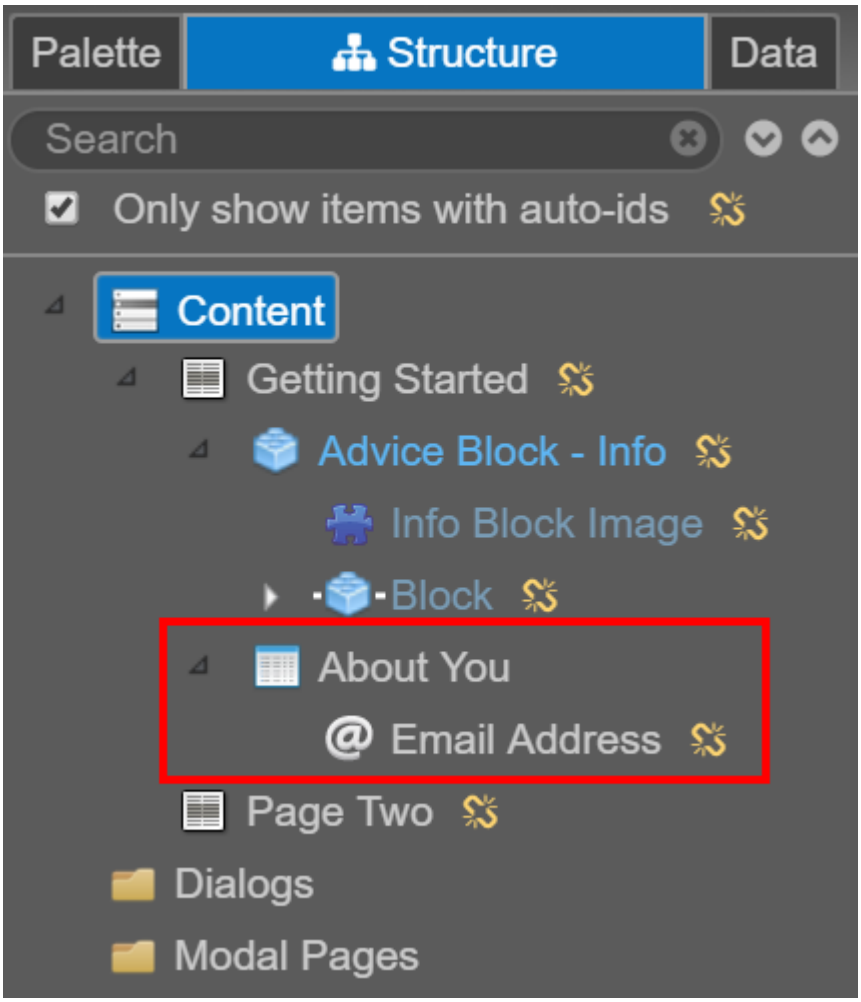
NOTE

It is recommended to limit the use of auto-ids on a form.

You may need to create a form quickly and choose to save time now by letting Maestro generate auto-ids that you can update later with better IDs. Being able to view only the components on a form with auto-ids allows you to find all components that need their IDs updated. You can select the Structure pane option Only show items with auto-ids to show only components that use an auto-generated ID name.

If a container is not using an auto-generated ID, but at least one of the child components is, you will still see the container listed in the Structure panel when Only show items with auto-ids is selected.

The example below shows the About You section when Only show items with auto-ids is selected. Even though About You does not have an auto-id, the child component Email Address does, so the About You section is also displayed.



Change Component ID

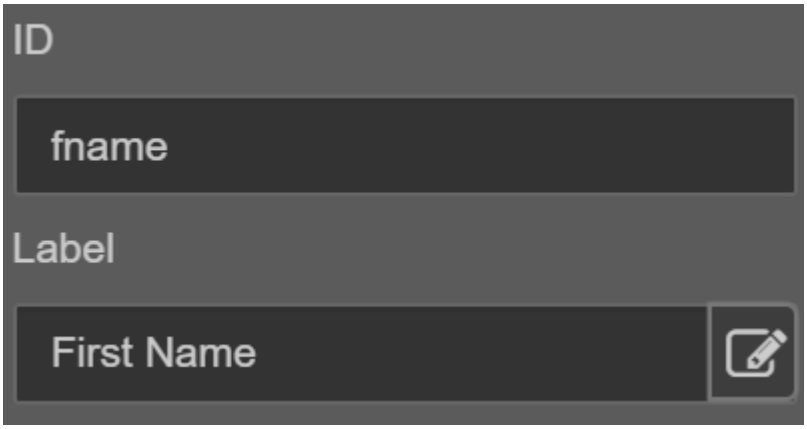
You can change the component ID from this default value at any time. Changes to ID are reflected in other properties.

- The first time you change the ID property, the Label property is updated based on the ID. The value for Label is the ID with spaces removed and converted to camel-case. For example, if you enter “firstName” as the ID of a new Text Field, Label is updated to display “First Name”. Any subsequent change to ID does not update Label. However, you can change the Label property directly if needed.
- Every time you update the ID property, the XML Name and/or Full Path properties on the Integration tab are also updated to match the ID.

The example below shows a partial view of the Properties pane with a new component added to a form. ID was initialized with the value `firstName`, causing Label to be updated automatically with the value `First Name`.

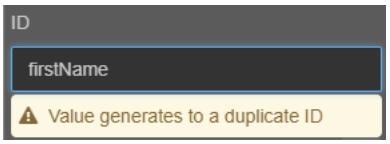


The following example shows that Label remains as `First Name` when ID is updated to `fname`.



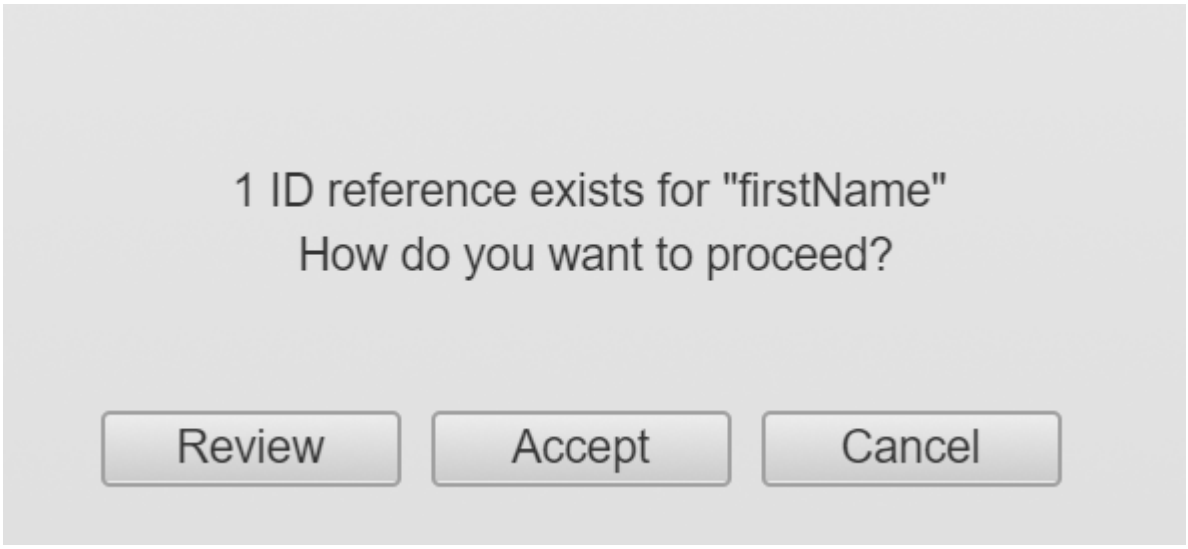
The ID property is also linked to the XML Name and Full Path properties. When ID changes, XML Name and/or Full Path are updated if Use Auto-generated XML Name or Use Entity XML Path on the Integration tab is selected. If these options are not selected, the XML Name and/or Full Path will not be updated automatically when ID changes but can be changed directly as required.

As you type a value into the component ID property field, Maestro checks whether there is an existing component with the same name and reports a duplicate ID if one is found.



When you leave the component ID property field, Maestro checks whether there are any references to the existing ID. If any such references are found, a dialog box is displayed with options to review or accept your change.

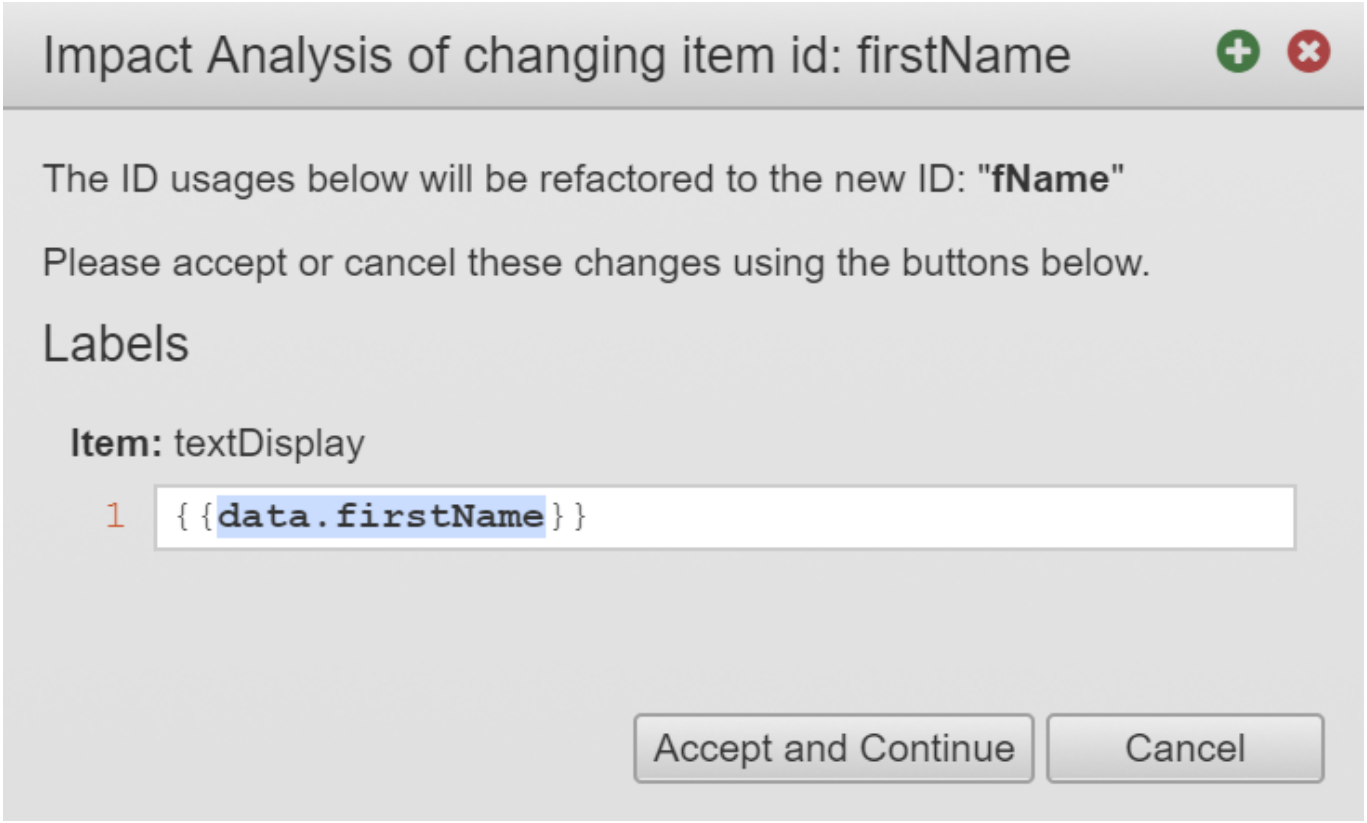




You can do one of the followings:

- Click Review to view a brief analysis of the impact of changing the component ID.
- Click Accept to change the component ID and update all references.
- Click Cancel to close the dialog and leave the component ID unchanged.

If you decide to review your change, the Impact Analysis dialog is displayed showing a list of references to the original component ID prior to the change. The Impact Analysis dialog provides options to accept or cancel your change.



You can do one of the followings:

- Click Accept and Continue to change the ID and refactor all the references listed.
- Click Cancel to close the Impact Analysis dialog and return to the Maestro Editor without changing the ID.

Component Labels

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Maestro requires every component to have a Label property containing a human-readable name used to represent the field, which will give form users an indication of what this UI element is for and what user's input it requires. You can change the labels at any time as needed. However, we recommend choosing easily understandable names at the design time.

The default value of the label is always the same as the name of the field type - but this is almost always immediately updated by the form designer. The label usually appears at the top or left of a field. In some cases, a field may only have a label, for example, the Text Display field, or it may not have a label. For a more complex element such as a header, the label may be just one of several pieces of text that are contained in the field.

Maestro comes with two types of component's labels:

- Static
- Dynamic

Static Component Label

The static label is the basic unformatted text label, where the formatting is done by the default form styles. Most labels are static labels. You have to configure a static component label in the Properties pane manually and this configuration can't be changed dynamically, for example with a JavaScript.

To change a static label:

1. Open a form in the Maestro editor.
2. Select a component in the form and click the Properties pane.
3. Change the label name in the field or click Edit to type a longer label's name or include some formatting options using he rich text editor.



4. Click Save.

The text within this property appears within the component and is used to identify the data that the applicant needs to enter into the component. For example, this Text Field component has a label of "First Name":



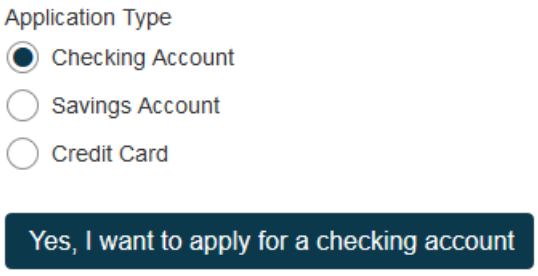
Dynamic Component Label

A dynamic component label allows its text to be dynamically changed at any time, often with a JavaScript. However, a form developer can directly apply custom HTML, or AngularJS expressions in the label to derive the text from other fields in the form. If a dynamic label contains a JavaScript expression, it is evaluated as the applicant interacts with a form, so the label can change accordingly.

For example, a dynamic label for a Button component might be defined with the following content:

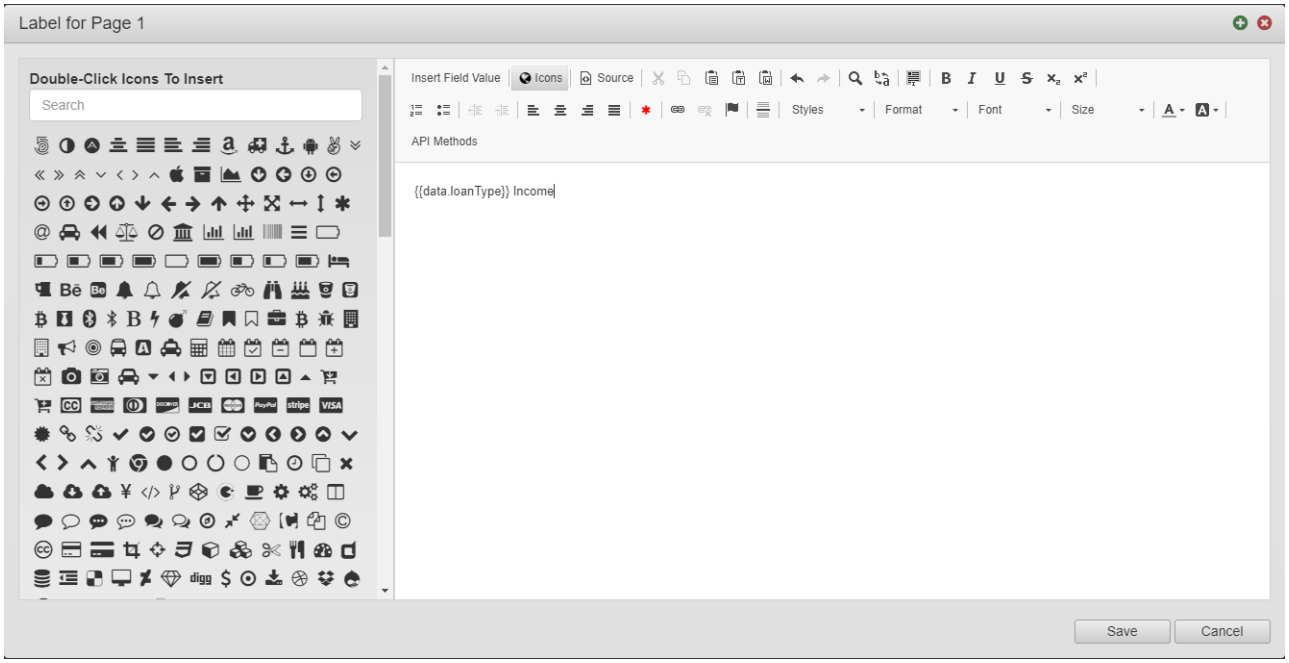
```
Yes, I want to apply for a {{ data.applicationType }}
```

This label refers to the data of a "applicationType" component. As the data of that component changes, the label of the button changes.



To change a dynamic label:

1. Open a form in the Maestro editor.
2. Select a component in the form and click the Properties pane.
3. Click Edit to access the rich text editor where you can reference other component's values to make this label name change dynamically.



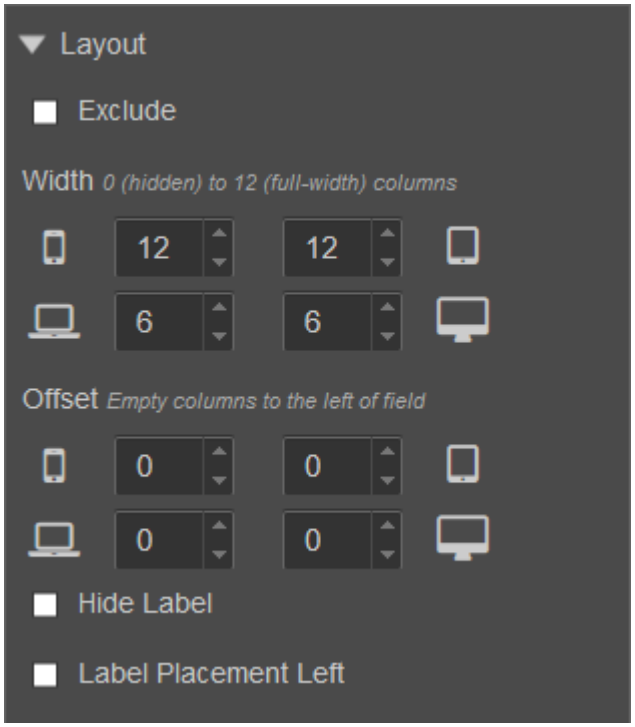
4. Click Icons to select an icon using the Icon Picker and insert it into the component. You can use search to find an icon you need. | 21.05 This feature was introduced in 21.05.
5. Click Save.

Hide a Label

Most components should have a label. There are, however, times where you might like to hide the label from the applicant.

To hide a label:

1. Select a component via the **View** pane or **Wireframe**.
2. Inside the **Properties** pane, expand the **Layout** properties.



3. Enable the **Hide Label** option.

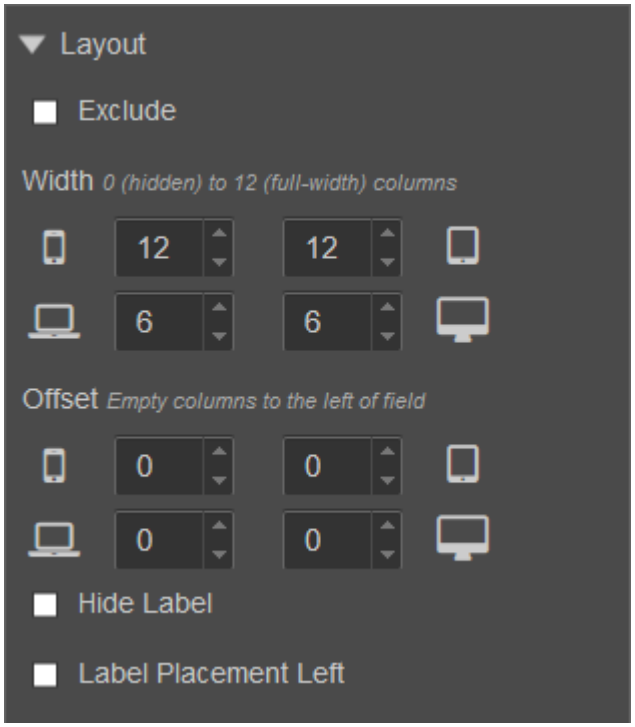
To show a label, disable the **Hide Label** option.

Change Label Placement

For the sake of usability, labels should generally be placed above input fields. This is how Maestro's components are configured by default. You can, however, change the placement of a label to the left of an input field. This is typically done on receipts as a way of saving space.

To change the placement of the label:

1. Select a component via the **View** pane or **Wireframe**.
2. Inside the **Properties** pane, expand the **Layout** properties.



3. Enable the **Label Placement Left** option.

Next, learn about a component layout.

Component Layout

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A layout is a way to arrange Journey Maestro components on a form, so they are laid out according to device or browser screen sizes.

The layout allows you to configure the following aspects of components on the form.

- The width of each component.
- The location of components relative to each other, for example, a field appears to the side or below a field that precedes it.
- The adjustment of both the component and placement of fields when the overall width of the screen changes.

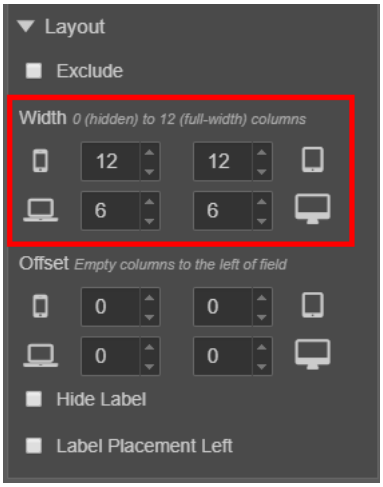
Configure a Component Layout

A good component layout is extremely important for creating a good user journey. A layout can be the difference between a positive or a negative user experience, so it's crucial to choose the most appropriate component layout for each of your forms.

You can use the following powerful techniques for creating good component layouts:

- The form itself defines three responsive breakpoints, creating four standard screen sizes. Since modern devices come in many sizes, and because devices can also be rotated into either portrait or landscape mode, these standard screen sizes are only roughly indicative of actual devices. However, the four screen sizes equate to: SmartPhone, Tablet, Small Desktop Screen and Large Desktop Screen.
- Each container in the form is broken down into a 12-column grid. 12 is a useful number because it can be divided in half, thirds, quarters, and in other ways, making it flexible but simple. (It's no coincidence that many historical measurement systems use twelve as a base.)
- Each field can be configured to use any number of columns, from 1 to 12. This determines the field's width.
- You can continue placing additional fields to the right of other fields. However, as soon as the total column count of all the fields on a row exceeds 12, that field and any subsequent fields will be moved to the next line.

The layout configuration is shown below.



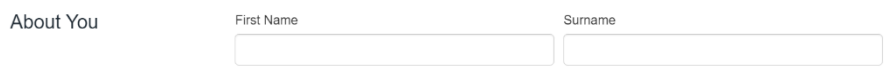
In this example, a text field takes up half the available space on larger devices. However, on smaller devices, the user is likely having issues reading the field labels or being able to enter data into the small field areas. Hence the default size on smaller devices is to take up all the space on a row. This is also why the default location of the field's label is always on the top of the field – this ensures that the label and field don't spread out too much on narrower devices.

The breakpoints for screen sizes are determined at the template level. These are set to appropriate defaults which usually cover the standard screen sizes that are available in the market today, but your Template Designer may change them if desired.

In several of the Avoka-approved templates, the breakpoint between a small screen and medium is 400 pixels (px), between medium and large is 560px and large to full screen is 780px. This means that any screen size under 400 pixels is considered a small screen, screen sizes between 400px and 560 is medium, 560px to 780px is large and anything over 780 is an extra-large screen.

The screen size will always equate to 12 columns, no matter the actual pixel width. So 400px equals 12 columns, 560px equals 12 columns, and so on. The only thing that changes is the width of the columns. Dividing 12 columns across 560 pixels would mean that the columns are wider than when you divide 12 columns across 400 pixels.

Basically, each row in your form contains 12 columns. When designing a form, you can decide how the 12 columns are divided. To get two equal text fields on one line you would need to change both text fields to 6 columns: 6 + 6 = 12.



If you don’t want them to be equal, you could set one to 7 columns and the other to 5 columns: 7 + 5 = 12.

Another possibility with the 12-column grid is to use multiple blocks on a row to give more flexibility.

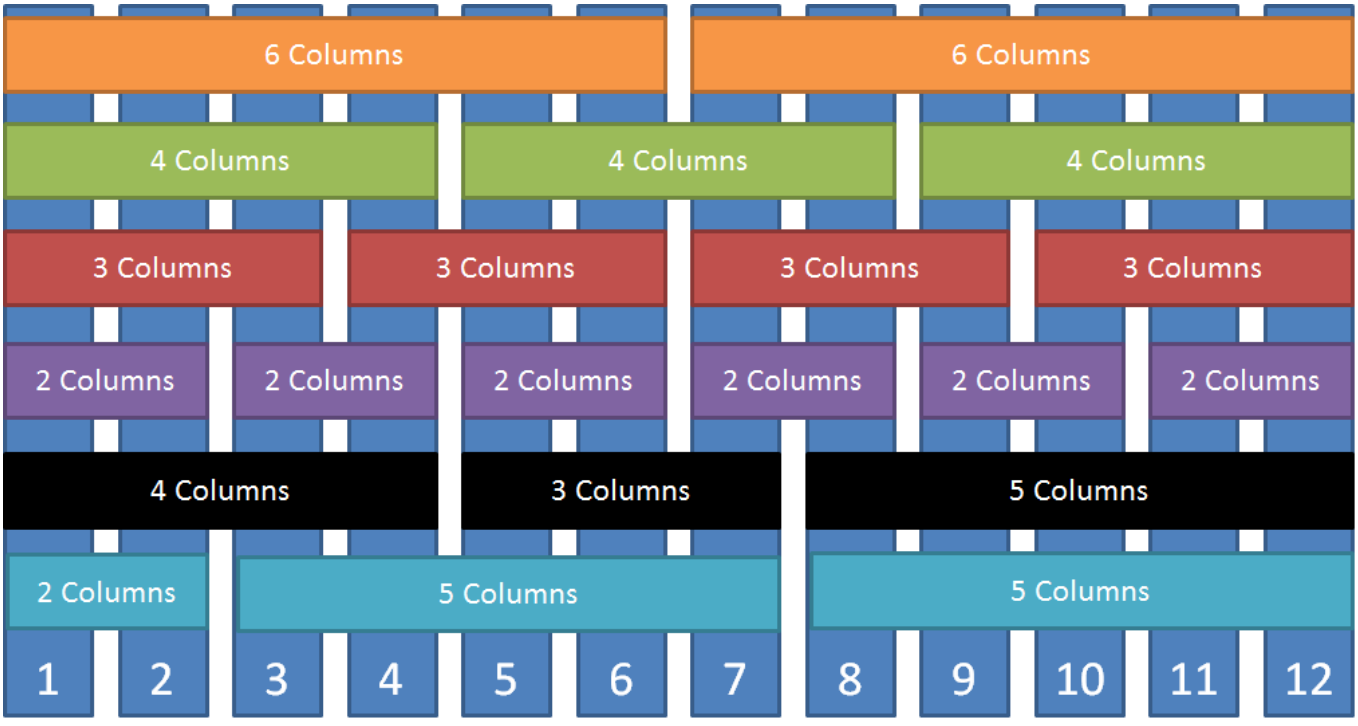
In the screenshot below, there are two blocks on a row, each block containing components. The blocks are both set to 6 columns wide. The components within the block have their own 12 column grid which allows them to be arranged within the available space of the block. In block one there are two text fields both set to 6 columns wide. In the second block, there are three text fields, all three set to 4 columns wide.



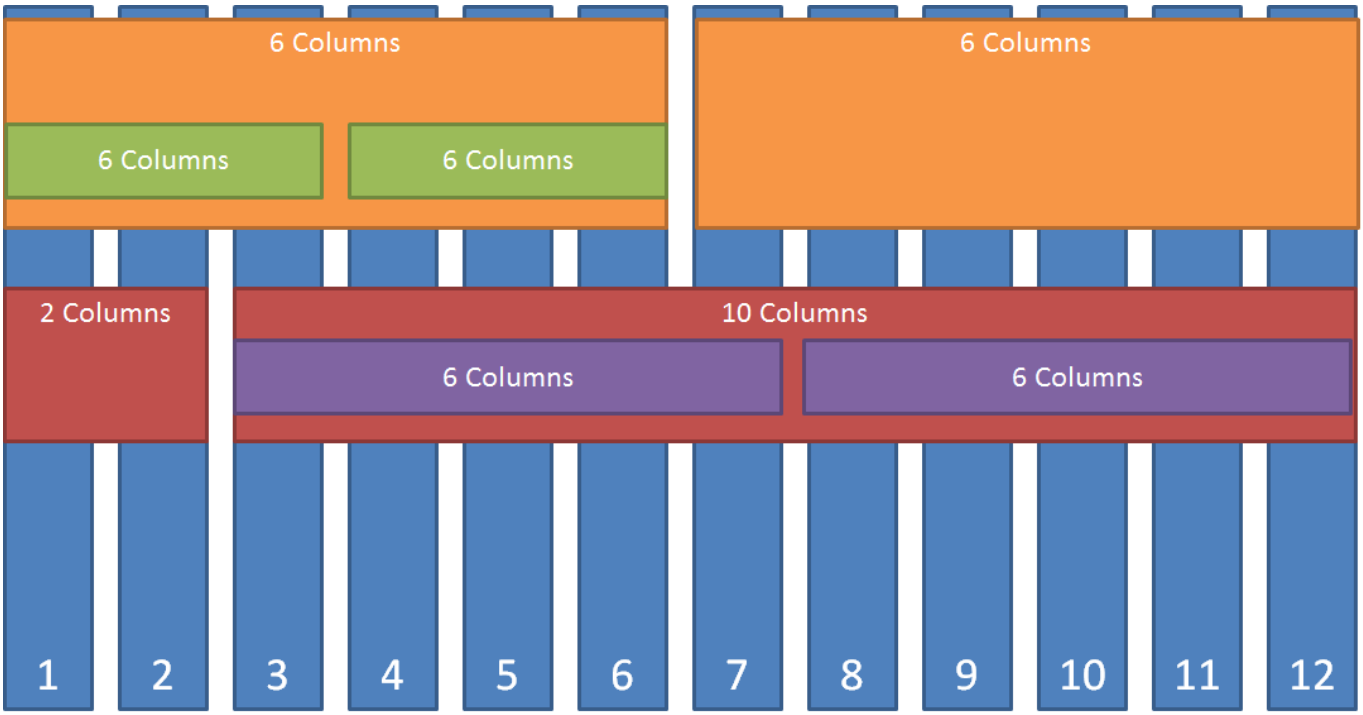
Example Layouts

The following examples illustrate several different Journey Maestro component layouts.

Example 1



Example 2

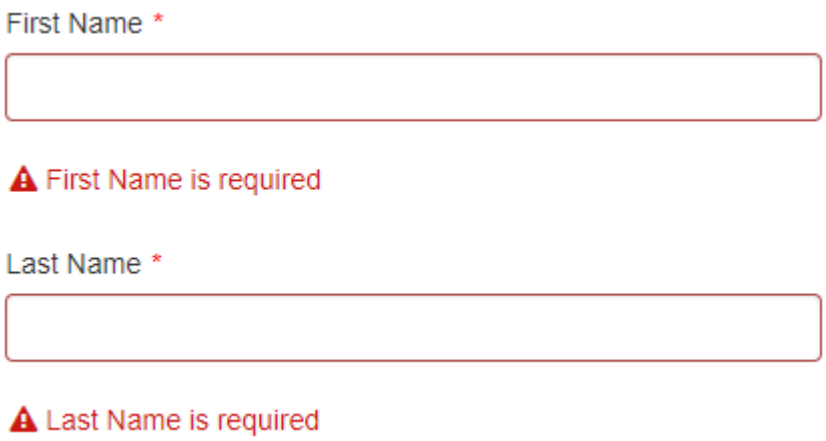


Next, learn how to validate a component layout.

Validation Message Layouts

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

If a user tries to submit a form with invalid fields, such as empty mandatory fields, an error message will appear beneath each field:



But if two or more fields are side-by-side, the error messages will appear in a list beneath both fields:



Some people find this behavior surprising as it doesn't show the clearest possible relationship between the error message and the invalid field. They would prefer the error messages to appear beneath the fields:



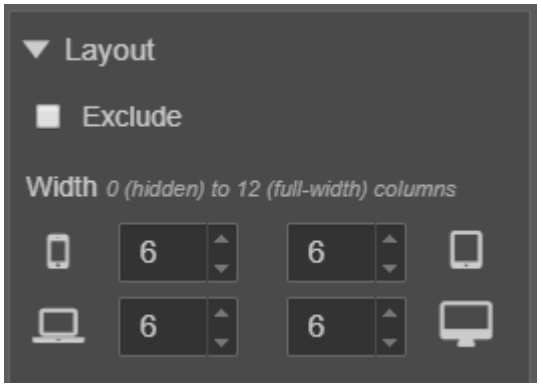
The above style isn't the default approach because, if you have longer error messages or if there are more than a couple of fields in a single row, the error messages can appear crowded:



But there are arguments in favor of this style. It's just one of those details where there isn't a completely correct answer, so the standard behavior has to be settled on. Just because error messages don't always appear beneath their fields by default though, doesn't mean it isn't possible. It does, however, require a few extra steps.

To force error messages to appear beneath their fields:

1. Right-click a component in the **Navigator** pane.
2. Click **Wrap in Block**.
3. Repeat the above steps for all relevant components.
4. Adjust the **Layout** properties for the **Block** components so they can fit in the same row.



5. Drag and drop the **Block** components into the same row.



If, for example, you have a couple of **Text Field** components, you should end up with a component hierarchy that looks like this:

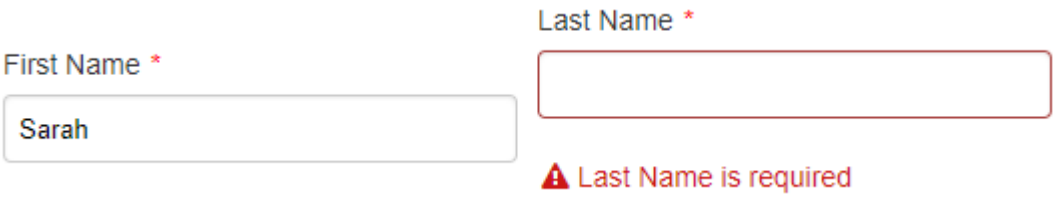


Based on these changes, the text fields will be side-by-side, and because they're wrapped in **Block** components, any error messages will appear within the blocks. The blocks force the error messages to appear directly beneath the relevant field:



Fixing Vertical Alignment

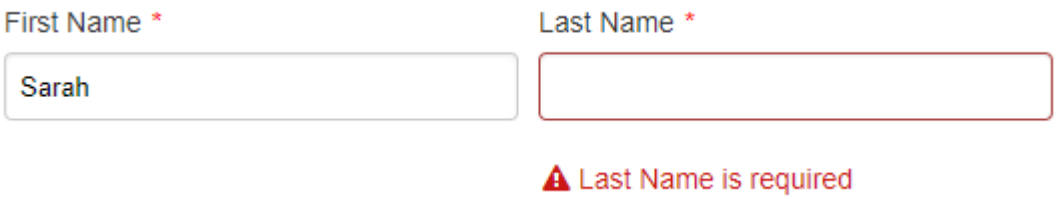
By solving the first problem, we've created another one. To see what's wrong, enter valid data into one of the fields and submit the form. You'll see that the error message beneath the invalid field breaks the alignment with the valid field:



Fortunately, the solution is simple:

1. Inside the Maestro Editor, select a **Block** component.
2. Open the **Styles** pane on the right side of the screen.
3. Open the **Item Appearance > Layout** panel.
4. Set the **Vertical Alignment** dropdown to **Top**.
5. Repeat this process for all other **Block** elements.

After re-building the form, the alignment issue should be fixed.



Next, learn about component versions.

Component Versions

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Component versions work similarly to Form Versions. Once an initial version of a component is built, new versions of this component can be created to keep track of any changes made to the component as it grows and matures.

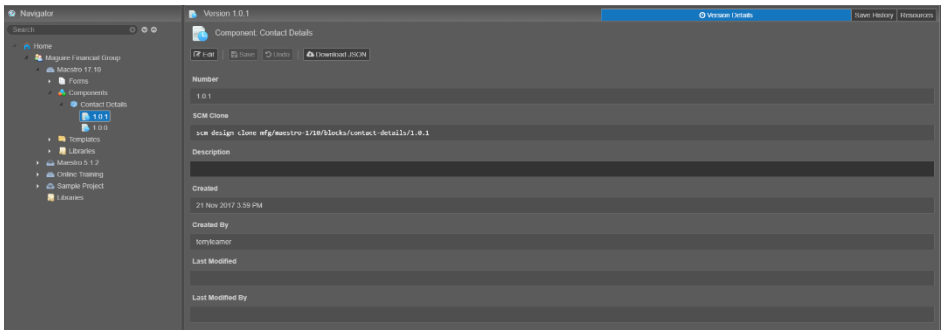
You can use the Maestro editor and the Maestro Dashboard to manage component versions:

- Create a new version of a component.
- Open a specific version of a component.
- Delete a specific version of a component.

Create a Component Version from the Maestro Dashboard

To create a new component version from the Maestro Dashboard:

1. Open a project in your organization.
2. Navigate to the Components folder within a project.
3. Select a component to view the existing versions. You can also expand a component from the Navigator pane to see its versions.



4. Click New Version.
5. Enter a Description for the new version of this component.
6. Click Create. The new version is created, and the version details are displayed in the Dashboard. You can also see the new version listed under the component in the Navigator panel.
7. Select this version and click Edit to open in the Maestro editor.

NOTE

Journey Maestro prior to v17.10 doesn't use semantic versioning[1], so components version numbers are increased incrementally by whole values, for example: version 1, version 2, version 3, and so on.

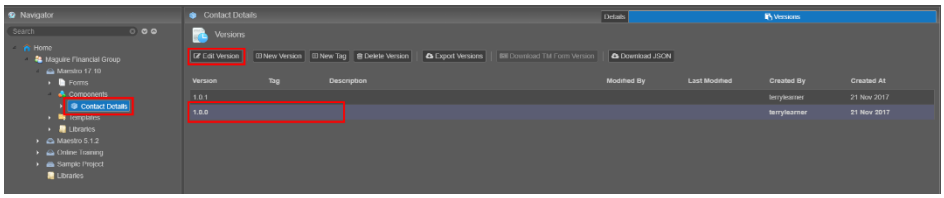
Open a Specific Component Version

Journey Maestro allows you to choose which component version to open. When you double-click an existing component in the Maestro Dashboard, the latest version of the component is opened. However, sometimes you may want to open an earlier version of the component. This is usually the case when there are bugs or issues with the current version of a component and you want to open the working version to compare the configurations or even revert the changes.

There are two methods to open an existing version of a component in Maestro:

Method 1

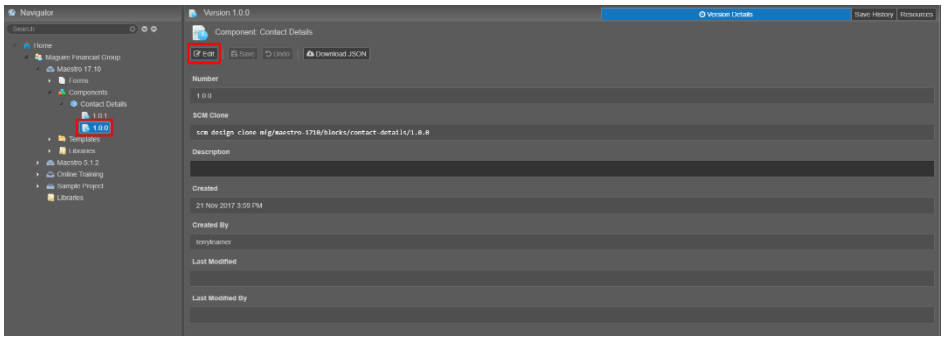
1. Open a project in your organization.
2. Select the component in the Navigator pane of the Maestro Dashboard.
3. Select the earlier version you want to open in the Versions tab.
4. Click Edit Version.



[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.

Method 2

1. Open a project in your organization.
2. Expand the component in the Navigator pane.
3. Select the version you want to open.
4. Click Edit.



NOTE

Journey Maestro prior to v17.10 doesn't use semantic versioning[1], so components version numbers are increased incrementally by whole values, for example: version 1, version 2, version 3, and so on.

Next, learn how to add a component.

[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.

Add a Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to add components to a form to capture the data that is required to implement a Customer Experience. The term component is the general name for anything that can be dragged from the Maestro Palette into a form. Components can be anything from an image, to a single data entry field, to a large complex reusable block. Components come in a variety of control types with different properties and behaviors. The available components are displayed in the Maestro Palette pane.

Components are organized within category folders. You can browse each category to find the component you want to use on your form, or you can use the search box at the top of the Palette.

At the top of the component list, you will find the favorites category. This category displays the most commonly used components. You can add or remove components from the favorite list based on your own personal preferences. The changes you make to the Favorites list will be stored in your Maestro account, not within the form. The components you add or remove to favorites will be displayed each time you open a form in Maestro.

NOTE

Prior to Maestro v18.05, the View pane is called the Structure pane.

There are several ways to add a component to a form.

- Drag a Component from the Palette to the Form. Choose a component from the Palette and drag it onto the form near where you want it to appear, observing the shape and orientation of the insertion bar. Drop the component when the insertion bar indicates the desired location.
- Double-Click a Component. Select the component or container in the form where you want a component to be inserted. Double-click a component in the Palette. The component is inserted directly below the selected component or at the bottom of the selected container.

  NOTE

  If you have multiple components to add, you can keep double-clicking – each new component is added below the previous component that was added.

- Add Below an Existing Component. Select the component in the form above where you want the component to be inserted. Right-click the component from the Palette and select Add Below. The component is inserted directly below the selected component.
- Drag a Component from the Palette to the Component Hierarchy. If the Palette and View panes are displayed side by side, you can drag a component from the Palette and drop it in the View pane. The new component is added to the form at a position determined by where in the View pane you dropped the component. If you drop the component on a container, the new component appears at the bottom of the container. Otherwise, the new component is positioned where you dropped it relative to the other components.

Once you have added a component to a form, you can update its ID.

Next, learn where and how to insert a component.

Insert a Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro allows you to insert a component in a form using one of the following methods.

Insert a component using the Maestro editor

The insertion bar indicates a wide variety of possible insert locations by using different positions, sizes and shapes to represent the insertion point. The insertion point is represented by a selection area, which indicates the parent layout of the component, and an insertion bar which indicates its position to its siblings.

The insertion bar will appear either as a vertical or horizontal bar. A horizontal insertion bar will insert the component vertically between two other components, and a vertical bar will insert a component horizontally between two components.

If the insertion point is at the beginning or end of the list of siblings, then the component will be inserted regardless of the insertion points horizontal or vertical orientation. If the the insertion point is vertical, and there is space for the new component to occupy within the layout, the new component will appear on the same line as an previously placed components. If there isn't sufficient space to accommodate the component, it will be placed on the following line.

Insert a component using the component hierarchy

Similar to the Maestro editor, an insertion point is displayed when you drag a component from the Palette panel to the View panel.

In the View panel, the insertion point appears as a triangle above, below or on the desired component if it's a container. You can change the insertion point while dragging by moving up and down the hierarchy. If the drag target is a component, the insertion point will appear above indicating the dragged component will be inserted before the drag target, and the same is true for below and after.

If the drag target is a container of other components, you can insert the component into the container by placing the insertion point on the drag target. Releasing the dragged component will add the component to end of the children. If you drag a component onto a container that is collapsed, the container will expand, displaying any children it contains, allowing you to navigate deeper into the structure without releasing the dragged component.

You can change any mistakes or misplaced components by simply dragging the component to the correct location. You can also use the Undo button, moving back the History or simply deleting the component and trying again.

Next, learn how to delete a component.

Delete a Component

MaestroThe UI design product. | Form Builder | 18.11 This feature was updated in 18.11.

Maestro allows you to remove a specific component or a component version from a form, if it is not required any more.

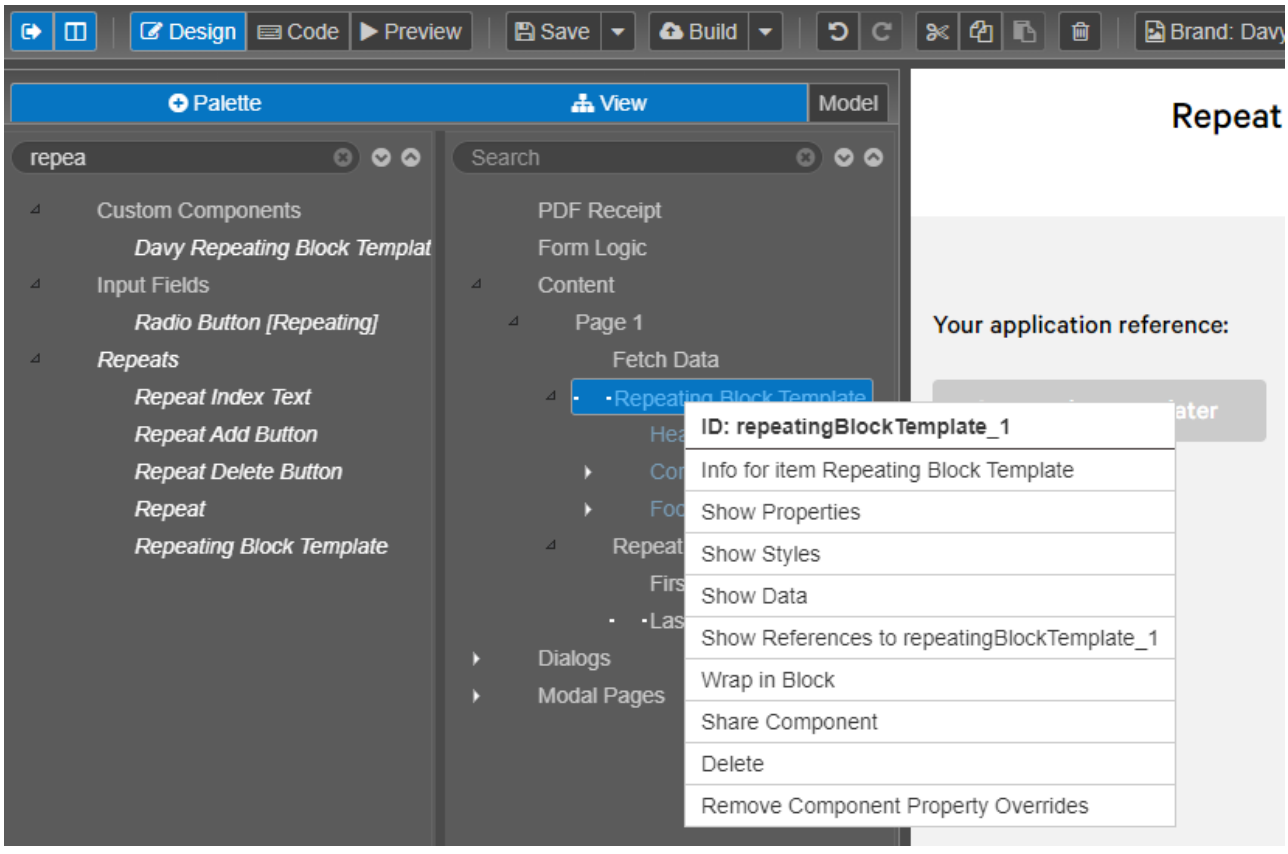
NOTE

Once you have deleted a component, you can't undo this, because Journey Maestro removes the component permanently. Therefore, you should consider all possible impact of deleting a component because it can affect many, or even all, forms using this component.

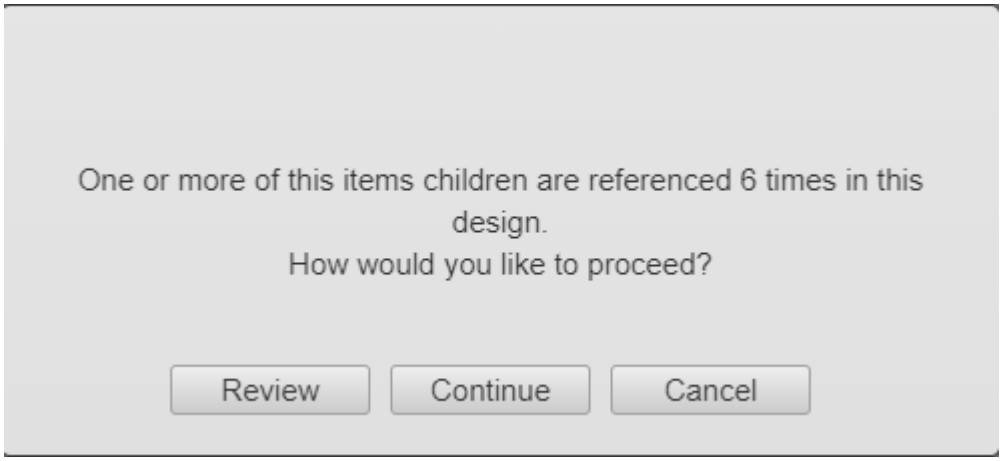
Journey Maestro provides several methods to delete a component from a form.

Delete a component using the context menu

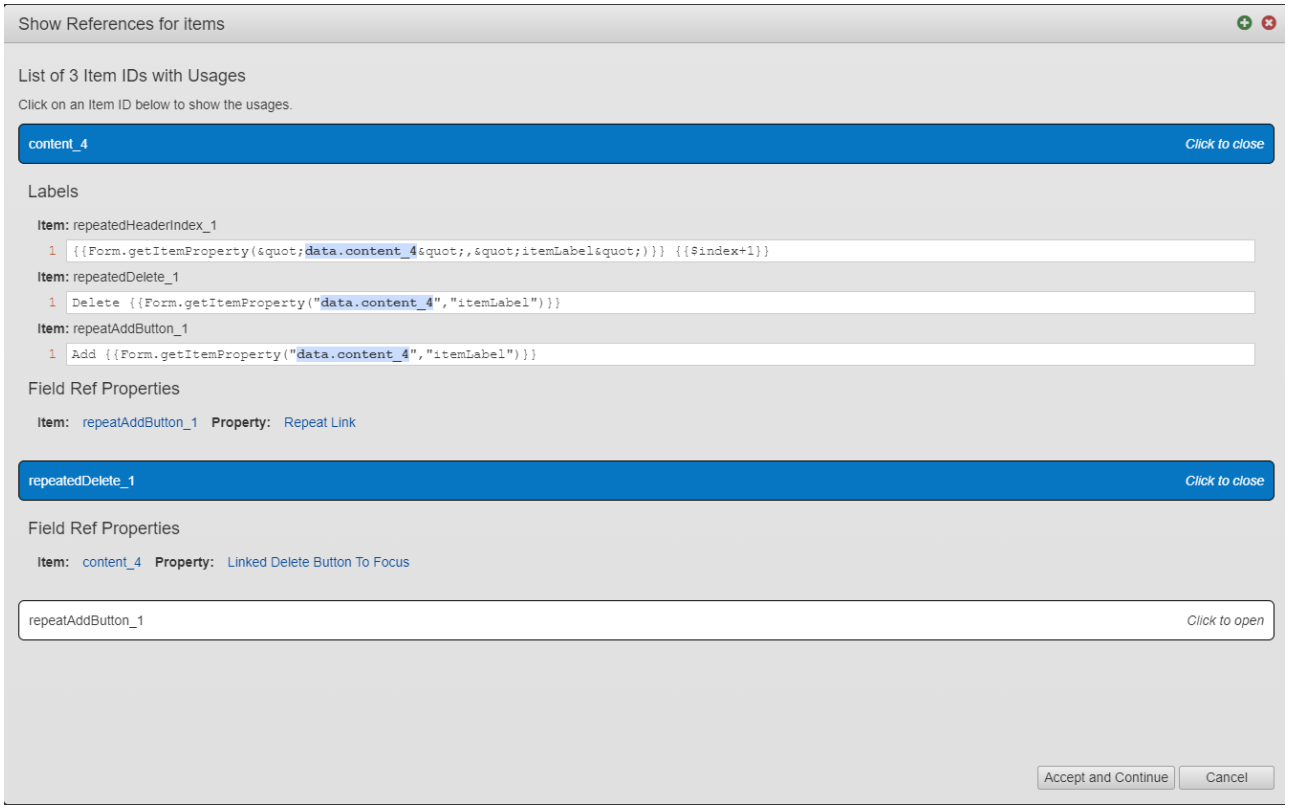
1. Right-click the component in the View panel (the Structure panel for Maestro prior to v18.10) or right-click it directly on the form.



2. Click Delete. If the component is referenced by any other component, you must review it and confirm deletion as shown below.



3. Click Reviewto check an impact analysis that lists any references to the component you are deleting. Click on each Item ID to check how it's used



4. Click Accept and Continue to delete the selected component. Any references to the deleted component on the form are unchanged, so you need to make changes in the form as required. Click Cancel to close the dialog without deleting the component.

5. Click Continue to delete the component, without reviewing the current references. Any references to the deleted component remain in the form, so you need to update or remove these references manually. Click Cancel to close the dialog without deleting the component.

Delete a component using the toolbar

1. Select the component from the View panel (the Structure panel for Maestro prior to v18.10) or select it directly on the form.



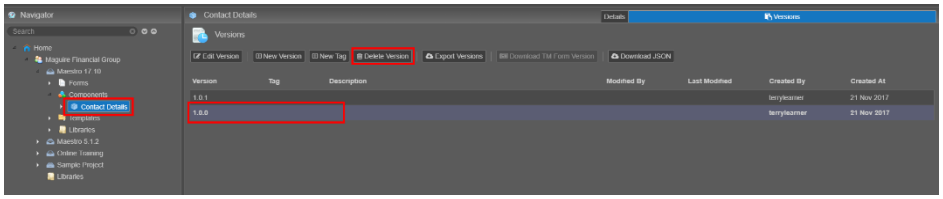
2. Click Delete on the toolbar. If the component is referenced by any other component, you must review it and confirm deletion as described above.

Delete a component using the keyboard

1. Select the component from the View panel (Structure panel prior to Maestro v18.05) or directly on the form
2. Click Delete on your keyboard. If the component is referenced by any other component, you must review it and confirm deletion as described above.

Delete a component version

1. Open a project in your organization.
2. Select the component in the Navigator panel.
3. Select the version to delete in the Versions tab.



4. Click Delete Version and click OK to confirm your decision. If the component is referenced by any other component, you must review it and confirm deletion as described above.

Next, learn how to insert a component.

Component Form Dependencies

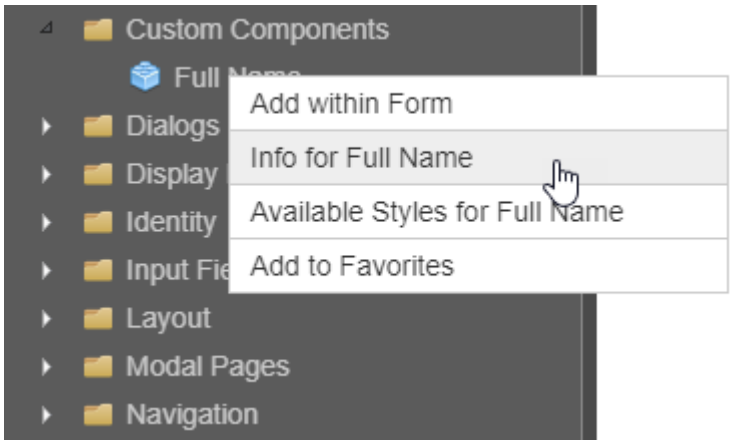
MaestroThe UI design product. | Form Builder | 18.11 This feature was introduced in 18.11.

As a form is developed, it begins to depend on a range of libraries and components. The more projects and forms a single organization is working on however, the more difficult it can be to keep track of these dependencies.

To assist with this, Maestro provides a Component Details modal that allows Template Designers and Form Builders to keep track of where components are located.

To access the Component Details modal:

1. Right-click a component via the Navigator pane.



2. Click Info for <component name>.

If you view the details for a component that hasn't been published to a library, such as the default Email Address component, you'll only see the following details under the Published Library heading:

Property	Description
Library Name	The name of the library.
Library Type	This is also known as the Library Scope. Options include Release, Organization, and Project.
Version Number	The version number of the library.



When a component has been published to an organization or project-level library, however, additional details will be available under the Source Details heading:

Property	Description
Organization	The organization that owns the component.
Project Name	The project that contains the original component.
Design Type	This value will always be "Components".
Component Name	The name of the component.
Version	The version number of the component.
URL	A direct link to the original, editable component.



NOTE

As the above image shows, a message will appear beneath the URL if the editable version of the component is located in a separate project.

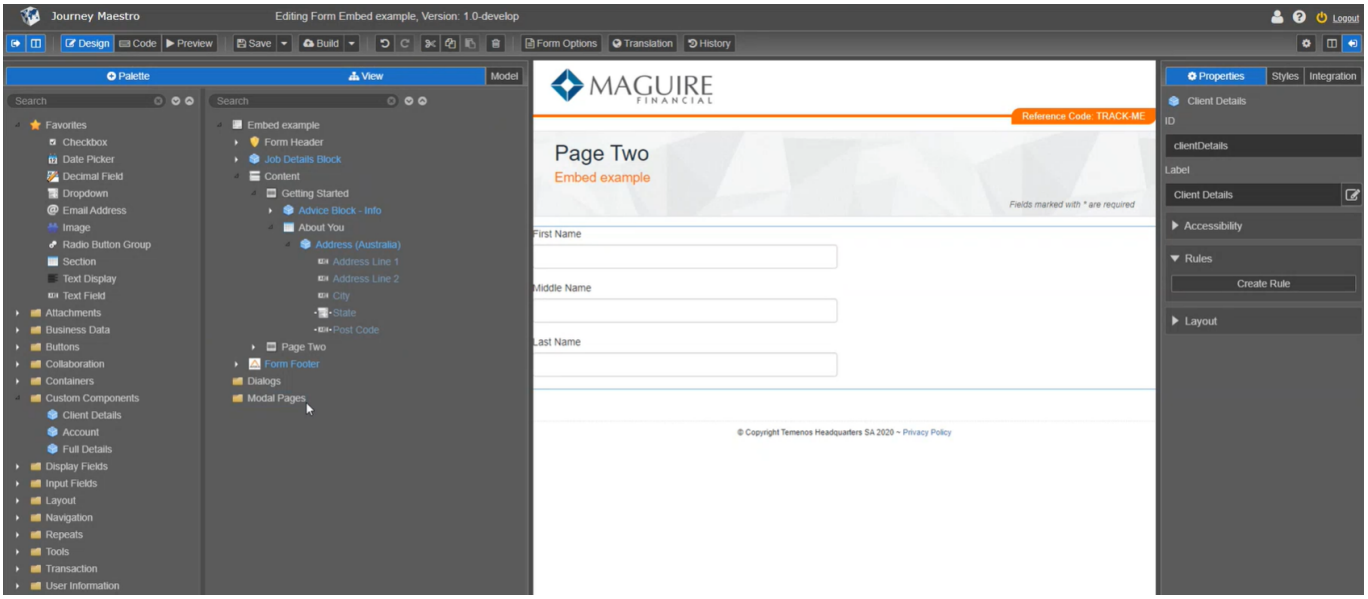
Next, learn about Embedded Component Inheritance.

Embedded Component Inheritance

MaestroThe UI design product. | Form Builder | 20.05 This feature was updated in 20.05.

Journey Maestro gives you an ability to embed shared components into the form, introduce impact analysis for component modifications and detect possible conflicts when publishing to a library. This functionality is built on our previous work, release 19.11, making the inheritance relationships between forms, templates, components and libraries more transparent.

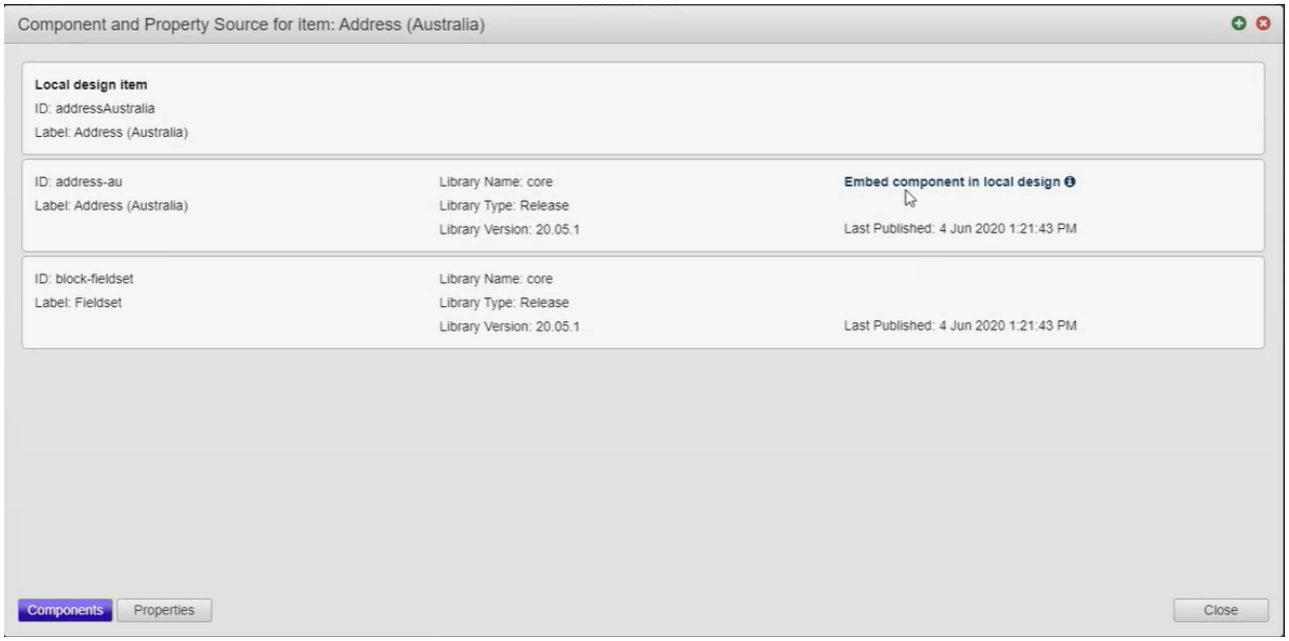
To demonstrate the new functionality, let's open a simple form that uses the Address (Australia) core component.



In the past, to add a component to the existing elements, you can only do it at the end of the list. If you try to insert it somewhere else, say in the middle, it won't work and you'll get the Can't insert before pre-defined component children error message. Now, you can embed this Address (Australia) component into a form, so you can edit it and it won't be updated when this core component is updated.

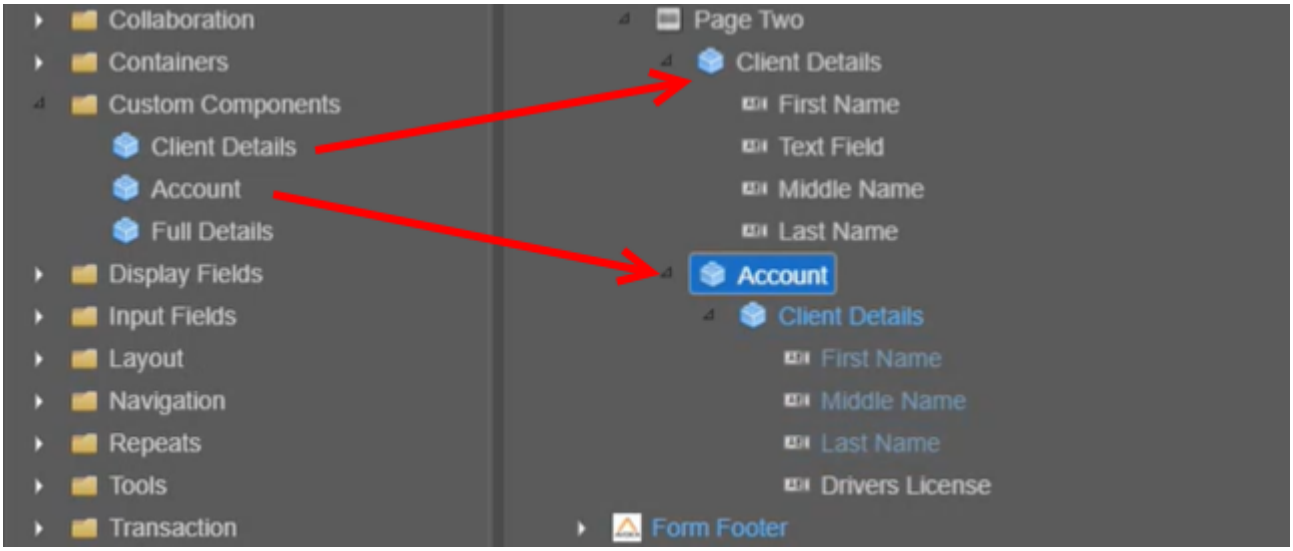
To embed a component into a form:

1. Open a form.
2. Select the component (it's highlighted blue).
3. Right click on the component and select Info for item Address (Australia).



4. Select Embed component in local design to make this component a part of your form. Now, you can add new elements to the embedded component. There is no linking between the core component and this embedded one anymore, so any update of the core component won't change your form.

You can do the same for Custom Components , so your can use them as templates. You can also embed nested custom components, which keep their structure as shown below.



If a component has properties, select the component, right click, select properties, and embed it. Properties injected into a labels, they are substituted into a binding. If they are referenced in the rules, they are substituted too.

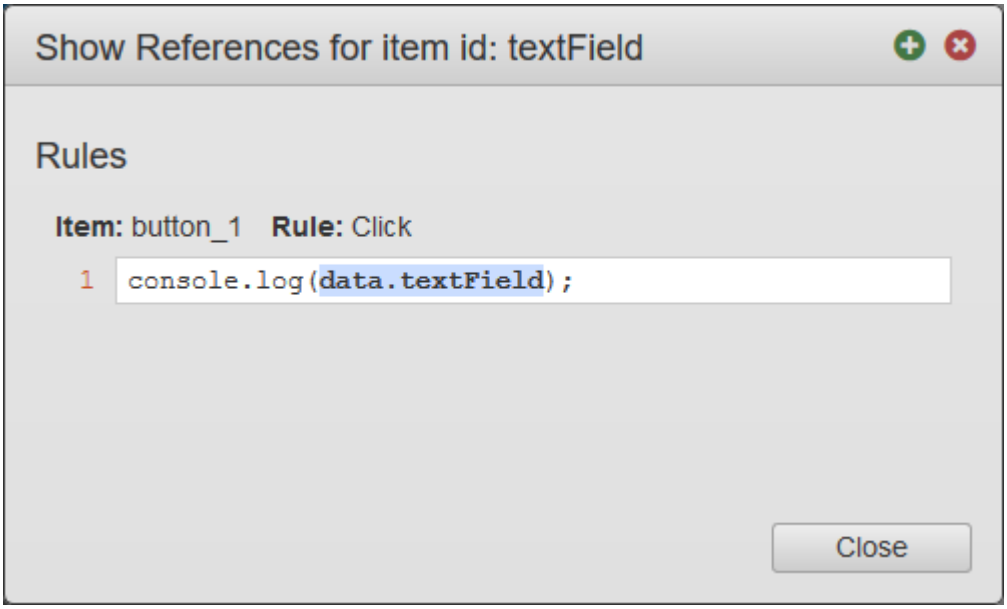
If you build widgets in the component editor, they are almost always embeddable. However, you can't embed native widgets into a form and the embed component link will not be available, if the widget has additional resources, such as an HTML template, CSS or JavaScript, etc. The footer, for example, has a separate CSS file so is classed as a native widget.

Next, check our components best practices.

Show References

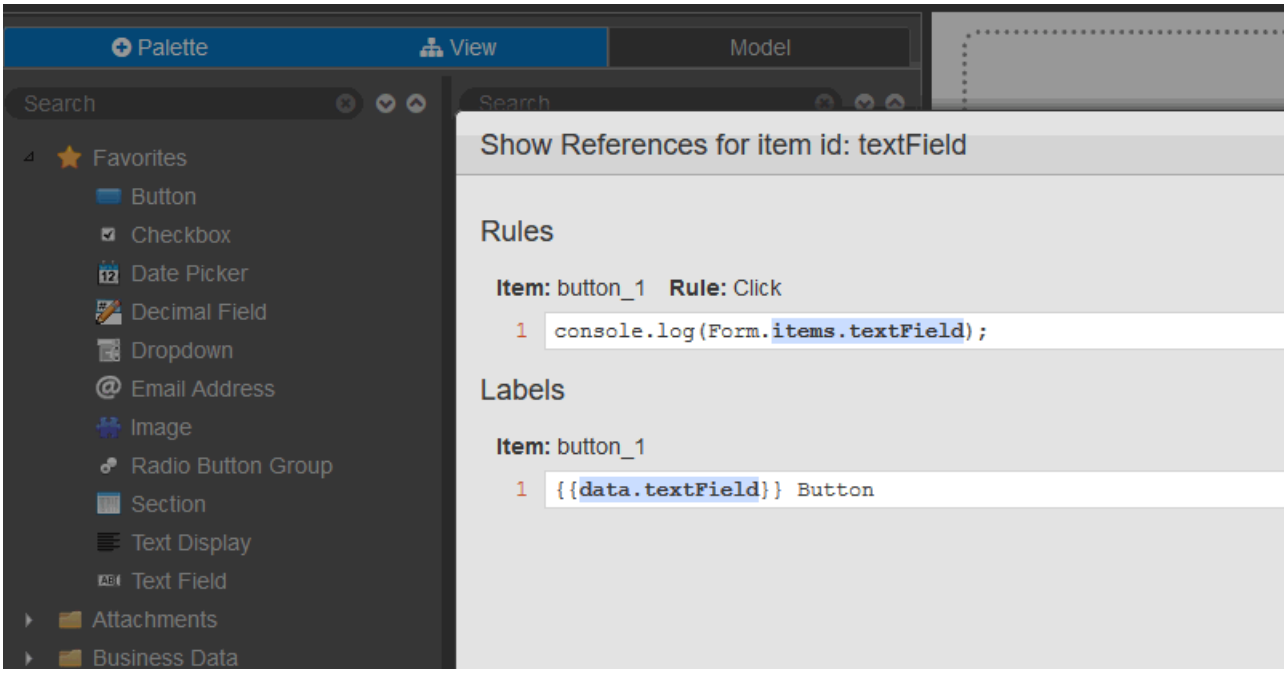
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Maestro editor allows you to view all references to a selected component so you can find out where the component is used from. Maestro does it by scanning the component code and properties for matching field references using the component's ID. For example, if there's a component with an ID of `textField` and a rule that contains a reference to `data.textField`, that rule will appear in the **Show References** dialog for the `textField` component.

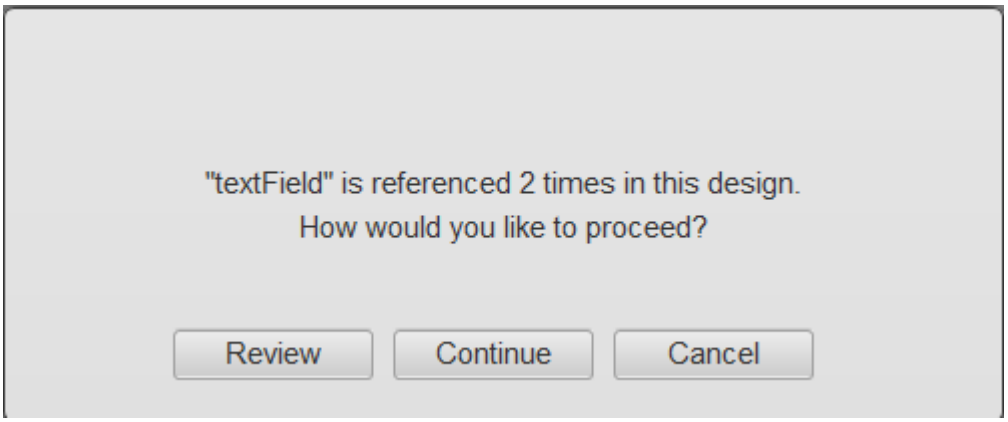


To view component references:

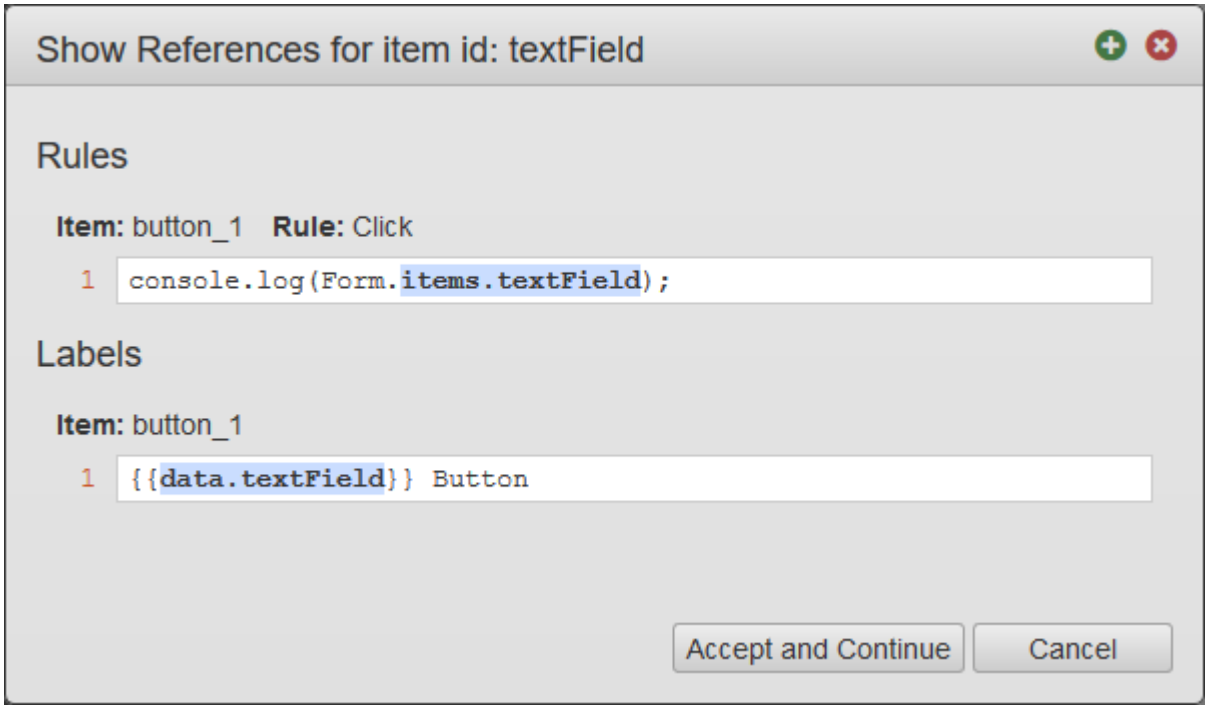
1. Open a form or a template in the Maestro editor and click the View tab.
2. Right-click the component in the Palette pane or Wireframe and select Show References to [COMPONENT ID].
3. Click the reference content to be taken to where the reference is defined. For example, if a component is referenced in a rule, clicking the reference content will open that rule in Code Mode.



4. Right-click the component and select Delete to remove the component that is referenced elsewhere. The following warning will appear.



5. Click **Review** to open the **Show References** dialog for the component you're attempting to delete, allowing you to view the references that will be affected by deleting the component. In this case though, it will also include an **Accept and Continue** button.



6. Click **Accept and Continue** to delete the component. The references to the component will no longer work.
7. Click **Continue** to delete the component. The references to the component will no longer work.
8. Click **Cancel**, so the component won't be deleted.

Next, learn about the component ID.

Import and Export a Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro components belong to Maestro assets and resources. When you create a new component, it is stored in a project you are working in. However, you can move or copy it to other projects, which is a simple way to reuse the component, see shared components. You can export and then import a component to achieve this.

To export a component, select the component from the Maestro Dashboard and click Export. Once you have exported the component, you can import it to a different location.

To import a component, select the component from the Maestro Dashboard and click Import.

> NOTE
>
> When moving or copying a component, be aware that the component is not available within the project until it is published into a library.

If you have imported the entire project or the library, the published version of the components in the imported project is already available. However, if a component is imported on its own, you need to publish it first, so it is available to forms within the project.

Repeating Components

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro provides the repeating components that allow you to add (repeat) multiple instances of a component to a form so that you can implement various business requirements, for example, to sum up a total of values stored in repeating components in a form. There are the following repeating components, which you can find on the Palette tab in the Repeats folder:

- Repeat
- Repeat Add Button
- Repeat Delete Button
- Repeat Index Text
- Repeating Block Template

You can directly access and inspect the data model of a repeating component after you build and render a form. Type `maestro.Form.data` into the browser console to see the repeats that are an array with each child element in the array having the following 3 properties:

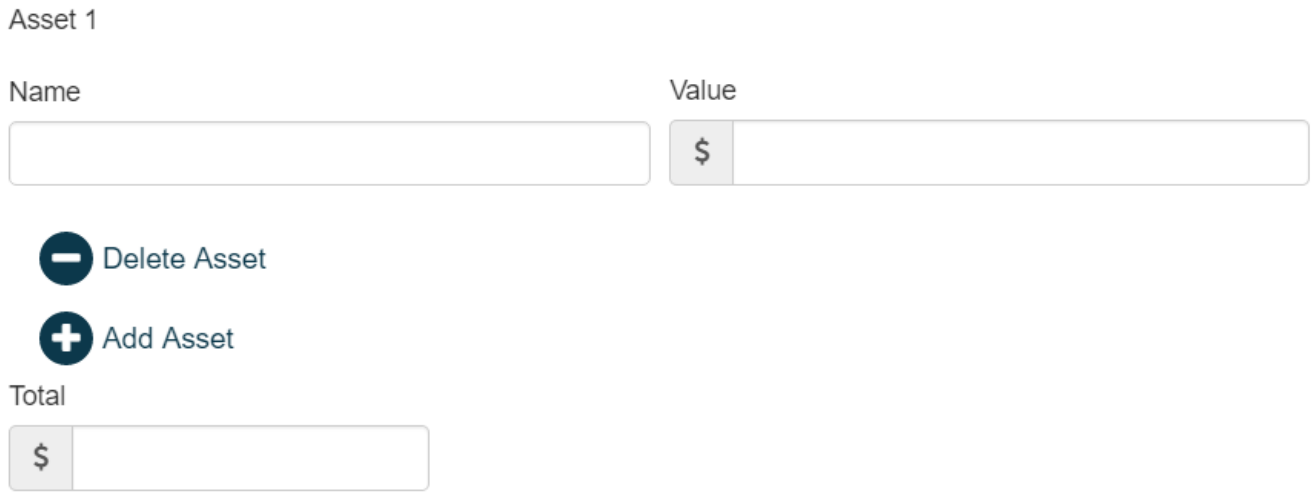
- `$r` is the array
- `$i` is the array index of a row
- `$p` is the parent data scope

Next, learn about the repeat component.

Repeating Block Calculations

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro allows you to apply Calculation rules, see calculation rules overview, to a Repeating Block component to implement customer requirements. You can use this functionality to calculate the sum of specific components within a repeating block. For example, a form requires a user to provide asset's name and its values in dollars ($). Once values have been entered, a total sum of all assets is calculated and displayed as the Total.



A user can add more assets to the form as required. The total amount is then re-calculated to reflect the additional component in the block.

To create a calculation rule on a repeating block, follow these steps:

1. Open a project in your organization.
2. Open the component in the Maestro editor.
3. Add a component outside of the repeating block template.
4. Select the component and click Create Rule in the Properties pane.
5. Select Script under Calculation.
6. Navigate to Repeating Block Template > Content > Repeated Content.
7. Select the component to hold the result of calculation, which is the total in this example.
8. Right-click the component and select Get sum of repeating data.
9. Click Save. The calculation rule is added to the Properties pane.

Re-usable Components

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro provides a variety of different types of re-usable assets you can use to build many forms that require a high degree of consistency. Re-usable components are one of the more frequently reused assets, which allow Form Builders to create forms that encapsulate the same functionality.

Re-usable components can be anything from a customized address to a large multilayer component built using a repeating block template. You can apply them many times in various scenarios across different forms and projects.

Journey Maestro allows you to reuse components at the following levels:

- Component level
- Form level
- Style level

Re-use Components at the Component Level

You can reuse the following Journey Maestrocomponents at a component level.

- Native components

  Native components are the basic widgets that are supplied with Maestro. These include components for entering a single data value, such as data entry fields, checkboxes, radio buttons, and dropdown lists. There are also a number of more sophisticated components such as numeric and number fields, and more. Many components have at least two subparts, the label and the data entry area. Some also include an icon (such as a currency indicator), and most will allow for an optional help icon to display help text. A date field is arguably a more complex component, because it includes a pop-up calendar control - but is still really just a simple component.Under the covers, a simple component consists of an HTML definition, a Maestro descriptor that tells Maestro what properties to display in the Properties tab, and optional JavaScript and CSS. However, as a Maestro user, you are shielded from the underlying complexity, and simply drag and drop the component into your form. These components are usually shipped in the product, although they can be built by anyone with HTML, CSS and JavaScript skills.

- Custom Native components

A custom native component is often made up of more sub-parts than a simple component. For example, a Heading consists of not only the Heading text, but sub-text, help text, a collapse/expand button, and a significant amount of JavaScript and CSS. An even more complex component such as a Driver's License scanning component may have multiple data entry fields, a camera button, descriptive images, and more. However, logically it is still a completely self-contained component, and occupies only a single line in the structure pane.In reality, a complex component has exactly the same structure as a simple component, just there is more "stuff". It is still made up of HTML, JavaScript and CSS, it's just that there is usually more of it. There is no fixed boundary between a simple and a complex component, although any component with more than one visible data entry area within it would usually be considered to be complex. Avoka staff, or someone with HTML, CS and JavaScript skills can create these components..

- Custom component

  Sometimes it is useful to create a re-usable variant of either a simple or complex component. The main reasons for this are:A component with a re-usable business rule, particularly a validation rule. An email field is an example of this.A component (usually a radio button or drop down list) with a specific set of values. An example of this is a drop down list of countries.A component with a particular XML binding. A custom component can be defined and re-used only within a single form, or can be published to a library for use among multiple forms. A Maestro Developer can create these components.

- Re-usable Block

  A re-usable block is a re-usable collection of simple components. 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 complex 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. Any Maestro user can create these components.

Re-use Components at the Form Level

You can reuse the following Journey Maestrocomponents at a form level:

- Template

  A template is an empty form that can be used as the starting point for new forms. Generally, a template contains all the so-called "chrome" that goes around the outside of the actual components that a Form Builder will add to the form. The chrome includes content such as

navigators, headers and footers, and standard dialogs. A template can be created based on an existing template or can be built from scratch. A Maestro Template Designer can build this component.

- Brand

  A template usually contains one or more brands. A brand allows a Form Builder to select a particular set of colors, fonts and other visual elements that can be applied to a form. A brand generally applies to a specific template, because the colors and fonts are applied to the "chrome" of the template, such as headers, footers, and navigators. The brand also applies to the components that will be added to the form.It is useful to separate the brand from the template in this way, because this allows an organization to come up with a single overall "shape" of all of their forms, but still allows them to define different color schemes, logos, and other branding elements for each of the sub-entities within their organization. This is particularly helpful for larger organizations who have multiple sub-organizations that have their own brand, or for organizations who white-label their products to other organizations. A Maestro Template Designer can build this component.

Reuse Components at the Style Level

You can reuse the following Journey Maestrocomponents at a style level:

- Simple Shared Style

  Sometimes you may want to style a component in a particular way. For example, you may want to create a Red Text Input. One way to do this is just to change the color using the Styles tab. However, this might result in different people choosing a different shade of red. Instead, you can create a shared style that defines the correct shade of red, has a name, and can be applied to any Text input.The shared style can be used simply within your own form, or can be published to a library for re-use among multiple forms. Any Maestro User can reuse this component.

- Complex Shared Style

  Maestro provides the capability to modify many of the simpler CSS style elements using an intuitive graphical editor. However, sometimes it may be desirable to create much more complex visual effects using the power of CSS. Maestro allows custom CSS to be used, but this requires the skills of someone who understands the intricacies of CSS. A CSS expert can reuse this component.

Styles are usually created by a Template Designer who shares them in Journey Maestro, so a Form Builder can use them in new forms.

You cannot make a Red Label Text Field as a Custom Component. This is because Custom Components only contain custom properties, which only affect functionality, but not look and feel. You will need to create a Red Label Custom Style, which can be applied to any Text Field.

In Maestro, you can control component functionality and styling by two different re-usable elements, which don't overlap:

- Custom components that are responsible for functionality.
- Custom styles that are responsible for look and feel.

Native Components Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Native components are the core components that come pre packaged with Journey Maestro. Native components can be customized and shared.

Native components are the basic widgets that are supplied with Maestro. These include components for entering a single data value, such as data entry fields, checkboxes, radio buttons, and dropdown lists. There are also a number of more sophisticated components such as numeric and number fields, and more. Many components have at least two sub-parts, the label and the data entry area. Some also include an icon (such as a currency indicator), and most will allow for an optional help icon to display help text. A date field is arguably a more complex component, because it includes a pop-up calendar control - but is still really just a simple component.Under the covers, a simple component consists of an HTML definition, a Maestro descriptor that tells Maestro what properties to display in the Properties tab, and optional JavaScript and CSS. However, as a Maestro user, you are shielded from the underlying complexity, and simply drag and drop the component into your form. These components are usually shipped in the product, although they can be built by anyone with HTML, CSS and JavaScript skills.

KEYBOARD SHORTCUT

Most of native component have a keyboard shortcut assigned to them.

Next, learn about Attachment Field Component

Attachment Field

MaestroThe UI design product. | Form Builder | 20.11 This feature was updated in 20.11.

Maestro comes with the Attachment Field component (widget) that allows a form user to upload multiple attachments, such as files and images, which are a part of an onboarding process and are submitted along with a form.

NOTE

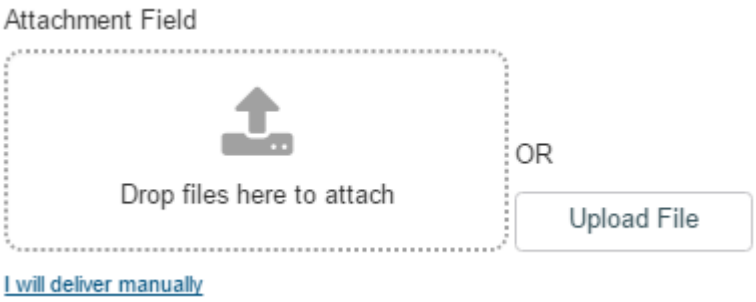
Since Maestro 19.05, all standard versions of the Attachment Field widget support a form space portal using CSRF[1] tokens.

NOTE

Starting in Maestro 20.11, the Attachment Field component is the part of the core library. Prior to this, it was part of the Maguire release library.

Usage

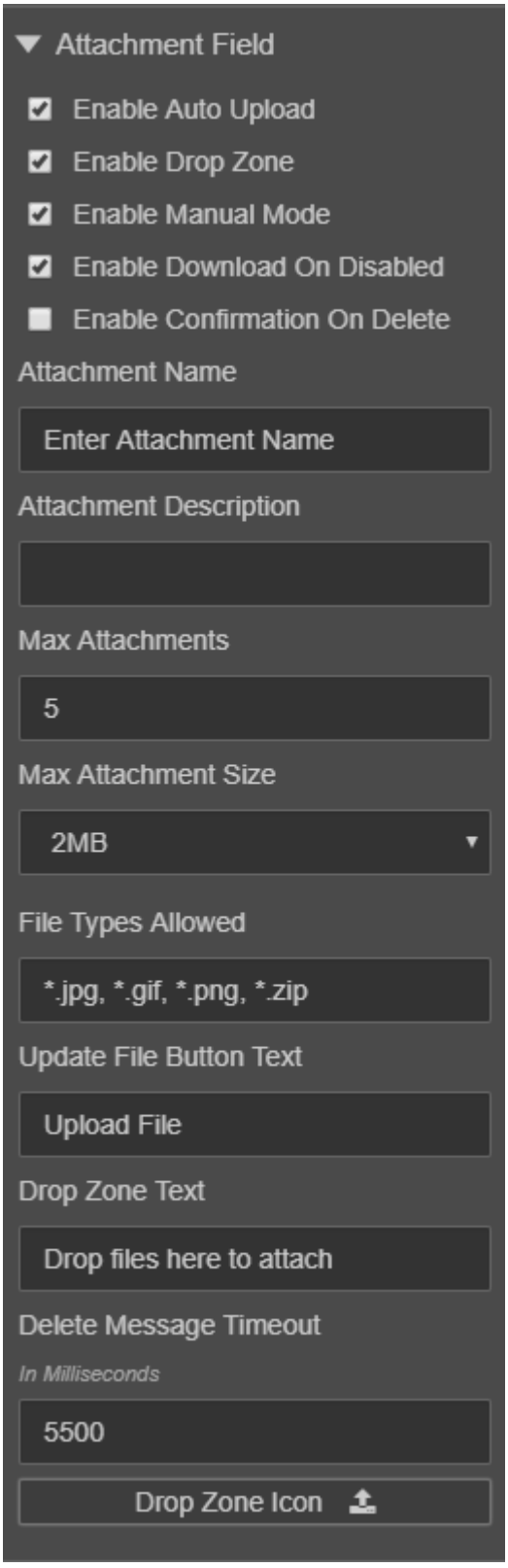
1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Attachment Field** component within the **Attachments** folder.



3. Drag the component into the **View** pane or the **Wireframe**.

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

4. Configure the component's properties via the **Properties** pane.



Properties

The following properties are available to the Attachment Field component:

Enable Auto Upload
Boolean enableAutoUpload

Enable the ability to auto upload attachments. It automatically uploads the attachment once the user drags it into the drop zone or uses the upload button. This option may not work when previewing the form, it only works once the TM form version is built.

Enable Drop Zone
Boolean showDropZone

Enable the ability to show attachment drop zone area, so the user can use the drop zone to drag and drop attachment files there.

Enable Manual Mode
Boolean enableManual

Enable manual attachment feature, which displays the I will deliver manually text on the field. The user can select this option to indicate that they will be sending the attachments via a different method, such as by email or post.

Enable Download On Disabled
Boolean allowDownloadDisabled

Enable download on disable, so users can download attachments from a disabled attachment field.

Enable Confirmation On Delete
Boolean allowConfirmDelete

enable confirmation on delete, so users will be prompted before an attachment is deleted.

Attachment Name
attachmentName

Enter the attachment name, which allows you to distinguish multiple attachment fields per form.

Attachment Description
attachmentDescription

Enter the attachment description, which allows you to distinguish multiple attachment fields per form.

Max Attachments
Integer queueLimit

Select the maximum number of attachments that a user can upload to a form which will be included in a form submission. Once the maximum number is met, the options to include additional attachments (drop zone and upload button) will disappear.

Max Attachment Size
Option maxSize

Select the maximum size of each attachment. If you want to set the maximum total size for all attachments that will need to be configured in Journey Manager.

The following options are available from the **Max Attachment Size** dropdown:

Label	Value
0.5MB	524288
1MB	1048576
2MB	2097152
3MB	3145728
4MB	4194304
5MB	5242880
6MB	6291456
7MB	7340032
8MB	8388608
9MB	9437184
10MB	10485760
15MB	15728640
20MB	20971520

Label	Value
25MB	26214400

File Types Allowed
fileTypes

Enter the file types allowed in the format provided. You can add more types by following the format displayed. For example, if you wanted to add PDF to the allowed types you would type *.pdf.

Update File Button Text
chooseFileButton

Enter text for the button used to upload files for attachment.

Drop Zone Text
dropZoneText

Enter the text for the drop zone used to upload files as attachments.

Delete Message Timeout
Integer deleteMessageTimeout

Delete Message Timeout

Drop Zone Icon
Icon dropZoneIcon

Select the button icon to display in the drop zone. Check with the Template Designer if you can change this icon, as icons are typically customized in the templates.

Error Messages

Invalid File
invalidFile

Error to display if the file is invalid

Invalid Size
invalidSize

Error to display if the file size is 0KB

Exceeds Maximum Size Limit
exceedsSizeLimit

Error to display if the file size exceeds the maximum limit

Exceeds Maximum Total Size Limit
exceedsTotalSizeLimit

Error to display if the total attachment size for this attachment name exceeds the maximum limit

Virus Warning
virusFoundOnUpload

Error to display if the attachment contains a virus when it is uploaded

Max Attachment Count Error
tooManyUploads

Error to display if there are too many attachments uploaded

Help Text

Popover Help Text
Help Text helpText

Success Messages

Successfully Uploaded
uploadSuccessful

Message to display if the file is successfully uploaded.

Successfully Removed
removeSuccessful

Message to display if the file is successfully removed

Thumbnail Options

Enable Image Preview
Boolean enableCanvas

Enable canvas image preview of any images uploaded in the attachment field in the form.

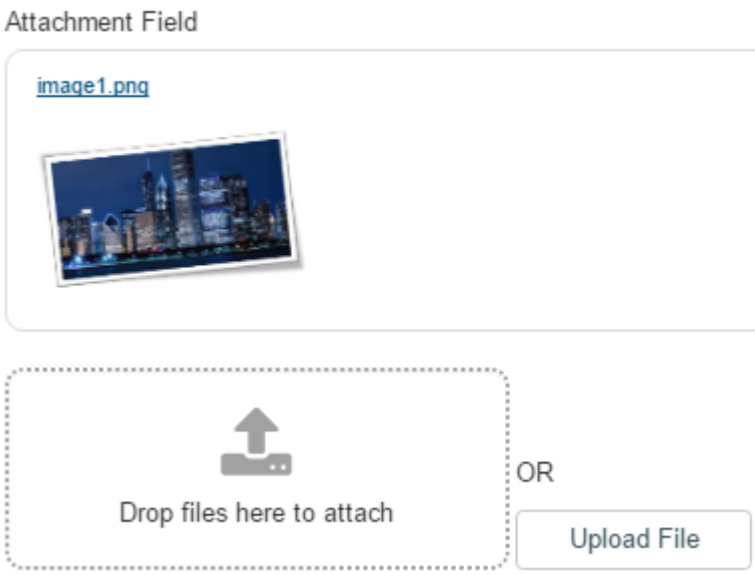


Image Preview Height (pixels)
Integer canvasHeight

Height for image to resize to in the preview mode.

Image Preview Width (pixels)
Integer canvasWidth

Width for image to resize to in the preview mode.

Transact Field | 20.11 This feature was removed in 20.11

Image Source
Option imageSource

Select the image source

The following options are available from the **Image Source** dropdown:

Label	Value
Camera	camera
Library	library

Maximum Dimension (pixels)
Integer maxDimension

Original attachment to be resized to this dimension.

Multi Select Attachments
Boolean multiSelect

Ability to multi select attachments

NOTE

You can also use the Attachment Field v2 widget that has additional functionality. | 22.10 This feature was introduced in 22.10..

Next, learn about the Countries component.

Card Content Template Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

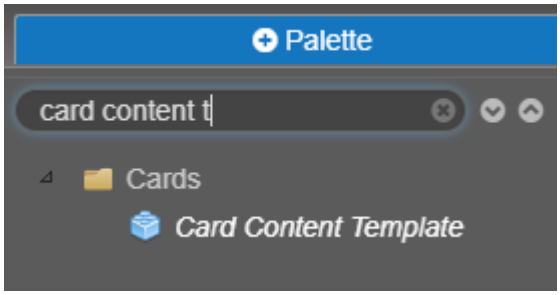
Maestro provides you with the Card Content Template component to simplify the process of collecting card details and representing these details on a card display. A card is a type of a layout that collects and displays details about an applicant or form user. You can use this component to:

- Add one or several card details to a form.
- Select and edit card details.
- Add advanced card details by adding multiple components to the template.

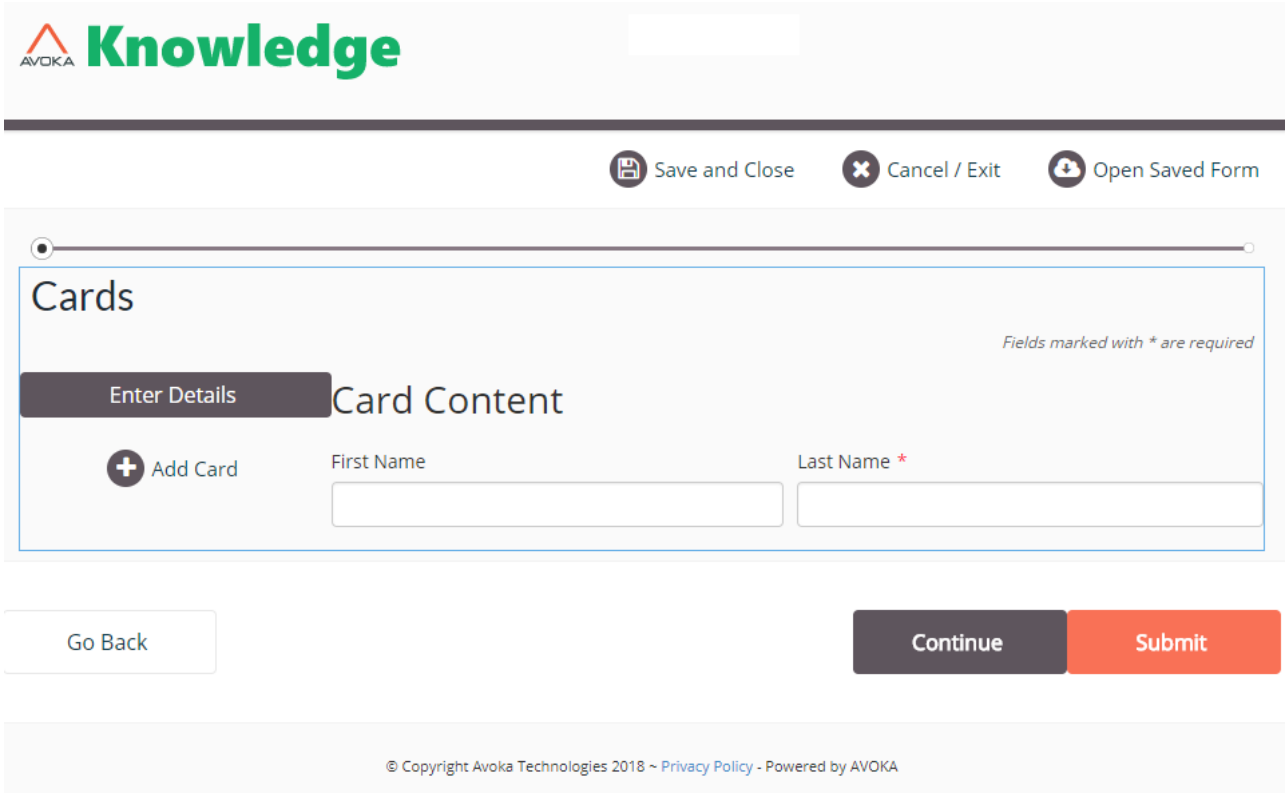
The Card Content Template is used to build the details that make up a card. For cards-based layout forms, the Card Content Template is the recommended starting point as it allows form users to enter details for one or many cards. Once entered, the Card Content Template provides form users with the ability to navigate between different card details for selecting and editing purposes. The Card Content Template is quite similar to the Repeating Block Template in that it allows multiple instances of the same set of components to be added to a Maestro form.

To configure the Card Content Template component:

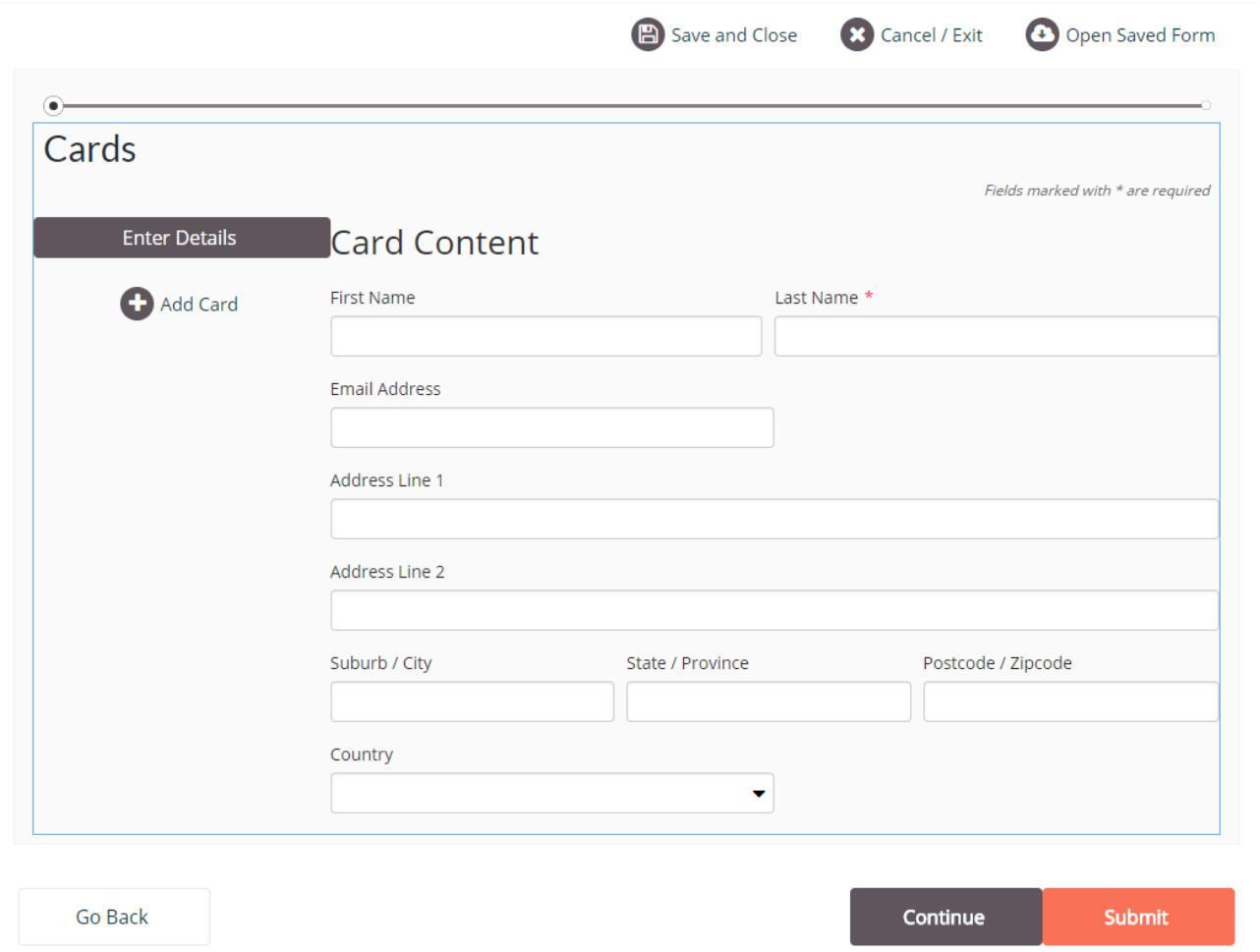
1. Open a Maestro form.
2. Click the Palette tab.
3. Expand Cards to see all available Cards library.



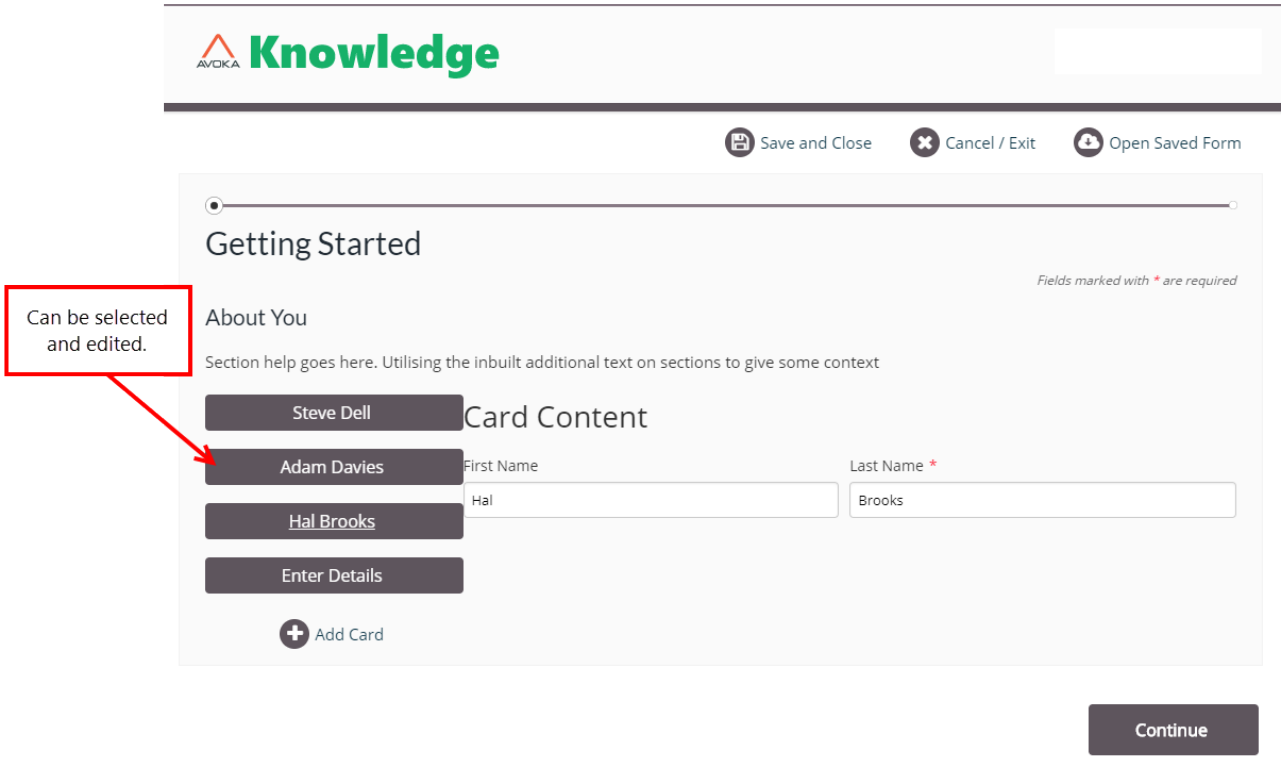
4. Click the Card Content Template. This component combines the functionality of both the Card Selector and Card Content components.
5. Drag and drop the component on your form. The example below shows the Card Content Template with two text fields.



6. Add several components if required. The example below shows the Card Content Template with multiple components.



7. Edit the component's details. The Enter Details tab displays the first and last name of each card entered. This is the default behavior, but this can be changed by updating the labels.
8. Click Add Card to enter details for another card. You can repeat this process as many times as required, but not exceeding the maximum number of repeats that is configured in the Properties panel.
9. Click Save.
10. Click Build and Render Form to see how this rule is displayed when the form is built and rendered. In the example, details for three users (three cards) are collected. Each of these options can be selected and edited as needed. If a form user selects a component in the Card Content Template, and then saves and resumes, the component that was selected before the form was saved is still selected when the user returns to the form.


Knowledge
Save and Close
Cancel / Exit
Open Saved Form
Getting Started
Fields marked with * are required
Can be selected and edited.
About You
Section help goes here. Utilising the inbuilt additional text on sections to give some context
Steve Dell
Adam Davies
Hal Brooks
Enter Details
Card Content
First Name
Last Name *
Hal
Brooks
Add Card
Continue

Checkbox Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

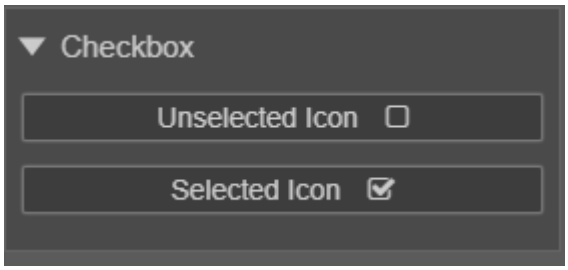
Maestro provides you with the Checkbox component to add a checkbox to the form.

To configure the Checkbox component:

1. Open a Maestro form.
2. Click the Palette tab and expand Input Fields.
3. Select Checkbox and drag and drop it on the form.



4. Click the Properties tab and find the Checkbox section.



5. Select Unselected Icon when the checkbox is not selected.
6. Select Selected Icon when the checkbox is selected by the user.
7. Click Save.
8. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Currency Field

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Currency Field component (widget), which you can use in several scenarios.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Currency Field** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Currency Field component:

Currency Icon Icon currencyIcon
Select the currency icon

Max Leading Digits Integer maxLength

Decimal Places Integer places
Number of decimal places to display

Accessibility

Placeholder Text Text placeholder

Help Text

Popover Help Text
Help Text helpText

Options

Prevent Negative
Boolean preventNegative

Decimal Point
Text decimalPointChar

Thousands Separator
Text thousandsSeparator

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Next, learn about the Decimal Field component.

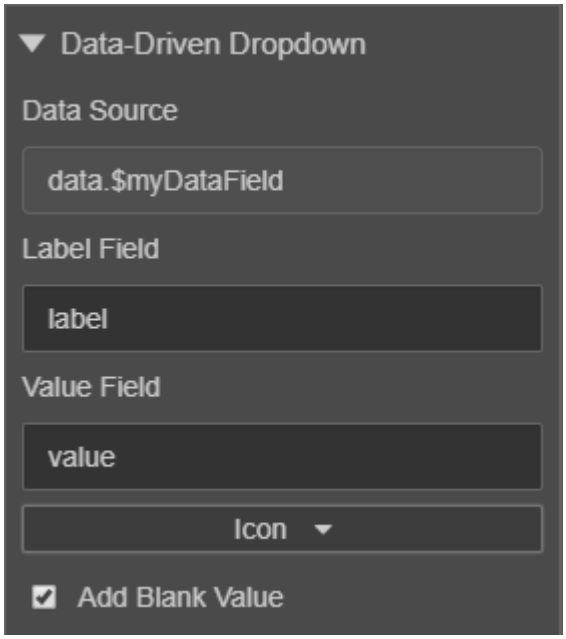
Data-Driven Dropdown Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

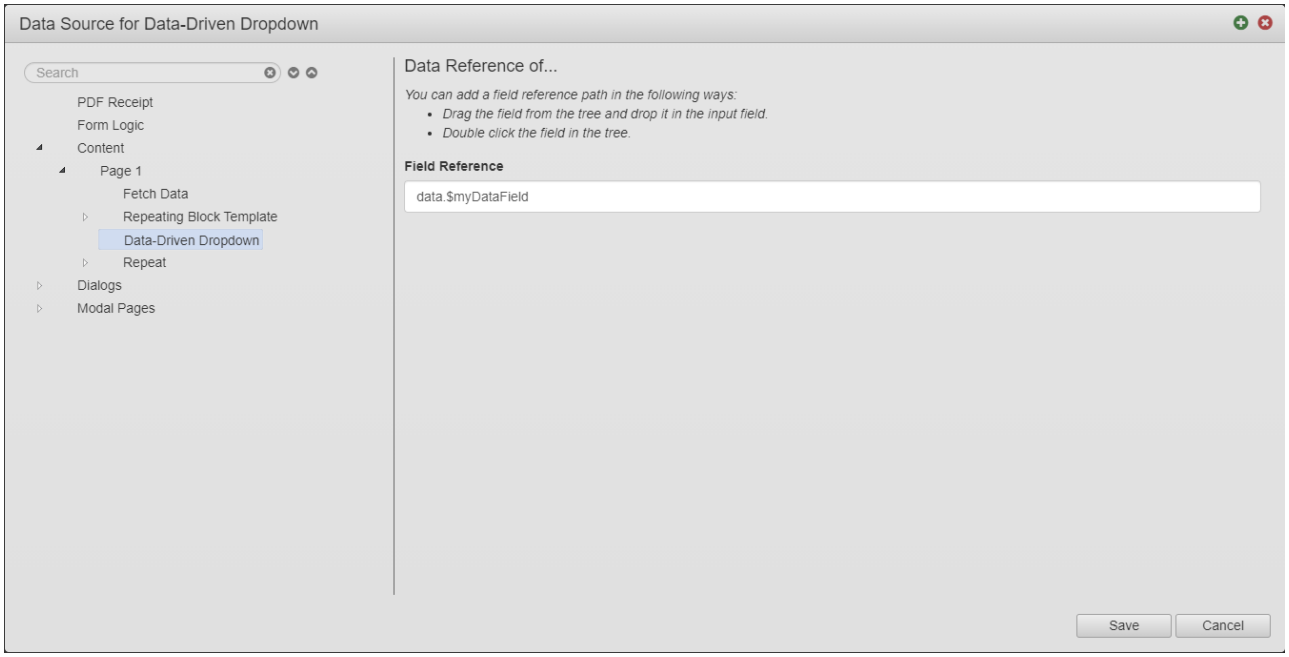
Maestro provides the Data-Driven Dropdown component that allows you to build dynamic selection lists to implement various customer requirements.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the Data-Driven Dropdown component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane under the Data-Driven Dropdown section.



5. Check the Data Source field that holds a path to source data, which is a key to the data to be shown in the dropdown lists. Click the field to edit the path.



For more information, see use cases below.

6. Update Label Field which is set to the field in each array object to use as a label. It should not be blank. The default is label.
7. Update Value Field which is set to the field in each array object that contains the value to include in submission XML for that option. It should not be blank. The default is value.
8. Icon allows you to select another icon to indicate a dropdown. Ask the Template Designer if you can change this icon, as icons are typically customized in the templates.
9. Select the Add Blank Value checkbox to add a blank value to the dropdown list.
10. Click Save.
11. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Use Cases

There are several ways to configure the Data-Driven Dropdown component, but the basic steps are as follows:

- Add a data field which stores your initial data. It is Product List In our example.
- Add your Data-Driven Dropdown and configure it to point to a relevant JavaScript object.
- Add the JavaScript script to your form to create this object using the string in your Data Field.

More generic steps are shown below:

- Add a script which creates a map.
- Add a component which uses the map.
- Add a data field to store initial data, for example:

```
[
   {
      "label":"Single investor",
      "value":"singleSuper",
      "brand":"Corporate Bank"
   },
   {
      "label":"Joint investor",
      "value":"joint",
      "brand":"Corporate Bank"
   },
   {
      "label":"Single investor (non-super)",
      "value":"singleNonSuper",
      "brand":"Corporate Bank"
   },
   {
      "label":"Company investor",
      "value":"company",
      "brand":"Corporate Bank"
   },
   {
      "label":"Product A",
      "value":"p3",
      "brand":"Retail Bank"
   },
   {
      "label":"Product B",
      "value":"p4",
      "brand":"Retail Bank"
   },
   {
      "label":"Product Z",
      "value":"p5",
      "brand":"Retail Bank"
   }
]
```

Where the `label` and `value` pairs correspond to the Data-Driven Dropdown properties. It also includes additional property, `brand`, which may be used later on., but for now, consider it optional.

Tip: The following online tool is great for converting an excel file to JSON format:

http://www.csvjson.com/csv2json

When you copy and paste columns, just select a "tab" separator. Ensure that your columns have titles.

Adding a script which creates a map:

```
data.$MccCodes = [
  {
    "MCC": 742,
    "Description": "VETERINARY SERVICES",
    "label": "742 VETERINARY SERVICES"
  },
  {
    "MCC": 763,
    "Description": "AGRICULTURAL CO-OPS",
    "label": "763 AGRICULTURAL CO-OPS"
  },
  {
    "MCC": 780,
    "Description": "HORTICULTURAL AND LANDSCAPING SERVICES",
    "label": "780 HORTICULTURAL AND LANDSCAPING SERVICES"
  },
  {
    "MCC": 1520,
    "Description": "GENERAL CONTRACTORS",
    "label": "1520 GENERAL CONTRACTORS"
  }
];
```

Adding a Data-Driven Dropdown, and configuration

After dragging on a "Data-Driven Dropdown" component, configure the following:

Choose a name for the Data-Source. It needs to be unique, and you can ignore the $ symbol, it's a convention, not syntactical. We will use this name later, and in this example has been set as "data.$productData"

The Label Field and the Value Field need to map to keys in your JSON (not required to edit anything if you have used the template above)

Adding a script which creates an object out of your Data Field

At some point in form execution (normally on page transition or shoehorned into a validation script or calculation script on an unrelated field), you need to add some code to convert your field value into an object.

Do not add the code to a click script on the Data-Driven Dropdown, as the execution order of the script & other events will trip you up.

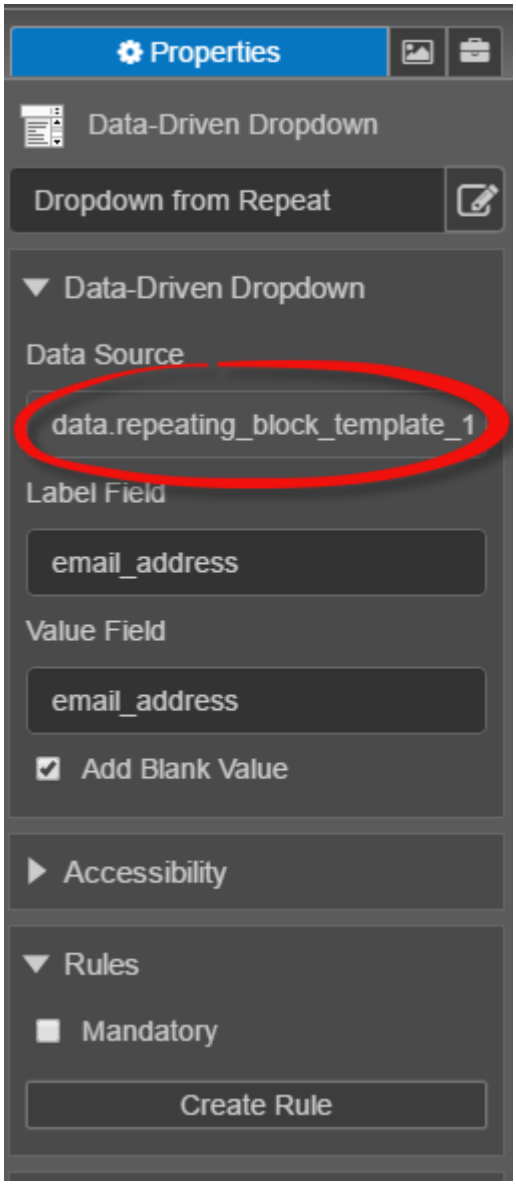
```
// data.productList is the field which contains the JSON string with keys and values
// data.$productData is the object which the Data-Driven Dropdown uses
// brand is an optional key in the JSON string
// data.tenant is a field which contains the name of current brand, and in this example
is prefilled into the form by services on form render.

var tempData = JSON.parse(data.productList );
data.$productData = tempData.filter(function(arrValue,arrIndex,arr) {
  if (arrValue && arrValue.brand && arrValue.brand.toLowerCase().indexOf(data.ten-
ant.toLowerCase())!=0) {
    return false;
  }
  return true;
});
return undefined;
```

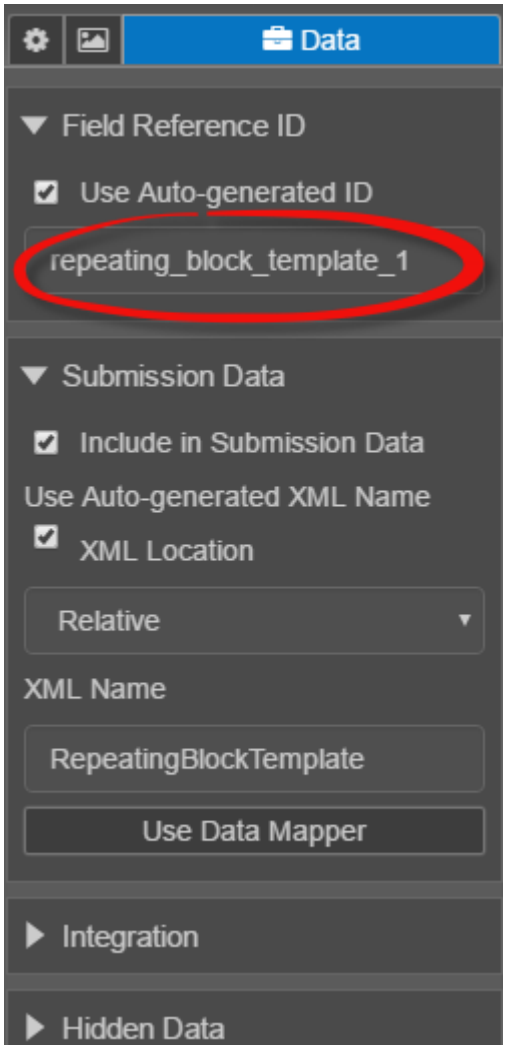
Data-Driven Dropdown Component Based on a Repeating Component

To configure a dropdown option for each item in a repeating component:

1. Configure the Data Source of the Data-Driven Dropdown to the data node for that repeat component.
2. Configure the Label and Value properties to the appropriate field IDs within the repeat.



3. The data source references the repeating block.
4. The Label to be shown in the dropdown will be found in the field email address.
5. The value that is included in the submission is also that of email address.
6. Configure the repeat component.
7. The data tab of the repeat displays the referenced id.
8. Within the repeat, the email_address field is referenced.



Data-Driven Dropdown Component Based on a Range Component

NOTE

You can also use this example for other generated sequence.

An array such as a range, for example years from 1900 to 2020, can be initialized as a transient data value in the Form load event, so it can be used as the Data Source for a data-driven dropdown. You can use this helper method to generate a range of values:

```
Calc.rangeSelection(10, 20)
```

Data-Driven Dropdown Component Based on Text Input

You could base the dropdown values on a text value such as an input field for testing, or the result of a dynamic data service call by converting the text to an appropriate array format. You can

use this method, depending on data format, which takes a string in Composer-style comma-and-pipe delimited format and produces an array suitable for data-driven dropdowns:

```
Calc.delimitedSelection(data)
```

NOTE

Each option is delimited by a comma, and within the option, the Label and Value are delimited by a pipe.

Cascading Dropdowns

You can create cascading dropdowns by setting the Data Source of one to be the top level of an array, for example from a JSON source, and using its Change rule script to set a transient data value to an array nested within the top-level object. This transient data value then forms the Data Source for the second data-driven dropdown.

Next, learn about the Date Picker component.

Decimal Field

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Decimal Field component (widget), which you can use in several scenarios.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Decimal Field** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Decimal Field component:

Max Leading Digits
Integer maxLength

Decimal Places
Integer places

Number of decimal places to display

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Options

Prevent Negative
Boolean preventNegative

Decimal Point
Text decimalPointChar

Thousands Separator
Text thousandsSeparator

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

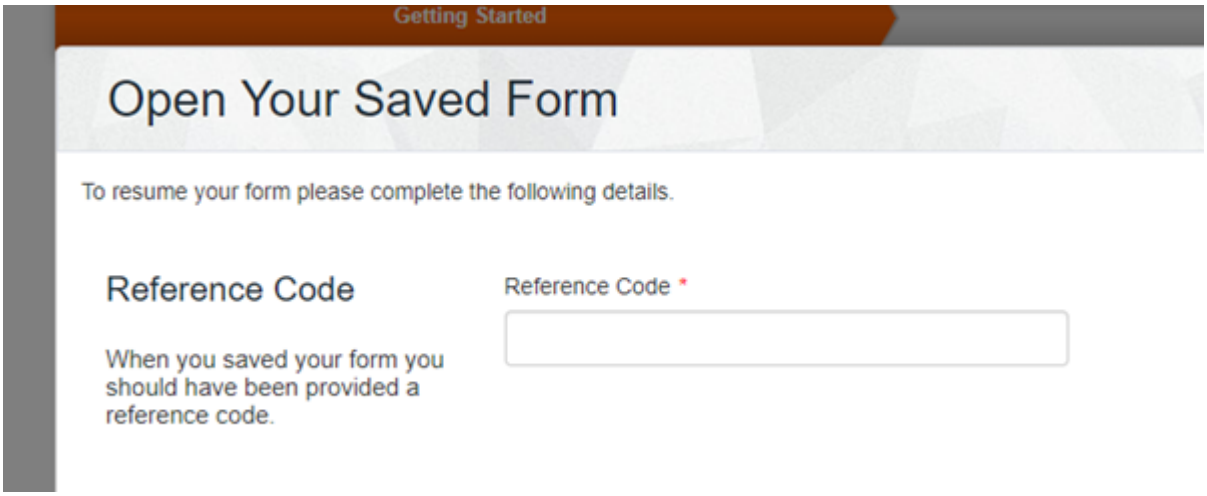
Next, learn about the Percentage Field component.

Dialogs Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Dialogs are pop-up windows that appear within the current page, in front of other content. Typically, they're used to capture information that might not need to be captured from every user.

For example, sometimes users save a form and return to it later, but many users complete a form in a single session. For this reason, saving a form is often handled within a dialog. If a user wants to save a form, they can do so, but if not, the user won't be presented with the option.



Dialogs can contain most of the components available to the form, but minimizing the amount of components in a dialog is recommend as they're a smaller, more condensed interface elements. If you place too many components inside a dialog, it can appear cramped and overwhelming.

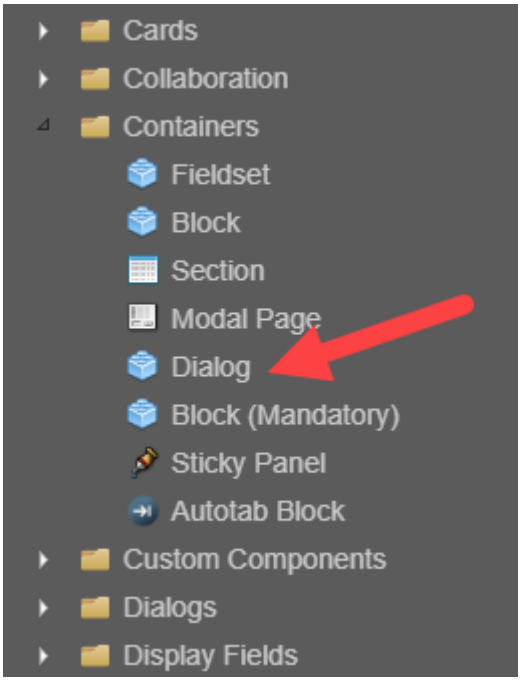
NOTE

Some components cannot be inserted into a dialog, such as modal pages or other dialogs.

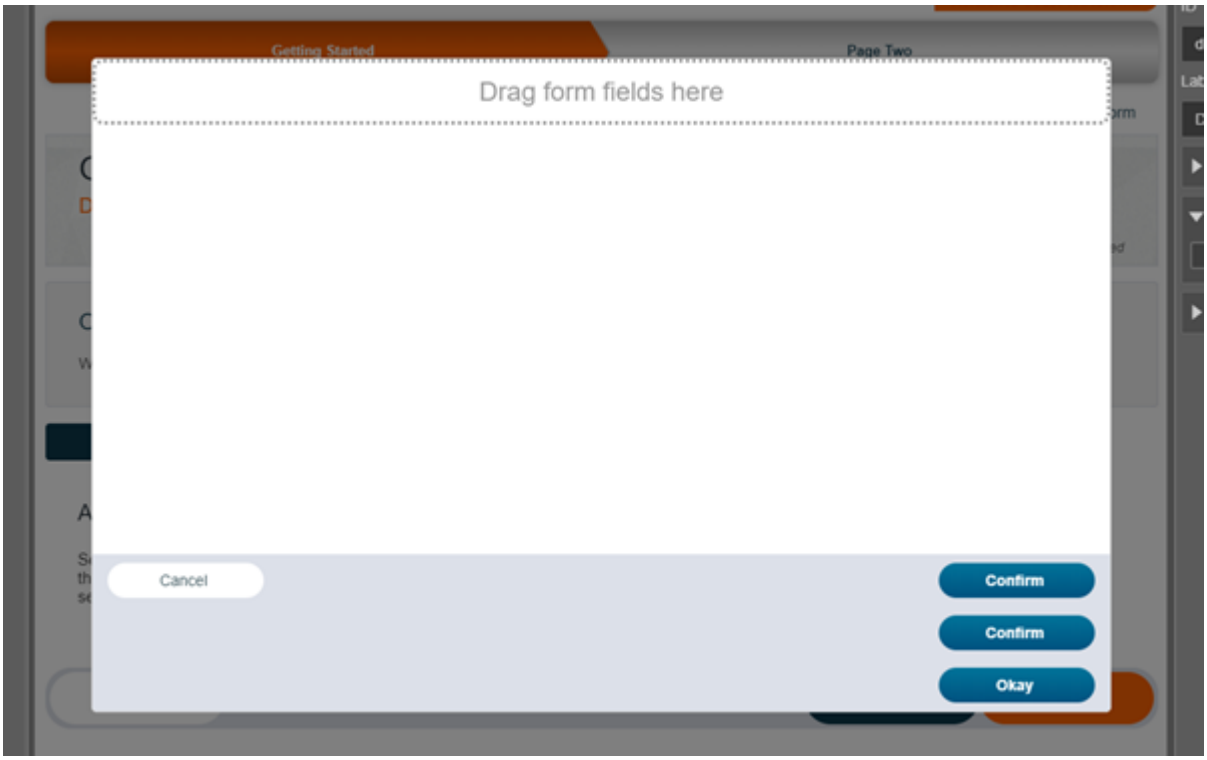
Dialog Types

Out of the box, Maestro provides a range of different dialog components.

The most flexible type of dialog is the Dialog component. It's available via the Palette pane, inside the Containers folder.



This dialog enforces the least amount of default functionality.



Additional dialogs are available via the Dialogs folder. The exact range of available dialogs, however, will depend on the template the form or template is based on.

Maguire	Blue Steel
Save Confirm	Cancel Confirm

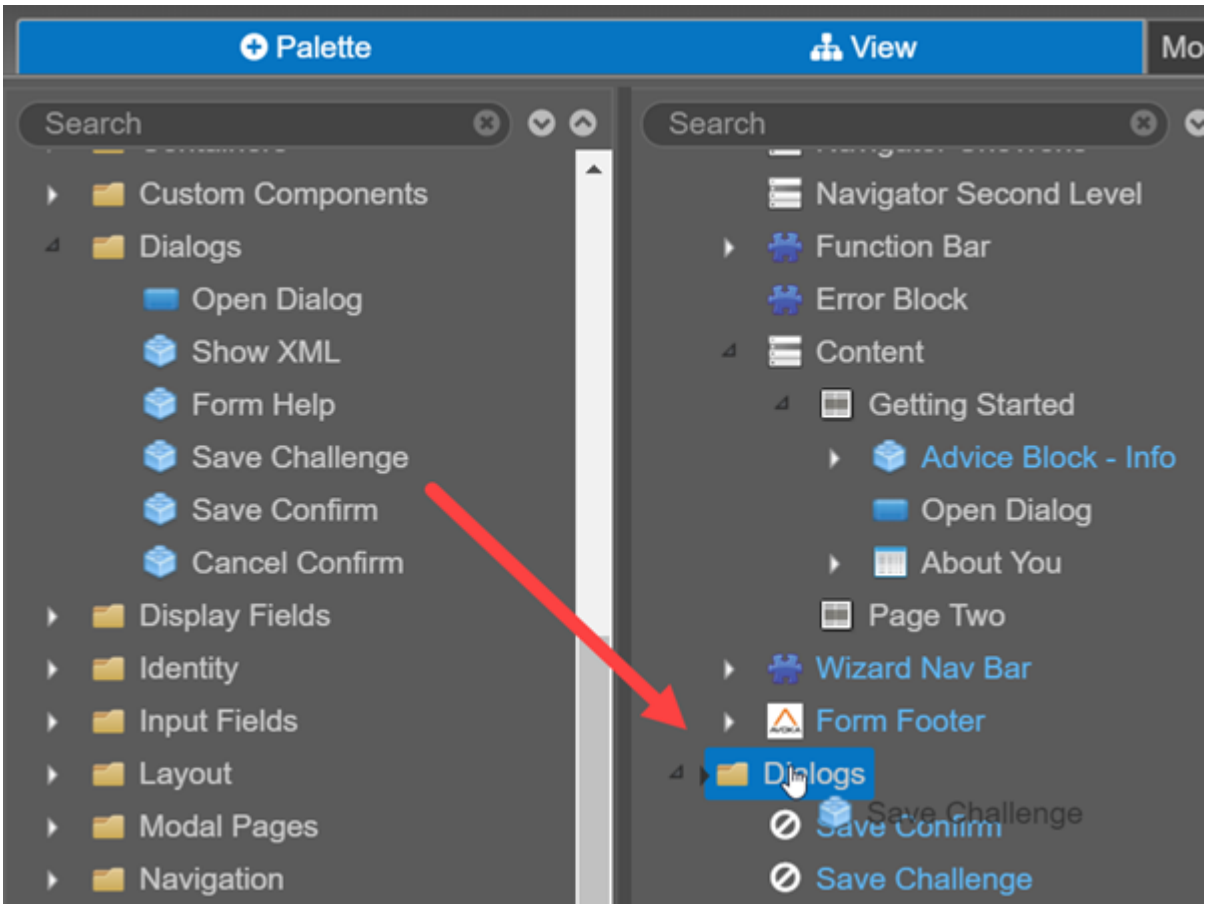
Save Challenge	Show XML
Cancel Confirm	Save Confirm
Form Help	Save Confirm Dialog
Show XML	Save Challenge Dialog
	Full Screen Menu
	Form Help

NOTE

Although different templates have dialogs with the same names, the dialogs themselves contain different content.

Creating Dialogs

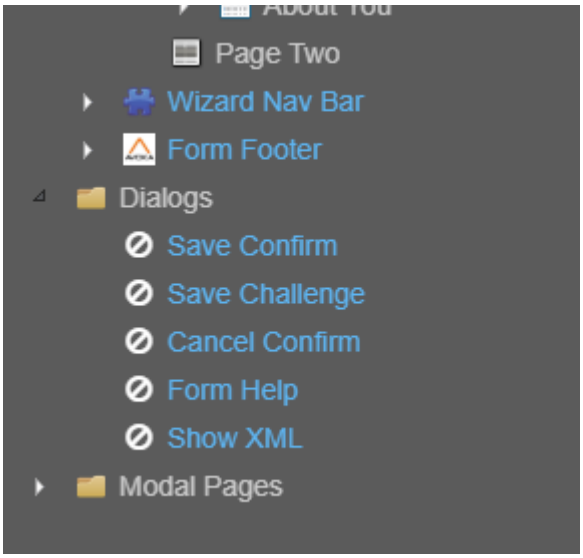
Unlike most other components, dialogs cannot be dragged directly into a form. Instead, they need to be dragged into the Dialogs folder, inside the Structure pane (also known as the **View** pane in earlier versions of Maestro).



Configuring Dialogs

To configure a dialog, simply select it. The dialog will appear within the form editor and its properties will appear in the Properties pane.

If a cross-out icon appears beside the name of a dialog, the dialog can only be edited via the template. If you want to configure this dialog but don't have access to the template, you will need to speak to the person who does.



Open a Dialog

To open a dialog within a form, use the `Form.showDialog` method inside a rule:

```
Form.showDialog("dialog_id_goes_here");
```

But replace `dialog_id_goes_here` with the ID of a dialog.

Typically, the opening of a dialog is triggered by a click rule.

To close a dialog, use the same method without any arguments:

```
Form.showDialog();
```

Next, learn how to use dialogs to implement a save and resume feature.

Fieldset

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Fieldset component, which is a container component, allowing you to group several fields together. This is especially useful when fields are not stand-alone and only make sense in the context of a group. For example, you should use the Fieldset component to group a credit card details, such as a name of a card holder, credit card number and expiry date, or multiple checkbox components, such as contact preferences.

Some components, for example, the Address, Date of Birth Dropdown, and Time, already have their fields grouped according to required functionality, so there is no need to place them into a Fieldset. If you do so, you only create some confusion about the hierarchy of the components. This also may prevent some assistive technologies[1], such as screen-readers, to correctly interpret the content of the form.

The Field is not just for aesthetic purposes, but also helps users with assistive technologies to navigate through the form.

NOTE

We recommend using this component to increase the accessibility of a form.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Fieldset** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.



[1]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.

4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Fieldset component:

Help Text

Label
Label label

Provide a meaningful description of this container identifying a group of fields. The label text (legend text) can be used by assistive technologies to inform the user of the context for fields within the Fieldset.

Popover Help Text
Help Text helpText

Provide a popover help text

Hide Label

Select Hide Label to suppress the label text, and use this component for grouping only.

Label Placement Left

Select Label Placement Left to align the label to the left. The label is aligned above the fields by default, which 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 label without having to move the cursor too far and reduces the chances of finding the wrong field by mistake.

For more information, see Maestro form accessibility.

Next, learn about the "Block" on page 775 component.

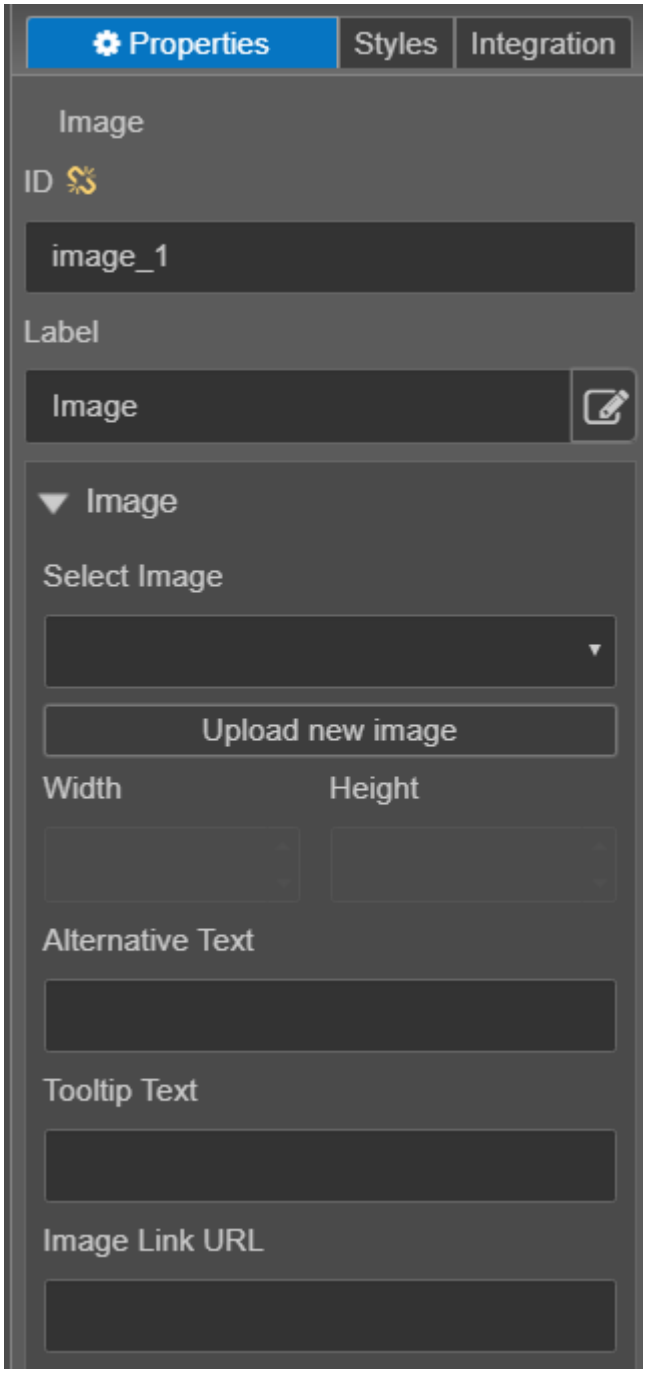
Image Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Image component that allows you to insert an image into a form.

To configure the Image component on a form:

1. Open a Maestro form.
2. Click the Palette tab and expand Display Fields.
3. Select Image and drag and drop it on the form.
4. Click the Properties tab and find the Image properties.



5. Use the Select Image dropdown list to select from images currently available in the library. You can see images from libraries available to this form, such as the project or organization library.
6. Click Upload new image to upload a new image that is not already in the library. You should always check the available images in the dropdown prior to uploading your own image.
7. Use Width and Height to resize the image you can enter the width and height that you want the image to be.

8. Alternative Text - The text entered is used for accessibility, particularly people with vision difficulties. There are two types of images, decorative and informative. Decorative images include separator lines or arrows for buttons. These types of images should not contain alternative text, therefore you can leave the option blank. Informative images include images such as logos or banners. These types of images should always contain alternative text that represents the image displayed.
9. Tooltip Text - The text entered will display on devices, with a mouse, when the mouse hovers over the image.
10. Image Link URL - If a URL is entered, a new browser tab opens linking to the Image Link URL when the user clicks the image.
11. Click Save.
12. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

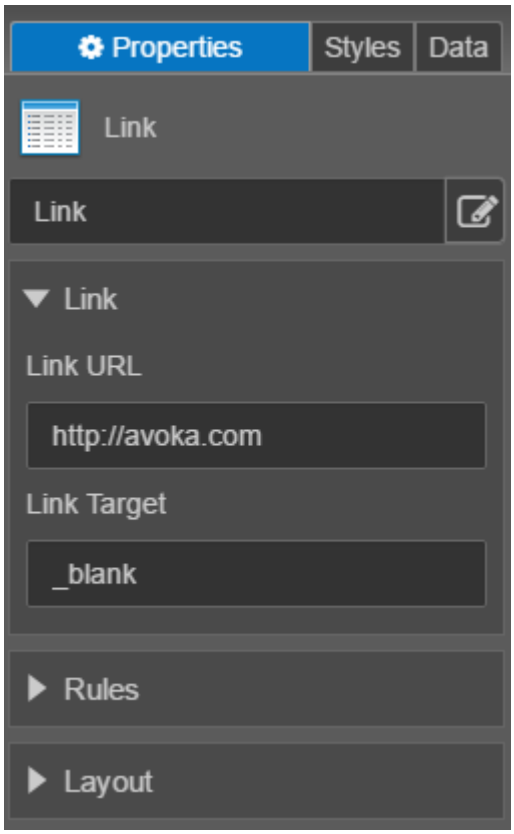
Link Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Link component that allows you to add a text link to a form. This can be used to redirect users to a URL away from the form, for example to direct users to websites that contain terms and conditions or privacy statements. However, when users navigate away from the form, there is always a chance that they won't return, so use this component with this information in mind.

To configure the Link component on a form:

1. Open a Maestro form.
2. Click the Palette tab and expand Display Fields.
3. Select Link and drag and drop it on the form.
4. Click the Properties tab.
5. Edit the following Link properties:



6. Label - the text that displays on the form, which users select to follow the link's URL.
7. Link URL - the URL for the link.
8. Link Target - defines where the link opens when users click on it. The options are:

- _blank- Opens the linked document in a new window or tab. This is the default.
- _self -Opens the linked document in the same frame as it was clicked.
- _parent - Opens the linked document in the parent frame.
- _top - Opens the linked document in the full body of the window.

9. Click Save.
10. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Modal Pages Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Modal pages are pages that can be made to appear at any point during the transaction experience. When opened, they appear immediately in front of the current page's content, similar to how dialogs behave. What makes them different from dialogs is that they fill the entire window, hiding all of the content behind them.

IMAGE

> NOTE
>
> Some components cannot be inserted into a modal pages, such as dialogs or other modal pages.

Modal Page Types

Out of the box, Maestro provides a range of different types of modal page components.

The most flexible type of modal page is the Modal Page component. It's available via the Palette pane, inside the Containers folder.

IMAGE

This modal page enforces the least amount of default functionality.

IMAGE

Additional modal pages are available via the Modal Pages folder. The exact range of available modal pages, however, will depend on the template the form or template is based on.

Maguire	Blue Steel
Cancel Success	Cancel Success
Submit Success	Submit Success
Save Success	Share Success
	Save Success

NOTE

Although different templates have modal pages with the same names, the modal pages themselves contain different content.

Create a Modal Page

Unlike most other components, modal pages cannot be dragged directly into a form. Instead, they need to be dragged into the Modal Pages folder that's, inside the Structure pane (also known as the **View** pane in earlier versions of Maestro).

IMAGE

Configure a Modal Page

To configure a modal page, simply select it. The modal page will appear within the form editor and its properties will appear in the Properties pane.

IMAGE

If a cross-out icon appears beside the name of a modal page, it can only be edited via the template. If you want to configure the modal page but don't have access to the template, you will need to speak to the person who does.

IMAGE

Open a Modal Page

To open a modal page within a form, use the `Form.showModal` method inside a rule:

```
Form.showModal("modal_id_goes_here");
```

But replace `modal_id_goes_here` with the ID of a modal.

Typically, the opening of modal pages is triggered by a click rule.

To close a modal page, use the same method without any arguments:

```
Form.showModal();
```

Multiline Text Area Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

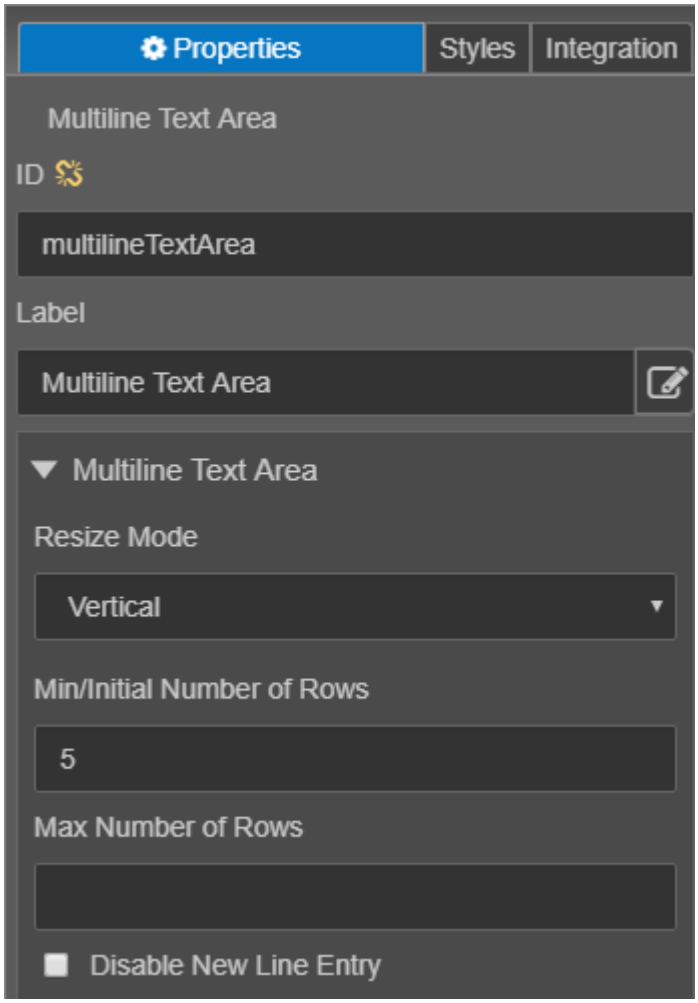
Maestro provides the Multiline Text Area component that accommodates large amounts of text entered by form users. You can configure it to either expand vertically or have fixed number of rows.

To configure the Multiline Text Area component on a form:

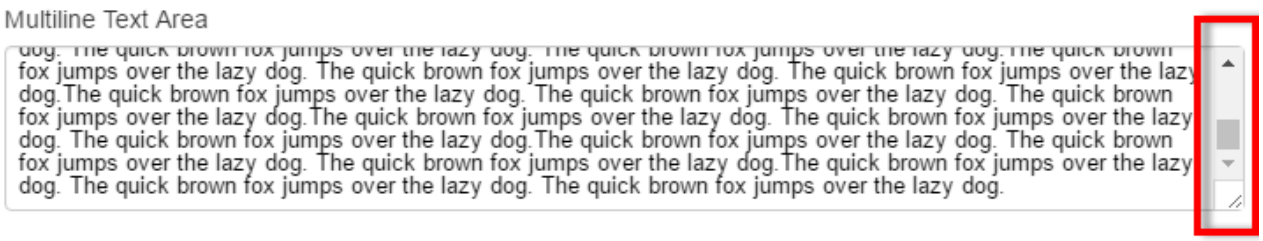
1. Open a Maestro form.
2. Click the Palette tab and expand Input Fields.
3. Select Multiline Text Area and drag and drop it on the form.



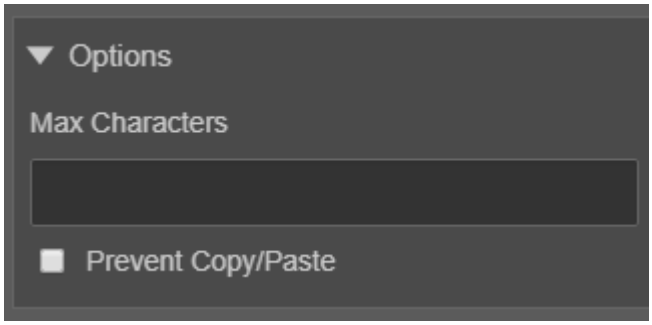
4. Click the Properties tab and find the Multiline Text Area section.



5. Resize Mode - Use the dropdown to select Vertical or Fixed. Vertical allows the component to expand vertically if the user enters text that fills more than the initial space. Fixed locks the multiline text area from expanding, no matter how much text the user enters.
6. Min/Initial Number of Rows - This is the starting side of the Multiline Text Area. You can determine the starting number of rows, which is the height of the entry box.
7. Max Number of Rows - You can limit the number of rows added based on the user's entry. If a max number of rows is determined, the multiline text area does not adjust the size, however the user is still able to enter additional text. A scroll bar and an option to expand the area will be displayed once the user passes the maximum number of rows.



8. Click the Properties tab and find the Options section.



9. Max Characters - allows you to set the maximum number of characters allowed in the multiline text area. Letters, numbers, special characters and spaces are included in the total number of characters.
10. Prevent Copy/Paste - Select this checkbox to prevent the user from copying and pasting to this field.
11. Click Save.
12. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

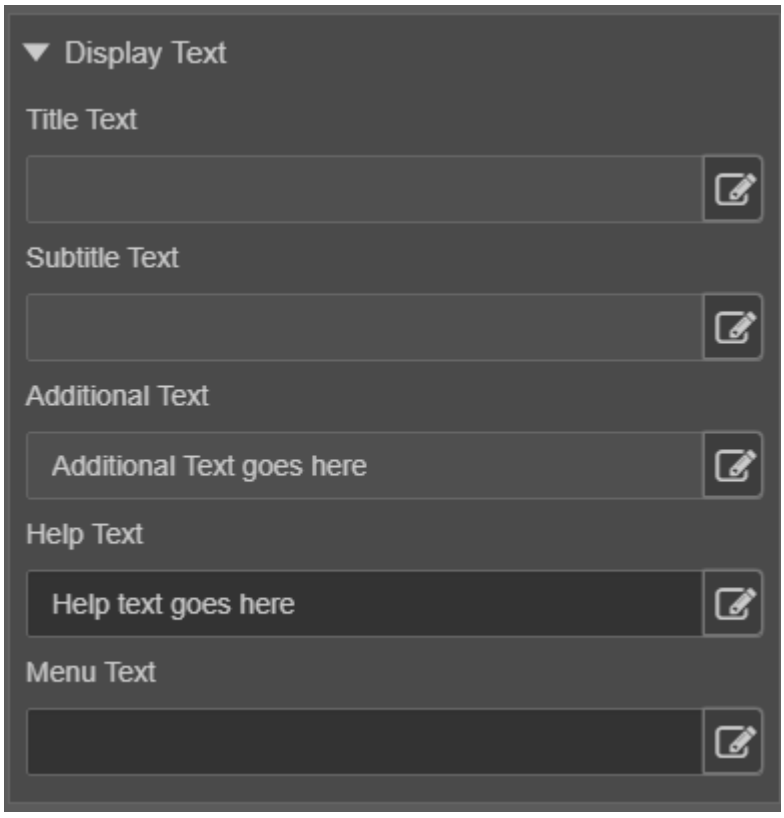
Page Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Page component that you can use to break up a longer form into multiple pages. When you add a page component to the form, the page name will also be added to the Navigator (if a Navigator is used in the form). You can remove the page from the Navigator if needed using the Properties panel. You can also navigate from one page to another by using the Continue and Go Back buttons at the bottom of the page (if used in the template). The Continue button is named as Submit on the final page, and the Go Back button is hidden on the first page.

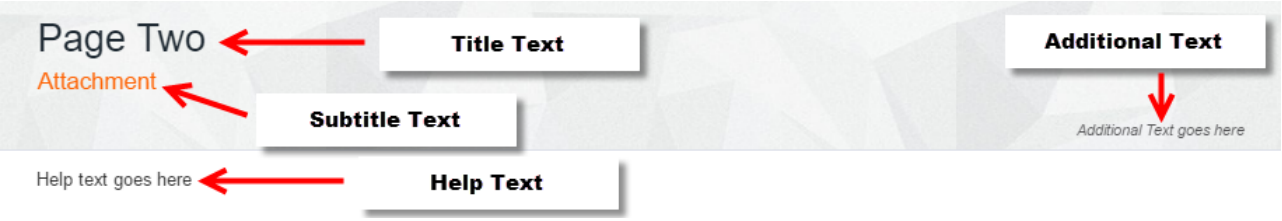
To configure the Page component on a form:

1. Open a Maestro form.
2. Click the Palette tab and expand Navigation.
3. Select Page and drag and drop it in the Content folder.
4. Click the Properties tab and find the Display Text section.

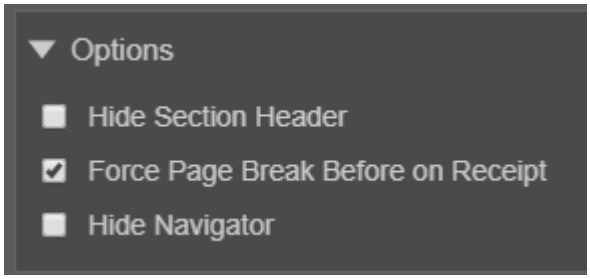


5. Title Text - displays the title of the page in the prominent way. It uses the page label, item.label, by default.

6. Subtitle Text - displays a sub-title for the page, which is set to the name of the form, Form.view.label, by default.
7. Additional Text - is displayed to the right of the page heading, if the value is provided.
8. Help Text - is displayed under the page heading, if the value is provided.
9. Menu Text - if a page has a long name, you can include a shortened version of the page name as the Menu Text.



10. Click the Properties tab and find the Options section.



11. Hide Section Header - hides the entire section header, including Title, Subtitle and Additional Text.
12. Force Page Break Before on Receipt – select it to force a new page on the printed or receipt version of the form.
13. Hide Navigator – select it to hide the navigator on this page. You can still navigate using the Go Back or Continue buttons. This can be useful for smaller forms that don't need a navigator, or for pre-condition pages that display prior to the Navigator being shown.
14. Click Save.
15. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Next, learn about Percentage Field.

Percentage Field

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Percentage Field component (widget), which you can use in several scenarios.

NOTE

This component inherits properties and shared styles from the Decimal Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Percentage Field** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Percentage Field component:

Max Leading Digits
Integer maxLength

Decimal Places
Integer places

Number of decimal places to display

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Options

Decimal Places
Integer places

Number of decimal places to display

Prevent Negative
Boolean preventNegative

Decimal Point
Text decimalPointChar

Thousands Separator
Text thousandsSeparator

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Next, learn about the "Data-Driven Dropdown" on page 815 component.

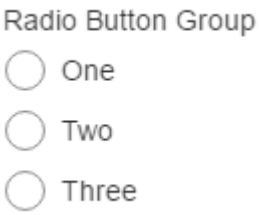
Radio Button Group Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

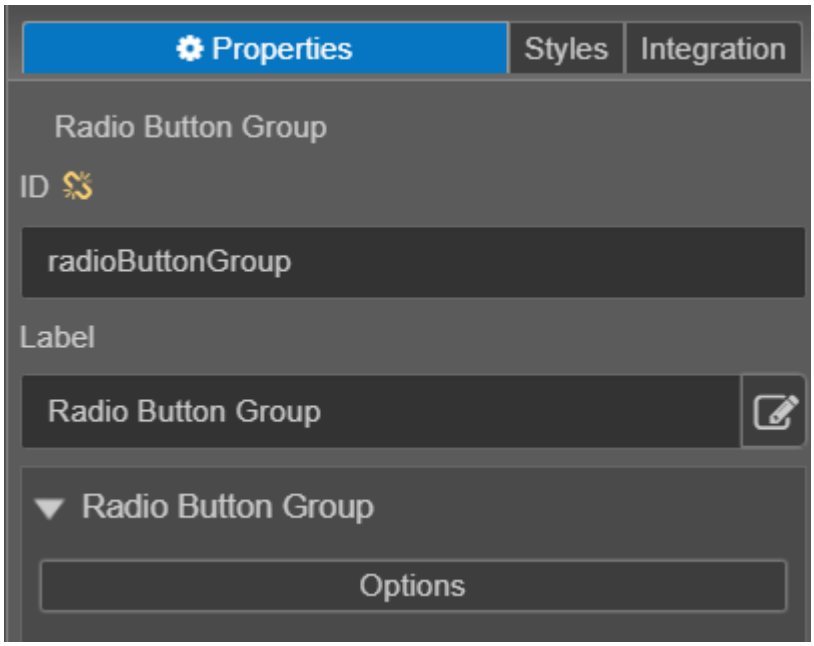
Maestro provides the Radio Button Group component that allows you to add a group of radio buttons to a form.

To configure the Radio Button Group component on a form:

1. Open a Maestro form.
2. Click the Palette tab and expand Input Fields.
3. Select Radio Button Group and drag and drop it on the form.



4. Click the Properties tab and find the Radio Button Group section.

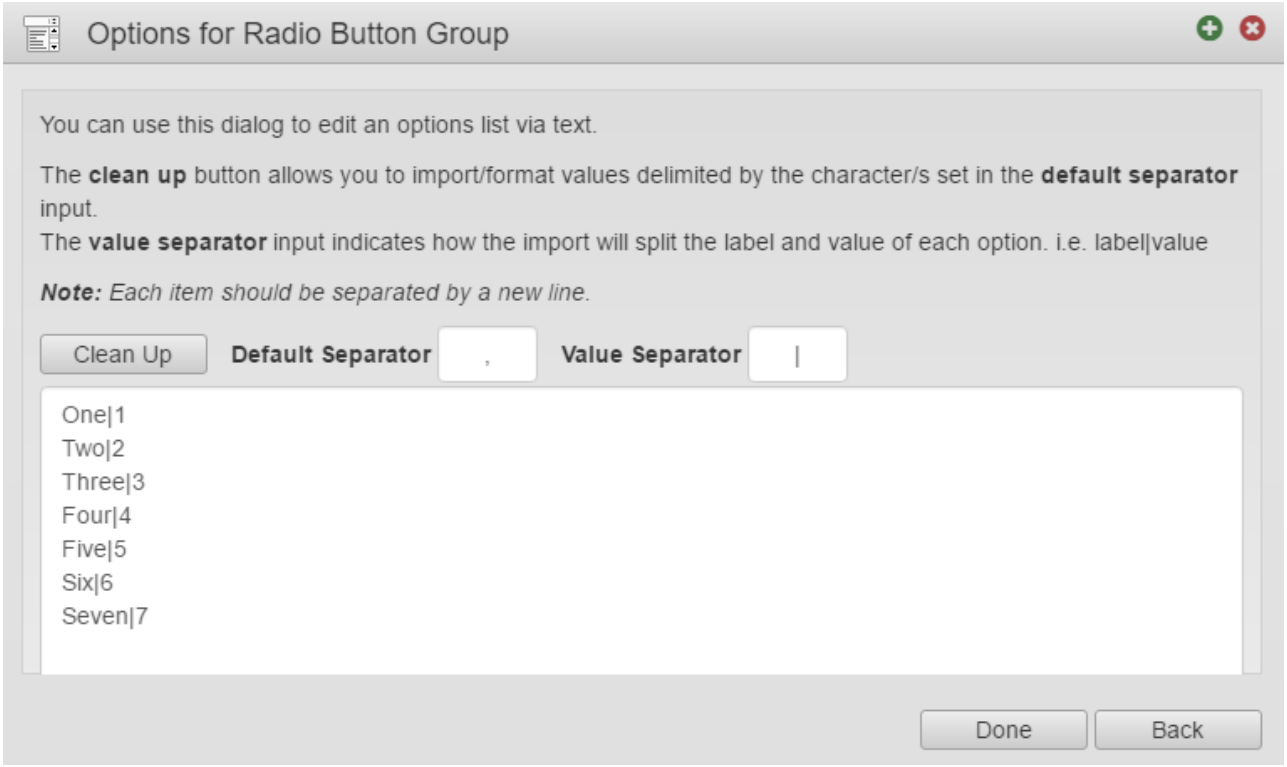


5. Options - allows you to enter each radio button option: radio button label, value and default. Each option added will be a radio button that the user can select. It displays the Options for Radio Button Group window. This is where you enter the options available to

the user.



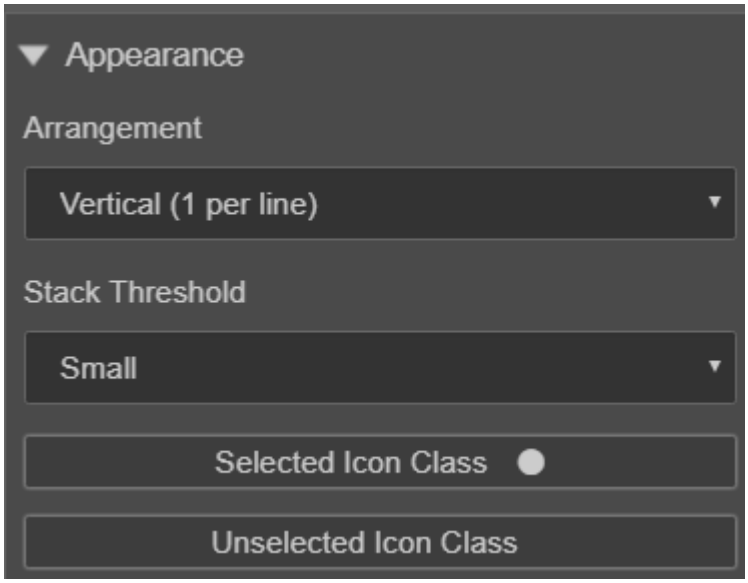
- Label - displays on the form next to the radio button.
- Value (if different) - Used for data collection and in scripts or rules based on the user's selection.
- Default - Select the check next to the option you want to be selected by default. If there is not a default selection, leave the checkboxes unchecked.
- Add - This button will add options to the radio button group. Once you add an option you will need to complete the Label, Value (optional) and Default (optional).
- Move Up / Move Down - to rearrange the radio button group options. Select the option you want to rearrange then click the Move Up or Move Down button.
- Edit As Text – allows to view and edit the radio button options in a text editor window as shown below. Using this view, allows you to copy text from an external source, such as an Excel spreadsheet, and paste it in here.



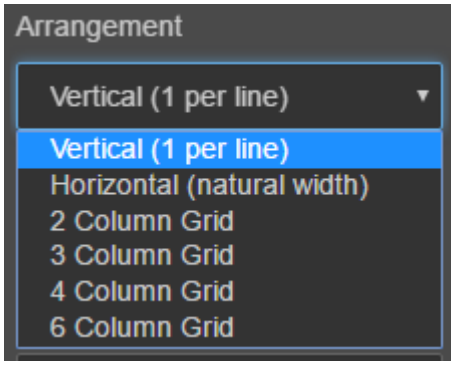
- You can select which Default Separator and Value Separator to use from the following options: One|1, Two|2, Three|3, etc. or each option can display on a separate line. This is summarized in the table below:

Item	Default Separator	Values on Separate Lines
With a Value (Value Separator)	One|1, Two|2, Three|3	One|1 Two|2 Three|3
Without a Value	One, Two, Three	One Two Three

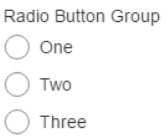
6. Click the Properties tab and find the Appearance section.



7. Click Arrangement to select how the radio button options are displayed on the form.



8. Stack Threshold -
9. Selected Icon Class -
10. Unselected Icon Class –
11. Appearance of the Arrangement Options is summarized in the table below:

Item	Arrangement
Vertical (1 per line)	
Horizontal (natural width)	

2 Column Grid



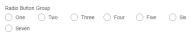
3 Column Grid



4 Column Grid



6 Column Grid



12. Click Save.
13. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

KEYBOARD SHORTCUT

Use the Tab key to iterate through the component and press Space key to select a radio button you want. Use the Left, Right, Up, and Down arrow keys to change the selection.

Graphical Radio Button

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Graphical Radio Button component (widget), which allows you to specify not only a text caption, but also an image from a predefined list of images or your own rich content. The Graphical Radio Button component replaces the usual radio button component. If you want to use a composite image, you can create a shared style for your Graphical Radio Button components and adding appropriate style rules to create and show this. You will need to factor in all the states so that the image is shown correctly when selected, not selected, hover, focus, active, disabled, error and so on. You can use different images for the selected and non-selected states, but we recommend using a single image.

NOTE

Graphical Radio Buttons are restricted to being used inside Graphical Radio Button Groups.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Graphical Radio Button** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Graphical Radio Button component:

Button Value Text value
Enter the value for this radio button

Appearance

Selected Icon Class Icon selectedIconClass

Selected icon class name

Unselected Icon Class
Icon unselectedIconClass

Unselected icon class name

Image

Button Image
Image src

Select from the available images resources, or upload a new one

Button Selected Image
Image srcSelected

Select from the available images resources, or upload a new one

Alternative Text
Text alternativeText

Type an alternative text for informative images, leave blank for decorative images.

Next, learn about the Graphical Radio Button Group component.

Graphical Radio Button Group

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Graphical Radio Button Group component (widget) so you can group Graphical Radio Buttons to present several mutually exclusive options on a form.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Graphical Radio Button Group** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Graphical Radio Button Group component:

Help Text

Popover Help Text Help Text helpText

Next, learn about the "Radio Button Group" on page 825 component.

Icon Button

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Icon Button component (widget).

NOTE

This component inherits properties and shared styles from the Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Icon Button** component within the **Buttons** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Icon Button component:

Icon Icon icon
Select the button icon

Icon Position Option iconPos
Select the button icon position The following options are available from the **Icon Position** dropdown: **Label Value** Left left Right right

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this button

Next, learn about the "Button Group" on page 755 component.

Image

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with an Image component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Image** component within the **Display Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Image component:

Select Image Image src
Select from the available images resources, or upload a new one

Alternative Text Text alternativeText
Type an alternative text for informative images, leave blank for decorative images.

Tooltip Text Text toolTip
This text will be added to the title attribute of the image.

Image Link URL Text imageLink
This url will be added as an anchor tag around the image. It will open a new tab.

Next, learn about Link component.

Link

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with an Link component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Link** component within the **Display Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Link component:

Link URL
Url linkUrl

Link Target
Text target

Next, learn about the "Percentage Indicator" on page 806 component.

Repeating Block Template

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

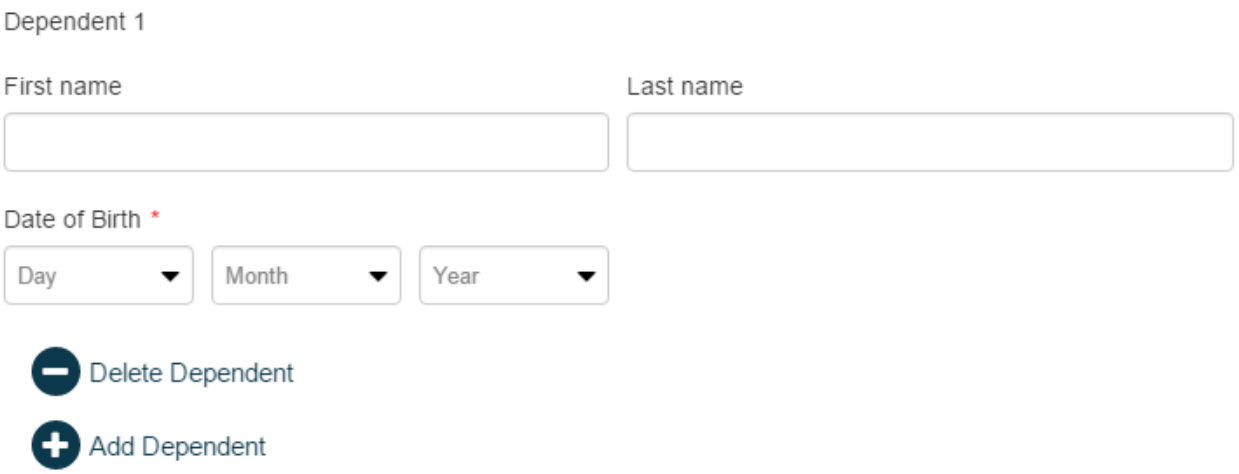
Maestro provides the Repeating Block Template component to allow users to add multiple instances of an item to a form. This is a useful when the number of instances varies for each form user, for example, when you need to add first and last names of all family members.

To configure the Repeating Block Template component on a Maestro form:

1. Open a Maestro form.
2. Select Repeats from the Palette.
3. Select Repeating Block Template and drag it onto the form.



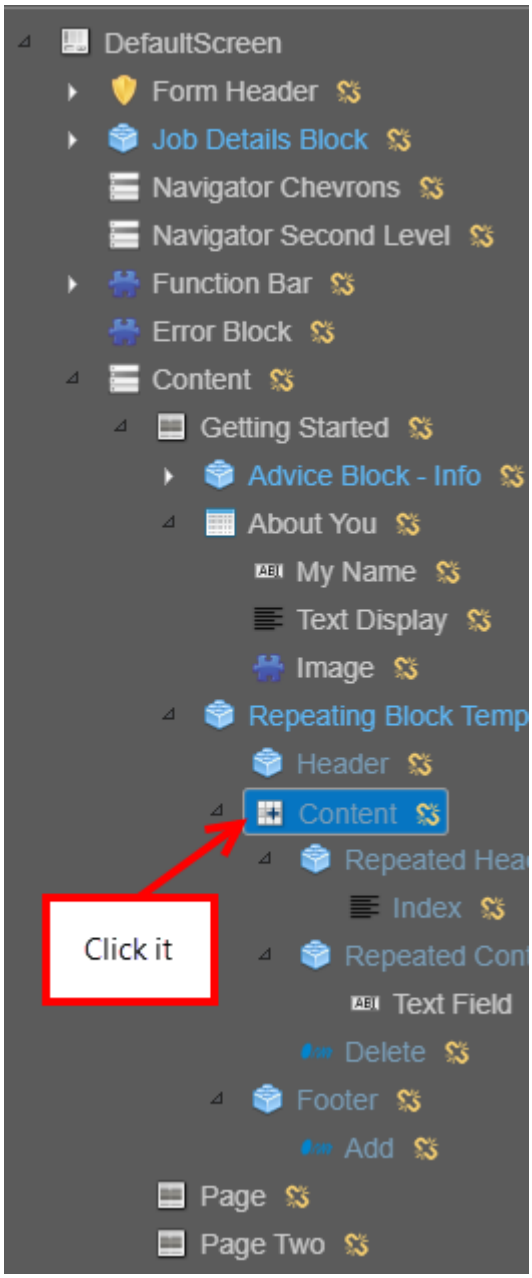
4. Add components to be repeated to the Repeating Block Template component, as shown below.



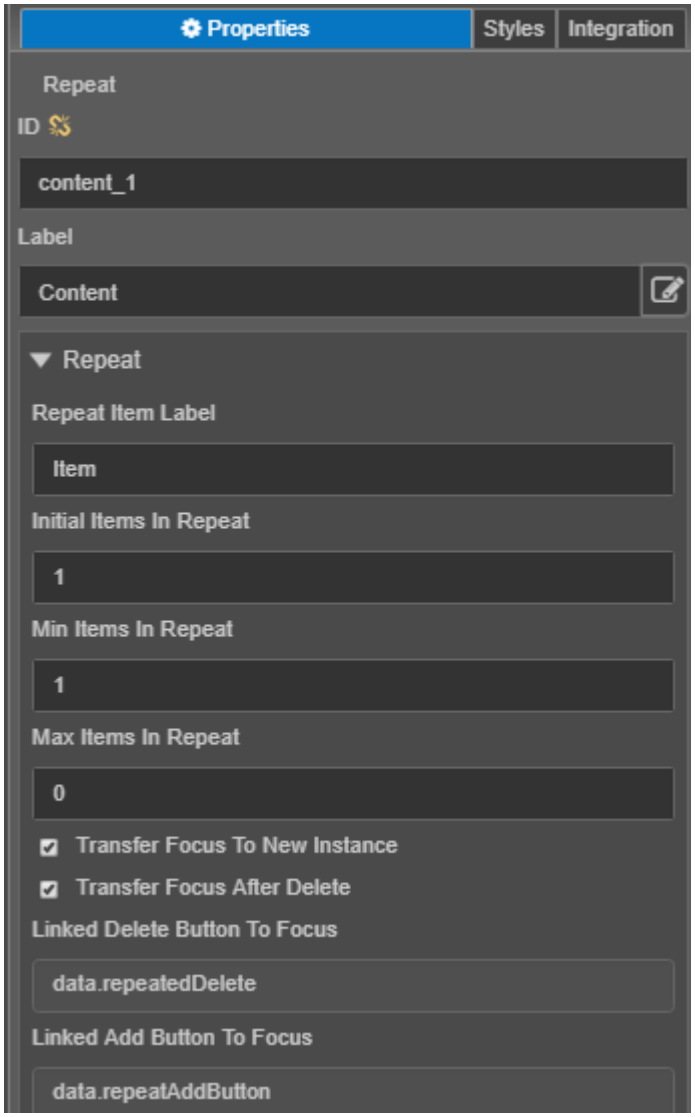
The Repeating Block Template is made up of several different components. It is important to customize the correct part of the repeating block template depending on what you are

trying to achieve. For example, when items are added to the Repeating Block Template, they are added to the Repeated Content block. However, if you would like to change the Repeat Item Label or the Min or Max items in the repeat, you will need to select the Content block.

5. Select the Content of the Repeating Block Template to edit it.



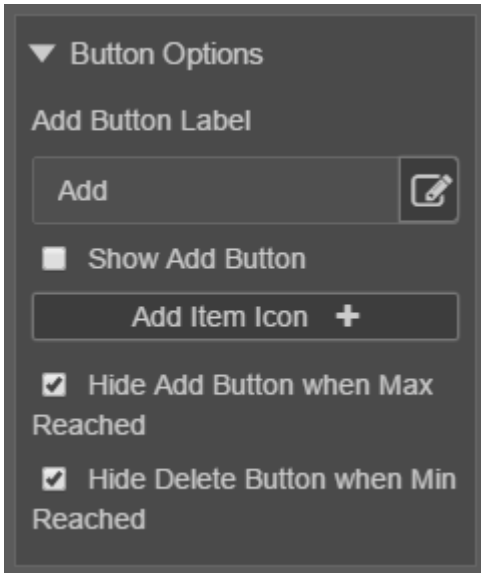
6. Click the Properties tab.
7. Edit the Repeat properties.



- Repeat Item Label - changes the heading within the repeating section and the button labels.
- Initial Items in Repeat - the starting number of repeating items, which defaults to 1. Change it if required, but there must have at least one item in the repeat block.
- Min Items in Repeat - The user can add or remove items. This is the minimum number of items that must be displayed on the form. You must have at least one item. Once the minimum is reached, the Delete button can disappear.
- Max Items in Repeat - You set a maximum number of repeating items for the form user. Once the maximum is reached the Add button can disappear. Enter zero to indicate that there is no maximum number of items.
- Transfer Focus To New Instance - When the user adds a new instance/item, the focus will be on the first component of the new instances/items.

- Transfer Focus After Delete - When the user deletes an instance/item, the focus will be on the first component of the previous instance/item.
- Linked Delete Button To Focus - When the user deletes an instance/item, the focus will be on the delete button of the previous instance/item in the repeat.
- Linked Add Button To Focus - When the user deletes all instances/items in the repeat, the focus will be on the remaining add button. This setting is only relevant where the Min Items in Repeat is set to 0.

8. Edit the Button Options properties.



- Add Button Label - allows you to change the wording on the Add button. The default is "Add" plus the Repeat Item Label that is “Add {{item.properties.itemLabel}}”. Click Edit to edit the button text in the editor, which allows you to use properties from other elements of the repeating block.
- Hide Add Button when Max Reached – select it to hide the Add button when the Max Items In Repeat is reached. This is selected by default.
- Hide Delete Button when Min Reached – select it to hide the Delete button when the Min Items In Repeat is reached. This is selected by default.

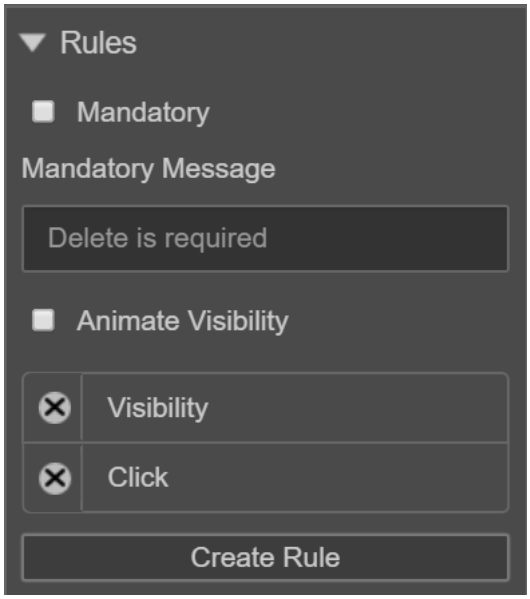
9. Edit Rules to implement specific customer requirements.
10. Click Save.
11. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Hide Delete Button for a Single Item in the Repeating Block Template

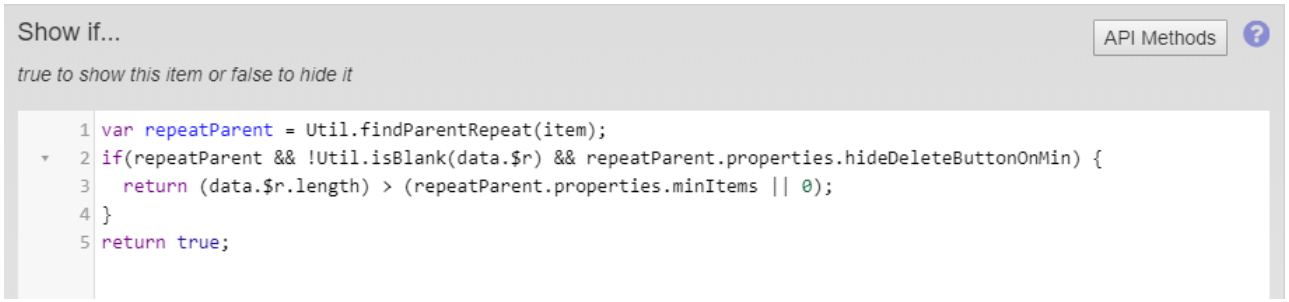
Each time a new item in a repeat block is created, Journey Maestro creates an Add Item button and a Delete Item button on the item. These buttons are used to allow the form user to continue to add items until they have entered all items that are required of them. However, there may be times where a Form Builder doesn't want to give the form user the option to delete an item until a certain validation rule has passed. The Form Builder may also only want the delete button to appear on certain repeat items.

To hide a delete button for a single item in a Repeating Block:

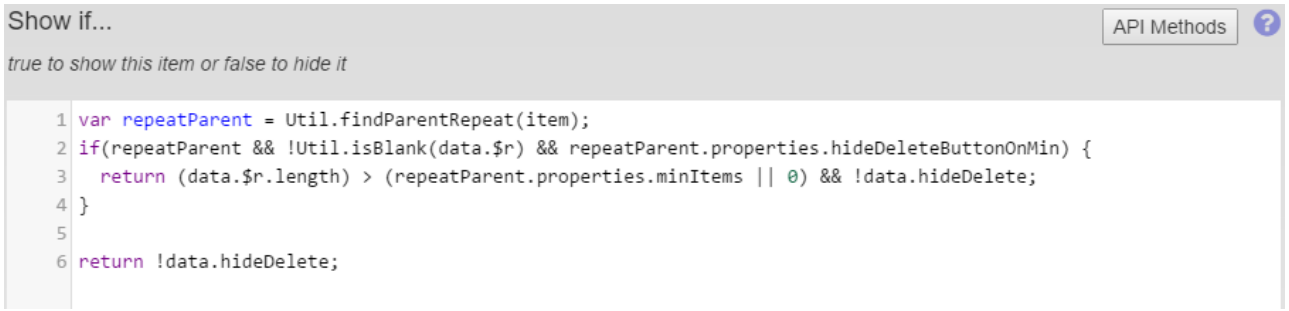
1. Configure the repeating block template component.
2. Select the Delete Item button from the Repeating Block. Delete Item buttons in the Repeating Block Component have their own visibility rules that can be modified to create expanded validation and sometimes extra functionality.
3. Click the Properties tab and expand Rules of a default Delete Item button.



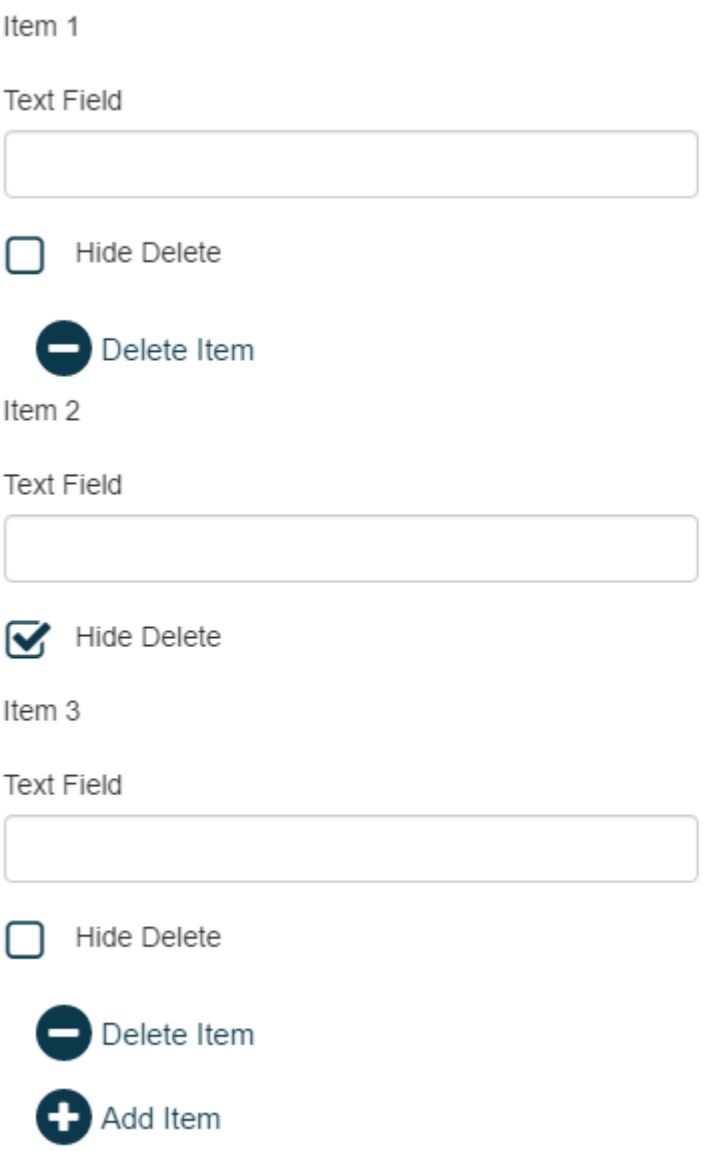
4. Click Visibility to edit the rule. This rule is created by default for all Delete Item buttons in a Repeating Block. To hide the delete button, you need to add additional logic to this rule.



5. Edit the JavaScript rule. In this example, we use a checkbox to decide which instances of the repeat block will display a delete icon button. When the Hide Delete checkbox is selected, the Delete Item button is not displayed for the following item, as shown below.



6. Click Save.
7. Click Build and Render Form to see how this rule is displayed when the form is built and rendered. The example below displays this visibility rule applied to a Repeating Block when the Hide Delete checkbox has been selected after the first Text Field is displayed and the result of this selection is that the Delete Item button is removed from the second item.



Next, learn about the section component.

Save Challenge Dialog

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

You need to configure the Save Challenge Dialog with the same security question as the Save Confirm Dialog. The Save Challenge Dialog is displayed when the user returns to complete a saved form.

The Save Challenge includes a section to enter the generated reference code that displays when the user saves their form. It also includes the security question section. This is where the user needs to enter the same answer entered when the form was saved.

When configuring the components within the two dialogs, you will need to ensure that the XML Name is different. By default, if the label is the same, the XML Name will also be the same. This will cause a duplicate binding error when you publish your form. You can decide to only change one component XML Name or change both component XML Names.

Often, you may need to use a custom Save Challenge Dialog. For a form to use the new custom dialog instead of the default `savechallenge` dialog, you can add the following JavaScript rule on form load:

```
if( window.location.href.toLowerCase().indexOf('savechallenge=true') > 0 ){
  maestro.Form.showDialog('data.customDialog'.split('.').pop());
}
```

Where `customDialog` is your custom dialog ID.

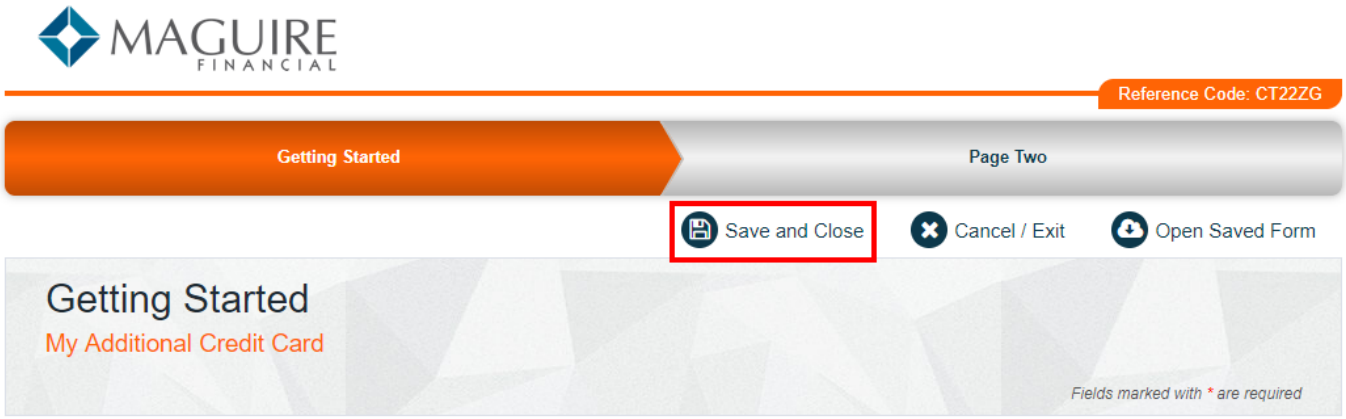
You also need to configure Journey Manager to enable the Save Challenge feature.

Next, configure security question for save challenge.

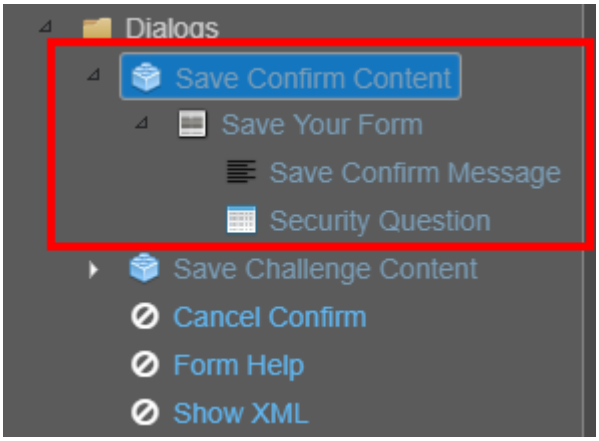
Save Confirm Dialog

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Save Confirm Dialog is displayed to form users when they click Save and Close to save the form and close it as shown below.



Maestro uses this page to collect the answers to the configured security challenge questions. Security questions, which are presented when the user saves a form, are configured from the Save Confirm Dialog itself as shown below.



Extension points must be configured in the form's template in order for the save confirm dialog to function in a meaningful way.

Next, learn about the Section component.

Select Language

MaestroThe UI design product. | Form Builder | 22.04 This feature was updated in 22.04.

Maestro provides the Select Language component (widget), which allows form users to select a language that they want to use on a form. You must configure a language or multiple languages for a form before a user can select them.

NOTE

This component inherits properties and shared styles from the Dropdown component.

Usage

1. Open the **Palette** pane in the Maestro editor.
2. Locate the **Select Language** component within the **User Information** folder.



3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.
5. Configure the component's styles via the **Styles** pane.

Properties

The following properties are available to the Select Language component:

Populate using Form translations Boolean useFormLanguages
Use the translations in the Form to populate the list

Sort Options Boolean sortOptions
Check to automatically sort options by label

Hide on single language
Boolean hideSingleLanguage

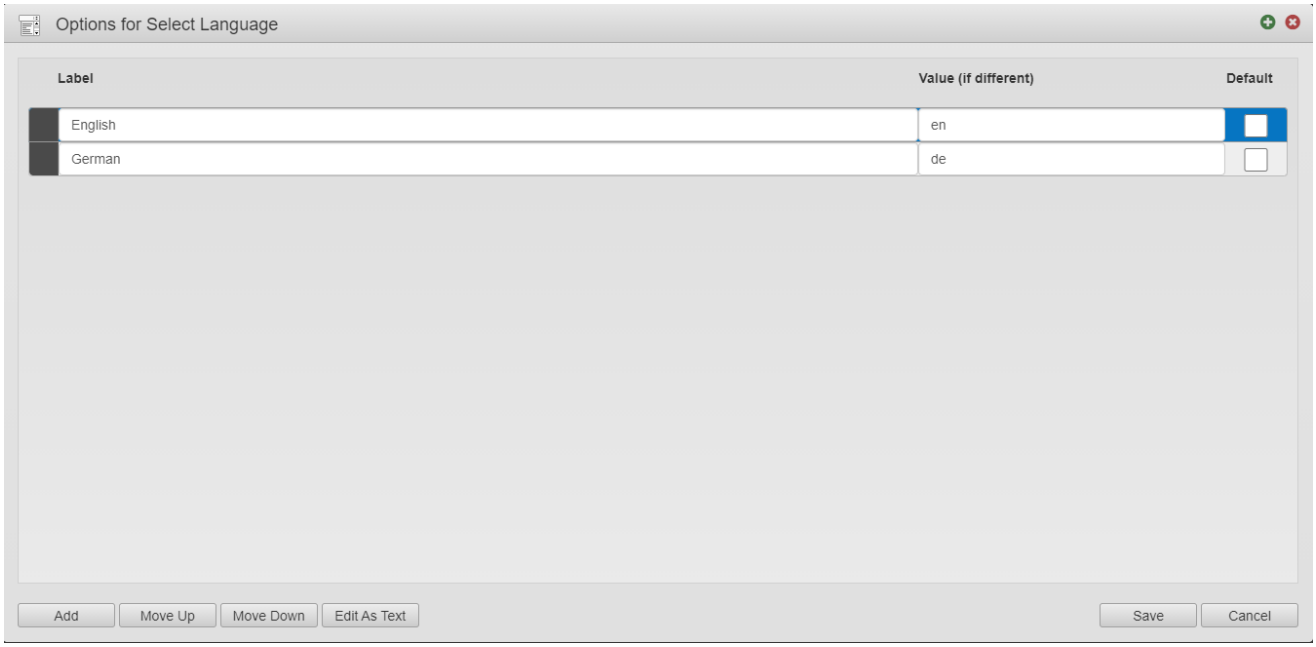
Hide this widget when the form only has a single language

Options
List options

The options list for the dropdown

By default, the **Options** property contains the following values:

Label	Value
Language name 1	Locale code 1
Language name 2	Locale code 2

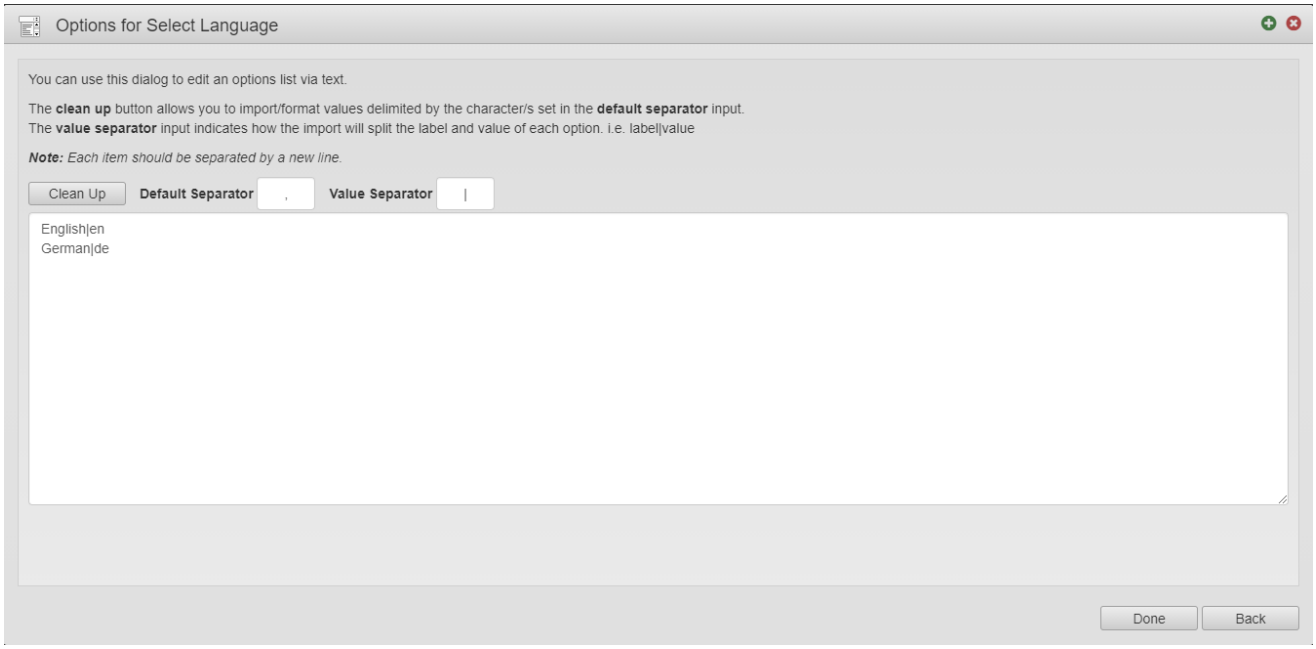


Update the language details or select the Default checkbox to make a selected language a default one and click Save.

Click Add to add a new line to the list where you need to add a language name and a locale code.

Select a language and click Move Up or Move Down to re-order its position in the list of languages.

Click Edit as Text to see the language configuration as a text file. You can change the separators used in the text.



Sort Options
Boolean sortOptions

Check to automatically sort options by label

Icon
Icon icon

Select the dropdown icon

Add Blank Value
Boolean addBlankValue

Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Styles

The following styles are available to this component:

Avalon Language Selector

```
form[name='form'] {
  .avalon-language-selector {
    width: auto;
    margin-right: 20px;
    .wdg-label {
      display: none;
    }
    .av-select-overlay {
      display: inline-block;
      float: right;
      width: 150px;
      background-color: transparent;
      border: 0;
      border-radius: 0;
      padding: 0;
      min-height: 30px;
      outline: none;
      box-shadow: none;
      cursor: pointer;
    }
    .wdg-input {
      .av-select-value {
        right: 25px;
        padding: 0;
        font-family: @ux-default-font-family;
        font-size: @ux-default-font-size;
        color: @ux-default-font-color;

      }
      .wdg-select-icon {
        padding: 0;
      }
      &:hover, &:focus {
        .av-select-value {
          text-decoration: underline;
        }
      }
    }
  }
}
```

After you've configured the component, you can preview the form to switch between the available languages to verify it works as expected.

Next, learn about the "Attachment Field" on page 671 component.

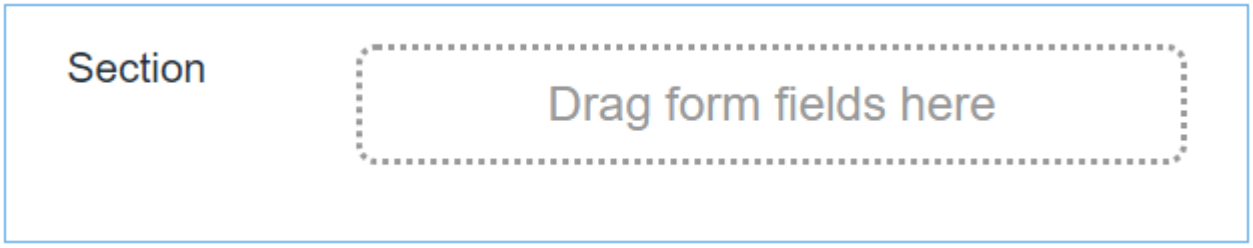
Section Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

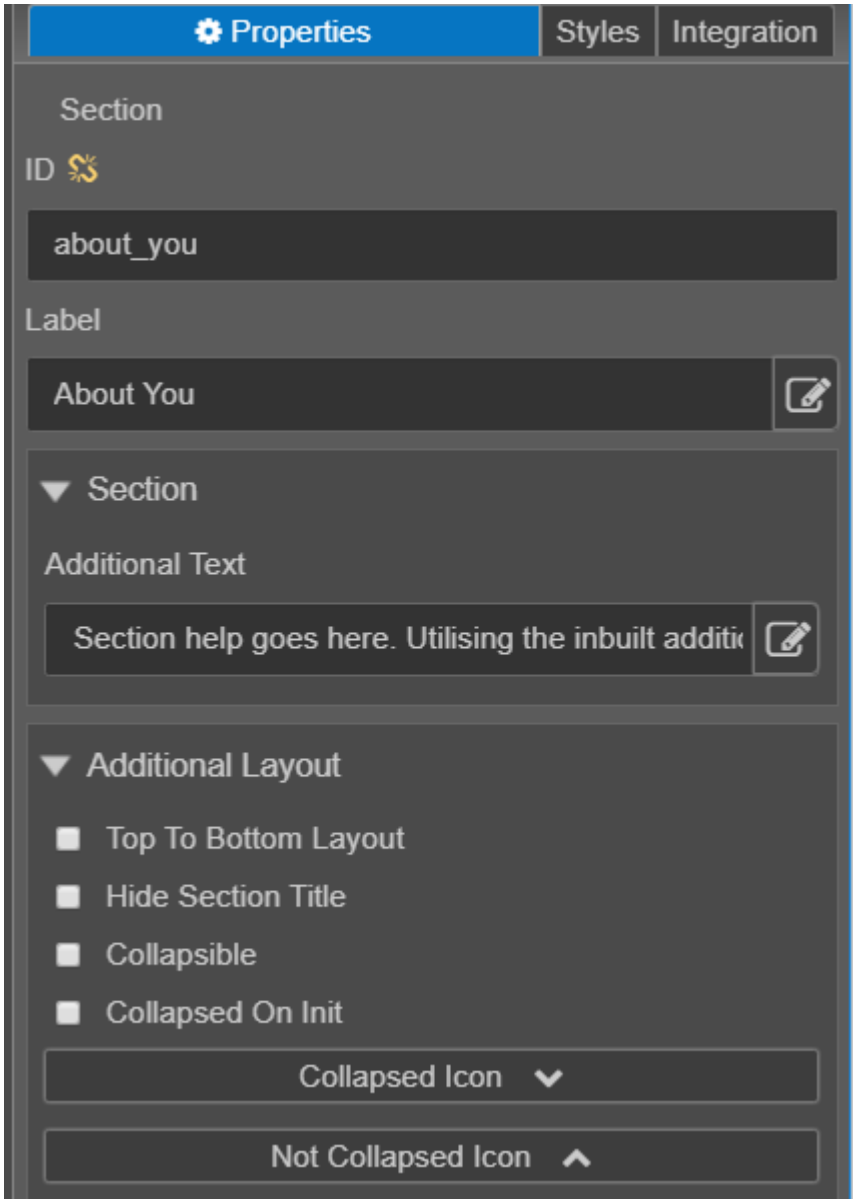
Maestro provides the Section component that allows you to break up a long page with many fields into logical sections. A Section has a section header and an optional explanatory text, plus it may have additional features, which allow uses to change the default layout, collapse, or and expand it.

To configure the Section component on a form:

1. Open a Maestro form.
2. Click the Palette tab and expand Containers.
3. Select Section and drag it onto the form.



4. Click the Properties tab and find the Section section.



5. Section Label displays at the top of the section on the form. The screenshot below shows section details:



6. Additional Text displays below the header within the section.

7. Select Top To Bottom Layout to change the layout of the section so that the header and additional text display above the components, rather than to the left.
8. Select Hide Section Title to hide the section title so that it isn't displayed on the form. The screenshot below shows a section with Hide Section Title enabled:



9. Select Collapsible to enable a user to collapse and expand the section on the form. The screenshot below shows a section with Collapsible enabled:



10. Select Collapsed On Init to make the section collapsed when first displayed. The user can expand and collapse the section as shown below:



11. Click Save.
12. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Next, learn about the Select Language component.

Slider

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Maestro comes with the Slider component (widget) that allows users to view and select a value or range of values from the range along a bar.

This component supports both left-to-right (LTR) and right-to-left (RTL) languages. For LTR languages, the smallest value appears on the far left end of the track and the largest value is on the far right. For RTL languages this orientation is reversed.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Slider** component within the **Input Fields** folder.



3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Slider component:

Configuration

Min Value Integer minValue
The minimum value of the slider.

Max Value
Integer maxValue

The maximum value of the slider.

Step Value
Integer step

The interval of the slider's value increment.

Show Ticks
Boolean showTicks

Shows tick marks at predetermined values, without their actual values, that the user can move the slider to. The predetermined values are calculated usingStep Value.

Show Ticks With Values
Boolean showTicksValues

Shows tick marks with their predetermined values that the user can move the slider to. The predetermined values are calculated using Step Value.

Value Prefix
Text valuePrefix

Adds this text to the value label as a prefix.

NOTE

This value can be translated to different languages including right-to-left languages. | 21.05 This feature was introduced in 21.05.

Value Suffix
Text valueSuffix

Adds this text to the value label as a suffix.

NOTE

This value can be translated to different languages including right-to-left languages. | 21.05 This feature was introduced in 21.05.

Add Space Between
Boolean addSpaceBetween

Adds a space between the prefix and the value label, and between the suffix and the value label.

Range Slider

Enable Range Slider
Boolean isRange

Shows the second thumb to allow the user to view and specify the range of values.

Min Range
Integer minRange

The minimum value of the range slider.

Max Range
Integer maxRange

The maximum value of the range slider.

Help Text

Popover Help Text
Help Text helpText

Shows a popover help text to provide some contextual help to the user. You can use the Maestro rich text editor to create the help text using various formatting and styles.

Next, learn about the "Multiline Text Area" on page 856 component.

Text Field Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Text Field component which allows you to create a text field in a form.

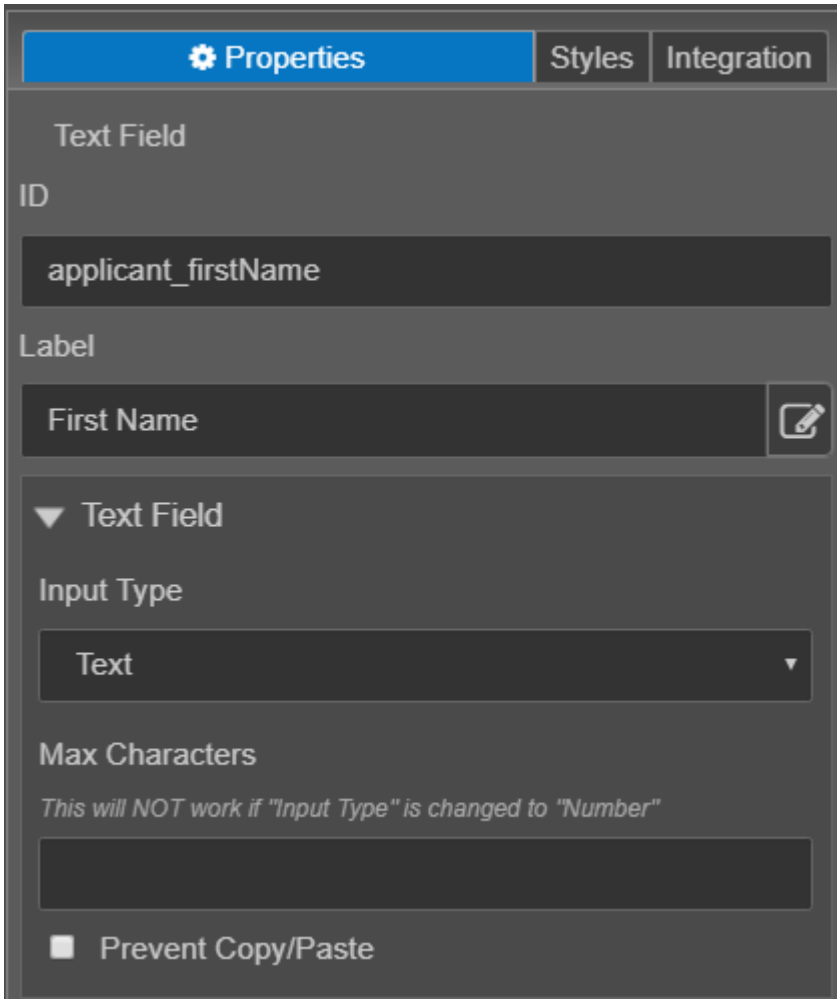
> NOTE
>
> For large amounts of text, you should use the Multiline Text Area component.

To configure the Text Field component on a form:

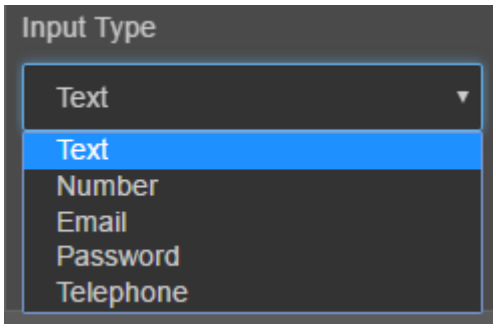
1. Open a Maestro form.
2. Click the Palette tab and expand Input Fields.
3. Select Text Field and drag and drop it on the form.



4. Click the Properties tab and find the Text Field section.



5. ID is the underlying Maestro name of the selected component.
6. Label displays the name to the user on the form. In the above example, the label is First Name.





- Input Type is specific to text related components. By default, when you drag a text field component on to a form, the input type defaults to Text. If you were to drag an email address component on to the form, the Email input type would be selected. Though you can change the input type, it is recommended that you change the component rather than the input type. For example, if you inserted a text field, but decide that you want an email address, you should delete the text field and add the email address component. The specific components have other pre-defined properties that are useful for Form Builders. For example, the email address component has a validation rule already created, whereas a text field, even if the input type is set to email, does not. The following input type are supported:
- Text - The Text input type should be used for text-based input. For more information on input fields, see https://www.w3schools.com/html/html_form_input_types.asp.
- Number is the input field used for numeric input (0-9). For example, if you want a user to enter their age, use a text field and set it as a number input type.
- Email is the input type used for text fields that should contain an e-mail address. If you select the email type for a component that is not created for email input, for example, a text field, the entered input will not be validated to ensure the user entered a valid email address. You can add a validation rule to the text field component, however, using the email address component (which already has the rule created) is much easier.
- Password is the input type used mask any characters entered into the component. The screenshot below displays the masked characters.
- Telephone - The Telephone input type is used for fields that should contain a telephone number. Similar to email addresses, you should use the specific phone number component, rather than changing the input type. The phone number component will contain the validation rule to ensure that the user is entering a valid phone number. You can add a validation rule to the text field component, however, using the phone number component (which already has the rule created) is much easier.

NOTE

Though these options are available for selection, it is recommended that when possible you use a component tailored to suit the needs of the data that will be collected by the field. For example, if want to collect an email address from a user, use the Email Address component from the Maestro Palette.

7. Max Characters - This property allows you to set the maximum number of characters that the user can enter into the field. Letters, numbers, special characters and spaces are each counted towards the maximum number of characters.
8. Prevent Copy/Paste - Select this option to prevent this text field from being copied to or pasted from the clipboard.
9. Click Save.
10. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Shared Components Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro encourages you to reuse forms' components wherever possible by providing an easy way to share components across multiple forms and projects.

Shared components are custom built blocks that can be reused throughout forms and can be made available to multiple forms by publishing them to a project library or an organization library.

Shared components can be created by Form Builders or Template Designers for re-use and consistency among forms. They can consist of a single component or a collection of components wrapped in a block. Shared components can be published to a library, so they are available to other forms and templates within the project or organization.

You can create a shared component from within a form using existing components or blocks, or by using the component editor from the Components folder in the Maestro Management Dashboard.

A shared component is a custom-built component within the business. A shared component can be added to your form just like any other component. The difference with a shared component is that it can be updated outside of the form. In this case, all forms using that shared component are also updated. This is a great way to re-use common components, but also keep them consistent across multiple forms.

Once the shared component is added to the form, you can change the Component ID. If the ID includes an entity, you can apply the entity to the child components within the shared component.

Shared components can be developed within the Components folder in the Maestro Dashboard, or within a form or template.

Shared components can be published to any category within the Palette panel. The default location is the Custom Components folder, but you can change it anytime.

Organizations usually build shared components for the following reasons:

- **Consistency** - to ensure that a component looks and functions the same wherever it is used.

- **Encapsulation of complexity** - to allow complex components to just be dropped in and used, without having to understand the internal implementation. This is particularly useful when a component makes dynamic data calls to back-end services.
- **Productivity** - allow forms to be built faster.
- **Enable change** - if a component needs to be changed, all the forms that use it will automatically inherit the changes.

Next, learn how to create a shared component.

Create a Shared Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

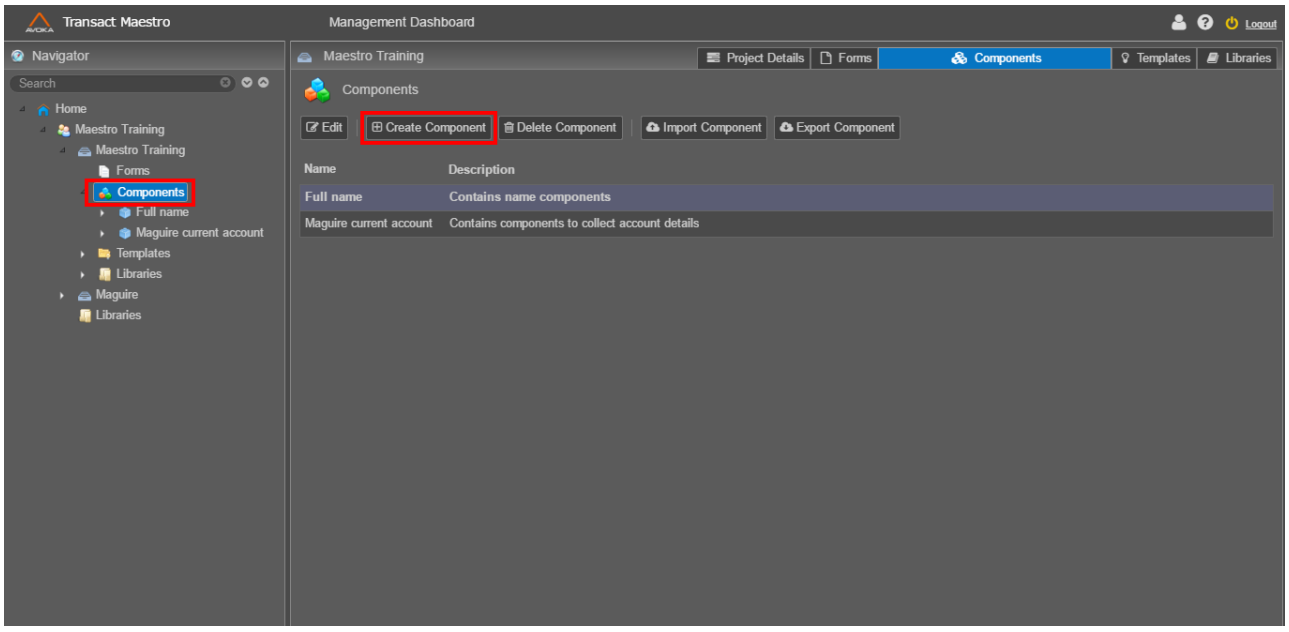
Maestro provides several ways to create a new shared component.

- Create from scratch in the Management Dashboard.
- Create from existing form items in the Maestro editor.

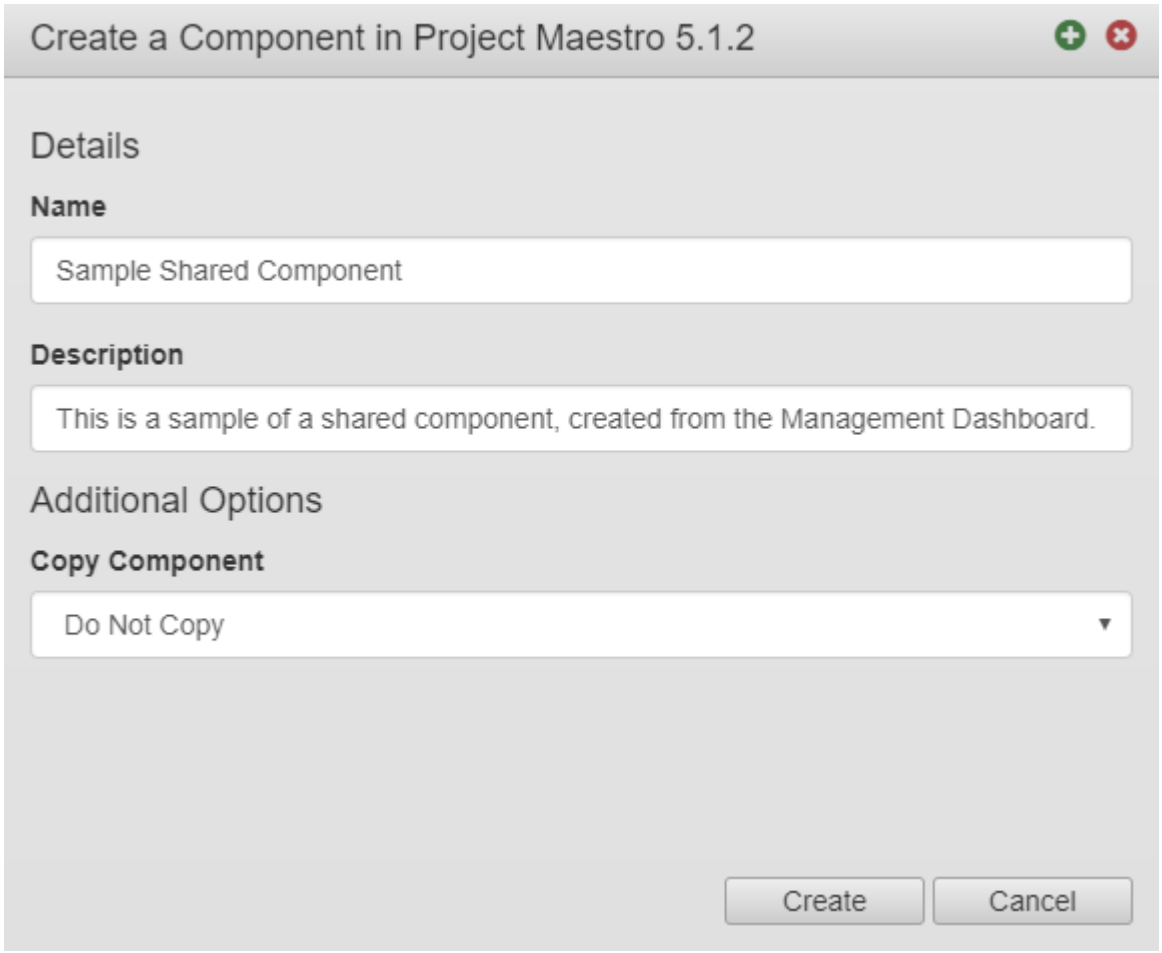
Create a Shared Component from the Management Dashboard

To create a shared component from Maestro's Management Dashboard, follow these steps:

1. Go to the Navigator pane.
2. Select Home > Organization > Project.
3. Select the Components folder in the project and click Create Component.



4. Edit the new component details.



5. Name - the component's name is used to identify the component in the Components folder.
6. Description – a description of the component, which is displayed when the Components folder is selected.
7. Copy Component – select from the following options:
    - Do Not Copy - This option allows you to create a brand new (blank) component. This option is recommended for shared components that you are building from scratch.
    - Copy and override an existing component - This option allows you to copy an existing component. The new component will replace the existing component in the library, which will update all forms using the existing component. The override will only replace the library resource, not the asset in the Components folder. Before selecting this option, it is important that you consider the usage of the component that you will be overriding.
    - Copy and create a new component - This option creates a new component based on an existing component. Selecting this option will create a copy without overriding the existing component.
8. Click Create.

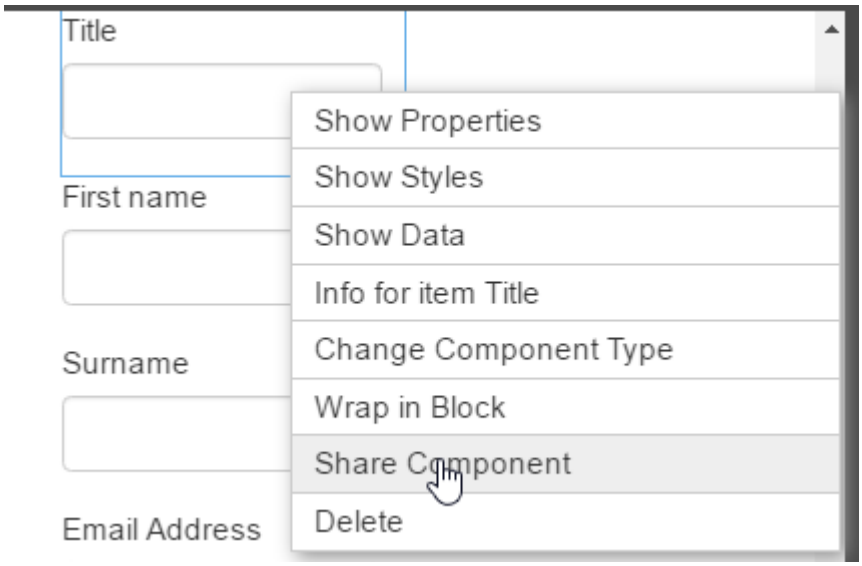
9. Edit the new component in the Maestro editor.
10. Publish the component.

Next, learn how to create a new shared component from existing items.

Create a Shared Component from Existing Form Items

To create a shared component from one or more existing form items in the Maestro editor:

1. Open a Maestro form.
2. Create a component.
3. Right click on the component in the Maestro editor and select Share Component.



4. If you are creating a shared component that contains more than one component, you need to wrap all components in a block first:
   - Select all components.
   - Right-click a selected component.
   - Click Wrap in Block from the dropdown list.
5. Edit the share component details.



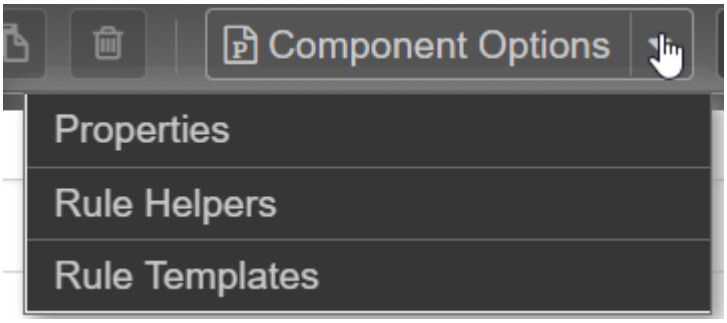
6. Name is the default name of the new component is the label of the selected item.
7. Description is a short meaningful description of the shared component. This is optional parameter, but we recommend to provide a description.
8. Palette Folder defines where the shared component can be found in the Palette pane. The default is Custom Components.
9. Select Publish for immediate use to make the shared component immediately available to in other forms.
10. Select Replace item with shared component to use the component as the basis for the shared component will now be replaced by the shared component (i.e. it will now automatically reflect any changes made to the shared component).
11. Select a library from the Publish Library dropdown list to publish it to the project library, if you want to make this shared component only available to forms in this project, or to the Organization library, if you want any form in the organization to access this shared component.
12. Click Save. You can continue editing the shared component in the Component editor or view it in the Management Dashboard.
13. Publish the component.

Next, learn how to edit a shared component.

Component Options

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Journey Maestro allows you to identify and extend functionality of custom and shared components using the Component Options, as shown below.



You can add the following options to a component from the Maestro editor:

- Properties
- Rule Helpers
- Rule Templates

NOTE

You cannot change the Component Options of core components.

Properties Options

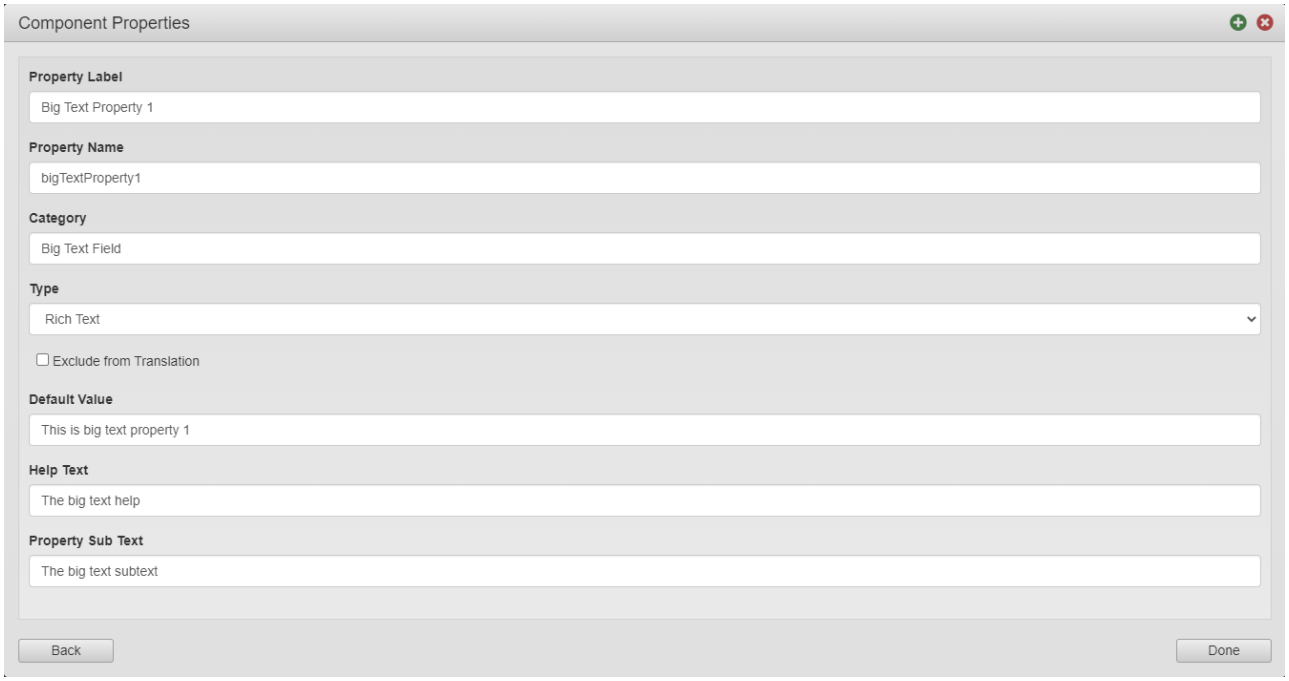
Maestro provides fine control of various component options and Properties options. You can use these options to configure different component attributes, including branding and help text.

To configure Properties options for a component:

1. Open a project in your organization.
2. Select the component in the Component folder from the Maestro Dashboard.
3. Click Edit Version.
4. Click Component Options and select Properties.



5. Click New Property and edit the details of a new property.



6. Property Label is visible on the component. This is the mandatory field.
7. Property Name is the unique identifier of the property, which is automatically generated based on the label name.
8. Category: Options, Input Fields, Background services, etc.

9. Select a property type from the Type dropdown list that can be one of the followings: a Boolean, Options, Field Ref Map, Help Text, etc. The functionality of the property will differ depending on its type.
10. Default Value identifies whether the property should be configured with a default value, which is true.
11. Help Text is the text that is displayed when a cursor hovers over the component, for example 'Controls whether to attempt a background check on a driver's license holder'.
12. Click Done.
13. Check the recently created property in the Component Properties window. You can also create another property if required.



14. Click Save.

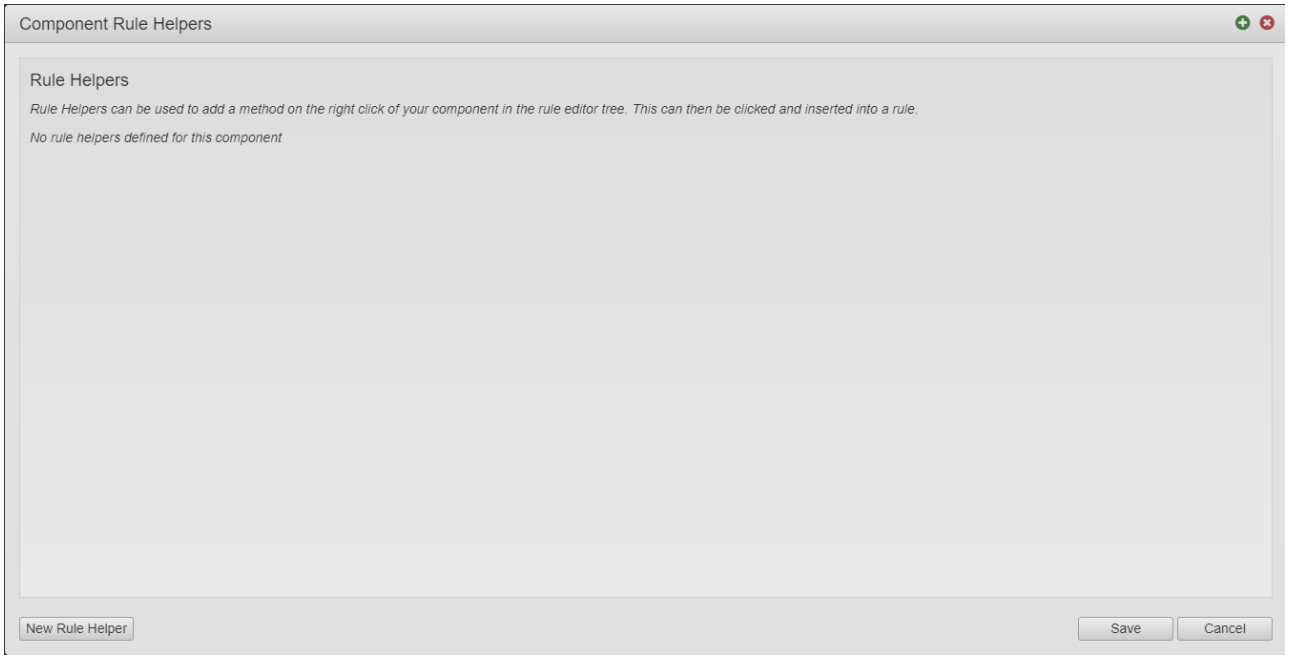
Rule Helper Options

Maestro provides fine control of various component options, including Rule Helper options. You can use these options to add methods to a component right-click event in the rule editor, so a Rule Helper will appear as a selectable option when users right-click the component in the rule editor tree.

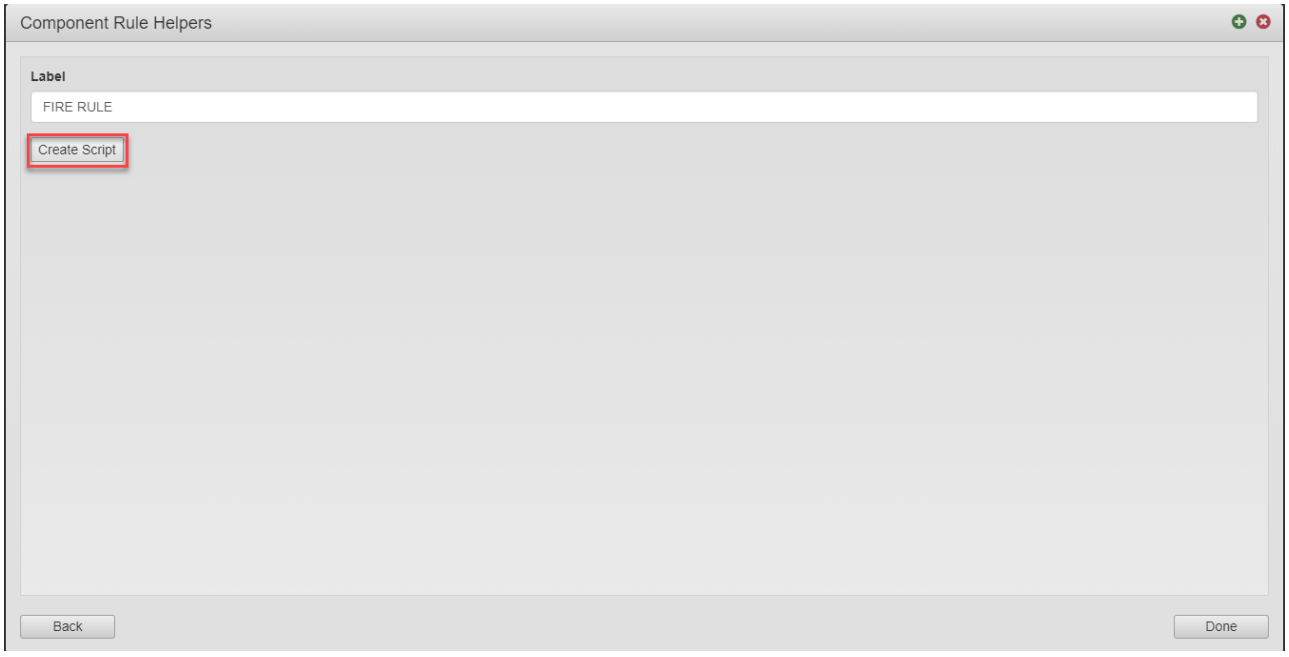
To configure Rule Helper options for a component:

1. Open a project in your organization.
2. Select the component in the Component folder from the Maestro Dashboard.
3. Click Edit Version.

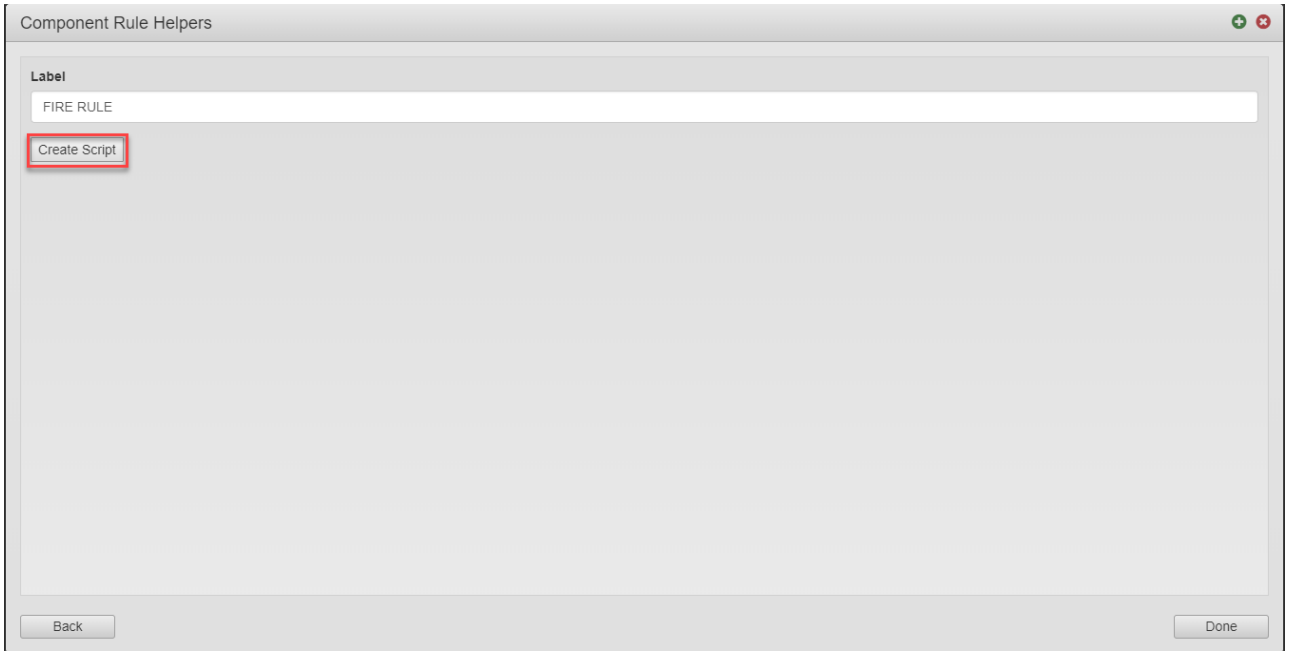
4. Click Component Options and select Rule Helpers.



5. Click New Rule Helper.



6. Type a name for a Label to identify the rule helper when a user right-clicks the component in the editor tree and click Create Script.



7. Enter a JavaScript for the new rule helper.



A JavaScript can follow this pattern: `Form.fireRule ("name of the Component Template", Form.items.component_ID, data)`, as shown in the example below:

```
Form.fireRule("testRule", Form.items.componentRuleTestID, data);
```

In this example, testRule is the name of the Component Template and componentRuleTest is the component ID of the component that the Rule helper is associated with. When a user right-clicks the component with the component ID componentRuleTestID, the configured Rule Helper is available as a selectable option.

> NOTE
>
> You can use insert-item keyword to reference the item selected from the tree.

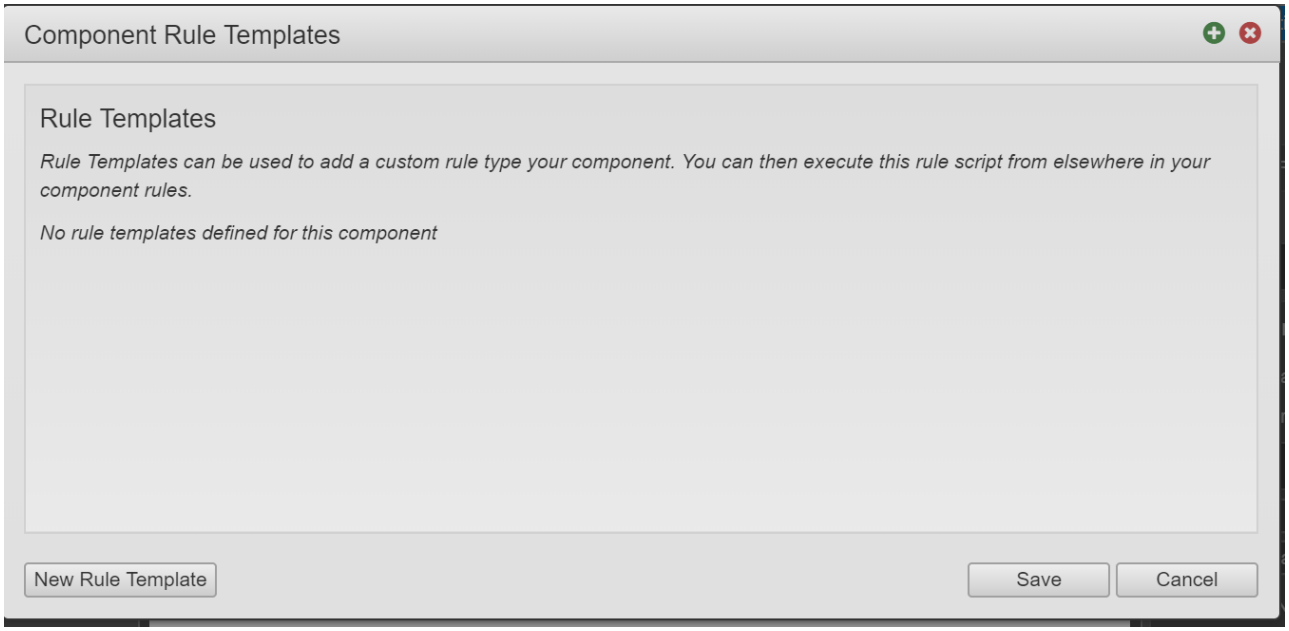
8. Click Done and then click Save.

Rule Template Options

Journey Maestro provides you with the fine control of various component options, including Rule Template options. You can use these options to add custom rule types to a component. Once a rule is configured, you can run its JavaScript from another location of your form.

To configure Rule Templates options for a component:

1. Open a project in your organization.
2. Select the component in the Component folder from the Maestro Dashboard.
3. Click Edit Version.
4. Click Component Options and select Rule Templates.



5. Click New Rule Template.



6. Enter the Name for the rule template, which automatically prefills the Type of this rule file and the code required to execute it, for example: `Form.fireRule("test1", item, data)`

> NOTE
>
> It is recommended to save this code for your further reference as you will need it for an executable command later.

7. Click Create Default Script to edit a rule script.



8. Click Done.



Next, check our best practices.

Override a Shared Component

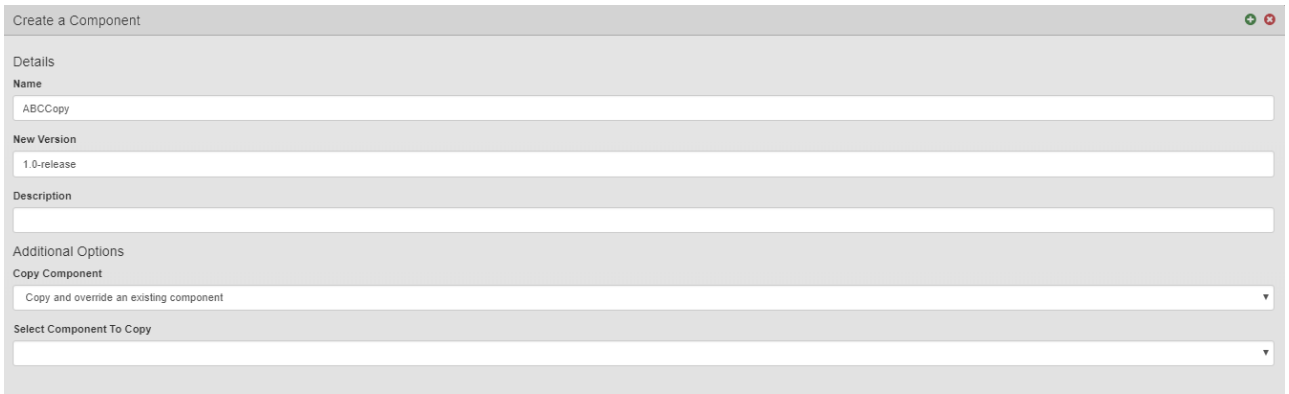
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to override shared components. Though there are a variety of Maestro components available from the palette, there may be times where the available components do not fit the needs of your form and you may want to override a native component with a custom component that you can share across multiple locations in a form. Other reasons for overriding shared components are:

- Update a component to fill a specific one-time only purpose
- Create a component that provides functionality that is specific to your company

To copy and override a shared component:

1. Create a shared component.
2. Publish the shared component.
3. Add the published shared component to a form.
4. Create another new shared component from the Maestro Dashboard. To use the copy and override option, you must create the shared component from the Maestro Dashboard. It is important to select copy and override an existing component from the Copy Component dropdown list. Once the shared component is created, you can edit it in the Maestro editor. The components from the original shared component will display in the editor.



5. Select the component that you want to copy and override, for example the first shared component ABC, from the Select Component To Copy dropdown list. Selecting ABC means that all changes made in the ABCCopy shared component replace the original ABC shared component.
6. Edit the shared component.

7. Click Publish. When you publish the shared component make sure you publish to the same library as the original. If you publish to a different library, it will not fully override the original and the shared component displayed in the form will be determined based on the precedence of the library used in the form.
8. Go back to the form and refresh the form. This updates the content of the original shared component with the contents that you added to the ABCCopy shared component.

Next, learn how to edit a shared component.

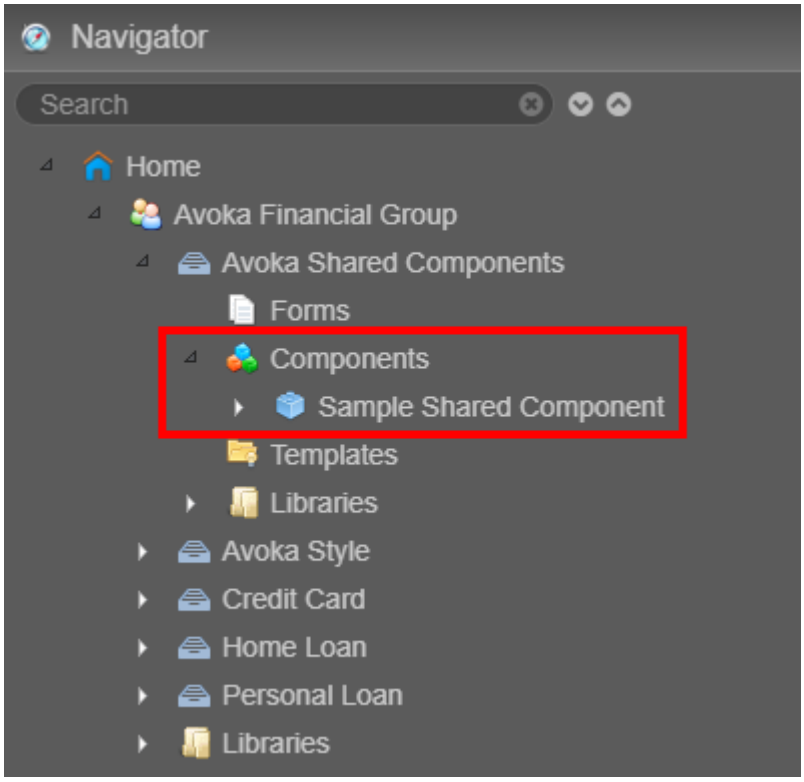
Edit a Shared Component

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

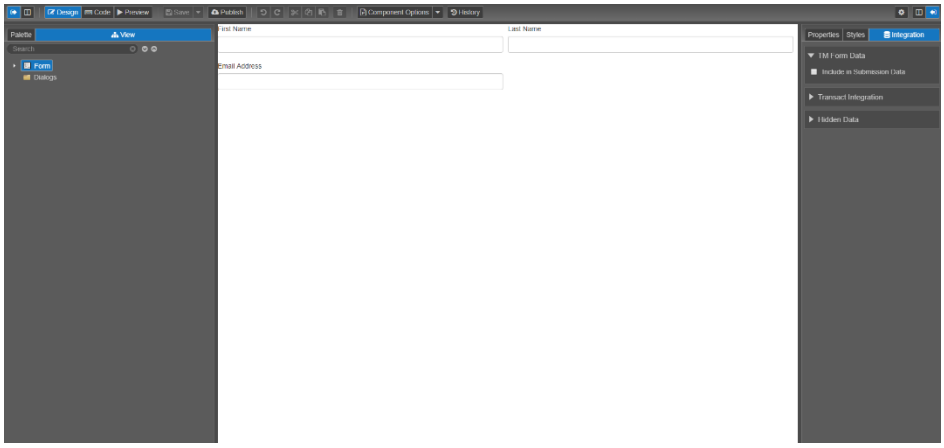
The Maestro editor allows you to edit existing shared components, for example a shared style, to implement customer requirements. Sometimes, your changes may override a shared component.

To edit a shared component:

1. Select the Navigator pane.
2. Select Home > Organization > Project.
3. Click theComponents folder.



4. Double-click a shared component to open it.



5. Edit the component as required and click Save when you are done.
6. Publish the component to make changes to the shared component available to this and other forms.

Next, learn how to override a shared component.

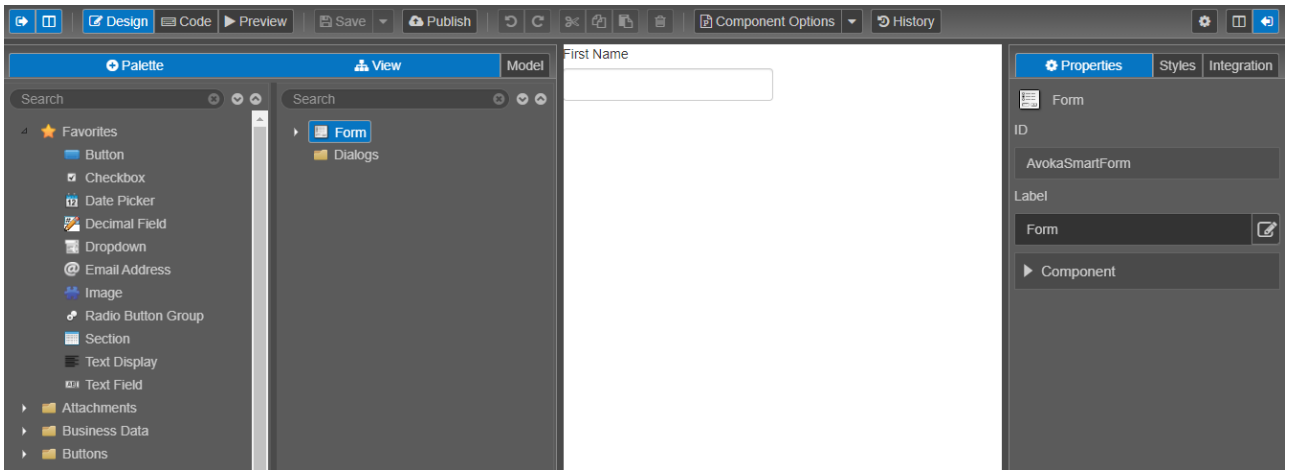
Publish a Shared Component

MaestroThe UI design product. | Form Builder | 20.05 This feature was updated in 20.05.

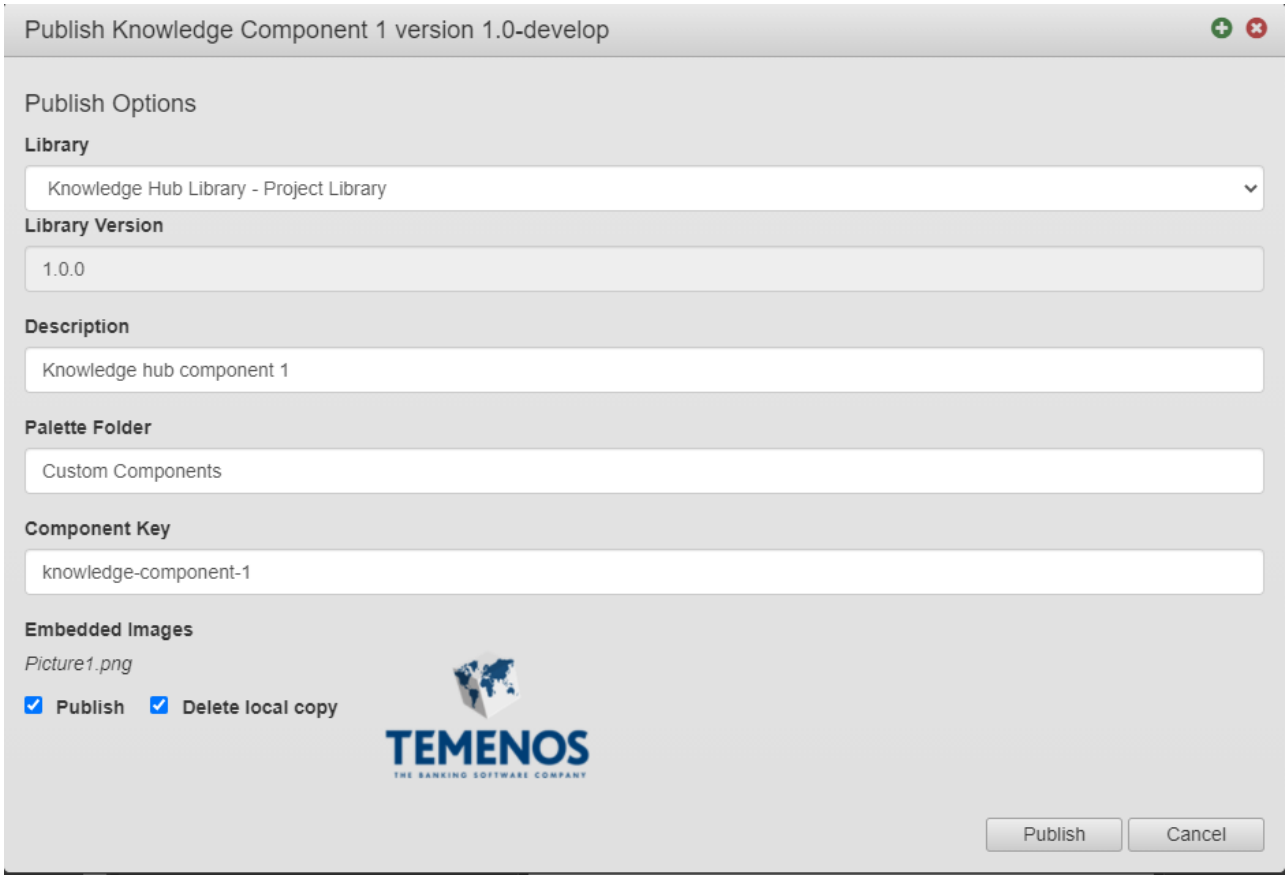
After you have created or updated a shared component, the Maestro editor allows you to publish it to a library so it can be used in forms or other components. Maestro provides the option to publish for immediate use, which means that the shared component will be immediately available to this form as well as other forms.

To publish a shared component:

1. Create a new shared component or find the existing one.



2. Click Publish.



3. Select a library, where the published shared component will be stored, from the Library dropdown list. It determines which forms have access to the shared component. It is recommended that you understand the organization, project and library structure, including where shared components should be published. When you select the library, the mapped library version will also be selected in the Library Version dropdown list below.
4. Select a specific version of the library from the Library Version dropdown list. | 18.05 This feature was introduced in 18.05.
5. Provide a description of the shared component in the Description field.
6. Update a name of the folder, where the shared component will be displayed in the Palette panel, in the Palette Folder field.
7. Review and update an uniquely identifier the shared component in the Component Key field.

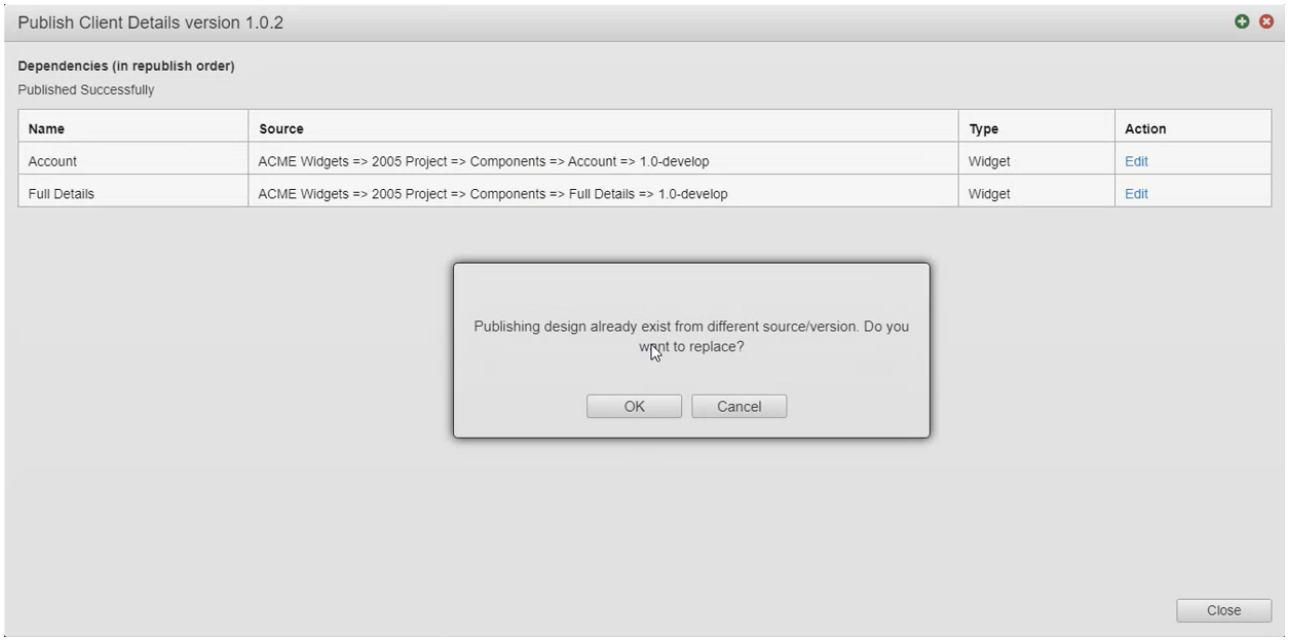
> NOTE
>
> The Component Key is automatically generated based on the name of the shared component, so we don't recommended changing it as it can lead to issues when the component is deployed in a Maestro form.

8. Select the Publish checkbox to publish listed shared styles or images that are only in the component design.
9. Select the Delete local copy checkbox to remove the local copy of the listed shared styles or images.
10. Click Publish to publish the component, if Maestro has not detected any impact changes, or click Next to review the impact analysis of component changes before publishing. | 20.05 This feature was introduced in 20.05.



For example, if you have a few components- a Client Details and Account components - where the Client Details component is the base component that is used in the Account component. They are already published as 1.0.1. After you create a new version of the client details component as 1.0.2 and try to publish it, Maestro will detect a clash and present you with a list of the dependencies that need to be republished.

11. Review the impact of the change and whether the impacted components should be re-published and click Publish to publish the shared component.
12. Click OK to confirm your decision.



13. Click Edit against each component to perform some action, for example, you can re-publish it.

Next, learn how to edit a shared component.

Custom Native Components Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A custom native component is a custom component that can be shared with other Maestro Form Builders and Platform Developers. A native component consists of a collection of files that are packaged together in a ZIP file using a specific folder structure, as shown below.

File Name	Description
library-config.xml	A file with this exact name must be in the ZIP file's root folder. This file identifies the ZIP file as a Maestro Library and provides meta-data such as dates and author.
widgets	A folder with this exact name must be in the ZIP file's root folder. This folder contains a sub-folder for each native component.
component sub-folder	Each component has a sub-folder under the top-level "widgets" folder. A component sub-folder contains the files that define the native component.
widget.json	A file with this exact name must be in the component sub-folder. This file includes JSON properties that defines various component attributes.
widget.png	A 24x24 pixel, 32-bit color image with a transparency layer. This is used as the component's icon in the Maestro Palette pane.
widget.html	HTML code defining the component's structure.
widget.css	CSS code defining the component's look and feel.
widget.js	JavaScript code defining the component's behavior.
other.js	Other JavaScript files that the component depends on, such as jQuery plugins.

Multiple native components can be packaged into a single ZIP file. The ZIP file is shared in Maestro by importing it into a project or organization library. Once imported, the components in the ZIP file are available on the Maestro Palette pane and can be used like any other component.

Next, learn how to develop a custom native component.

Custom Native Component Development

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A component in Journey Maestro can be as simple or complex as the Form Builder wants or needs it to be. A Maestro component can be as basic as a one-line snippet of HTML[1] code to display a piece of text, or it can be as complex as displaying high fidelity graphics, or communicating with multiple servers.

Before developing any native component for your form, check if you can implement the required functionality using the available components in the Maestro Palette. It is important to consider and understand the use case of a new native component. For example, if it requires a server connection, it will be useless in a form that will be deployed for offline use.

You can develop custom native components using the web-standard resources such as JSON, HTML, CSS, and JavaScript by following the recommended process outlined here.

1. Created a new folder to hold all the files necessary for your native component. Name the folder the same as your native component.
2. Create a JSON file.
3. Create a HTML file.
4. Create a JavaScript file.
5. Use a jQuery.
6. Package and distribute the completed custom native component.

Create a Custom Native Component JSON File

Every custom native component that you develop in Maestro must have a JSON[2] file, so your first step is to create a valid JSON file, for example, `widget.json` and populate it with relevant information about the new native component

[1]Bounce is when a user views a form but never starts it and simply closes the window immediately or navigates away.

[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.

NOTE

JSON is a subset of the JavaScript object literal syntax, so you must surround your JSON property names with double quotation marks “”

A JSON file of this component is shown below:

```
{
	"label": "Component A",
	"span": {"xs": 12, "sm": 12, "md": 4, "lg": 4},
	"propDefs": {
		"property1": {
			"label": "Property 1",
			"type": "text",
			"defaultValue": "Default",
			"helpText": "Specify a value for the custom property of this component"
		},
		"property2": {
			"label": "Property 2"
		}
	},
	"html": "component.html",
	"js": "component.js",
	"css": "component.css",
	"icon": "component.png"
}
```

A typical custom native component JSON file can contain the following elements:

Element	Required	Example	Description
label	Yes	"label": "Component A"	The text displayed in the Maestro palette for this native component. Note that this is the only required element in a native component's JSON file; for example, `{ "label": "Hello World" }`.
span	No	"span": {"xs": 12, "sm": 12, "md": 4, "lg": 4}	The span value of the native component, which is the number of columns that the native component will take up in the form. The default value is 3. Maximum value is 12.

propDefs	No	"propDefs": { "property1": {...}, ...}	The list of all properties defined for this component. Each property can have its own set of property-value pairs. For more information, see the property table below.
html	No	"html": "component.html"	The name of the native component's HTML file. The default is component.html.
extraHTML	No		An additional HTML file or an array of HTML files that are generated in the completed form's head section. It's useful for adding HTML for popup dialogs and other similar functions.
js	No	"js": "component.js"	The name of the native component's JavaScript file. No JavaScript file is loaded by default.
css	No	"css": "component.css"	The name of the native component's CSS file. No CSS file is loaded by default.
icon	No	"icon": "dropdown.png"	The name of the native component's icon file, for example, a PNG, GIF or JPG image file. The default is a generic icon.
rows	No	"icon": "component.png"	Native components that need to contain other components should have a blank rows collection, which is an array within an array. By default, the component doesn't have rows collection, asit doesn't contain other components.
noData	No		Set to true for display native components that should not have entries in the form's output data.
validChildren	No		An array of component types that can be added to this component type. This prevents adding or moving components of other types into this component, and also creates options in the native component's right-click menu to add the specified component types as children. Set `validChildren` to an empty array to prevent any new content being added to the native component directly. If the native component contains a container, content can be added to the native component using this property.

selections	No	An object containing selection name-keyed selection arrays, typically one, that if not already there, are added to the form's selection collection. This allows the component to provide a default selection set to use in a drop-down, radio group, etc. The name of each selection array must not contain any non-alphanumerical characters other than underscores. Underscores are translated as spaces in the property editor window.

A JSON file can have a number of properties under `propDefs`, which are a keyed map of custom properties that appears in the Maestro property editor which can be referenced in your native component's code. The key of each property item within the properties object is the internal name of the custom property. For example, the name you use when referring to the item in your native component's code. This name must be camelCase, for example, StyleGuide.

A property element can contain the following elements:

Property Element	Required	Example	Description
label	Yes	"label": "Options",	The display name for the custom property that will be displayed in the Maestro property editor. If omitted or left blank, the default is, "text". In the above example, the label is, "My Property".
type	No	"type": "text"	This determines the input type in the Maestro property editor.
value	No	"value": "One"	The value of the property.
defaultValue	No	"defaultValue": "Default"	This is the default value that this property has when this component is first added to a form. The default is null.
helpText	No	"helpText": "My Help"	Any additional information that may be relevant to the Form Designer within the Maestro property editor. No help text link is displayed by default.
linked	No	"linked": "true"	If set to true, updates to the item's label is also set this property. This only occurs if this property is not currently overridden in the item, or if its value already matches that of the label.

The custom native component properties have different types.

If this component or widget extends another component and you are setting a property against a property defined on its parent, you can override their values. There are cases where there is a propDef to add properties that the user can set and properties that are setting the values on an inherited type.

```
{
	"category": "Input Fields",
	"label": "Dropdown",
	"html": "dropdown.html",
	"icon": "dropdown.png",
	"propDefs": {
		"options": {
			"label": "Options",
			"type": "list",
			"value": [
				{
					"label": "One",
					"value": "One"
				},
				{
					"label": "Two",
					"value": "Two"
				},
				{
					"label": "Three",
					"value": "Three"
				}
			],
			"helpText": "The options list for the dropdown"
		},
		"helpText": {
			"label": "Popover Help Text",
			"type": "helpText",
			"category": "Help Text",
			"value": ""
		},
		"icon": {
			"label": "Icon",
			"type": "icon",
			"value": "fa fa-caret-down",
			"helpText": "Select the dropdown icon"
		},
		"addBlankValue": {
			"label": "Add Blank Value",
			"type": "boolean",
			"value": true,
```

```
                        "helpText": "Check to add a blank value to the dropdown"
                },
                "placeholder": {
                        "label": "Placeholder Text",
                        "type": "text",
                        "category": "Accessibility",
                        "value": ""
                }
        },
        "styleTargets": ["wdg-select-icon"],
        "span": {"xs": 12, "sm": 12, "md": 6, "lg": 6},
        "ruleHelpers": [
                {
                        "label": "String value...",
                        "name": "stringValue",
                        "initDialog": true
                }
        ]
}
```

```
{
	"type": "dropdown",
	"label": "Title",
	"category": "Business Data",
	"properties": {
		"options": [
			{
				"label": "Mr",
				"value": "Mr"
			},
			{
				"label": "Mrs",
				"value": "Mrs"
			},
			{
				"label": "Miss",
				"value": "Miss"
			},
			{
				"label": "Ms",
				"value": "Ms"
			},
			{
				"label": "Other",
				"value": "Other"
			}
		]
	}
}
```

This is the common case, the first one is the `/dropdown/widget.json` and the second one is the `/dropdown-title/widget.json`. The directory name is the type and it contains a `widget.json`. In the dropdown you have a propDefs for options being declared, but in the dropdown-title you have properties being set for this widget

Create a Custom Native Component HTML File

After you have created the JSON file, `widget.json`, the next step is to build the native component's HTML file. Every native component created for use in Maestro needs a HTML file, for example, `component-widget.html`. This HTML file should define the content that the native component will place in the form's HTML document. The HTML file should have a single top-level element,

usually a `div`. Within this top-level element, using standard (or custom) tag names and attributes, any valid HTML script can be placed.

A Maestro custom native component can be as simple as a component designed to display a simple piece of text, for example, `Hello World`. You can use the following process to create a simple Hello World custom native component:

1. Create the HTML file `component-widget.html`.

   ```
   <div>Hello, World!</div>
   ```

2. Place both the `widget.json` and `component-widget.html` files into a folder named `Hello World`.
3. Zip the `Hello World` folder as a `Hello_World.zip`.
4. Use Maestro to deploy the library of components `Hello_World.zip` to your form.

This Hello World native component doesn't do much so it is not very useful. Normally, components are used to capture user input data. To do that, Maestro uses the AngularJS[1] JavaScript library to extend basic HTML syntax with two-way data binding. You can use this functionality to make the Hello World native component more useful by adding the code below to the `component-widget.html` file.

```
<div>
   Enter your name: <input type="text" ng-model="data[item.id]"/>
   <p>Hello, {{data[item.id]}}!</p>
</div>
```

This updated version has a text input field whose value is bound to `data[item.id]` using the `ng-model` attribute. Whatever value is inputted into data[item.id] will form part of the form's submission data. In this case, this will be the text that is entered into the input field by the user, for example, a user's name.

Below the input field, the value of `data[item.id]` (the text entered into the text field) is displayed in a greeting paragraph, taking advantage of the two-way binding.

---

[1]A workflow is progression of steps (tasks, events, interactions) that comprise a work process, involve two or more persons, and create or add value to the organization's activities. In a sequential workflow, each step is dependent on occurrence of the previous step; in a parallel workflow, two or more steps can occur concurrently.

Create a Custom Native Component JavaScript File

By now, you have created the component's JSON and the HTML files that capture information, provides feedback and stores a value in the form's submission data.

So far, we have illustrated that no JavaScript is required to create a native component that can be used in Maestro. However, JavaScript can be used when creating native components for use in Maestro forms. A JavaScript file should be used when the native component needs to define custom behaviors or processes. The JavaScript file needs to conform to a special structure. This structure means that all JavaScript files used to create native components must begin with the basic structure as shown below.

```
define(["app"], function(app) { /* Your code here */ }
```

The recommended practice for providing a native component with some type of functionality is in the form of AngularJS objects. As described in the AngularJS developer guide and API reference guide (https://docs.angularjs.org/guide), if objects can be filters, directives, controllers or services, they are available to be used in JavaScript files.

The following list provides a brief explanation of filters, directives, controllers and services:

- Filters - Provide value transformation functions (For example text formatting)
- Directives - Enable inspection or manipulation of the DOM (Document Object Model). For example, adding an event handler.
- Controllers – Provide values and functions for binding. For example, managing a complex component data type.
- Services – Provide everything else. For example, integration with a remote server.

You can use the above-mentioned syntax in your native component to convert a text string into an UPPERCASE text string:

```
define(["app"], function(app) {
	app.ng.filter("upperCase", function() {
		return function(text) {
			return text.toUpperCase();
		};
	});
}
```

The code in the above image creates a filter called `upperCase`. The filter function takes a piece of text and returns the UPPERCASE version. This same function could be used in the Hello World native component's HTML file, `component-widget.html`, by pushing the data through the filter function. The image below demonstrates how this filter function can be applied to the HTML file of the Hello World native component.

```
<div component="item">
   Enter your name: <input type="text" ng-model="data[item.id]"/>
   <p>Hello, {{data[item.id] | upperCase}}!</p>
</div>
```

With the filter applied to the HTML file, the greeting provided to the user when the Hello World native component is loaded, will be shown in uppercase - HELLO, NAME!.

Create a Custom Native Component jQuery File

All Maestro forms include a full jQuery[1] library by default, and all element references passed into angular directive link methods are jQuery objects with the full jQuery API available. This makes it possible to use familiar jQuery methods and functions. However, it is recommended that you investigate whether you can use Angular data binding or built-in directives to meet your needs before doing any work on the DOM[2].

You can build custom Angular directives for use in your custom native component.

```
define(["app"], function(app) {
	app.ng.directive("mySpecialDirective", [function() {
		return function($scope, $el, $attr) {
			$el.text("Hello from my special directive");
		};
	}]);
}
```

---

[1]A reference number is a unique identifier assigned to each transaction.

[2]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.

If there is some part of your native component that needs to access properties of the Form View item, the JavaScript file should use Angular dependency injection to reference the form service object and call the `Form.getItem()` method.

```
define(["app"], function(app) {
	app.ng.directive("mySpecialDirective", ["Form", function() {
		return function($scope, $el, $attr) {
			Form.getItem().then(function(item) {
				$el.text(item.properties.messageSpecifiedByFormDesigner);
			});
		};
	}]);
}
```

The above example is quite contrived and could be handled in a much simpler way by binding the custom property value in the markup and using no JavaScript at all.

A native component can often specify a custom validation method, which is called before any validation rules are specified by a Form Builder. An example of this is the Date Picker component from the Maestro Palette. You can add a custom validation to the component's date range properties as shown below.

1. Obtain a reference to the View item, as explained above.
2. Add a `$validation` method to the View item.

   ```
   item.$validation = function(data, value) {
   	value = + value;    // Ensure value is a number
   	if (value < 5) return "Too low";
   	else if (value > 10) return "Too high";
   	else return true;
   }
   ```

   This method takes two parameters - the item's parent data context and its own data value - and returns a validation result - true for valid, false for an error message.

3. Add the validation directive attribute `data-av-ok` to the top-level `div` of your native component's HTML file.

You can apply a similar approach to create a custom mandatory value checker, for example, `$hasValue`. This method is used to override the normal way of checking whether a value has been

entered, if the field is marked as mandatory. Instead of checking if a value is provided, a mandatory field of this type will only pass if the field's length is greater than 5.

```
item.$hasValue = function(data, value) {
       return value.length > 5;
}
```

NOTE

A file attachments component uses this method.

Package a Custom Native Component

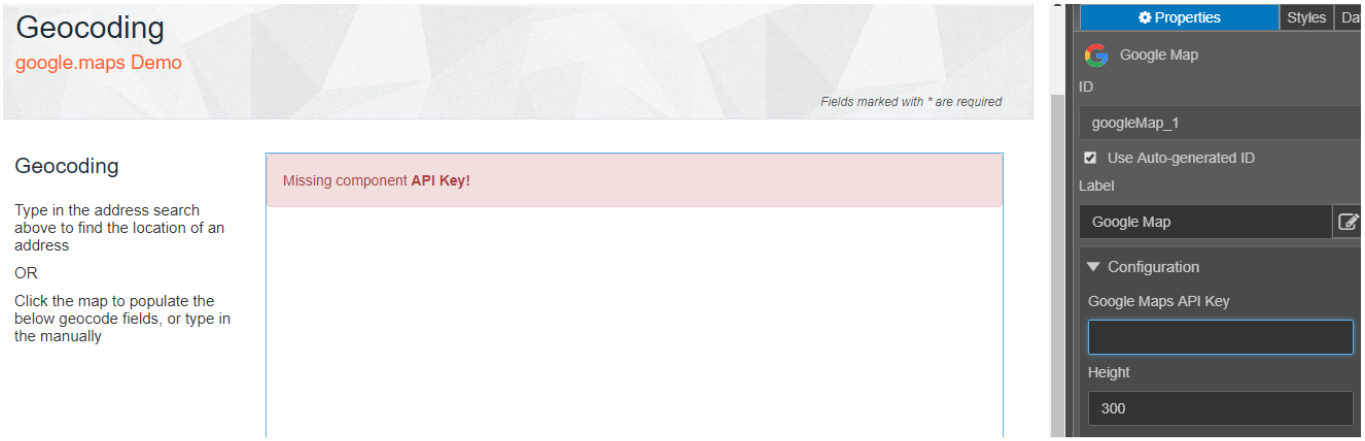
After you have created all files required by a custom native component, you can package and deploy it following this process:

1. Add a library's XML manifest including a library's name and description to the component's directory.
2. Zip the content of the directory.
3. Use the gulp[1] tasks to automatically package and deploy libraries into a Journey Maestro server.
4. Import the library into Journey Maestro.

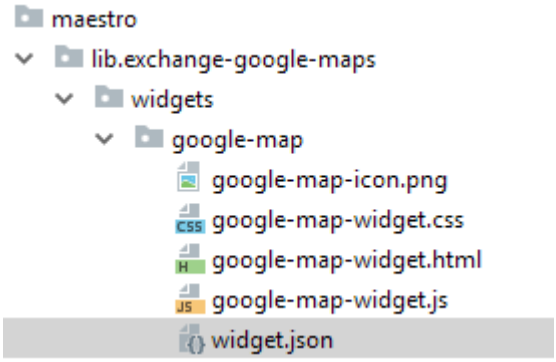
Enforcing Required Configuration Properties

Sometimes, as a component designer, we want to define a component property that is required to be set. Since we don't have a built-in Maestro function to mark a property as "mandatory" field, we can still show a warning message like below directly in Maestro design view to remind the form designer to set this value. "Missing component API Key!" message will disappear as soon as "Google Maps API Key" property has been set.

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



This technique can only be used when you develop a native component. Suppose you have native component called "Google Map" which structure looks like this.



In the widget,json, it defines a property "apiKey", then what you can do is to add a in google-map-widget.html like below

```
<div class="av-form-field google-map" data-ng-style="{height: item.properties.height+ 'px'}"
    <div ng-if="!item.properties.apiKey" class="alert alert-danger">
        Missing component <strong>API Key!</strong>
    </div>
    <div id="google-map-widget-container">
    </div>
</div>
```

The logic is actually pretty simple. If item.properties.apiKey value is blank, then show this "Missing component API Key" message with "alert alert-danger" class. The will be hidden when form designer set an api key to the property.

Next, learn about the Native Component Development Toolkit.

Create a Native Component Library Automatically

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

You can use the gulp watch task to automatically create a native component library while you are developing your components in a Visual Studio Code editor.

To create a library in Visual Studio Code:

1. Install the Native Component Development Toolkit.
2. Download Visual Studio Code and install it.
3. Start the Visual Studio Code IDE.
4. Select File/Open Folder, then open a folder where you installed the development toolkit
5. Select View/Integrated Terminal to open its Integrated Terminal.
6. Type `gulp watch` into the Integrated Terminal window
7. Edit your component files in the `src` folder.
8. Click File/Save save them, so the `gulp watch` builds your Maestro library in the `dist` directory.
9. Import this library into your Maestro instance.

NOTE

If you don't want to continuously have the `gulp watch` task running, you can use `gulp zip` command to generate your Maestro library manually.

For more information, see Creating native components for Maestro (Eclipse/ IntelliJ Developers).

Custom Native Component Property Types

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The custom properties defined in a custom native component's JSON file can have different types (property types). A property's type determines the following:

- The editor that is displayed to the Form Designer for setting the property
- The data type that the property's `value` will have when accessed from within the native component's code. A property's `defaultValue` element must also match this data type.

Properties of certain types can have additional settings within the property object. For example, properties of type `option` must specify a setting for options. This is done to indicate the value choices available to the Form Designer.

The table below identifies several of the property types associated with a native component's JSON file.

Type	Editor	Value Datatype	Additional Settings
text	Text input	string	
integer	Number picker with text entry and up and down buttons.	number	min, max – (Optional) restrict the range of the value entered.
boolean	Checkbox	boolean	
option	Dropdown List. Maestro shows the list and the form designer selects a value from it. The selected value is stored in the item properties.	string (the chosen option)	options – (Required) an array of values to choose between. These can be strings. If the values are strings, then they can be used for both the text in the editor dropdown and the value assigned to the property. If separate labels and values are needed, the array can contain objects with label and value properties.
list	Selection property editor dialog. Maestro allows the list to be edited by adding and removing entries. The resulting array is stored in the	array of objects	value – the initial entries of the list as an object array. Each object requires value and label properties. The initial list may be an empty array.

	item properties.		
color	Text input	string	
url	Text input	string (must be valid URL)	
image	Image picker. Uploads and/or selects a resource image.	object with fileName, width, height properties	value - an object with fileName, width and height properties. Image can be referenced in HTML by `{{Resource.imagePath(item.-properties.image.fileName)}}`.

Native Component Development Toolkit

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Temenos Journey Manager platform provides you with the Maestro Native Component Development Toolkit to simplify the creation of custom native components through the use of templates, and automated tasks. This toolkit is based on the gulp automation tool, which executes custom tasks defined in a `gulpfile.js` JavaScript file.

Installation

To install the toolkit:

1. Install NodeJs.
2. Download avoka-native-component-tools.zip.
3. Extract the zip file into an empty directory on your computer. You can do this by executing the following command:

   ```
   unzip avoka-native-component-tools.zip
   ```

4. Open a NodeJs command prompt window and change directory to the directory you just created.
5. Type `npm install` and wait for the npm installer to finish.

Using the Toolkit

You can define the following tasks:

- `new` – Generate a new component in the `src` folder, based on files in the template folders.
- `zip` – Create a Maestro Library in the dist folder by bundling all files in the `src` folder into a Zip file.
- `watch` – Helps automate the code generation and packaging process. It continuously monitors all files in the `src` folder structure and runs the zip task to generate a Maestro library, if it has been changed. It also continuously monitors `new-component.properties` and runs the new task to generate a new component if it has been changed.
- `redist` – Create a new toolkit distribution file by packaging all of the templates into a new toolkit. This is useful if you customize your templates and want to share them with others.

To run any of these tasks in the toolkit installation folder, type `gulp <task>`. A few examples are provided below:

```
gulp new
gulp zip
gulp watch
```

The `gulp new` task takes an optional parameter to identify which template to use as a model for your new component. Usage is `gulp new -templateName`, where `templateName` is the name of a sub-folder in the template folder. If you don't supply this optional parameter, a default (`-default`) is assumed, which uses the component template files in `template/default` directory.

NOTE

The `gulp new` task regenerates your component's source files from the files in the template folder that overwrites any changes in the '`src/<componentName>` directory. To avoid this, change the value of the '`component_filename` property in the '`new-component.properties` file immediately after you generate a new component, so the `gulp new` task uses the `component_filename` and it won't overwrite your original code.

To generate a new Maestro library with the default example component provided with the toolkit:

1. Type gulp new to create a new component, based on the values in the two .properties files, in a src folder.
2. Type gulp zip to package the new component as a Maestro library and named as the `dist/maestro-lib-<filename>.zip`.

To create a new component with customized `filenames` and `metadata`, follow the steps below:

1. Edit `library.properties` and `new-component.properties` to set your preferences.
2. Type gulp new to generate a new component in the src directory.
3. Edit the new component's files in the `src/widgets/<filename>` directory.
4. Type gulp zip to package all the components in the src directory as a Maestro library.

Next, learn how to create a native component library automatically.

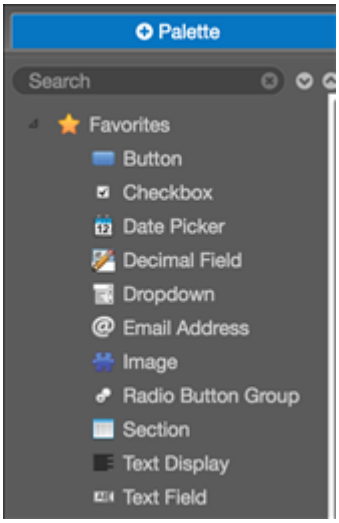
Native Component Development Using Eclipse or IntelliJ

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

This guide is for Eclipse or IntelliJ developers who would like to create a native component library for Maestro. It is assumed that the audience already knows:

- How to use an IDE such as Eclipse or IntelliJ
- Is familiar with Maestro, related features and terminology
- Can navigate around HTML, CSS and Javascript

Maestro comes with a rich set of components that are contained in core libraries assigned at the organisation and project level. What this means is that when you create forms, your palette is complete with all of the usual features that you would expect to find in web development such as text fields, radios etc.



There may however be scenarios in which components do not perform the function you need and it is impossible to override the behaviour of an existing component in order to get the result you need.

In this case you may need a native component and this document deals with the creation

There are three primary documents found across Confluence and in the knowledge base (see Related Articles) which touch on the parts of the process, but none which I think attack this from a viewpoint of assembling a basic component and understanding the processes involved in a

step by step fashion within an IDE. This guide will cover this and enable the developers to get up and running quickly.

This document will provide details on how to create components which:

- Are natively built outside of Maestro,
- Which enable best use of the Angular framework that's provided within Maestro AND,
- Which tap into the core benefits of Angular to provide rich two-way data binding.

This document does not deal with the rules and regulations around component creation.

Nor does it dive into details which are covered in other guides - it is intended to get development teams up and running.

Let's look at a few examples of native custom components.

Custom Button

we are going to create a button with some custom styling and add it to a library in Maestro.

1. Eclipse Setup - Create a new Web Project in Eclipse with the following directories.

   ```
   buildfiles >
   <Name>Library >
   <Name>Library > widgets >
   zip >
   ```

   Where <Name> is the name of the library

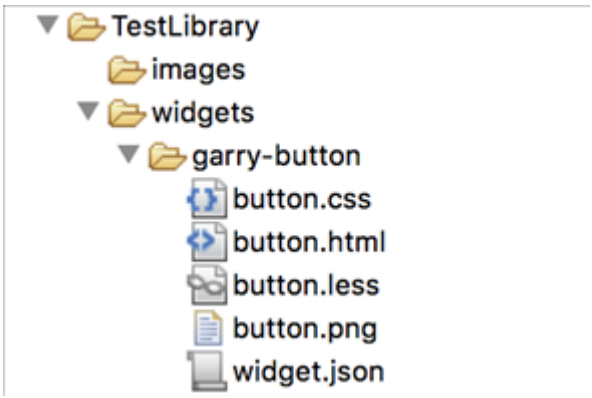
2. Create the ANT build file - Now create the ANT build file in buildfiles, and call it build<Name>.xml. Use this source and make sure that the project-name and folder-to-zip are the same names as your <Name>Library >

```
<project name="maestro-components" default="zip" basedir=".">
 <property name="project-name" value="TestLibrary" />
 <property name="folder-to-zip" value="TestLibrary" />
 <property name="dir" value="${basedir}\.."/>
 <property name="target-zip-folder" value="zip" />
 <target name="clean">
  <delete file="${project-name}.zip" />
  <delete dir="${target-zip-folder}" />
 </target>
 <target name="zip" depends="clean">

  <zip destfile="/${dir}/${target-zip-folder}/${project-name}.zip" basedir-
r="/${dir}/${folder-to-zip}" excludes="*.DS_Store,*.project" />
 </target>
</project>
```

This build file is responsible not just for zipping the correct structure, but ensuring that certain files which will cause the import to fail are excluded as well.

3. Create the custom component structure - In your <Name>Library > directory create this structure: `\ widgets \ <elementname>`
4. Baseline default files - And copy in the default files from the pack resources (css, html, less, png, json) to \ widgets \ <elementname>. Copy in the library config into the root of <Name>Library >. Add in a small image that will serve as the icon in Maestro palette. You should end up with a structure similar to this:



5. Create JSON Widget configuration - Modify widget.json to ensure that it references the resources in the pack, for example:

```
{
 "category": "Buttons",
    "label": "Garry Button",
    "html": "button.html",
    "icon": "button.png",
 "css": "button.css",
    "propDefs": {
        "icon": {
            "label": "Icon",
            "type": "icon",
            "imageStyleTarget": "Button icon",
            "value": "",
            "helpText": "Select the button icon"
        },
        "iconPos": {
            "label": "Icon Position",
            "type": "option",
            "value": "left",
            "options" : [{"label":"Left","value":"left"},{"label":"Right","-
value":"right"}],
            "helpText": "Select the button icon position"
        }
    },
    "defaultStyleTarget": "wdg-button",
    "span": {"xs": 12, "sm": 12, "md": 4, "lg": 4},
    "noData": true
}
```

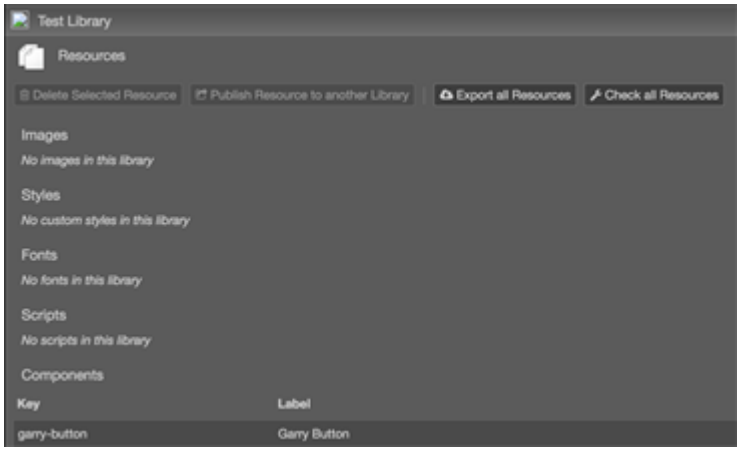
6. Create Widget HTML - Now modify the button.html to serve as your angular component.

```
<div>
 <button type="button" class="btn wdg-button">
  <i class="wdg-button-icon av-icon-button"
     data-ng-if="item.properties.icon && item.properties.iconPos == 'left'"
     data-ng-class="item.properties.icon"></i>
  <span data-av-compile class="wdg-button-label"
     data-ng-bind-html="item.htmlLabel || item.label"></span>
  <i class="wdg-button-icon av-icon-button"
     data-ng-if="item.properties.icon && item.properties.iconPos == 'right'"
     data-ng-class="item.properties.icon"></i>
 </button>
</div>
```

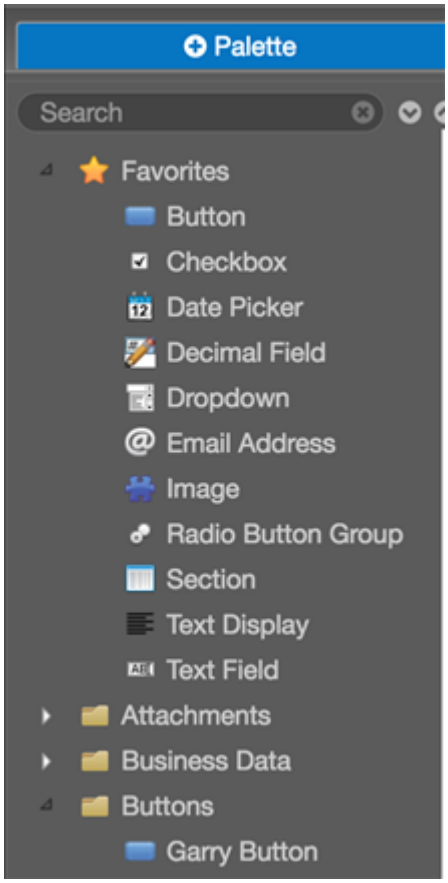
7. Create Widget Style (CSS) - Make sure button.css is as below:

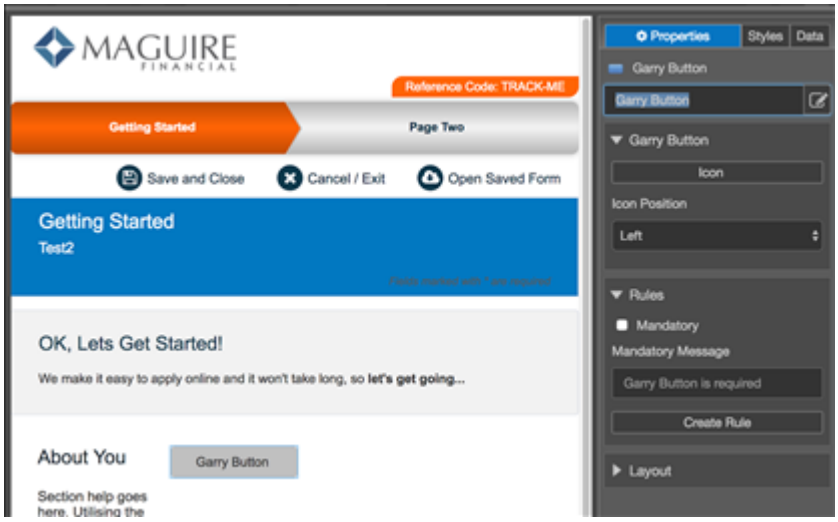
```
.wdg-button {
  width: 100%;
}
.wdg-button .wdg-button-label {
  padding-right: 4px;
}
.wdg-button .wdg-button-icon {
  display: inline-block;
  width: 28px;
  height: 28px;
  line-height: 28px;
  text-align: center;
  font-size: 18px;
}
.wdg-button .wdg-button-icon + .wdg-button-label {
  display: inline-block;
  padding: 4px 4px 0 5px;
  vertical-align: top;
}
```

8. Build the library - Run the ANT Build, and you should see a `TestLibrary.zip` file created within the `zip` directory
9. Import the library to Maestro - Access Maestro and go to Libraries. Now use the Import button and choose the ZIP file and once it's imported you should see your component listed.



10. Drag onto your form - Access Maestro and drag the component onto your form.





Angular Dynamic Button

In this next example, we will be providing a native component which uses angular to provide some additional functionality and data binding in the form of a count.

Follow the above exercise to step 4, but use the following button.html

```
<button ng-click="count = count + 1" ng-init="count=item.properties.count" class="btn
wdg-button">
  {{item.label}}
</button>
<span>
  count: {{count}}
</span>
```

Modify the original widget.json and use the one below. Note that you are changing the category, label and also adding in a count parameter which is referenced by the button.html binding to item.properties.count above. Use the other files associated with the previous example.

```
{
 "category": "Buttons",
    "label": "Angular Button",
    "html": "button.html",
    "icon": "button.png",
 "css": "button.css",
    "propDefs": {
        "icon": {
            "label": "Icon",
            "type": "icon",
            "imageStyleTarget": "Button icon",
            "value": "",
            "helpText": "Select the button icon"
        },
        "iconPos": {
            "label": "Icon Position",
            "type": "option",
            "value": "left",
            "options" : [{"label":"Left","value":"left"},{"label":"Right","-
value":"right"}],
            "helpText": "Select the button icon position"
        },

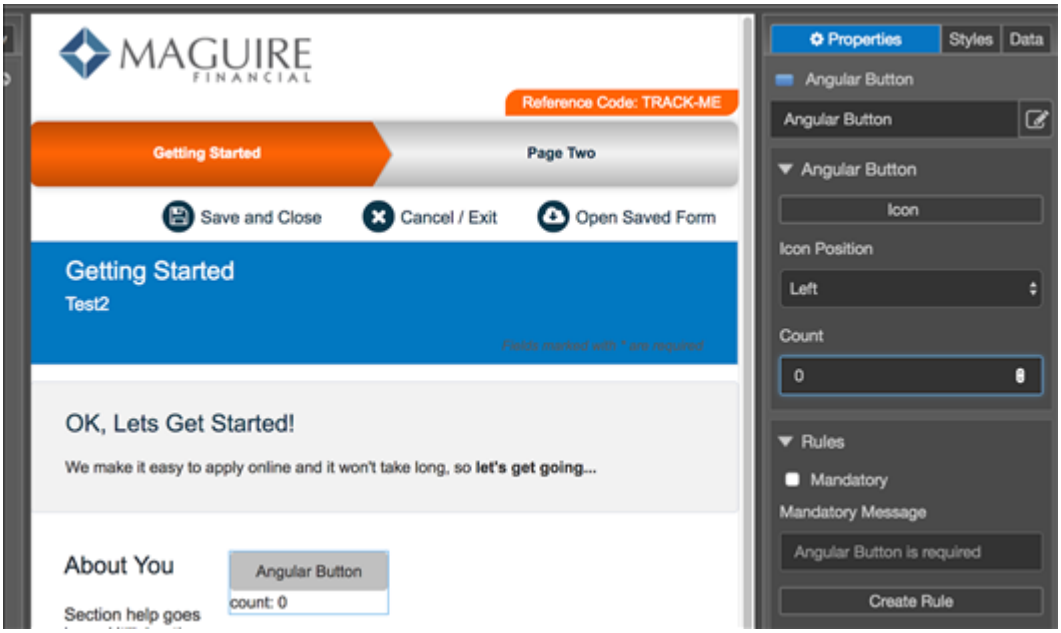


         "count":{
           "label": "Count",
             "type": "integer",
             "value": 0,
             "helpText":"Define the default value for the count"
         }

    },
    "defaultStyleTarget": "wdg-button",
    "span": {"xs": 12, "sm": 12, "md": 4, "lg": 4},
    "noData": true
}
```

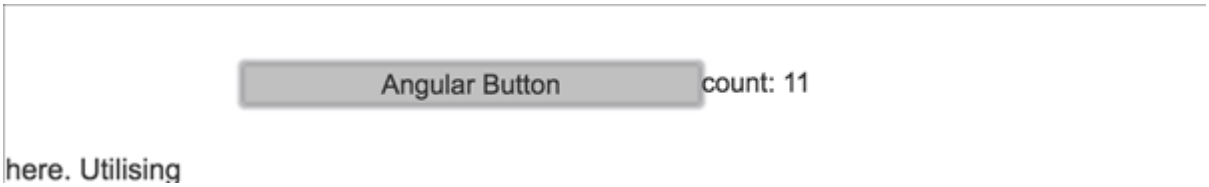
Now build the ZIP and import to Maestro.

From the buttons category drag across the Angular Button



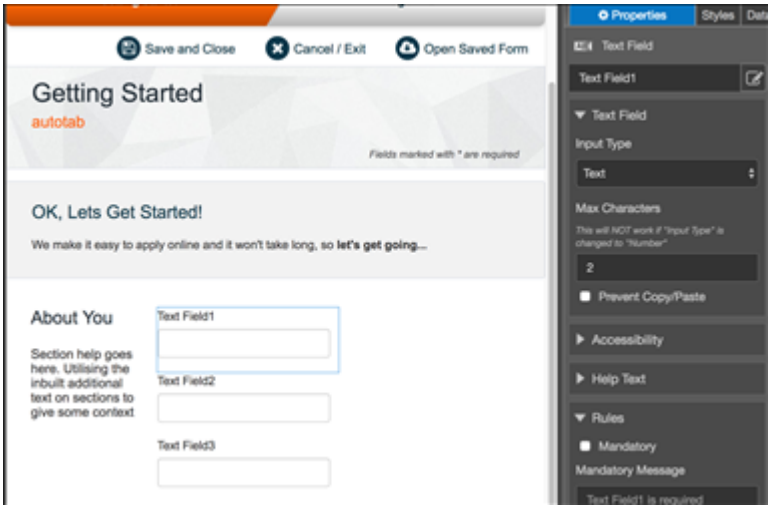
Now configure the count at 10 and preview the form:

Press the button and watch the dynamic count binding



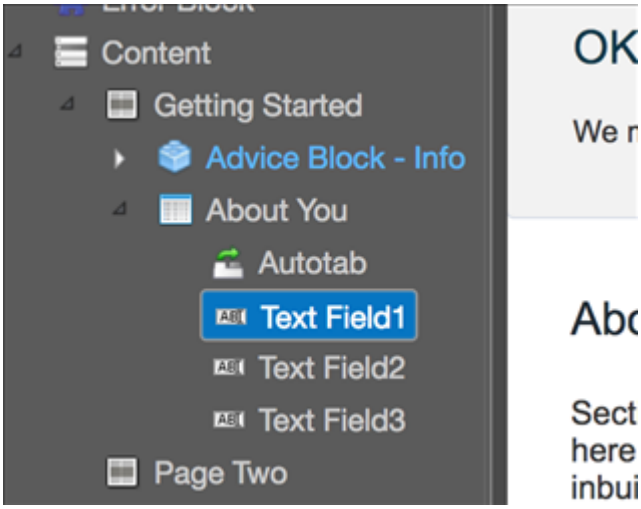
Auto-tabbing Component

The auto-tab component (autotab.zip) allows you to ensure that the cursor automatically moves between fields in Maestro once the maximum number of characters has been reached. In the example form below, this means that after 2 characters it will tab from text field 1 to 2, then 3 for you.

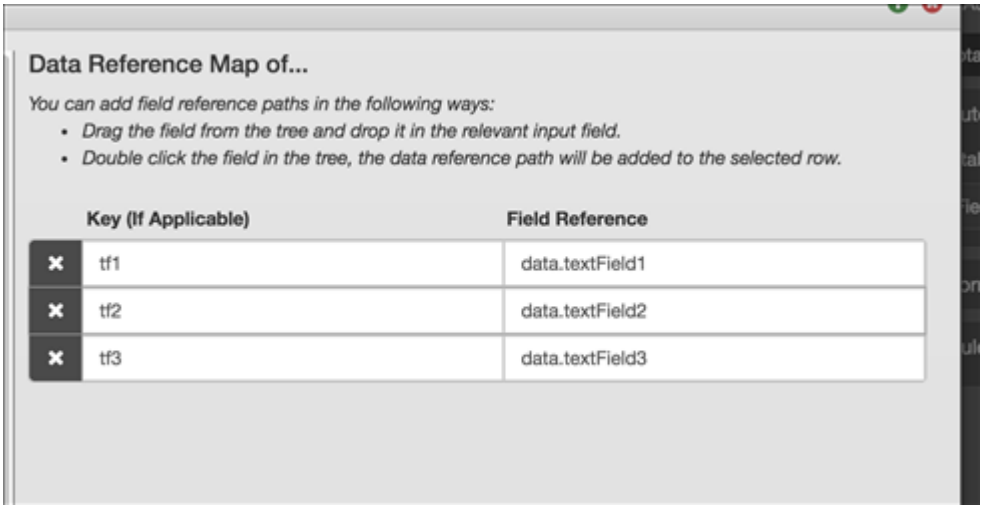


Once you have imported this component, set the max characters setting appropriately for each input field on the form.

Then drag in the autotab component



Now configure the auto tab component making sure that the autotab targets are set up with the fields in question, and that the data reference map has a field reference which correctly prefixes the data element (example below)



Pack Resources

Resource	Description
component.css	Default css file for component
component.html	Default html file for component
component.less	Additional header CSS
component.png	Default image for icon in Maestro for the component
widget.json	Configuration for the component
.library-config.xml	Default library configuration
MaestroComponents2.zip	Contains all files as above

You can download the MaestroComponents2.zip example here.

Next, learn how to create a native component library automatically.

List of Maestro Components

MaestroThe UI design product. | Template Designer Form Builder | 22.10 This feature was updated in 22.10.

Maestro comes with the following components, which you can use in your application forms:

Icon	Name	Category	Library
	ABN (Australian Business Number)	Business Data	core
	ACN (Australian Company Number)	Business Data	core
	Address (Australia)	Business Data	core
	Address (Global)	Business Data	core
	Address (USA)	Business Data	core
	Address Lookup	Deprecated	core
	Advice Block - Info	User Information	maguire
	Advice Block - Promo	User Information	maguire
	Attachment Field	Attachments	maguire
	Autocomplete Field	Input Fields	core
	Autotab Block	Containers	core
	Autotab Controller	Tools	core
	Avalon Info Dialog	Avalon Components	avalon
	Block (Mandatory)	Containers	core
	Block	Containers	core
	Button Group	Buttons	core
	Button	Buttons	core
	Cancel / Exit	Navigation Functions	navigation
	Cancel Confirm	Dialogs	maguire
	Cancel Success	Modal Pages	maguire
	Card Content Template	Cards	cards
	Card Content	Cards	cards
	Card Selector	Cards	cards
	Checkbox	Input Fields	core
	Chevron Navigator (Embedded)	Deprecated	navigation

Click To Call	Buttons	core
Content	Deprecated	navigation
Countdown Timer	Tools	core
Countries	Business Data	core
Currency Field	Input Fields	core
Data Field	Tools	core
Data-Driven Checkbox Group	Input Fields	core
Data-Driven Dropdown	Input Fields	core
Data-Driven Radio Group	Input Fields	core
Date Picker	Input Fields	core
Date of Birth Dropdown (Deprecated)	Deprecated	core
Date of Birth Dropdown	Business Data	core
Decimal Field	Input Fields	core
Delete Item	Deprecated	core
Dialog Content	hidden	core
Dialog Navigation Bar	Dialogs	maguire
Dialog	Containers	core
Dialog (Avalon)	Dialogs	avalon
Domain Object	Domain Model	domain
Dropdown (Years)	Business Data	core
Dropdown Bundle Navigation	Collaboration	core
Dropdown Multi Select	Input Fields	core
Dropdown	Input Fields	core
Dynamic Data Button	Buttons	core
Email Address	Business Data	core
Email PDF Receipt Button	Post Submission	maguire
Email Saved Form Button	Post Submission	maguire
Error Block	Navigation	navigation
Fieldset	Containers	core
Follow On Options Block	Post Submission	maguire
Footer	hidden	navigation

	Form Background Image	Display Fields	core
	Form Bundle Dialog	Collaboration	bundles
	Form Bundle Job Tasks	Collaboration	maguire
	Form Footer	Display Fields	core
	"Form Footer (Avalon)" on page 809	Display Fields	avalon
	Form Header	Display Fields	core
	Form Help	Dialogs	maguire
	Form Resume Button	Post Submission	maguire
	Form	Transaction	core
	Function Bar	Navigation	navigation
	Gender	Business Data	core
	Google Address Lookup	Business Data	core
	Graphical Radio Button Group	Input Fields	core
	Graphical Radio Button	Input Fields	core
	Header	hidden	navigation
	Horizontal Divide	Layout	core
	Icon Button	Buttons	core
	Image	Display Fields	core
	JavaScript Library	Tools	core
	Job Details Block	Collaboration	core
	Job Number	Collaboration	core
	Jump to Issue	Buttons	core
	Landing Page Image	Avalon Components	avalon
	Launch Child Form	hidden	core
	Link	Display Fields	core
	LinkedIn Auto-Fill	Identity	identity
	Mastersoft Address Lookup	Business Data	core
	Missing Field	hidden	core
	Mitek License Auto-Fill	Deprecated	identity
	Mobile Number (AU)	Business Data	core
	Mobile Number (US)	Business Data	core

	Modal Mobile Header	Navigation	navigation
	Modal Page	Containers	core
	Modal Page Logo	Avalon Components	avalon
	Multiline Text Area	Input Fields	core
	Navigation Wizard Bar	Navigation	navigation
	Navigator Step Count	Navigation	navigation
	Navigator Wizard Bar	Navigation	navigation
	Navigator	Navigation	navigation
	"Navigator (Avalon)" on page 872	Navigation	avalon
	Open Bundle Dialog Button	Collaboration	bundles
	Open Dialog	Dialogs	core
	Open Saved Form	Navigation Functions	navigation
	PDF Display	Display Fields	pdf
	Page Controller	Navigation	navigation
	Page Controller (Avalon)	Navigation	avalon
	Page Group	Navigation	navigation
	Page	Navigation	navigation
	Page (Avalon)	Navigation	avalon
	Payment Type Selector	Transaction	core
	Percentage Field	Input Fields	core
	Percentage Indicator	Display Fields	core
	Phone Number (AU)	Business Data	core
	Phone Number (US)	Business Data	core
	Popover Text Display	User Information	maguire
	Popover Widget Container	User Information	maguire
	Preview Test Block	Collaboration	core
	Progress Bar	Avalon Components	avalon
	Radio Button Group	Input Fields	core
	Radio Button [Repeating]	Input Fields	core
	Receipt Download Button	Post Submission	maguire
	Repeat Add Button	Repeats	core

	Repeat Delete Button	Repeats	core
	Repeat Index Text	Repeats	core
	Repeat	Repeats	core
	Repeating Block Template (Deprecated)	Deprecated	maguire
	Repeating Block Template	Repeats	core
	Route Select	Collaboration	core
	Save Challenge	Dialogs	maguire
	Save Confirm	Dialogs	maguire
	Save Success	Modal Pages	maguire
	Save and Close	Navigation Functions	navigation
	Second Level Navigator	Navigation	navigation
	Section Level 1	hidden	core
	Section	Containers	core
	Section (Avalon)	Containers	avalon
	Select Language	User Information	core
	Session Timeout Controller	Avalon Components	avalon
	Share Form Button	Post Submission	maguire
	Show XML	Dialogs	maguire
	Signature Pad	Business Data	core
	Slider	Input Fields	core
	Small Dialog	Dialogs	avalon
	State (AU)	Business Data	core
	State (Codes) (US)	Business Data	core
	State (US)	Business Data	core
	Step Name	Collaboration	core
	Sticky Panel	Containers	core
	Submit Success	Modal Pages	maguire
	Submit	Transaction	core
	TFN (Tax File Number) (AU)	Business Data	core
	Text Display	Display Fields	core
	Text Field	Input Fields	core

	Tile List	Collections	collections
	Time	Input Fields	core
	Title	Business Data	core
	Toggle Icon Button	Buttons	core
	Tramline Navigator (Embedded)	Deprecated	navigation
	Tramline Navigator	Navigation	navigation
	Video Capture	Business Data	core
	Web Font	Tools	core
	Yes/No Radio Buttons	Business Data	core
	iFrame	User Information	core

NOTE

The `avalon` library contains the Avalon template including all related components and resources.

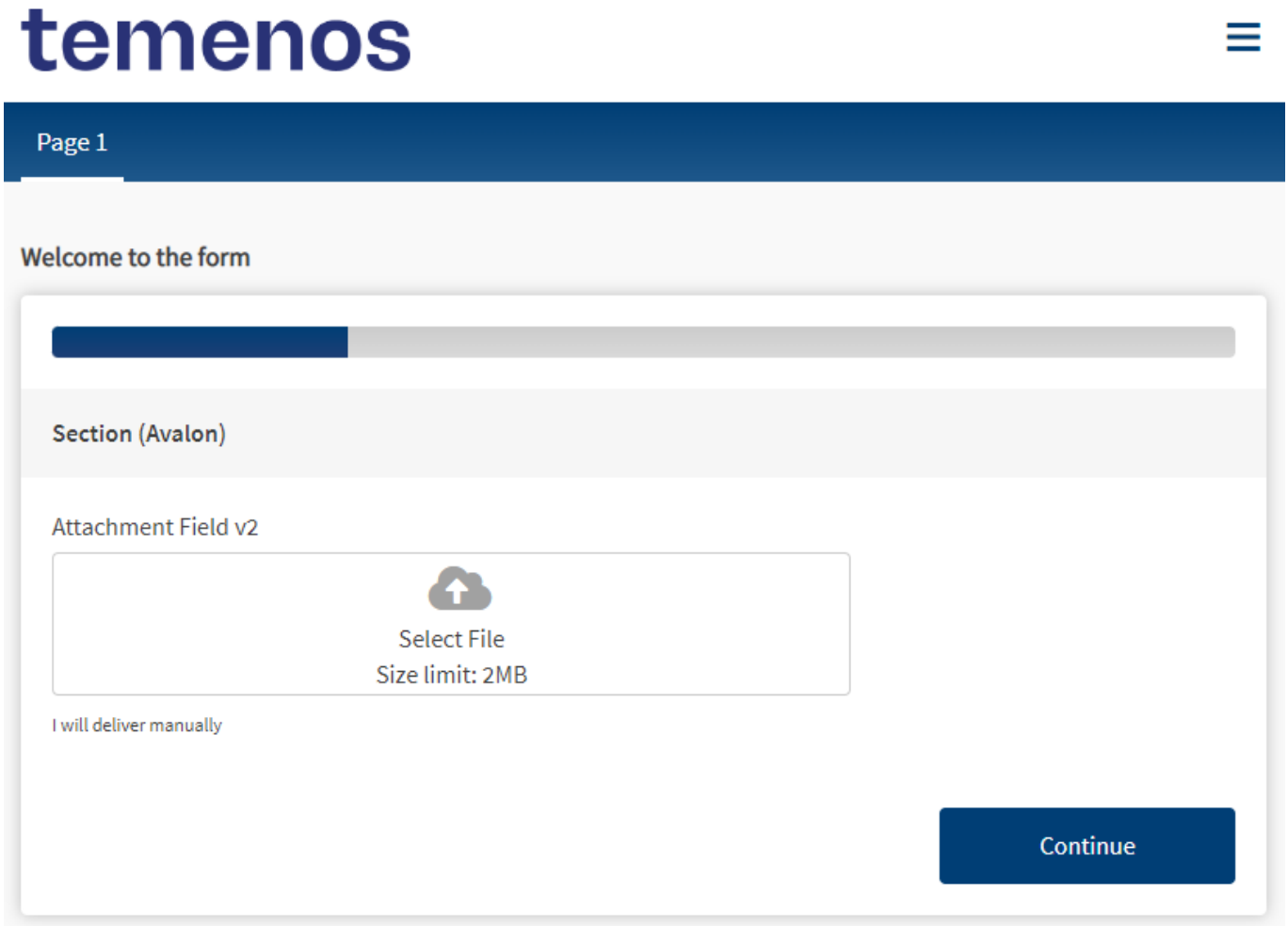
Next, learn about the Attachment Field component.

Attachment Field v2

MaestroThe UI design product. | Form Builder | 22.10 This feature was introduced in 22.10.

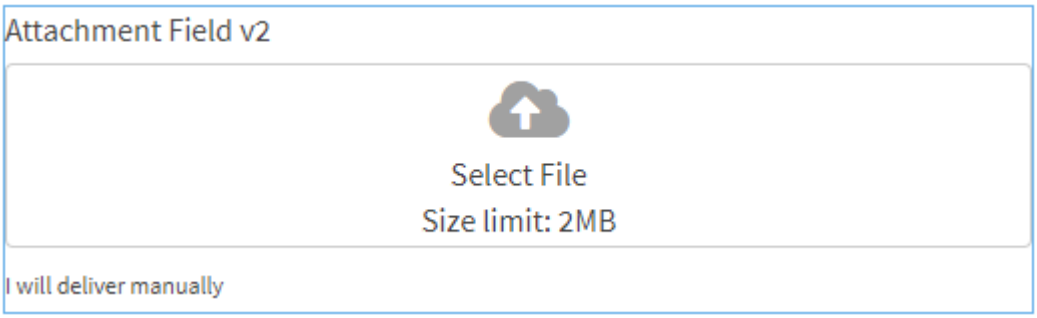
Maestro comes with the Attachment Field v2 component (widget) that allows a form user to upload multiple attachments, such as files and images, which are a part of an onboarding journey and are submitted along with a form. The Attachment Field v2 component extends functionality of the Attachment Field component.

A form with a rendered Attachment Field v2 widget is shown below:



Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Attachment Field v2** component within the **Attachments** folder.



3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane. For more information, see the Properties section below.
5. Configure the component's styles via the **Styles** pane. For more information, see the Styles section below.

Properties

The Attachment Field v2 component inherits all properties of the Attachment Field component, with the following additional properties:

Use Specified Attachment Filename
Leave blank for the original attached file name.

Drop Zone Drag Text
Button text for drag.

Drop Zone Icon Location
Select a location of the drop zone icon from the following options:

- Top
- Side

Fallback Error Message
fallbackErrorMessage

Fallback error message for translation

Additional Labels

Delete Attachment Confirmation Message
confirmDeleteAttachmentMessage

Confirm delete message for translation

Delete Attachment Button Tooltip
deleteAttachmentTooltip

Delete attachment button tooltip message for translation

Wait Message
waitMessage

Wait message for translation

Cancel Upload Button Label
cancelUploadButtonLabel

Label for cancel upload button for translation

Manual Attachment Button Label
manualAttachmentButtonLabel

Label for manual attachment button for translation

Upload Button Tooltip
uploadButtonTooltip

Tooltip for upload button for translation

Styles

The Attachment Field v2 component inherits all styles of the Attachment Field component, with the following additional styles:

Full Width Attachment Button

Adjust the width of the attachment button

```
form[name='form'] {
	.attachment-2-full-width-button {
		.wdg-drop-zone-container {
			width: 100%
		}
	}
}
```

Attachment Dropzone Hover

Adjust dropzone hover

```
@import (reference) "../core-brand-vars/core-brand-vars";

form[name='form'] {
       .attachment-2-dropzone-hover {
          .wdg-drop-zone-container {
          &:hover > .wdg-choose-file-button {
            //background-color: @av-secondary-contrast;
            background-color: @ux-primary-color;
            color: white;
          }
              &:hover .wdg-drop-zone-icon {
              color: white;
          }
        }
       }
}
```

Next, learn about the Countries component.

Attachment Field

MaestroThe UI design product. | Form Builder | 20.11 This feature was updated in 20.11.

Maestro comes with the Attachment Field component (widget) that allows a form user to upload multiple attachments, such as files and images, which are a part of an onboarding process and are submitted along with a form.

NOTE

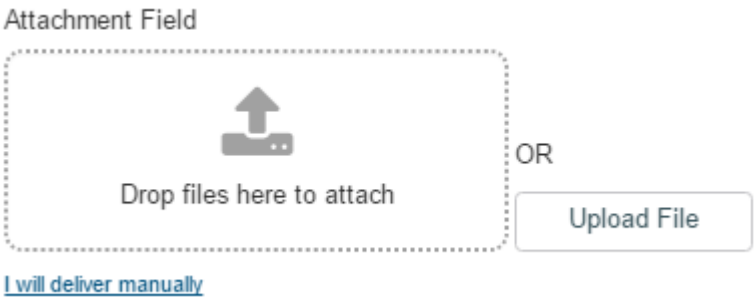
Since Maestro 19.05, all standard versions of the Attachment Field widget support a form space portal using CSRF[1] tokens.

NOTE

Starting in Maestro 20.11, the Attachment Field component is the part of the core library. Prior to this, it was part of the Maguire release library.

Usage

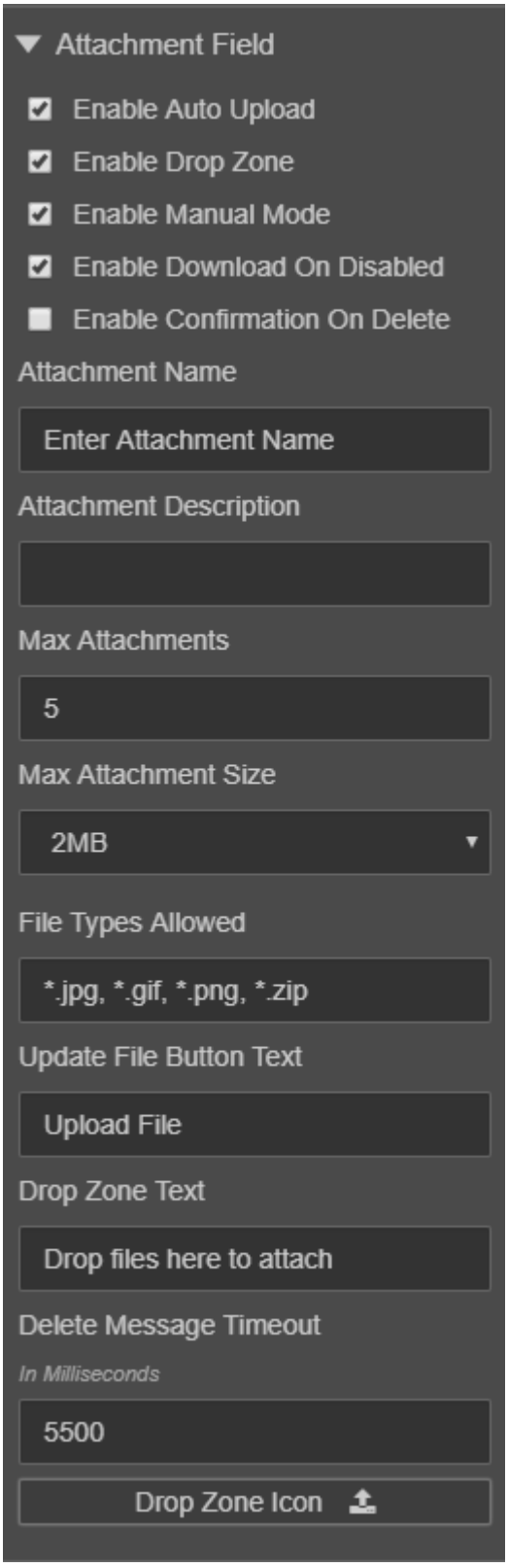
1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Attachment Field** component within the **Attachments** folder.



3. Drag the component into the **View** pane or the **Wireframe**.

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

4. Configure the component's properties via the **Properties** pane.



Properties

The following properties are available to the Attachment Field component:

Enable Auto Upload
Boolean enableAutoUpload

Enable the ability to auto upload attachments. It automatically uploads the attachment once the user drags it into the drop zone or uses the upload button. This option may not work when previewing the form, it only works once the TM form version is built.

Enable Drop Zone
Boolean showDropZone

Enable the ability to show attachment drop zone area, so the user can use the drop zone to drag and drop attachment files there.

Enable Manual Mode
Boolean enableManual

Enable manual attachment feature, which displays the I will deliver manually text on the field. The user can select this option to indicate that they will be sending the attachments via a different method, such as by email or post.

Enable Download On Disabled
Boolean allowDownloadDisabled

Enable download on disable, so users can download attachments from a disabled attachment field.

Enable Confirmation On Delete
Boolean allowConfirmDelete

enable confirmation on delete, so users will be prompted before an attachment is deleted.

Attachment Name
attachmentName

Enter the attachment name, which allows you to distinguish multiple attachment fields per form.

Attachment Description
attachmentDescription

Enter the attachment description, which allows you to distinguish multiple attachment fields per form.

Max Attachments
Integer queueLimit

Select the maximum number of attachments that a user can upload to a form which will be included in a form submission. Once the maximum number is met, the options to include additional attachments (drop zone and upload button) will disappear.

Max Attachment Size
Option maxSize

Select the maximum size of each attachment. If you want to set the maximum total size for all attachments that will need to be configured in Journey Manager.

The following options are available from the **Max Attachment Size** dropdown:

Label	Value
0.5MB	524288
1MB	1048576
2MB	2097152
3MB	3145728
4MB	4194304
5MB	5242880
6MB	6291456
7MB	7340032
8MB	8388608
9MB	9437184
10MB	10485760
15MB	15728640
20MB	20971520

Label	Value
25MB	26214400

File Types Allowed
fileTypes

Enter the file types allowed in the format provided. You can add more types by following the format displayed. For example, if you wanted to add PDF to the allowed types you would type *.pdf.

Update File Button Text
chooseFileButton

Enter text for the button used to upload files for attachment.

Drop Zone Text
dropZoneText

Enter the text for the drop zone used to upload files as attachments.

Delete Message Timeout
Integer deleteMessageTimeout

Delete Message Timeout

Drop Zone Icon
Icon dropZoneIcon

Select the button icon to display in the drop zone. Check with the Template Designer if you can change this icon, as icons are typically customized in the templates.

Error Messages

Invalid File
invalidFile

Error to display if the file is invalid

Invalid Size
invalidSize

Error to display if the file size is 0KB

Exceeds Maximum Size Limit
exceedsSizeLimit

Error to display if the file size exceeds the maximum limit

Exceeds Maximum Total Size Limit
exceedsTotalSizeLimit

Error to display if the total attachment size for this attachment name exceeds the maximum limit

Virus Warning
virusFoundOnUpload

Error to display if the attachment contains a virus when it is uploaded

Max Attachment Count Error
tooManyUploads

Error to display if there are too many attachments uploaded

Help Text

Popover Help Text
Help Text helpText

Success Messages

Successfully Uploaded
uploadSuccessful

Message to display if the file is successfully uploaded.

Successfully Removed
removeSuccessful

Message to display if the file is successfully removed

Thumbnail Options

Enable Image Preview
Boolean enableCanvas

Enable canvas image preview of any images uploaded in the attachment field in the form.

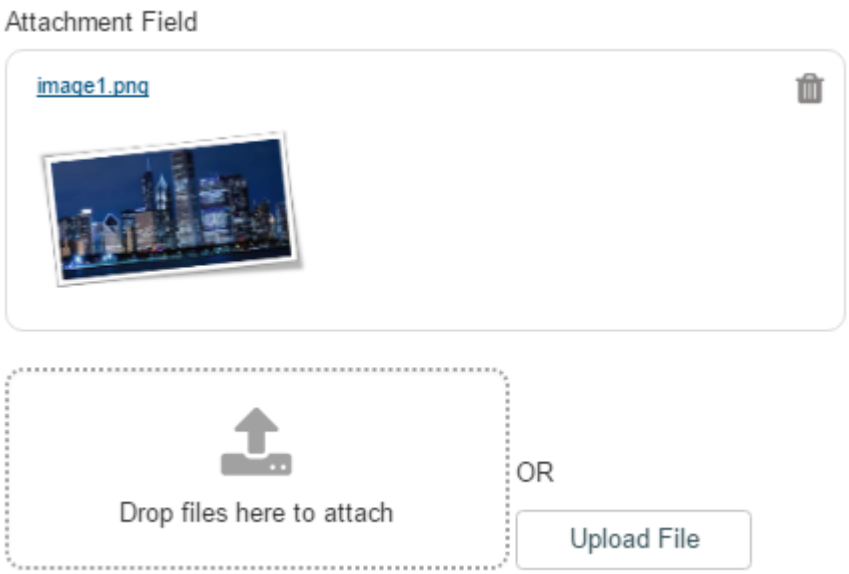


Image Preview Height (pixels)
Integer canvasHeight

Height for image to resize to in the preview mode.

Image Preview Width (pixels)
Integer canvasWidth

Width for image to resize to in the preview mode.

Transact Field | 20.11 This feature was removed in 20.11

Image Source
Option imageSource

Select the image source

The following options are available from the **Image Source** dropdown:

Label	Value
Camera	camera
Library	library

Maximum Dimension (pixels)
Integer maxDimension

Original attachment to be resized to this dimension.

Multi Select Attachments
Boolean multiSelect

Ability to multi select attachments

NOTE

You can also use the Attachment Field v2 widget that has additional functionality. | 22.10
This feature was introduced in 22.10..

Next, learn about the Countries component.

Landing Page Image

MaestroThe UI design product. + Avalon TemplateThis topic is related to the Avalon Template used in Maestro. | Form Builder | 22.04 This feature was introduced in 22.04.

Maestro comes with the Landing Page Image component that allows you to ? according to a form's navigation logic to build a user journey flow. This component is available when you use the Avalon Template. The Landing Page Image component extends functionality of the ? component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Landing Page Image** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.
5. Configure the component's styles via the **Styles** pane.

Properties

The following properties are available to this component:

Display Text

Title Text
Rich Text titleText

This text is used as a title. If the text is not provided, the default `{{T[item.id].label}}` value is used.

Subtitle Text
Rich Text subtitleText

This text is used as a subtitle. If the text is not provided, the default `<h1 ng-bind-html="T[Form.$Pages.currentPage.id].properties.formName">Form Name</h1>` value is used.

NOTE

You can use HTML tags in the Subtitle Text component, for example, `<samp>Welcome</samp> to <em>my</em> page!`. For more information, see rich text editor.

Additional Text
Rich Text additionalText

This text is used as an additional text

Help Text
Rich Text helpText

This text is used as a help text

Options

Hide Navigator
Boolean offMenu

A page that is in the flow of the form but not displayed in the menu

Page View

Hide From Navigator
Boolean hideFromNavigator

A page that is in the flow of the form but not displayed in the Navigator

Hide From Header
Boolean hideFromHeader

Hide or show the form header (if applicable for the template)

Hide From Footer
Boolean hideFromFooter

Hide or show the form footer (if applicable for the template)

Hide Navigation Buttons
Boolean hideNavigationButtons

Hide or show the navigational buttons

Hide Menu Button
Boolean hideFromHeader

Hide or show the form menu button

Page of No Return
Boolean pageNoReturn

The user will not be able to navigate back past this page

Navigator

Navigator Text
Rich Text

This text will be shown in the Navigator in place of the page label

Styles

The following styles are available to this component:

Avalon Page



Next, learn about the "Save and Close" on page 887 component.

Modal Page Logo

MaestroThe UI design product. + Avalon TemplateThis topic is related to the Avalon Template used in Maestro. | Form Builder | 22.04 This feature was introduced in 22.04.

Maestro comes with the Landing Page Modal Page Logocomponent that allows you to ? according to a form's navigation logic to build a user journey flow. This component is available when you use the Avalon Template. The Modal Page Logo component extends functionality of the ? component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Modal Page Logo** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.
5. Configure the component's styles via the **Styles** pane.

Properties

The following properties are available to this component:

Display Text

Title Text Rich Text titleText
This text is used as a title. If the text is not provided, the default `{{T[item.id].label}}` value is used.

Subtitle Text Rich Text subtitleText
This text is used as a subtitle. If the text is not provided, the default `<h1 ng-bind-html="T[Form.$Pages.currentPage.id].properties.formName">Form Name</h1>` value is used.

NOTE

You can use HTML tags in the Subtitle Text component, for example, `<samp>Welcome</samp> to <em>my</em> page!`. For more information, see rich text editor.

Additional Text
Rich Text additionalText

This text is used as an additional text

Help Text
Rich Text helpText

This text is used as a help text

Options

Hide Navigator
Boolean offMenu

A page that is in the flow of the form but not displayed in the menu

Page View

Hide From Navigator
Boolean hideFromNavigator

A page that is in the flow of the form but not displayed in the Navigator

Hide From Header
Boolean hideFromHeader

Hide or show the form header (if applicable for the template)

Hide From Footer
Boolean hideFromFooter

Hide or show the form footer (if applicable for the template)

Hide Navigation Buttons
Boolean hideNavigationButtons

Hide or show the navigational buttons

Hide Menu Button
Boolean hideFromHeader

Hide or show the form menu button

Page of No Return
Boolean pageNoReturn

The user will not be able to navigate back past this page

Navigator

Navigator Text
Rich Text

This text will be shown in the Navigator in place of the page label

Styles

The following styles are available to this component:

Avalon Page



Next, learn about the "Save and Close" on page 887 component.

Avalon Info Dialog

MaestroThe UI design product. + Avalon TemplateThis topic is related to the Avalon Template used in Maestro. | Form Builder | 22.04 This feature was introduced in 22.04.

Maestro comes with the Avalon Info Dialog component that allows you to ? according to a form's navigation logic to build a user journey flow. This component is available when you use the Avalon Template. The Avalon Info Dialog component extends functionality of the ? component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Avalon Info Dialog** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.
5. Configure the component's styles via the **Styles** pane.

Properties

The following properties are available to this component:

Display Text

Title Text
Rich Text titleText

This text is used as a title. If the text is not provided, the default `{{T[item.id].label}}` value is used.

Subtitle Text
Rich Text subtitleText

This text is used as a subtitle. If the text is not provided, the default `<h1 ng-bind-html="T[Form.$Pages.currentPage.id].properties.formName">Form Name</h1>` value is used.

NOTE

You can use HTML tags in the Subtitle Text component, for example, `<samp>Welcome</samp> to <em>my</em> page!`. For more information, see rich text editor.

Additional Text
Rich Text additionalText

This text is used as an additional text

Help Text
Rich Text helpText

This text is used as a help text

Options

Hide Navigator
Boolean offMenu

A page that is in the flow of the form but not displayed in the menu

Page View

Hide From Navigator
Boolean hideFromNavigator

A page that is in the flow of the form but not displayed in the Navigator

Hide From Header
Boolean hideFromHeader

Hide or show the form header (if applicable for the template)

Hide From Footer
Boolean hideFromFooter

Hide or show the form footer (if applicable for the template)

Hide Navigation Buttons
Boolean hideNavigationButtons

Hide or show the navigational buttons

Hide Menu Button
Boolean hideFromHeader

Hide or show the form menu button

Page of No Return
Boolean pageNoReturn

The user will not be able to navigate back past this page

Navigator

Navigator Text
Rich Text

This text will be shown in the Navigator in place of the page label

Styles

The following styles are available to this component:

Avalon Page



Next, learn about the "Save and Close" on page 887 component.

Progress Bar

MaestroThe UI design product. + Avalon TemplateThis topic is related to the Avalon Template used in Maestro. | Form Builder | 22.04 This feature was introduced in 22.04.

Maestro comes with the Progress Bar component that allows you to ? according to a form's navigation logic to build a user journey flow. This component is available when you use the Avalon Template. The Progress Bar component extends functionality of the ? component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Progress Bar** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.
5. Configure the component's styles via the **Styles** pane.

Properties

The following properties are available to this component:

Display Text

Title Text Rich Text titleText
This text is used as a title. If the text is not provided, the default `{{T[item.id].label}}` value is used.

Subtitle Text Rich Text subtitleText
This text is used as a subtitle. If the text is not provided, the default `<h1 ng-bind-html="T[Form.$Pages.currentPage.id].properties.formName">Form Name</h1>` value is used.

NOTE

You can use HTML tags in the Subtitle Text component, for example, `<samp>Welcome</samp> to <em>my</em> page!`. For more information, see rich text editor.

Additional Text
Rich Text additionalText

This text is used as an additional text

Help Text
Rich Text helpText

This text is used as a help text

Options

Hide Navigator
Boolean offMenu

A page that is in the flow of the form but not displayed in the menu

Page View

Hide From Navigator
Boolean hideFromNavigator

A page that is in the flow of the form but not displayed in the Navigator

Hide From Header
Boolean hideFromHeader

Hide or show the form header (if applicable for the template)

Hide From Footer
Boolean hideFromFooter

Hide or show the form footer (if applicable for the template)

Hide Navigation Buttons
Boolean hideNavigationButtons

Hide or show the navigational buttons

Hide Menu Button
Boolean hideFromHeader

Hide or show the form menu button

Page of No Return
Boolean pageNoReturn

The user will not be able to navigate back past this page

Navigator

Navigator Text
Rich Text

This text will be shown in the Navigator in place of the page label

Styles

The following styles are available to this component:

Avalon Page



Next, learn about the "Save and Close" on page 887 component.

Session Timeout Controller

MaestroThe UI design product. + Avalon TemplateThis topic is related to the Avalon Template used in Maestro. | Form Builder | 22.04 This feature was introduced in 22.04.

Maestro comes with the Session Timeout Controller component that allows you to ? according to a form's navigation logic to build a user journey flow. This component is available when you use the Avalon Template. The Session Timeout Controller component extends functionality of the ? component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Session Timeout Controller** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.
5. Configure the component's styles via the **Styles** pane.

Properties

The following properties are available to this component:

Display Text

Title Text
Rich Text titleText

This text is used as a title. If the text is not provided, the default `{{T[item.id].label}}` value is used.

Subtitle Text
Rich Text subtitleText

This text is used as a subtitle. If the text is not provided, the default `<h1 ng-bind-html="T[Form.$Pages.currentPage.id].properties.formName">Form Name</h1>` value is used.

NOTE

You can use HTML tags in the Subtitle Text component, for example, `<samp>Welcome</samp> to <em>my</em> page!`. For more information, see rich text editor.

Additional Text
Rich Text additionalText

This text is used as an additional text

Help Text
Rich Text helpText

This text is used as a help text

Options

Hide Navigator
Boolean offMenu

A page that is in the flow of the form but not displayed in the menu

Page View

Hide From Navigator
Boolean hideFromNavigator

A page that is in the flow of the form but not displayed in the Navigator

Hide From Header
Boolean hideFromHeader

Hide or show the form header (if applicable for the template)

Hide From Footer
Boolean hideFromFooter

Hide or show the form footer (if applicable for the template)

Hide Navigation Buttons
Boolean hideNavigationButtons

Hide or show the navigational buttons

Hide Menu Button
Boolean hideFromHeader

Hide or show the form menu button

Page of No Return
Boolean pageNoReturn

The user will not be able to navigate back past this page

Navigator

Navigator Text
Rich Text

This text will be shown in the Navigator in place of the page label

Styles

The following styles are available to this component:

Avalon Page



Next, learn about the "Save and Close" on page 887 component.

Countries

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Countries component.

NOTE

This component inherits properties and shared styles from the Dropdown component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Countries** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Countries component:

Options
List options

The options list for the dropdown

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Sort Options
Boolean sortOptions

Check to automatically sort options by label

Icon
Icon icon

Select the dropdown icon

Add Blank Value
Boolean addBlankValue

Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the Gender component.

Gender

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Gender component.

NOTE

This component inherits properties and shared styles from the Dropdown component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Gender** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Gender component:

Options
List options

The options list for the dropdown

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Sort Options
Boolean sortOptions

Check to automatically sort options by label

Icon
Icon icon

Select the dropdown icon

Add Blank Value
Boolean addBlankValue

Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "State (Codes) (US)" on the next page component.

State (Codes) (US)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the State (Codes) (US) component.

NOTE

This component inherits properties and shared styles from the Dropdown component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **State (Codes) (US)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the State (Codes) (US) component:

Options
List options

The options list for the dropdown

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Sort Options
Boolean sortOptions

Check to automatically sort options by label

Icon
Icon icon

Select the dropdown icon

Add Blank Value
Boolean addBlankValue

Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "State (US)" on the next page component.

State (US)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the State (US) component.

NOTE

This component inherits properties and shared styles from the Dropdown component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **State (US)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the State (US) component:

Options
List options

The options list for the dropdown

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Sort Options
Boolean sortOptions

Check to automatically sort options by label

Icon
Icon icon

Select the dropdown icon

Add Blank Value
Boolean addBlankValue

Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "State (AU)" on the next page component.

State (AU)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the State (AU) component.

NOTE

This component inherits properties and shared styles from the Dropdown component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **State (AU)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the State (AU) component:

Options
List options

The options list for the dropdown

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Sort Options
Boolean sortOptions

Check to automatically sort options by label

Icon
Icon icon

Select the dropdown icon

Add Blank Value
Boolean addBlankValue

Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Title" on the next page component.

Title

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Title component.

NOTE

This component inherits properties and shared styles from the Dropdown component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Title** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Title component:

Options
List options

The options list for the dropdown

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Sort Options
Boolean sortOptions

Check to automatically sort options by label

Icon
Icon icon

Select the dropdown icon

Add Blank Value
Boolean addBlankValue

Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Yes/No Radio Buttons" on the next page component.

Yes/No Radio Buttons

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Yes/No Radio Buttons component.

NOTE

This component inherits properties and shared styles from the Radio Button Group component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Yes/No Radio Buttons** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Yes/No Radio Buttons component:

Options
List options

List of options

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Appearance

Arrangement
Option arrangement

Defines the layout arrangement of the radio options on the page.

The following options are available from the **Arrangement** dropdown:

Label	Value
Vertical (1 per line)	vertical
Horizontal (natural width)	horizontal
2 Column Grid	6
3 Column Grid	4
4 Column Grid	3
6 Column Grid	2

Min Width (px)
Integer minWidth

Radio Button Minimum Width in Pixels.

Stack Threshold
Option stackThreshold

Select which responsive threshold the radio options will stack on top of each other, or 'None' where stacking is not desired.

The following options are available from the **Stack Threshold** dropdown:

Label	Value
Small	sm
Medium	md
Large	lg
None	none

Selected Icon Class
Icon selectedIconClass

Selected icon class name

Unselected Icon Class Icon unselectedIconClass
Unselected icon class name

Help Text

Popover Help Text Help Text helpText

Next, learn about the "Email Address" on the next page component.

Email Address

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Email Address component.

NOTE

This component inherits properties and shared styles from the Text Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Email Address** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Email Address component:

Input Type
Option inputType

Select the input type

The following options are available from the **Input Type** dropdown:

Label	Value
Text	text
Number	number
Email	email
Password	password
Telephone	tel

Max Characters
Integer maxLength

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Mobile Number (AU)" on the next page component.

Mobile Number (AU)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Mobile Number (AU) component.

NOTE

This component inherits properties and shared styles from the Text Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Mobile Number (AU)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Mobile Number (AU) component:

Input Type
Option inputType

Select the input type

The following options are available from the **Input Type** dropdown:

Label	Value
Text	text
Number	number
Email	email
Password	password
Telephone	tel

Max Characters
Integer maxLength

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Mobile Number (US)" on the next page component.

Mobile Number (US)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Mobile Number (US) component.

NOTE

This component inherits properties and shared styles from the Text Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Mobile Number (US)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Mobile Number (US) component:

Input Type
Option inputType

Select the input type

The following options are available from the **Input Type** dropdown:

Label	Value
Text	text
Number	number
Email	email
Password	password
Telephone	tel

Max Characters
Integer maxLength

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Phone Number (AU)" on the next page component.

Phone Number (AU)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Phone Number (AU) component.

NOTE

This component inherits properties and shared styles from the Text Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Phone Number (AU)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Phone Number (AU) component:

Input Type
Option inputType

Select the input type

The following options are available from the **Input Type** dropdown:

Label	Value
Text	text
Number	number
Email	email
Password	password
Telephone	tel

Max Characters
Integer maxLength

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Phone Number (US)" on the next page component.

Phone Number (US)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Phone Number (US) component.

NOTE

This component inherits properties and shared styles from the Text Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Phone Number (US)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Phone Number (US) component:

Input Type
Option inputType

Select the input type

The following options are available from the **Input Type** dropdown:

Label	Value
Text	text
Number	number
Email	email
Password	password
Telephone	tel

Max Characters
Integer maxLength

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Address (Global)" on the next page component.

Address (Global)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Address (Global) component.

NOTE

This component inherits properties and shared styles from the Fieldset component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Address (Global)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Address (Global) component:

Help Text

Popover Help Text Help Text helpText

Next, learn about the "Address (USA)" on the next page component.

Address (USA)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Address (USA) component.

NOTE

This component inherits properties and shared styles from the Fieldset component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Address (USA)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Address (USA) component:

Help Text

Popover Help Text Help Text helpText

Next, learn about the "Address (Australia)" on the next page component.

Address (Australia)

MaestroThe UI design product. | Form Builder | 18.11 This feature was introduced in 18.11.

Maestro comes with the Address (Australia) component.

NOTE

This component inherits properties and shared styles from the Fieldset component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Address (Australia)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Address (Australia) component:

Help Text

Popover Help Text Help Text helpText

Next, learn about the "ABN (Australian Business Number)" on the next page component.

ABN (Australian Business Number)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the ABN (Australian Business Number) component.

NOTE

This component inherits properties and shared styles from the Text Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **ABN (Australian Business Number)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the ABN (Australian Business Number) component:

Input Type
Option inputType

Select the input type

The following options are available from the **Input Type** dropdown:

Label	Value
Text	text
Number	number
Email	email
Password	password
Telephone	tel

Max Characters
Integer maxLength

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the ACN (Australian Company Number) component.

ACN (Australian Company Number)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the ACN (Australian Company Number) component.

NOTE

This component inherits properties and shared styles from the Text Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **ACN (Australian Company Number)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the ACN (Australian Company Number) component:

Input Type
Option inputType

Select the input type

The following options are available from the **Input Type** dropdown:

Label	Value
Text	text
Number	number
Email	email
Password	password
Telephone	tel

Max Characters
Integer maxLength

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Google Address Lookup" on the next page component.

Google Address Lookup

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Google Address Lookup component.

NOTE

This component inherits properties and shared styles from the Fieldset component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Google Address Lookup** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Google Address Lookup component:

Always Mandatory
Boolean addressMandatory

Google Configuration

Country Code Filter
Text countryCodeFilter

Enter the country code to filter the search results

Language Code
Text language

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Mastersoft Address Lookup" on the next page component.

Mastersoft Address Lookup

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Mastersoft Address Lookup component.

NOTE

This component inherits properties and shared styles from the Fieldset component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Mastersoft Address Lookup** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Mastersoft Address Lookup component:

Always Mandatory
Boolean addressMandatory

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Dropdown (Years)" on the next page component.

Dropdown (Years)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Dropdown (Years) component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Dropdown (Years)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Dropdown (Years) component:

Starting Year Integer minYear
Please enter the Starting Year

Use Descending Order Boolean useDescending
Select to use descending order

Icon Icon icon
Select the dropdown icon

Add Blank Value Boolean addBlankValue
Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Next, learn about the "Date of Birth Dropdown" on the next page component.

Date of Birth Dropdown

MaestroThe UI design product. | Form Builder | 17.10 This feature was introduced in 17.10.

Maestro comes with the Date of Birth Dropdown component (widget).

NOTE

This component inherits properties and shared styles from the Block (Mandatory) component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Date of Birth Dropdown** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Date of Birth Dropdown component:

Help Text

Popover Help Text
Help Text helpText

Options

Mandatory Setting
Option mandatoryOption

Select how many fields are mandatory.

The following options are available from the **Mandatory Setting** dropdown:

Label	Value
At Least One Mandatory	one

Label	Value
All Mandatory	all

Hide Mandatory On Labels
Boolean showChildrenIndicators

Enable mandatory asterisk on child element labels.

Validation

Min Age
Integer minAge

Enter the minimum age allowed.

Max Age
Integer maxAge

Enter the maximum age allowed.

Must be a past date
Boolean pastDate

Check to validate that date is in the past.

Next, learn about the "Video Capture" on the next page component.

Video Capture

MaestroThe UI design product. | Form Builder | 19.05 This feature was introduced in 19.05.

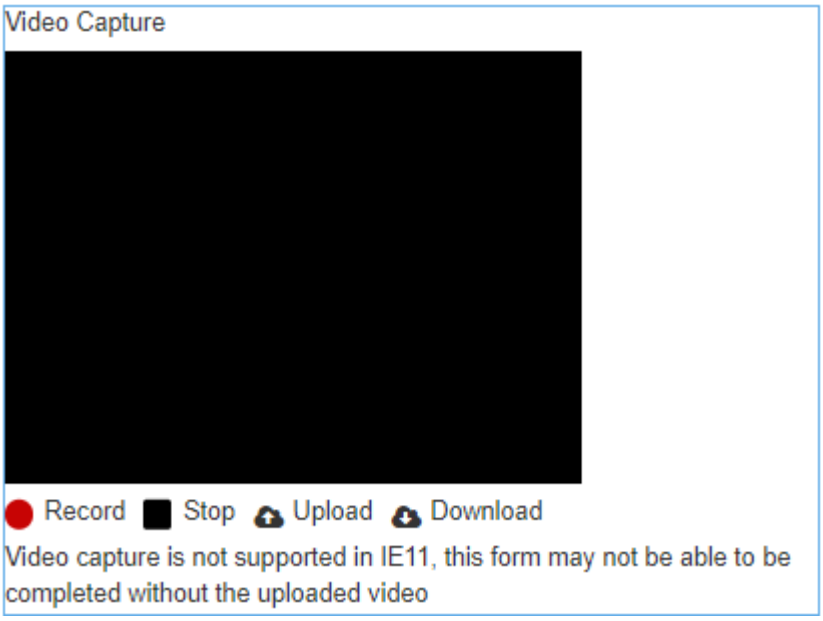
Maestro comes with the Video Capture component that can capture video and audio, which allows you to verify the user's identity via a gesture. You can also add the rules that fire when a video is uploaded to Manager or whether a webcam is not available after clicking record.

NOTE

The Video Capture component is not supported in E11 browser. For more information, see Temenos Journey Manager forms system requirements.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Video Capture** component within the **Business Data** folder.



3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Video Capture component:

Record Audio
Boolean audio

Select to record audio.

Record Video
Boolean video

Select to record video.

Video Width (px)
Integer videoWidth

Customize the video size.

Video Height (px)
Integer videoHeight

Customize the video size.

Record Delay (seconds)
Integer recordDelay

Specify delay recording in seconds.

Max Record Length (seconds)
Integer maxLength

Set a max length of recording in seconds.

Record Button Text
recordBtnText

Enter the record button text

Stop Button Text
stopBtnText

Enter the stop button text

Upload Button Text
uploadBtnText

Enter the upload button text

Download Button Text
downloadBtnText

Enter the download button text

Attachment Metadata

Attachment Name
attachmentName

Enter the attachment name

Attachment Description
attachmentDescription

Enter the attachment description

Configuration

Enable Progress Message
Boolean enableProgress

Check to enable loading message popup

Progress Message
Text progressMsg

Text to display in loading message

Next, learn about the "TFN (Tax File Number) (AU)" on the next page component.

TFN (Tax File Number) (AU)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the TFN (Tax File Number) (AU) component.

NOTE

This component inherits properties and shared styles from the Text Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **TFN (Tax File Number) (AU)** component within the **Business Data** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the TFN (Tax File Number) (AU) component:

Input Type
Option inputType

Select the input type

The following options are available from the **Input Type** dropdown:

Label	Value
Text	text
Number	number
Email	email
Password	password
Telephone	tel

Max Characters
Integer maxLength

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Signature Pad" on the next page component.

Signature Pad

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

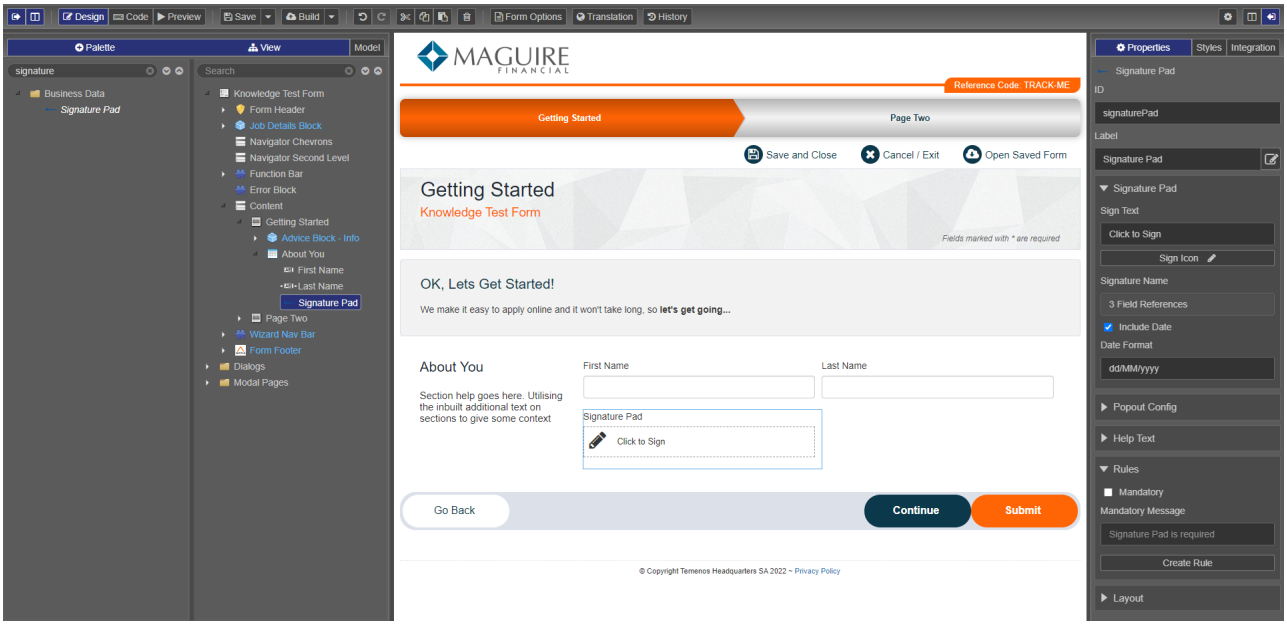
Maestro comes with the Signature Pad component enabling a form user to sign an application form digitally. This component allows the user to create a signature by drawing it on the pad, saving it, clearing the pad to redraw the signature, or canceling it. The image below shows how the Signature Pad component appears on a rendered form.



Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Signature Pad** component within the **Business Data** folder.

3. Drag the component into the **View** pane or the **Wireframe**.



4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Signature Pad component:

Sign Text
Text signText

The text used for empty signature box

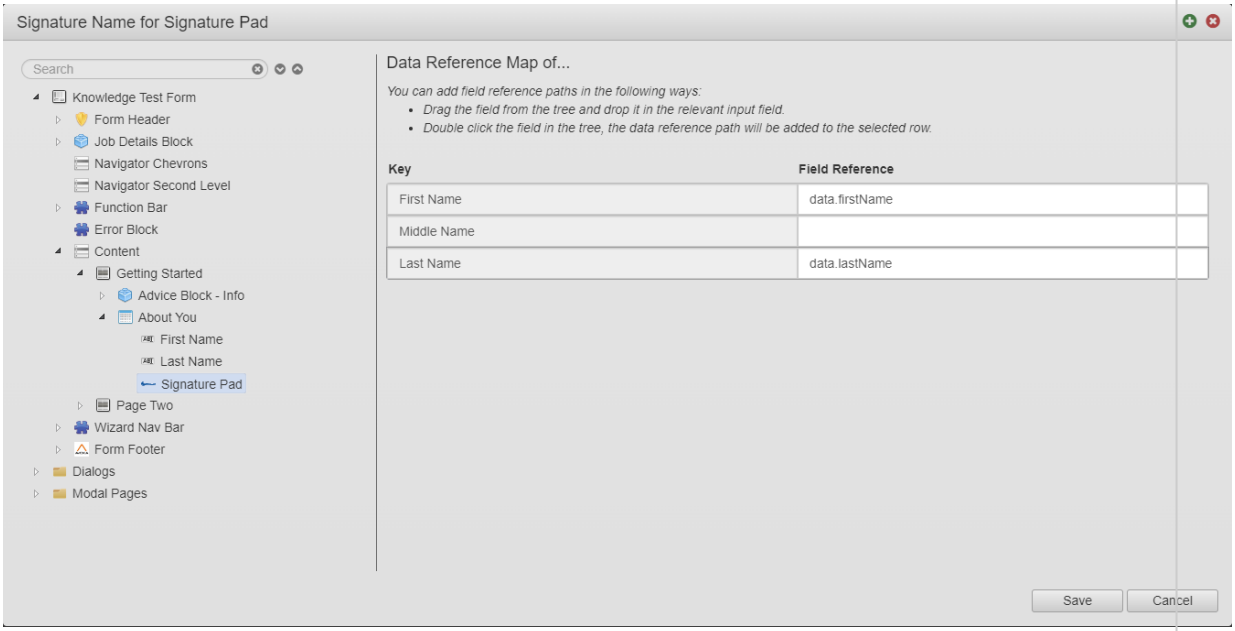
Sign Icon
Icon signIcon

The icon used for empty signature box

Signature Name
Field Ref Map signatureNameRef

Map form's field references for first, middle and last names that is displayed in the rendered Signature Pad component.

1. Click 3 Field References to map up-to 3 field references in the Signature Name for Signature Pad window.



2. Drag and drop a field from the Navigation bar to the Field Reference column against a Key you want to map. Repeat it for other references if needed.
3. Click Save to update the mapping.
4. Preview the form and Click to Sign to test the Signature Pad.



You can now see the first and last names appear at the bottom left corner of the component.

Include Date

Boolean includeDate

Select the Include Date checkbox to show the date when the user signs the form at the bottom left corner of the component, as shown above.

NOTE

Every time the user updates a signature, the signature timestamp (`data.signaturePad.signatureDate`) is updated. You can reference this value in other JavaScript code by right-clicking the Signature Pad component in Code View and selecting Signature Date.

Date Format
Date dateFormat

Enter the date format that is used to render the date and time when the form is signed, for example `dd/MM/yyyy HH:mm`. The default is `dd/MM/yyyy`.

Help Text

Popover Help Text
Help Text helpText

Popout Config

Dialog Title
Text dialogTitle

The title to display on the signature pad dialog

Cancel Button Label
Text cancelButtonLabel

The cancel button label

Cancel Button Icon
Icon cancelButtonIcon

The icon used for the cancel button.

Clear Button Label
Text clearButtonLabel

The clear button label

Clear Button Icon
Icon clearButtonIcon

The icon used for the clear button.

Save Button Label
Text saveButtonLabel

The save button label

Save Button Icon
Icon saveButtonIcon

The icon used for the save button.

Next, learn about the "Button" on the next pagecomponent.

Button

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Button** component within the **Buttons** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Button component:

Icon
Icon icon

Select the button icon

Icon Position
Option iconPos

Select the button icon position

The following options are available from the **Icon Position** dropdown:

Label	Value
Left	left
Right	right

Accessibility

Aria Label
Text ariaLabel

Enter the Aria Label for this button

Next, learn about "Click To Call" on the next page component.

Click To Call

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Click To Call component.

NOTE

This component inherits properties and shared styles from the Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Click To Call** component within the **Buttons** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Click To Call component:

Phone Number Text phoneNumber
Enter the phone number
Icon Icon icon
Select the button icon
Icon Position Option iconPos
Select the button icon position The following options are available from the **Icon Position** dropdown:

Label	Value
Left	left
Right	right

Accessibility

Aria Label
Text ariaLabel
Enter the Aria Label for this button

Next, learn about the "Dynamic Data Button" on the next page component.

Dynamic Data Button

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Dynamic Data Button component.

NOTE

This component inherits properties and shared styles from the Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Dynamic Data Button** component within the **Buttons** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Dynamic Data Button component:

Icon
Icon icon

Select the button icon

Icon Position
Option iconPos

Select the button icon position

The following options are available from the **Icon Position** dropdown:

Label	Value
Left	left
Right	right

Accessibility

Aria Label
Text ariaLabel

Enter the Aria Label for this button

Configuration

Service Name
Text serviceName

The name of the dynamic data service to call

Enforce Input Field Validation
Boolean enforceFieldValidation

Dynamic Data Button will check that input fields are valid before performing call.

Input Field Mapping
Field Ref Map inputFieldRefs

The fields to pass into the dynamic data call

Response Field Mapping
Field Ref Map responseFieldRefs

The fields to populate with response data

Enable Progress Message
Boolean enableProgressMsg

Check to enable loading message popup

Progress Message
Text progressMsg

Text to display in loading message

Next, learn about the "Jump to Issue" on the next page component.

Jump to Issue

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Jump to Issue component.

NOTE

This component inherits properties and shared styles from the Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Jump to Issue** component within the **Buttons** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Jump to Issue component:

Icon Icon icon
Select the button icon

Icon Position Option iconPos
Select the button icon position The following options are available from the **Icon Position** dropdown: **Label Value** Left left Right right

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this button

Next, learn about the "Icon Button" on the next page component.

Icon Button

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Icon Button component (widget).

NOTE

This component inherits properties and shared styles from the Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Icon Button** component within the **Buttons** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Icon Button component:

Icon Icon icon
Select the button icon

Icon Position Option iconPos
Select the button icon position The following options are available from the **Icon Position** dropdown: **Label Value** Left left Right right

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this button

Next, learn about the "Button Group" on the next page component.

Button Group

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Button Group component.

NOTE

This component inherits properties and shared styles from the Radio Button Group component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Button Group** component within the **Buttons** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Button Group component:

Options
List options

List of options

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Appearance

Arrangement
Option arrangement

Defines the layout arrangement of the radio options on the page.

The following options are available from the **Arrangement** dropdown:

Label	Value
Vertical (1 per line)	vertical
Horizontal (natural width)	horizontal
2 Column Grid	6
3 Column Grid	4
4 Column Grid	3
6 Column Grid	2

Min Width (px)
Integer minWidth

Radio Button Minimum Width in Pixels.

Stack Threshold
Option stackThreshold

Select which responsive threshold the radio options will stack on top of each other, or 'None' where stacking is not desired.

The following options are available from the **Stack Threshold** dropdown:

Label	Value
Small	sm
Medium	md
Large	lg
None	none

Selected Icon Class
Icon selectedIconClass

Selected icon class name

Unselected Icon Class Icon unselectedIconClass
Unselected icon class name

Help Text

Popover Help Text Help Text helpText

Next, learn about the "Toggle Icon Button" on the next page component.

Toggle Icon Button

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Toggle Icon Button component.

NOTE

This component inherits properties and shared styles from the Icon Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Toggle Icon Button** component within the **Buttons** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Toggle Icon Button component:

Icon When On Icon icon
Select the button icon
Icon When Off Icon offIcon
Select the button icon

Next, learn about the "Card Content" on the next page component.

Card Content

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Card Content component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Card Content** component within the **Cards** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Card Content component:

Selected Index Field Field Ref selectedIndexField
Assign a data field to store the index of the currently selected card

Minimum Cards Integer minItems
Minimum number of cards that should exist

Initial Cards Integer initialItems
Initial number of cards that should exist

Maximum Cards Integer maxItems
Maximum number of cards that should exist, or 0 for no maximum

Next, learn about the "Card Content Template" on the next page component.

Card Content Template

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Card Content Template component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Card Content Template** component within the **Cards** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Card Selector" on the next page component.

Card Selector

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Card Selector component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Card Selector** component within the **Cards** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Card Selector component:

Card Content Field Ref contentRef
Reference to linked card content item
Label Field Text labelId
ID of field within card to use as selector label
Add Card Label Text addLabel
Used on Add button, which is hidden if blank
Delete Card Label Text deleteLabel
Used on Delete button, which is hidden if blank

Show Delete On Each Card Boolean showDeleteOnEach
Show a delete button on each card item

Next, learn about the "Form Bundle Job Tasks" on the next page component.

Form Bundle Job Tasks

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Form Bundle Job Tasks component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Form Bundle Job Tasks** component within the **Collaboration** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Form Bundle Job Tasks component:

Show Job Description Boolean includeJobDescription
Enable the ability to show the job description if it is available.

Show Task Messages Boolean includeTaskMessages
Enable the ability to show the task messages.

Task Complete Image Image taskCompleteImage
Task Complete Image.

Task Not Done Image Image taskNotDoneImage
Task Not Done Image

Download Receipt Image Image taskDownloadReceiptImage
Download Receipt Image

Next, learn about the "Dropdown Bundle Navigation" on the next page component.

Dropdown Bundle Navigation

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Dropdown Bundle Navigation component.

NOTE

This component inherits properties and shared styles from the Data-Driven Dropdown component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Dropdown Bundle Navigation** component within the **Collaboration** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Dropdown Bundle Navigation component:

Data Source Field Ref dataPath
Path to source data

Label Field Text labelField
Path from data to label

Value Field Text valueField
Path from data to value

Icon
Icon icon

Select the dropdown icon

Add Blank Value
Boolean addBlankValue

Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Job Details Block" on the next page component.

Job Details Block

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Job Details Block component.

NOTE

This component inherits properties and shared styles from the Block component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Job Details Block** component within the **Collaboration** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Job Details Block component:

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this block

Hide from assistive technology Boolean ariaHidden
Hide component from assistive technology (aria-hidden)

Next, learn about the "Job Number" on the next page component.

Job Number

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Job Number component.

> NOTE
>
> This component inherits properties and shared styles from the Text Display component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Job Number** component within the **Collaboration** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Job Number component:

Accessibility

Enable Focus On Headings Boolean enableHeadingFocus
This will enable focus on heading elements (h1, h2) etc

Next, learn about the "Step Name" on the next page component.

Step Name

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Step Name component.

NOTE

This component inherits properties and shared styles from the Text Display component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Step Name** component within the **Collaboration** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Step Name component:

Accessibility

Enable Focus On Headings Boolean enableHeadingFocus
This will enable focus on heading elements (h1, h2) etc

Next, learn about the "Preview Test Block" on the next page component.

Preview Test Block

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Preview Test Block component.

NOTE

This component inherits properties and shared styles from the Block component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Preview Test Block** component within the **Collaboration** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Preview Test Block component:

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this block

Hide from assistive technology Boolean ariaHidden
Hide component from assistive technology (aria-hidden)

Next, learn about the "Route Select" on the next page component.

Route Select

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Route Select component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Route Select** component within the **Collaboration** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Route Select component:

Icon Icon icon
Select the dropdown icon

Add Blank Value Boolean addBlankValue
Check to add a blank value to the dropdown

Accessibility

Placeholder Text Text placeholder

Help Text

Popover Help Text Help Text helpText

Next, learn about the "Fieldset" on the next page component.

Fieldset

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Fieldset component, which is a container component, allowing you to group several fields together. This is especially useful when fields are not stand-alone and only make sense in the context of a group. For example, you should use the Fieldset component to group a credit card details, such as a name of a card holder, credit card number and expiry date, or multiple checkbox components, such as contact preferences.

Some components, for example, the Address, Date of Birth Dropdown, and Time, already have their fields grouped according to required functionality, so there is no need to place them into a Fieldset. If you do so, you only create some confusion about the hierarchy of the components. This also may prevent some assistive technologies[1], such as screen-readers, to correctly interpret the content of the form.

The Field is not just for aesthetic purposes, but also helps users with assistive technologies to navigate through the form.

NOTE

We recommend using this component to increase the accessibility of a form.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Fieldset** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.



[1]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.

4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Fieldset component:

Help Text

Label
Label label

Provide a meaningful description of this container identifying a group of fields. The label text (legend text) can be used by assistive technologies to inform the user of the context for fields within the Fieldset.

Popover Help Text
Help Text helpText

Provide a popover help text

Hide Label

Select Hide Label to suppress the label text, and use this component for grouping only.

Label Placement Left

Select Label Placement Left to align the label to the left. The label is aligned above the fields by default, which 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 label without having to move the cursor too far and reduces the chances of finding the wrong field by mistake.

For more information, see Maestro form accessibility.

Next, learn about the "Block" on the next page component.

Block

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Block component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Block** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Block component:

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this block

Hide from assistive technology Boolean ariaHidden
Hide component from assistive technology (aria-hidden)

Next, learn about the "Modal Page" on the next page component.

Modal Page

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Modal Page component, which represents a start or an end point where a form user must perform a task, an action, or make a decision, for example submit form data or cancel a form. A Modal Page can contain critical information or involve multiple tasks.

NOTE

This component inherits properties and shared styles from the Form component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Modal Page** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Modal Page component:

Page Icon Image favicon
The browser page / tab icon for this form

Popover Help Icon Icon popoverHelpIcon
The icon used for popover help

Auto-Send Confirmation Emails Boolean autoSendEmails
This property will auto-send an email on submit and save to the contact email address

Auto-Scroll Checkbox/Radio Button Selection
Boolean checkboxRadioAutoScroll

This property will enable auto-scroll to checkboxes/radio buttons to the middle of screen on selection

Leave Page Warning
Boolean enableLeavePageWarning

This property will prompt a confirm dialog if the user tries to leave the page.

Leave Page Warning Message
Text leavePageWarningMessage

This property will replace the default message for the leave page warning.

Mandatory Message Suffix
Text mandatoryMsgSuffix

This text will be concatenated to the end of the label when a field is deemed mandatory.

Progress Message
Text progressMsg

This message will be used when Form.submit, Form.taskUrlNavigate and Form.postData methods are called.

Accessibility

Announcement Message
Text alertMsg

This message will be announced to screen readers when the modal has loaded. Otherwise the label will be used

Dynamic Page Title
Boolean dynamicPageTitle

If checked, the title will be updated with the current progress of the application.

New Window Link Warning Message
Text linkWarningMsg

This text will be concatenated to the end of link text and read out on focus to screen readers.

Enable Mobile Zooming
Boolean enableZoom

This will enable pinch-to-zoom on mobile devices. This is not normally needed with good responsive design, but some accessibility reviews require it.

Component

Component JavaScript
Js componentJs

This property will enable the upload of a JavaScript file, which will get added to the component on publish

Google Analytics

Google Tracking ID
Text gaTrackingId

Tracks page load and navigation views using Google Analytics. For more information, see how to add Google Tag Manager to a Maestro form.

Base URL
Text gaBaseUrl

(Optional) Identifies form within site context.

Transact Manager

Session Expiry Redirect Timeout (Seconds)
Integer sessionExpiryRedirectTimeout

Perform redirect after session expiry timeout

Auto Background Save Interval (Seconds)
Integer autoSaveInterval

Performs an auto background save using the interval (In Seconds) entered, if set to blank it will never auto-save

Next, learn about the "Dialog" on page 790 component.

Dialog

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Dialog component.

NOTE

This component inherits properties and shared styles from the Block component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Dialog** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Dialog component:

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this block

Hide from assistive technology Boolean ariaHidden
Hide component from assistive technology (aria-hidden)

Next, learn about the "Block (Mandatory)" on the next page component.

Block (Mandatory)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Block (Mandatory) component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Block (Mandatory)** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Block (Mandatory) component:

Help Text

Popover Help Text
Help Text helpText

Options

Mandatory Setting
Option mandatoryOption

Select how many fields are mandatory.

The following options are available from the **Mandatory Setting** dropdown:

Label	Value
At Least One Mandatory	one
All Mandatory	all

Hide Mandatory On Labels
Boolean showChildrenIndicators

Enable mandatory asterisk on child element labels.

Next, learn about the "Section" on page 785 component.

Section (Avalon)

MaestroThe UI design product. | Form Builder | 22.04 This feature was introduced in 22.04.

Maestro comes with the Section (Avalon) component. This component is available when you use the Avalon Template. The Section (Avalon) component extends functionality of the ? component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Section (Avalon)** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.
5. Configure the component's styles via the **Styles** pane.

Properties

The following properties are available to this component:

Additional Text
Rich Text infoText

Additional Layout

Top To Bottom Layout
Boolean topToBottom

Hide Section Title
Boolean hideTitle

Collapsible
Boolean collapsibleSection

Collapsed On Init
Boolean collapsedOnInit

Collapsed Icon
Icon hideIcon

This icon is shown when the section is collapsed. You can choose an icon from the pre-defined collection of icons using the Icon Picker.

Not Collapsed Icon
Icon showIcon

Styles

The following styles are available to this component:

Next, learn about the "Sticky Panel" on page 787 component.

Section

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Section component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Section** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Section component:

Additional Text
Rich Text infoText

Additional Layout

Top To Bottom Layout
Boolean topToBottom

Hide Section Title
Boolean hideTitle

Collapsible
Boolean collapsibleSection

Collapsed On Init
Boolean collapsedOnInit

Collapsed Icon

Icon hideIcon

Not Collapsed Icon

Icon showIcon

Next, learn about the "Sticky Panel" on the next page component.

Sticky Panel

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Sticky Panel component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Sticky Panel** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Sticky Panel component:

Offset (px) Integer offset
Offset in pixels.

Confine To Parent Height Boolean confine
Confine To Parent Height.

Float over Page Boolean floatOver
If Checked Panel will float over page.

Next, learn about the "Autotab Block" on the next page component.

Autotab Block

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Autotab Block component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Autotab Block** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Autotab Block component:

Format

Format
Option format

Allowed characters

The following options are available from the **Format** dropdown:

Label	Value
All	all
Text	text
Alpha	alpha
Alphanumeric	alphanumeric
Number	number
Custom	custom

Pattern
Text pattern

Pattern regex - characters matching this regex will be disallowed. Refer to

https://github.com/Mathachew/jquery-autotab.

Next, learn about the "Open Dialog" on page 791 component.

Dialog

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Dialog component.

NOTE

This component inherits properties and shared styles from the Block component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Dialog** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Dialog component:

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this block

Hide from assistive technology Boolean ariaHidden
Hide component from assistive technology (aria-hidden)

Next, learn about the "Block (Mandatory)" on page 781 component.

Open Dialog

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Open Dialog component.

NOTE

This component inherits properties and shared styles from the Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Open Dialog** component within the **Dialogs** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Open Dialog component:

Icon Icon icon
Select the button icon

Icon Position Option iconPos
Select the button icon position The following options are available from the **Icon Position** dropdown: **Label Value** Left left Right right

Accessibility

Aria Label
Text ariaLabel

Enter the Aria Label for this button

Configuration

Dialog To Open
Field Ref dialogRef

The name of the dialog or modal page

Next, learn about the "Show XML" on page 794 component.

Dialog Navigation Bar

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Dialog Navigation Bar component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Dialog Navigation Bar** component within the **Dialogs** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Dialog Navigation Bar component:

Dialog Nav Buttons
Option dialogNavButtons

Select the buttons to include in nav bar.

The following options are available from the **Dialog Nav Buttons** dropdown:

Label	Value
None	none
Okay	okay
Okay/Cancel	okay-cancel
Open Form	open
Save Form	save
Abandon Form	abandon

Block Or Field To Validate
Text validationRef

Block Or Field To Validate.

Next, learn about the component.

Show XML

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Show XML component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Show XML** component within the **Dialogs** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Form Help" on page 798 component.

Small Dialog

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Small Dialog component.

NOTE

This component inherits properties and shared styles from the Block component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Small Dialog** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to this component:

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this block

Hide from assistive technology Boolean ariaHidden
Hide component from assistive technology (aria-hidden)

Next, learn about the "Block (Mandatory)" on page 781 component.

Dialog (Avalon)

MaestroThe UI design product. | Form Builder | 22.04 This feature was introduced in 22.04.

Maestro comes with the Dialog (Avalon) component. This component is available when you use the Avalon Template. The Dialog (Avalon) component extends functionality of the ? component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Dialog (Avalon)** component within the **Containers** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to this component:

Additional Text
Rich Text infoText

Additional Layout

Top To Bottom Layout
Boolean topToBottom

Hide Section Title
Boolean hideTitle

Collapsible
Boolean collapsibleSection

Collapsed On Init
Boolean collapsedOnInit

Collapsed Icon
Icon hideIcon

Not Collapsed Icon
Icon showIcon

Styles

The following styles are available to this component:

Next, learn about the "Sticky Panel" on page 787 component.

Form Help

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Form Help component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Form Help** component within the **Dialogs** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Save Challenge" on the next page component.

Save Challenge

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Save Challenge component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Save Challenge** component within the **Dialogs** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Save Confirm" on the next page component.

Save Confirm

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Save Confirm component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Save Confirm** component within the **Dialogs** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Cancel Confirm" on the next page component.

Cancel Confirm

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Cancel Confirm component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Cancel Confirm** component within the **Dialogs** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Form Background Image" on the next page component.

Form Background Image

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Form Background Image component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Form Background Image** component within the **Display Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Form Background Image component:

Select Image Image src
Select from the available images resources, or upload a new one

Next, learn about the "Form Header" on the next page component.

Form Header

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Form Header component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Form Header** component within the **Display Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Form Header component:

Show Help Boolean showHelp
Enable the ability to show form help area

Tracking Code

Show Tracking Code Boolean showTrackingCode
Enable the ability to show form tracking code

Tracking Code Prefix Text trackingCodePrefix
Tracking code prefix text

Next, learn about the "Image" on the next page component.

Image

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with an Image component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Image** component within the **Display Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Image component:

Select Image Image src
Select from the available images resources, or upload a new one

Alternative Text Text alternativeText
Type an alternative text for informative images, leave blank for decorative images.

Tooltip Text Text toolTip
This text will be added to the title attribute of the image.

Image Link URL Text imageLink
This url will be added as an anchor tag around the image. It will open a new tab.

Next, learn about Link component.

Link

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with an Link component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Link** component within the **Display Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Link component:

Link URL
Url linkUrl

Link Target
Text target

Next, learn about the "Percentage Indicator" on the next page component.

Percentage Indicator

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Percentage Indicator component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Percentage Indicator** component within the **Display Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Text Display" on the next page component.

Text Display

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Text Display component (widget) that allows you to display rich textual context including icons to users.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Text Display** component within the **Display Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Text Display component:

Accessibility

Enable Focus On Headings Boolean enableHeadingFocus
This will enable focus on heading elements (h1, h2) etc

Next, learn about the "Form Footer" on the next page component.

Form Footer

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Form Footer component.

NOTE

This component inherits properties and shared styles from the Block component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Form Footer** component within the **Display Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Form Footer component:

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this block

Hide from assistive technology Boolean ariaHidden
Hide component from assistive technology (aria-hidden)

Next, learn about the "LinkedIn Auto-Fill" on page 810 component.

Form Footer (Avalon)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Form Footer (Avalon) component. This component is available when you use the Avalon Template. The Form Footer (Avalon) component extends functionality of the ? component.

NOTE

This component inherits properties and shared styles from the Block component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Form Footer (Avalon)** component within the **Display Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to this component:

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this block

Hide from assistive technology Boolean ariaHidden
Hide component from assistive technology (aria-hidden)

Styles

The following styles are available to this component:

Next, learn about the "LinkedIn Auto-Fill" on the next page component.

LinkedIn Auto-Fill

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the LinkedIn Auto-Fill component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **LinkedIn Auto-Fill** component within the **Identity** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the LinkedIn Auto-Fill component:

Api Config

Client ID
Text apiKey

Auto Authorize
Boolean authorize

Help Text

Popover Help Text
Help Text helpText

Options

Description
Rich Text description

Completed Message
Rich Text completedMessage

Opt Out on Init
Boolean optOutInitFlag

Opt Out Button Caption
Rich Text optOutButton

Opt Out Button Caption
Rich Text prefillButton

Opt In Button Caption
Rich Text optInButton

Opt In Image
Image optInImage

Select from the available images resources, or upload a new one

Prefill Mappings

Prefill Mappings
Field Ref Map prefillMappings

Next, learn about the "Checkbox" on the next page component.

Checkbox

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Checkbox component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Checkbox** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Checkbox component:

Unselected Icon Icon unselectedIconClass
Icon for unselected checkbox

Selected Icon Icon selectedIconClass
Icon for selected checkbox

Help Text

Popover Help Text Help Text helpText

Next, learn about the "Currency Field" on the next page component.

Currency Field

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Currency Field component (widget), which you can use in several scenarios.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Currency Field** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Currency Field component:

Currency Icon Icon currencyIcon
Select the currency icon

Max Leading Digits Integer maxLength

Decimal Places Integer places
Number of decimal places to display

Accessibility

Placeholder Text Text placeholder

Help Text

Popover Help Text
Help Text helpText

Options

Prevent Negative
Boolean preventNegative

Decimal Point
Text decimalPointChar

Thousands Separator
Text thousandsSeparator

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Next, learn about the Decimal Field component.

Data-Driven Dropdown

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Data-Driven Dropdown component that allows you to build dynamic selection lists to implement various customer requirements.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Data-Driven Dropdown** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Data-Driven Dropdown component:

Data Source Field Ref dataPath
Path to source data, which is a key to the data to be shown in the dropdown lists.
Label Field Text labelField
Path from data to label
Value Field Text valueField
Path from data to value
Icon Icon icon
Select the dropdown icon

Add Blank Value Boolean addBlankValue
Check to add a blank value to the dropdown

Accessibility

Placeholder Text Text placeholder

Help Text

Popover Help Text Help Text helpText

You can also update a Data-Driven Dropdown triggered by an input Text Field. In this cae, you have to use the input from a text field to both trigger and to make a dynamic data services call. This DDS call will retrieve the data to populate a data-driven dropdown. We will use the example of a zip code text field where the user types the zip code and a zip code lookup service returns the list of cities in a zip code and populates them in a DDD.

To do so, you need to:

1. Add a Text Field to your form. This will be the zip code field. Here we presume that the field reference ID for the zip-code field is "zipcode".
2. Add a Data Field to your form. This will store the data returned by the DDS call. We presume that the field reference ID for the Data field is cityDataField.
3. Add a Data-Driven Dropdown to your form. Make the data source for the DDD point to "data.cityDataField".
4. Add an "On Change" script to the Text Field in order to map the request and response parameters for your DDS call and to make the call. It should look something like the JavaScript below.

Example Javascript to map request and response fields, to make DDS request and to update the Data Field is shown below:

```
// The key "ZIPCODE" is presumed to be the input parameter for the "Zipcode Lookup Ser-
vice"
var inputFieldRefs = [{
    key: "ZIPCODE",
    ref: "data.zipcode"
}];

// The key "cities" is presumed to be the output parameter returned from the "Zipcode
Lookup Service"
var outputFieldRefs = [{
    key: "cities",
    ref: "data.cityDataField"
}];

var serviceName = "Zipcode Lookup Service";
var dataParams = Form.convertToFieldDataMap(inputFieldRefs, data);

// Character input must be length of valid zipcode before making request
if(data.zipcode && data.zipcode.length === 5) {
  proceed();
}

// Lookup promise
function proceed() {
    // Make asynchronous call to DDS and when the response is received set the field data
or log an error
    DynamicData.call(serviceName, dataParams).then(function(response) {
        // Update the form field data.cityDataField with the response data formatted for
a Data-Drive Dropdown.
        Form.setFieldDataFromResponse(outputFieldRefs, response, data);
    }).catch(function(err) {
        console.log("Error: ", err);
    });
}
```

Make sure you have a service written to return the data back as JSON in the correct format. The service in our example, "Zipcode Lookup Service", is presumed to return a JSON response format that looks like:

```
{"cities":[{"label":"city1","value":"city1"},{"label":"city2","value":"city2"}]}
```

Next, learn about the "Dropdown" on page 821 component.

Decimal Field

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Decimal Field component (widget), which you can use in several scenarios.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Decimal Field** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Decimal Field component:

Max Leading Digits
Integer maxLength

Decimal Places
Integer places

Number of decimal places to display

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Options

Prevent Negative
Boolean preventNegative

Decimal Point
Text decimalPointChar

Thousands Separator
Text thousandsSeparator

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Next, learn about the Percentage Field component.

Dropdown

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Dropdown component (widget), which allows form users to make a selection from a list of available options.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Dropdown** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Dropdown component:

Options
List options

The options list for the dropdown

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Sort Options
Boolean sortOptions

Check to automatically sort options by label

Icon
Icon icon

Select the dropdown icon

Add Blank Value
Boolean addBlankValue

Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Graphical Radio Button Group" on page 852 component.

Graphical Radio Button

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Graphical Radio Button component (widget), which allows you to specify not only a text caption, but also an image from a predefined list of images or your own rich content. The Graphical Radio Button component replaces the usual radio button component. If you want to use a composite image, you can create a shared style for your Graphical Radio Button components and adding appropriate style rules to create and show this. You will need to factor in all the states so that the image is shown correctly when selected, not selected, hover, focus, active, disabled, error and so on. You can use different images for the selected and non-selected states, but we recommend using a single image.

NOTE

Graphical Radio Buttons are restricted to being used inside Graphical Radio Button Groups.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Graphical Radio Button** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Graphical Radio Button component:

Button Value Text value
Enter the value for this radio button

Appearance

Selected Icon Class Icon selectedIconClass

Selected icon class name

Unselected Icon Class
Icon unselectedIconClass

Unselected icon class name

Image

Button Image
Image src

Select from the available images resources, or upload a new one

Button Selected Image
Image srcSelected

Select from the available images resources, or upload a new one

Alternative Text
Text alternativeText

Type an alternative text for informative images, leave blank for decorative images.

Next, learn about the Graphical Radio Button Group component.

Radio Button Group

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Radio Button Group component (widget) so you can group Radio Buttons to present several mutually exclusive options on a form.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Radio Button Group** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Radio Button Group component:

Options
List options

List of options

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Appearance

Arrangement
Option arrangement

Defines the layout arrangement of the radio options on the page.

The following options are available from the **Arrangement** dropdown:

Label	Value
Vertical (1 per line)	vertical
Horizontal (natural width)	horizontal
2 Column Grid	6
3 Column Grid	4
4 Column Grid	3
6 Column Grid	2

Min Width (px)
Integer minWidth

Radio Button Minimum Width in Pixels.

Stack Threshold
Option stackThreshold

Select which responsive threshold the radio options will stack on top of each other, or 'None' where stacking is not desired.

The following options are available from the **Stack Threshold** dropdown:

Label	Value
Small	sm
Medium	md
Large	lg
None	none

Selected Icon Class
Icon selectedIconClass

Selected icon class name

Unselected Icon Class
Icon unselectedIconClass

Unselected icon class name

Help Text

Popover Help Text Help Text helpText

Next, learn about the "Radio Button [Repeating]" on the next page component.

Radio Button [Repeating]

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Radio Button [Repeating] component (widget).

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Radio Button [Repeating]** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Radio Button [Repeating] component:

Selected Icon Class Icon selectedIconClass
Selected icon class name

Unselected Icon Class Icon unselectedIconClass
Unselected icon class name

Next, learn about the "Text Field" on the next page component.

Text Field

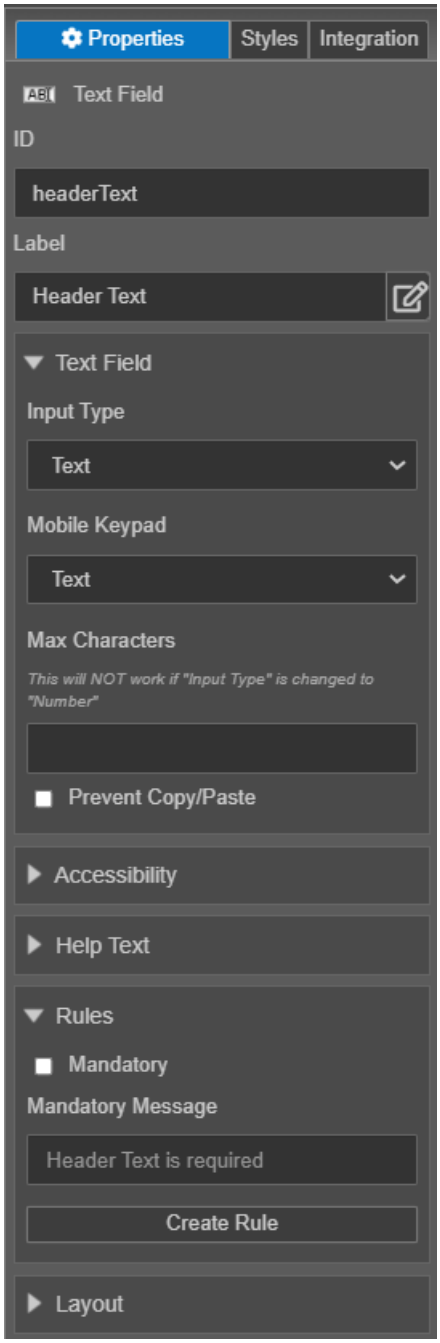
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Text Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Text Field** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.

4. Configure the component's properties via the **Properties** pane.



Properties

The following properties are available to the Text Field component:

Input Type Option inputType
Select the input type

The following options are available from the **Input Type** dropdown:

Label	Value
Text	text
Number	number
Email	email
Password	password
Telephone	tel

Max Characters
Integer maxLength

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Time" on page 834 component.

Percentage Field

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Percentage Field component (widget), which you can use in several scenarios.

NOTE

This component inherits properties and shared styles from the Decimal Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Percentage Field** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Percentage Field component:

Max Leading Digits Integer maxLength

Decimal Places Integer places
Number of decimal places to display

Accessibility

Placeholder Text Text placeholder

Help Text

Popover Help Text
Help Text helpText

Options

Decimal Places
Integer places

Number of decimal places to display

Prevent Negative
Boolean preventNegative

Decimal Point
Text decimalPointChar

Thousands Separator
Text thousandsSeparator

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Next, learn about the "Data-Driven Dropdown" on page 815 component.

Time

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Time component.

NOTE

This component inherits properties and shared styles from the Block (Mandatory) component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Time** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Time component:

Help Text

Popover Help Text
Help Text helpText

Options

Mandatory Setting
Option mandatoryOption

Select how many fields are mandatory.

The following options are available from the **Mandatory Setting** dropdown:

Label	Value
At Least One Mandatory	one

Label	Value
All Mandatory	all

Hide Mandatory On Labels
Boolean showChildrenIndicators
Enable mandatory asterisk on child element labels.

Next, learn about the "Data-Driven Checkbox Group" on the next page component.

Data-Driven Checkbox Group

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Data-Driven Checkbox Group component so you can group Checkbox components to present several options on a form.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Data-Driven Checkbox Group** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Data-Driven Checkbox Group component:

Data Source
Field Ref dataPath

Path to source data

Label Field
Text labelField

Path from data to label

Appearance

Vertical Checkbox Group
Boolean verticalGroup

Select to make a vertical checkbox group.

Checkbox Min Width (px)
Integer minWidth

Checkbox Button Minimum Width in Pixels.

Unselected Icon
Icon unselectedIconClass

Icon for unselected checkbox

Selected Icon
Icon selectedIconClass

Icon for selected checkbox

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Data-Driven Radio Group" on the next page component.

Data-Driven Radio Group

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Data-Driven Radio Group component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Data-Driven Radio Group** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Data-Driven Radio Group component:

Data Source Field Ref dataPath
Path to source data

Label Field Text labelField
Path from data to label

Value Field Text valueField
Path from data to value

Selected Icon Class Icon selectedIconClass
Selected icon class name.

Unselected Icon Class
Icon unselectedIconClass

Unselected icon class name.

Appearance

Arrangement
Option arrangement

Defines the layout arrangement of the radio options on the page.

The following options are available from the **Arrangement** dropdown:

Label	Value
Vertical (1 per line)	vertical
Horizontal (natural width)	horizontal
2 Column Grid	6
3 Column Grid	4
4 Column Grid	3
6 Column Grid	2

Min Width (px)
Integer minWidth

Radio Button Minimum Width in Pixels.

Stack Threshold
Option stackThreshold

Select which responsive threshold the radio options will stack on top of each other, or 'None' where stacking is not desired.

The following options are available from the **Stack Threshold** dropdown:

Label	Value
Small	sm
Medium	md
Large	lg
None	none

Help Text

Popover Help Text	
Help Text	helpText

Next, learn about the "Date Picker" on the next page component.

Date Picker

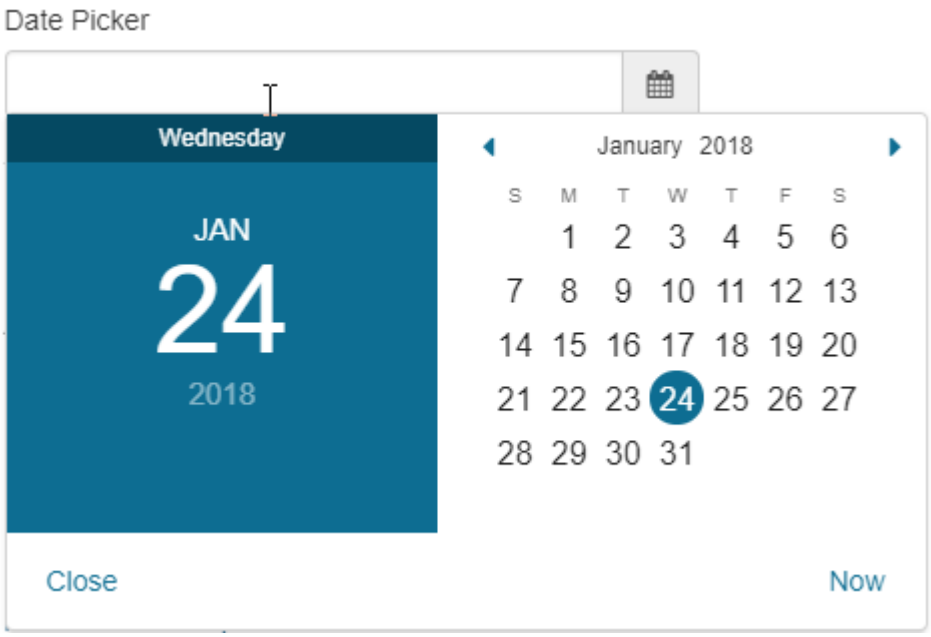
MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Maestro comes with the Date Picker component (widget) that allows form users to select a date and time on a form.

This component supports both left-to-right (LTR) and right-to-left (RTL) languages.



The rendered date picker component is shown below.



Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Date Picker** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Date Picker component:

Date/Time Data Format
Text dateFormat

Enter the date and time format that is stored in the underlying XML

Date/Time Display Format
Text dateDisplayFormat

Enter the date display format

Display Mode
Option displayMode

Select the display mode

The following options are available from the **Display Mode** dropdown:

Label	Value
Date Only	date
Full Mode (Date and Time)	full

Prefill with Today's Date
Boolean prefillTodaysDate

Check to prefill with today's date

Prefer Non-US Input
Boolean auDateInput

Check to prefer using non-US date format.

NOTE

The US date format is m/d/yy or m/d/yyyy.

Open Calendar Icon
Icon inputIcon

The icon used for the open calendar button. You should check with the Template Designer if you can change this icon, as icons are typically customized in the template.

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Terminology

Close Button Label
Text closeBtnLabel

Now Button Label
Text nowBtnLabel

Month Labels
Text monthLabels

A comma separated list of month labels from January to December

Month Abbreviations
Text monthShortLabels

A comma separated list of month abbreviations from January to December

Day Labels
Text dayLabels

A comma separated list of day labels from Sunday to Saturday

Validation

Min Date (dd/mm/yyyy)
Date minDate

Minimum Date allowed. For example, a valid date may be between 01/01/2017 (min) to 31/12/2017 (max).

Max Date (dd/mm/yyyy)
Date maxDate

Minimum Date allowed.

Must Be A Past Date
Boolean pastDate

Check if the date must be in the past.

Must Be A Future Date
Boolean futureDate

Check if the date must be in the future.

Include Today
Boolean includeToday

Check to include today in validation.

Validation Messages

Past Date
Text pastDateValidationMsg

Enter the past date validation message, which is displayed if the Must Be A Past Date option is selected. The user will receive the message if they do not follow the validation rules. The Form Builder can customize the messages displayed when validation is not met.

Past Date (Inc Today)
Text pastDateIncTodayValidationMsg

Enter the past date (Inc Today) validation message, which is displayed if the Must Be A Past Date and Include Today options are selected. The user will receive the message if they do not follow the validation rules.

Future Date
Text futureDateValidationMsg

Enter the future date validation message, which is displayed if the Must Be A Future Date option is selected. The user will receive the message if they do not follow the validation rules.

Future Date (Inc Today)
Text futureDateIncTodayValidationMsg

Enter the future date (Inc Today) validation message, which is displayed if the Must Be A Future Date and Include Today options are selected. The user will receive the message if they do not follow the validation rules.

Next, learn about the Autocomplete Field component.

Autocomplete Field

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Autocomplete Field component, which allows you to create fields that auto-complete user's input text as they are entering it, for example, a client's address.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Autocomplete Field** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Autocomplete Field component:

Accessibility

Placeholder Text
Text placeholder

Include Button Next To Label
Boolean includeButton

Descriptive Text
Rich Text descriptiveText

This text will be used in an aria described by on the input

No Results Message
Text noResultsFoundMsg

This text will be alerted to the user if 0 results are found

No Results Message Part 2
Text noResultsFoundMsgIfExtra

This text will be concatenated to the No Results Message if Extra Item Message is NOT blank

One Result Found Message
Text oneResultFoundMsg

This text will be alerted to the user if 1 result is found

More Than One Result Found Message
Text resultsFoundMsg

This text will be alerted to the user if more than 1 result is found

Prefix Text
Rich Text prefixText

This text will be placed on the left of the button

Button Label
Rich Text buttonLabel

Configuration

Clear Field When Not Matched
Boolean clearWhenNotMatched

Min Characters To Show Suggestion
Integer minCharTrigger

Max Items In Suggestion List
Integer itemLimit

Continue Typing Message
Text continueMessage

Extra Item Message
Text extraItemMessage

Data Source

Data Source
Option dataSource

Select the data source

The following options are available from the **Data Source** dropdown:

Label	Value
TM Dynamic Data Service	service
Reference Data (Prepop)	refData

Use Transact Functions
Boolean useTransactFunctions

Uses the Transact function call instead of legacy dynamic data calls

Reference Data Name
Field Ref dataPath

Path to source data

Trigger Property Name
Text triggerName

Trigger Field name in JSON Result

Suggestion List Names
Text suggestionListNames

Names that will show on each suggestion list item

Dds Configuration

First Service Name
Text DynamicDataServiceName

First Service Version
Text serviceVersion

Response Data Key
Text responseKey

Delay Of Triggering First DDS Call
Integer delayDDSCall

First Input Field Map
Field Ref Map inputFieldRefs

The fields to pass into the dynamic data call

First Response Field Map
Field Ref Map dataResultMappings

Paths to set result data to

Second Service Version
Text secondServiceVersion

Second Response Data Key
Text secondResponseKey

Help Text

Popover Help Text
Help Text helpText

Second Dds Configuration

Second Service Name
Text secondDynamicDataServiceName

Second Input Field Mapping
Field Ref Map secondInputFieldRefs

The fields to pass into the dynamic data call

Second Response Field Mappings
Field Ref Map secondDataResultMappings

Paths to set result data to

Enable Progress Message
Boolean enableProgressMsg

Check to enable loading message popup

Progress Message Text progressMsg
Text to display in loading message

Next, learn about the "Slider" on page 853 component.

Graphical Radio Button Group

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Graphical Radio Button Group component (widget) so you can group Graphical Radio Buttons to present several mutually exclusive options on a form.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Graphical Radio Button Group** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Graphical Radio Button Group component:

Help Text

Popover Help Text Help Text helpText

Next, learn about the "Radio Button Group" on page 825 component.

Slider

MaestroThe UI design product. | Form Builder | 21.05 This feature was updated in 21.05.

Maestro comes with the Slider component (widget) that allows users to view and select a value or range of values from the range along a bar.

This component supports both left-to-right (LTR) and right-to-left (RTL) languages. For LTR languages, the smallest value appears on the far left end of the track and the largest value is on the far right. For RTL languages this orientation is reversed.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Slider** component within the **Input Fields** folder.



3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Slider component:

Configuration

Min Value Integer minValue
The minimum value of the slider.

Max Value
Integer maxValue

The maximum value of the slider.

Step Value
Integer step

The interval of the slider's value increment.

Show Ticks
Boolean showTicks

Shows tick marks at predetermined values, without their actual values, that the user can move the slider to. The predetermined values are calculated usingStep Value.

Show Ticks With Values
Boolean showTicksValues

Shows tick marks with their predetermined values that the user can move the slider to. The predetermined values are calculated using Step Value.

Value Prefix
Text valuePrefix

Adds this text to the value label as a prefix.

NOTE

This value can be translated to different languages including right-to-left languages. | 21.05 This feature was introduced in 21.05.

Value Suffix
Text valueSuffix

Adds this text to the value label as a suffix.

NOTE

This value can be translated to different languages including right-to-left languages. | 21.05 This feature was introduced in 21.05.

Add Space Between
Boolean addSpaceBetween

Adds a space between the prefix and the value label, and between the suffix and the value label.

Range Slider

Enable Range Slider
Boolean isRange

Shows the second thumb to allow the user to view and specify the range of values.

Min Range
Integer minRange

The minimum value of the range slider.

Max Range
Integer maxRange

The maximum value of the range slider.

Help Text

Popover Help Text
Help Text helpText

Shows a popover help text to provide some contextual help to the user. You can use the Maestro rich text editor to create the help text using various formatting and styles.

Next, learn about the "Multiline Text Area" on the next page component.

Multiline Text Area

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Multiline Text Area component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Multiline Text Area** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Multiline Text Area component:

Resize Mode Option resizeMode
Select the resize mode The following options are available from the **Resize Mode** dropdown: **Label** **Value** Vertical vertical Fixed none

Min/Initial Number of Rows Integer minRows
Number of Rows

Max Number of Rows Integer maxRows
Number of Rows

Disable New Line Entry
Boolean disableNewLine

Disable New Line Entry

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Options

Max Characters
Integer maxLength

Prevent Copy/Paste
Boolean preventCopyPaste

Enabling this property will prevent copy and paste

Next, learn about the "Dropdown Multi Select" on the next page component.

Dropdown Multi Select

MaestroThe UI design product. | Form Builder | 22.04 This feature was updated in 22.04.

Maestro comes with the Dropdown Multi Select component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Dropdown Multi Select** component within the **Input Fields** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Dropdown Multi Select component:

Options
List options

The options list for the dropdown

By default, the **Options** property contains the following values:

Label	Value
One	One
Two	Two
Three	Three

Sort Options
Boolean sortOptions

Check to automatically sort options by label

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Next, learn about the "Horizontal Divide" on the next page component.

Horizontal Divide

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Horizontal Divide component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Horizontal Divide** component within the **Layout** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Cancel Success" on the next page component.

Cancel Success

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Cancel Success component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Cancel Success** component within the **Modal Pages** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Submit Success" on the next page component.

Submit Success

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Submit Success modal page component, which you can use to display a successful response from a form submission to Manager.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Submit Success** component within the **Modal Pages** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

NOTE

Check the Submit component.

Next, learn about the "Save Success" on the next page component.

Save Success

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Save Success component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Save Success** component within the **Modal Pages** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Modal Mobile Header" on the next page component.

Modal Mobile Header

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Modal Mobile Header component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Modal Mobile Header** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Navigator Step Count" on the next page component.

Navigator Step Count

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with an Navigator Step Count component, which displays a current step (page) out of the total number of steps (pages) that a form consists of. You can use this components along with the Navigator component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Navigator Step Count** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the Navigator Wizard Bar component.

Navigation Wizard Bar

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Navigation Wizard Bar component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Navigation Wizard Bar** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Navigation Wizard Bar component:

Options

Next Button Label Text nextButtonLabel
Enter the next Button label.

Submit Button Label Text submitButtonLabel
Enter the submit button label.

Back Button Label Text backButtonLabel
Enter the back button label.

Next, learn about the Navigator component.

Navigator

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with an Navigator component that allows a form user to move between pages of a form.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Navigator** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Navigator component:

Show Left Hand Navigation
Boolean leftMenu

Select this option to move navigation from the top to the left of the page

Show Mobile Menu Button
Boolean includeMenuButton

Select this option to show the mobile menu

Accessibility

Aria Label
Text navAriaLabel

Enter the Aria Label for this navigation element

Tab Focus
Option tabFocus

If set to reachable, the tab index is set to 0, else it will be set to -1

The following options are available from the **Tab Focus** dropdown:

Label	Value
Reachable	0
Not reachable	-1

Disable All Non-Active Navigator Items
Boolean disableOnNonActive

If checked, all non-active navigator items will have aria-disabled set to true

Navigator type (role)
Text navItemGroupRole

The value entered will appear within the role attribute for navigation item group/container

Navigator item type (role)
Text navItemRole

The value entered will appear within the role attribute for each navigation item

Left Hand Navigation

Show Second Level
Boolean includeSecondLevel

Select this option to move navigation from the top to the left of the page

Next, learn about the "Second Level Navigator" on the next page component.

Second Level Navigator

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Second Level Navigator component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Second Level Navigator** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Second Level Navigator component:

Display on Side Boolean leftMenu
Select this option to move navigation to top to bottom

Equal Width Boolean equalWidth
Select this option ensure equal item widths

Accessibility

Aria Label Text navAriaLabel
Enter the Aria Label for this navigation element

Menu Item Group Role Text navItemGroupRole
The value entered will appear within the role attribute for navigation item group/container

Menu Item Role Text navItemRole
The value entered will appear within the role attribute for each navigation item

Next, learn about the Tramline Navigator component.

Tramline Navigator

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Tramline Navigator component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Tramline Navigator** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Tramline Navigator component:

Tramline Images

Active Image
Image activeImage

Select an image for the active page.

Inactive Image
Image inactiveImage

Select an image for the inactive page.

Visited Image
Image visitedImage

Select an image for the visited pages.

Next, learn about the "Page Group" on page 874 component.

Navigator (Avalon)

MaestroThe UI design product. | Form Builder | 22.04 This feature was introduced in 22.04.

Maestro comes with an Navigator (Avalon) component that allows a form user to move between pages of a form.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Navigator (Avalon)** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to this component:

Show Left Hand Navigation
Boolean leftMenu

Select this option to move navigation from the top to the left of the page

Show Mobile Menu Button
Boolean includeMenuButton

Select this option to show the mobile menu

Accessibility

Aria Label
Text navAriaLabel

Enter the Aria Label for this navigation element

Tab Focus
Option tabFocus

If set to reachable, the tab index is set to 0, else it will be set to -1

The following options are available from the **Tab Focus** dropdown:

Label	Value
Reachable	0
Not reachable	-1

Disable All Non-Active Navigator Items
Boolean disableOnNonActive

If checked, all non-active navigator items will have aria-disabled set to true

Navigator type (role)
Text navItemGroupRole

The value entered will appear within the role attribute for navigation item group/container

Navigator item type (role)
Text navItemRole

The value entered will appear within the role attribute for each navigation item

Left Hand Navigation

Show Second Level
Boolean includeSecondLevel

Select this option to move navigation from the top to the left of the page

Styles

The following styles are available to this component:

Next, learn about the "Second Level Navigator" on page 869 component.

Page Group

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Page Group component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Page Group** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Page Group component:

Display Text

Menu Text Rich Text menuText
This Text will be used in the Navigator Menu instead of Page label

Next, learn about the "Page Controller" on page 877 component.

Page Group (Avalon)

MaestroThe UI design product. | Form Builder | 22.04 This feature was introduced in 22.04.

Maestro comes with the Page Group (Avalon) component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Page Group (Avalon)** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to this component:

Display Text

Menu Text Rich Text menuText
This Text will be used in the Navigator Menu instead of Page label

Styles

The following styles are available to this component:

Next, learn about the "Page Controller" on page 877 component.

Navigator Wizard Bar

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with an Navigator Wizard Bar component that allows a form user to move between pages of a form. This is an alternative to the Navigator component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Navigator Wizard Bar** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Navigator Wizard Bar component:

Options

Next Button Label Text nextButtonLabel
Enter the Next Button Label for this navigation element
Submit Button Label Text submitButtonLabel
Enter the Submit Button Label for this navigation element
Back Button Label Text backButtonLabel
Enter the Back Button Label for this navigation element

Next, learn about the Navigator component.

Page Controller

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Page Controller component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Page Controller** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Page Controller component:

Background Save On Page Change
Boolean saveOnPageChange

Form will save on page change

Validation Mode
Option validationMode

Select the page validation mode

The following options are available from the **Validation Mode** dropdown:

Label	Value
Unconstrained, validate on submit	unconstrained
Sequential, validate on page change	sequential

Page Transition Delay
Integer transitionDelay

Provide how long the duration of the page transition will be

Restart At Last Saved Page
Boolean restartAtLastSavedPage

If checked, the form will return to the page it was on after save and resume.

Accessibility

Focus After Page Change Mode
Option focusMode

Smart focus sets focus to navigator if present. If not, find first heading.

The following options are available from the **Focus After Page Change Mode** dropdown:

Label	Value
Smart Focus	smart
Do Nothing	none

Next, learn about the "Page" on page 881 component.

Page Controller (Avalon)

MaestroThe UI design product. | Form Builder | 22.04 This feature was introduced in 22.04.

Maestro comes with the Page Controller (Avalon)component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Page Controller (Avalon)** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to this component:

Background Save On Page Change
Boolean saveOnPageChange

Form will save on page change

Validation Mode
Option validationMode

Select the page validation mode

The following options are available from the **Validation Mode** dropdown:

Label	Value
Unconstrained, validate on submit	unconstrained
Sequential, validate on page change	sequential

Page Transition Delay
Integer transitionDelay

Provide how long the duration of the page transition will be

Restart At Last Saved Page
Boolean restartAtLastSavedPage

If checked, the form will return to the page it was on after save and resume.

Accessibility

Focus After Page Change Mode
Option focusMode

Smart focus sets focus to navigator if present. If not, find first heading.

The following options are available from the **Focus After Page Change Mode** dropdown:

Label	Value
Smart Focus	smart
Do Nothing	none

Styles

The following styles are available to this component:

Next, learn about the "Page" on the next page component.

Page

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Page component that allows you to group components according to a form logic so they make up a user journey flow.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Page** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Page component:

Display Text

Menu Text Rich Text menuText
This Text is used in the Navigator Menu instead of Page label

Options

Hide Navigator Boolean offMenu
A page that is in the flow of the form but not displayed in the menu

Next, learn about the "Save and Close" on page 887 component.

Page (Avalon)

MaestroThe UI design product. + Avalon TemplateThis topic is related to the Avalon Template used in Maestro. | Form Builder | 22.10 This feature was introduced in 22.10.

Maestro comes with the Page (Avalon) component that allows you to group components according to a form's navigation logic to build a user journey flow. This component is available when you use the Avalon Template. The Page (Avalon) component extends functionality of the Page component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Page (Avalon)** component within the **Navigation** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.
5. Configure the component's styles via the **Styles** pane.

Properties

The following properties are available to this component:

Display Text

Title Text
Rich Text titleText

This text is used as a title. If the text is not provided, the default `{{T[item.id].label}}` value is used.

Subtitle Text
Rich Text subtitleText

This text is used as a subtitle. If the text is not provided, the default `<h1 ng-bind-html="T[Form.$Pages.currentPage.id].properties.formName">Form Name</h1>` value is used.

NOTE

You can use HTML tags in the Subtitle Text component, for example, `<samp>Welcome</samp> to <em>my</em> page!`. For more information, see rich text editor.

Additional Text
Rich Text additionalText

This text is used as an additional text

Help Text
Rich Text helpText

This text is used as a help text

Options

Hide Navigator
Boolean offMenu

A page that is in the flow of the form but not displayed in the menu

Page View

Hide From Navigator
Boolean hideFromNavigator

A page that is in the flow of the form but not displayed in the Navigator

Hide From Header
Boolean hideFromHeader

Hide or show the form header (if applicable for the template)

Hide From Footer
Boolean hideFromFooter

Hide or show the form footer (if applicable for the template)

Hide Navigation Buttons
Boolean hideNavigationButtons

Hide or show the navigational buttons

Hide Menu Button
Boolean hideFromHeader

Hide or show the form menu button

Page of No Return
Boolean pageNoReturn

The user will not be able to navigate back past this page

Navigator

Navigator Text
Rich Text

This text will be shown in the Navigator in place of the page label

Styles

The following styles are available to this component:

Avalon Page

```
form[name='form'] {
  .avalon-page {
    width: ~"calc(100% + 10px)";
    margin: 0;
    padding: 0;
    border: 0;
    border-radius: 0;
    box-shadow: none;
    .wdg-section-1-container {
      margin: 0;
      padding: 0;
      border: 0;
      border-radius: 0;
      box-shadow: none;
    }
    .wdg-section-1-outer-wrapper {}
    .wdg-section-1-wrapper {
      padding: 0;
    }
    .wdg-section-1-header {}
    .wdg-section-1-icon {
      display: none;
    }
    .wdg-section-1-title {
      margin: 0;
      padding: 0 0 20px 0;
      outline: none;
    }
    .wdg-section-1-form-name {
      display: none;
    }
    .wdg-section-1-additional-text {
      display: none;
    }
    .wdg-section-1-body {
      padding: 0;
    }
  }

  &.av-receipt {
    .avalon-page {
```

```
      width: 100%;
    }
  }
  // Fullscreen page overrides
  .av-fullscreen {
    .wdg-section-1-wrapper {
      max-width: @ux-max-form-width;
      margin-left: auto;
      margin-right: auto;
    }
    .wdg-section-1-header {
      display: none;
    }
    .wdg-section-1-body {
      padding: 0;
    }
  }
}
```

Next, learn about the "Save and Close" on the next page component.

Save and Close

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Save and Close component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Save and Close** component within the **Navigation Functions** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Repeat Index Text" on the next page component.

Repeat Index Text

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Repeat Index Text component (widget).

NOTE

This component inherits properties and shared styles from the Text Display component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Repeat Index Text** component within the **Repeats** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Repeat Index Text component:

Accessibility

Enable Focus On Headings Boolean enableHeadingFocus
This will enable focus on heading elements (h1, h2) etc

Next, learn about the "Repeat Add Button" on the next page component.

Repeat Add Button

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Repeat Add Button component (widget), which allows a form user to add the whole Repeat component multiple times to a form. This is especially handy when adding another name or address field to a form.

NOTE

This component inherits properties and shared styles from the Icon Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Repeat Add Button** component within the **Repeats** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Repeat Add Button component:

Repeat Link Field Ref repeatLink
Link to the repeat

Next, learn about the Repeat Delete Button component

Repeat Delete Button

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Repeat Delete Button component (widget), which allows a form user to remove the whole repeat component from a form. This is especially handy when deleting another name or address field from a form. If you need to add a repeat component, use the Repeat Add Button.

NOTE

This component inherits properties and shared styles from the Icon Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Repeat Delete Button** component within the **Repeats** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

See properties of the Icon Button component.

Next, learn about the "Repeat" on the next page component.

Repeat

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Repeat component (widget), which is a container for other components, such as text fields and labels, that allows a form user to add the whole repeat component multiple times to a form, by clicking a Repeat Add Button. This is especially handy when adding another customer's name or address field to a form.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Repeat** component within the **Repeats** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Repeat component:

Repeat Item Label Text itemLabel
Can be used on multiple references, such as add and delete buttons
Initial Items In Repeat Integer initialItems
Enter the initial number of items in repeat
Min Items In Repeat Integer minItems
Enter the minimum number of items allowed in repeat
Max Items In Repeat Integer maxItems

Enter the maximum number of items allowed in repeat. Set to 0 for unrestricted

Transfer Focus To New Instance
Boolean transferFocus

Transfers the focus to the first field in new instance

Transfer Focus After Delete
Boolean transferFocusAfterDelete

Transfers the focus to the previous instance

Linked Delete Button To Focus
Field Ref linkedDeleteButton

Link the delete button which will get focused to when deleted

Linked Add Button To Focus
Field Ref linkedAddButton

Link the add button which will get focused to when deleted if no instances remain

Button Options

Add Button Label
Rich Text addItemLabel

Used on Add button

Show Add Button
Boolean showAddButton

Check to show the add button.

Add Item Icon
Icon addButtonIcon

Used on Add button

Hide Add Button when Max Reached
Boolean hideAddButtonOnMax

Check to hide the add item button when item max is reached

Hide Delete Button when Min Reached
Boolean hideDeleteButtonOnMin

Check to hide the delete item button when item min is reached

Next, learn about the "Repeating Block Template" on the next page component.

Repeating Block Template

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Repeat Block Template component (widget), which allows you to build a small part of form that can be repeated multiple times on a form by a use clicking a Repeat Add Button. This is especially handy when adding another name or address field to a form.

NOTE

This component inherits properties and shared styles from the Block component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Repeating Block Template** component within the **Repeats** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Repeating Block Template component:

Accessibility

Aria Label Text ariaLabel
Enter the Aria Label for this block

Hide from assistive technology Boolean ariaHidden
Hide component from assistive technology (aria-hidden)

To learn more about the best practices for using this component, read here.

Next, learn about the "Repeating Block Template (Deprecated)" on the next page component.

Repeating Block Template (Deprecated)

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro provides the Repeat Block Template (Deprecated) component (widget).

This component is deprecated so you should use the Repeating Block Template component instead.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Repeating Block Template (Deprecated)** component within the **Deprecated** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Data Field" on the next page component.

Data Field

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Data Field component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Data Field** component within the **Tools** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Web Font" on page 903 component.

Bundle Resource Library

MaestroThe UI design product. | Form Builder | 21.11 This feature was updated in 21.11.

Maestro comes with the Bundle Resource Library component, which allows you to add library's resources, such as images, JavaScripts[1], and CSS[2], to a form's FAR[3] file so they can be used at runtime without needing to first include them in the form as a direct resource. You can use this widget for runtime branding.

Usage

1. Open the **Palette** pane in the Maestro editor.
2. Locate the Bundle Resource Library component within the **Tools** folder.
3. Drag the component into the **View** pane or the **Wireframe**.

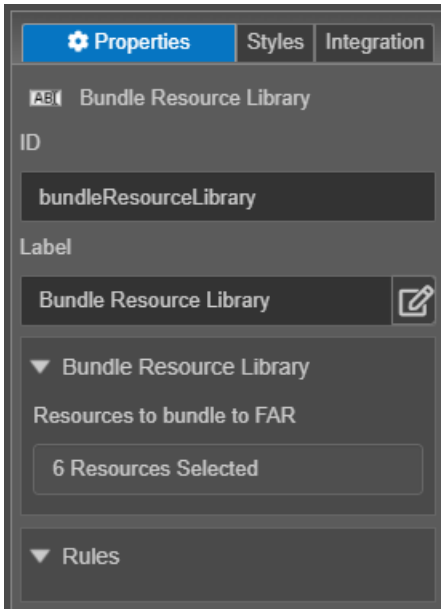
---

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

[2]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.

[3]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.

4. Configure the component's properties via the **Properties** pane.

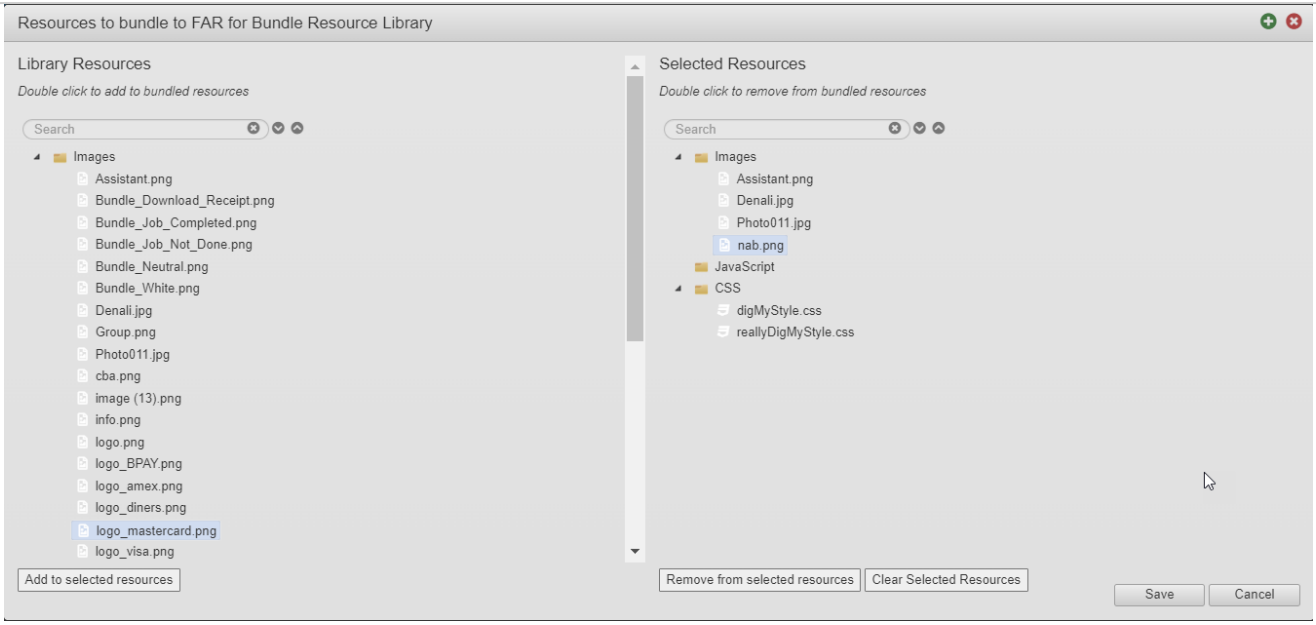


Properties

The following properties are available to the Bundle Resource Library component:

Bundle Resource Library

Resource to bundle to FAR
Click to add resources to a form's FAR file or view resources already added to the FAR file.



Select a resource or several resources from the Library Resources pane and click Add to selected resources to add them to the component.

Select a resource or several resources from the Selected Resources pane and click Remove from selected resources to delete them from the component.

Click Clear Selected Resources to delete all resources from the component.

Next, learn about the Runtime CSS Resource component.

Runtime CSS Resource

MaestroThe UI design product. | Form Builder | 21.11 This feature was updated in 21.11.

Maestro comes with the Runtime CSS Resource component, which allows you to add CSS[1] files to a form's FAR[2] file so they can be used at runtime without needing to first include them in the form as a direct resource. You can use this widget for runtime branding.

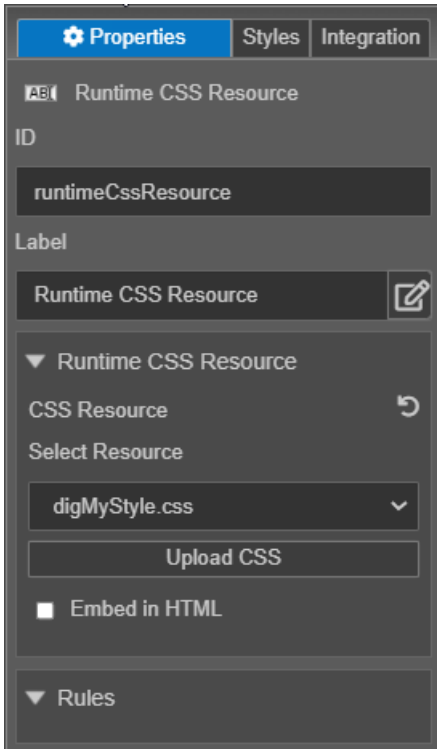
Usage

1. Open the **Palette** pane in the Maestro editor.
2. Locate the Runtime CSS Resource component within the **Tools** folder.
3. Drag the component into the **View** pane or the **Wireframe**.

[1]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.

[2]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.

4. Configure the component's properties via the **Properties** pane.



Properties

The following properties are available to the Runtime CSS Resource component:

Runtime CSS Resource

CSS Resource
Click to refresh

Select Resource
Select a CSS resource from the dropdown list of uploaded CSS files.

Upload CSS
Click to select a CSS file to upload to the form's FAR file.

Embed in HTML
Select to embed the CSS into the form.

Next, learn about the Web Font component.

Web Font

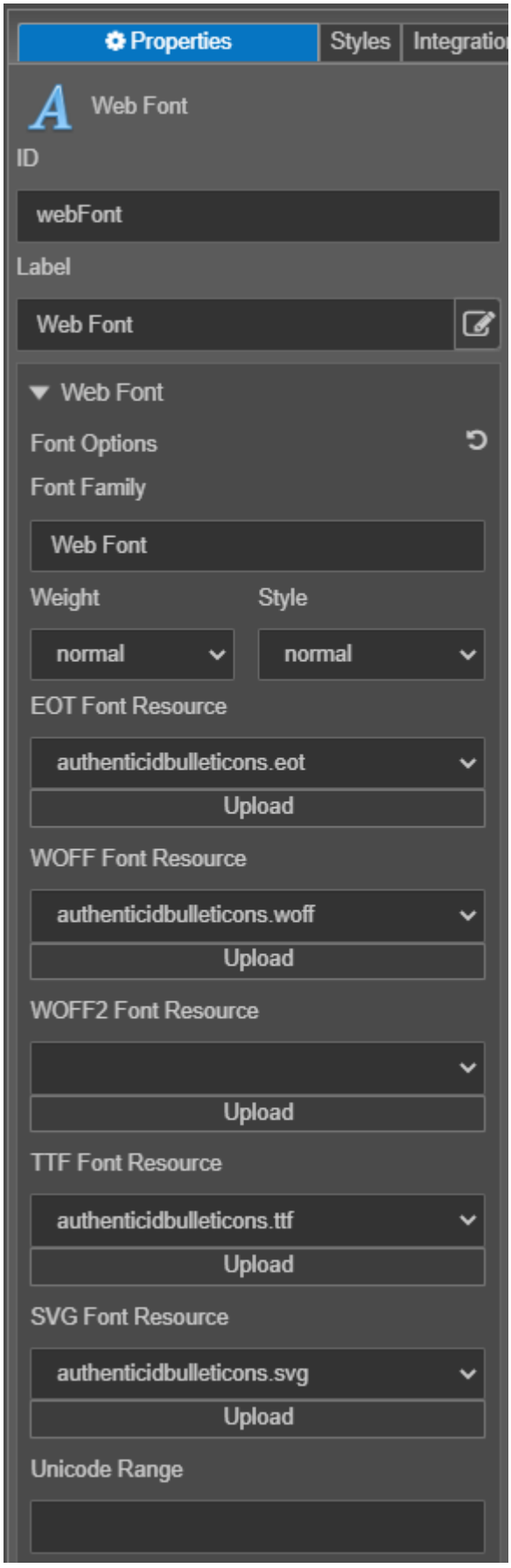
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Web Font component, so you can upload fonts as a resource and configure one to be used in a form.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Web Font** component within the **Tools** folder.
3. Drag the component into the **View** pane or the **Wireframe**.

4. Configure the component's properties via the **Properties** pane.


Properties
Styles
Integratio
Web Font
ID
webFont
Label
Web Font
Web Font
Font Options
Font Family
Web Font
Weight
Style
normal
normal
EOT Font Resource
authenticidbulleticons.eot
Upload
WOFF Font Resource
authenticidbulleticons.woff
Upload
WOFF2 Font Resource
Upload
TTF Font Resource
authenticidbulleticons.ttf
Upload
SVG Font Resource
authenticidbulleticons.svg
Upload
Unicode Range

Properties

The following properties are available to the Web Font component:

Web Font

Font Options Font font
Upload the font resources and configure the font or click Remove Local Override to clear the uploaded font
Font Family
Type a family of a font
Weight
Select a weight of a font from the dropdown list
Style
Select a style of a font from the dropdown list
EOT Font Resource
Select an Embedded OpenType (EOT)[1] font resource from the dropdown list or click Upload to use another EOT font file

[1]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

WOFF Font Resource
Select a Web Open Font Format (WOFF)[1] font resource from the dropdown list or click Upload to use another WOFF font file
WOFF2 Font Resource
Select a Web Open Font Format 2 (WOFF2)[2] font resource from the dropdown list or click Upload to use another WOFF2 font file
TTF Font Resource
Select a TrueType[3] font resource from the dropdown list or click Upload to use another TTF font file
SVG Font Resource
Select an SVG[4] font resource from the dropdown list or click Upload to use another SVG font

[1]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.

[2]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.

[3]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.

[4]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.

file

> NOTE
>
> To use SVG as a font in Maestro, first, you need to add a Web Font component to a template and configure it with an SVG font file. Then, you use @font-face CSS in the shared styles to apply the SVG as a font.

Unicode Range
Type an unicode range CSS descriptor to set the specific range of characters to be used from a font defined by `@font-face` and made available for use on the current page.

Next, learn about the "JavaScript Library" on the next page component.

JavaScript Library

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **JavaScript Library** component within the **Tools** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the JavaScript Library component:

Code Js jsLibrary
Upload the JS resource

Next, learn about the "Countdown Timer" on the next page component.

Countdown Timer

MaestroThe UI design product. | Form Builder | 19.05 This feature was updated in 19.05.

Maestro comes with the Countdown Timer component, which you can use in several scenarios, such as showing a time left before the form expires or managing a user session timeout.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Countdown Timer** component within the **Tools** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Countdown Timer component:

Seconds To Expire
Integer secondsToExpire

Number of seconds after which to fire OnExpiry event.

Options

Countdown Time Format
Text countdownFormat

Format pattern for countdown timer, as defined by AngularJS date filter. This only applies when the Show Countdown Value is selected.

Show Countdown Value
Boolean showCountdown

Show the value of the countdown timer.

Operating Mode
Option operatingMode

Act either as an idle timer, or require full manual control.

The following options are available from the **Operating Mode** dropdown:

Label	Value
Idle Timer	idle
Manual Operation	manual

The Idle Timer is reset on any user keystrokes or clicks on the page, regardless of whether it is on any component you can interact with or not.

Restart After Expiry
Boolean restartAfterExpiry

Act as an interval timer, by restarting each time it fires the `onExpiry` event

Countdown from Timestamp
Boolean countdownFromTimestamp

If this option is selected, when a timer is started the timestamp is recorded and the timer update triggers once per second for an active tab. For the inactive tab, that can be reduced to once per minute under certain circumstances. This means that the timer will expire once the timer is run after the start time plus the timer duration passes.

If this option is not selected, the timer will update once per second and decrement the number of seconds remaining, which will work as intended when the window is active for the duration of the timer. Due to this limitation, it is recommended to have this option enabled for all timers.

Rules

You can add a rule to the Countdown Timer component that will invoke specific actions on the timer using Maestro JavaScript API. The event type `onExpiry` is triggered when the timer finishes.

Mandatory
Boolean

A rule is required

Mandatory Message
Text

A mandatory message text

Rule Helpers

Rule helpers are context menu options that help you create a snippet of code for a specific task on relating to this component.

Reset Timer

The Countdown Timer component can be reset in two ways depending on its Operating Mode:

- Idle Timer / Auto - the timer is reset on user interaction with the form. Usually, it is used for session timeouts.
- Manual Operation - you control and reset the timer using the `Form.getItemFromPath('data.countdownTimer_1').resetTimer()` Maestro JavaScript API. Additionally, Manual Operation relies on user defined action through On Expiry rule in that widget.

Cancel Timer

The Countdown Timer component can be cancelled using the `Form.getItemFromPath('data.sessionTimeoutCountdownTimer').cancelTimer()` Maestro JavaScript API

Start Timer

The Countdown Timer component can be started using the `Form.getItemFromPath`

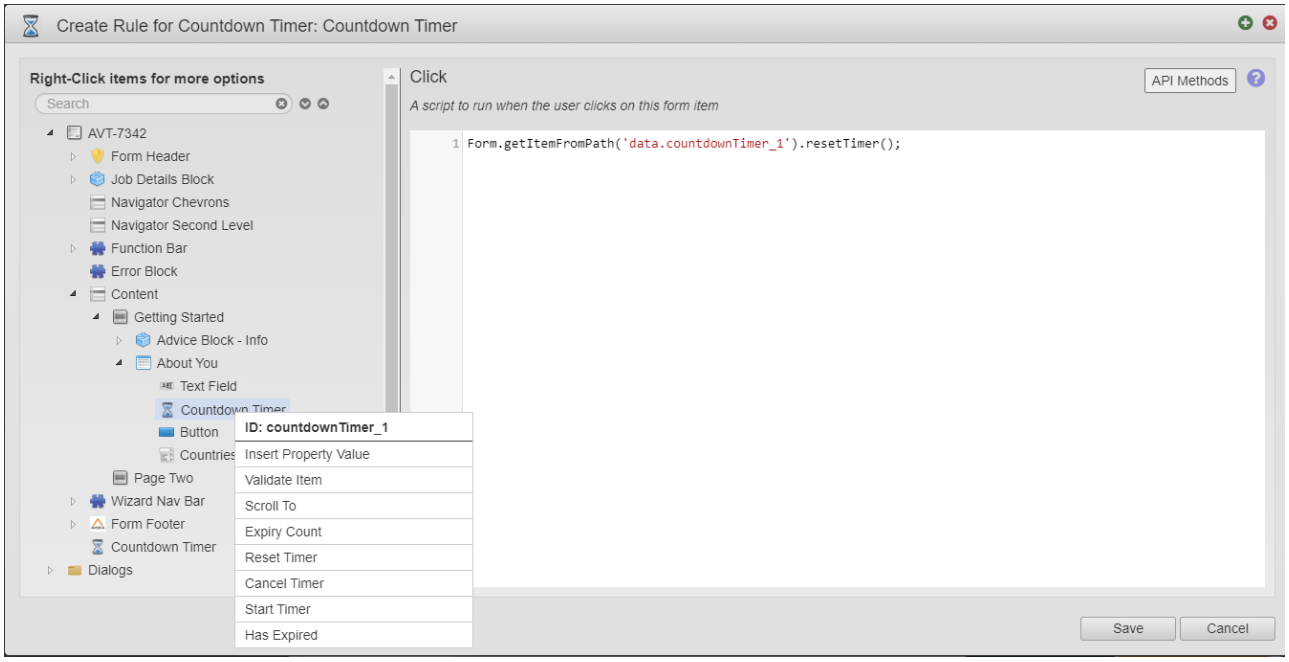
`('data.sessionTimeoutCountdownTimer').startTimer()` Maestro JavaScript API

Format Remaining Time

`Form.getItemFromPath('data.countdownTimer').formatTimeRemaining()` that uses the Countdown Time Format property to format it.

Use a rule helper to add an interaction with this widget to the JavaScript of a rule:

1. From the object that needs to interact with the timer, click Create Rule and select a rule you want to add to the timer, for example, Click event on a Button.
2. Select the Countdown Timer component and right-click on it



3. Select an action from the context menu. The action is added to the JavaScript script.
4. Click Save.

In the example above, we have added a click rule to the timer, which resets the timer by calling the `Form.getItemFromPath('data.countdownTimer_1').resetTimer()` Maestro JavaScript API.

Next, learn about the Autotab Controller component.

Autotab Controller

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Autotab Controller component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Autotab Controller** component within the **Tools** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Autotab Controller component:

Autotab Targets
Field Ref Map targets

Select Text Fields you want to apply auto-tabbing. Tabbing order is determined by DOM order, not by order of Field References. Use only Field Reference column - Key column is not applicable, leave blank.

Format

Format
Option format

Allowed characters

The following options are available from the **Format** dropdown:

Label	Value
All	all
Text	text
Alpha	alpha
Alphanumeric	alphanumeric

Label	Value
Number	number
Custom	custom

Pattern

Text pattern

Pattern regex - characters matching this regex will be disallowed. Refer to https://-github.com/Mathachew/jquery-autotab.

Next, learn about the "Form" on the next page component.

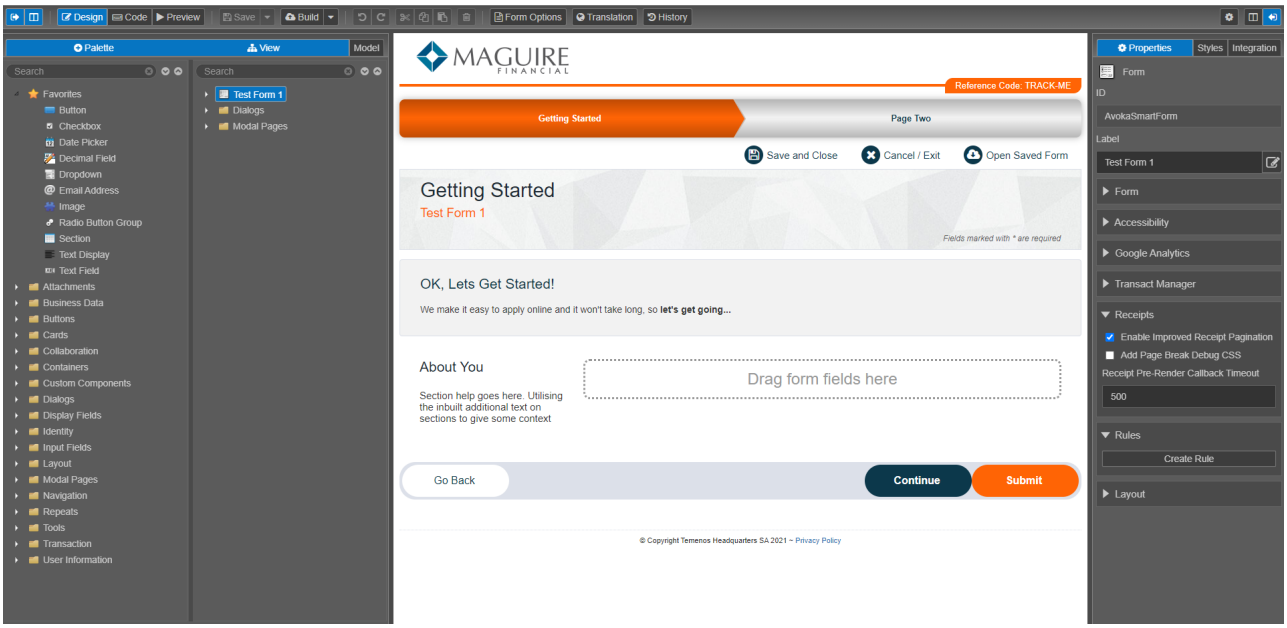
Form

MaestroThe UI design product. | Form Builder | 19.05 This feature was updated in 19.05.

Maestro comes with the Form component, which is a foundation of any user form, and it's where you start building your form from.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Form** component within the **Transaction** folder.
3. Drag the component into the **View** pane or the **Wireframe**.



4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Form component:

Page Icon Image favicon
The browser page / tab icon for this form

Popover Help Icon
Icon popoverHelpIcon

The icon used for popover help

Auto-Send Confirmation Emails
Boolean autoSendEmails

This property will auto-send an email on submit and save to the contact email address

Auto-Scroll Checkbox/Radio Button Selection
Boolean checkboxRadioAutoScroll

This property will enable auto-scroll to checkboxes/radio buttons to the middle of screen on selection

Leave Page Warning
Boolean enableLeavePageWarning

This property will prompt a confirm dialog if the user tries to leave the page.

Leave Page Warning Message
Text leavePageWarningMessage

This property will replace the default message for the leave page warning.

Mandatory Message Suffix
Text mandatoryMsgSuffix

This text will be concatenated to the end of the label when a field is deemed mandatory.

Progress Message
Text progressMsg

This message will be used when `Form.submit`, Form.taskUrlNavigate and `Form.postData` methods are called.

Accessibility

Dynamic Page Title
Boolean dynamicPageTitle

If checked, the title will be updated with the current progress of the application.

New Window Link Warning Message
Text linkWarningMsg

This text will be concatenated to the end of link text and read out on focus to screen readers.

Enable Mobile Zooming
Boolean enableZoom

This will enable pinch-to-zoom on mobile devices. This is not normally needed with good responsive design, but some accessibility reviews require it.

Component

Component JavaScript
Js componentJs

This field is only used when editing a component. If a JavaScript resource is selected it will be included as part of the component when it is published to the selected library.

Google Analytics

Google Tracking ID
Text gaTrackingId

Tracks page load and navigation views. To use Google Analytics, configure the Page Tracking feature in Manager for this form. The built in Google Analytics events that the Maestro runtime framework fires checks this value and will trigger on a non-blank value and will use the Google Analytics configuration as defined in the Page Tracking feature.

Base URL
Text gaBaseUrl

Identifies form within site context. (Optional)

Transact Manager

Session Expiry Redirect Timeout (Seconds)
Integer sessionExpiryRedirectTimeout

This is the timeout between a form being successfully submitted and the form redirecting to the Session Expired page of the Manager Portal. If this value is blank the form will default to a value of 300 seconds. The Session Expired page can be modified within the Manager portal using `Form.view.properties.sessionExpiryRedirectTimeout`.

Auto Background Save Interval (Seconds)
Integer autoSaveInterval

Performs an auto background save using the interval (In Seconds) entered, if set to blank it will never auto-save

Receipts | 19.05 This feature was introduced in 19.05.

Enabled Improved Receipt Pagination
Boolean enableImprovedPageBreaks

Performs improved receipt pagination. The most important thing to know about page breaks in printed HTML is that once a print related directive fails, for example, setting a block to stay as a whole on a page goes over a page, all child fields will ignore page related directives. For example, setting page-break-inside to avoid on a section that goes over a page can cause fields to split over the page break with part of the text on one page and the rest on the next.

Add Page Break Debug CSS
Boolean addPageBreakDebugCss

Adds page break in the debug CSS to a Maestro form that will show you the unbreakable

parts of your form as well highlight areas that may be problematic. This greatly helps with debugging pagination issues, for example, when a generated receipt generation is split across the page break. The colors explanations are summarized in the table below.

Border Color	Border Style	Explanation
Gray	Dashed	Hard coded CSS applied by Maestro to low level widgets. The only way to change this is by overriding the CSS.
Blue	Solid	Indicates that the row containing the field has been explicitly marked unbreakable by the user adding the pbu-avoid class.
Purple	Solid	Indicates that a page break will be forced at this point in the form.
Green	Dashed	Shows rows that are automatically marked as unbreakable by the default row level pagination JavaScript that makes lowest level rows automatically unbreakable.
Magenta	Solid	Highlights side by side content. This is potentially problematic content that will behave differently on Chrome or Puppeteer and PhantomJS.

Receipt Pre-Render Callback Timeout

Integer prePhantomCallbackTimeout

This timeout is used during rendering receipts in PhantomJS[1] to allow extra time after the form has loaded to allow all the form to fully initialize. There are various cases where this is needed and there is no single answer as to whether this is needed or what to use. If a receipt does not seem to be working as intended, try setting this value. Examples of cases where additional time may be required:

- External resources such as images are used
- Receipts with large numbers of repeats
- Large receipts may need additional time to fully initialize

[1]PhantomJS is a scripted, headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment. PhantomJS is based on WebKit making it a similar browsing environment to Safari and Google Chrome (before Chrome's fork of WebKit evolved into Blink). It is open-source software released under the BSD License.

Rules

Create Rules
Specifies rules for the form's events.

Layout

Specifies layout of the form.

Next, learn about the "Submit" on the next page component.

Submit

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Submit component so you can configure how a form is submitted.

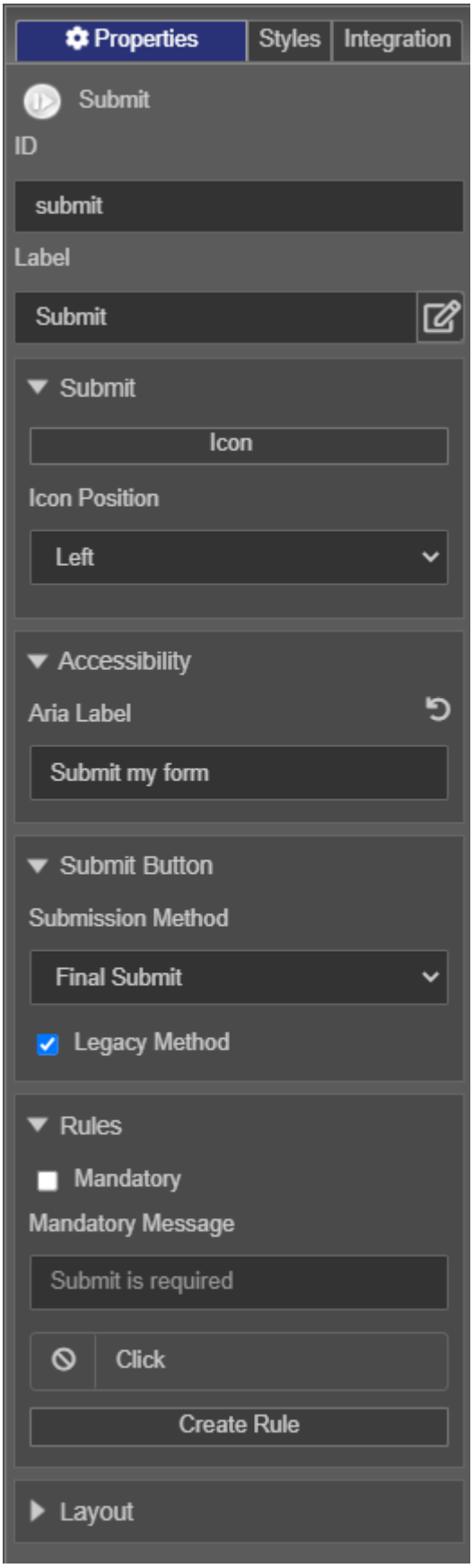
> NOTE
>
> This component inherits properties and shared styles from the Button component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Submit** component within the **Transaction** folder.

3. Drag the component into the **View** pane or the **Wireframe**.

4. Configure the component's properties via the **Properties** pane.


Properties
Styles
Integration
Submit
ID
submit
Label
Submit
Submit
Icon
Icon Position
Left
Accessibility
Aria Label
Submit my form
Submit Button
Submission Method
Final Submit
Legacy Method
Rules
Mandatory
Mandatory Message
Submit is required
Click
Create Rule
Layout

Properties

The following properties are available to the Submit component:

Icon
Icon icon

Select the button icon

Icon Position
Option iconPos

Select the button icon position

The following options are available from the **Icon Position** dropdown:

Label	Value
Left	left
Right	right

Accessibility

Aria Label
Text ariaLabel

Enter the Aria Label for this button, for example, Submit my form. For more information, see ARIA documentation

Submit Button

Submission Method
Option submissionMethod

Select the submission method from the **Submission Method** dropdown, which has the following options:

Label	Value
Final Submit	Submitted
Save Draft	Saved
Abandon Form	Abandoned

Legacy Method
Boolean legacyMethod

Use old-style HTML form post and redirect

Next, learn about the "Payment Type Selector" on the next page component.

Payment Type Selector

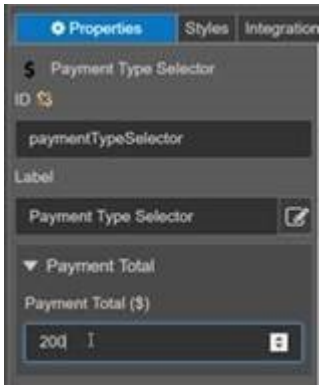
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Payment Type Selector component.

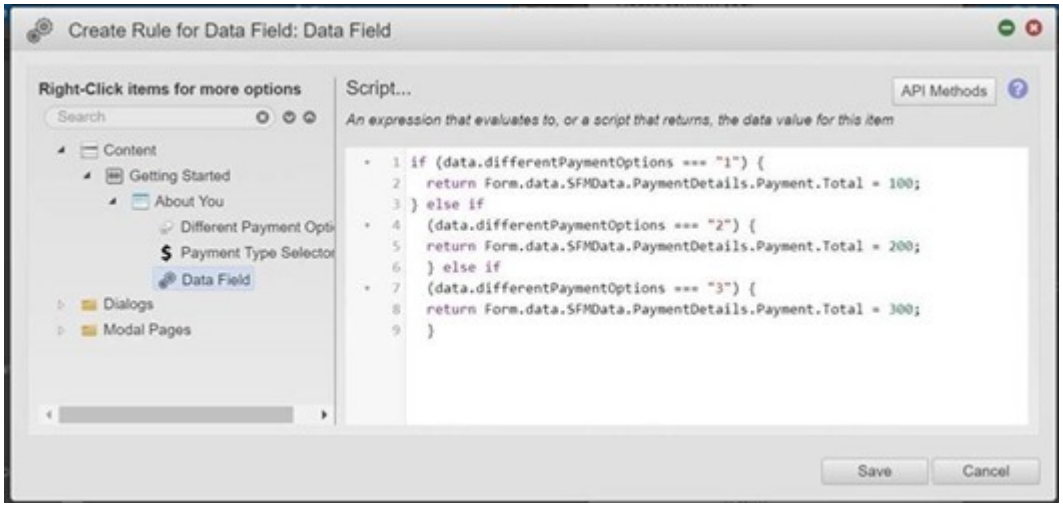
Businesses often require payment interface functionality within a form. The steps below detail the procedure to follow in order to configure payment interface functionality within your smart form. Note that Manager does not store, process or transmit payment card data; any payment card data interaction is directly between the end user's browser and the payment gateway or tokenization provider. We recommend that, when enabling our payment interface features, you ensure appropriate security, access and change control measures are in place to prevent accidental or malicious change of configurations.

Usage

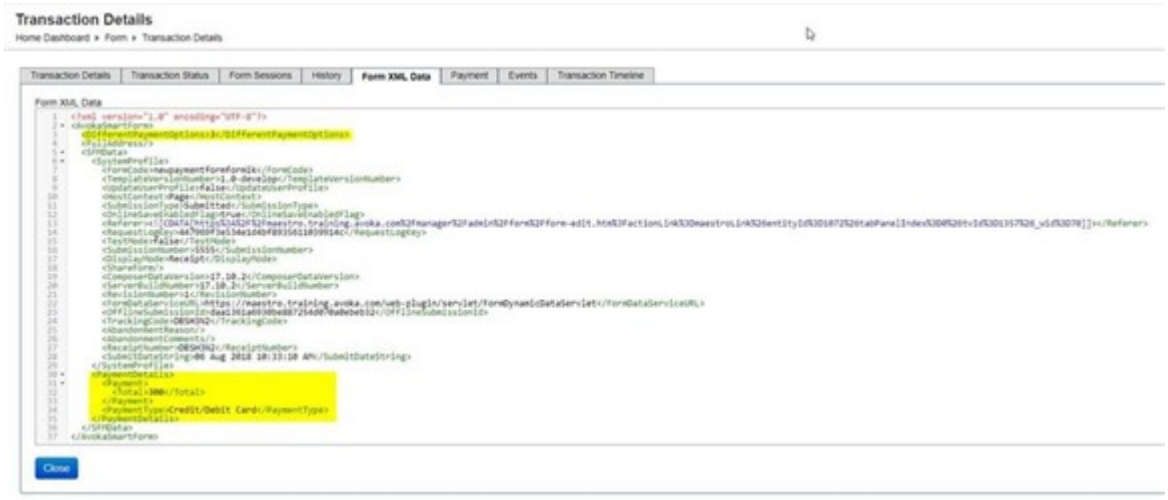
1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Payment Type Selector** component within the **Transaction** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.
5. It has to be non-zero for the Payment page to appear after submission. If you require a Fixed payment amount, update the Payment Total ($) property of the Payment Type Selector.



6. If you require a flexible payment amount, use a data field to update the XML element `Form.data.SFMData.PaymentDetails.Payment.Total`.



7. Manager will use the following values from the XML data to display the payment pages after a user submits the form:



Properties

The following properties are available to the Payment Type Selector component:

BPAY Supported Boolean optionBPAY
Check to include BPAY
Credit/Debit Card Info Text Rich Text ccDbInfoText
Credit card/Debit card info text.

BPAY Info Text
Rich Text bpayInfoText

Credit card/Debit card info text.

Selected Icon Class
Icon selectedIconClass

Selected icon class name.

Unselected Icon Class
Icon unselectedIconClass

Unselected icon class name.

Help Text

Popover Help Text
Help Text helpText

Image Resources

Select VISA Image
Image srcVISA

Select from the available images resources, or upload a new one

Select MasterCard Image
Image srcMASTER

Select from the available images resources, or upload a new one

Select AMEX Image
Image srcAMEX

Select from the available images resources, or upload a new one

Select Diners Image
Image srcDiners

Select from the available images resources, or upload a new one

Select BPAY Image
Image srcBPAY

Select from the available images resources, or upload a new one

Payment Total

Payment Total ($)
Integer paymentTotal

Enter the payment total

Supported Credit Cards

Visa
Boolean optionVISA

Check to include VISA

Master Card
Boolean optionMASTER

Check to include Master Card

AMEX
Boolean optionAMEX

Check to include AMEX

Diners
Boolean optionDINERS

Check to include Diners

Next, learn about the "iFrame" on the next page component.

iFrame

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the iFrame component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **iFrame** component within the **User Information** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the iFrame component:

URL Url url

Height Integer height

Next, learn about the "Advice Block - Info" on the next page component.

Advice Block - Info

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Advice Block - Info component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Advice Block - Info** component within the **User Information** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Advice Block - Promo" on the next page component.

Advice Block - Promo

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Advice Block - Promo component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Advice Block - Promo** component within the **User Information** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Popover Text Display" on the next page component.

Popover Text Display

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Popover Text Display component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Popover Text Display** component within the **User Information** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Properties

The following properties are available to the Popover Text Display component:

Help Text

Popover Help Text Rich Text popoverContent
Enter the text to display in popover.

Popover Options

Popover Button Label Rich Text PopoverButtonText
Enter the text to display. This will be read out on focus for screen readers.

Next, learn about the "Popover Widget Container" on the next page component.

Popover Widget Container

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Popover Widget Container component.

Usage

1. Open the **Palette** pane in the Maestro Editor.
2. Locate the **Popover Widget Container** component within the **User Information** folder.
3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.

Next, learn about the "Select Language" on the next page component.

Select Language

MaestroThe UI design product. | Form Builder | 22.04 This feature was updated in 22.04.

Maestro provides the Select Language component (widget), which allows form users to select a language that they want to use on a form. You must configure a language or multiple languages for a form before a user can select them.

NOTE

This component inherits properties and shared styles from the Dropdown component.

Usage

1. Open the **Palette** pane in the Maestro editor.
2. Locate the **Select Language** component within the **User Information** folder.



3. Drag the component into the **View** pane or the **Wireframe**.
4. Configure the component's properties via the **Properties** pane.
5. Configure the component's styles via the **Styles** pane.

Properties

The following properties are available to the Select Language component:

Populate using Form translations Boolean useFormLanguages
Use the translations in the Form to populate the list

Sort Options Boolean sortOptions
Check to automatically sort options by label

Hide on single language
Boolean hideSingleLanguage

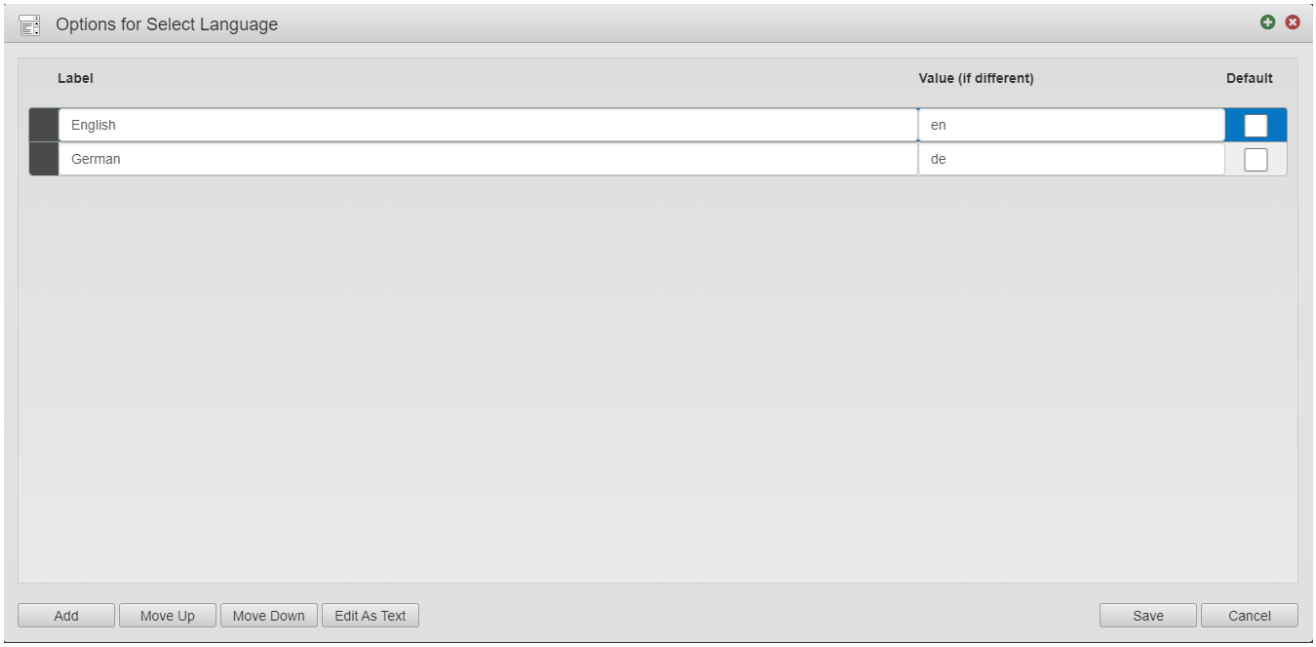
Hide this widget when the form only has a single language

Options
List options

The options list for the dropdown

By default, the **Options** property contains the following values:

Label	Value
Language name 1	Locale code 1
Language name 2	Locale code 2

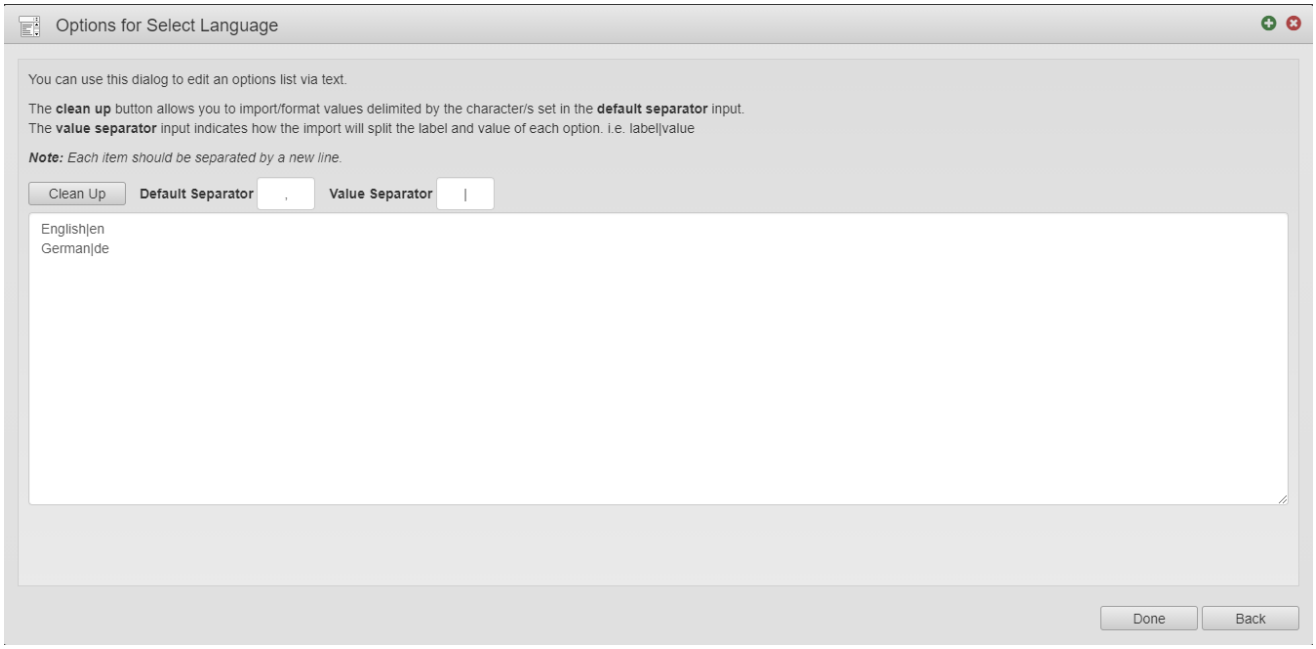


Update the language details or select the Default checkbox to make a selected language a default one and click Save.

Click Add to add a new line to the list where you need to add a language name and a locale code.

Select a language and click Move Up or Move Down to re-order its position in the list of languages.

Click Edit as Text to see the language configuration as a text file. You can change the separators used in the text.



Sort Options
Boolean sortOptions

Check to automatically sort options by label

Icon
Icon icon

Select the dropdown icon

Add Blank Value
Boolean addBlankValue

Check to add a blank value to the dropdown

Accessibility

Placeholder Text
Text placeholder

Help Text

Popover Help Text
Help Text helpText

Styles

The following styles are available to this component:

Avalon Language Selector

```
form[name='form'] {
  .avalon-language-selector {
    width: auto;
    margin-right: 20px;
    .wdg-label {
      display: none;
    }
    .av-select-overlay {
      display: inline-block;
      float: right;
      width: 150px;
      background-color: transparent;
      border: 0;
      border-radius: 0;
      padding: 0;
      min-height: 30px;
      outline: none;
      box-shadow: none;
      cursor: pointer;
    }
    .wdg-input {
      .av-select-value {
        right: 25px;
        padding: 0;
        font-family: @ux-default-font-family;
        font-size: @ux-default-font-size;
        color: @ux-default-font-color;

      }
      .wdg-select-icon {
        padding: 0;
      }
      &:hover, &:focus {
        .av-select-value {
          text-decoration: underline;
        }
      }
    }
  }
}
```

After you've configured the component, you can preview the form to switch between the available languages to verify it works as expected.

Next, learn about the "Attachment Field" on page 671 component.

Maestro Receipts Overview

MaestroThe UI design product. | Form Builder ■ Platform Developer | 21.05 This feature was updated in 21.05.

Maestro allows you to build receipts that are read-only statements for various stages of form application processes, for example, a successful form submission, or all pages of a form concatenated together. It uses a printable CSS so it's rendered similarly to a printed version. By default, Maestro auto-generates a receipt based on the design and structure of the form. A receipt can be an HTML or a PDF file. You can always preview a receipt in the Maestro editor.

Sometimes, you may want to create custom a receipt having a very different structure to a web form or optimized for printing. The later is often a case as a dynamic and interactive form can be a lot less space efficient, as well as presenting data on a page along with the interactive form used for data collection which may not be efficient for creating a printable record of the transaction. The Maestro editor allows you to create these receipts using the Receipt Designer tool.

If a form has a receipt design, then when you build a form with the Use Separate Receipt Design option selected, Maestro generates a FAR file containing two complete Maestro designs - a form and a receipt, and an XML configuration file specifying which receipt design to use when the FAR file is published in Manager. If there is no receipt in the FAR file, the interactive form is used for the receipt.

When creating a receipt, a Form Builder will:

- Use the Receipt Designer to create a custom receipt
- Remove tooltips, help text, and other changes to make the most of receipt improvements

NOTE

Manager delivers receipts to applicants and also allows you to perform some minor receipt configuration.

Next, learn how to use Receipt Designer.

Receipt Designer

MaestroThe UI design product. | Form Builder | | 19.05 This feature was introduced in 19.05.

Maestro allows you to design a receipt using the Receipt Designer feature, which is available in the Maestro editor since version 19.05.

Creating a receipt is very similar to creating a form, so only a basic understanding of form development is required. We will illustrate it with a concrete example where we will:

1. Create a form
2. Create a receipt
3. Download the Form Version Archive file
4. Deploy the form to Manager
5. Test the receipt

Create a form

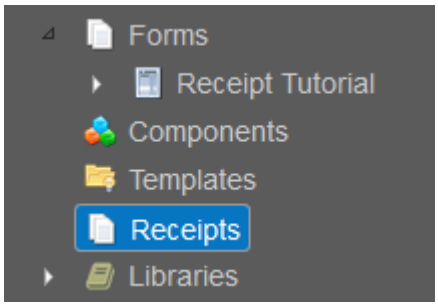
1. Create a form with First Name and Last Name text fields in your project.



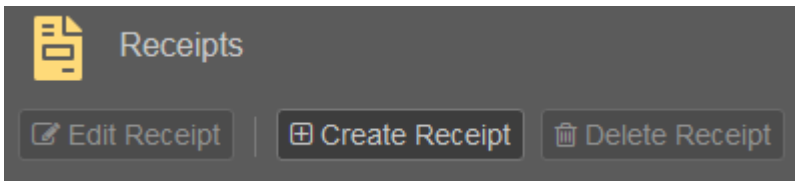
2. Select the **Maguire - Default Form** template when creating the form to make the form submittable without any further configuration. This isn't strictly necessary though, as we can use a Receipt Test Harness to test the receipt without navigating and submitting the form.

Create a receipt

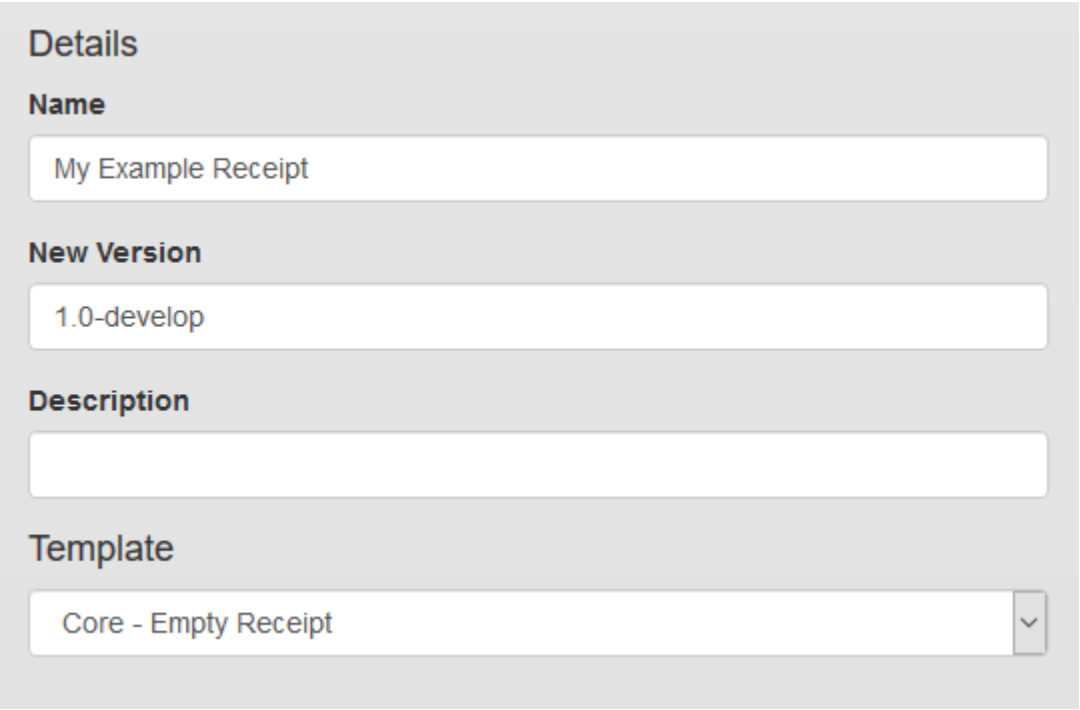
1. Click **Receipts** in the same project as the form.



2. Click **Create Receipt**.



3. Enter a name for the receipt in the **Name** field.
4. Select the **Core - Empty Receipt** template from the **Template** dropdown list. This is the simplest receipt template you can use.



5. Click **Create**.

NOTE

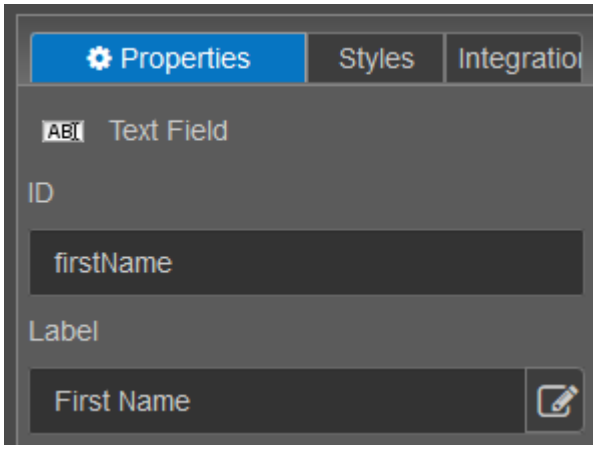
In a real-world project, we recommend creating and publishing a custom receipt template based on an existing receipt template. Functionally, receipt templates are identical to form templates, so all standard template practices apply.

6. After the receipt opens in a new window, drag two text fields into the receipt.



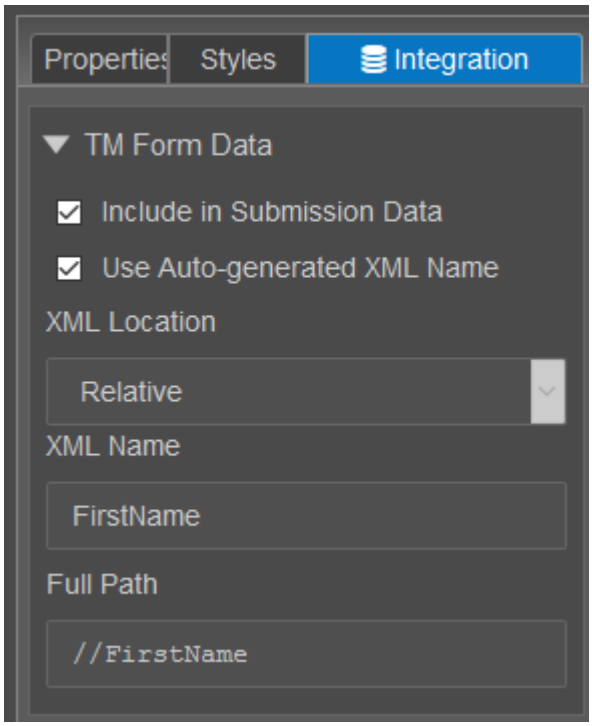
For the data from a form to appear in the receipt, the fields need to have the same XML path. To accomplish this, we suggest creating a domain model and sharing it between the receipt and form. In this topic however, we'll just manually ensure the component IDs are the same.

7. Change the ID of the first text field to firstName and the ID of the second text field to lastName. Their labels should automatically regenerate.



NOTE

By default, XML names are auto-generated based on component IDs. If this option is disabled via the **Integration** properties, you'll need to manually set the XML names rather than the component IDs.



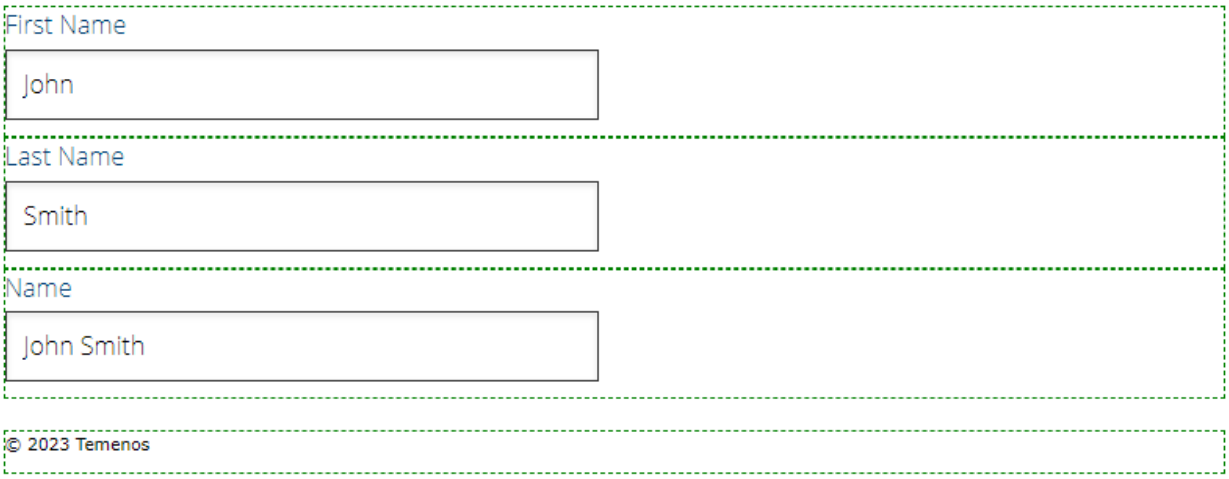
So far we've recreated the form in the receipt, but typically, receipts will look different from their forms.

Here is one example of how to approach this:

1. Drag a Text Field component into the receipt.
2. Change the ID of the component to fullName.
3. Click **Create Rule**.
4. Under the **Calculation** heading, click **Script**.
5. Copy and paste the following code into the code editor: `return data.firstName + " " + data.lastName;`
6. Click **Save**.

   This code concatenates the first name and last name into a single line of text.

7. To see this, switch to the **Preview** tab and fill out the First Name and Last Name fields. The Name field will update with the combined result of these values.



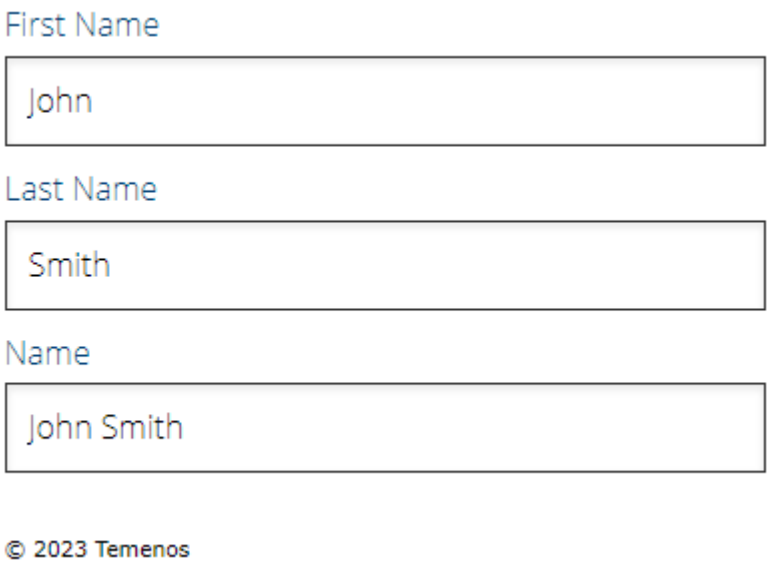
NOTE

The dotted outlines in the preview show the page breaks to help you debug receipts.

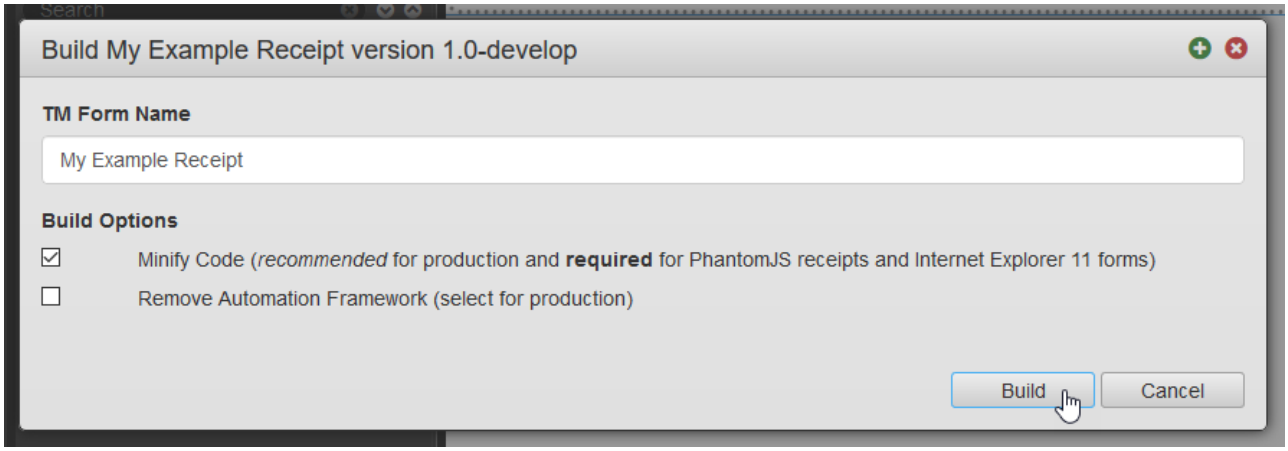
To switch off the page break debugging:

- Select the form root and click the Properties tab.
- Expand the Receipt and clear the Add Page Break Debug CSS checkbox.

The receipt preview without the page break is shown below:

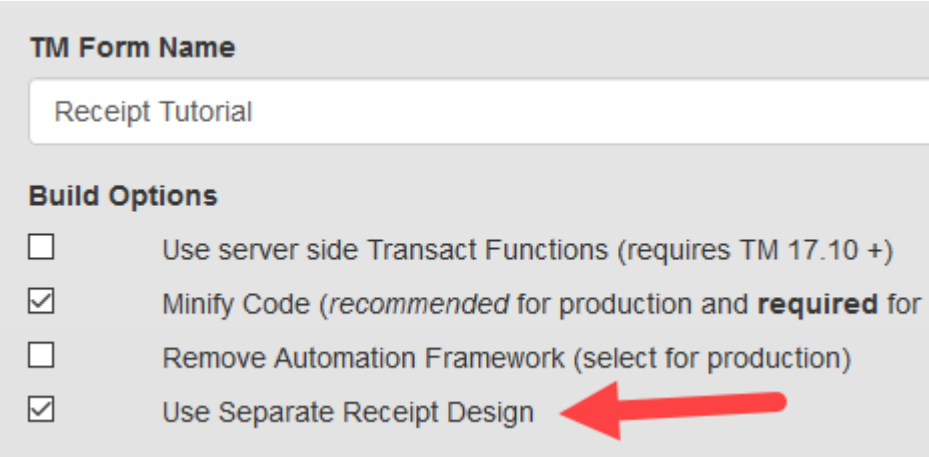


8. Finally, click **Build** to make the receipt available to the form.

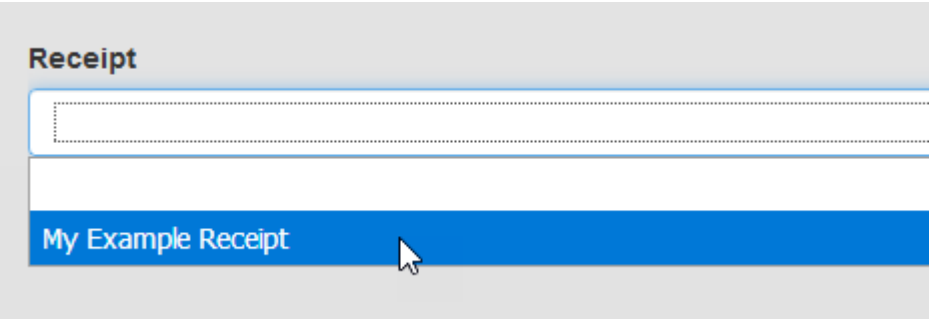


Download the Form Version Archive file

1. Return to the form and refresh the UI.
2. Click **Build > Build and Download TM Form Version** to build the form.
3. Enable **Use Separate Receipt Design**.



4. Select a receipt from the **Receipt Version** dropdown. If the receipt isn't available, verify that you clicked the **Build** button in the previous step.



5. Select a version number from the **Receipt Version** dropdown.

6. Click **Build**.
7. Click **Download TM Form Version** to download the Form Version Archive file.

NOTE

Multiple forms can use the same receipt.

Deploy the form to Manager

NOTE

Whether or not a custom receipt has been designed in Maestro, the process of deploying a form to Manager is the same.

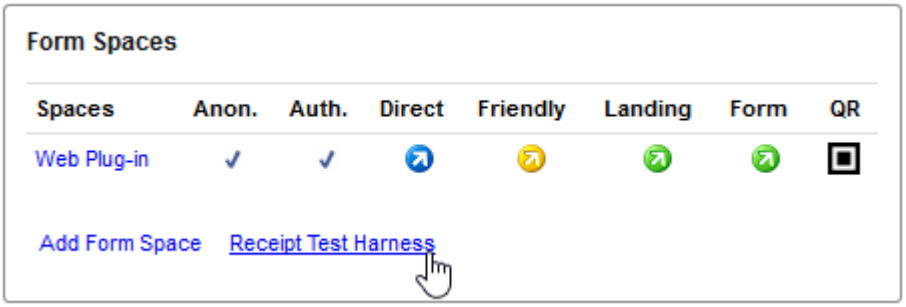
1. Log in to Manager.
2. Navigate to **Forms > Forms**.
3. Click **Import Form**.
4. Click **Browse**.
5. Select the Form Version Archive file downloaded in the previous step.
6. Click **Upload**.
7. Select **Create a Form**.
8. Click **Proceed to Import**.
9. Click **Import**.

For more information, see import forms.

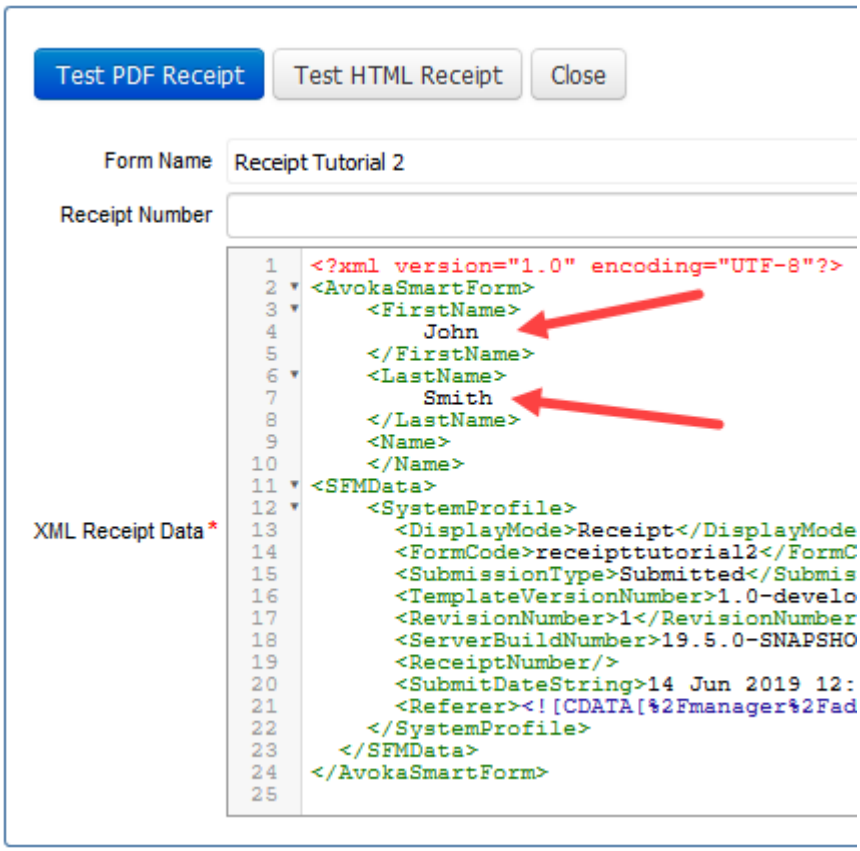
Test the receipt

Rather than testing the receipt by repeatedly submitting the form, use Manager's Receipt Test Harness feature:

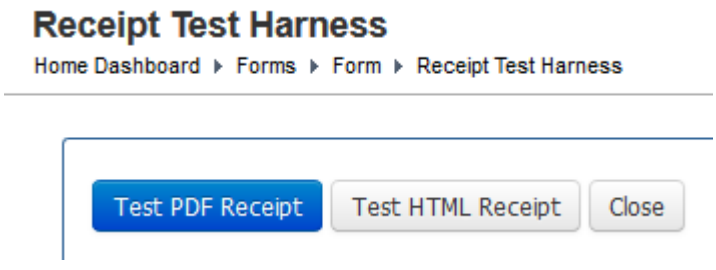
1. Navigate to the **Form Dashboard** for the imported form.
2. Click **Receipt Test Harness**.



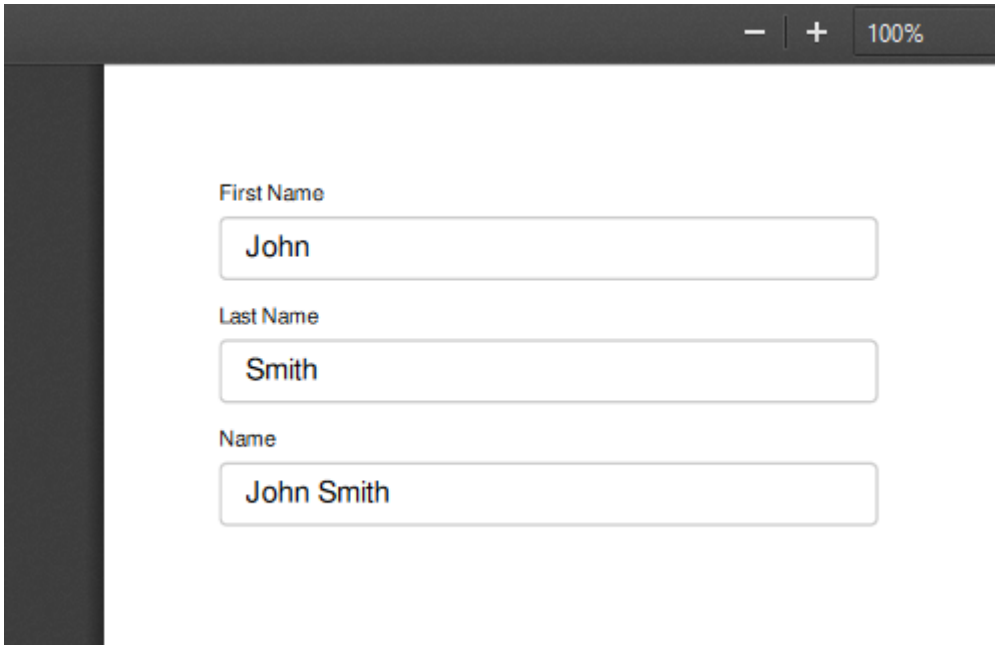
3. Inside the **XML Receipt Data** field, change the values of the FirstName and LastName fields.



4. Click **Test PDF Receipt** or **Test HTML Receipt**.



The receipt contains the data from the form, as shown below:



This is a simple receipt, but it does demonstrate:

- How a receipt can be designed in Maestro
- How data flows from a form and into a receipt
- How form rules can affect the display of data

Now, with these fundamentals, you can easily create more complex receipts.

Next, learn how to edit a form in Design Mode.

Build a Receipt

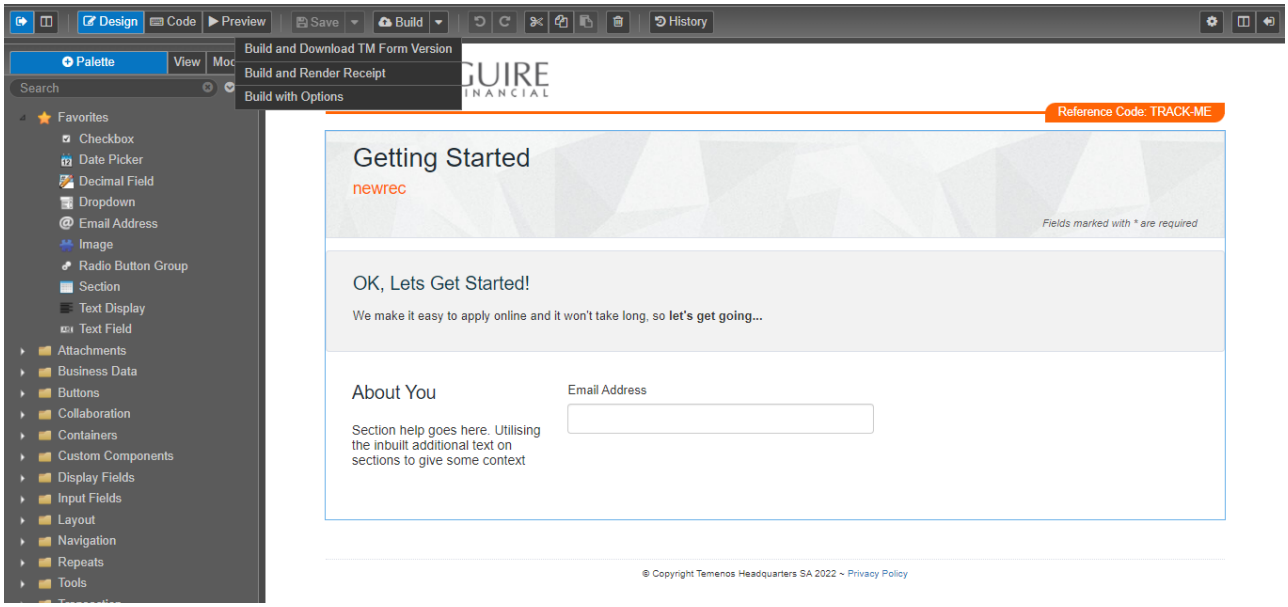
MaestroThe UI design product. | Form Builder ▪ Platform Developer | 21.05 This feature was updated in 21.05.

Maestro allows you to build a receipt in a similar fashion as you build a form. You should always build a receipt after you've created it or after you've done some changes to it. While you are editing the receipt, you can just build it or build it with various publishing options. However, receipts have less building options as that of the forms.

The receipt is fetched each time the form is built so that Maestro can pick up the latest version. If the receipt has been changed since its last built, Maestro will prompt you to re-built it. The Maestro editor makes this publishing easier for you. While you are editing a form, you can open the receipt and make a change to it, build the receipt and then go back to the form tab and click Build. The new changes from the receipt will be included in the form build.

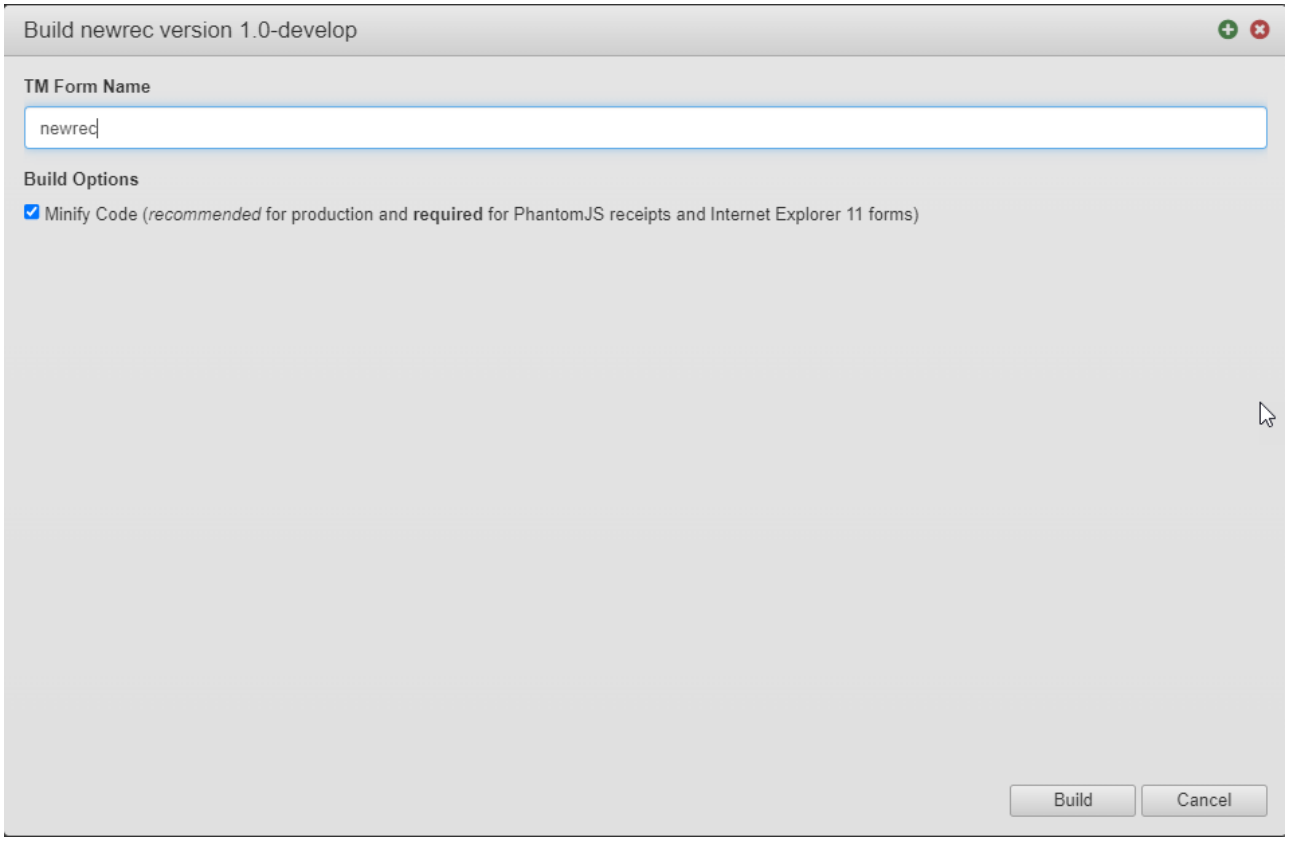
To build the receipt of a Maestro form:

1. Open a form in the Maestro editor and select one of the following options from the Build dropdown menu:



2. Click Build to build the receipt.
3. Click Build and Download TM Form Version to build a TM Form Version and then immediately download the ZIP file needed to import the Form Version into Manager.
4. Click Build and Render Receipt to build a TM Form Version and render the receipt in a new tab.

5. Click Build with Options to apply the following options to the receipt before building the TM Form Version.



   - Enter a new TM Form Version name in the TM Form Name field, in case you want to change it from the default name, which is the existing name of the receipt.
   - Select the Minify Code checkbox to reduce the size of the receipt and related JavaScript[1] resources without changing its functionality so they can be sent out to a client browser more efficiently.

Next, learn how to edit a receipt.

[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.

Shared Styles

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to create shared styles that can then be used to attach visual attributes to one or more components. Out of the box, Maestro's templates provide their own range of shared styles. You can, however, create custom shared styles. You can think of a shared style as a place for grouping of CSS logically.

Styling should be done by a Template Designer within the template. Templates ensure that all forms follow a corporate style guide. The Template Designer can help to determine whether you should be making style changes in the form or if the template needs to be updated. If a change is needed and it applies to other forms, it is best that it is done in the template so that all forms can access the change. This type of management will allow all forms to continue to follow the corporate style-guide, which keeps all forms consistent.

Templates allow for corporate brand control and for all forms to have a consistent look and feel. Templates are created by a Template Designer based on a corporate style guide. The responsibility of the Template Designer includes styling areas such as the navigator, footer, and styling of specific components.

Ideally, individual forms and components should never deviate from a corporate style guide. A Form Builder should not change any styling without first talking to the Template Designer.

Usually, the Template Designer creates a shared style and make it available to other components so Form Builders can use it. The style can be added a component and then moved up and down to adjust its precedence as required. When the shared style is not needed or should not be used by Form Builders, the Template Designer can remove it from a component.

You can change the style of components in brands.

Next, learn how to create a shared style.

Create a Shared Style

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to create shared styles that can then be used to attach visual attributes to one or more components. Out of the box, Maestro's templates provide their own range of shared styles. You can, however, create custom shared styles. You can think of a shared style as a place for grouping of CSS logically.

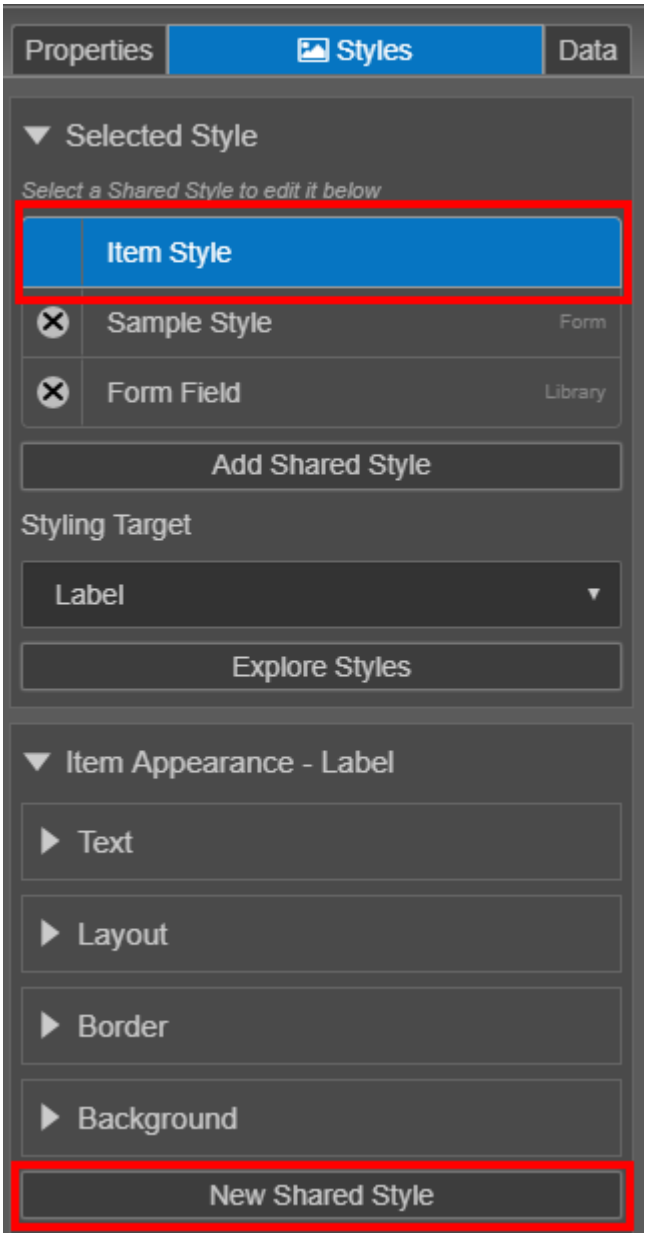
There are two ways to create a shared style:

- Create a new shared style and then apply it to a component
- Update the Item Style of a component and then save as a new shared style

To create a shared style:

1. Open a design in the Maestro editor.
2. Select a component and click the Styles tab to view available shared styles. For more information, see the Styles pane.

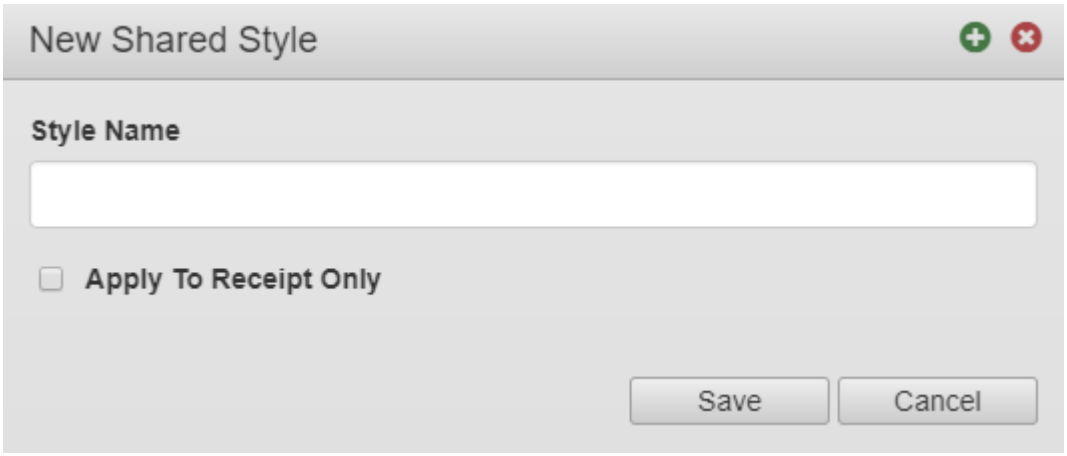
3. Expand the **Item Appearance - Container** pane.



4. Click **New Shared Style**.

> NOTE
>
> This button is only visible when the **Item Style** is selected and the **Item Style** has not been edited via the GUI.

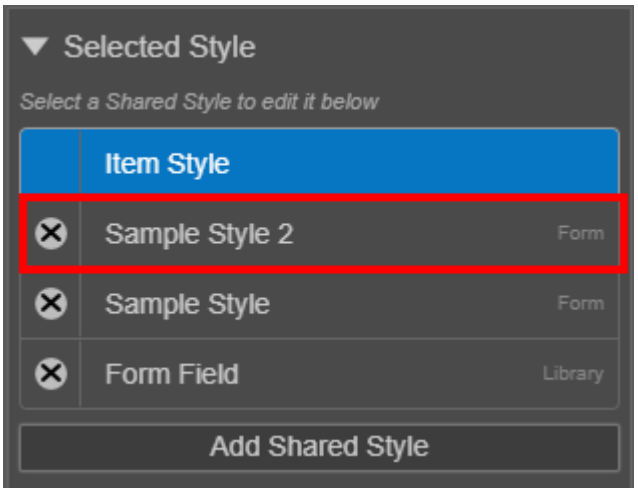
5. Enter a name for the style in the **Style Name** field.



6. Select the **Apply To Receipt Only** checkbox to apply the style only when the component is rendered as a receipt. This is optional.

> NOTE
>
> You can change this option after the style has been created.

7. Click **Save**. The new style will appear under the **Selected Style** heading.



> NOTE
>
> The new style is not selected by default. Make sure you always check the selected style before applying any style changes.

Next, learn how to make a shared style available.

Make a Shared Style Available

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to make a shared style available to other component. When a shared style is created, it's applied to the instance of the component upon which it was created and made available to all components of the same type.

For example, a shared style created on a Section component will be applied that Section component and made available to other Section components.

It is, however, possible to make styles available to other types of components.

To make a shared style available to a component:

1. Open a design in the Maestro editor.
2. Right-click on a component in the Palette tab and select **Available Styles for [Component]**. For more information, see the Palette pane.
3. Select a style from the **Make another style available dropdown**.
4. Click **Save**. All components of that type will now have access to that style. For more information, see available component styles.

NOTE

If you adjust some aspect of the styling, the changes will immediately be applied to all fields of that type in your entire form.

Next, learn how to apply a shared style.

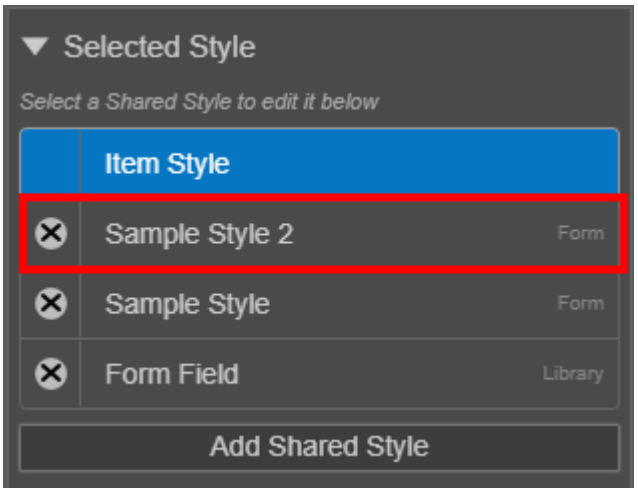
Apply a Shared Style

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to apply a shared style to a component to change it look and feel. You usually want to do this after you have created a new shared style.

To apply a new shared style to a component:

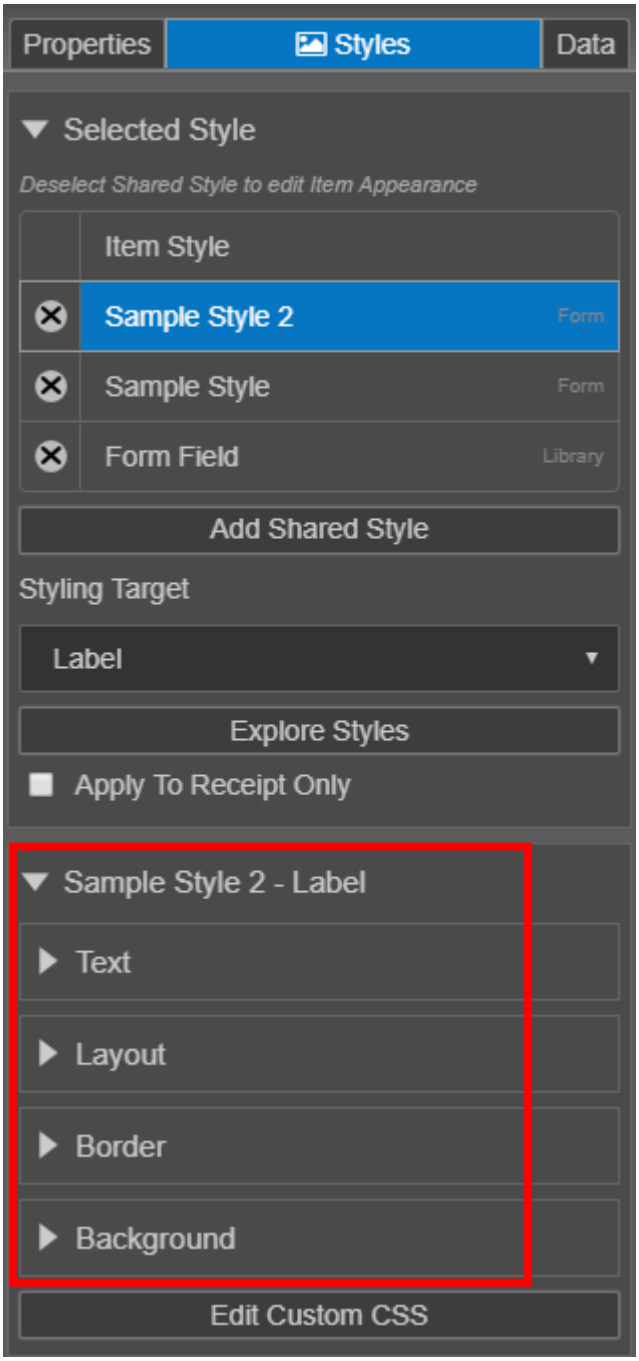
1. Open a design in the Maestro editor.
2. Select a component and click the Styles tab



3. Select the new style and then select the correct Styling Target.

> NOTE
>
> The new style is not selected by default. Make sure you always check the selected style before applying any style changes.

4. Expand the Item Appearance section to display the selected style name.

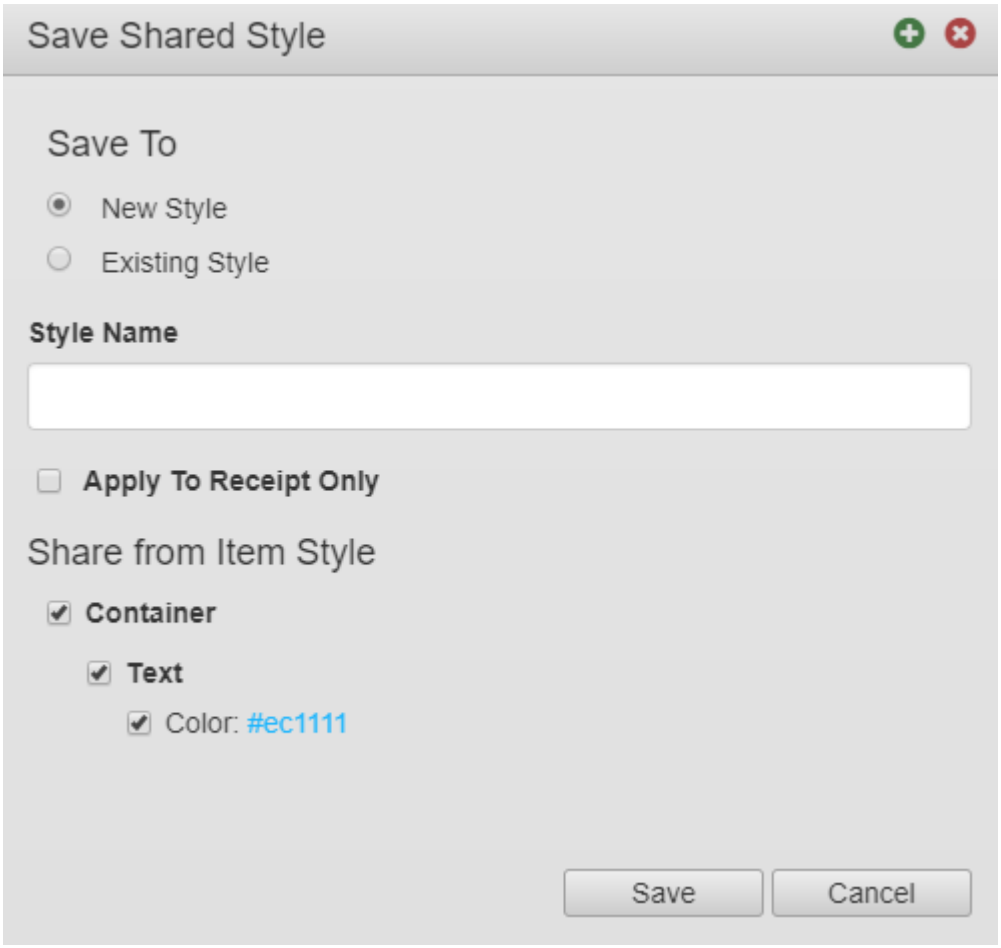


NOTE

You can also click the Edit Custom CSS button to make you styling changes using CSS.

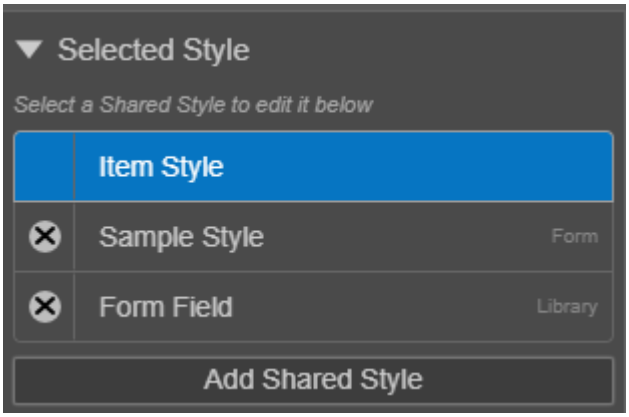
5. Click Save to Shared Style to keep your changes.

6. Select some of the following options shown below:



- Select New Style to create a new shared style. Or, select Existing Style o save the changes to an existing shared style.
- Enter a name of the new style you are creating in the Style Name field, which is displayed when the New Style options is selected. Or, select an existing style from the Select Style dropdown list, which is displayed when the Existing Style option is selected.
- Select the Apply To Receipt Only checkbox to create styles that are only displayed in the receipt, and not used in the form.
- Under Share from Item Style, select what changes, made to the Item Style, you want to be saved in the new or existing style.

- Click Save. The new style will be displayed in the Selected Style list:



> NOTE
>
> The word Form displayed next to the new shared style indicates that the style has not been published, and is only available in the template or from where it was created.

> NOTE
>
> The word Library displayed next to the new shared style indicates that the style is being pulled from a library. Many standard components include a default style that is inherited from the library that they are defined in. This may vary depending on how Maestro has been configured for your organization. You may remove or modify this standard style, but generally, this should be avoided. You can always add it back if you remove it accidentally.

> NOTE
>
> If you want to use a shared style that you have created in other forms, then you will need to publish this shared component to a library.

7. Click Clear Items Styles to undo any styling changes made since the last time the style was saved.

Next, learn how to add a shared style.

Add a Shared Style

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to add an existing shared style to a component, which you normally do after you created a new shared style.
To add a shared style to a component:

1. Open a design in the Maestro editor.
2. Select a component and click the Styles tab to view available shared styles. For more information, see the Styles pane.
3. Click **Add Shared Style**.
4. Select a style from the dropdown list.

   NOTE

   If a style isn't available from the dropdown, it was probably created for a different type of component. To resolve this issue, see style availability.

5. Click **Save**.

Add a Shared Style to Multiple Components

If you want to add the same set of styles to multiple components of the same type, you don't have to do it one-by-one. You can instead apply the same styles to all components of the same type in just a couple of clicks.

To add a shared style to multiple components:

1. Add a style to one of the components.
2. Right click the component in the **View** pane.
3. Click **Add Shared Styles to Every [Component]**.

Next, learn how to move a shared style.

Move a Shared Style

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to move shared styles. The order in which the styles appear in the **Styles** tab can affect how the styles are rendered in a form. This is referred to as style precedence. You can, however, control the order of the styles by moving them up and down the list.

To move a shared style:

1. Open a design in the Maestro editor.
2. Select a component and click the Styles tab to view available shared styles. For more information, see the Styles pane.
3. Right click a style.
4. Click **Move style up** or **Move style down**.
5. Click **Save**.

It's difficult to predict how style precedence will affect the appearance of a component. If the styles don't appear to be applied in the correct order, some trial-and-error will be required.

NOTE

If a style can't be moved up or down, such as when there's only one style in the list, these options will not be available.

Next, learn how to remove a shared style.

Remove a Shared Style

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to remove a shared style from a component, so the style is not available for Form Builders.

To remove a shared style from a component:

1. Open a design in the Maestro editor.
2. Select a component and click the Styles tab to view available shared styles. For more information, see the Styles pane.
3. Click the **X** icon next to the name of a style.
4. Click **Save**. The style will disappear from the **Selected Style** heading.

NOTE

You can add the style back when you need it.

Next, learn how to ?.

Template Overview

MaestroThe UI design product. | Form BuilderTemplate Designer | 21.11 This feature was updated in 21.11.

Maestro allows you to create templates that accelerates application form development process and unifies look and feel of forms, which are based on the templates. Templates are mostly created by Template Designers.

If your organization creates forms using Maestro, they probably intend to create multiple forms with similar layouts and styles. To streamline this process, Maestro allows you to use templates. You can think of a template as a form that is published in a library so it can be used as a starting point in a new form development.

The primary purpose of a template is to define a structure that can be used across multiple forms. A template usually includes a header, footer, and navigation or progress bar. You might, for instance, want to use the same header, footer, and navigation for more than one form. To accomplish this, you can create these components once and place them in a template. Then, when creating a form, you select a template so the form is populated with these components.

From that point onward, any changes made to the template – and any changes made to the components in that template – will be reflected in any forms using the template.

Templates are also responsible for brands, which are used to control the appearance of templates, including their colors, heights, widths, and padding.

All templates have a default brand that cannot be deleted. Most organizations will only need this default brand. You can, however, create multiple brands to create variations of a template. This is called white-labelling[1]. Maestro editor gives you all mechanisms to create your own while-label brands.

Using a template provides you with the following benefits:

- Avoid implementing the same styles and behavior in multiple places

[1]White-label software is software that is purchased by a company from a service provider and re-branded as its own.

If you have multiple forms with similar styles, the only way to implement those styles across the forms without templates is to duplicate the work. Even if you're just working on a couple of forms, this is an unproductive use of time.

- Update the look and functionality of multiple forms at once

  When the design of forms are based on a template, updating the design of those forms is simply a matter of publishing a new version of the template. The changes will be automatically reflected in any form using the template.

- Prevent Form Builders from introducing inconsistencies into the design of a form

  When designing a template, you can use Extension Points to control what parts of the template the Form Builders can edit and add content to. This prevents them from making changes that should be made within the template, ensuring consistency across dozens (or even hundreds) of forms.

- Design templates and forms simultaneously

  Templates and forms can be developed independently and simultaneously. This improves the overall workflow, as Template Configurators and Form Builders won't cause any conflicts as they will work on different resources - a template and a form..

- Work faster with a fully-fledged starting point

  In Maestro, all templates must be based on a template that we've bundled with the software. These templates are feature-rich starting points that allow you to quickly implement styles and functionality specific to your organization, rather than having to repeatedly reinvent the wheel.

- Save time (and money) with less testing

  You can't avoid testing your templates and forms entirely, but because your templates will be based on one of our templates, you'll benefit from the testing we do in-house.

In Maestro, all forms must be based on a template. Templates are responsible for the look and feel of forms. By using templates, you can define a source of truth of how an organization's forms should look. Then, if the organization wants to update the look and feel of its forms, the changes only need to be made in the template, rather than in each form

Most Maestro templates have the following common elements:

- Form Header
- Navigator
- Function Bar
- Wizard Navigation Bar
- Form Footer

These elements can be present or positioned differently in various template. For more information, see common template elements.

Templates are created in the same Maestro editor that's used for creating forms. The same components are available via the **Palette** pane, but you'll mostly use different components in a template than you would in a form. Typically, templates contain components that you want to appear consistently across forms. They don't contain content that will most likely be different across forms, such as form fields or Page components.

The table below provides examples of components that typically appear in templates, forms, or both templates and forms. Maestro allows you to put any component in both templates and forms though, so these examples are just guidelines, rather than strict enforcements.

Templates Only	Forms Only	Templates & Forms
Template Controller	Page Controller	Section
Narrator Controller	Page Group	Button
Progress Bar	Page	Image
JavaScript Library	Text Field	Text Display
Web Font	Dropdown	Block

Within a template, it's also possible to define Theme Options and Item Properties, which can then be used to configure a template without diving into the underlying code. These features are intertwined with brands.

When creating a template, a Template Designer:

1. Defines the structure that surrounds a form, such as, header, footer, navigation, and so on
2. Defines what Theme Options and Item Properties can be configured at the brand-level
3. Adds and remove shared styles
4. Sets extension points
5. Creates brands. This is an optional step.
6. Publishes the template to a library

A Template Designer uses the Maestro editor to:

- Create templates
- Explore component's styles, for example, to understand an effective style of a component
- Change components' styles in Styles panel so they are available to all components
- Override shared components that are used in templates
- Exclude or include components from and in the Palette pane to customize it for Form Builders

However, more often than not, a Template Designer uses CSS directly to create a template and change its styles as needed.

Maestro comes with several ready-to-use templates:

- "Avalon Template Overview" on page 974
- "Maguire Template Overview" on page 982

Next, learn how to change default properties.

Common Template Elements

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A Maestro template has several distinct areas or elements, which mostly managed (created, edited, or updated) by a Template Designer, who also decides which areas of the form has access to and which areas can be edited using the Form Options. functionality in Maestro.

Forms always has access to the content area, which is the form main editable area and it is where all components are added. Forms can also have access to specific dialogs and modal pages. In rare cases, the form can have access to other elements of the form, but this is unlikely.

Most templates have the following common elements:

Form Header is at the top of the form. The header displays the logo and the reference code that is used when a user returns later to complete the form. It is common for the form header to be configured in the template NOT the form.

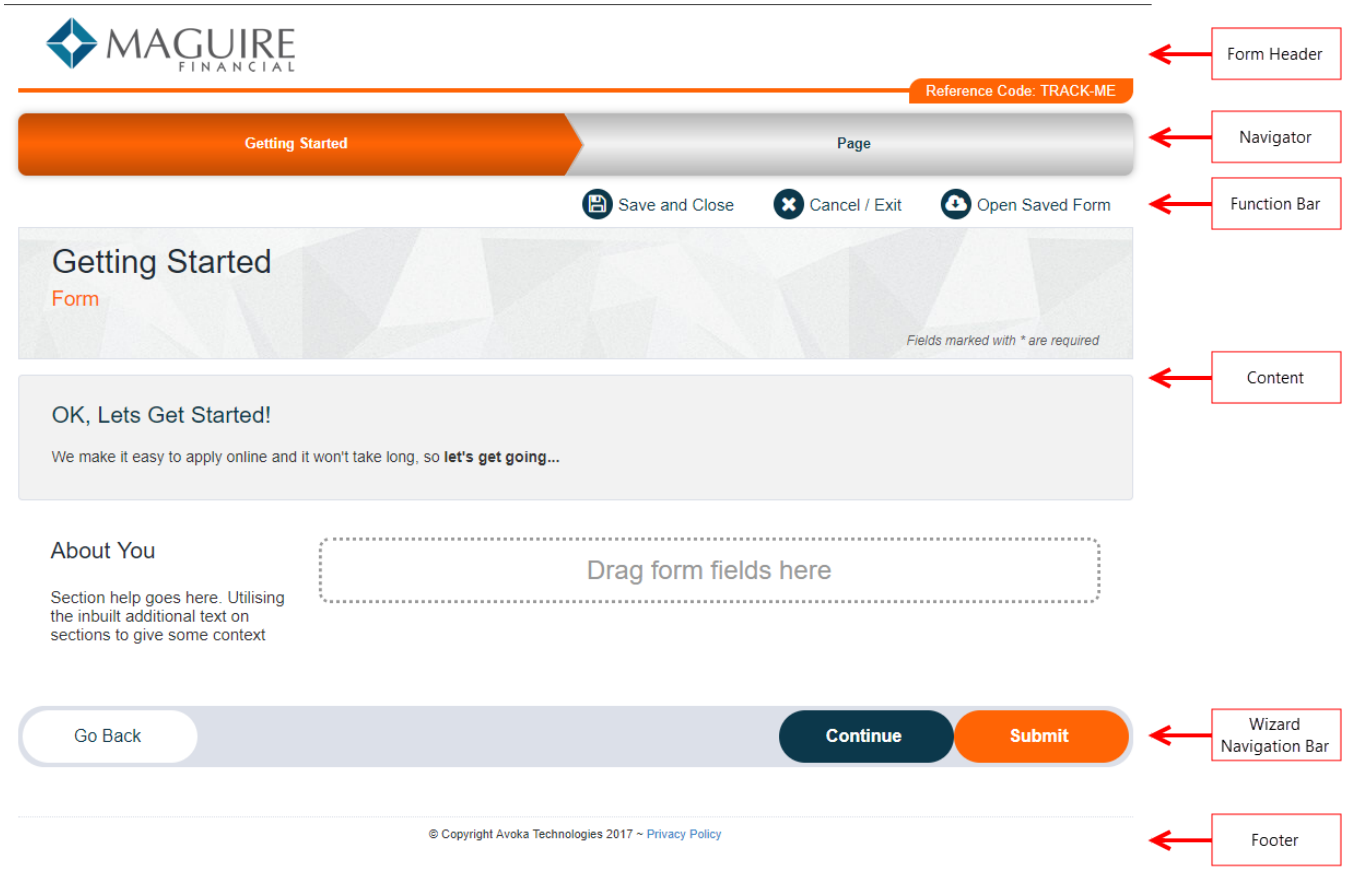
Navigator displays all the pages in the form and allows easy navigation between pages. The Form Builder can have access to make basic changes to the navigator such as edit page names and create page groups. However, the look and feel of the navigator is solely the responsibility of the Template Designer.

Function Bar contains buttons to save and close, cancel or exit and open a saved form. The Function Bar is exclusively managed and configured by the Template Designer, however, the Form Builder usually can enable or disable buttons that are present in the function bar.

Wizard Navigation Bar is at the bottom of the form and contains the navigation buttons to allow users to move from page to page, as well as submit the form when finished. The wizard navigation bar is exclusively configured and managed by the Template Designer.

Form Footer is at the bottom of a form and holds some specific details such as copyright. The Form Builder doesn't usually have access to this area.

The image below shows typical locations of the above mentioned elements.



Next, learn how to create a template.

Avalon Template Overview

MaestroThe UI design product. + Avalon TemplateThis topic is related to the Avalon Template used in Maestro. | Template Designer | 21.11 This feature was introduced in 21.11.

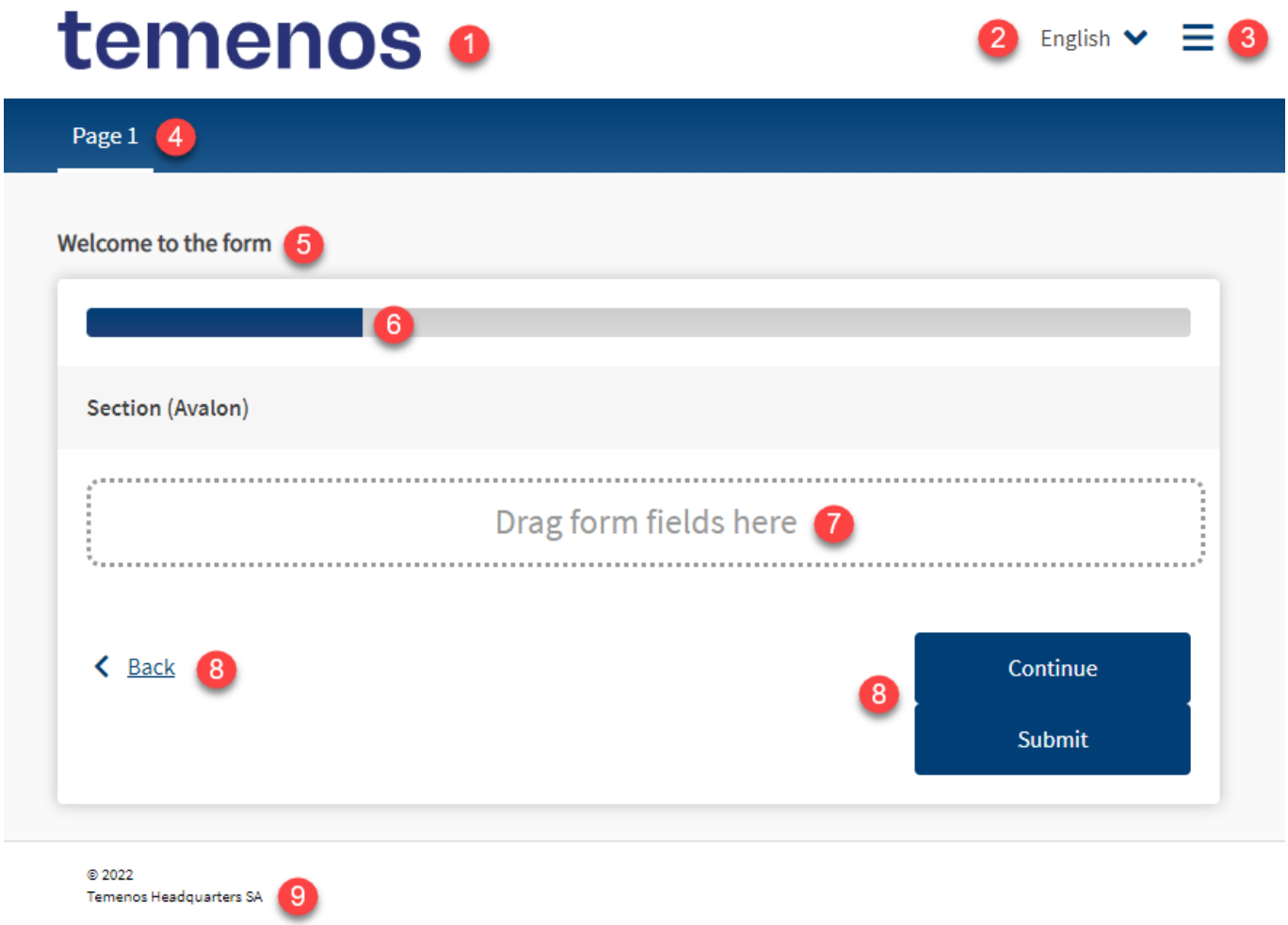
Maestro comes now with a new template, the Avalon template, which brings a more modern look and feel and encapsulates common design patterns, as well as helps speed development time frames. The template addresses growing demands in more contemporary UI look and feel and new UX elements, as well as to allow developers to design and build new forms using Temenos approved look and feel elements and styles.

NOTE

The Maguire Template has been removed from the Journey Manager installer, so you should use the Avalon template instead.

Maestro uses templates to standardize appearance for your forms. A template you select, when creating a form, makes a significant difference to how your form looks, so you must always use a template. Templates are created and maintained by a Template Designer. When updates are made to a template, the changes will be applied automatically to all forms using that template, once it has been published.

The standard Avalon template is show below.



The template has a well-defined structure with components and their sub-components laid in a particular order. It's important to understand the recommended structure of the template before you endeavor to change this structure to one that suits your requirements.

Template Elements

The Avalon template includes the following elements:

1. Header
2. Language selector
3. Menu
4. Current Page Title , for example, "Page 1"
5. Current Page Subtitle , for example, "Welcome to the form"
6. Progress Bar
7. Page Content - this is a placeholder where a Form Builder is allowed to add new form elements, such as components

8. Navigation Block
9. Footer

Template-Specific Components

When you build forms based on the Avalon template, the forms have access to the Avalon-specific components under the following categories:

Avalon Components

- Landing Page Image
- Modal Page Logo
- Avalon Info Dialog
- Progress bar
- Session Timeout Controller

Containers

- Section (Avalon)

Dialogs

- Dialog (Avalon)

Display Fields

- Form Footer (Avalon)

Navigation

- Navigator (Avalon)
- Page Group (Avalon)
- Page Controller (Avalon)
- Page (Avalon)

> NOTE
>
> You can find these components by expanding each category in the Palette pane of the Maestro editor.

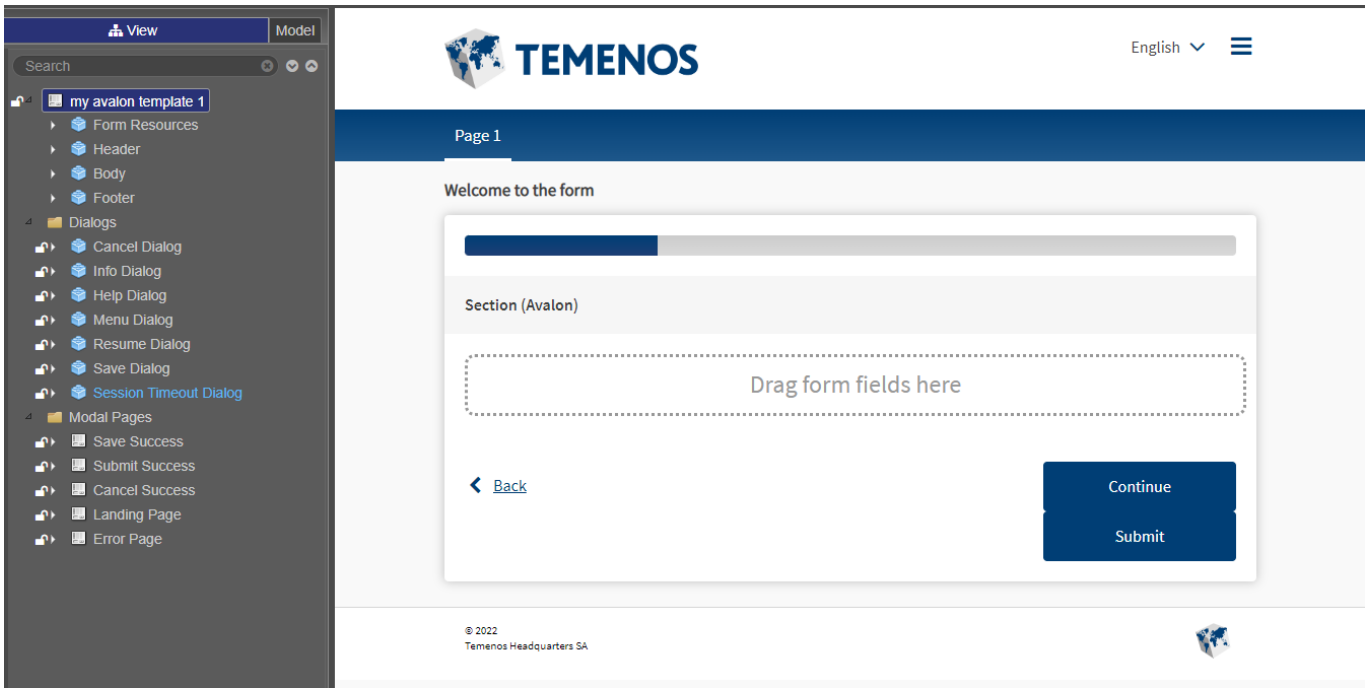
Next, learn how to create a form using a template.

Avalon Template Structure

MaestroThe UI design product. + Avalon TemplateThis topic is related to the Avalon Template used in Maestro. | Template Designer | 21.11 This feature was introduced in 21.11.

A new template that you create based on the default Avalon Template has a well-defined structure. Let's examine its structure to understand how it works and how you can extend it.

When you open the new template in the Maestro editor, you should see something like this.



The standard Avalon template has pre-configured components arranged in a particular order. Components come with some elements so you can use them out of the box, or you can add, remove, modify and change the order of them as needed.

The new template can be split into the following groups:

- Template's content
- Dialogs
- Modal Pages

Let's look at each groups in detail.

Template's Components

The main Avalon template's components and their children components are listed below:

- Form Resources
  - Fonts
    - Source Sans Pro Light
    - Source Sans Pro Regular
    - Source Sans Pro Semi Bold
    - Source Sans Pro Bold
  - Scripts
    -
  - Template Configuration
    - Session Timeout Controller
  - Form Configuration
    - Language Selection
      - Locale
  - Style Master Content
- Header
  - Top Header
  - Sub Header
- Body
  - Error Block
  - Subtitle Block
    - Content
  - Content
    - Progress
      - Progress Bar
    - Form
      - Page 1
        - Section (Avalon)
      - Page 2
      - Page 3
      - Page 4
    - Navigation
      - Left
        - Back
      - Right
        - Continue
        - Submit

- Footer
  - Content
    - Footer Text
    - Footer Icons

NOTE

The template has an extension point set at the root, so you may need to move it to limit the scope where content can be modified.

The Body > Content is where you may be doing most of your changes. If this is the case, it is a good idea to set the extension point there

Dialogs

The Avalon template comes with the following Dialogs:

- Cancel Dialog
  - Dialog Body
    - Dialog Content
      - Cancel Application
    - Dialog Footer
      - Cancel
      - Confirm
    - Dialog Header
    - Close
- Info Dialog
- Help Dialog
- Menu Dialog
- Resume Dialog
- Save Dialog
- Session Timeout Dialog

NOTE

Each Dialog has an extension point set at the root, so you may need to move it to limit the scope where content can be modified.

Modal Pages

The Avalon template comes with the following Modal Pages:

- Save Success
  - Header
    - Top Header
      - Logo
    - Sub Header
  - Body
    - Save Page Content
      - Application Saved
      - Share Form
  - Footer
    - Content
      - Footer Text
      - Footer Icons
- Submit Success
  - Header
  - Body
  - Footer
- Cancel Success
- Landing Page
- Error Page

> NOTE
>
> Each Modal Page has an extension point set at the root, so you may need to move it to limit the scope where content can be modified.

Next, learn about template JSON definition.

Avalon Template JSON Definition

MaestroThe UI design product. + Avalon TemplateThis topic is related to the Avalon Template used in Maestro. | Template Designer | 22.04 This feature was updated in 22.04.

The Avalon Template has a well-defined structure, which is defined in a JSON[1] file. Every time you modify the template, Maestro persists these changes by updating the JSON file. You can download this file and view it in a text editor.

Let's look at the JSON file in detail.

blockInfo

brandOptions

brands

breakpoints

childExtensions

Next, learn how to create a temp

late.

[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.

Maguire Template Overview

MaestroThe UI design product. + Maguire TemplateThis topic is related to the Maguire Template used in Maestro. | Template Designer | All versions This feature is related to all versions.

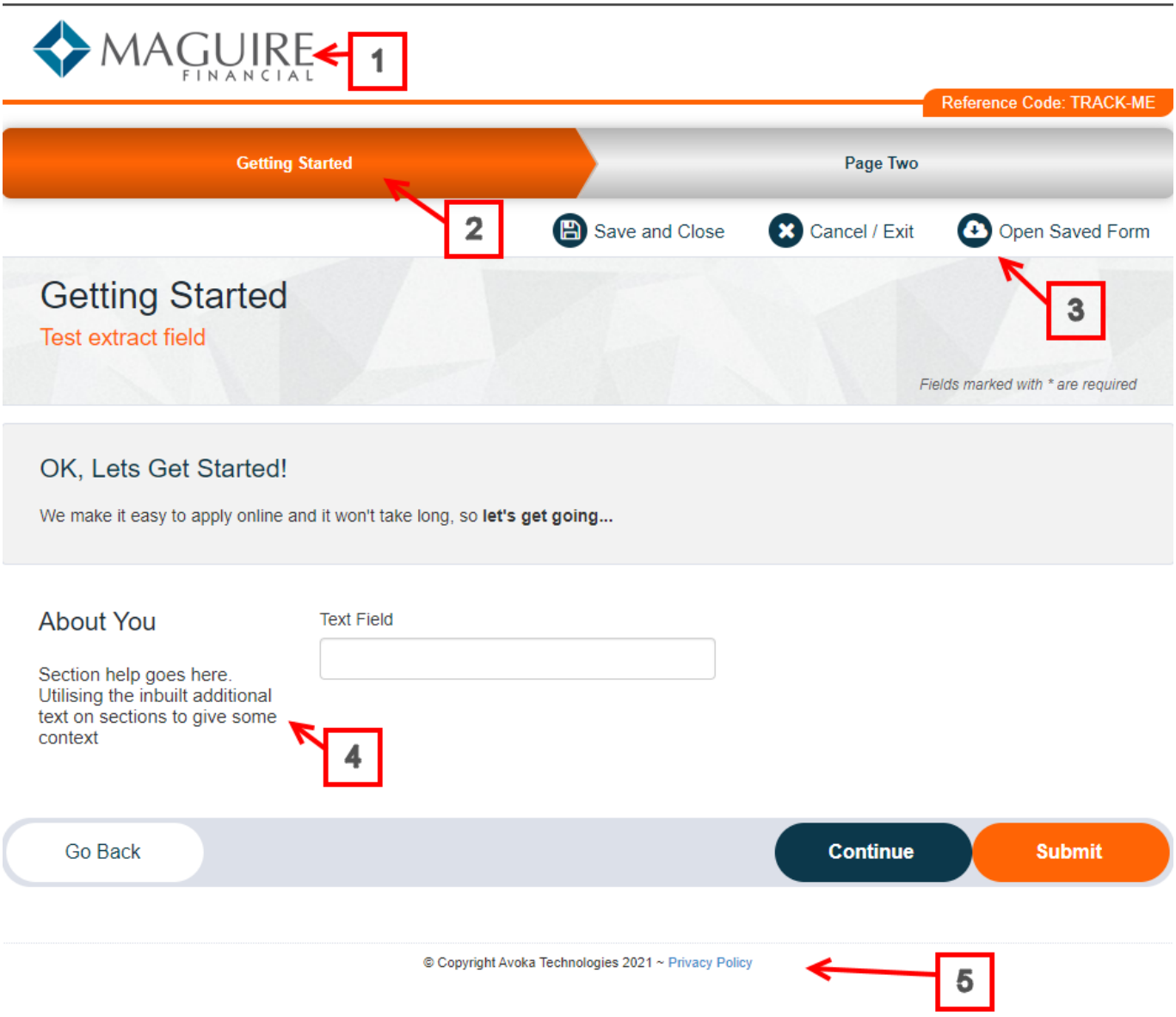
Maestro provides several templates, including the Maguire Template, which allow you to standardize an appearance of application forms and receipts you develop. Templates are created and maintained by a Template Designer. When updates are made to a template, the changes will be applied automatically to all forms using that template, once it has been published. The Maguire template can't be edited and should only be used to create new templates, not new forms.

The template you choose when creating a form will make a significant difference to how your form looks and behaves., so you always must use a template as a starting point.

The template usually includes the following elements, as visualized below:

- Header (1)
- Navigation bar (2)
- Button bar (3)
- Footer (5)

The only element not managed by the template is the form content itself (4).



NOTE

There is no relationship between the Maquire form space and Maquire template. The Maguire template is used for building forms, whilst the Maguire form space is a staff and customer portal for managing submissions, assisted workflows and so on. The Maguire form space was removed from the installer, | 18.11 This feature was removed in 18.11

Next, learn how to create a form using a template.

Create a Template

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to create templates that accelerates form development process and unifies look and feel of forms, which are based on the templates. Templates are mostly created by Template Designers.

In Maestro, a template is the starting point for all forms. Templates contain the style, brand, colors, and other configurations used by Maestro forms. Each Maestro project contains one or several default templates, which include some default assets and resources. A Template Designer decides what assets and resources belong to each template. However, templates should be used by form developers as a guideline when working on new assets and resources.

New templates are always created based on existing or default templates. Creating a new template from scratch is extremely difficult and time-consuming so it should only be undertaken by an experienced Template Designer. We recommend starting with creating a placeholder template first, which is a simple custom template that contains the correct extension points, but no other styling elements. This allows a Form Builder to start using it right away for form development. Then, the placeholder template is further configured and customized by a Template Designer to follow the branding requirements. This approach allows the Form Builder and Template Designer to work simultaneously and the Form Builder can begin work immediately on the form.

To get a new template and start using it, you should:

1. Create a new template
2. Set an extension point
3. Publish the template to a library
4. Customize the template

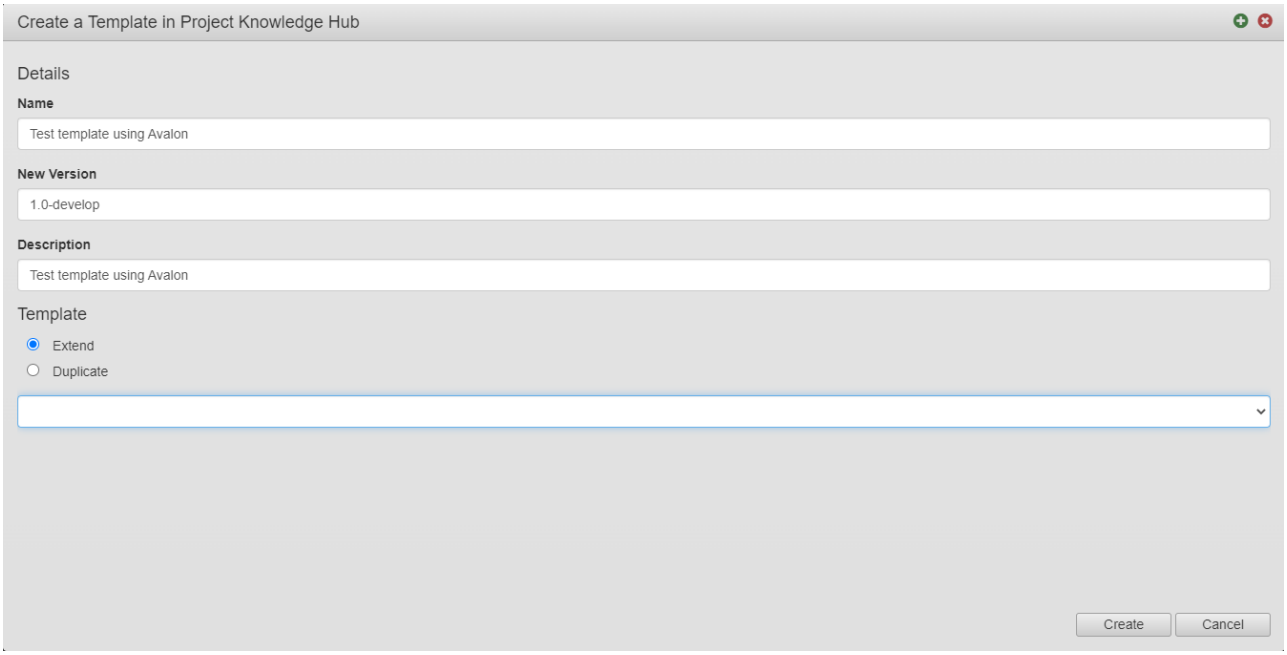
Let's look at each of these steps in more detail.

Create a Template

To create a template:

1. Open a project from the Management Dashboard, for example, your Styles project or your organization's project directory. If you do not have a Styles project, see project configuration.

2. Select Templates and click Create Template.



3. Enter a new template name in the Name field.
4. Change a system generated version in the New Version field to adjust it to your requirements. Maestro uses semantic versioning.
5. Provide a meaningful description in the Description field so it's clear for others what this template is for and what it does.
6. Select a base template from the dropdown list.
7. Select Extend to link the new template to the base template, so if it is changed, these changes will appear in the new template. Since most templates are based on an Temenos approved template, it is a good idea to use the this option.
8. Select Duplicate to copy the base template, so there is no link between the two templates. It means any changes to the base template won't appear in the new template.
9. Click Create.

> NOTE
>
> The template should open in a new tab. If the template doesn't open, verify that it hasn't been blocked by your browser's pop-up blocker.

Set Extension Points

Once you have created your new template, you will need to set an extension point to restrict form's content for a Form Builder to modify.

Publish the Template

After the extension points are added, you need to publish the template so that the Form Builder can create a form based on it. We will call this template the placeholder template. The placeholder template needs to be published to an organization's library so new forms can be created with this template.

When creating new forms, there are several default templates available for selection. Additionally, you will have access to several custom-built templates specific to your business. When selecting a template for a new form, always select one of the custom-built templates, as the default templates can't be edited and should only be used for creating new templates.

Customize the Template

Now that the placeholder template has been published, the Form Builder can create a form and begin development. While the Form Builder is working on the form the Template Designer can further configure and customize the template. At various stages of template development the template can be published when required. Any changes to the template will be seen by the Form Builder as they develop the form.

Next, learn how to publish a template.

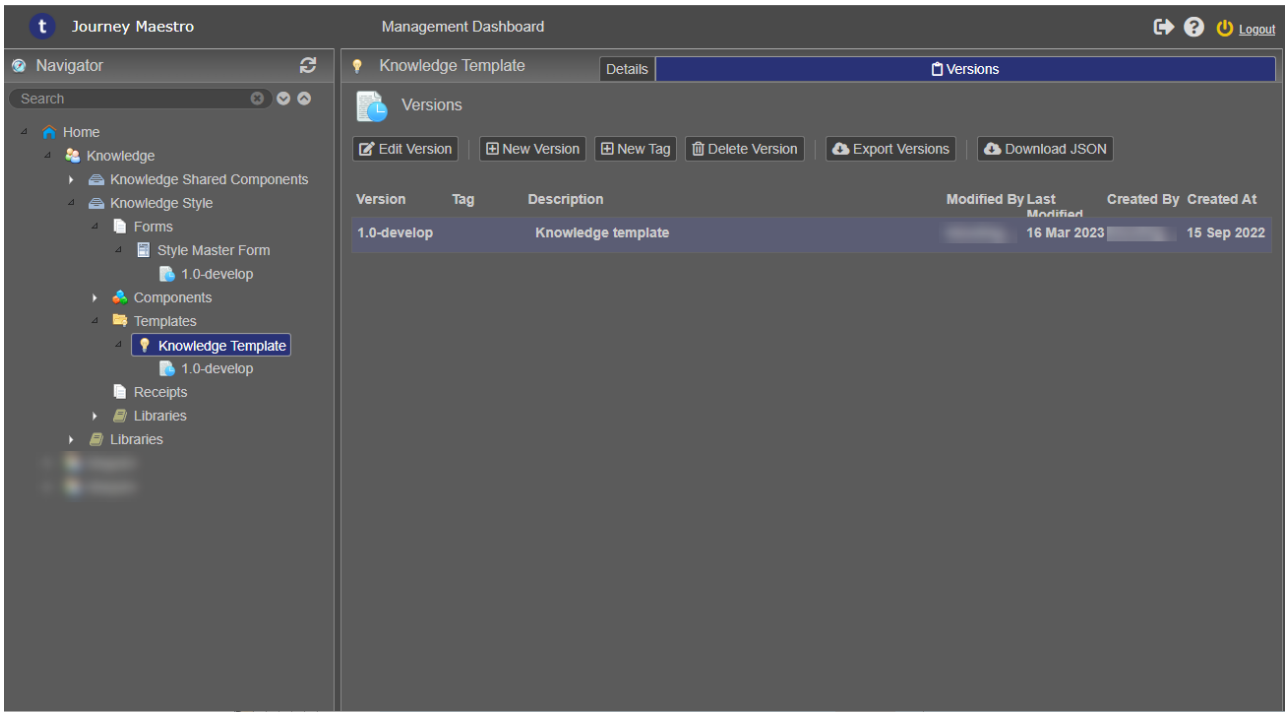
Tag a Template

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

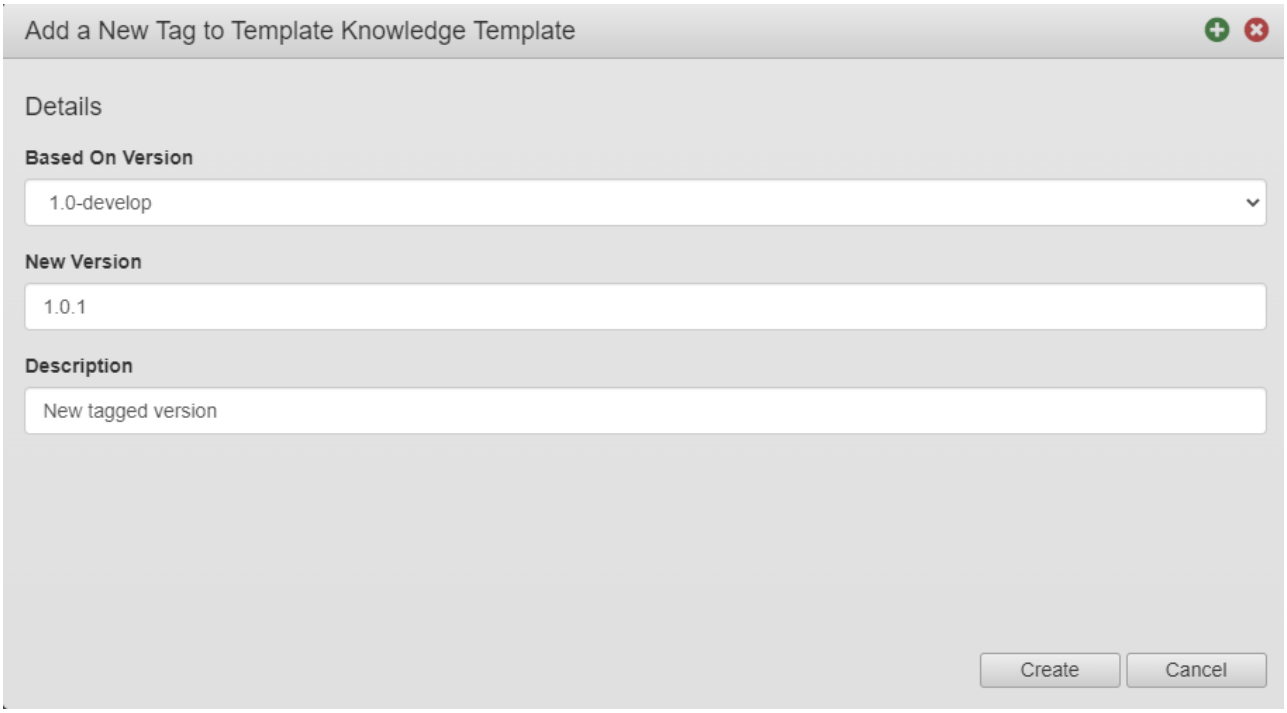
Maestro allows you to create a read-only version of a template by tagging its specific version. After that, the tagged version of the template will remain unchanged forever, so it doesn't permit you to add new components and, if you change any configuration, there is no option to save these changes. You can use this functionality to capture a template snapshot, which is handy when you wish to come back to, or use for application releases and so on.

To tag a template:

1. Open a project from the Management Dashboard.
2. Select a Template you wish to tag.

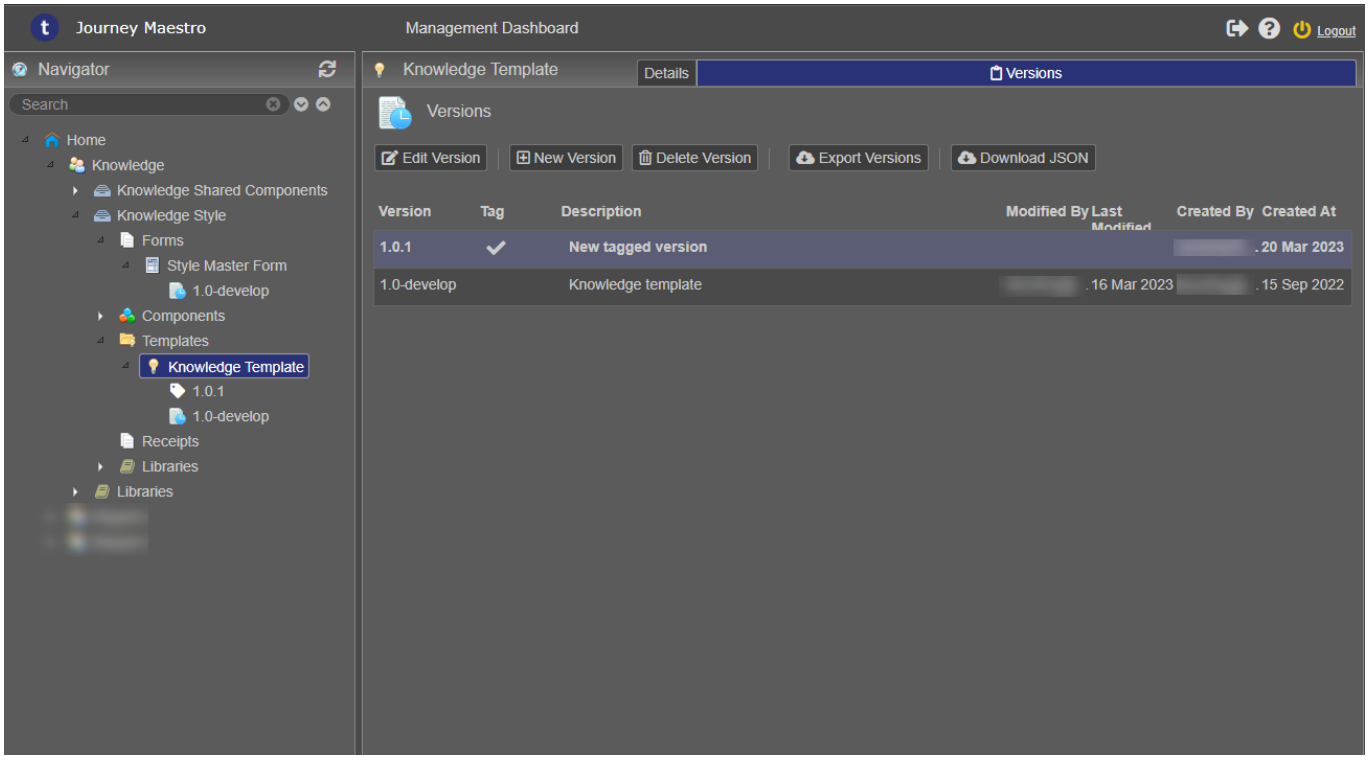


3. Click New Tag to populate the tagged version details.



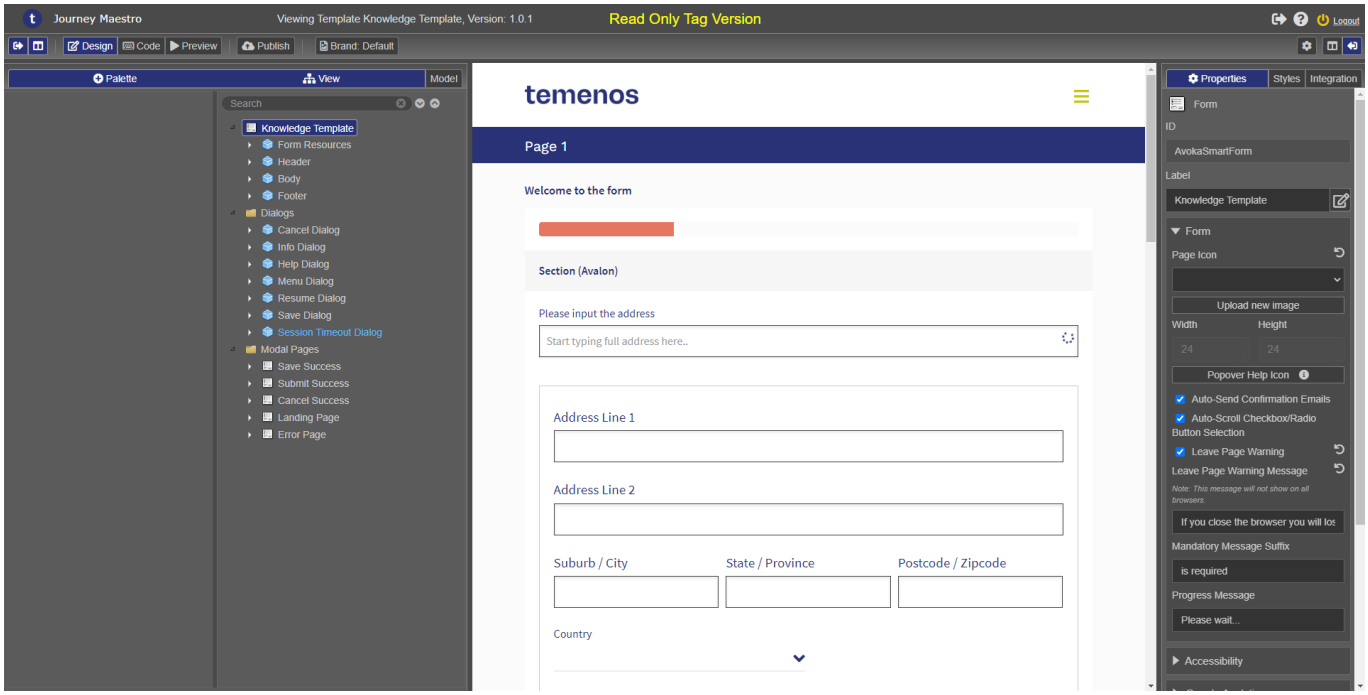
4. Select a version of the template to tag from the Based On Version dropdown list.
5. Enter a template version in the New Version field or leave it as Maestro prefills it for you, which is a recommended option.
6. Provide a meaningful description in the Description field .
7. Click Create.

You can now check the tagged template version in the Navigator pane.



In this example, the tagged version is 1.0.1 and it is marked with the tag sign () in the Navigator pane and with the check sign () in the Tag column.

If you open the tagged version in the Maestro editor, you can see that this template is a read-only.



Next, learn how to publish a template.

Publish a Template

MaestroThe UI design product. | Form BuilderTemplate Designer | 21.11 This feature was updated in 21.11.

Maestro allows you to publish a template to a library so it can be used by different forms. You do this after creating and testing a new template.

To publish a template to a library:

1. Open a template in the Maestro editor.
2. Click Publish.



3. Select a library from the Library dropdown list. The Library Version shows you the available library version.
4. Select the Publish checkbox to include embedded styles, which are defined in the form, so they will be published to the selected library as well.

   NOTE

   This option is not shown if styles are not defined in the form.

5. Select the Delete local copy checkbox to delete the local copy of embedded styles.

   NOTE

   This option is not shown if styles are not defined in the form.

6. Click Publish.

You can check which template version is present in the library, as shown below:



NOTE

If you make any changes to the template after publishing it, you must re-publish it again for the changes to take effect in forms that use it.

Next, learn how to export a template.

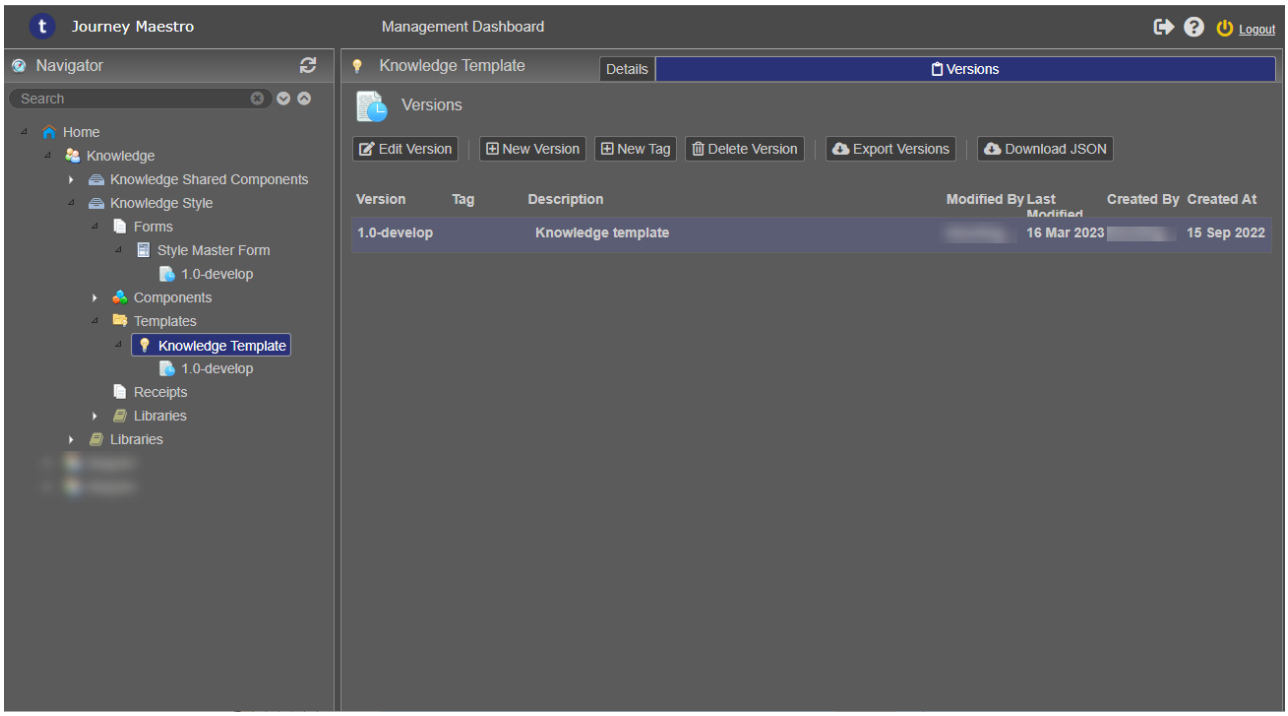
View Templates

MaestroThe UI design product. | Form Builder | 22.10 This feature was updated in 22.10.

Maestro allows you to view and manage templates.

To view the list of templates:

1. Open your project in the Navigator pane and click Templates to view the list of templates created in your project.
2. Select a template in the Templates folder to view available versions of this template.



   The templates are listed in ascending order by template versions. The list includes the following information, which helps you use templates correctly:

   - Version: a semantic version of a template
   - Tag: whether a template version is tagged or not
   - Description: a meaningful description of a template
   - Modified by: who modified a template
   - Last Modified: when a template was modified
   - Created By: who created a template
   - Created At: when a template was created
3. Click Edit Version to open a selected template in the Maestro editor for editing.
4. Click New Version to create a new version of a selected template.

5. Click New Tag to tag a selected template.
6. Click Delete Version to delete the selected template.

> NOTE
>
> Use `Shift` or `Ctrl` key to select multiple templates. | 21.05 This feature was introduced in 21.05.

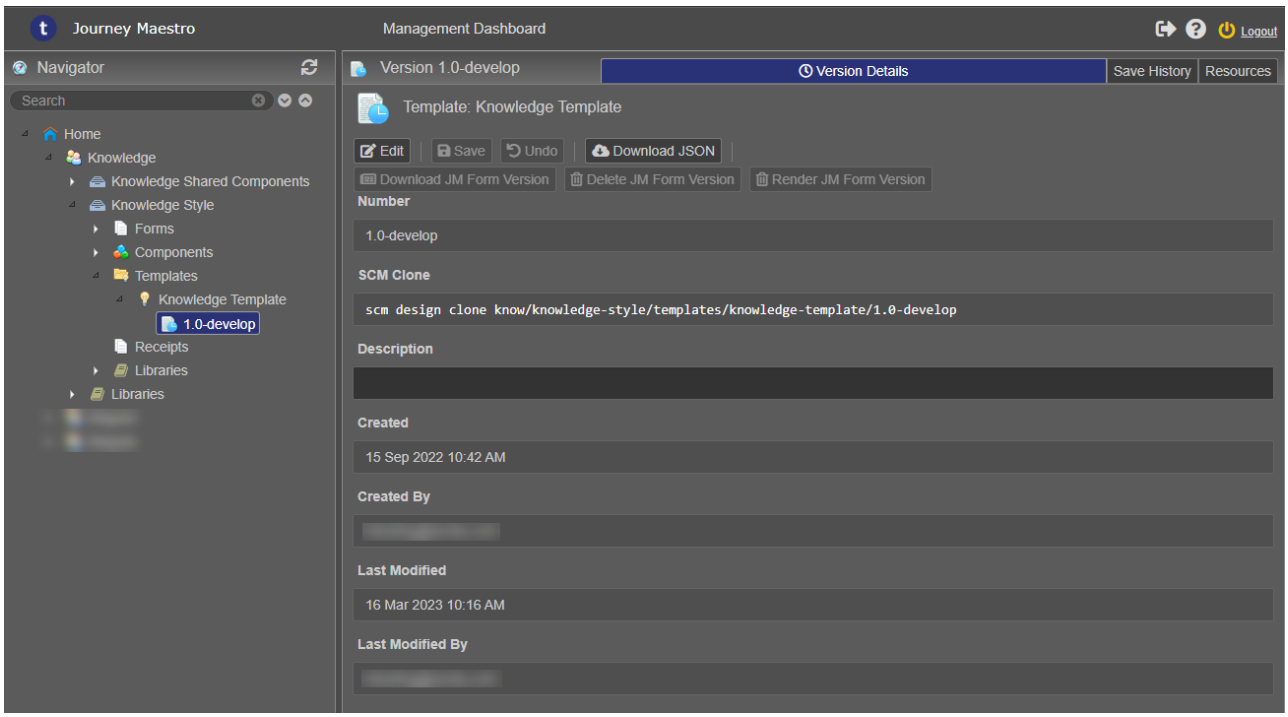
7. Click Export Versions to export the selected templates.

> NOTE
>
> Use `Shift` or `Ctrl` key to select multiple templates. | 21.05 This feature was introduced in 21.05.

8. Click Download JSON to download JSON files of the selected templates.

> NOTE
>
> Use `Shift` or `Ctrl` key to select multiple templates. | 21.05 This feature was introduced in 21.05.

9. Double-click a selected template to view its details.

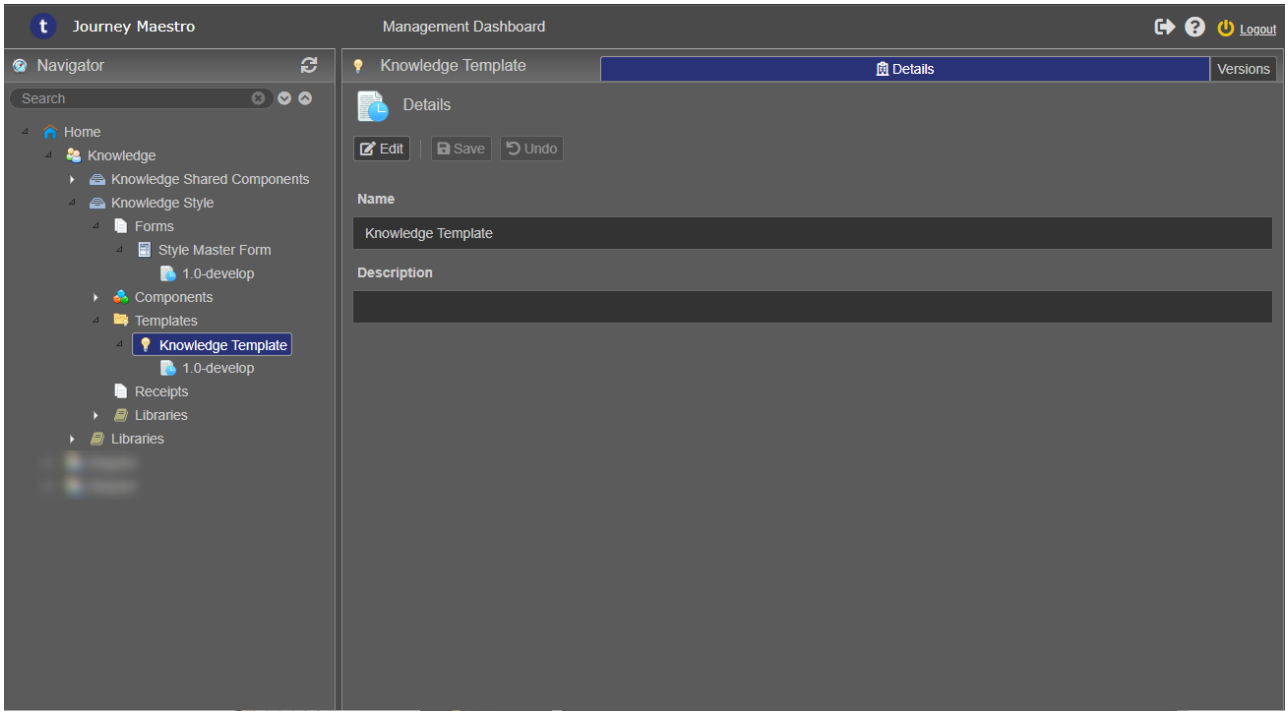


Edit a template description in the Description field and click Save to update the changes

NOTE

You can't update the name and version number of the template.

10. Select the template and click the Details tab to see the template's details.



Edit the template's name and description in the corresponding Name and Description fields and click Save to persist the changes.

Next, learn how to export a template.

Export a Template

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to import and export templates, so you can reuse them across different projects, organizations or environments. This saves you time and effort, as you don't need to create a new template from scratch. This process is also commonly used to migrate templates from one environment to another. You can use this functionality to:

- Rollback one or several templates
- Re-create one or several templates in another Maestro
- Promote one or several template

To export a template:

1. Open a project in the Navigator pane where you want to export a template from and click Templates to view the list of templates created in your project.
2. Select the template you wish to export from the list and click Export Template. Maestro exports the template into a form archive ZIP file, which will be downloaded to your machine.

NOTE

You can also export a template by exporting a library, which contains the template.

Next, learn how to import a template.

Import a Template

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to export and import templates, so you can reuse them across different projects, organizations or environments. This saves you time and effort, as you don't need to create a new template from scratch. This process is also commonly used to migrate templates from one environment to another. You can use this functionality to:

- Rollback one or several templates
- Re-create one or several templates in another Maestro
- Promote one or several template

NOTE

When you import a template to a different project, it is not available to forms within the project until the template is published to a project's library. However, if you import the entire project or the library, the published version of the template is already available to forms.

To import a template:

1. Open a project in the Navigator pane where you want to import a template to and click Templates to view the list of templates created in your project.
2. Click Import Template and choose a template archive ZIP file.

NOTE

You can also import a template by importing a library, which contains the template.

Next, learn how to update default properties of a template.

Update Template Default Properties

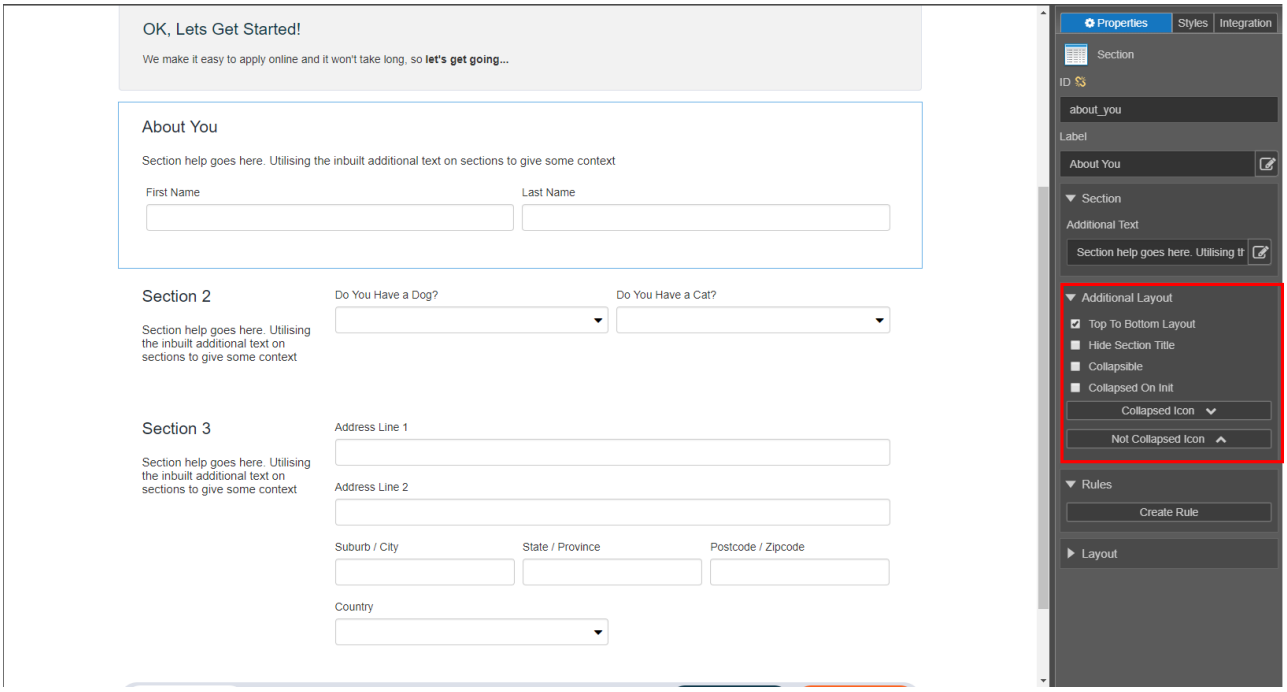
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to change default properties of a template after you have created it, so you can customize certain properties for a component type and it will be reflected in a form that uses that component. For example, you decide that the heading of a section need to appear above the content, rather than to the left, which is the default option.

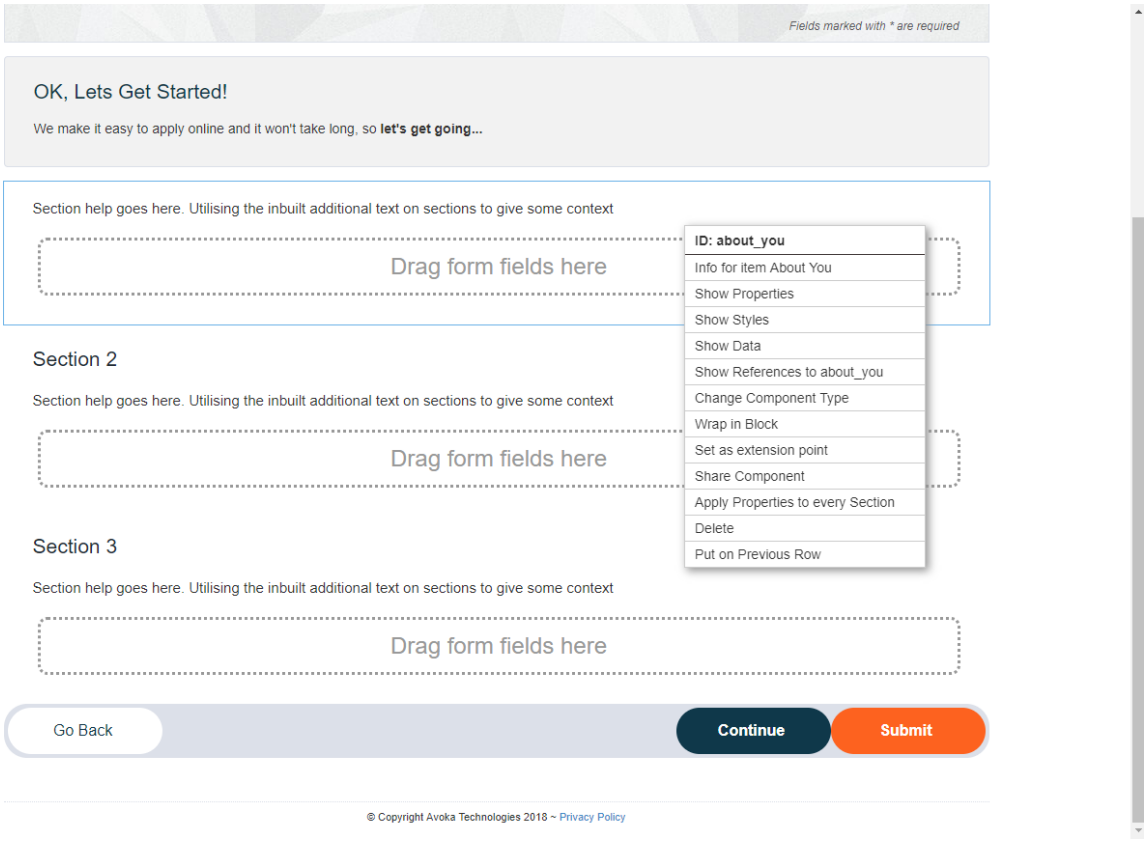
Changing the default properties of a component type should only be done in the template by a Template Designer, because it will be applied to all components of the same type that may affect many forms using this component.

To change the default properties on a component in a template:

1. Open a template in the Maestro editor.
2. Select a component you want to modify, for example, the About You section component.
3. Select the Properties pane to update the default properties of the component. For example, you may change the layout of the section to be Top to Bottom Layout.



4. Right-click the section component with the updated properties and select Apply Properties to every Section.



5. Click Save to update the template.
6. Click Publish to apply the updated component changes to a library so in all forms, based on this template, all sections will be displayed with the Top To Bottom layout.

NOTE

Changes to the component properties will only be applied to new components added to a form. Forms with existing components will not be updated. You will need to manually change the properties for existing components or delete the component and re-add it to the form.

Next, learn how to change a template in a form.

Change a Template of a Form

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to change a template that a form uses. When you create a new form, you have an option to select a template to be used by the form. It is important to choose a correct template at the first place, but, you can always change the template later, if necessary.

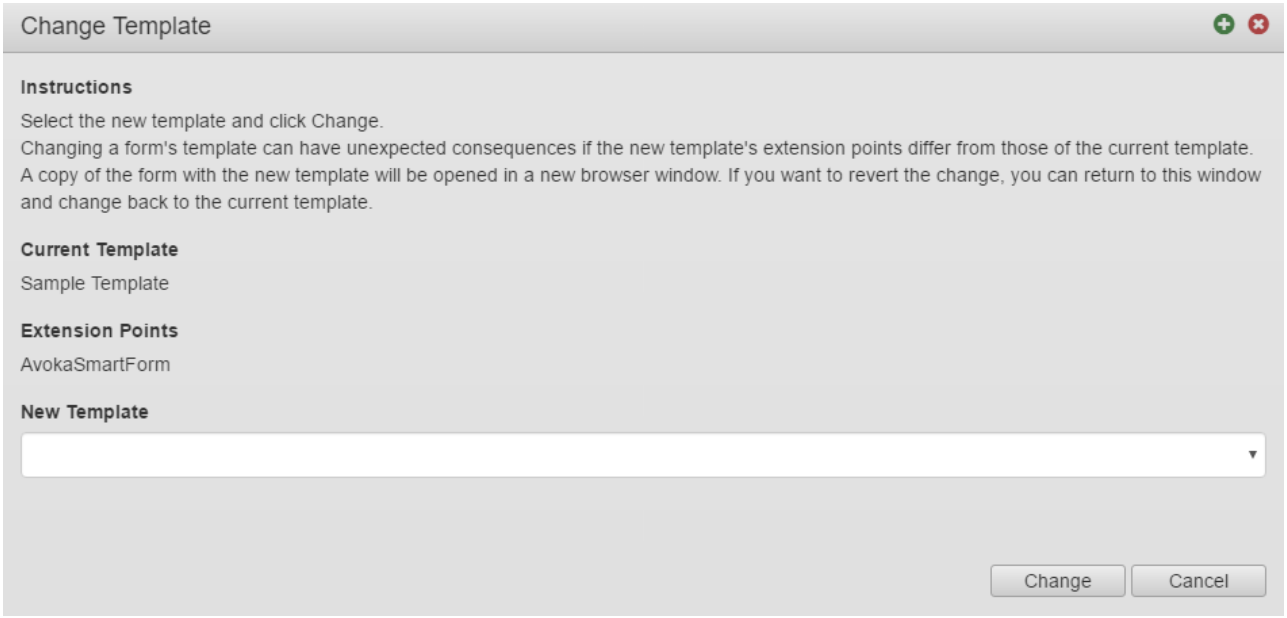
NOTE

We recommended speaking to a Template Designer before changing the template because it can and will affect the form in many ways.

When designing a template, a Template Designer may limit the Form Builder's access to certain areas within the form by using extension points. When the template of the form is changed, the extension points in the new template may be different to those in the original template. If the extension points are different, the Form Builder may lose content within the form.

To change a template of a form:

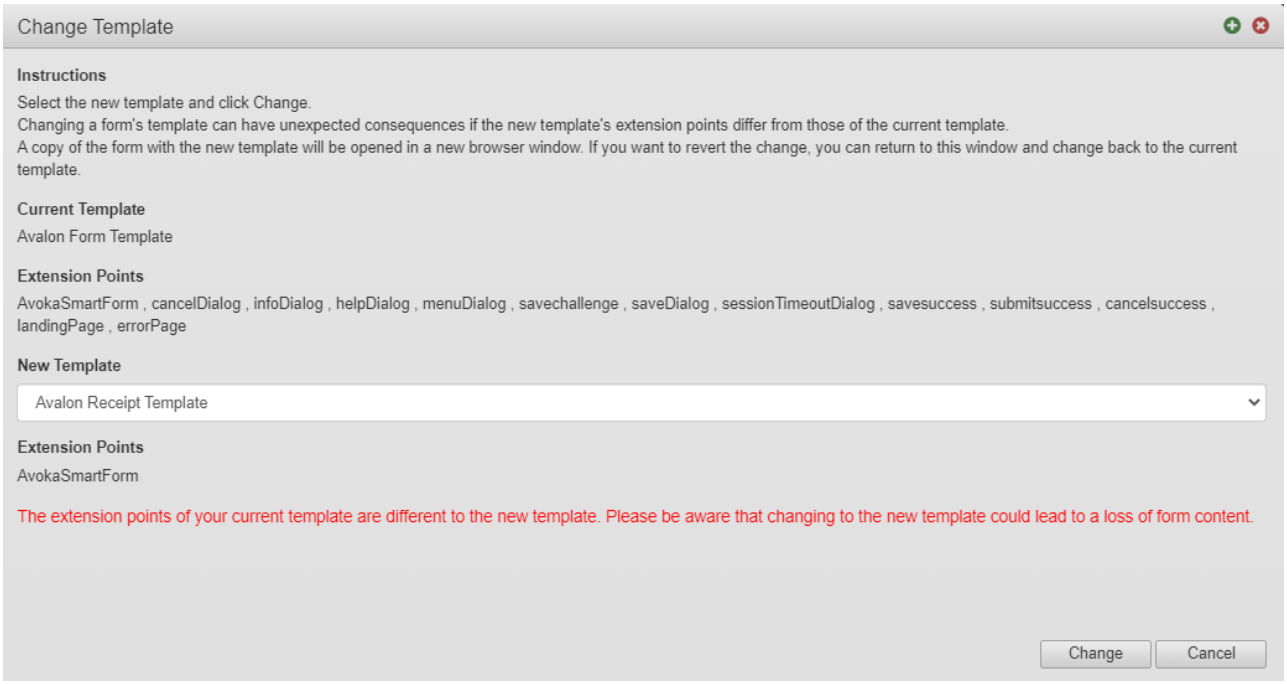
1. Open a form in the Maestro editor.
2. Click Settings Menu in the Toolbar and select Change Template from the dropdown list.



NOTE

The Extension Points section displays the extension points in the current template. `AvokaSmartForm` means that no specific extension points have been set. When no extension points are set, the Form Builder has access to all areas of the form. Most of the time, the template will have extension points set to limit the Form Builder's access and prevent changes to company branding or style guides.

3. Select a new template from the New Template dropdown list. If you select a new template, which has different extension points to the current template, the following warning message is shown. It also lists the extension points in the new template. In this example, the template has an extension point set on the content, which is the main area of the form, and two dialogs, which are the save confirm and save challenge.



4. Click Change. The form using the new template will open in a new browser tab, so you can view it and check if you are happy with the change. This gives you the following options:
   - If you decide to keep the form with the new template, you can close the form in the original browser tab and continue working with the form in the new browser tab.
   - If you don't want to keep the form with the new template, close the new browser tab and return to the form in the original browser tab where you can revert the form to the original template.

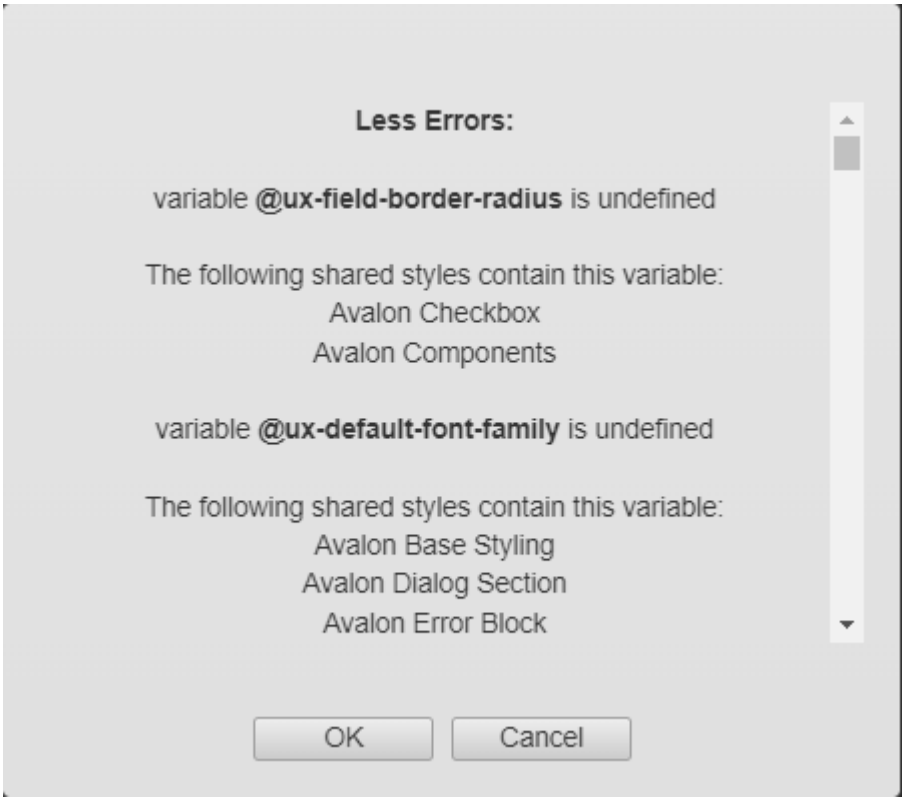
NOTE

The Change Template dialog in the original browser tab now displays a message explaining what has happened and what to do next.



Click Back to return to the original template. The form will again open in a new window with the selected template applied. You can close the form in the original browser tab and continue working with the form in the new browser tab.

5. If there are any major issues, the following windows is displayed with the detailed description of errors.



6. Click OK to continue with changing the template or click Cancel to dismiss the template change. Alternatively, you can close the new browser tab and return to the form in the original browser tab.

Next, learn how to .

Brands Overview

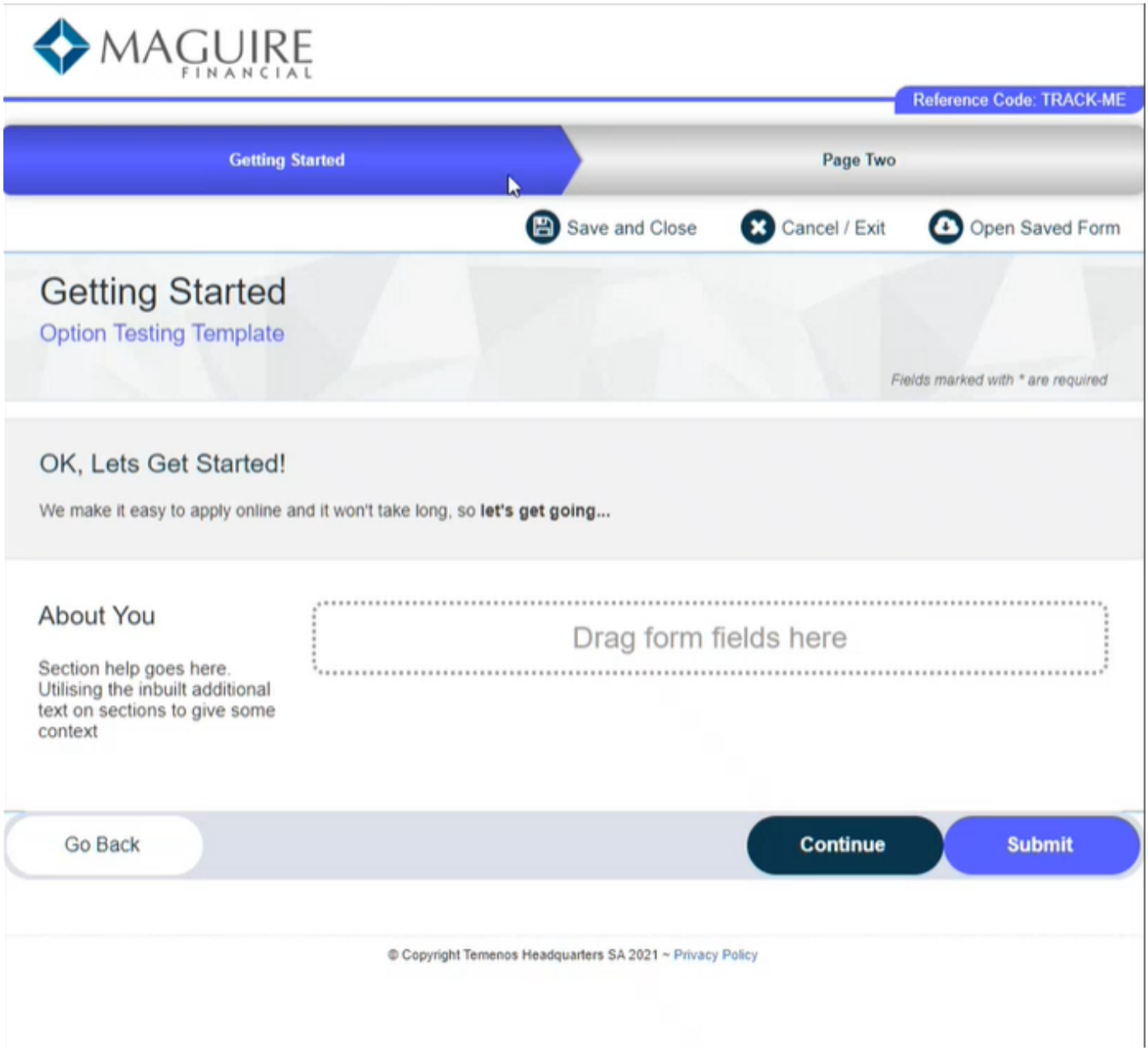
MaestroThe UI design product. | Template Designer ▪ Form Builder | 21.11 This feature was updated in 21.11.

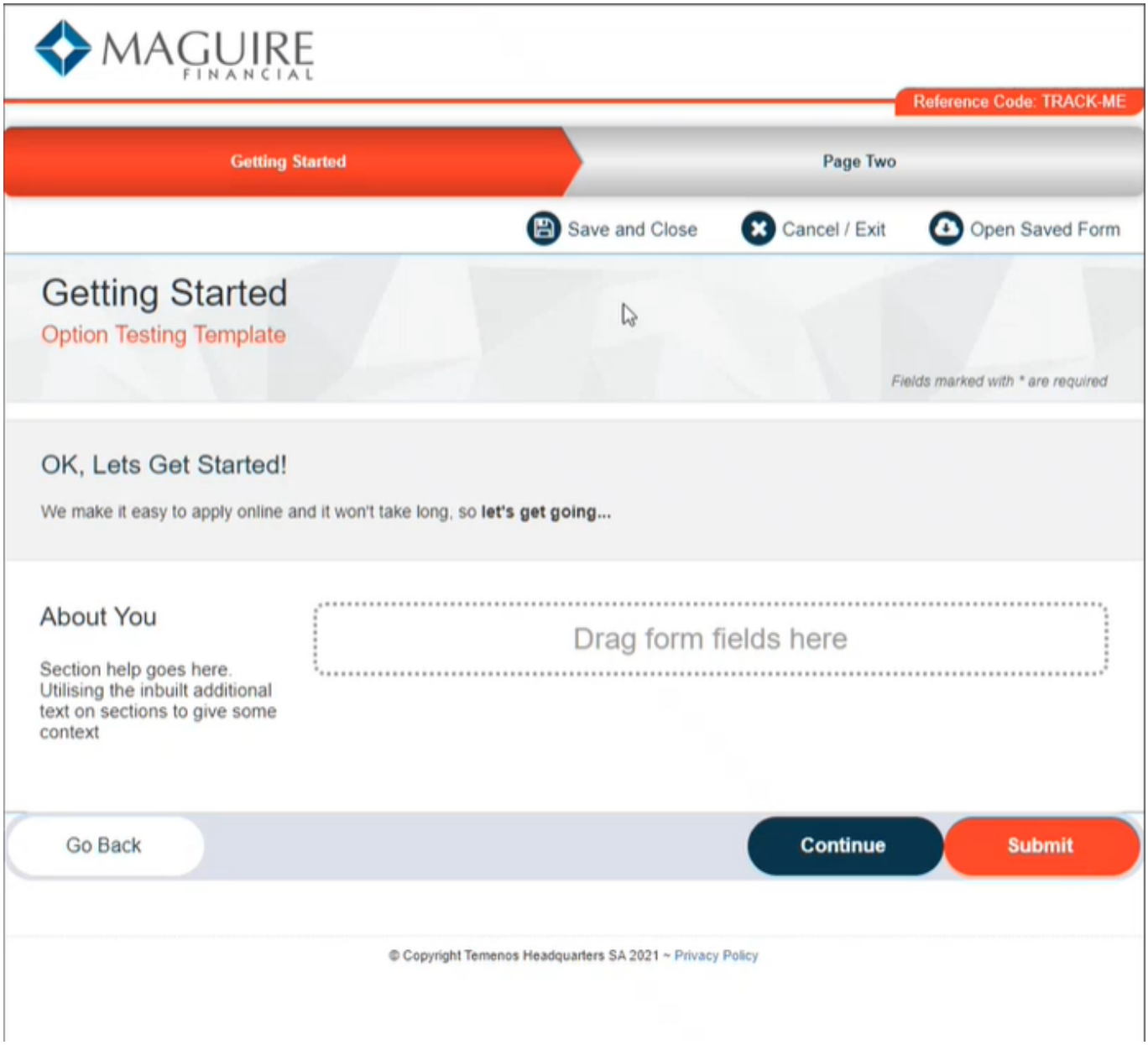
When an organization creates a template, they often want to reuse the structure of the template while still being able to adjust many of its attributes, such as background colors and button styles. This is commonly done for the sake of white-labelling, so an organization can create configurable copies of its templates to eliminate the need of creating a new template every time they enter into a white-labeling deal. This means that the same template can be used for different companies.

To streamline this process, Maestro allows you to use a brand, which is a variation of a template.

A brand defines the physical appearance of a Maestro form and is stored within a template. Brands control colors, fonts, and other aspects of a form. Brands are created in the template by the Template Designer. Brands ensure that all the forms have consistent colors, fonts, and other design elements. If a change to a brand is required, for instance a new color, all that is needed is a modification to the brand and a re-publish of the form – each form does not need to be modified individually.

For example, you may want to have a template defining the same form structure, but with different colors, heights, and widths configured in several brands. This is illustrated in the example below.


MAGUIRE
FINANCIAL
Reference Code: TRACK-ME
Getting Started
Page Two
Save and Close
Cancel / Exit
Open Saved Form
Getting Started
Option Testing Template
Fields marked with * are required
OK, Lets 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 additional text on sections to give some context
Drag form fields here
Go Back
Continue
Submit
© Copyright Temenos Headquarters SA 2021 ~ Privacy Policy



Every Maestro template has at least one brand, named Default. Even if you don't plan to have multiple brands, you still need to use the Default brand to configure the Theme Options and Item Properties within the template. You can't rename or delete the Default brand.

You can create as many brands as required and then apply a brand to a form as you work on. When a brand is not needed any longer, you can remove it. It's recommended that only a Template Designer manage templates.

A brand shares the same underlying structure as the original template, meaning all of the components in the View pane are the same, but the properties and styles can be adjusted on a per-brand basis.

When you edit a form with multiple brands in the Maestro editor, you can see extra options, such as the Brand button where you can switch a brand and the Build with Options where you can choose which brand to use while generating a TM Form Version file.

When creating a brand, a Template Designer will:

1. Configure Theme Options
2. Configure Item Properties

Maestro allows you to use design-time and runtime branding in your forms.

Design-time branding

Design-time branding is when you create a form based on a certain brand available in your template, then build the form using that brand, and, finally, deploy the form to Manager where it is rendered as per the brand it uses.

Since Maestro 21.11, you can generate multiple form versions of a form using several brands. However, these form version files need to be imported separately as they will either be going to the same Manager instance in different organizations or form spaces or going to different Manager instances.

Runtime branding

Runtime branding is a new powerful concept in branding, which is available starting with Maestro 21.11. Runtime branding is based on JSON configuration file that specifies the details of a brand that contains images, CSS[1] content, translation setting, brand properties that can be referenced by rules. The runtime branding is made up of several blocks, such as `Form.updateImageByName()`, `Form.updateCssByName()`, `Translation.fetch()`, which a developer can use (or call) individually based on their requirements. Alternatively, they can use `Form.updateBranding()`, which when used with a specified JSON structure can handle the above functionality while just specifying the JSON file. For more information, see the Maestro API reference documentation.

---

[1]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.

When you create a form, you define your brand in a JSON file, which you store in Manager. Manager pass this file into the form using various methods, such as the form prefill option, a Manager service call, a HTTP GET / POST, so the form is rendered accordingly. If you need to change something in a brand, you don't need to change anything in the form in Maestro. Instead, you simply update the JSON file in Manager so the form picks up the changes from there.

Runtime branding uses the enhanced form build process and the new framework as well as the Runtime CSS Resource and Bundle Resource Library components. The Bundle Resource Library adds resources, which the form has access to, into the FAR file, so you can add JavaScript, CSS and images that can be used at runtime without needing to first include them in the form as a direct resource that is used. There is also the ability to specify images to use a URL so they can be picked up from a portal resource.

Next, learn how to create a brand.

Create a Brand

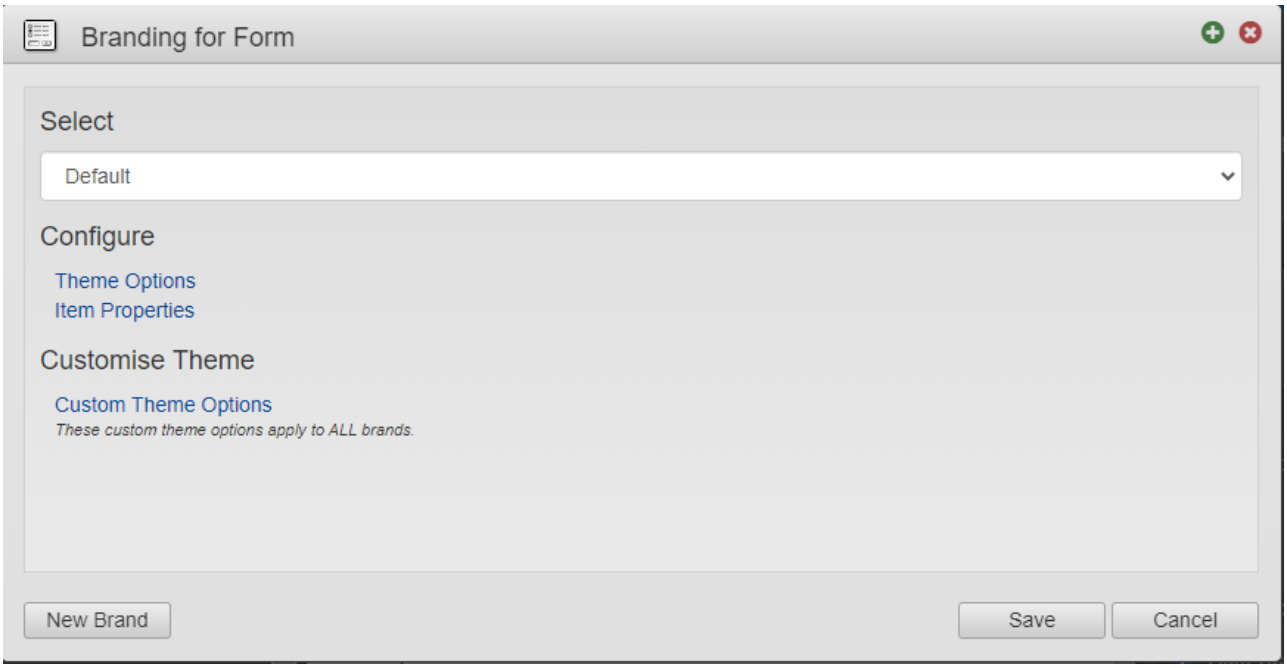
MaestroThe UI design product. | Template Designer ▪ Form Builder | 21.11 This feature was updated in 21.11.

Maestro allows you to create a new brand in a template, so you can leverage on common structure, styles and other form configuration while building a form. You can't create new brands in a form. Usually, a Template Designer creates brands in a template used by your forms.

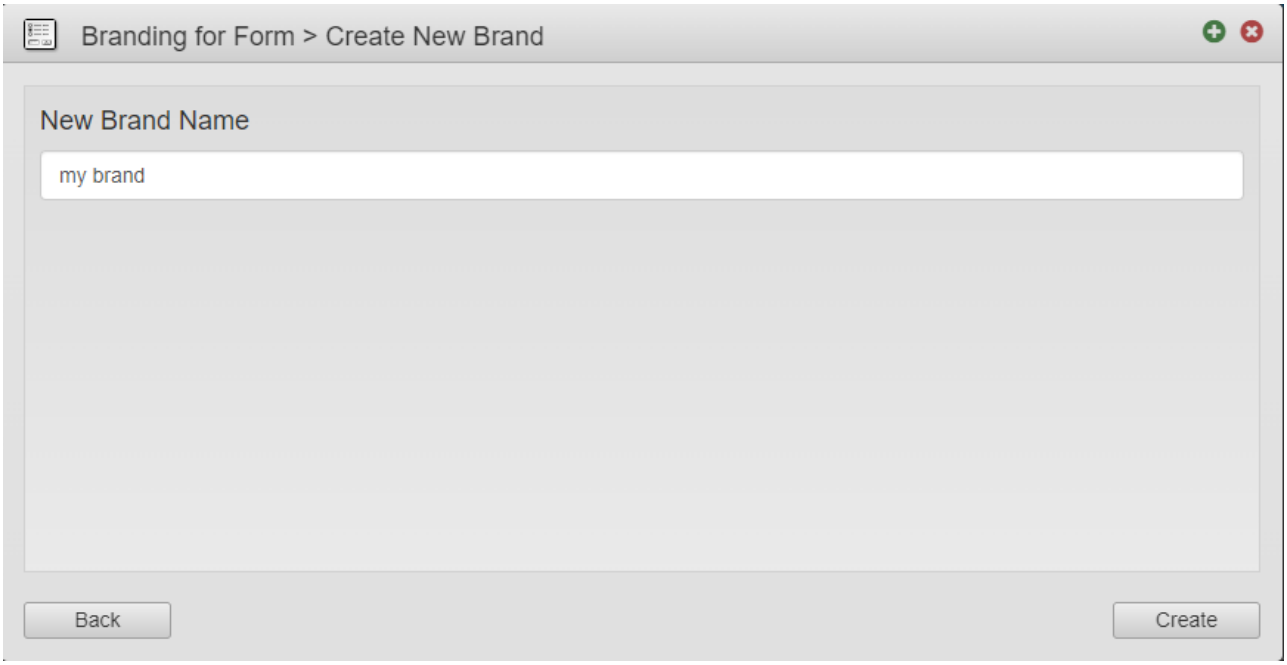
If a template has more than one brand, the Brand button displays in the Maestro Editor when editing a form. The Brand button allows switching between the available brands that were created and configured in the selected template. The Brand button also displays the name of the brand that is currently being used by the form.

To create a brand in a template:

1. Open a template in the Maestro editor.
2. Click Brand in the Maestro toolbar.



3. Click New Brand.



4. Enter a name for the brand in the **New Brand Name** text field.
5. Click **Create**.
6. Click Theme Options to configure brand's theme options.
7. Click Item Properties to configure brand's properties.
8. Click Custom Theme Options to add or modify theme options applicable to all brands.
9. Click Save.

Next, learn how to delete a brand.

Delete a Brand

MaestroThe UI design product. | Template Designer ▪ Form Builder | 21.11 This feature was updated in 21.11.

Maestro allows you to delete a brand from a template, so you can maintain integrity of templates by removing brands you don't need any more. Forms, which use a deleted brand, still work - there just won't be an option to build them with that brand any more. If you use the offline build, it only builds the brands that actually exist, even if other ones are requested.

Usually, a Template Designer deletes brands in a template used by your forms.

To delete a brand from a template:

1. Open a template in the Maestro editor.
2. Click Brand in the Maestro toolbar.
3. Select a brand you want to delete from the **Select** dropdown.
4. Click **Delete Brand** and confirm it.

Next, learn how to apply a brand to a form.

Apply a Brand to a Form

MaestroThe UI design product. | Template Designer Form Builder | All versions This feature is related to all versions.

Maestro allows you to switch between brands as you work on a form in the Maestro editorMaestro editor. This way you apply all brand's specific styles, themes and item properties to the form so you can see how it looks. This is different to the runtime branding.

To switch between available brands:

1. Open a form in the Maestro editor,
2. Click Brand on the Maestro toolbar.



NOTE

You can only see the Brand button if a form uses a template containing more than one brand.

3. Select a brand you want to apply to the form the Brand dropdown list and click Save. The new brand name appears in the Maestro toolbar.
4. Click Save to safe the form.
5. Click Build with Options to build the form using the selected brand.

NOTE

If you want to use different brands on the same form, you will need to apply the brand and then build a TM Form Version of each form that contains a different brand.

You can use the `maestro.Form.view.selectedBrand` method to change brands when the form is open in the Maestro editor. However, this value is not passed through to the form when it is built, so you can't use this method for runtime branding.

Next, learn how to configure theme options.

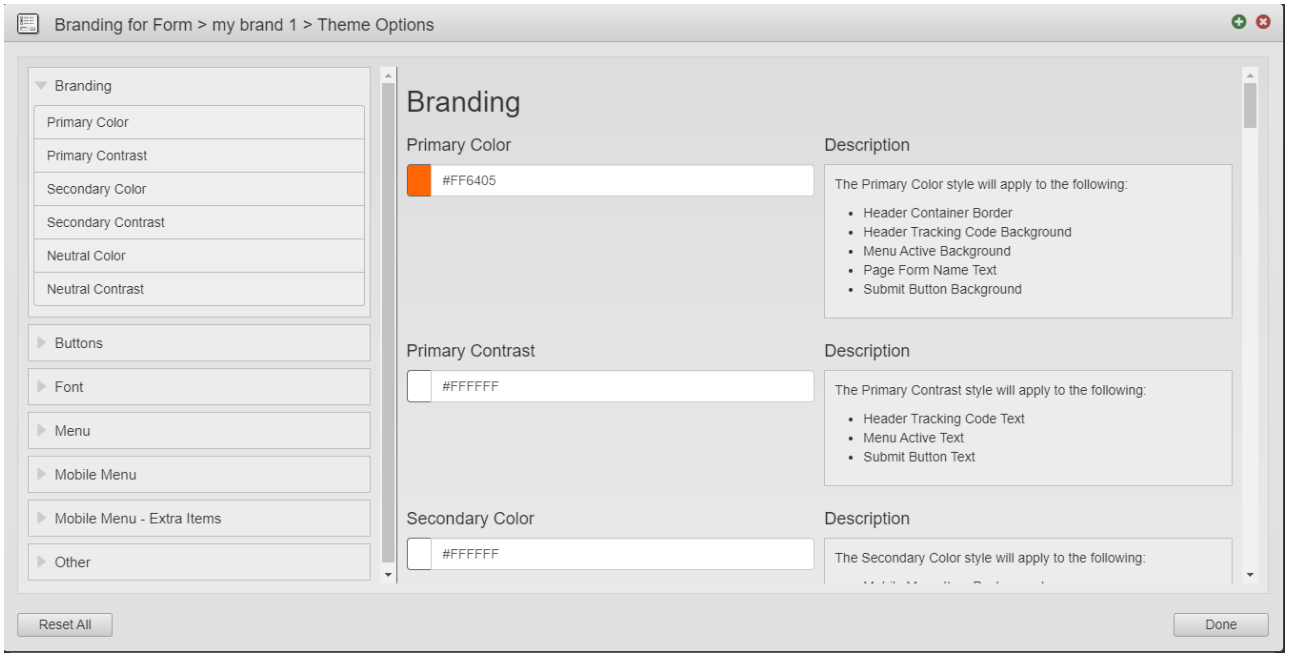
Brand Theme Options

MaestroThe UI design product. | Template Designer Form Builder | 21.11 This feature was updated in 21.11.

Maestro allows you to configure brand's specific styles, such as color scheme, font sizes, buttons and menus, using the Theme Options functionality. This way you can have different styles for various brands in one template, and you can make a common change once and have it applied within multiple styles. The Theme Options ensures that common styling elements are consistent throughout the brand. It also saves development time. You can think of Theme Options as a built-in CSS[1] file.

To configure brand's Theme Options in a template:

1. Open a template in the Maestro editor.
2. Click Brand in the Maestro toolbar.
3. Select a brand and click Theme Options.



[1]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. Find an option you want to update. Options are grouped by the following categories:
    - Branding allows you to define the fundamental color scheme of your brand. You should try to create your brand using only the colors in the Branding category, as these will be used consistently through various elements in the form. Only override the values in the more detailed areas of the brand in special circumstances.
    - Buttons
    - Font
    - Menu allows you to define the specific colors used in the Navigators. It inherits from the colors defined in the Branding category. The Description with each of these colors will indicate which elements it affects.
    - Mobile Menu - Extra Items allows you to define the specific colors used in the Navigators.
    - Other contains colors and other settings related to other aspects of the forms, such as buttons and sections.
5. Update the option. You can update multiple options as required.
6. Click Done when you have finished updating the options to go back to your brand configuration.

    NOTE

    You can select another brand to edit its Theme Options without exiting the window.

7. Click Save to save the changes.

Next, learn how to create new theme options.

Custom Theme Option

MaestroThe UI design product. | Template Designer Form Builder | 21.11 This feature was updated in 21.11.

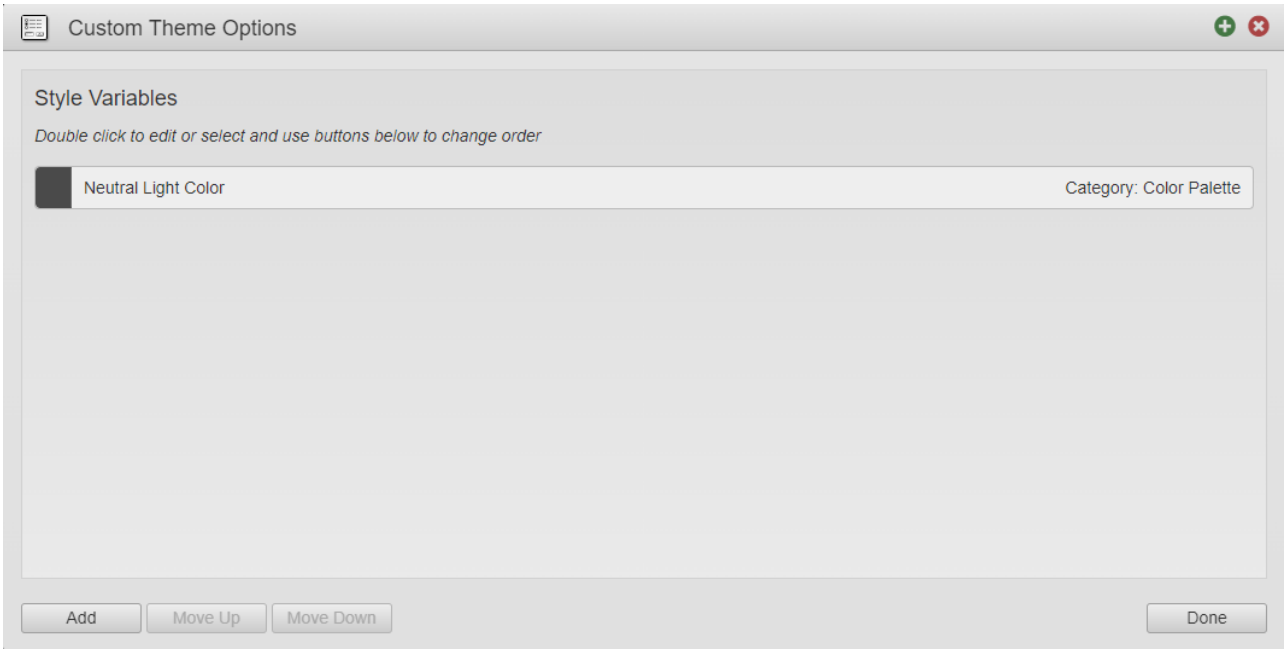
Maestro allows you to create a new custom theme options in a template, so you can add new variables and use them across various styles to improve usability and maintainability of your templates and forms. Usually, a Template Designer creates new theme options in a template as there may need to be some UI design and considerations done prior to that.

Before you create a new variable, we recommend checking the list of existing variables to see if there is one that suits your needs.

View Custom Theme Options

To view custom theme options in a template:

1. Open a template in the Maestro editor.
2. Click Brand in the Maestro toolbar.
3. Click Custom Theme Options to see the list of all custom theme options created in the template. .



4. Double-click a style variable from the list to open it in the editor. For more information, see how to create a custom theme option.

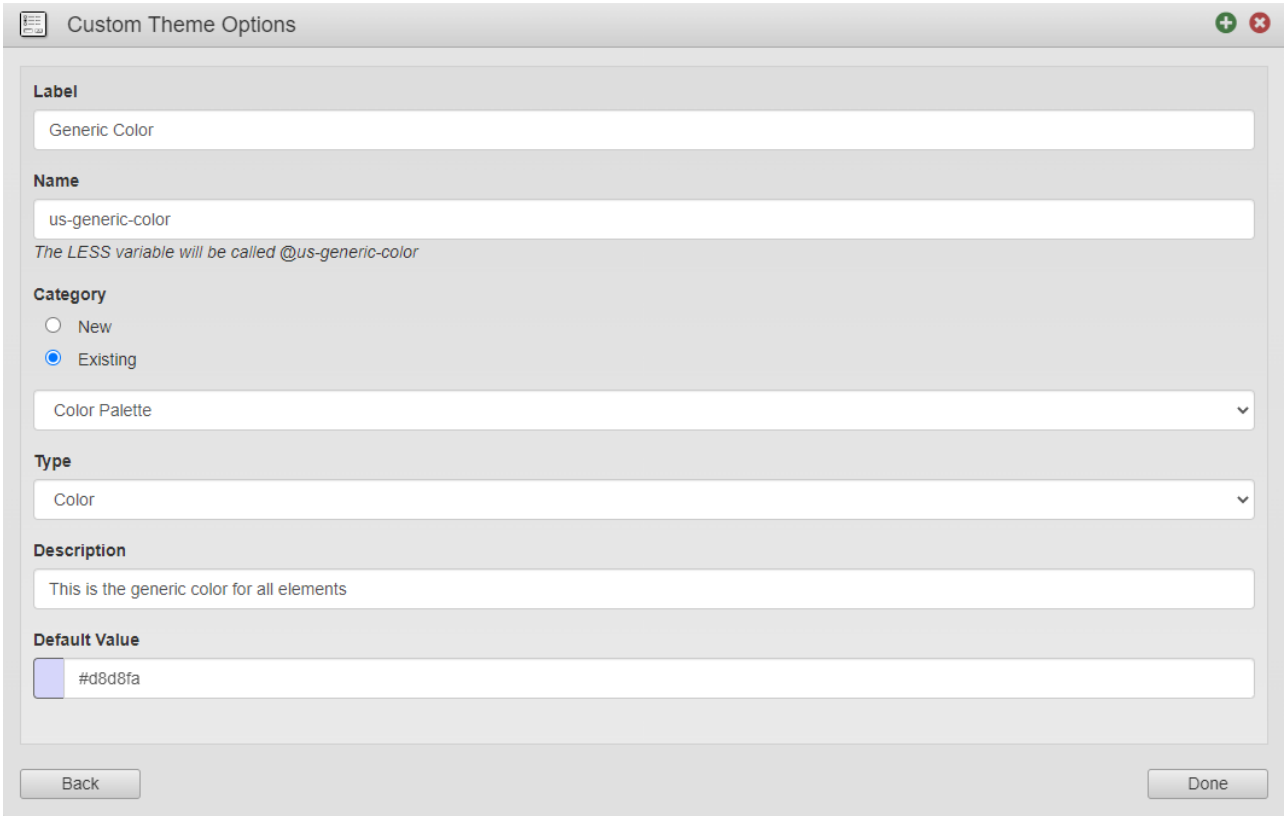
5. Click Move Up to change the order of the precedence of the selected theme option by moving it up.
6. Click Move Down to change the order of the precedence of the selected theme option by moving it down.
7. Click Done to save the changes.

If you don't see variables you need, you can create a new theme option.

Create Custom Theme Options

To create a custom theme option in a template:

1. Open a template in the Maestro editor.
2. Click Brand in the Maestro toolbar.
3. Click Custom Theme Options and then click Add to create a new theme option.



4. Enter a meaningful name for the theme option in the **Label** text field.

   NOTE

   Try to create more generic theme options so they can be used in various styles or parts of a style, for example, Neutral Light Color.

5. Correct a variable name in the **Name** text field. The Maestro editor auto-generates this name based on the label you have entered above, but you can change it as necessary.
6. Select a category under the Category section that can be:
   - New - to create a new category
   - Existing - to select from the list of existing categories, which can be one of the following:
     - Navigator
     - Layout
     - Fonts
     - Color Palette
     - Fields
     - Buttons
7. Select a variable type from the Type dropdown list, which can be one of the following:
   - Text
   - Color
   - Pixels
8. Enter a meaningful description in the Description text field.
9. Provide a default value in the Default Value text field.
10. Click Done to save the changes.

The new theme option should appear in Theme Options under the category you have selected. Now, you can use it in styles of your components. It's important to understand that you don't really publish new variables so they will become available in styles. Instead, you publish the template, which defines the variables, and you publish shared styles, which use these variables, to the organization-level stye library. For more information, see organization structure.

Next, learn how to configure item properties.

Brand Item Properties

MaestroThe UI design product. | Template Designer Form Builder | 21.11 This feature was updated in 21.11.

Maestro allows you to configure component's properties, such as logo or background layer, differently for each brand in your template using the Item Properties functionality. This way every form using template gets the same settings, so it is like a global option.

NOTE

Compare this to Form Options where you can set component's properties same for all brands, but every form that uses a template can be set differently. These are change on a form level.

The primary purpose of Item Properties is to allow you to control the look and behavior of multiple components by modifying a single property. This allows you to avoid making identical changes to multiple components, saving time and decreasing the likelihood of inconsistent behavior appearing in the template.

Another benefit of Item Properties is that they provide a central source of truth for adjusting the most essential properties. This means you don't have to dig around the structure of the form to find the components that you'll almost certainly want to configure.

To explain how it works, let's look at the Logo's Item Property that appears in the top left-hand corner of the template is an Image component. This component has a property which defines which image to use. Because the Image component is part of the template's header - which is not available for editing - there is no way of modifying it. To make the image logo configurable in the brand, the Template Designer has added this particular property to the Item Properties in the brand.

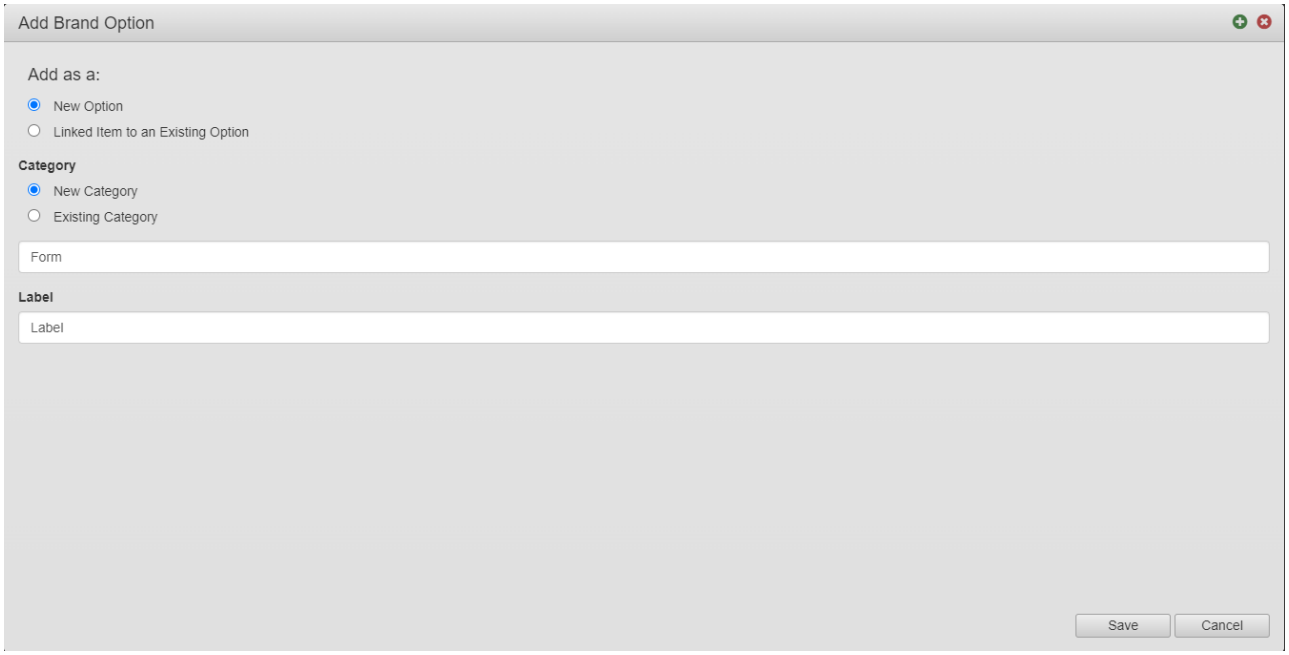
Brand's Item Properties work like this - first, you need to add a component's property to brands in a template you are building. Then you edit a value of the property for each brand using Item Properties. After you save and publish the template, you build a form using one of the brands. You can always remove the property from the brand when it's needed any more.

To add a component's property to a brand:

1. Open a template in the Maestro editor.
2. Select a component in the Design mode and click the Properties tab.

3. Right click on a property you want to add, for example, Label or Hide Section Header, and select Add to Brand Options.

> NOTE
>
> You also have a choice of adding a component's property to Form Options. However, if you've already added the property to brands, you can't add it to form options. You must remove it from brands first.



4. Select how you want to add a property from the Add as a radio button group:
   - New Option to add as a new property.
   - Linked Item to Existing Option to add as linked to an existing property and then select an existing property from the Select Existing Option dropdown list.
5. Select a category to use for the property from the Category radio button group:
   - New Category to create a new category for the new property and type a category name in the field.
   - Existing Category to add a new property to the already existing category and then select an existing category from the dropdown list.

> NOTE
>
> Adding categories help you group and navigate properties, especially, when there are many of them.

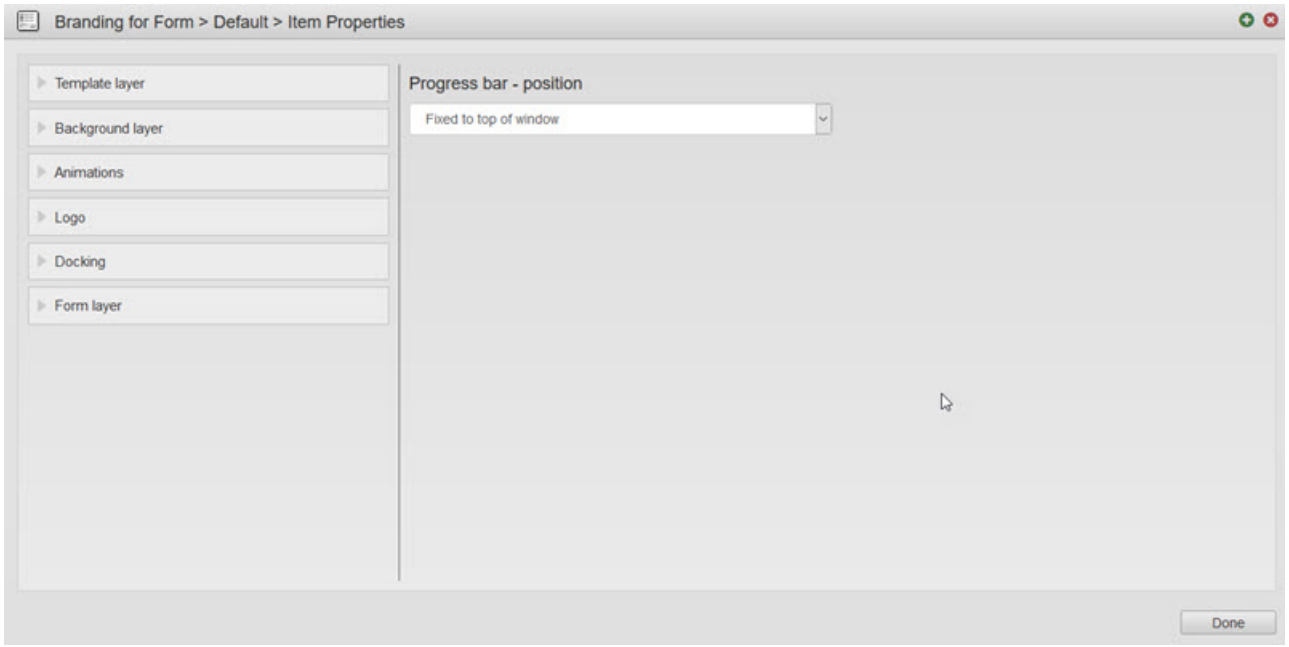
6. Type a name for the property in the Label field
7. Click Save. The property is added to Item Properties, where you can edit it.

Now, you can access this property using the Item Properties option.

To configure a component's property in a template:

1. Open a template in the Maestro editor.
2. Click Brand in the Maestro toolbar.
3. Select a brand and click Item Properties.

> NOTE
>
> You can configure the Default brand as well.



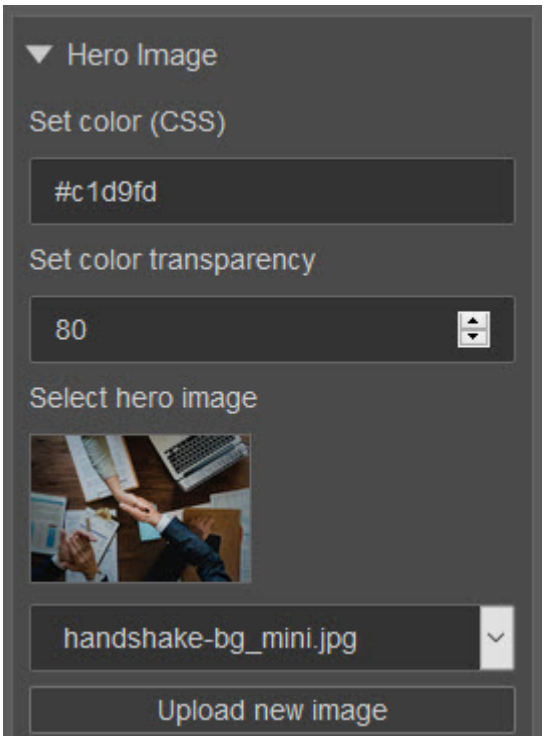
4. Find an item property you need and update it. Item Properties are grouped by categories.
5. Click Done to go back to your brand configuration and click Save.
6. Click Save in the Maestro toolbar to save the template.
7. Click Publish to make the template changes to be available to forms that use it.

> NOTE
>
> The template changes may not be available to a forms depending on where the extension points are in the template - anything that's inside the extension point (that can be changed in the form) can't be updated by the template

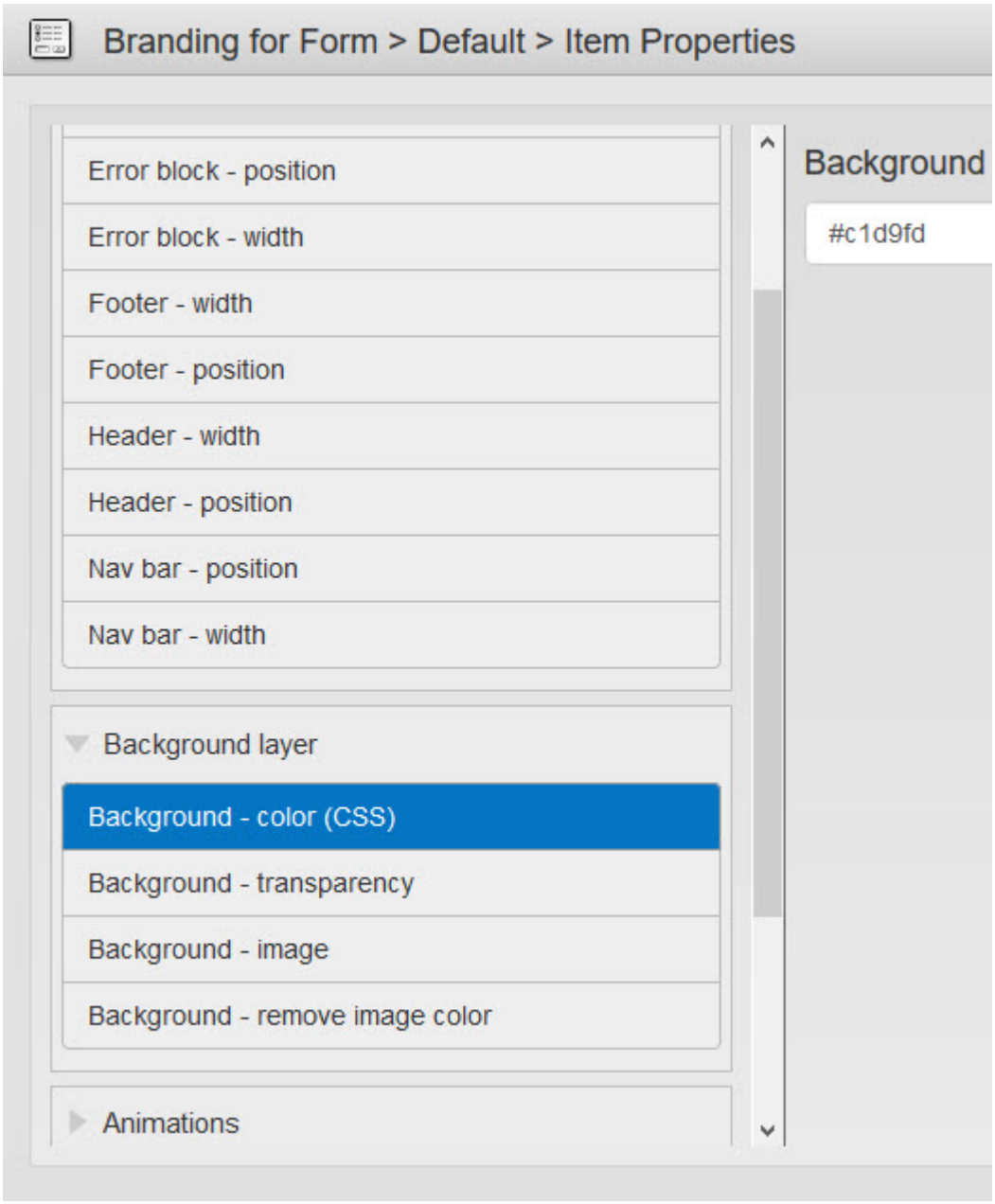
To remove an Item Property from a brand:

1. Open a template in the Maestro editor.
2. Select a component in the Design mode and click the Properties tab.
3. Right click on Label and select Remove from Brand Options. The Item Property is removed from all brands defined in the template.

It's important to note that, if a property is available via the Item Properties window, you should not edit that property on the component itself. For example, if you select the Background Layer component, you may be tempted to configure the component via its Properties pane, as shown below:

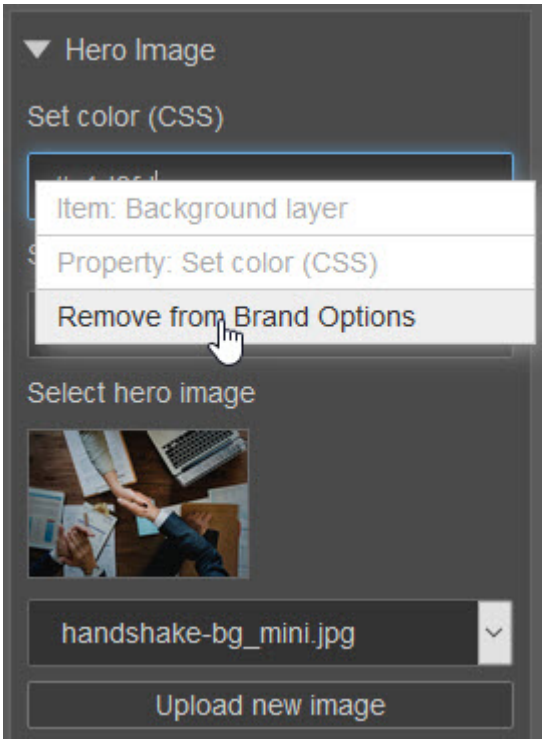


However, if you change the properties of this component, they won't be saved. This is because the properties have been made available using the Item Properties functionality, as shown below:



As a result, they must be configured via the Brands' Item Properties window.

At the moment, the only way to identify how a component's property should be modified is to right-click on the property in the **Properties** pane. If the **Remove from Brand Options** option is available, the property must be configured via the Item Properties window.



Next, learn how to change the style of components in a brand.

Change the Style of Components in Brands

MaestroThe UI design product. | Template Designer Form Builder | All versions This feature is related to all versions.

Brands allows you to configure the look and feel of the contents of a template, so you can control the styling of any components of a particular type that are added to a Maestro form by a Form Builder.

To change the style of components in a brand:

1. Open a template in the Maestro editor and select the Style tab.
2. Drag an instance of the component into the template (you will delete the component before publishing).
3. Adjust the styling as required.
4. Click Save to Shared Style to save your changes. Use a meaningful name when saving the style.
5. Right-click the component in the Palette pane, and then select Available Styles for <Component Type>.
6. Select your newly created shared style from the dropdown menu of available styles.
7. Click Save.

NOTE

You can remove this style from the same dialog by clicking the x next to the style.

To apply the shared style to the component type:

1. Right click on any instance of the component in the form.
2. Select Apply Shared Styles to every <component type>.

For more information, see the Style pane.

Next, learn more about shared styles.

Runtime Branding

MaestroThe UI design product. | Template Designer ' Form Builder | 21.11 This feature was introduced in 21.11.

Maestro allows you to update a brand dynamically using the runtime branding feature. This functionality is based on JSON[1] configuration file that specifies the details of the brand, such as images, CSS[2] content, translation setting, as well as components and widgets' properties, which can be referenced by rules or by Form functions.

Maestro provides you with two specialized components for creating runtime brands - Runtime CSS Resource and Bundle Resource Library components.

Let's start with understanding the JSON structure and how it relays to a brand.

JSON Structure

The JSON file contains the following objects and their corresponding properties that help you define a brand, with some objects configured as single entities or as arrays:

---

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

[2]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.

css

This object has several properties specifying a CSS[1] file to use. You can also define an array of CSS objects, if you want to use different CSS files to control color, look and feel, and so on.

- id - an ID of CSS to update as defined in the ID property on a Runtime CSS Resource widget in the form.
- cssUrl - a new URL for the CSS to use, for example, using a relative URL like `./css/myCustomCss.css` will get the image from the FAR file. Setting the URL takes priority over setting CSS Content.
- cssContent - a new CSS content to inject into a style tag for the CSS name.

You can use the `Form.updateCssByName()` function to update named CSS with either a new URL or CSS content, switching between a URL and a CSS string which will remove the previous link or style tag as appropriate and replace it with the one based on the update required.

images

This object allows you to control an image using the following properties.

- imageName - the image name as defined in an image widget in the form.
- imgUrl - a new URL for the image to use, which can be a relative or absolute URL. For example, a relative URL `./images/myImage.png` will specify the image in the FAR file.
- imgHeight - a new height for the image.
- imgWidth - a new width for the image.
- alternativeText - an alternative text for the image, this is populated into the alt attribute on the image.
- toolTip - a tooltip text for the image, this is populated into the title attribute on the image.
- imageLink - a URL for the image to go to, leaving this blank or null will not take the user anywhere on click.

You can use the `Form.updateImageByName()` function to update images in the form by its name specified in the image widget.

---

[1]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.

translation

You can define only one translation per brand by using either this object or the `translation_template` property described below.

- code - a language code, which is used by the `Translation.fetch()` function.
- url - a URL to a translation CSV file if it is not included in the FAR file. This is an optional parameter.

You can use the `Translation.fetch()` function to load the translation for the specified language code. For more information, see how to configure a new language.

If the translation loads, the interface will refresh with the localized terminology. If the translation doesn't load, an error will appear in the JavaScript Console. A simple way to implement it in a form is to add a button that calls `Translation.fetch("myLanguageCode", false, "/web-plugin/resources/mynewCSC.csv")`.This translates the form based on a translation file that's at an external URL. In this example, it's in a portal resource on the Manager server, but it could be anywhere.

translation_template

This property specify a template to use for your translation using the locale, for example `en_AU`.

dataValues

This object contains key / value pairs allowing you to update the form's data model. Values can be both plain text and rich text, such as HTML, defining component's fields. For example, the JSON's property `dataValues: { footerUrl: "https://myUrl.com" }` will be populated into the `data.footerUrl` data model element.

The following JSON example shows a simple brand configuration:

```
css: [{
    id: "runtimeCssResource",
    cssUrl: "",
    cssContent: ""
}],
images: [{
    imageName:"myLogo",
    imgUrl: "./images/anotherLogo.jpg",
    imgHeight: "",
    imgWidth: "",
    alternativeText: "",
    toolTip: "",
    imageLink: ""
}],
translation_template: "en_AU",
translation: {
    code: "en_AU",
    url: ""
},
dataValues: {
    footerText: "",
    headerText: ""
}
```

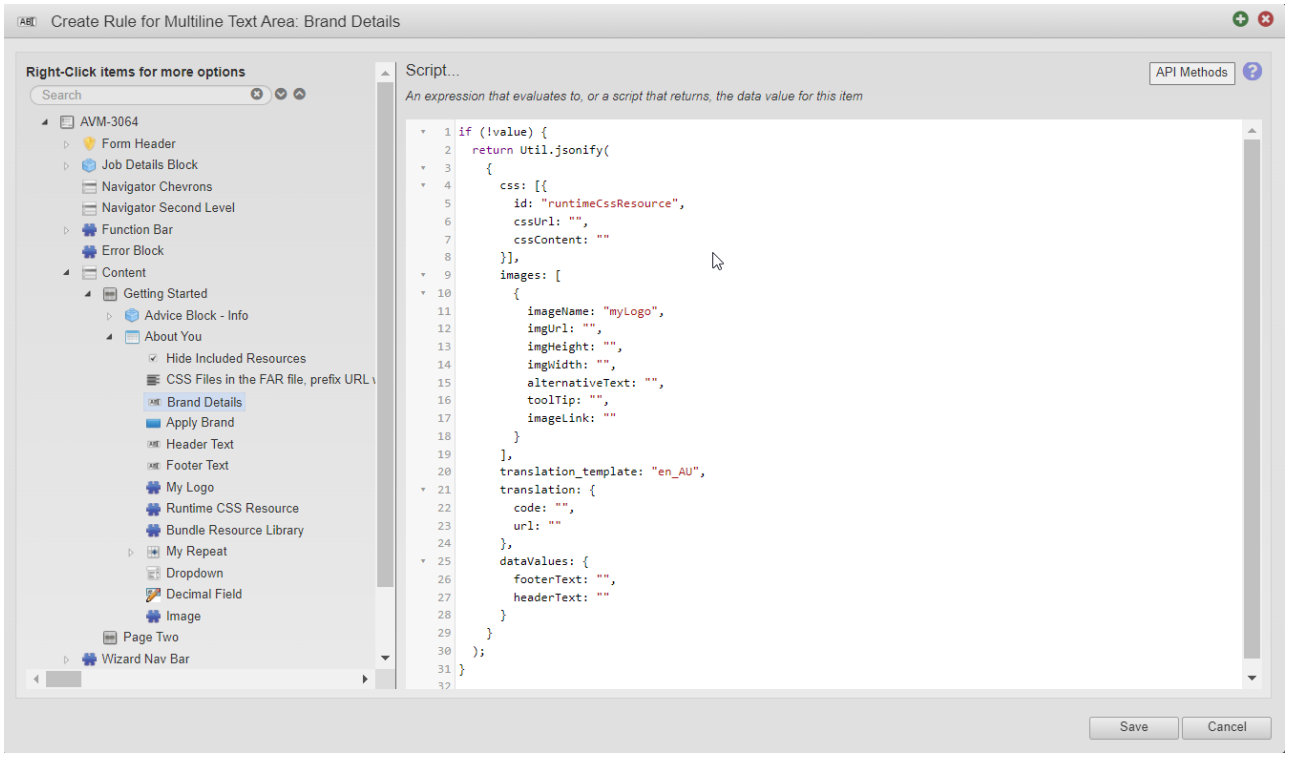
To illustrate how runtime branding works, imagine that you want to update the form's copyright footer text, which is configured in a brand as `Copyright {{data.footerText}}`. With runtime branding, all you need to do is to pass the updated `footerText: "My new copyright text"` parameter into the form, so the next time it is rendered in the browser, the updated copyright text is shown to the user. As a comparison, using the design-time branding approach, you would need to edit the brand, re-build the form and then deploy it in your Manager.

Rules

You can use Maestro rules to add and configure brand's properties using the JSON file.

To configure a brand:

1. Open a brand in the Maestro editor.
2. Select a component in the form.
3. Select the Properties tab and click Calculation under the Rules section.



4. Edit the content of the script using JSON syntax in the Script window.
5. Click Save.
6. Rebuild the form and render it to check the updated brand.

API

Maestro introduces the new JavaScript API in the Form namespace that you can use to dynamically update your brand.

- `Form.updateImageByName()`
- `Form.updateCssByName()`
- `Form.updateBranding()`
- `Translation.fetch()`

For more information, see the Maestro API reference documentation.

Next, learn how to create a brand.

Extension Points Overview

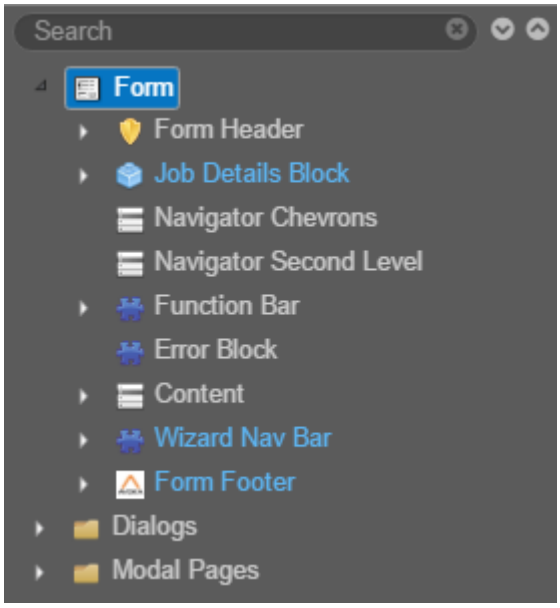
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the concept of Extension Points that define areas within a template where Template Designers allow Form Builders to add their content. Extension points are normally used to control common areas, such as headers, footers, and navigation bars, that should only be styled by a Template Designer. If no extension points are set in a template, the Form Builder has access to the entire form.

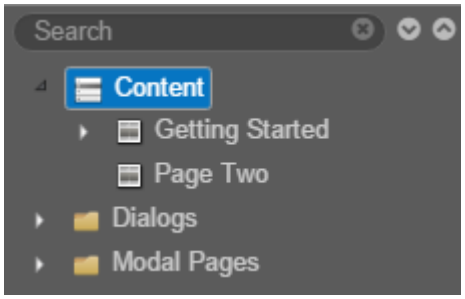
NOTE

An extension point should always be set on Content because this area is used for all elements of the form, which gives a Form Builder complete access to all the content of the extended container. However, it can be also set on the Dialogs and Modal Pages, if required.

To illustrate how an extension point works, let's look at the following template and a form structures:



Template Structure



Form Structure

In this example, the extension point is set on Content, so the Form Builder, who works with the form structure, won't be able to select the Form container or the Navigator components. However, the Template Designer has access to all elements in a form.

Extension points can't be set within other extension points. If there is an extension point already created for an outside container, a warning message is displayed, which tells you that the new extension point will cause an existing extension point to be removed. As a Template Designer, you have the option to restrict an extension point, or remove the extension point completely.

For example, if you have Getting Started set as an extension point (see the image above), you will receive a warning when you set Content or About You section as an extension point. If you accept the warning message, the extension points will be automatically removed from the conflicting container.

Changes to extension points are updated in forms once the template is published, so be aware that this may affect content within existing forms that use this template.

It is important to create templates with the correct extension points and then publish them as soon as possible. These templates are known as placeholder templates. They allow Form Builders to start working on the form, while you are still styling and modifying the template. This provides you with the ability to work in parallel, which significantly speeds up form development.

Next, learn how to set an extension point.

Set an Extension Point

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A new template that you create in Maestro doesn't come with extension points by default. You must add them manually to the areas where you want to restrict access to certain parts of a form. Once an extension point is set, Form Builders have the ability to add, remove or modify the elements within a container that is set as an extension point. The container's content now belongs to a form, so any changes made to this content in the template won't be automatically propagated to the form. This is an expected behavior and you should be aware of this when setting up extension points.

To set an extension point in a template:

1. Open a template in the Maestro editor.
2. Click the View tab and expand Form.
3. Select a container, for example Content, and right-click on it.

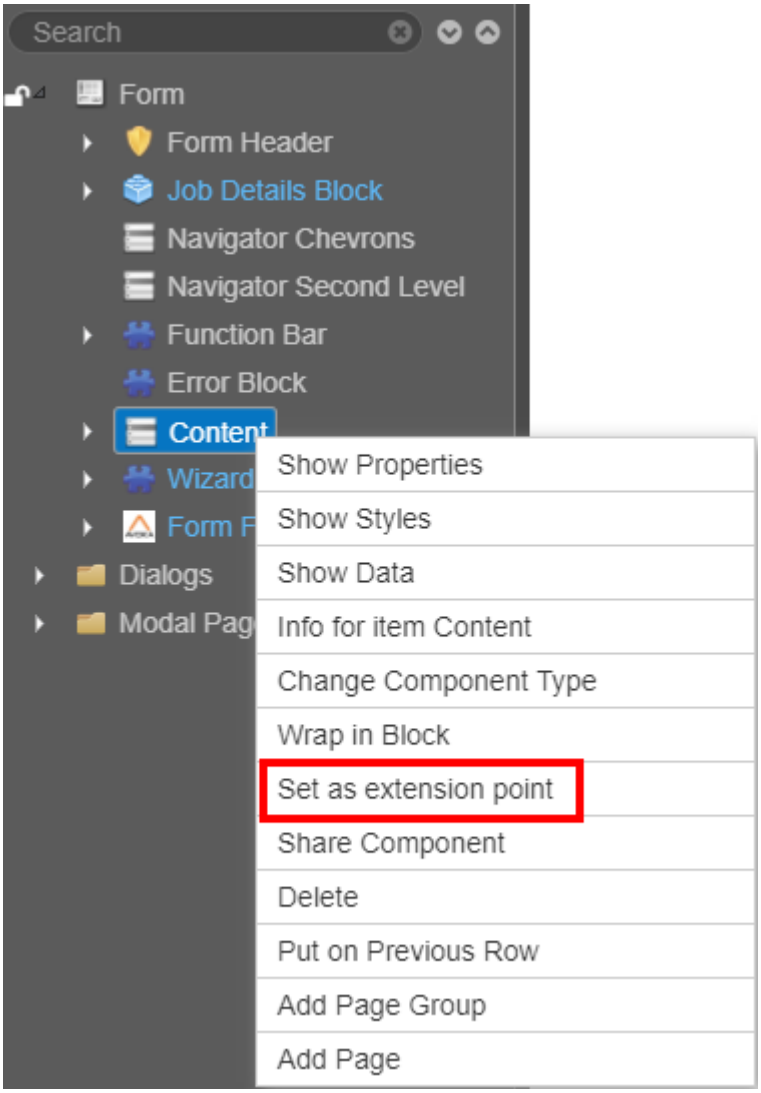
   NOTE

   We recommend applying an extension point to Content or Dialogs and Modal Pages content areas only.

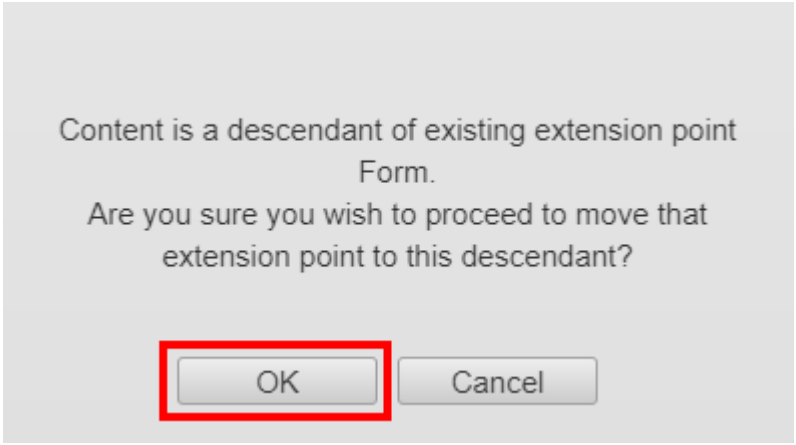
4. Select Set as extension point from the dropdown list to apply the extension point.

   NOTE

   Currently, Form has the extension point configured, which is represented with the unlocked padlock icon.



5. Click OK to confirm or click Cancel if you don't want to proceed.

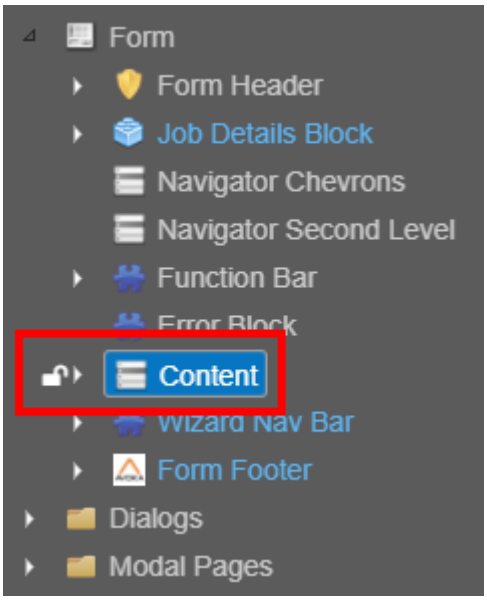


The warning message tells you that the extension point is being removed from Form, which means that other areas within the form structure will no longer be available in the form.

NOTE

Be aware that changes to extension points may affect content within existing forms that are using that template.

The Content now has the extension point, which is represented with the unlocked padlock icon.



6. Click Save to update the template.

Repeat the process above, if you need to add more extension point, for example, for Metadata and Narrator Controller under the Nuts & bolts container. You should notice the unlocked padlock icon next to each of the items that are set as an extension point.

NOTE

You must publish the template or re-publish it again for the change to the extension point to take effect in forms that use it.

Maestro allows you to restrict an Extension Point. That is, if there are multiple containers underneath a current extension point, you can set the extension point further down in the tree. However, you can only have one extension point within each level of the hierarchy. To reset back to a higher level in the hierarchy, set the extension point at a higher level - all lower levels will be automatically removed.

Next, learn how to remove an extension point.

Remove an Extension Point

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to remove an extension point from a template. You may need to do this if you created the extension point by mistake, want to move it or don't need it any more.

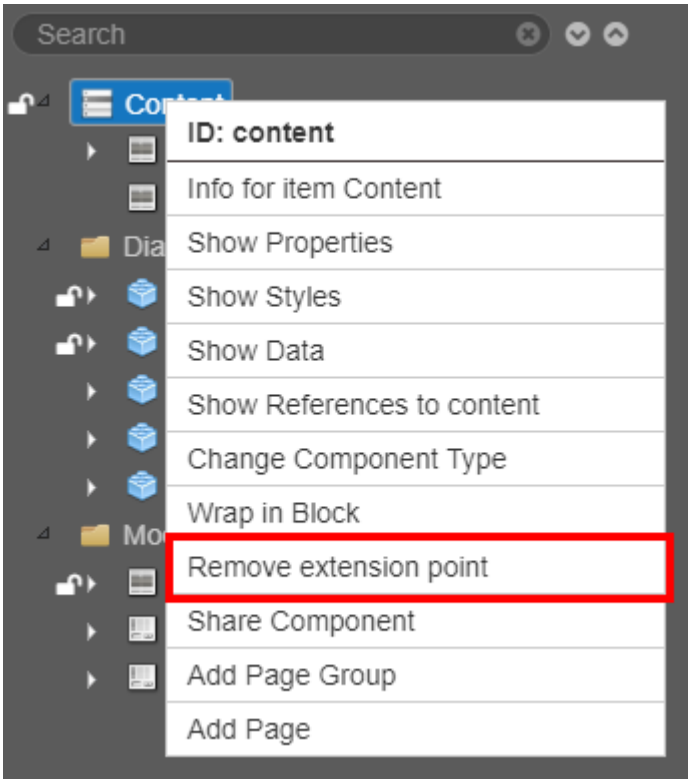
> NOTE
>
> Removing of extension points can affect forms in many different ways, so we recommend it to be done by a Template Designer.

To remove an extension point:

1. Open a template in the Maestro editor.
2. Click the View tab and expand Form.
3. Select a container that has an extension point configured, for example Content, and right-click on it.

   > NOTE
   >
   > The extension point is represented with the unlocked padlock icon. If you can't find this icon, it means there are no extension points set in the template.

4. Select Remove extension point from the dropdown list to delete the extension point.



5. Click Save to update the template.

NOTE

You must publish the template or re-publish it again for the change to the extension point to take effect in forms that use it.

If you have removed all extension points in the template, the Form Builder will have access to all elements of the form. This scenario is not ideal as all consistency controlled by the template will no longer be available.

Next, learn about save and resume.

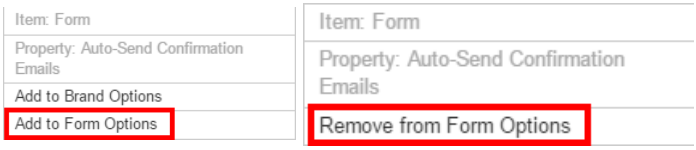
Setting Form Options for Extension Points

MaestroThe UI design product. | Template Designer | All versions This feature is related to all versions.

TO BE REMOVED?

The Template Designer can control which options display within the Form Options window.

Add to Form Options Remove from Form Options



1. From the hierarchy, View or Structure pane, select the element that contains the property you want to add to the Form Options window.
2. With the element selected, right-click the option from the Properties pane. If the property is not currently in the Form Options window you will see Add to Form Options. If the property is already in the Form Options window you will see Remove from Form Options.
3. When you select Add to Form Options, the Add Form Options window will display.

   Add as a:

   New Option - Thiswill create a new option within the Form Options window.

   Linked Item to an Existing Option - This will allow you to link the selected item to an existing item in the Form Options window.

   Category: This is the area within the Form Options window the property will display.

   New Category - You can create a new category within the Form Options window. Once selected you can enter a name.

   Existing category - This will add the property to an existing category within the Form Options window. When you select the option, you will then have a dropdown of the existing categories.

   Label: This is the name used for the property in the Form Options window.

3. Inverted Data: This is used to turn a negative property into a positive option. For example, if you add the Exclude property (which is a negative) to Form Options, you could select Inverted Data which would turn it into an Include option from within the Form Options window.



4. Once the selections are complete, click Save. The property will now be added to the Form Options window.

Save Challenge Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro comes with the Save Challenge feature, also called Save and Resume, which allows a form user to save a form, with all filled data, at any point and then open it later to resume filling the form from the point where the user left off. There are many business cases where it's very handy for a form applicant. For example, when they don't have all required information or documents on hands and therefore the form application has to be postponed until the form applicant will get it available.

When the user saves a form, they are given an automatically generated reference code. This reference code is defined in Manager ad is required to open the form. Along with the reference code, the form builder can also include a security question. Using the reference code, along with a security question, creates two factor authentications[1].

If implemented, when saving a form, a form user will be prompted to answer the security question. If the form user provides the correct details, the form user will be able to save the form with their data still in place when they return to the form. When returning to the form, the form user will be prompted to answer the security question again and provide the reference code that they were given when they initially saved the form.

Any component type can be used for the security question, such as a text field.

When the user saves their form, they are presented with the Save Confirm dialog. When the user returns to resume their form, they are presented with the Save Challenge dialog.

Save Challenge can be configured to popular answers to the configured questions with data previously entered by the user, and a save challenge can be configured with multiple questions using multiple components.

---

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

Next, learn about best practices on creating a save challenge with multiple components.

Create a Save Challenge

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

In this topic, we're going to create a save challenge from scratch. This will provide insight into how to create your own save challenges and how to customize save challenges in existing templates.

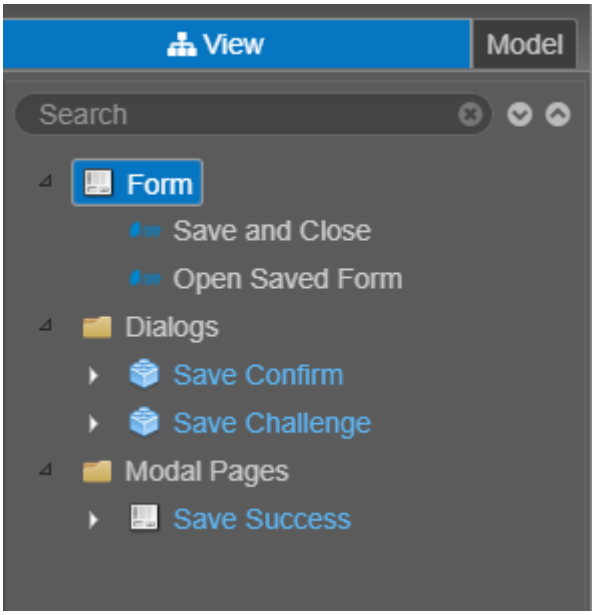
To setup this feature:

1. Create a Maestro form with the Core - Empty Form template.
2. Drag a Save Confirm component into the Dialogs folder. This dialog allows a user to save the form.
3. Drag a Save Success component into the Modal Pages folder. This modal loads after the user saves a form and displays the reference code they can use to re-open the form.
4. Drag a Save Challenge component into the Dialogs folder. This dialog allows the user to submit a reference code. If the reference code is valid, it will open the form associated with that code

   > NOTE
   >
   > In earlier versions of Maestro, the Save Challenge component was named TRN Save Challenge.

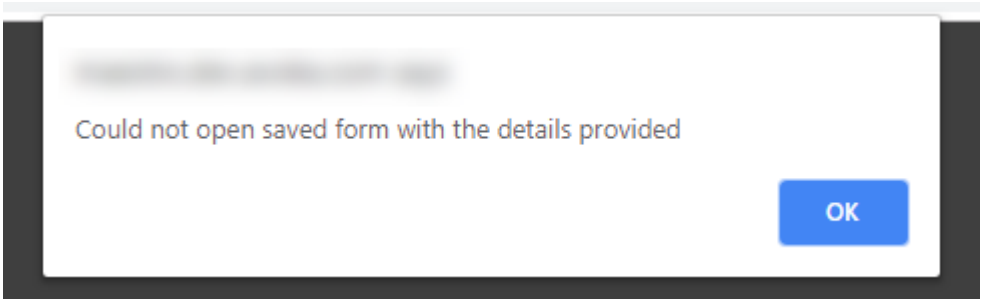
5. Drag a Save and Close component into the form. This button will open the Save Confirm dialog.
6. Drag an Open Saved Form component into the form. This button will open the Save Challenge dialog.

You should end up with a View pane that looks like this:



Based on these changes, the save challenge feature is now functional. If you render the form and click the Save and Close button, you'll see the reference code assigned to the transaction. Then, if you click the Open Saved Form button and enter the reference code, the saved form will open.

If you enter an incorrect reference code, an error will be returned:



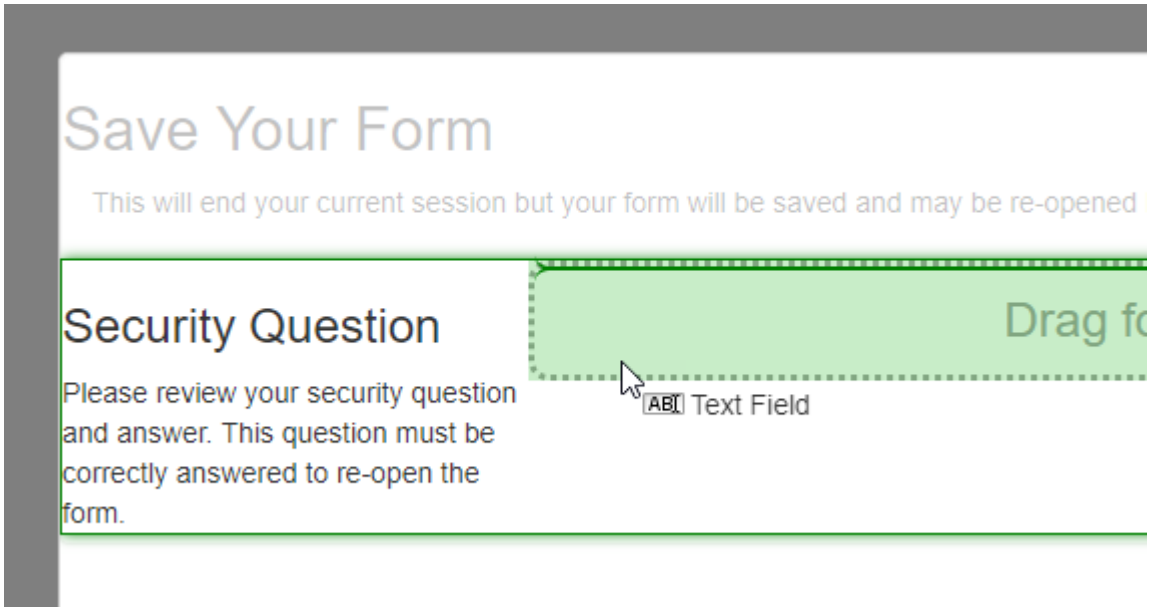
If you receive this error after entering a *correct* reference code, it's probably because you're using an earlier version of Maestro. In this case, you'll need to add a security question for the save challenge feature to work. Fortunately, adding a security question is considered a best practice.

Adding a Security Question

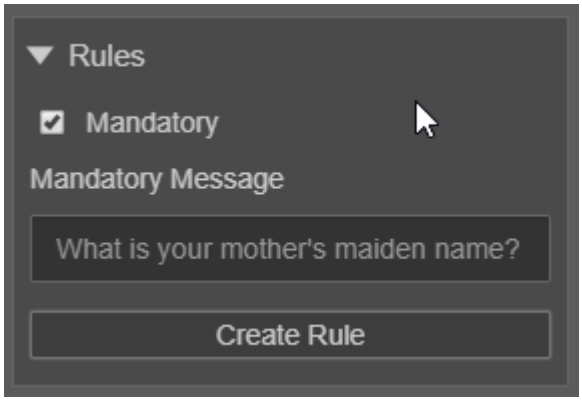
Security questions ensure an extra layer of protection for users who save their forms. Rather than simply entering a reference code, they're required to enter an additional piece of information to verify their identity.

To add a security question to the Save Confirm dialog:

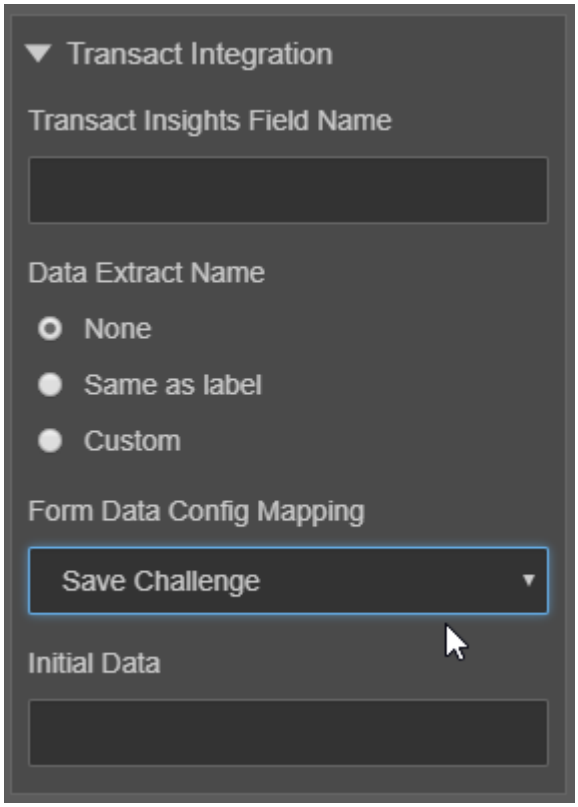
1. Drag a Text Field component into the Save Confirm component's Security Question block.



2. Change the label of the text field to a question, such as "What is your mother's maiden name?"
3. Enable the Mandatory rule to prevent users from saving a form without a security answer.

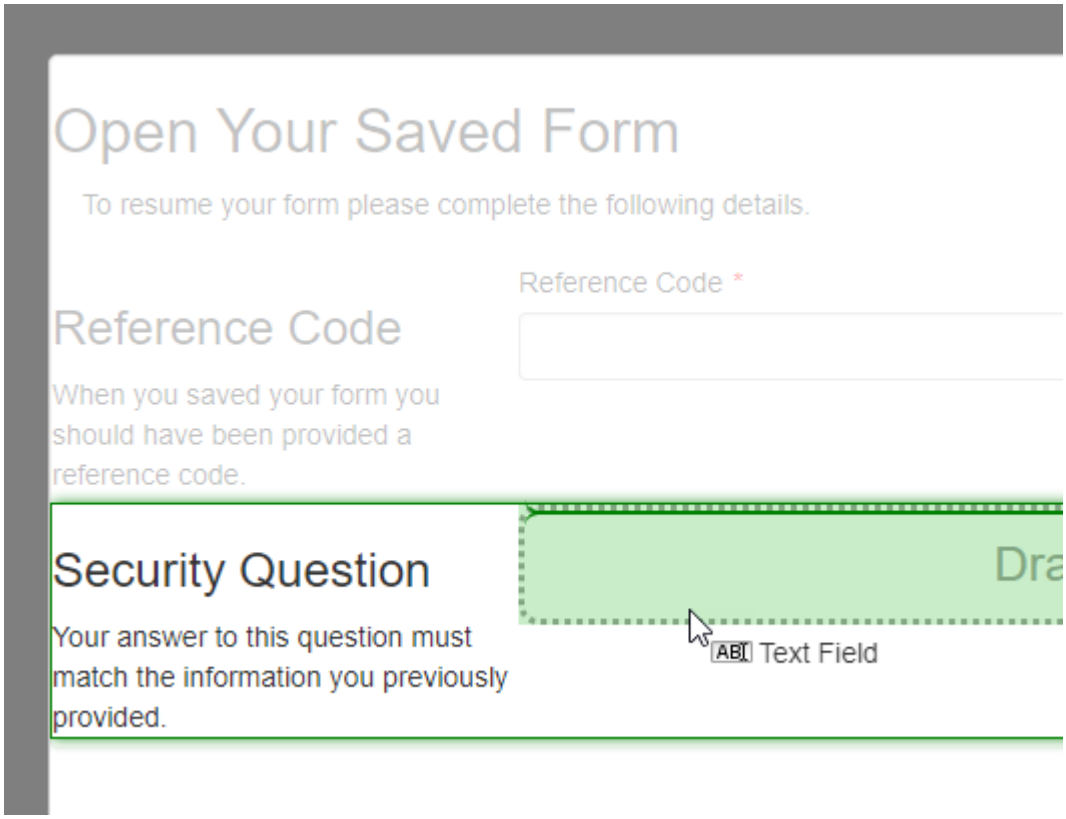


4. Switch to the Integration tab, expand the Integration panel, and select Save Challenge from the Form Data Config Mapping dropdown.



Then, to allow users to provide their security answer via the Save Challenge dialog:

1. Drag a Text Field component into the Save Challenge component's Security Question block.



2. Change the label of the text field to the same label as the previous text field.
3. Enable the Mandatory rule to prevent users from trying to open a form without a security answer.

It's important to note that only the first text field – the text field in the Save Confirm component – should be mapped using the Form Data Config Mapping dropdown. If you map the second text field, the original mapping will be lost, as only one field can be mapped to the Save Challenge option at any one time.

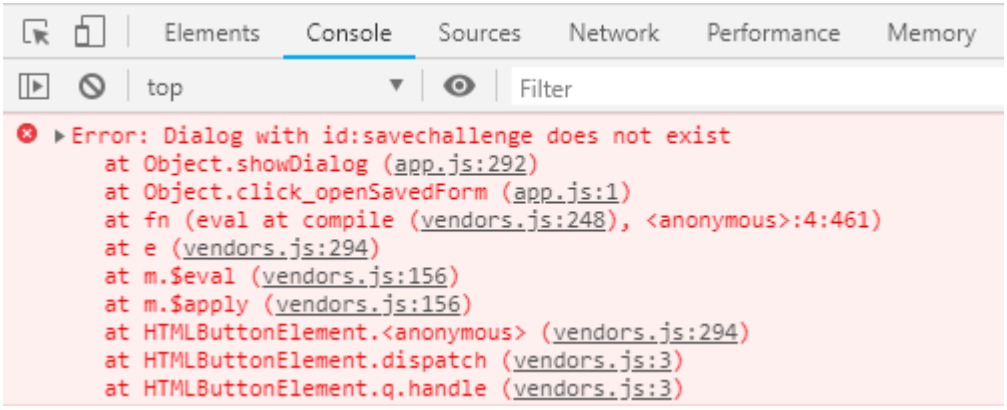
Based on these changes, users will now be required to answer a security question when saving and opening a form.

Troubleshooting Dialogs

The reason the Save and Close and Open Saved Form buttons automatically open the relevant dialogs is because of the IDs associated with the components. The Save Challenge component, for instance, has an ID of "savechallenge", and the Open Saved Form button has a rule that opens a dialog with the ID of "savechallenge":

```
Form.showDialog("savechallenge");
```

If a dialog isn't opening for whatever reason, it's worth verifying that the IDs are setup correctly. The easiest way to do this is to open the Google Chrome DevTools before attempting to open the dialog. If there's an error with the ID, it will appear via the Console:



Configure Security Questions for Save Confirm Dialog

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

To configure a save challenge security question in the save confirm dialog of a Maestro form.

1. Open a Maestro form.
2. Expand Dialogs.
3. Click Save Confirm.
4. Add a component to the security question section within the dialog.
5. Map the component as the save challenge.
6. Click the Integration tab.
7. Expand Integration.
8. Select Save Challenge from the Form Data Config Mapping dropdown menu.
9. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

You can only add one security question to the Save Confirm dialog. However, you can use a calculated data field to configure and use multiple components for the security question. When using multiple components and questions to create the security question and answer, ensure that the calculated field that combines the multiple questions, is the first field on the Challenge Dialog. Alternatively, you can update or override the JavaScript that performs security check.

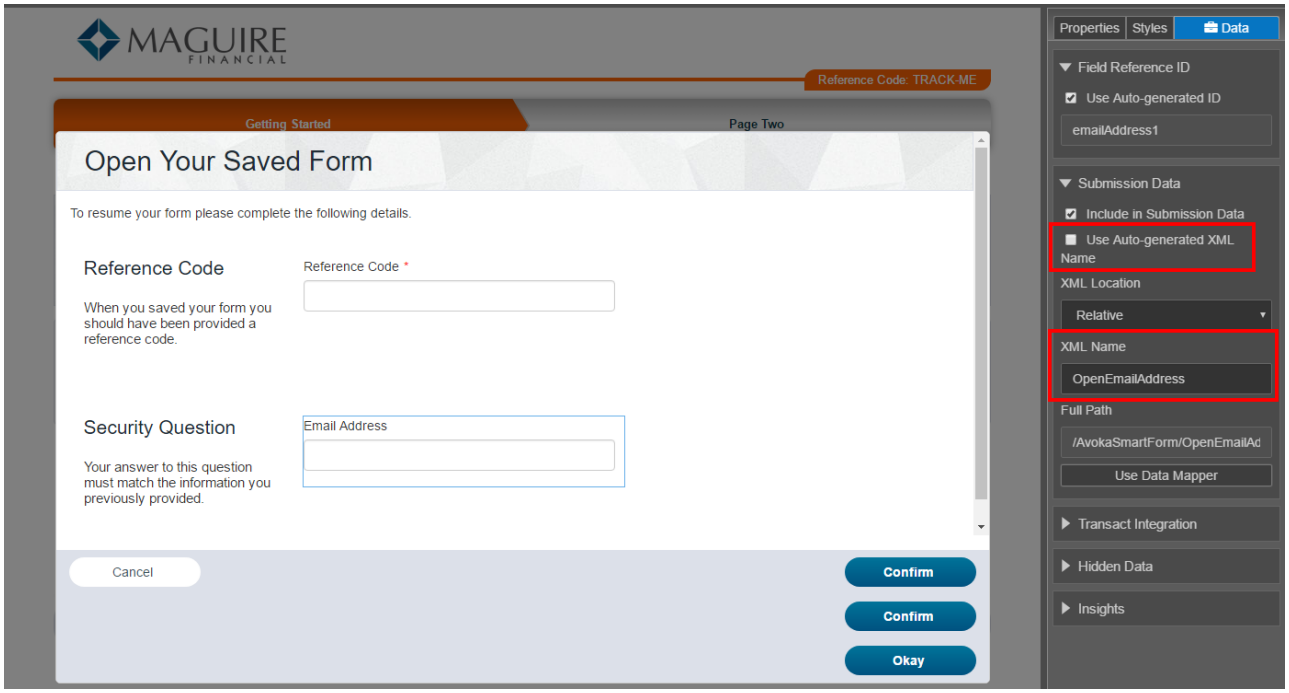
Next, configure the save challenge dialog security question.

Configure Security Questions in Save Challenge Dialog

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

To configure a save challenge security question in the save challenge dialog, follow these steps:

1. Open a Maestro form.
2. Expand Dialogs.
3. Select Save Challenge.
4. Click Copy to copy the component.
5. Paste the component within the Save Challenge security question section. The Save Challenge security question doesn't need to be mapped to the Save Challenge System Mapping. You only need to map the component displayed on the Save Confirm dialog, this is done when the save confirm dialog is configured with a security question.
6. Expand Save Challenge > Save Challenge Content.
7. Select Security Question.
8. Select the Integration tab.
9. Clear Use Auto-generated XML.
10. Change the XML Name.



12. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Populate Security Questions from User Entered Data

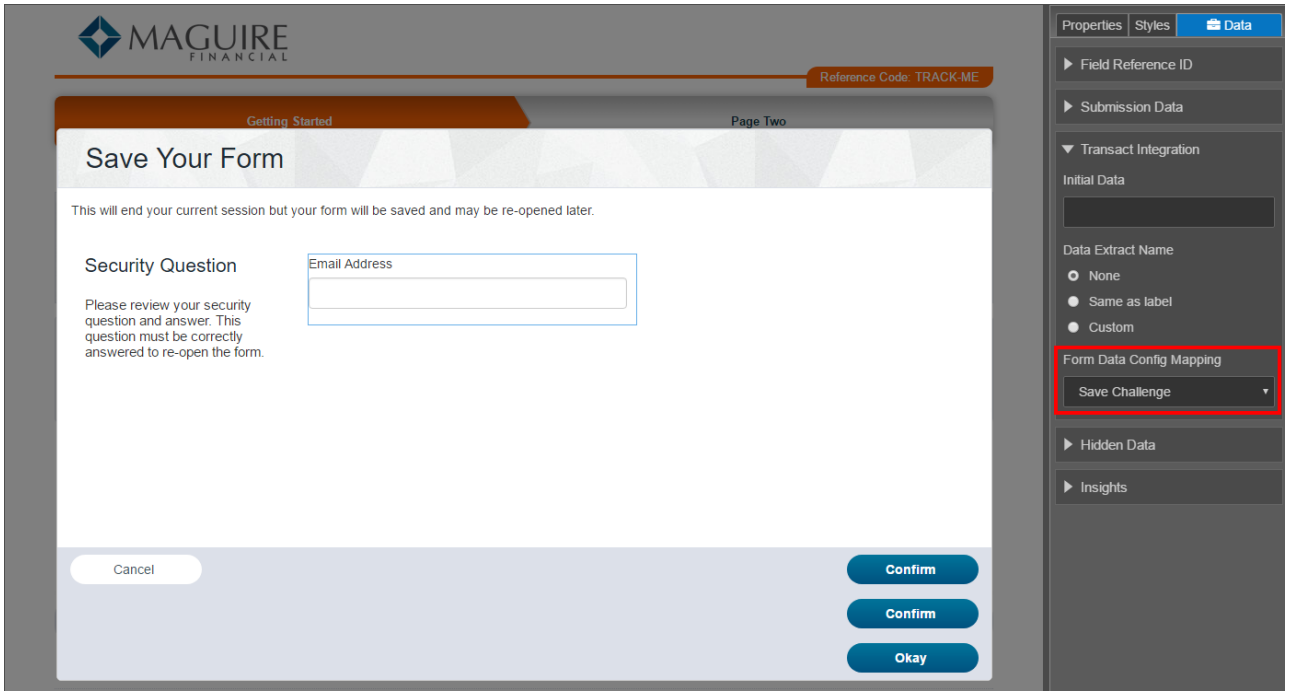
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Another option for the save challenge security question is to populate the answer from a component the user has already completed within the form.

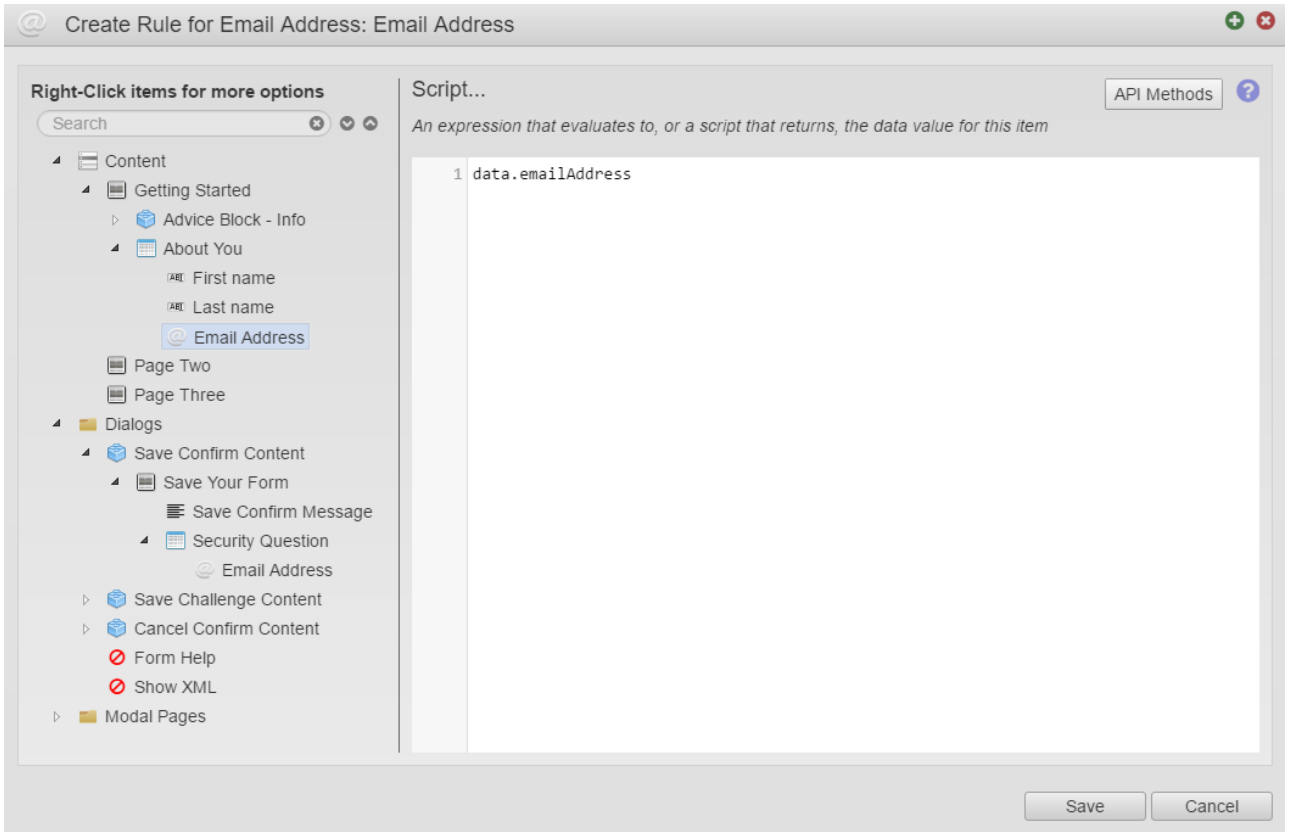
For example, if the answer to the security question is the user's last name, you have probably already asked the user for their last name within the form. You can calculate the field in this dialog to be equal to the same data they have already typed in the form. This makes the save process easier for the user.

To populate security questions from user entered data, follow these steps:

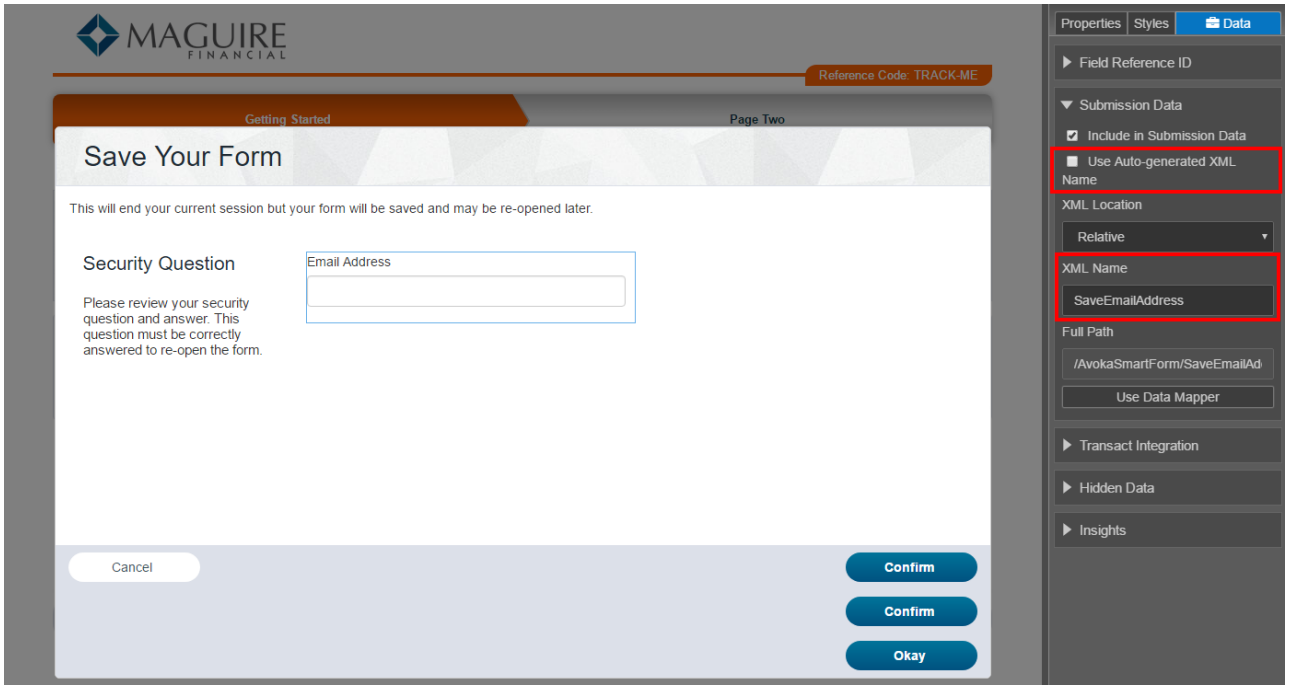
1. Open a Maestro form.
2. Expand Dialogs.
3. Select Save Confirm.
4. Add a component to the security question section within the dialog
   You can copy the component on the form and paste it to the Save Confirm dialog if preferred or you can drag a component from the Palette pane to the Security Question section of the dialog.
5. Map the component as the save challenge.
6. From the Data pane expand Integration.
7. Select Save Challenge from the Form Data Config Mapping dropdown.



8. With the component selected, click Create Rule on the Properties pane.
9. Select Script
10. Navigate to the existing component within the form and double-click to add it to the calculation.



11. Click Save.
12. Select the Integration tab.
13. Clear Use Auto-generated XML.
14. Change the XML Name.



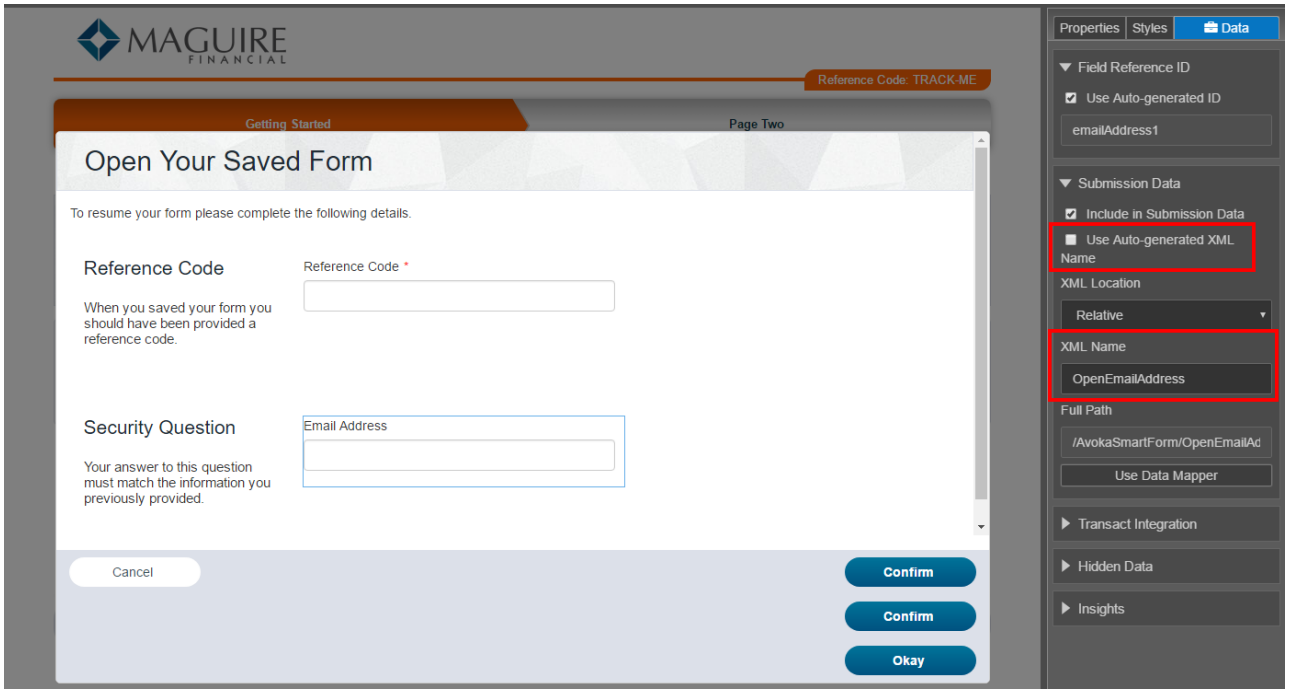
15. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Remember that you cannot have multiple components with the same XML Name. In this scenario you will end up with three components that could have the same XML Name - the component on the form, the component on the Save Confirm dialog and the component on the Save Challenge dialog. You need to ensure that all three components have different XML Names. In most cases you will likely change the components within the dialogs and leave the component on the form as an auto-generated XML name.

Remember you need to add the security question component to the Save Challenge dialog. Rather than recreating the component along with the calculation rule, you may find it easier to copy and paste the component from the Save Confirm dialog to the Save Challenge dialog.

1. Select the Save Confirm security question.
2. Copy the component using the Copy button on the toolbar or type CTRL + C on your keyboard.
3. Paste the component within the Save Challenge security question section.
4. Expand Save Challenge > Dialog Content and then select Security Question.

5. Click the Paste button on the toolbar or type CTRL + V on your keyboard.
6. Select the Data tab.
7. Clear Use Auto-generated XML.
8. Change the XML Name.



Once the save challenge has been configured, you can import it into Journey Manager.

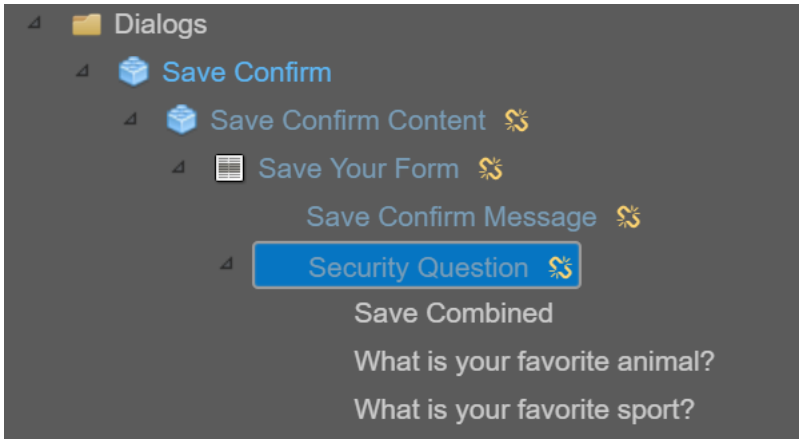
Setting up Security Questions with Multiple Components in the Save Confirm Dialog

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

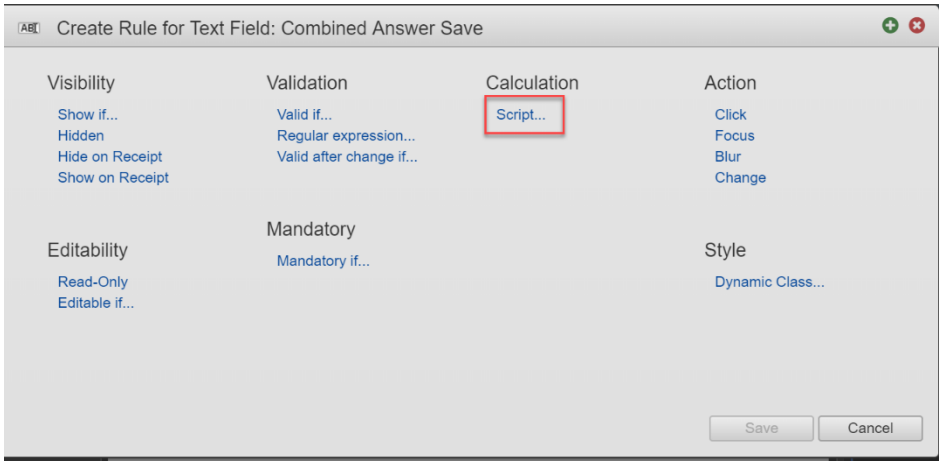
Setting up multiple save challenge security questions in the Save Confirm Dialog is almost identical to setting up a single security question Save Challenge.

To set up multiple security questions in the Save Confirm Dialog, follow the steps below:

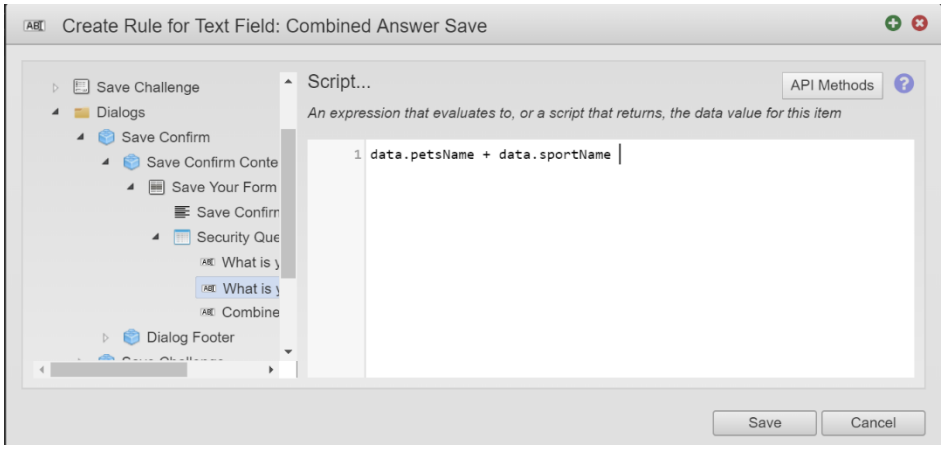
1. Expand Dialogs.
2. Select Save Confirm >Save Confirm Content > Save Your Form > Security Question.
3. Add multiple components to the security question section within the Dialogs section (for example, you could add multiple text fields and use the labels as questions).
4. Add a separate data field to the security question section (name the text field something similar to Save Combined). This data field will be the hidden field that will collect the answers to all the security questions. The screenshot below displays the tree once the data field and text fields have been added.



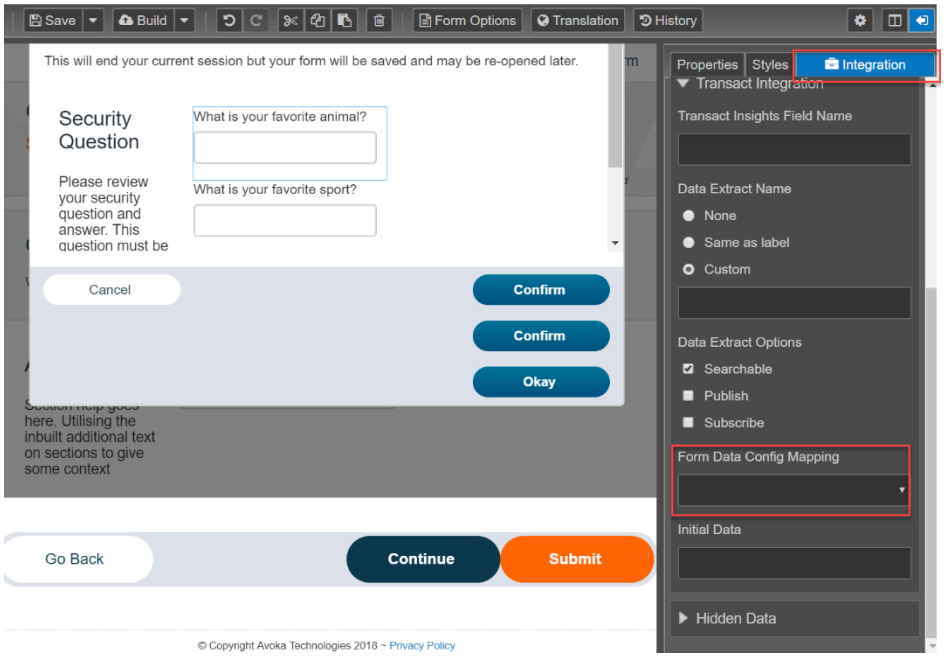
5. Create a calculation rule on this data field to concatenate the answers to the questions.
6. Select the data field Component.
7. Navigate to the Properties tab and scroll down to Rules.
8. Select Rules.
9. Click Create Rule.
10. Select Script... from the Calculation rules List.



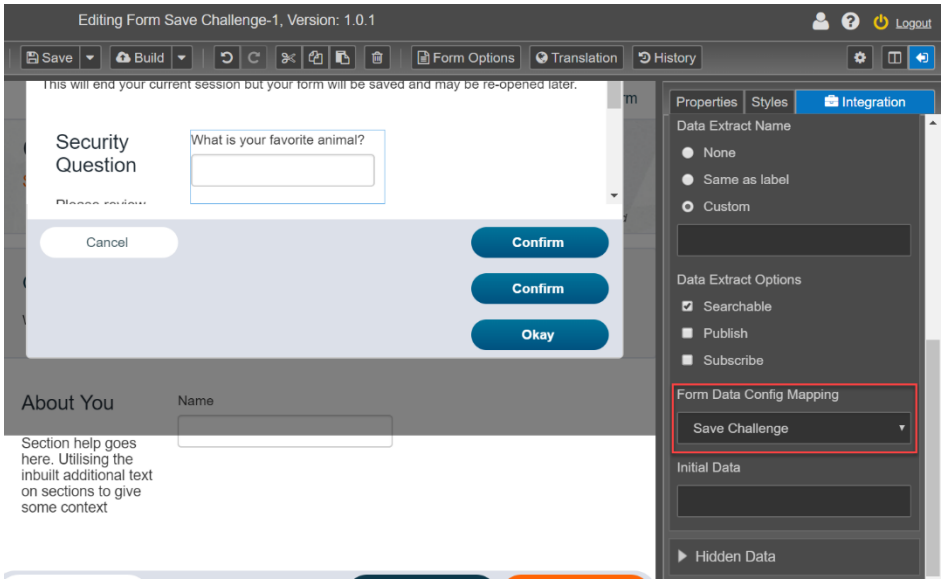
11. Create a concatenation calculation script by double-clicking the first security question and adding an additional (plus) sign and then double-clicking the second security question (this process will vary depending on the number of security questions that you implement). The example below displays the concatenation rule.



12. Click Save.
13. Map the data field as the Save Challenge (by following the steps below). Mapping the data field ensures that Manager knows what to search and compare when a form user answers each question.
14. From the Data pane, expand Transact Integration.



15. Select Save Challenge from the Form Data Config Mapping dropdown menu.



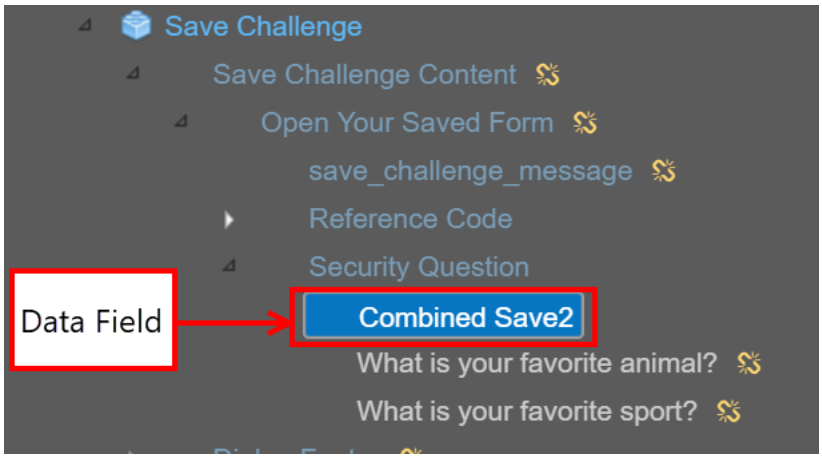
16. Once you have mapped the data field to be the Save Challenge, ensure that the Include in Submission Data checkbox has been checked. This checkbox is found on the Integration tab.
17. Once you have mapped the data field as the Save Challenge, you can move on to configuring the Save Challenge Dialog.

Configure Security Questions for Multiple Components in Save Challenge Dialog

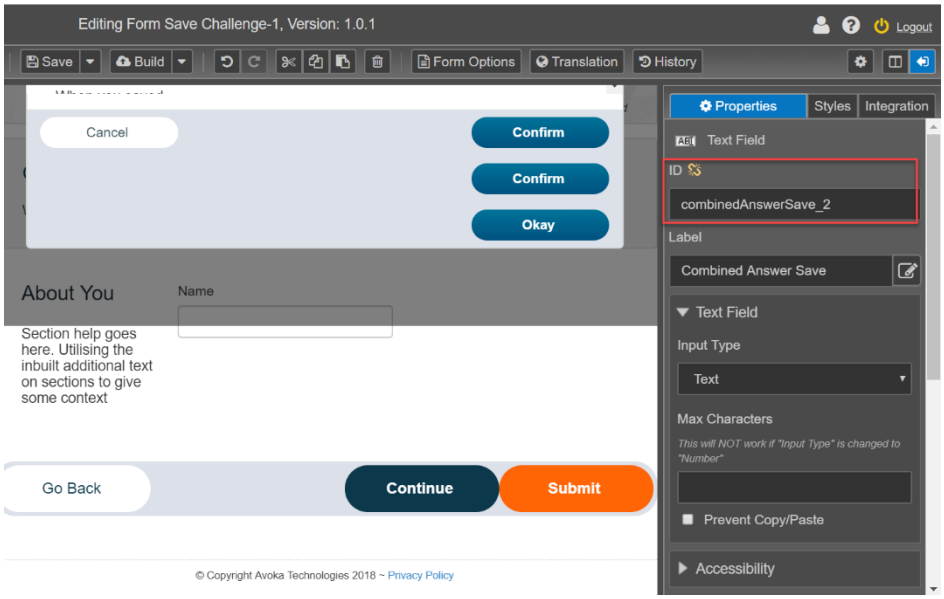
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Follow the steps below to set up a save challenge with multiple security questions in the Save Confirm Dialog.

1. Open a Maestro form.
2. Expand Dialogs.
3. Select Save Challenge >Save Challenge Content > Open Your Saved Form > Security Question.
4. Copy the components you configured in the Save Confirm Dialog using the Copy button on the toolbar or CTRL + C (copy the question text fields, along with the hidden combined answers data field).
5. Paste the components into the Save Challenge security question section (the Save Challenge security question does not need to be mapped to the Save Challenge System Mapping). When copying these components, it is essential that the data field is the first field in the tree. The data field should be the first field as this will ensure that the data field is the field that Manager will collect and compare with the originally entered answers. The screenshot below displays and highlights the data field as the first field in the tree.



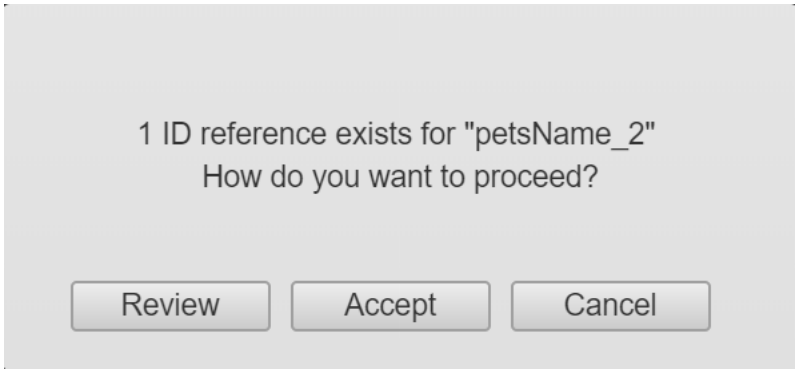
6. Change the IDs of the pasted text fields and data field. Select the component, then navigate to the Properties tab and change the ID in the ID field.



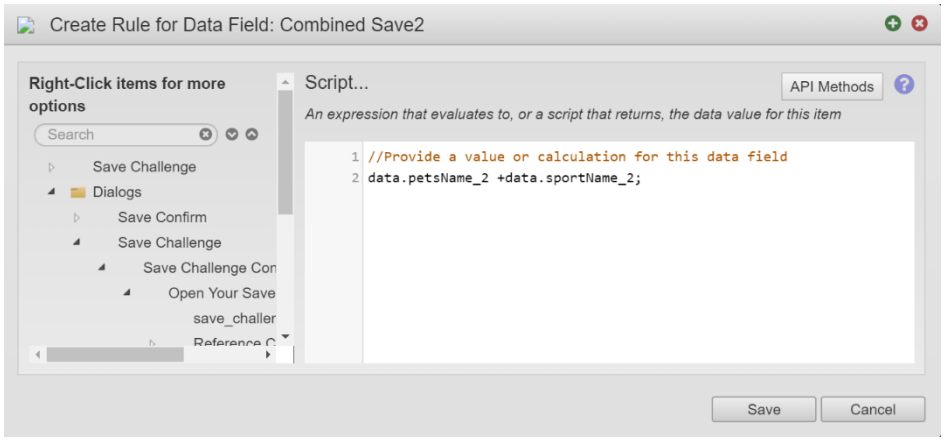
NOTE

It is recommended that you change the ID of all security question fields AND the ID of the data field. Changing the IDs will update the XML Name which will prevent any duplicate binding errors.

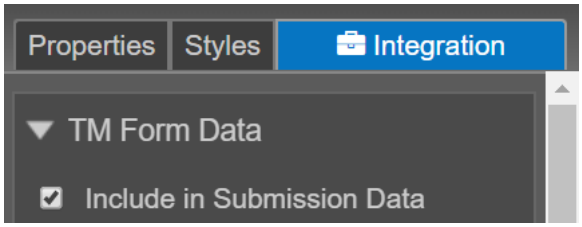
When you change an ID of a text field that is included in a calculation rule, Maestro will warn you that the ID you are attempting to change is currently referenced somewhere within the form. In this case, the one ID reference is coming from the calculation rule that was configured in the Save Confirm Dialog. The screenshot below displays this warning.



7. Ensure that the calculation rule that was copied from the Save Confirm dialog was updated to represent the new IDs. Do this by double-clicking the calculation rule. The screenshot below displays a simple calculation rule that concatenates the answers to two security questions.



8. Ensure that the Include in Submission Data has been checked. This checkbox is found on the Integration tab. The screenshot below displays the Include in Submission Data check-box.



Populate Multiple Security Questions from User Entered Data

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Though it is quite useful to configure a save challenge that ask users questions that are completely removed from the application experience, it can also be useful and – more convenient, to ask users security questions based on information that they previously entered in the form. For example, to return to a saved form, a user must provide the email address they entered when they were originally completing the form.

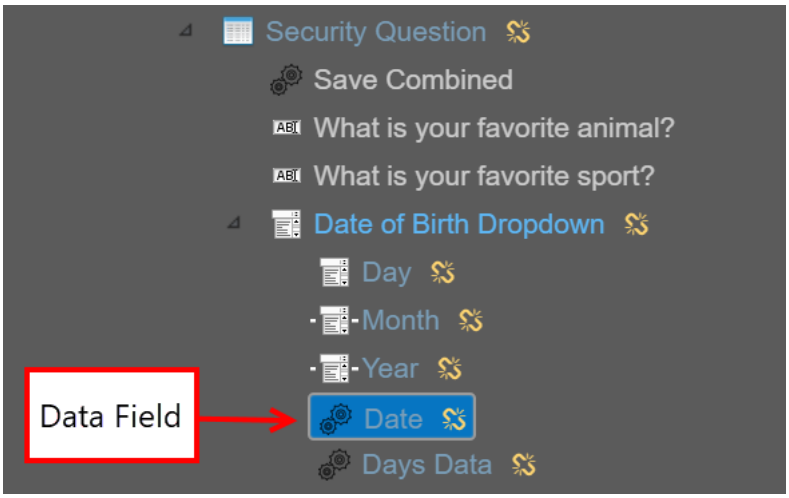
To populate multiple security questions, follow these instructions, except you will need to create change rules on each of the components that will work as the answers to the security questions. These change rules should be applied when configuring the Save Confirm Dialog.

```
if(!Util.isBlank(data.lastName)) {
	data.lastName1 =data.lastName;
	if(!Util.isBlank(data.day)&&!Util.isBlank(data.month)&&!Util.isBlank(data.year)) {
		data.day1 = data.day;
		data.month1 = data.month;
		data.year1 = data.year;
		data.date1 = data.date;
		data.daysData1 = data.daysData;
	}
}
```

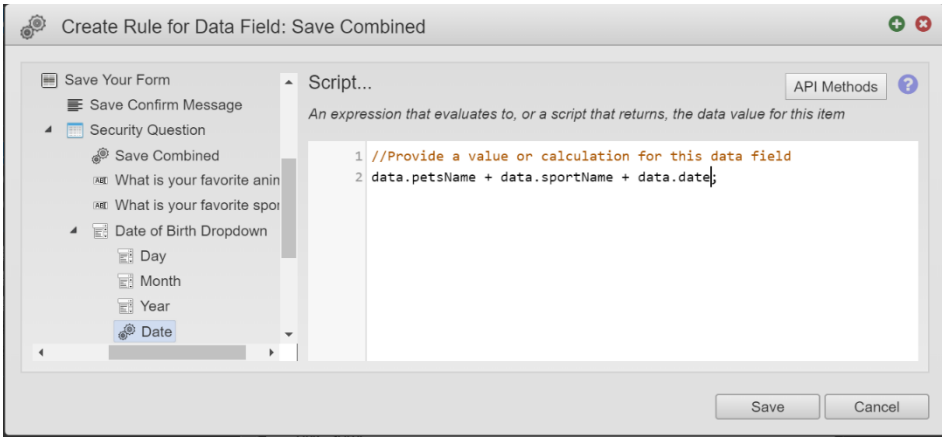
Save Challenge for Multiple Components Best Practices

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

When working with a Date of Birth component (or any compounded component) in a save challenge, it is important that you use the pre-configured data field to create the concatenation calculation rule. The screenshots below highlight that date data field that comes pre-configured for all Date of Birth components and show how this data field can be referenced in a concatenation calculation rule.



Enter the following JavaScript.



Maestro Business Rules

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A Maestro business rule is JavaScript code that is invoked by different events occurring in a Maestro form. Business rules have access to some form data and can obtain more information from Journey Manager. Maestro allows you to create and configure business rules to use in your Maestro forms. Business rules can originate from any of the following sources:

- Maestro's core engine firing life-cycle events, load and pre-submit.
- Visual components firing the standard rule types:
  - Visibility, Editability
  - Validation, Validate on Change
  - Calculation
  - Mandatory If
  - Click
  - Blur
  - Focus
  - Change
  - Dynamic Class
- Responses from calls to dynamic data services
- Custom rules defined by Native Components
- Your own rule code calling Form.fireRule

When a rule is invoked, its JavaScript code receives parameter objects:

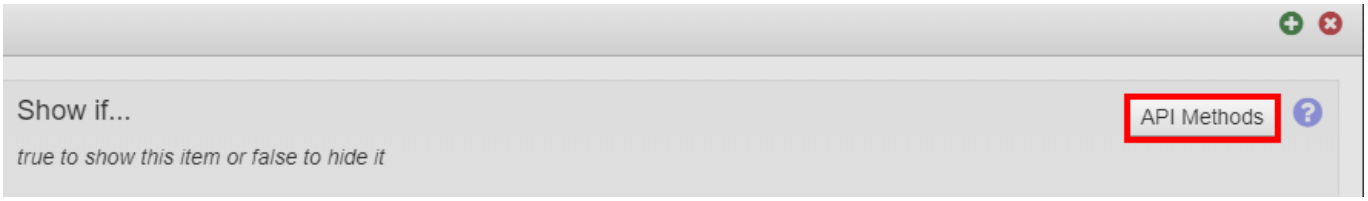
1. data
   - for non-repeated items, this is the form data object which holds values for all form fields plus additional system-related data.
   - for repeated items, this is the repeating object instance which holds all sibling repeating values, including any nested repeats, plus other properties such as the current index, the repeated data object array and the parent item.
2. item - Object holding the structure and definition of the form field including style and 'properties' properties, as well as repeat-aware `getData` and `setData` methods to get and set the form field's data.
3. info - Object describing the event which triggered the rule, or the object returned from a success or failure call in a remote data handler.

In addition to the parameters, there is a local 'value' variable that is set to the specific field instance value in the form data object. The 'value' variable is repeat-aware, so it represents either the sole data value, or the indexed data value in a repeat. Changing 'value' does not change the form data object, so it can be considered read-only.

The rule execution context also exposes several other objects which contain API methods you can call. These objects include:

- Form
- Util
- Calc
- DynamicData

The Maestro Rule Editor provides a quick access to the full method API documentation. Click API Methods as shown below.



To learn how to create a rule, see Creating Rules.

If you experience issues with your created business rules, see debugging business rules.

Types of Maestro Business Rules

Maestro comes with several pre-configured business rules types:

- Calculation Rule
- Visibility Rules
- Editability Rules
- Validation Rules
- Mandatory Rules

Introduction to Maestro Business Rules

MaestroThe UI design product. | Form Builder | Platform Developer

Maestro business rule functions can be created and edited in the Code View or in the rule editor dialog accessed from the Rules section of the properties panel.

Values in Scope within a Maestro Rule Function

All rule functions can access the current data value of the field to which they are attached using the variable `value` (see Shortcuts). In addition to that, they receive several other parameters, injected references and shortcut variables as described below.

Statements vs Expression

All Maestro business rules get packaged within a function body, providing various parameters and variables described further below. The returned value from this function is the result of the rule. So, for example the following validation rule would result in a Boolean true (meaning validation passed) or an error message string depending on the value of the field.

```
if (value > 17) {
    return true;
} else {
    return "Not old enough";
}
```

However, for a rule that only returns one calculation, you can enter a simple expression and omit the `return` keyword. It will be added into the generated function automatically. So the same rule could be expressed simply as follows:

```
value > 17 ? true : "Not old enough";
```

Due to the semantics of the JavaScript OR (`||`) operator, this can be simplified further:

```
value > 17 || "Not old enough";
```

The general rule of thumb is that if the result can be expressed as a simple calculation then it doesn't need the `return` keyword. If it requires control flow, nested functions or any other kind of statements then you should use `return` where appropriate.

Parameters

Maestro business rule functions receive three parameters:

- **data** - The data context that the rule operates in. For items that are not in a repeat, this will be the main form data context. Each instance of a repeat creates a new data context. Instances of nested repeats have a separate data context within the data context of their parent repeat, and so on. This can be used to look up other items of data, in the same context, or in parent, sibling or child contexts.
- **item** - The view item that this rule is attached to. This can be used for looking up properties of the item.
- **info** - An extra parameter that some component-specific rules use to supply additional information. For example, the Page Controller component Change rule has an info parameter with `from` and `to` properties that identify the pages being navigated from and to. To inspect the value of the info parameter for a specific rule in the JavaScript console add the following line:

```
console.log(info);
```

Navigating the Data Model

As explained above, a new data context is created for each instance of a repeat. To navigate to the parent context use `data.$p`. To navigate to the repeat item's own data entry in the parent context (which will be an array of all the data contexts in that repeat) use `data.$r`. To find out a data context's index within a repeat, use `data.$i`.

So, to get to a sibling context - say the previous instance in a repeat - you would first navigate to the repeat's data entry and then use square-brackets to address the relevant child context in the array

```
data.$r[data.$i - 1];
```

Navigating the View Model

If the item for which the rule is defined is in a container, you can access that container item using `item.$parent`.

Within any given container item, the child items are held in an array of rows, each of which is an array of child items within that row. So the first child of a containers first row would be `item.rows[0][0]`.

In this way you can navigate up, down and across the view hierarchy. However, to get to a specific item it's easier to look it up in the flattened map held in `Form.items`, such as

```
Form.items.myRadioButton;
```

The properties of each item are available on the key `properties`. So to get the placeholder text of a Text Field with ID `firstName` you would use

```
Form.items.firstName.properties.placeholder;
```

Global API Objects

In the previous section we saw the a reference to an object called `Form` - in that case to access its `items` property, containing an ID-keyed map of all the view items. There are several of these global objects. (They are not global in the sense of being available on the `window` object, but they are injected into the scope of each business rule function using Angular.) These objects (or *"Services"* in the Angular terminology) are described below. A full reference of API methods available on each of them can be found using the API Methods Reference link in the Code View.

`$rootScope`

The Angular `$rootScope` object - can be useful if you are an Angular expert. See the Angular documentation.

`$q`

The Angular `$q` object. This can be used for creating and aggregating Promises. See the Angular documentation, as well as general documentation about JavaScript Promises.

`$timeout`

The Angular timeout function. This is similar to JavaScript's standard setTimeout function, but ensures that any data updates are rendered again in the form. Also, it can be used with a call-back, as follows:

```
$timeout(function() {
    data.status = "Success";
}, 5000);
```

or it can be used to return a prommise, as follows:

```
$timeout(5000).then(function() {
    data.status = "Success";
});
```

Form

The Form service contains a number of properties and methods for general form interactions such as validation and submission.

Calc

The Calc service provides some useful methods for running calculations on data.

Resource

The Resource service provides methods for working with external resources, such as formatting image URLs.

Util

The Util service provides general utilities to assist with tasks such as searching and sorting arrays, walking the row/child view hierarchy and so on.

DynamicData

The DynamicData service provides methods for interacting with Dynamic Data Services published in Transact Manager.

Scroll

The Scroll service provides a way to scroll to and focus on particular form fields.

Insights

The Insights services enables rules to register events with Transact Insights analytics.

Translation

The Translation service has some helper methods for working with multi-lingual forms, although for basic access to localized terminology use the `T` shortcut described further below.

Other Shortcut Variables

value

This is a shortcut to the current data value for the field to which the rule is attached. So for a rule on a Text Input with ID lastName it is the equivalent of using `data.lastName`.

Rules

In the final published form, all the business rules are contained within a single Rules service. This shortcut lets you navigate to and even run business rules from within other business rules.

Business rule functions have names formatted as follows:

```
<rule-type>_<item-id></item-id></rule-type>
```

Basic standard rule types are `ok` (validation), `us` (editability), `sh` (visibility) and `eq` (calculation). Other action rule types have their full action name, such as `click`, `blur` and `change`. Components can also provide their own custom rule types.

So, the click rule on a button with ID `verify` could be accessed with

```
Rules.click_verify;
```

and the validation rule for a Decimal Field with ID `age` would be

```
Rules.ok_age;
```

Whilst you could call the age validation rule from another rule in the same data context with the code

```
Rules.ok_age(data, Form.items.age);
```

It would only return the validation state of that field. To properly validate the field, including display of approriate errors and setting form validation state, you should use the validation methods available in the `Form` service object.

T

This is the hierarchical map of all terminology items for the currently selected language in a multi-lingual form. So, to get the translated placeholder text for an item with ID firstName you could use

```
T.firstName.properties.placeholder;
```

SystemProfile

This is a shortcut to the SystemProfile node of the form's data, which contains metadata about the submission provided by Transact Manager.

Job

This is a shortcut to the Job section of the forms SystemProfile data. Useful for some collaboration job scenarios.

Create a Maestro Business Rule

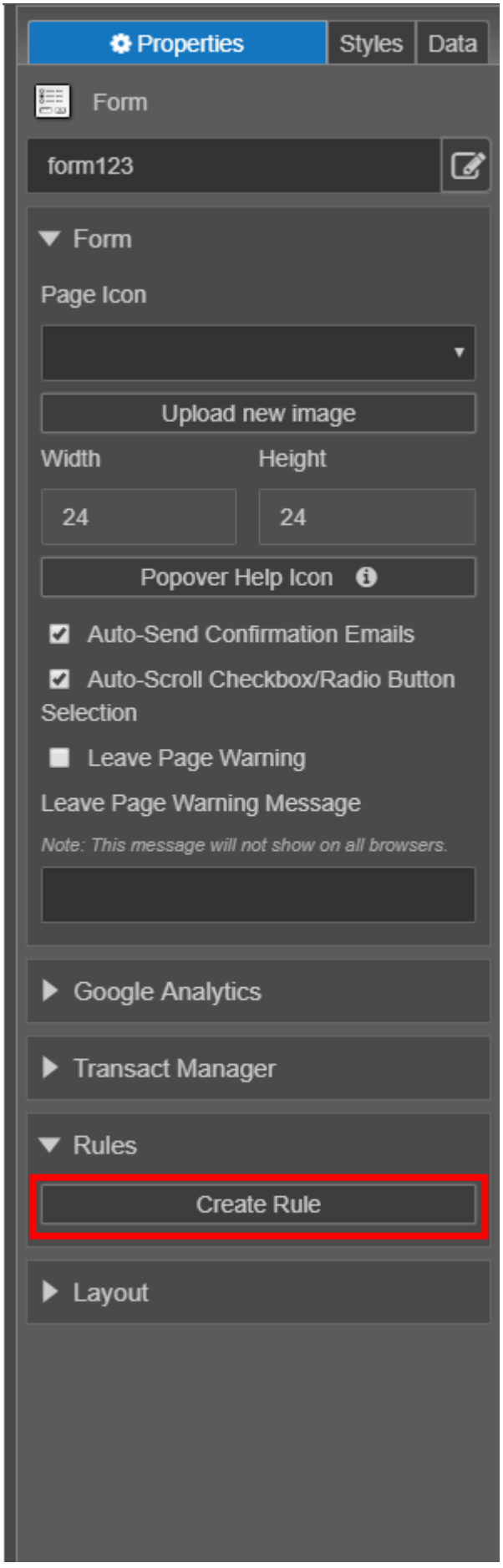
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to create various business rules to implement a desired functionality into a Maestro form.

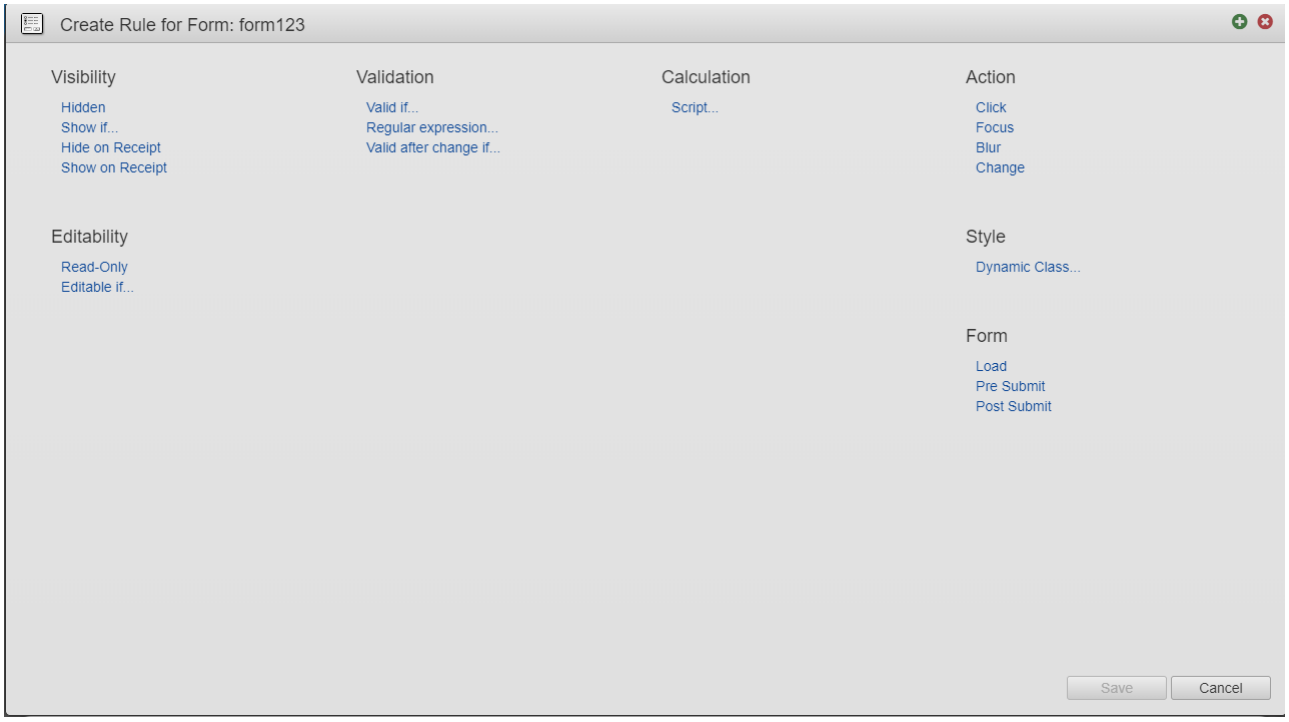
To create a business rule in a Maestro form, follow these steps:

1. Open a Maestro form in the Maestro editor.
2. Click the Properties tab.

3. Click Create Rule as shown below.


Properties
Styles
Data
Form
form123
Form
Page Icon
Upload new image
Width
Height
24
24
Popover Help Icon
Auto-Send Confirmation Emails
Auto-Scroll Checkbox/Radio Button Selection
Leave Page Warning
Leave Page Warning Message
Note: This message will not show on all browsers.
Google Analytics
Transact Manager
Rules
Create Rule
Layout

4. Select the type of rule you want to create from the Create Rule dialog.



5. Configure the rule.
6. Click Save to create the rule or Cancel to discard it. You can only create one rule of each type on each form field.

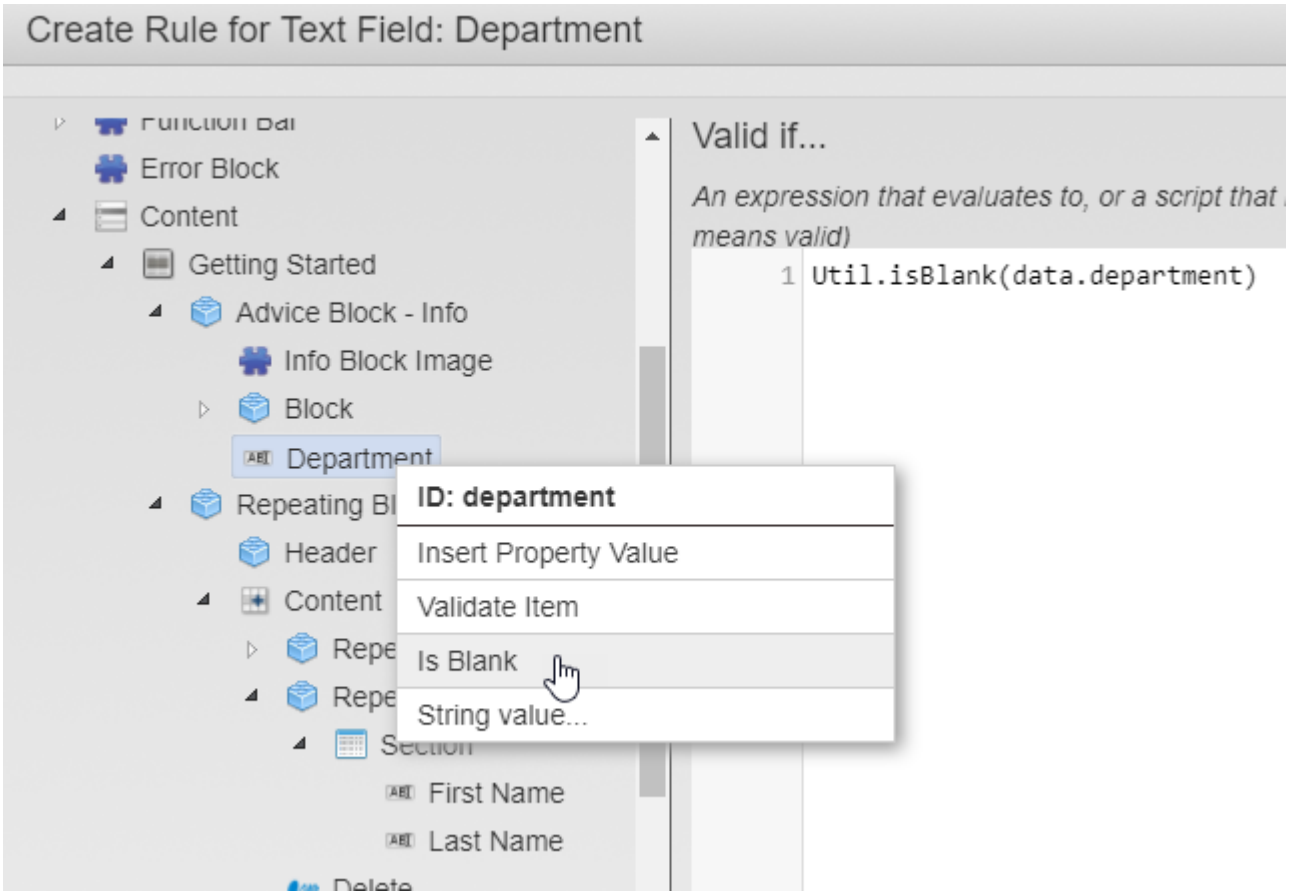
Configure a Maestro Business Rule

Once a business rule has been created for a Maestro form, you can to write some JavaScript to implement the rule. You can write it yourself, or you can use one of several Rule Helpers to create typo-free code. Rule Helpers are available inside the Rule Editor and allow you to pick fields from a tree structure, and then generate the chosen field name, or associated method calls into the editor source.

Double-click on an item in the tree to insert the field's data reference into the code pane where the cursor is currently located.

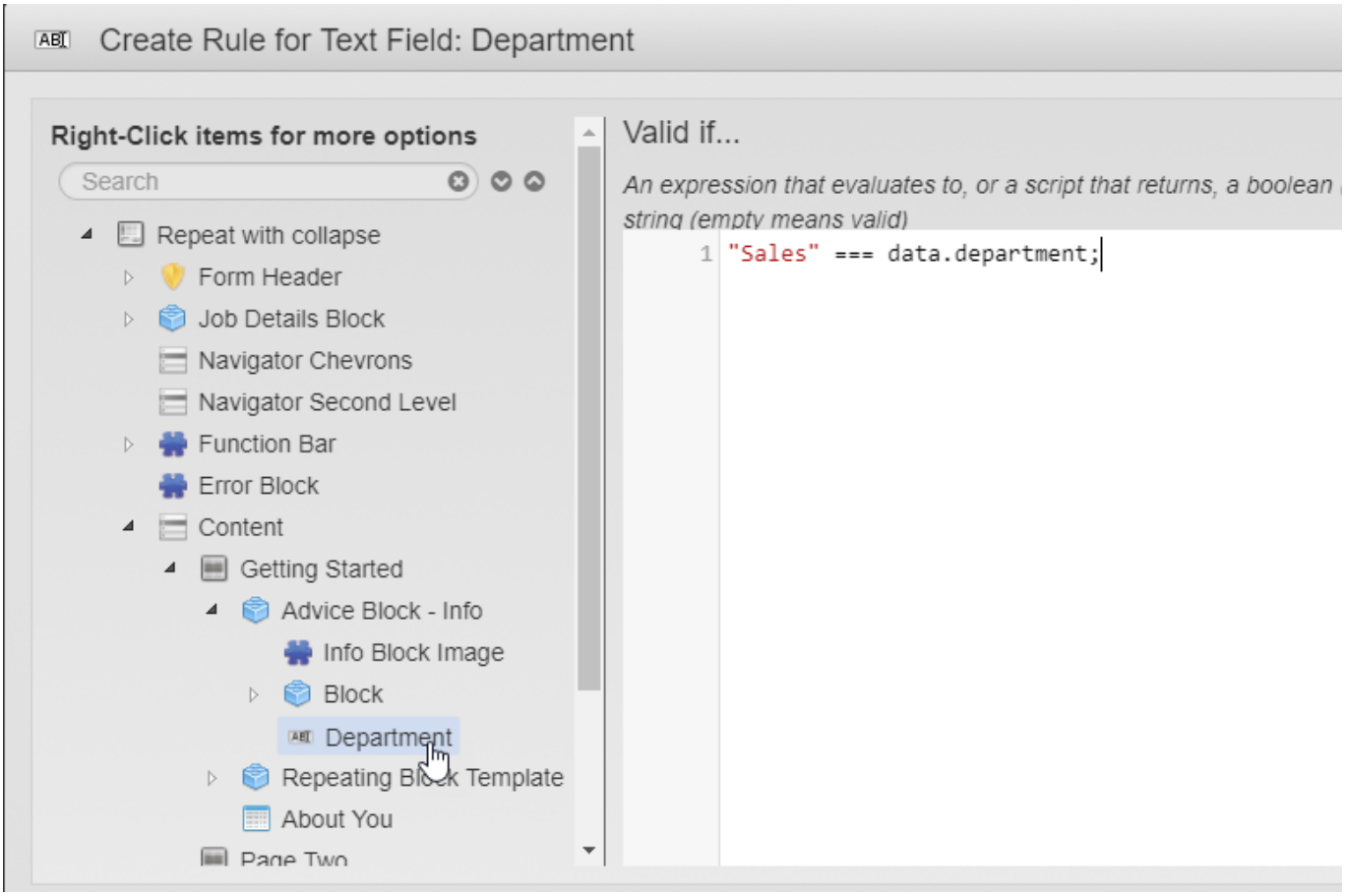
Right-click on an item in the tree to bring up the list of rule helpers applicable to that item type. Then select from the list to insert that templated rule code into the source, with the item data reference contained within it.

A Rule Helper used to insert code in the rule source is shown below.



Once generated into the rule source, it is only now necessary to place a '!' in front of the generated code and the non-empty check code is complete.

To illustrate how to insert a 'data.department' text into the JavaScript, double-click the Department field in the tree, so it's added to the code as shown below.



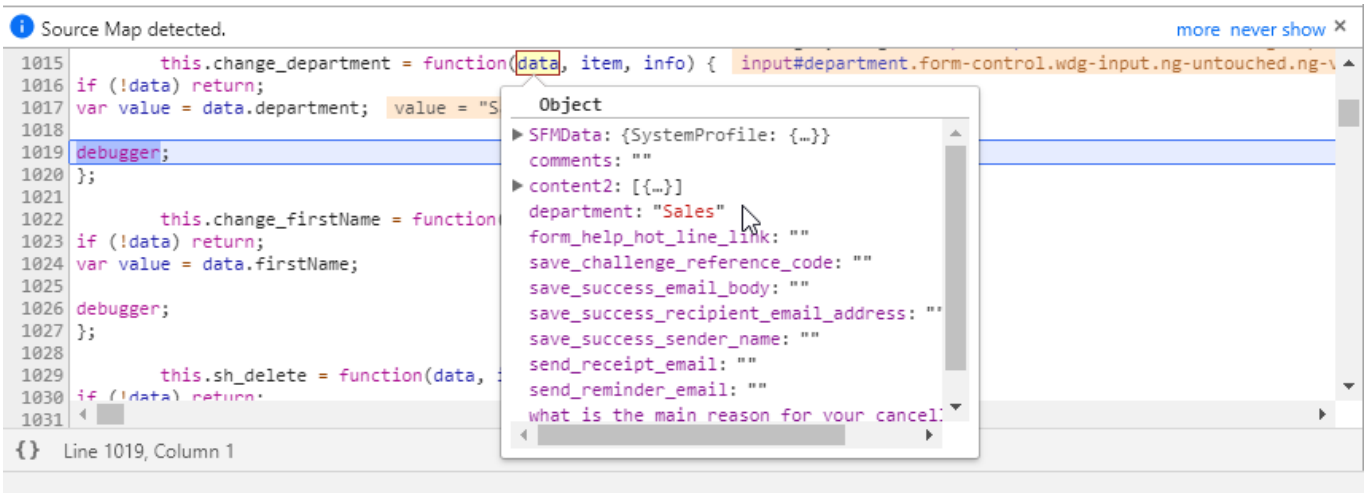
Next, learn about configuring business rules that are invoked on a form load event.

Business Rule Data Object

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A JavaScript, implementing a Maestro business rule, requires a data object [1] parameter, passed to it on its invocation, to perform a desired action.

In the example below, a change rule has been set on a field named `department`. This change rule triggers the debugger. The debugger is inspecting the `data` parameter that has been passed to the change rule and `data` is the form data object which will eventually become the form XML when passed to Journey Manager. Also, the `value` variable has been overridden with `Sales`.

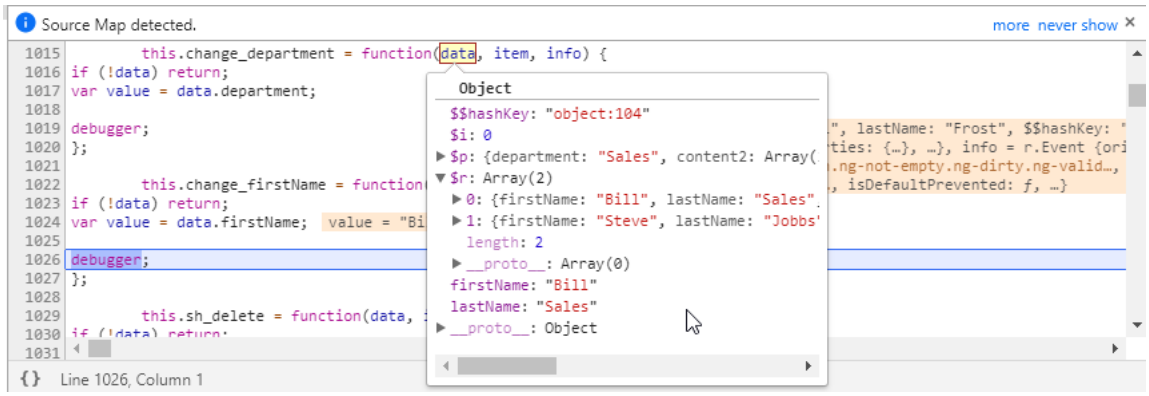


Let's check the data parameter being passed to a change rule for a repeating field called firstName. Notice the rule's 'data' parameter in the inspector is the same object as `data.$r[data.$i]`. This shows that the object `data.$r` holds all the repeated data fields for a repeat, and the object `data.$i` is the index of the repeated field occurrence whose rule is executing. If you are stopped in the rule code of a repeated item in a debugger, enter '`data.$r[data.$i] === data`' without the quotes in the console tab, and the result will be true.

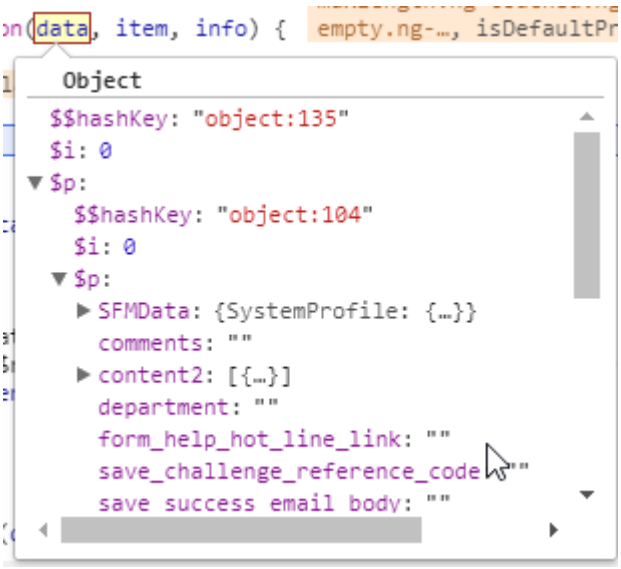
---

[1] 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).

In the example below, `data.$p` is the parent of the repeating object, `data.$r` is the array of repeating rows, and `data.$i` is the index of the affected row. `data.$p` above is the same as the data object in the preceding non-repeat example, since the item whose rule has fired is a member of a top-level repeating item, rather than a nested repeating item. If this item was nested one more level deep, its `data.$p` would point to its parent repeat instance, and that item's `0` would point to the root form data object (Form.data). In this way, you can get access to the full ancestry chain of repeated items.



A 2-level repeat would look like the screenshot below:



Next, learn about business rule item object.

Business Rule Info Object

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A Maestro business rule JavaScript requires an info object parameter, passed to it on its invocation, to perform a desired action.

The `info` parameter is an object describing the event which triggered the rule, or it can be the object returned from a success or failure call in a remote data handler. The value of the info object strongly depends on the rule's implementation. For most rules, the returned info object is undefined.

The info object can be described with the following structure:

```
info = {
       widgetTypes: {},
       images: [],
       fonts: [],
       scripts: [],
       styleImages: [],
       ruleScripts: {},
       extraHtml: {},
       formView: formView
}
```

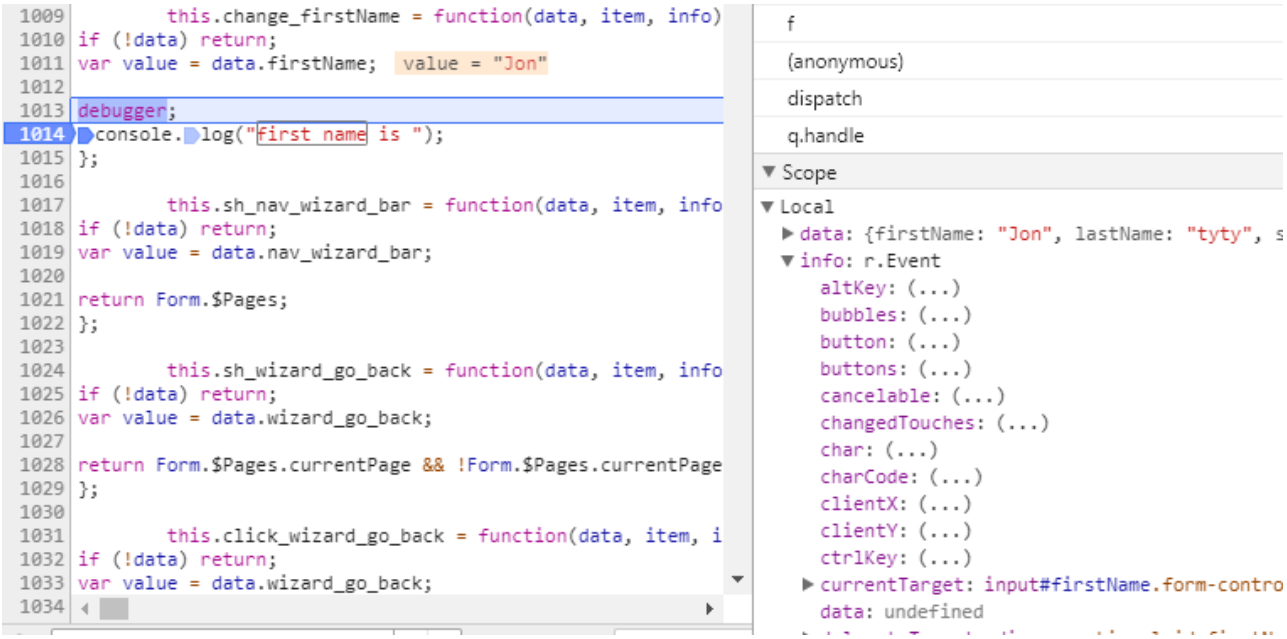
A Form Builder can always check the values of the info object in JavaScript by:

1. Adding debugger; line in the JavaScript code

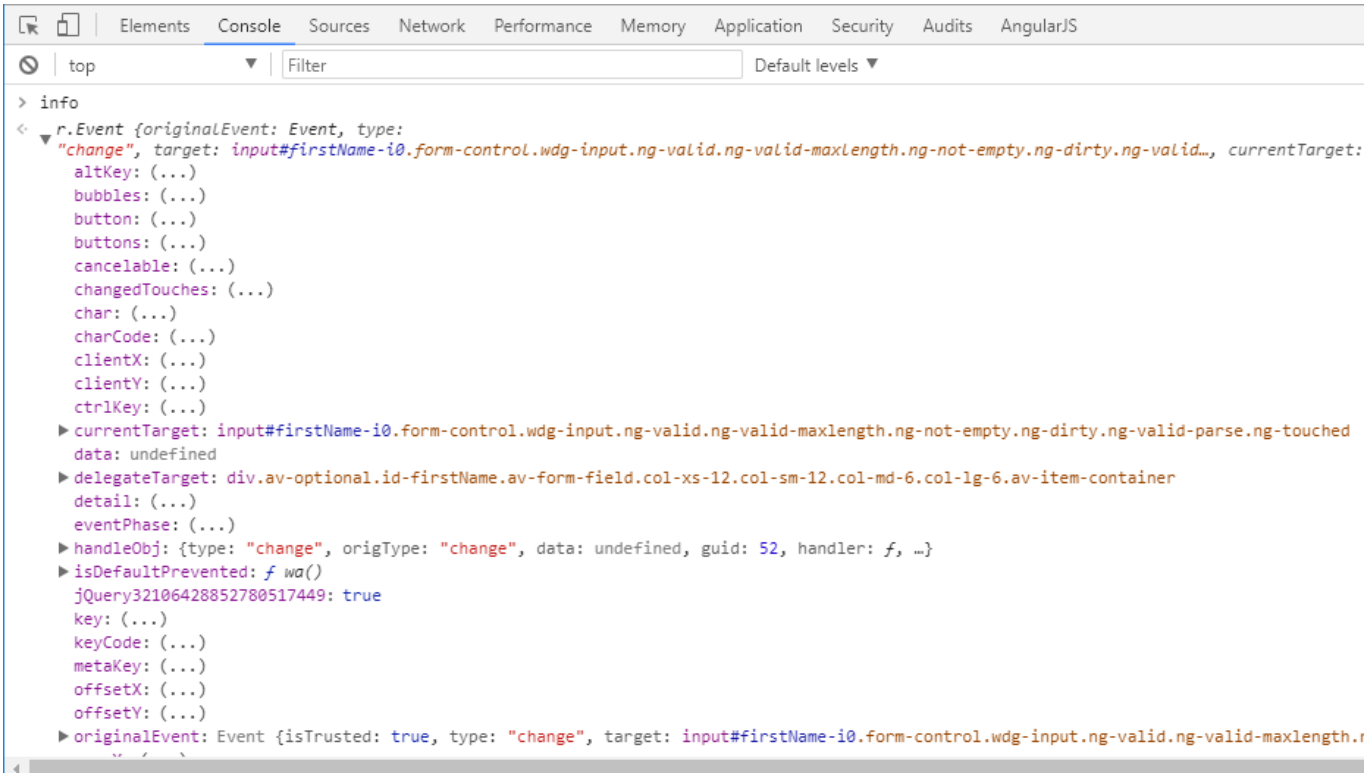
   ```
   info object
   debugger;
   console.log("first name is ");
   ```

2. Building with Options.
3. Navigating the form's property in the Browser with debugging on.

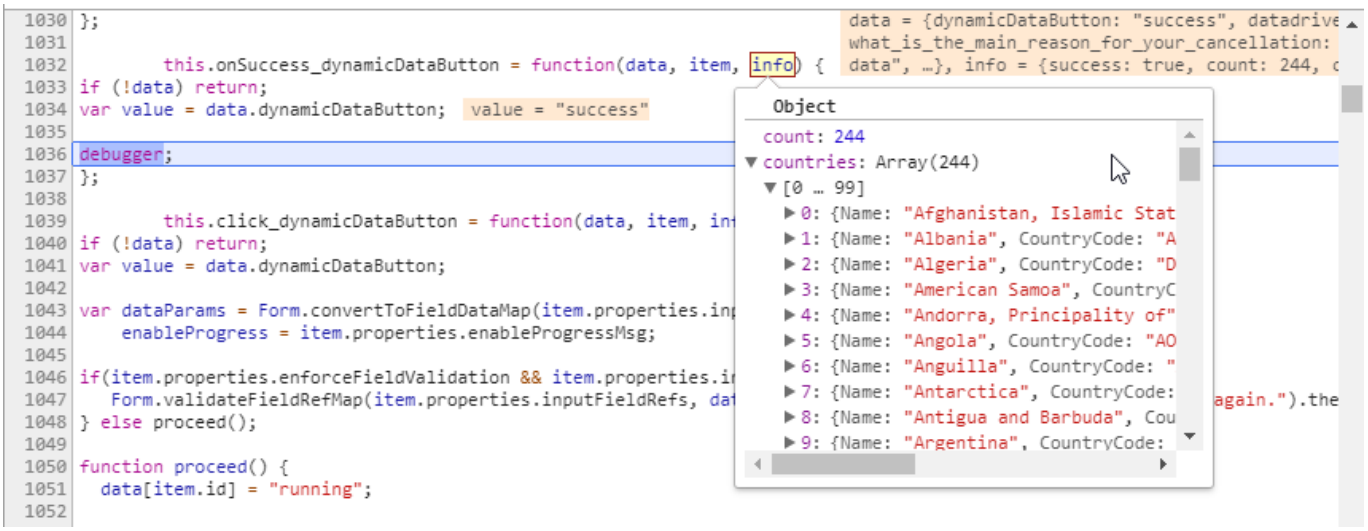
4. Viewing values in the debugger (displayed in the screenshot below).



The screenshot below shows the 'info' parameter expanded in the developer tools console. This event has been triggered by Angular code, which is the framework Maestro forms run within.



The screenshot below shows the info object returned from a dynamic data call. The returned data in this particular case is an array of country names and codes, but it could be anything, depending on the dynamic data service it invoked.



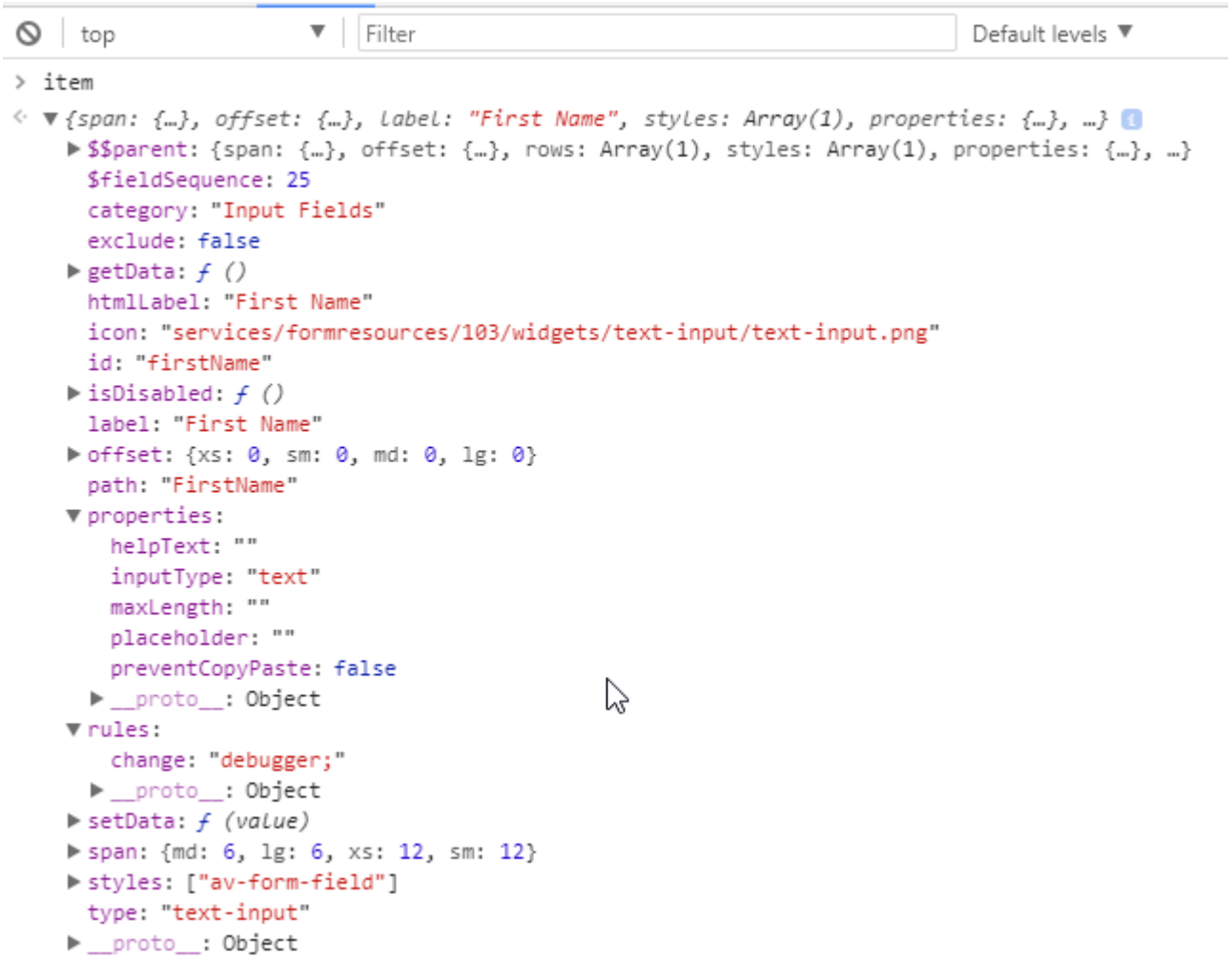
Next, learn about business rule data object.

Business Rule Item Object

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A Maestro business rule JavaScript requires an item object parameter, passed to it on its invocation, to perform a desired action.

The screenshot below shows the 'item' parameter expanded in the developer tools console. It contains among other information, tree structure information in $$parent, item properties in the 'properties' property, style information in the 'styles' property, getData/setData getter and setter methods for the data value, and rule code in the 'rules' array. Note that this example has only a change rule with code 'debugger;' which simply triggers the debugger on a change event.



Next, learn about business rule info object.

Create a Form Load Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to create form load rules, which are business rules that are invoked on a form load event. This event is triggered immediately after a page has been loaded. Form load rules can only be applied to a whole Maestro form, so you can't fire this rule on a single field. Load rules are created slightly differently to the way traditional business rules are created in Maestro.

To create a form load rule:

1. Open a form in the Maestro editor
2. Click Form Options.
3. Select Form Rules and click Create Rule.
4. Click Load the type of rule from the Create Rule dialog.
5. Configure the rule by writing a JavaScript that implements desired logic.
6. Click Save to create the rule.
7. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Next, learn how to how to debug Maestro business rules.

How to Debug Maestro Business Rules

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Chrome developer tools include many useful plugins and extensions that can be used for debugging Maestro forms and business rules. One of them is a debugger, which allows you to set breakpoints in your business rule's JavaScript, or any JavaScript.

To open the Developer tools, click `Ctrl +Shift+l` and click the Console tab.

A quick way to trigger the debugger from inside your own rule code is to include the 'debugger;' statement in your code as you are developing it. When the statement is executed in a rule, the debugger is activated, so you can inspect the values of variables in the debugger, or in the Console tab. Make sure to remove any debugger statements when you have finished debugging.

Tips for Writing Rules

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Rich Text Editor

Sometimes, the rich text editor can become confused about exactly what you are trying to achieve, and can interpret what you are typing in the wrong way. If you are getting unexpected results, you may want to flip to Source mode, and review the underlying HTML. You may have accidentally injected html tags you didn't expect.

Test often

When you are developing formulae, test your work often. Several types of errors can result in your form not displaying correctly in Preview mode. The more often you test, the more likely you are to find the bit of code that is causing the problem.

Use a field within a calculation

A calculation is assumed to be code, and therefore everything is interpreted as JavaScript.

When you insert a field into a calculation, simply reference the field id as follows:
`data.firstname`

If you need to include text, include it in single or double quotes.

For example:
`"Welcome " + data.firstname`

Insert a field into text

A caption or display field is interpreted as text, and so you need special syntax in order to have everything interpreted as code. Maestro uses double curly braces to inject code into the string.

For example, to insert a field value into a piece of text, such as a caption or a display field, use the following notation:
`{{ data.firstname }}`

You can perform calculations within the braces, like this:
`{{ data.firstname + " " + data.lastname }}`

The braces mean: "Treat whatever is inside the braces as a calculation rather than as text.

Hide text display until all dependent values are valid

Sometimes a text display can look strange if only some of the values that are used within it have been completed. To avoid this, simply create a Visibility rule that looks something like this:
`data.firstname != "" && data.lastname != ""`

or
`+data.term != 0 && +data.frequency != 0`

Treating field values as numbers

All data is inherently treated as text. If you want to force a field's value to be treated as a number, prefix it with a "+". (If you don't do this, when you try to add two numbers, they will be concatenated as strings rather than added together. You can also use JavaScript functions to format the result. For example:
`(+data.term * +data.frequency).toFixed(2)`

or
`(+data.term) + (+data.frequency).toFixed(2)`

Treating field values as boolean (true/false)

All data is inherently treated as text. If you want to force a field's value to be treated as a boolean or logical value, prefix it with `!!`. For example:
`if (!!data.is_married) ...`

Conditional values

If you have, for example, some values in a dropdown list, and want to turn these into more human readable values, you could use an if statement or a switch statement. However, if you want to embed the logic inside some text, you can use the following object key lookup syntax:
`{{ {'52': 'per week', '24': 'bi-monthly', '12': 'per month'} [data.frequency] }}`

This is a shortcut for:

Depending on the values of [data.frequency], use the following lookup values:

- '52' = 'per week'
- '24' = 'bi-monthly'
- '12' = 'per month'

This defines an in-line JavaScript object, and then uses the field's data value to index into that object. More information is available at this website.

Note that if there is no match, then the first pair will be used, and so this can be used as a default.

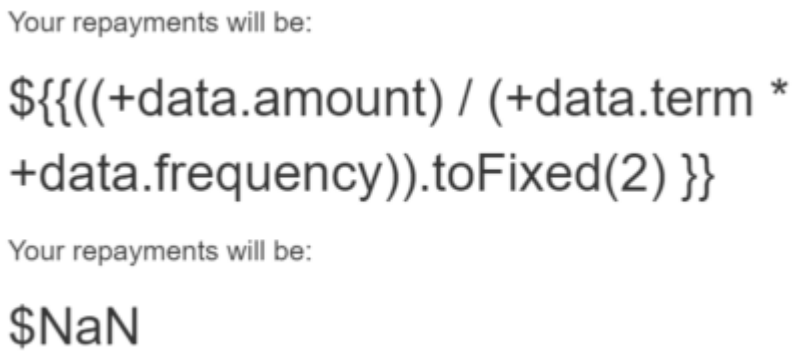
Text Display vs Dynamic Content

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

There are two types of components that can be used for displaying large or small blocks of rich text:

- Text Display
- Dynamic Content

They are almost identical, and work in the same way in Preview mode and the rendered form. The only difference is in Design Mode, the Text Display displays any embedded data in its raw form, whereas the Dynamic Content attempts to evaluate any embedded calculations. For example, this is a Text Display, followed by a Dynamic Content, in Design Mode.



Both fields contain the same calculated rich text. The first field displays the actual calculation text, whereas the second will attempt to evaluate the references (even though these fail in this example, and result in "Not a Number").

It is largely a matter of preference which of the two you should use.

For more information, see BusinessRulesOverview.docx

Calculation Rules

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to create various calculations rules, which you can add to components and fields of a Maestro form. The calculation rules are part of the business rules and currently consist of the Script rule.

Maestro adds calculation rules to a form in the following order when:

- Transaction XML data is loaded to a field on the form, so the rules will run before the Form Load rule is executedw
- A field is made visible
- A value of another field, that this rule depends on, changes

It is important to understand the order of rules execution to prevent empty values used in calculation rules so they don't break the rule logic or cause an exception.

You can use the Script rule in several distinct ways:

Mathematical Calculation

Mathematical Calculations allow you to create simple and complex mathematical calculations (+, -, *, etc.). Calculations can be created using data entered in components on the form, or by using defined numerical values. Mathematical calculation rules can be created in repeating blocks.

Duplicate Data

Duplicate Data from Another Component allows you to populate components in the form using data entered in another components.

Preset Data

You can preset some field's values as defaults on form load using a calculation rule. For example, you can select a default checkbox option when the form loads by setting its value to `true`, if the value is blank, using the following JavaScript:

```
if (Util.isBlank(data.checkBox)) {
    data.checkBox = true
}
```

It's important to note that we check whether the value is blank or not before setting its value, because if you open the form again after saving it when you have unselected the checkbox, the rule will set the default value again, as well as that of on a receipt.

NOTE

You can also set the value to `true` by using the Form Load rule, the Integration pane, or in the initial XML file, which Manager uses to prefill a form. The later is the easiest and most reliable option.

Concatenate Multiple Components

Concatenate Multiple Components allows you to take data entered into multiple components and combined the entered data into one component. An example of this is asking for a First Name and Last Name and combining them into a single Full Name component.

Next, learn how to create a mathematical calculation rule.

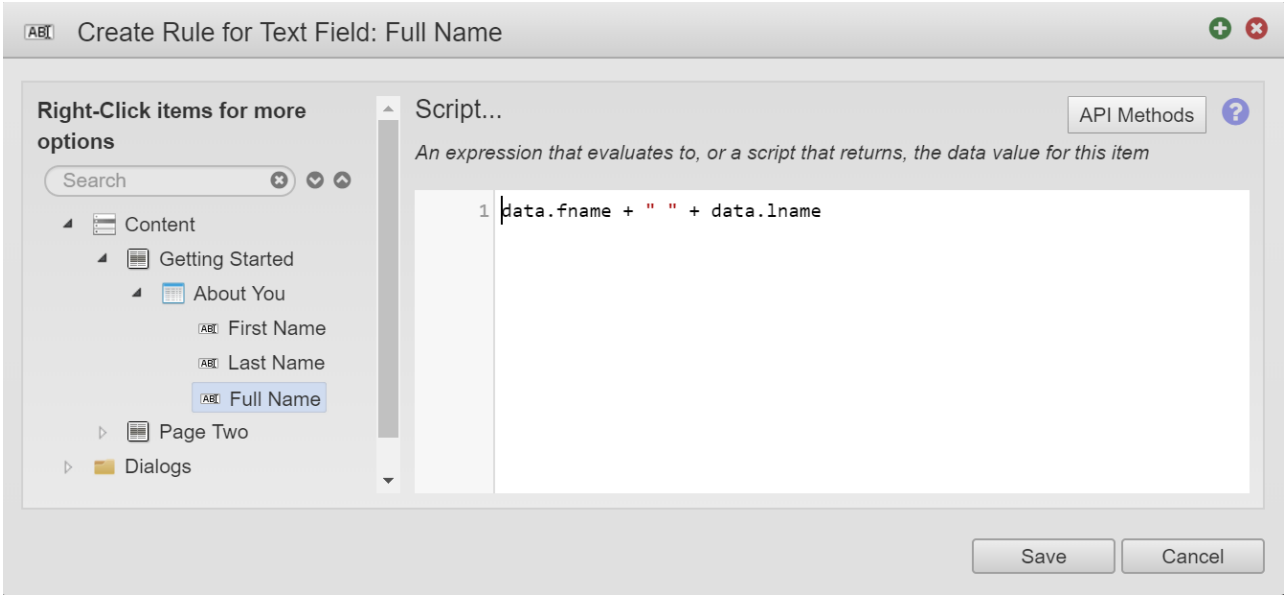
Concatenate Multiple Components Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Use the Concatenate Multiple Components calculation rule option when you want to combine multiple components and display them in a single component. The Concatenate Multiple Components option allows you to combine the value from more than one component into a different component. For example, you may have text fields for First Name and Last Name and you want the entered values from both to display in the Full Name field.

To create a concatenate multiple components business rule for a Maestro form, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select the component where you want the result to be displayed.
4. Click Create Rule.
5. Select Script under Calculation.
6. Expand the form structure as needed and double-click the component with the entered value you want to display in the select component.
7. Add + " " + sequence, which actually concatenates multiple components. After the plus sign we will enter a space surrounded by quotation marks along with another plus sign. You may use either single or double quotes as long as you are consistent. The plus sign concatenates when applied to text input and adds when applied to numbers. The " " adds a space between the combined components. The quotations need to be around any characters, including any spaces that you want to display in the resulting component.
8. Double-click another component you want to add to the selected component.
9. Repeat steps 7 and 8 until all components are added.



10. Click Save.
11. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

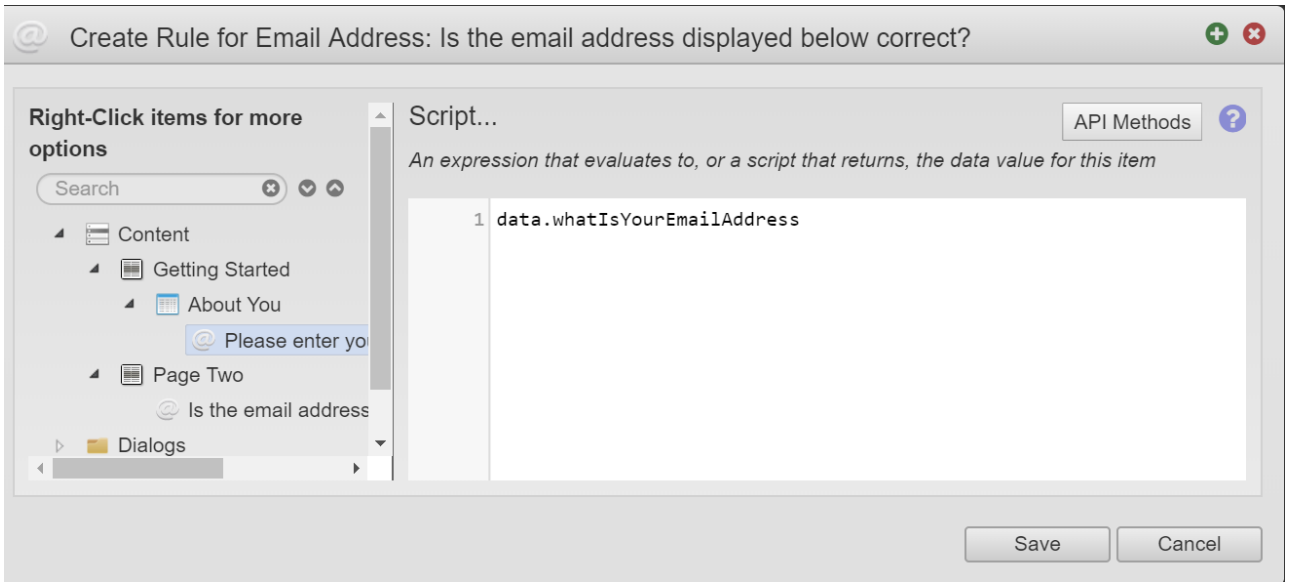
Duplicate Data Calculation Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Use the Duplicate Data calculation rule to display some sort of data that was previously collected and display it in a different place within the Maestro form. This option is commonly used to display a form user's name after they have entered their personal details into the form. The Duplicate Data option allows you to use data from different components in other parts of the form. For example, you could use an entered value such as first name in a text display. This allows you to dynamically customize areas of the form. You could also re-use entered data so that the form user doesn't have to re-enter data multiple times.

To create a duplicate data calculation rule, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select the component where you want the entered value from another component to be displayed.
4. Click Create Rule.
5. Select Script under Calculation.
6. Expand the form structure and double-click to add the component with the entered value you want to display. For example, the form user has been asked to enter their email address into the form. Once the user has entered their email address, and clicked continue, the form displays the email address and asks the user to confirm that the entered email address is correct.



7. Click Save. The script will be added to the window by coping the entered data from one component to another.
8. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

A calculation rule has been added to a second email address component, this component will be used to confirm that the entered email address is correct without the user having to enter their email address more than one time.

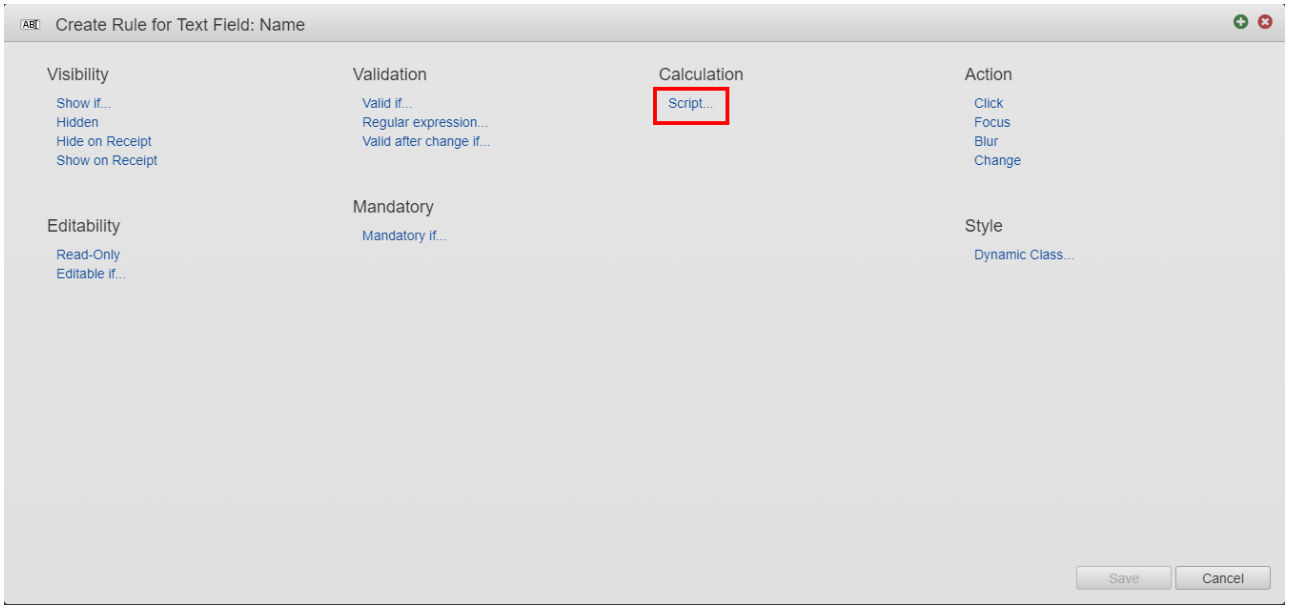
Mathematical Calculation Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

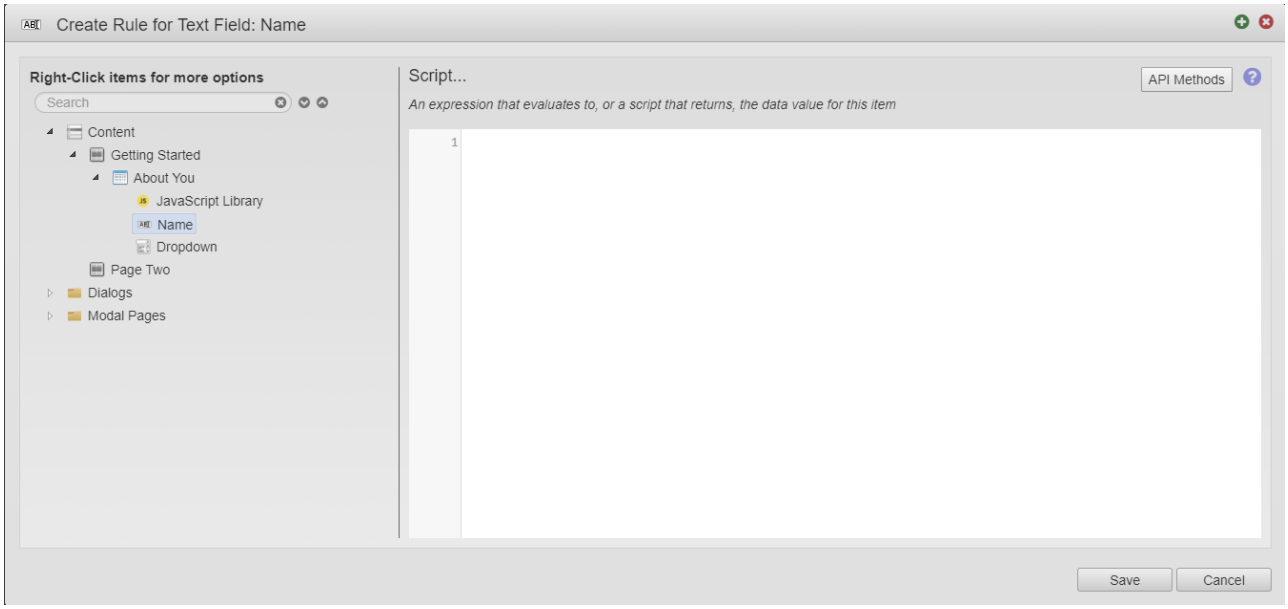
The Mathematical Calculation business rule option allows you to create simple and complex mathematical calculations (+, -, *, etc.) for a Maestro form. Calculations can be created using data entered in components on the form, or by using defined numerical values.

To create a Mathematical Calculation rule for a Maestro form, follow these steps:

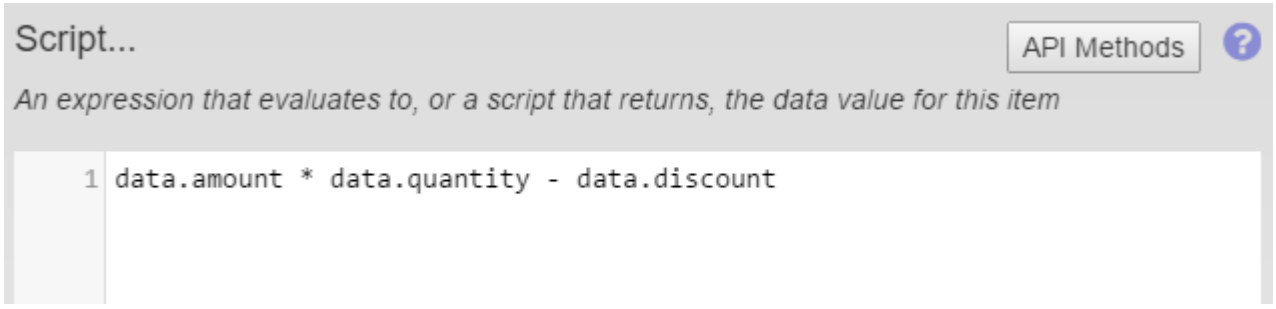
1. Open a Maestro form.
2. Click the Properties tab.
3. Select a component that displays a calculation result.
4. Click Create Rule.



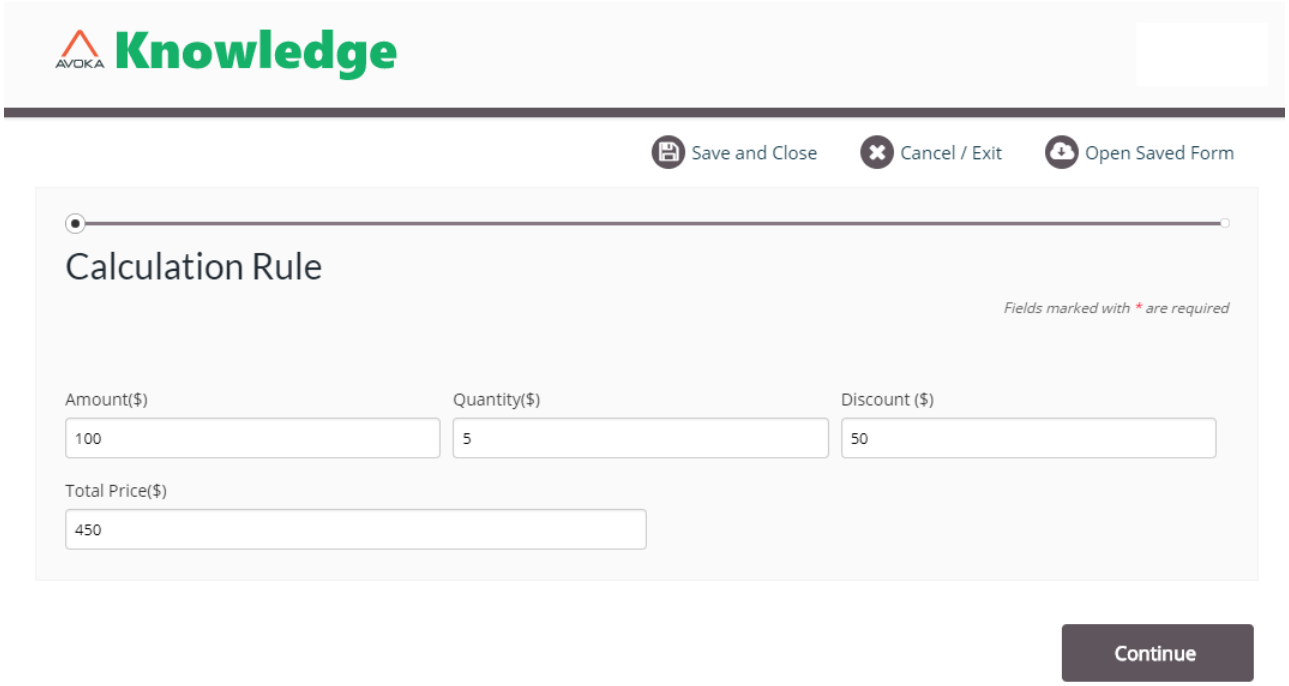
5. Click Script under Calculation as shown below.



6. Expand the form structure as needed and double-click the component you want to include in the calculation
7. Insert a mathematical symbol, for example `+,( ), - , *, /`
8. Double-click the next component you want to include in the calculation.
9. Repeat the steps 7 and 8 until you have completed the calculation, for example, the amount value is multiplied by the value entered into the quantity value and then the discount is subtracted from the value.



10. Click Save.
11. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.


Knowledge
Save and Close
Cancel / Exit
Open Saved Form
Calculation Rule
Fields marked with * are required
Amount($)
100
Quantity($)
5
Discount ($)
50
Total Price($)
450
Continue

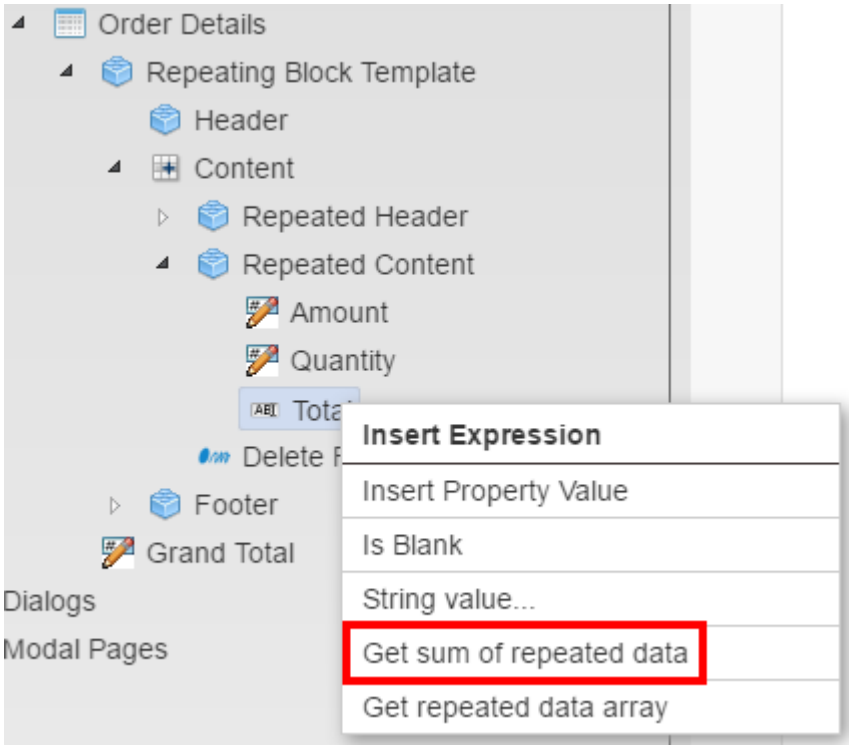
Repeating Blocks Calculation Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

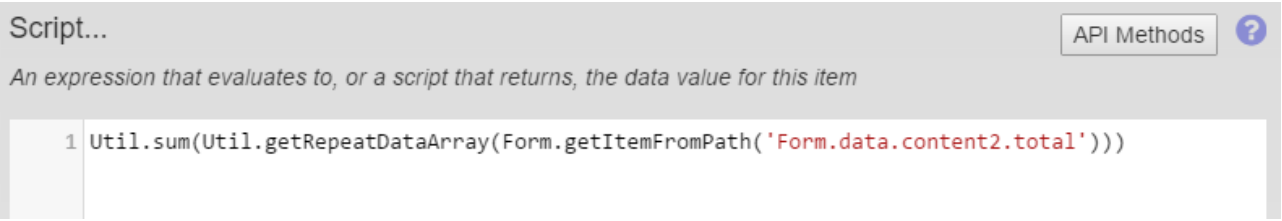
Use a calculation rule for repeating components to perform some summing up actions. For example, you may ask a form user to enter prices of all items they purchased at a supermarket and then display the total sum of the purchase.

To create a repeating calculation rule for a Maestro form, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select the component outside of the repeating block.
4. Click Create Rule.
5. Select Script under Calculation.
6. Expand the repeating block template as Repeating Block Template > Content > Repeated Content.
7. Right-click the component you want to sum.
8. Select Get sum of repeated data as shown below.



The following script is added to the window, which sums the component within each repeating instance.



9. Click Save.
10. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Editability Rule Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Editability Rules limit a form user's ability to edit component content in a Maestro form. There are two editability rules available in the Maestro Create Rule window:

- Read-Only
- Editable if

Create a Read-Only Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Read-Only editability rule makes any component the rule is applied to, read only. When a read-only business rule is applied to a component, form users will not be able to enter any values into the component. The Read-Only rule can be applied to any component on a Maestro form. The Read-Only Rule is useful when you only want to display content in a field, which doesn't need to be modified.

To create a Read-Only rule, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select the component you want to make read-only.
4. Click Create Rule.
5. Select Read-Only. The editability rule status is shown for a selected component in the Rules section.



Editable If Rule Overview

The Editable If rule makes a field editable only when certain conditions are met. By default, a component with an Editable If rule is not editable until the configured condition is met.

For more information, refer to EditableIfRuleCreate.docx

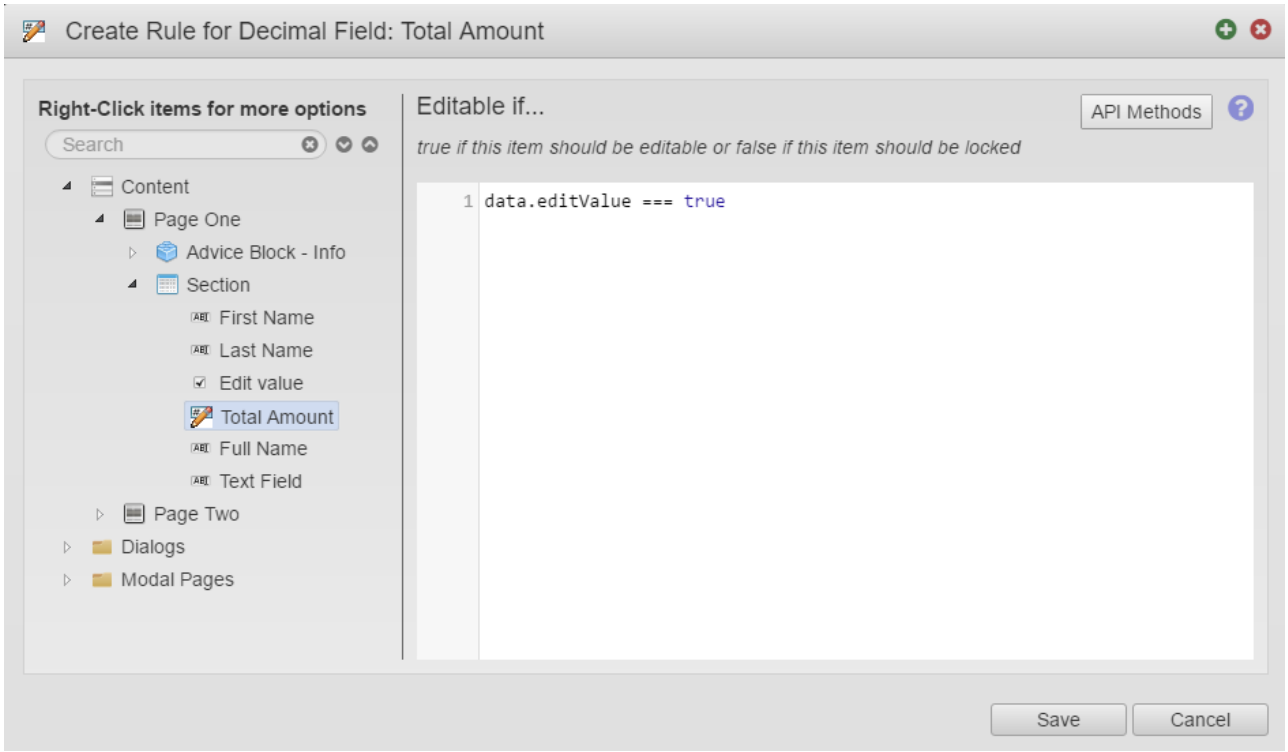
Create an Editable If Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Editable If editability rule makes a field editable only when certain conditions are met. By default, a component with an Editable If rule is not editable until the configured condition is met.

To create an Editable If rule for a Maestro form, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select the component to apply the rule to.
4. Click Create Rule.
5. Select Editable if.



6. Type a JavaScript that implements the rule or right-click on existing components. In this example, an Editable If rule checks if the Edit checkbox is selected then the Total Amount decimal field is editable.
7. Click Save.

NOTE

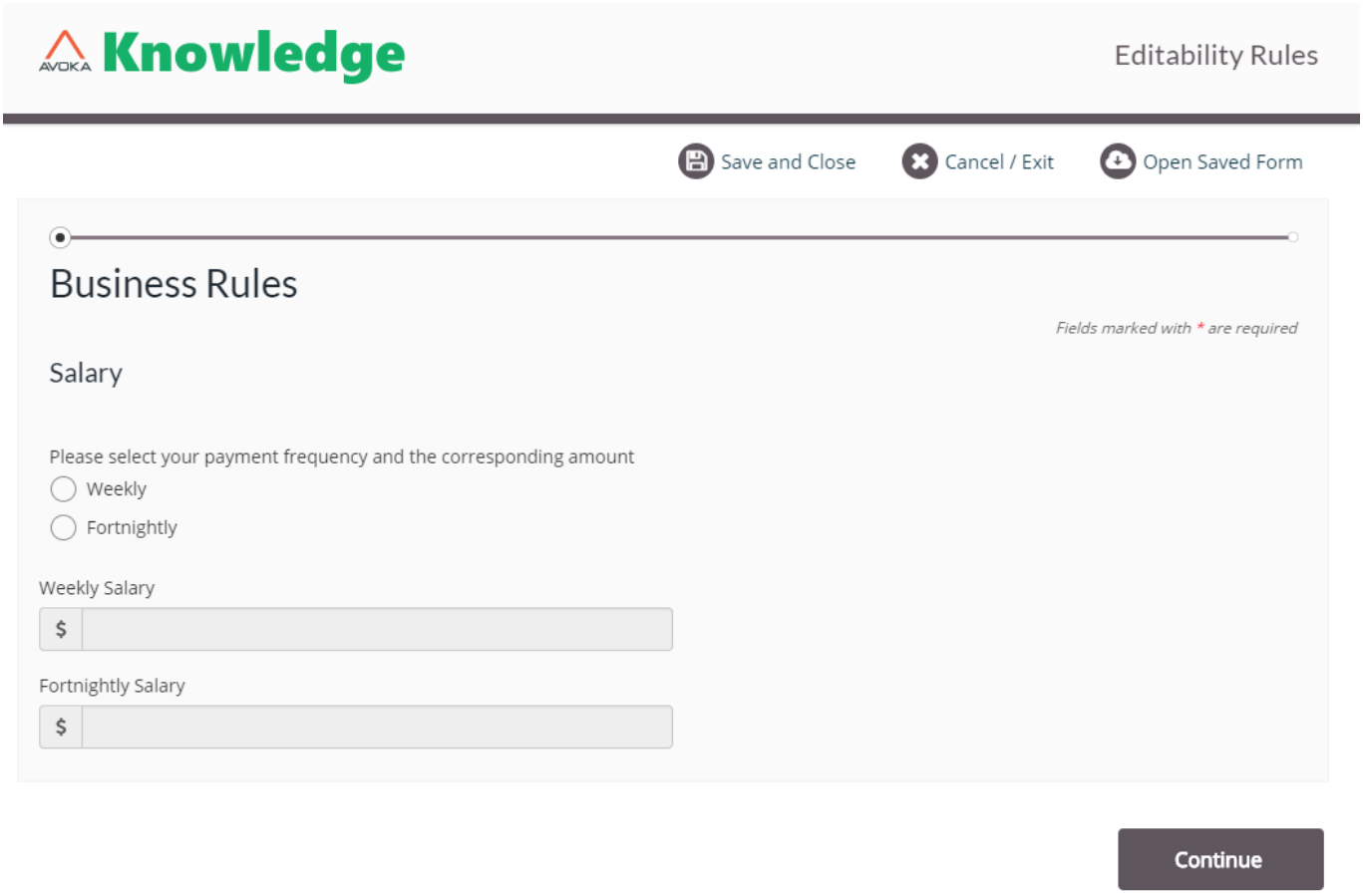
Check out the Editable If rule example.

Salary Editable If Rule Example

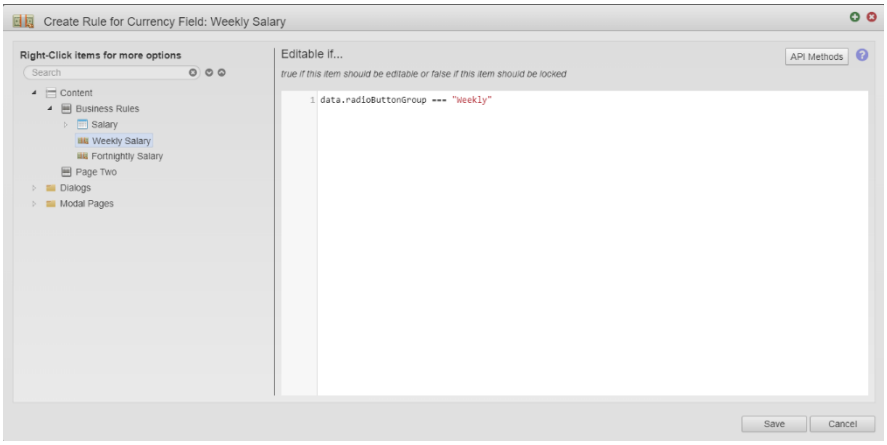
MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

This example uses the Editable If editability rule to allow a user to enter data only into specific components of a Maestro form. If the user selects the Weekly radio button option, then the Fortnightly Salary currency text field becomes not editable and vice versa.

A form below has a radio button group and two currency fields. A user can specify a payment frequency and the amount to be paid.



The Editable If rule is applied to the currency field named Weekly. The JavaScript checks if the Weekly radio button is selected from the radio button group and enables editing of the Weekly currency field. A user is able to enter the amount that to be paid in weekly installments.



The editable if rule that has been applied to the Fortnightly currency field. The script in this example says that if Fortnightly is selected from the radio button group, then the Fortnightly currency field will be editable (i.e the user will be able to enter the amount they will pay in fortnightly payments).

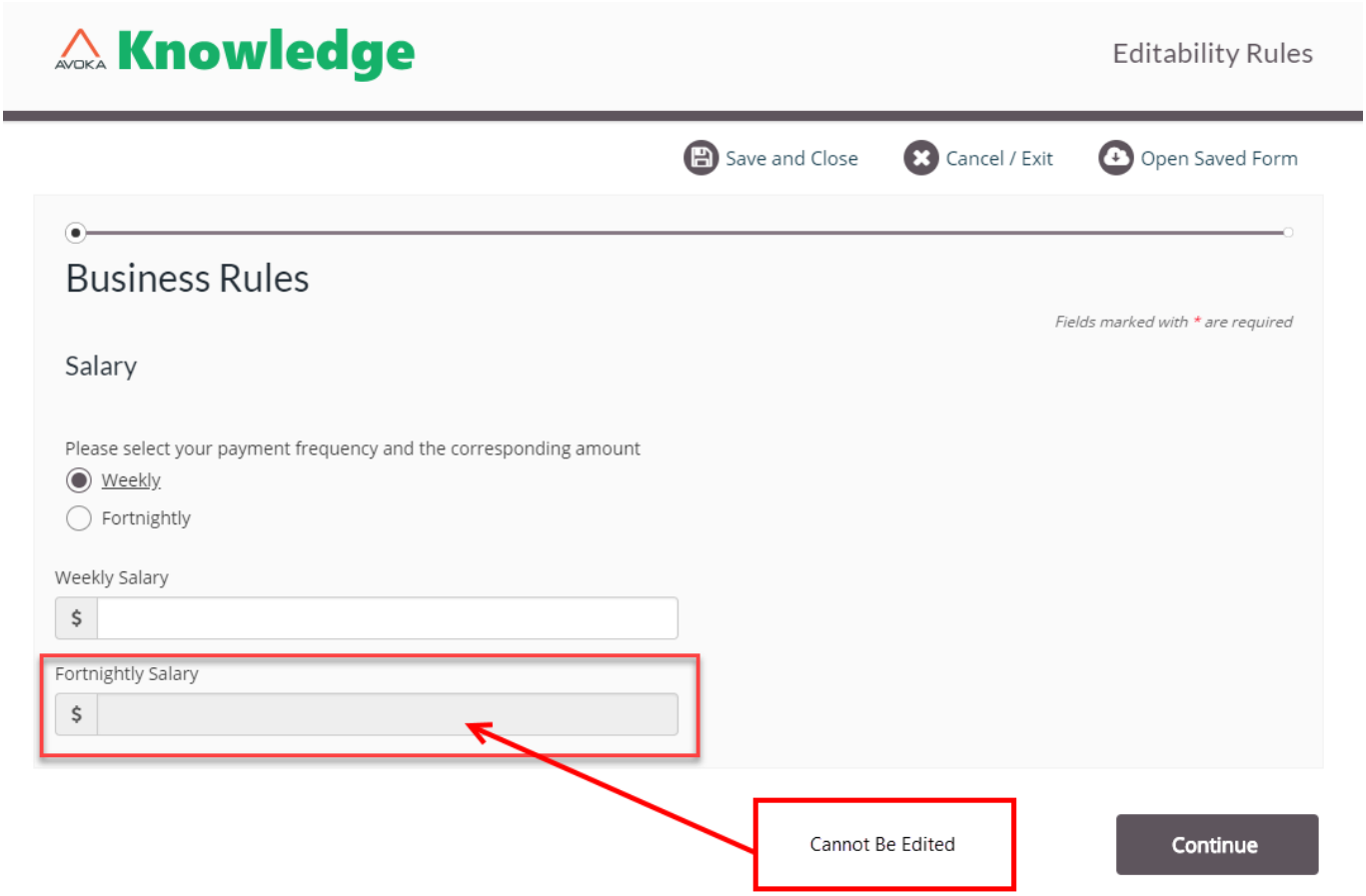




It displays calculation rules that have added to extend the functionality of this example.



It displays the form once the editability rules have been added to the form. When Weekly is selected from the radio button group, the user will not be able to enter input into the Fortnightly Salary currency field.



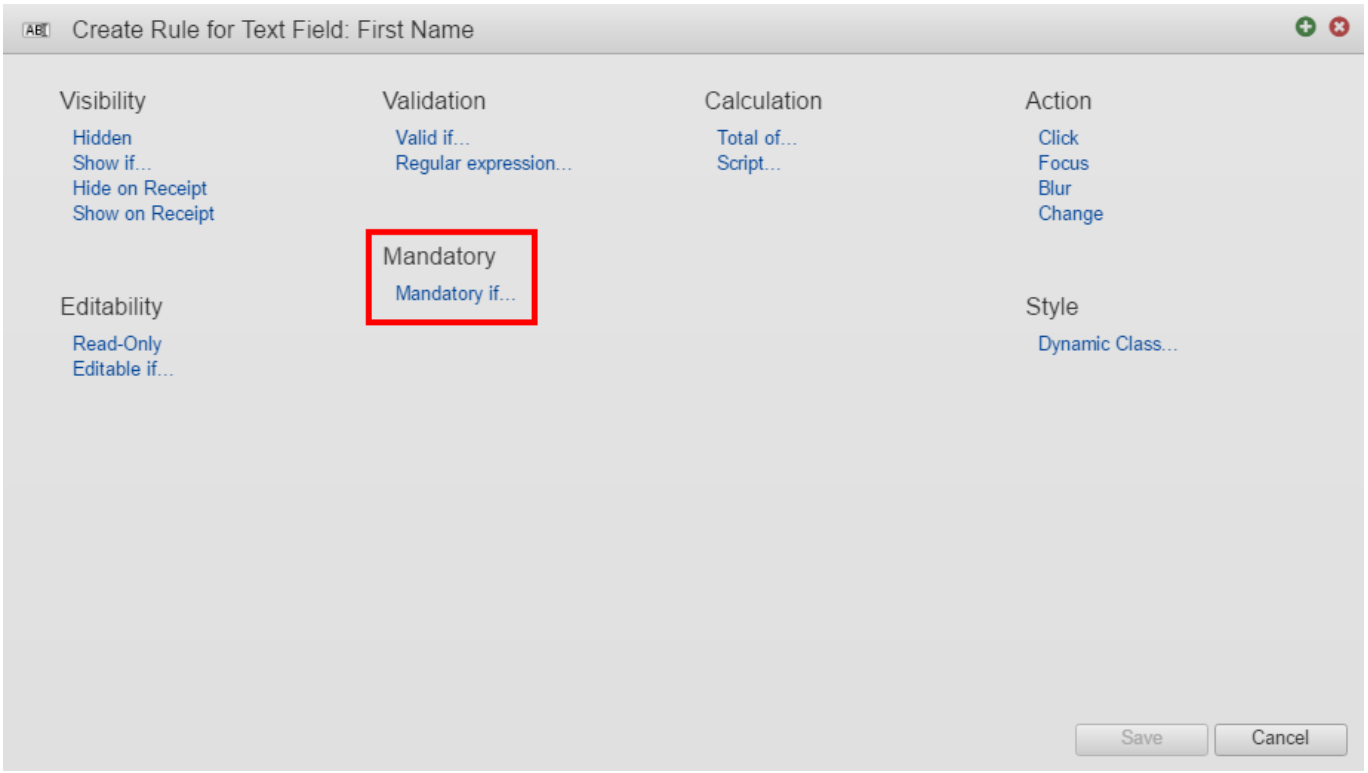
Salary Editable If Rule Example

Mandatory Rule Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A mandatory rule is a rule that requires valid input from a user before the user is allowed to continue with a form journey. Mandatory Rules should be created and used to identify components that are essential to a form journey. This means that no matter what happens, the user should not be able to continue with the form until they have provided valid input for these components. Maestro currently provides Mandatory If rules.

The Mandatory If rule allows you to create a script that marks the component as mandatory based on certain conditions.



Mandatory If Rule Overview

The Mandatory If rule allows you to create a script that marks the component as mandatory based on certain conditions.

To create a mandatory if rule, refer to MandatoryIfRuleCreate.docx

Create a Mandatory If Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Mandatory If rule allows you to create a script that marks the component as mandatory based on certain conditions.

To create a Mandatory If rule, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select a mandatory component.
4. Click Create Rule.
5. Select Mandatory If.
6. Enter some JavaScript to implement the rule.
7. Click Save.
8. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

Use the Mandatory Message from the Properties panel to customize the message displayed when the user does not complete the mandatory component.

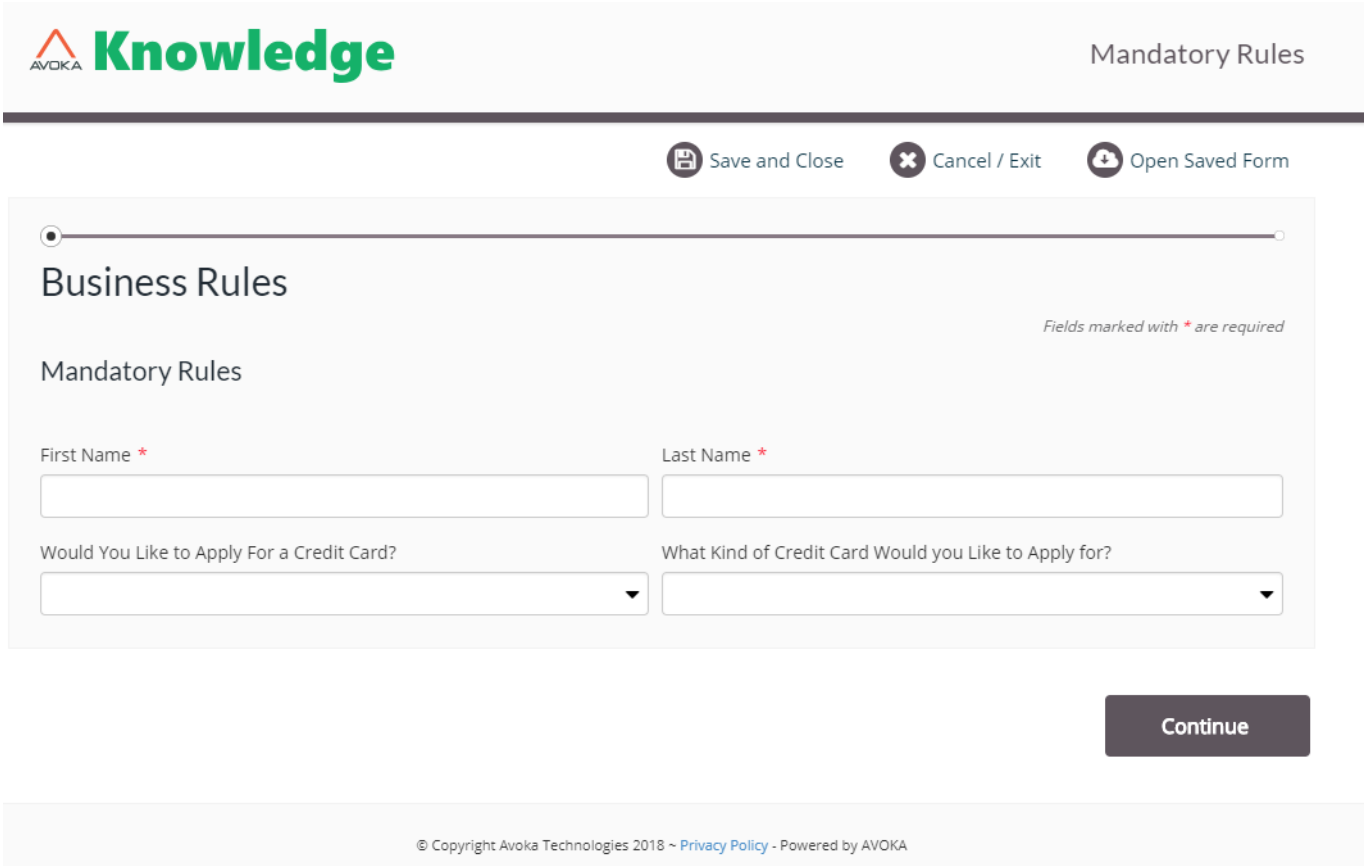
NOTE

Check the Mandatory If rule example.

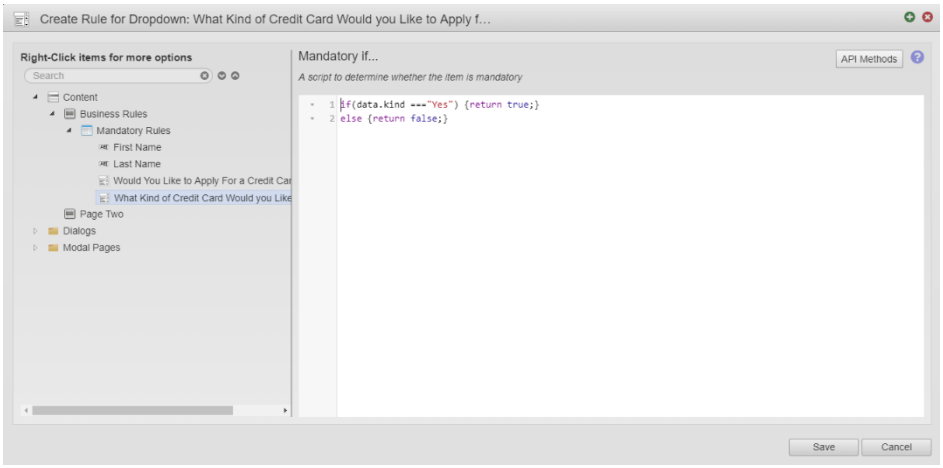
Mandatory If Rule Credit Card Application Example

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

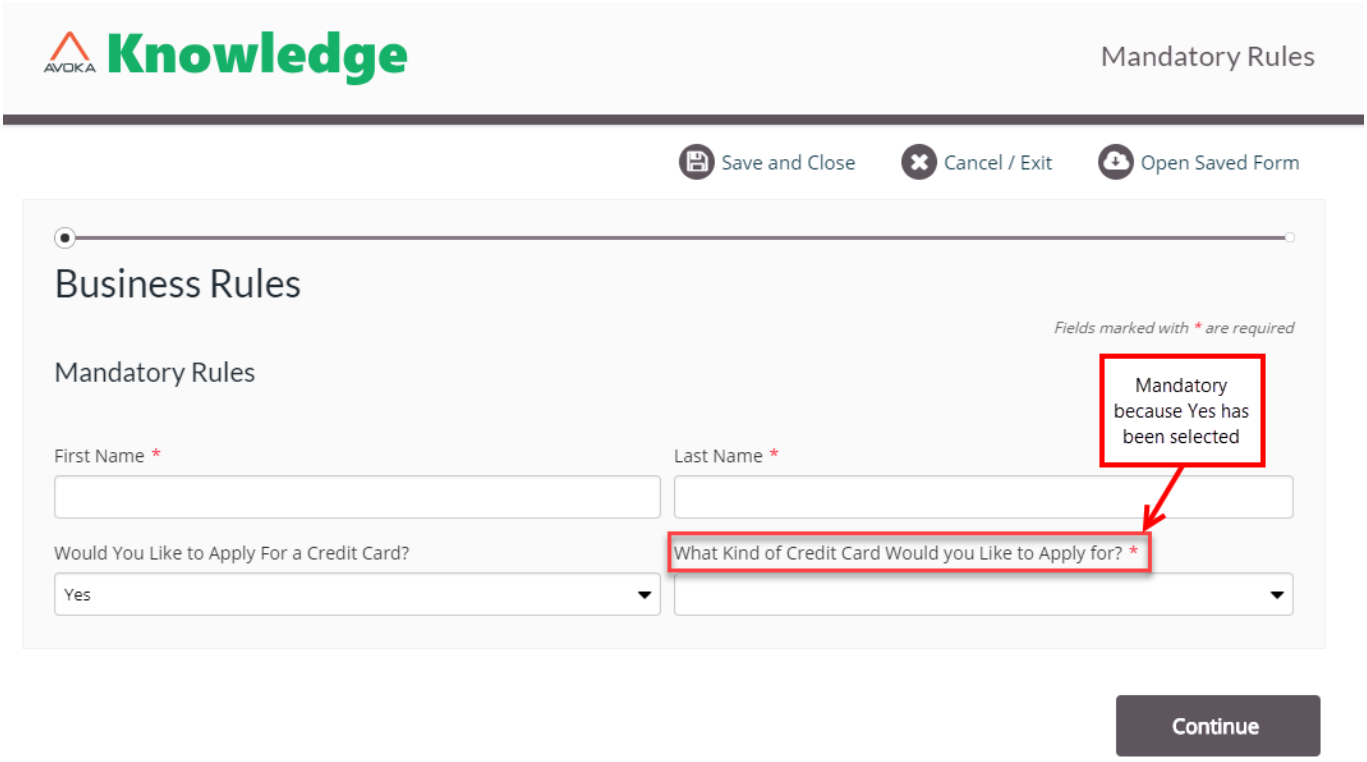
This example uses a credit card application form that asks customers if they would like to apply for a credit card. If the answer is yes, then the type of credit card question becomes mandatory as shown below.



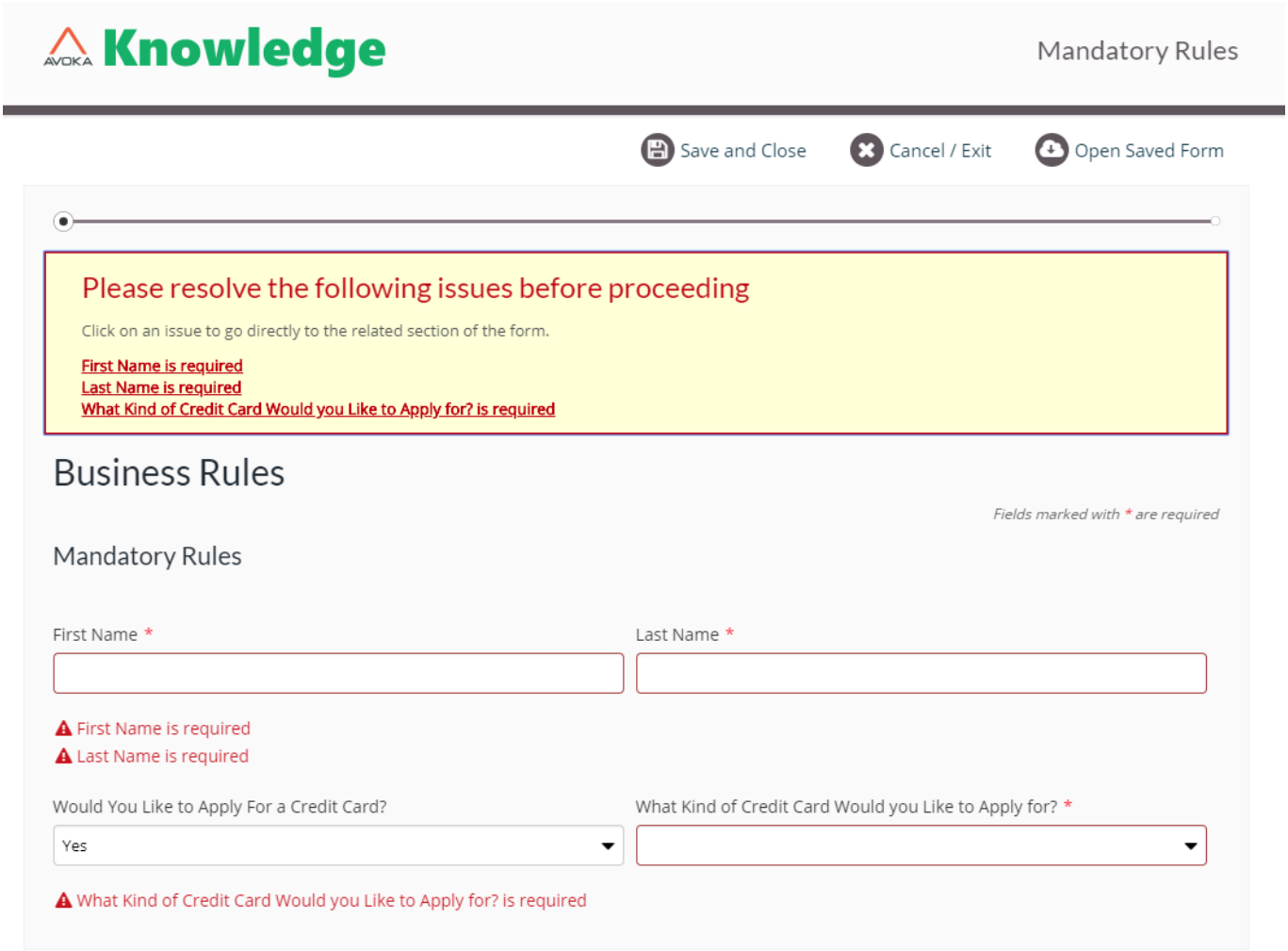
The above screenshot displays how the form appears in the browser initially.



The above screenshot shows the Mandatory If script that is applied to the component. In this example, the script checks if the component named kind is equal to "Yes" (that is, if the customer has selected the "Yes" option for the question "Would you like to Apply for a Credit Card?"). If this condition is met then the script returns true and this field becomes mandatory.



The above screenshot shows that the "What Kind of Credit Card Would you Like to Apply for?" dropdown has become mandatory as a result of the "Yes" option being selected from the first dropdown ("Would you like to Apply For a Credit Card?").



If a mandatory field is left blank and a user attempts to continue their form journey, the associated error message is displayed.

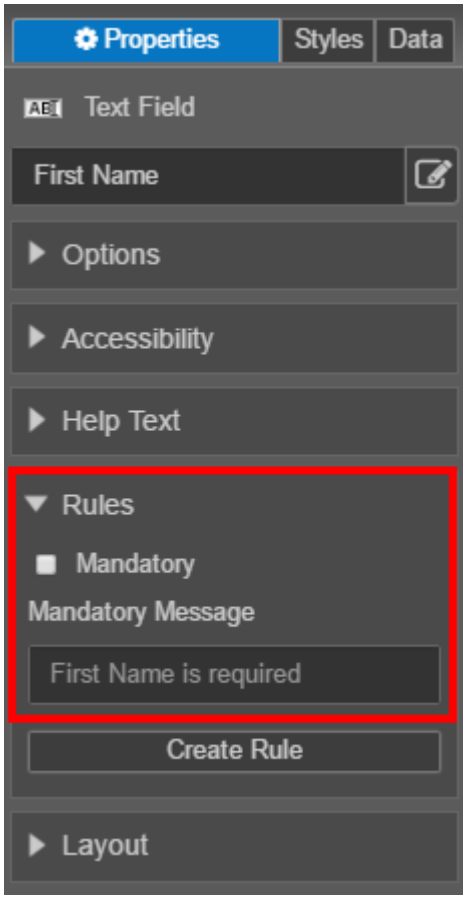
Make a Component Mandatory

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

This is quickest and easiest way to make a component mandatory. Alternatively, you can apply the Mandatory If Rule to component.

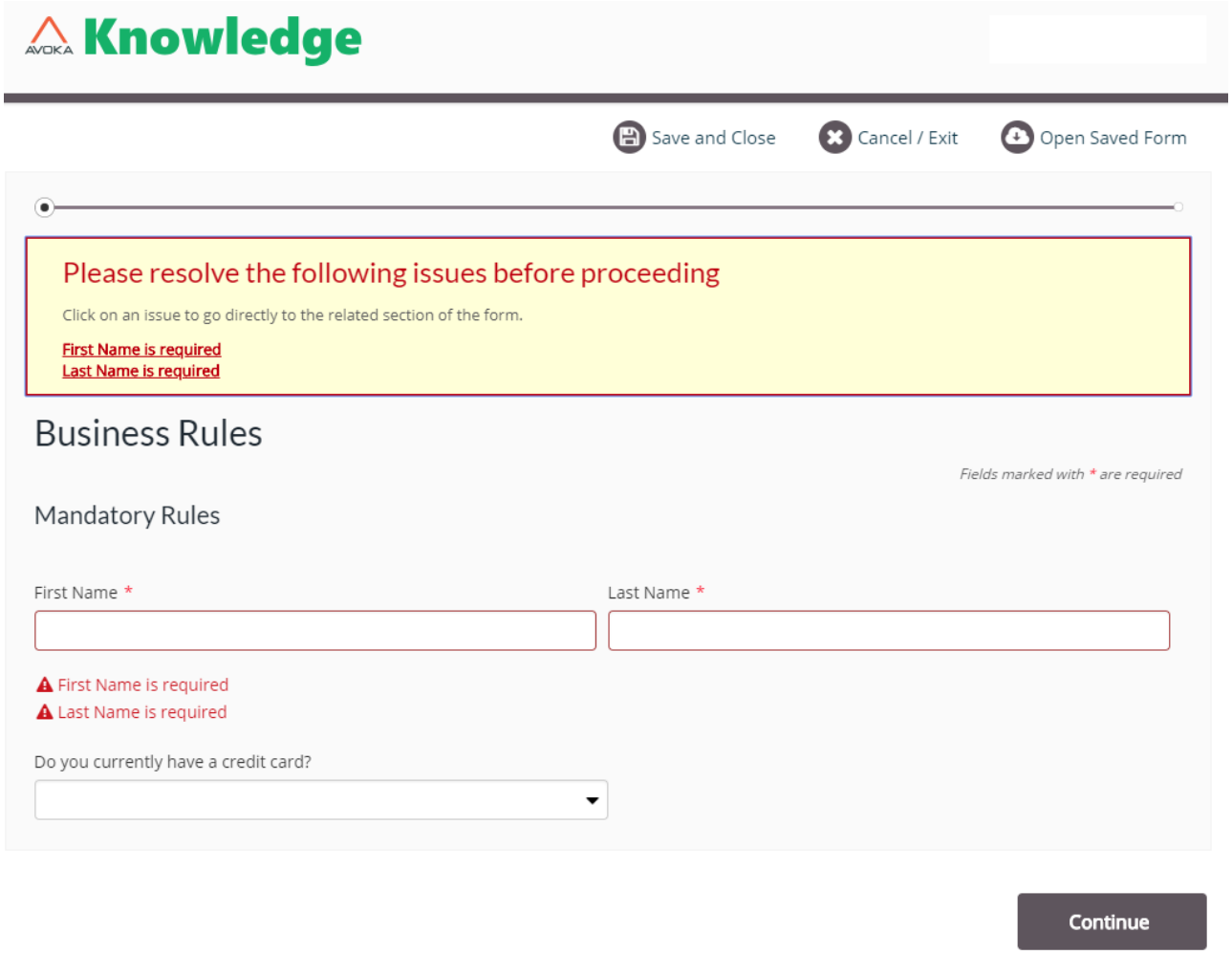
To make a component mandatory, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select a component that you want to make mandatory.
4. Select the Mandatory checkbox.



5. Update the Mandatory Message, which is displayed if a user attempts to move forward in their form journey without completing this mandatory component.
6. Click Build and Render Form to see how this rule is displayed when the form is built and rendered. For example, the Mandatory property has been applied to both the First Name

and Last Name fields.



Validation Rule Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

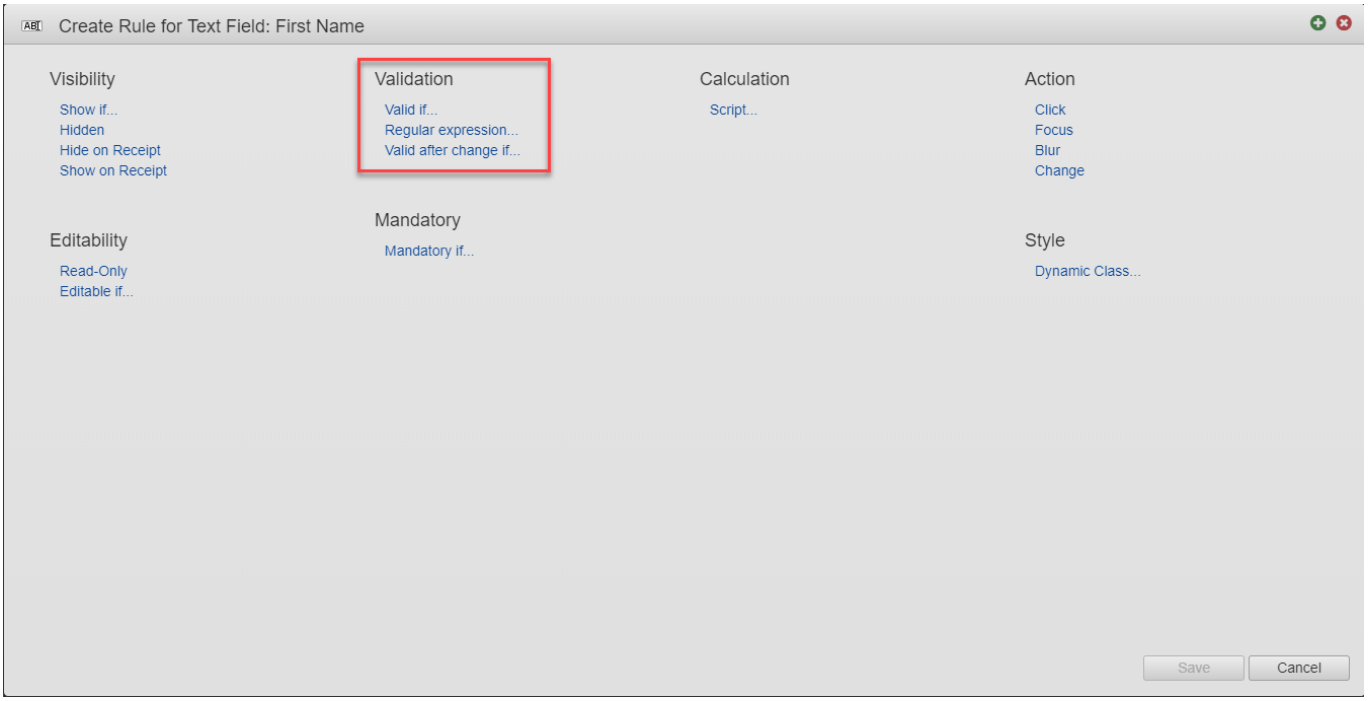
Maestro comes with a set of validation rules that allow you to verify data entered into a form by a user. The most common and simplest rule is a single component validation, which only checks that component data. Alternatively, more complex rules validate the data of several different components.

Validation rules are only invoked once a user has entered data into a field. It's possible to combine a Validation Rule with a Mandatory Rule to ensure that the user enters valid data.

A validation rule states the conditions in which the data in a component is considered valid. For example, a validation rule can generate true or false response to indicate an "OK" or an "Error" state.

There are three standard types of validation rules available in Maestro:

- Valid If
- Regular Expression
- Valid after change if



Next, learn how to use Regular Expressions in your rules.

Create a Regular Expression Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Maestro allows you to use Regular Expressions[1] in validation rules, so you can created a pattern that will match character combinations in strings. A pattern makes use of special text strings, which are like wildcard notations in your OS. For example, as you can use a wildcard of `*.txt` to find all text files in a File Manager, the regex equivalent is `^.*\.txt$`.

You can do much more with regular expressions. You could create the regular expression `\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b` to search for any email address. A very similar regular expression (replace the first `\b` with `^` and the last one with `$`) can be used to check whether the user entered a properly formatted email address. Regular expressions are very compact, so it can be written in just one line of code.

In JavaScript[2], regular expressions are also objects. These patterns are used with the exec and test methods of `RegExp`, and with the match, replace, search, and split methods of String.

Below are several patters that can be handy to use:

- an email address validation - ^[a-zA-Z0-9_-\.]+@[a-zA-Z0-9_-\.]+\.[a-zA-Z]{2,4}$
- an URL validation - ^http\://[a-zA-Z0-9-\.]+\.[a-zA-Z]{2,3}(/\S*)?$

It takes a bit of time to get familiar with RegExp and start using them correctly, so we recommend checking the following resources first :

- Regular Expression Library - http://regexlib.com/
- Regular Expressions 101 - https://regex101.com/
- Online regular expression editor and evaluator - http://regexr.com/
- https://www.debuggex.com

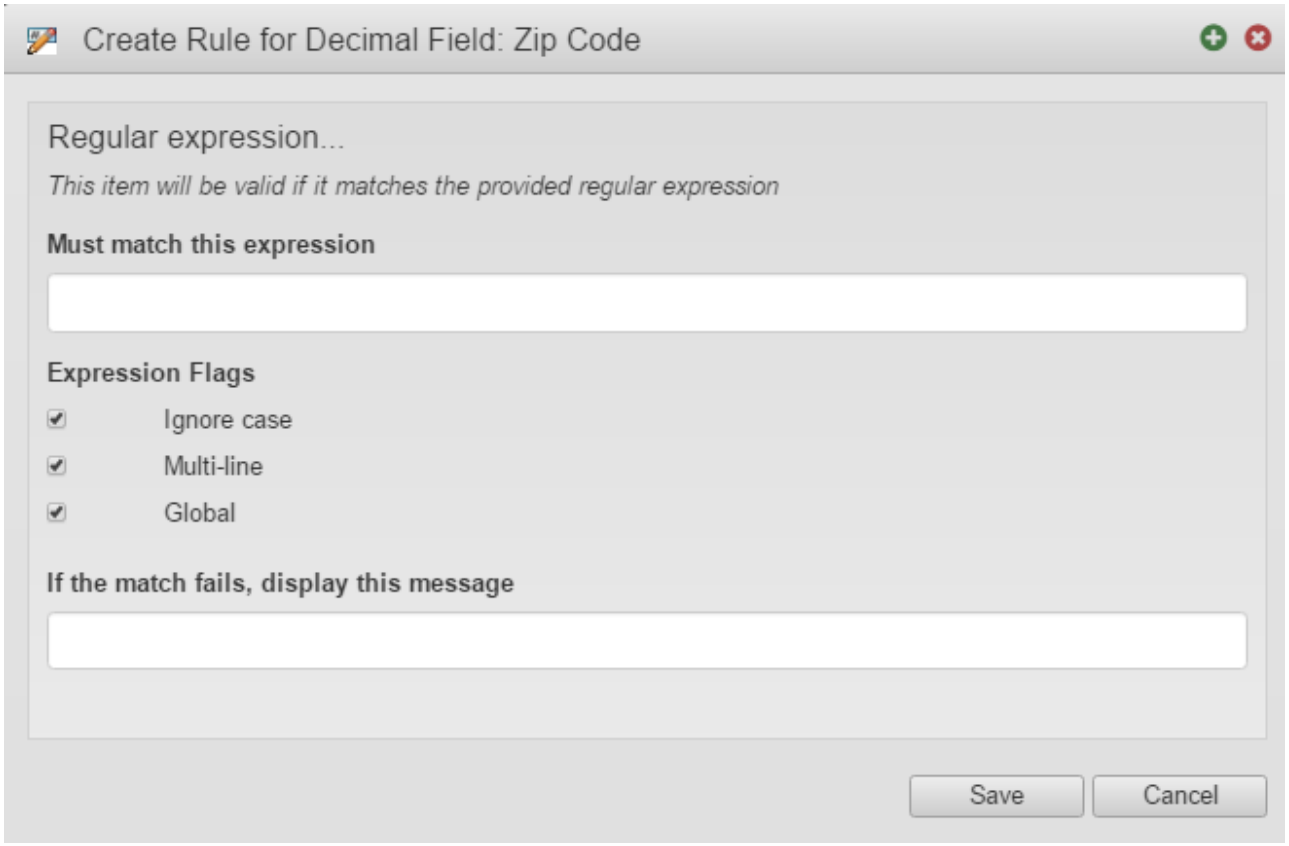
To create a rule with a RegExp:

---

[1]A regular expression (regex) is a sequence of characters that define a search pattern. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation.

[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.

1. Open a Maestro form and select a component from the **View** pane.
2. Select the Properties tab and click Create Rule.
3. Select Regular Expression, located under the **Validation** heading.



4. Enter a RegExp in the Must match this expression field. Define the JavaScript-based logic of the rule within the code editor. As a Validation rule, the Regular Expression rule should return `true` to indicate that the component is valid or `false` to indicate that the component is invalid. for example, a regular expressionvalidation rule.
5. Enter a message in the If the match fails, display this message field.
6. Select the Ignore Case checkbox to use case insensitive matching. For more information, see Mozilla documentation.
7. Select the Multi-line checkbox to apply the RegExp over multiple lines.
8. Select the Global checkbox to find all matches rather than stopping after the first match..
9. Click Save. The rule will appear in the **Properties** pane, under the **Rules** heading.

NOTE

You can only attach one rule of each type to a component.

Next, learn how to create a Valid If Rule.

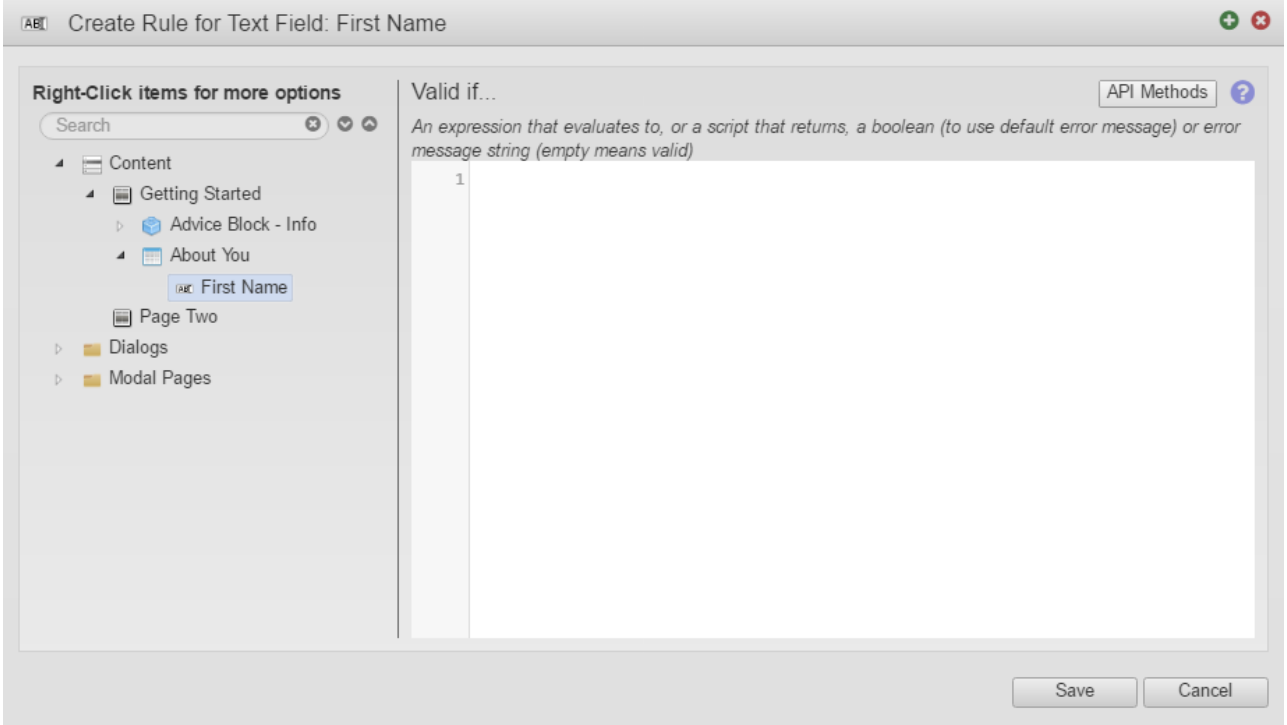
Create a Valid If Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Valid If validation rule can be created to evaluate a component value using JavaScript, which returns true if the value is valid or false otherwise. If the valid if rule returns false, the validation fails, and an error message can be displayed. You can write a JavaScript script or use existing JavaScript functions of any complexity to evaluate the component's value.

To create a Valid If rule:

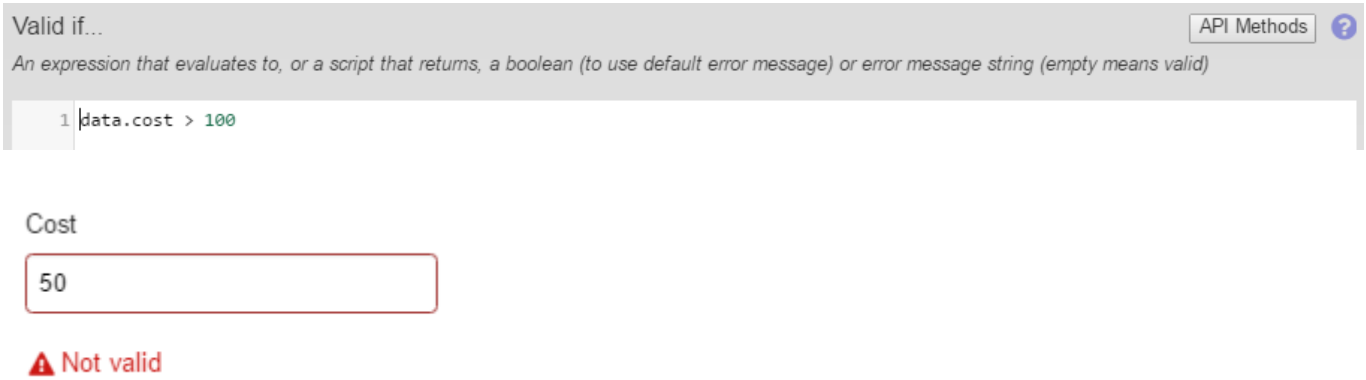
1. Open a Maestro form.
2. Click the Properties tab.
3. Select a component.
4. Click Create Rule.
5. Select Valid If.



6. Type a JavaScript that validates the selected component.
7. Click Save.

To help you understand more how the Valid If rule works, we listed several examples below.

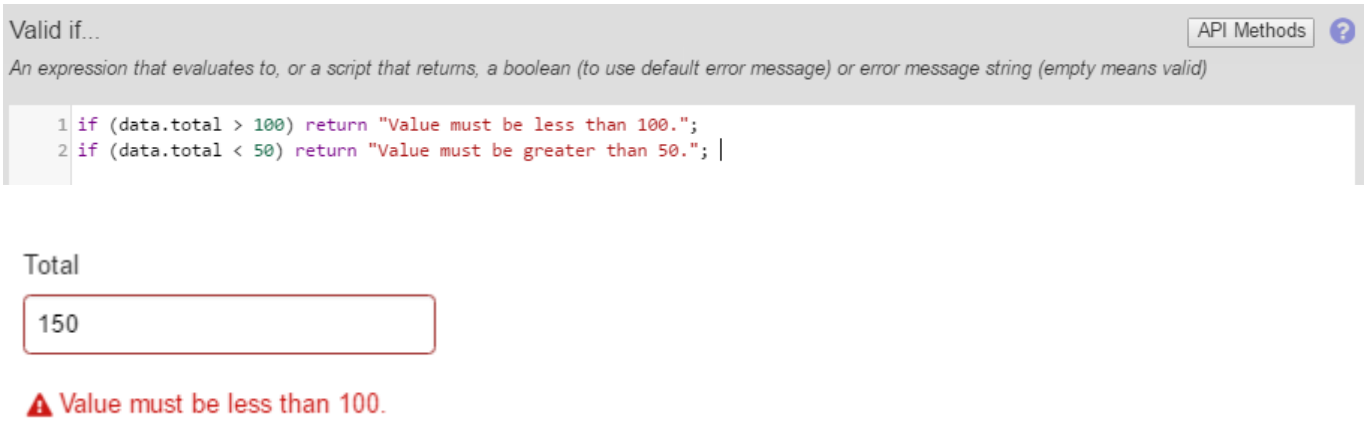
Basic Valid If Rule with Default Error Message



Basic Valid If Rule with Custom Error Message



Valid If Rule with multiple conditions



Next, learn how to Create a Valid After Change If Rule.

Valid After Change If Rule Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The valid after change if rule is a validation rule that is used when you want to perform API based validation on a component, once the value in the component has changed. This rule validates locally, and only validates remotely when the value being validated has changed in some way. The valid after change if rule only runs after the value in the validated component has changed and the standard validation rule has passed.

The valid after change if rule should be used when there is an API call required to validate the value in a component.

For example, when you use a regular email address component, you can do format validation using the browser (e.g. ensuring that the entered email address contains an @ symbol). However, this type of validation will not validate whether the entered email address actually exists. To validate if the email address actually exists, you can use the valid after change if rule to interrogate the mail server and create an API request to confirm that the entered email address actually exists and is valid. Interrogating the mail server using a standard validation rule is not appropriate as a standard validation rule gets called even if the value has not changed, whereas the valid after change if rule will only execute once the value in the component has changed.

You should use both the standard validation rule and the Valid after change if rule together and the validate after change rule will only be called if the standard validation rule passes.

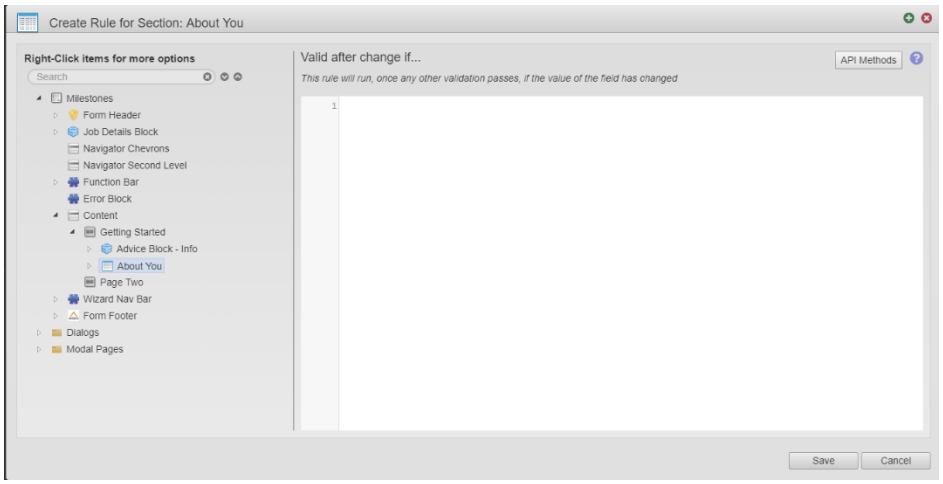
Next, learn how to create a valid after change if rule.

Create a Valid After Change If Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

To create a Valid after change if rule, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select the component.
4. Click Create Rule.
5. Select Valid after change if.

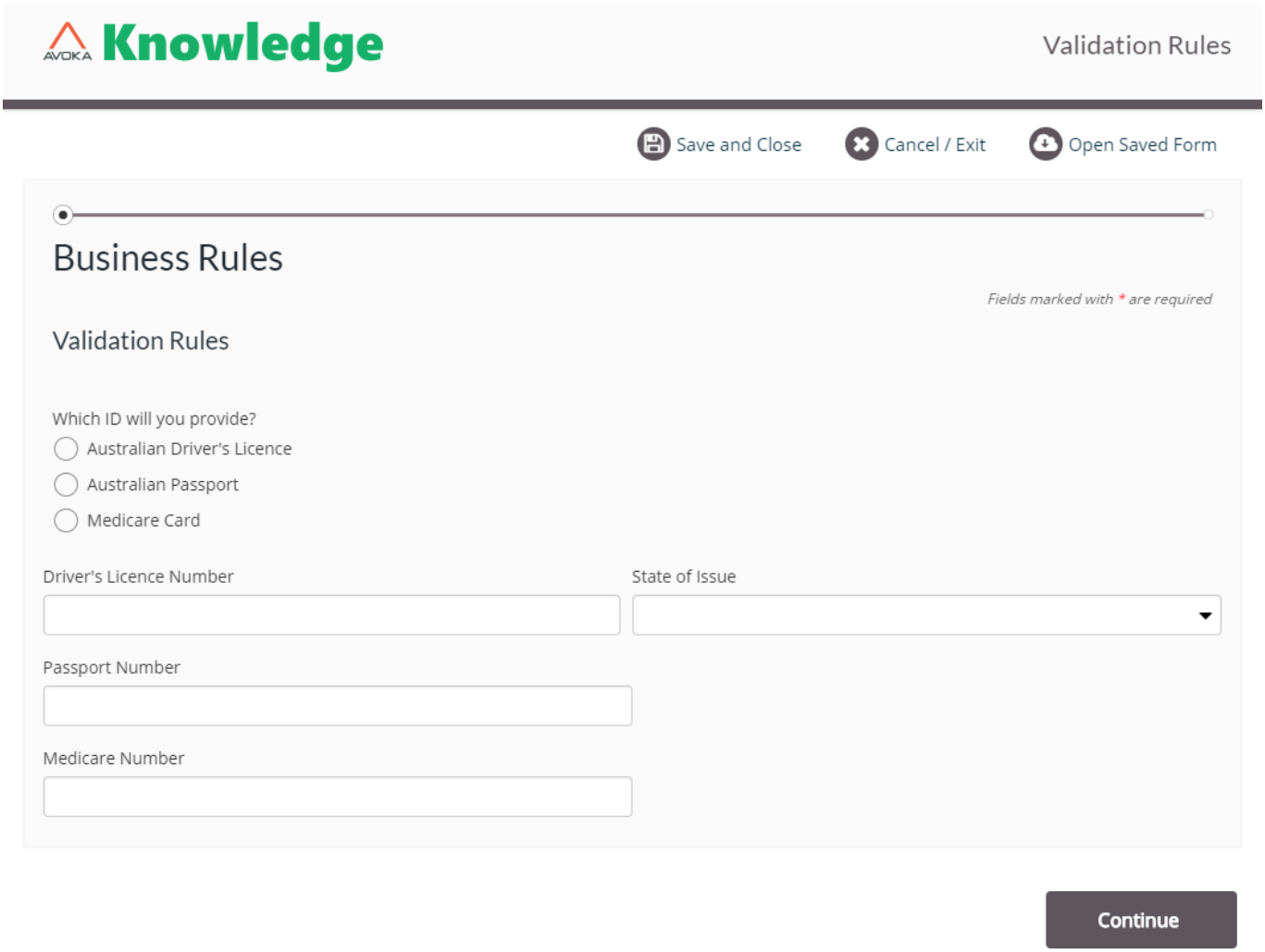


6. Type a JavaScript that implements the validation rule.
7. Click Save.

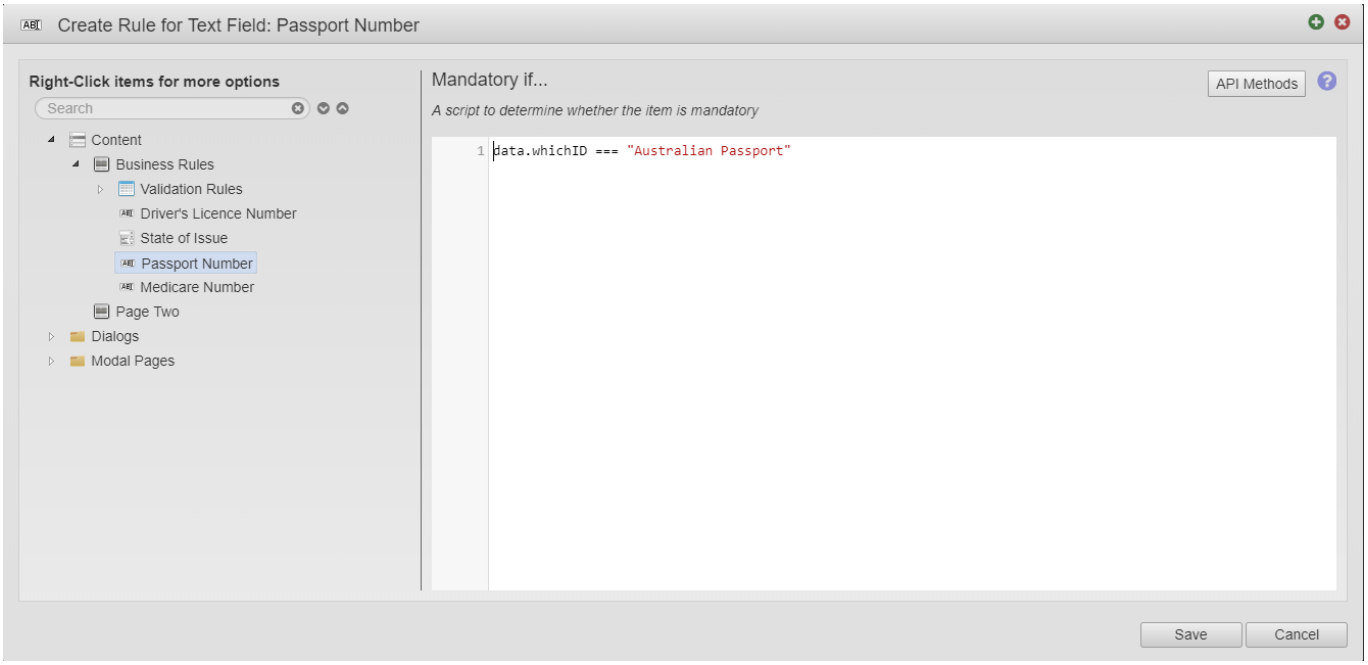
Australian ID Validation Rule Example

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

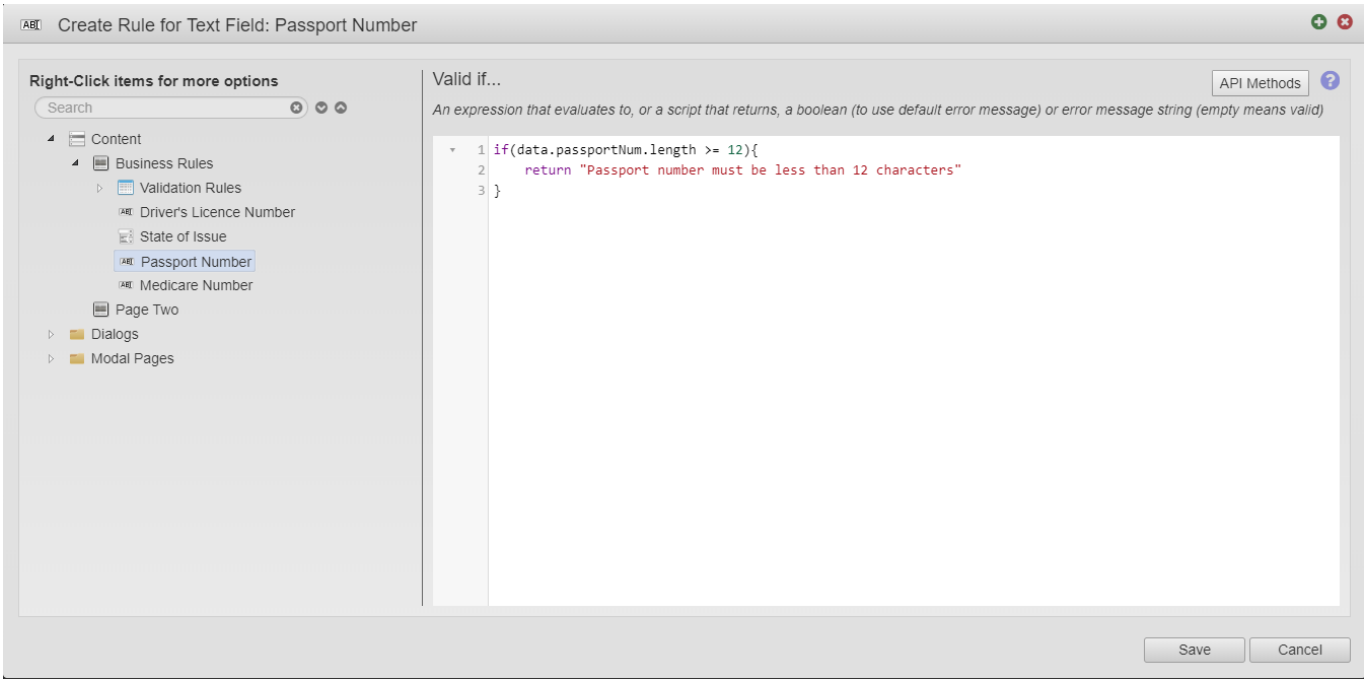
This example uses three validation rules to check the input from a user who is attempting to identify themselves. The form below provides several ID options, which the user can chose from and then provide the appropriate ID information in to components.



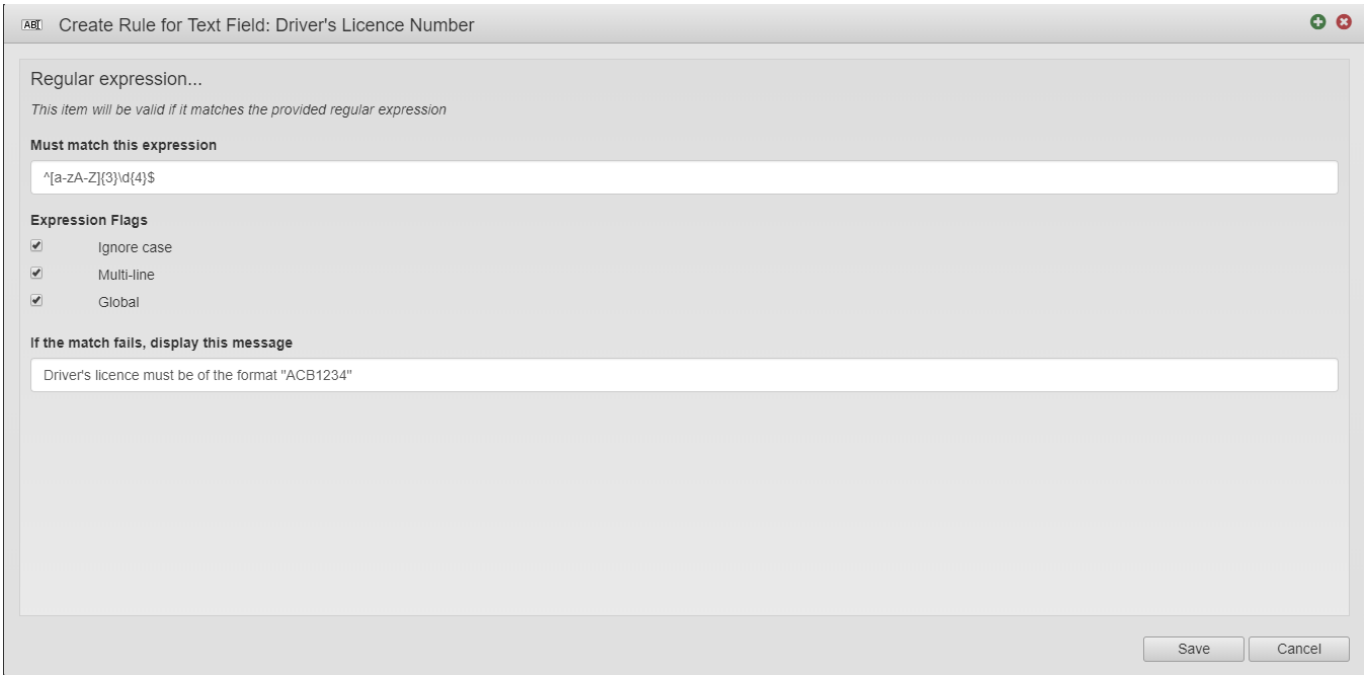
This Mandatory if rule applies a script to the Passport Number text field.



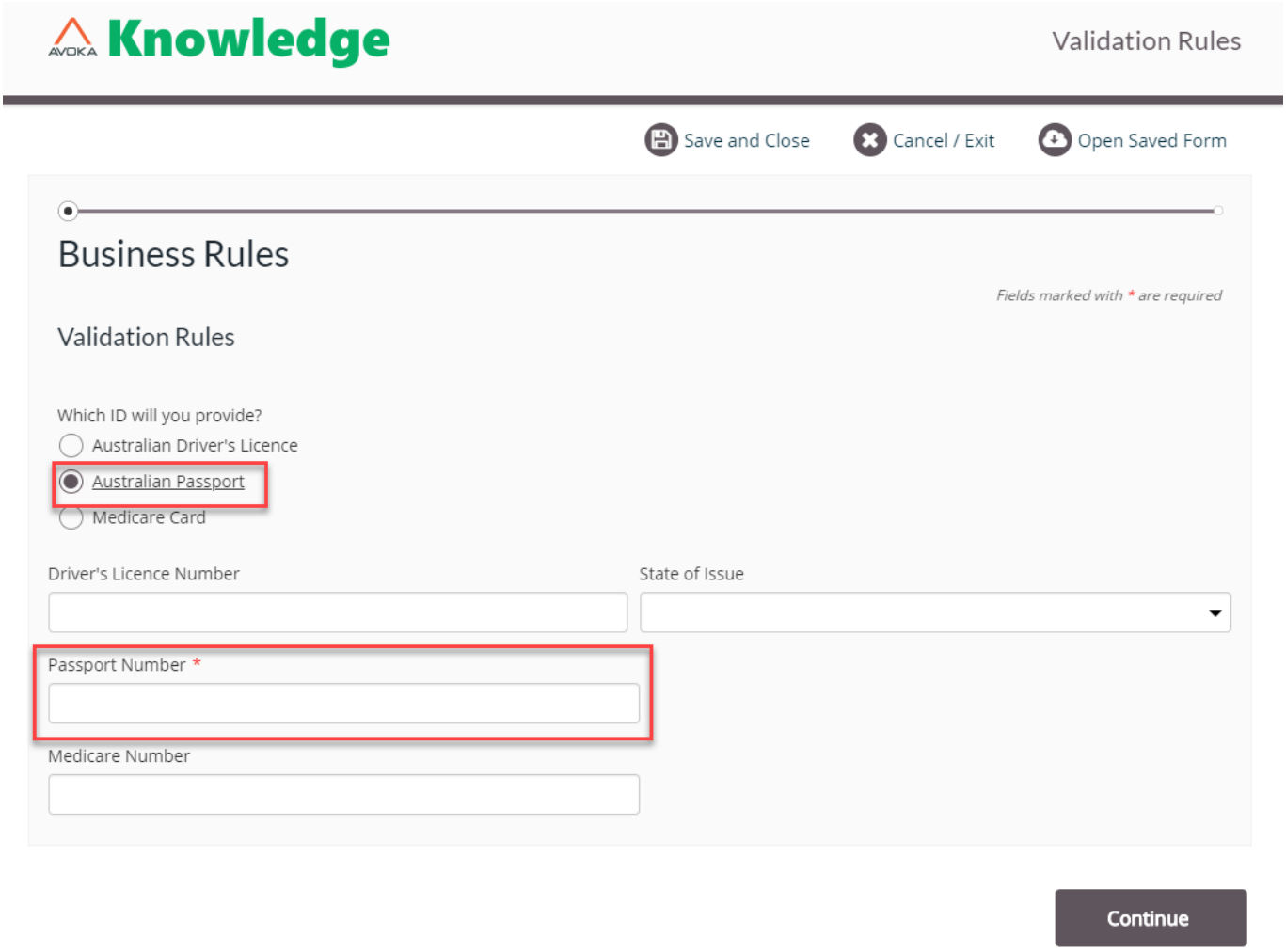
This Valid If rule applies a script to the Passport Number text field, which checks if its input data exceeds or is equal to 12 characters long and deems it invalid and show the error "Passport number must be less than 12 characters" message to the user.



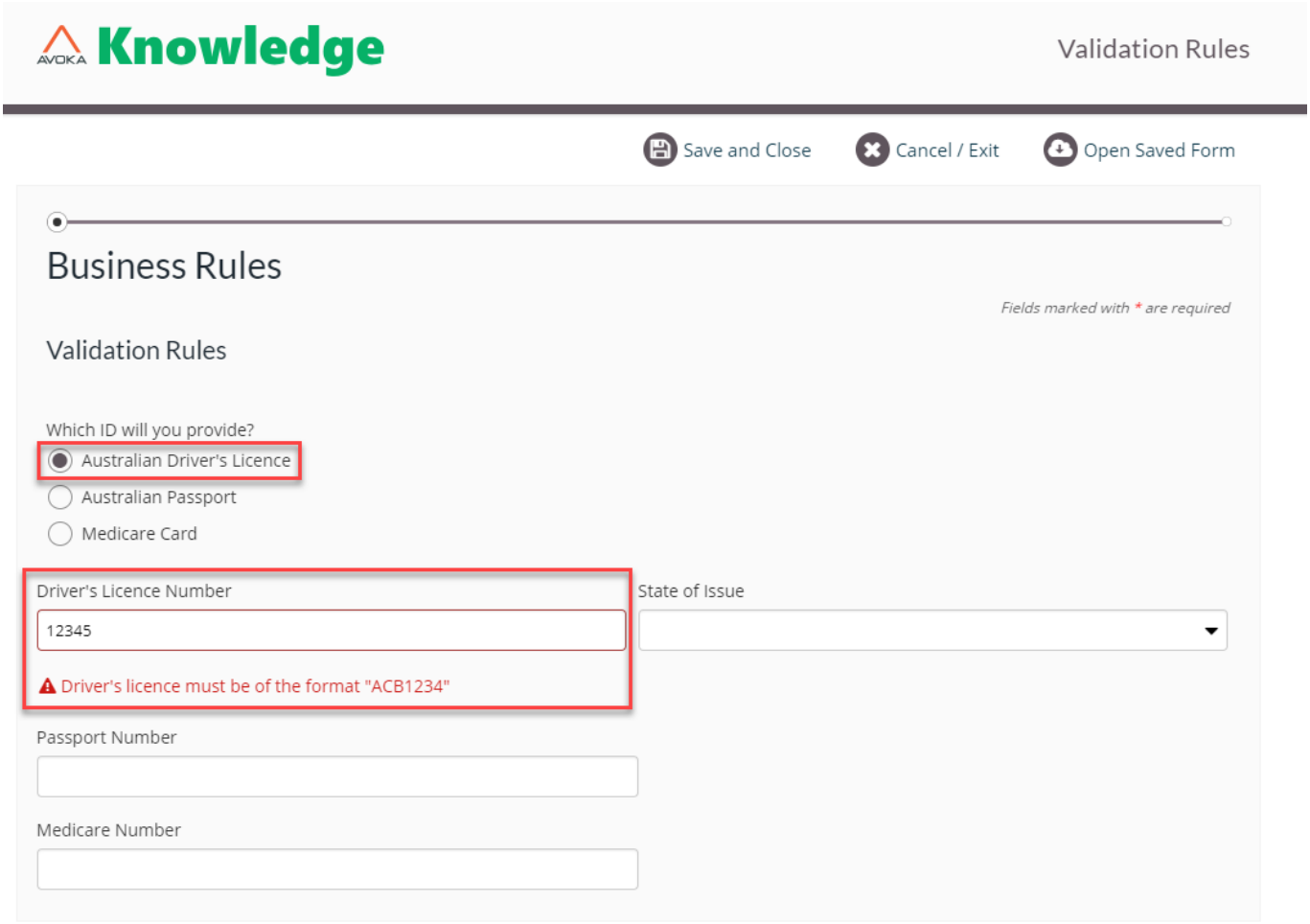
Next rule is a Regular expression that has been applied to the Driver's License text field. In this script, all input of the text field must match the expression ^[a-zA-Z]{3}\d{4}$ and if the input does NOT match this expression, then the message, "Driver's license must be of the format "ACB1234" is displayed to the user.



This displays the form that has been built and then the user has selected. "Australian Passport" from the radio button group. This selection has triggered the mandatory rule and the Passport Number text field has now been made mandatory.



This displays the error message when the input of the Driver's License is not successfully validated.



Australian ID Validation Rule Example

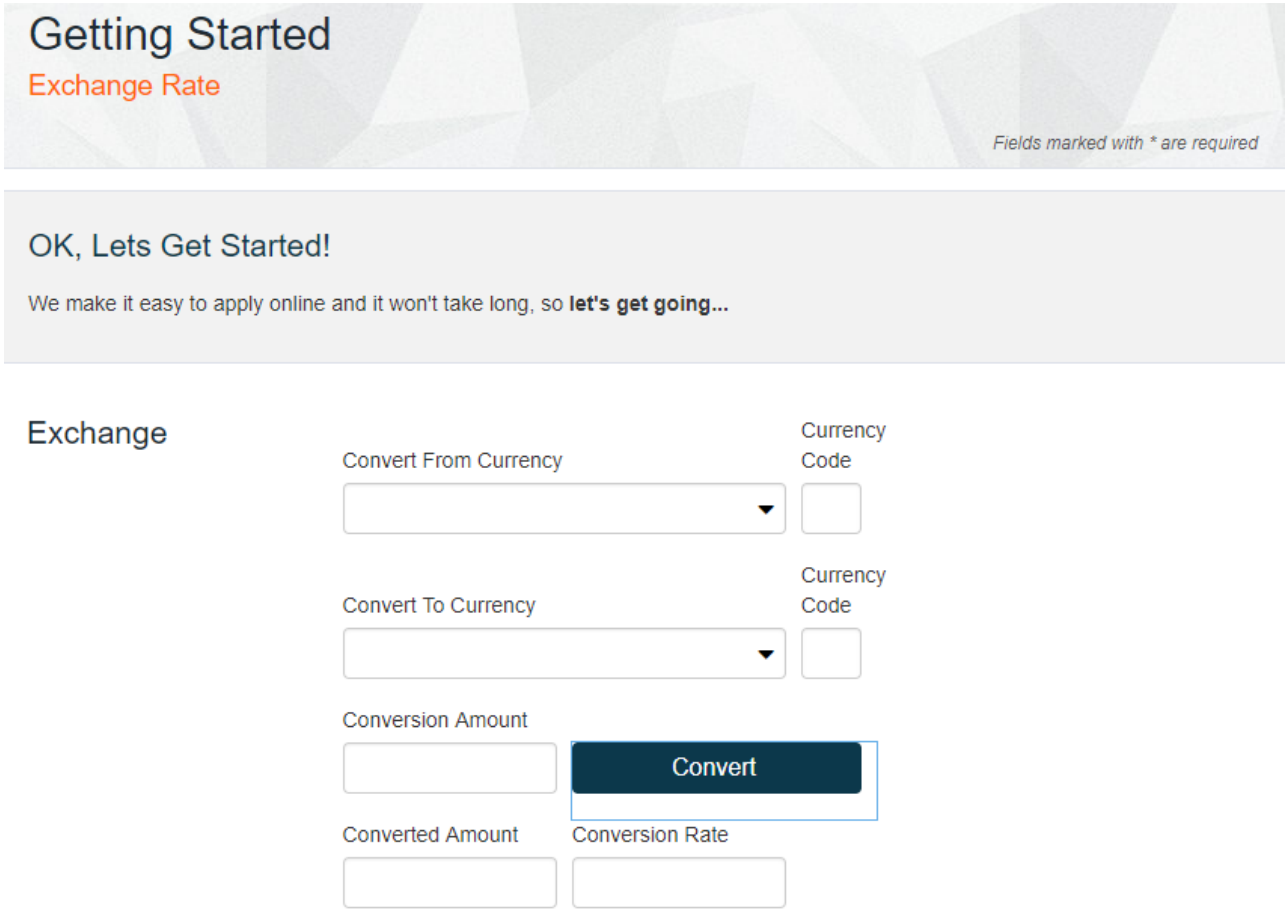
Exchange Rate Valid After Change If Rule Example

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The example demonstrates the Valid after change if rule, which retrieves an exchange rate value from a Journey Manager.

To create the exchange rate rule, follow these steps:

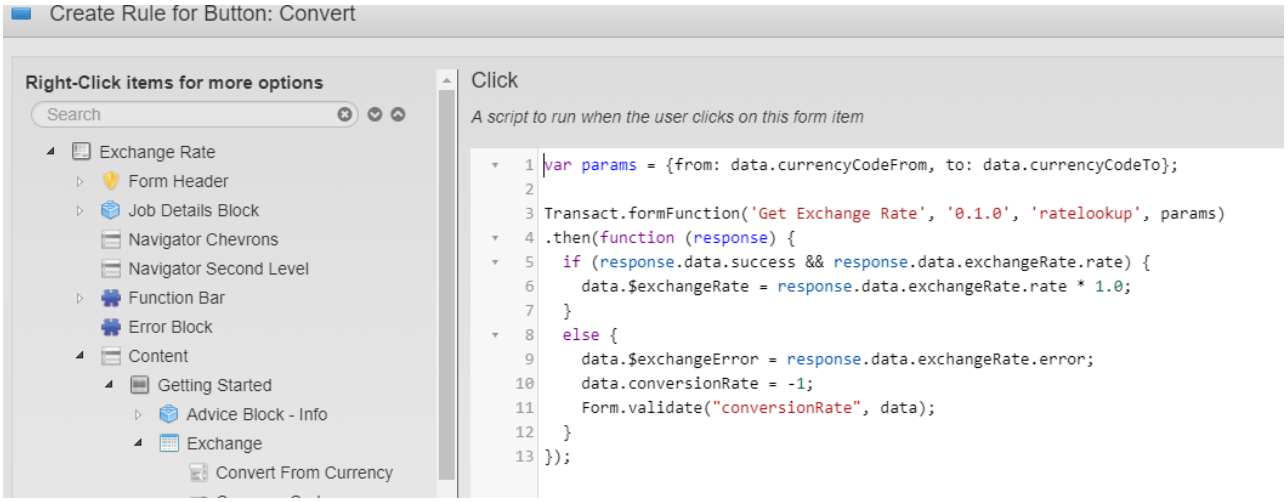
1. Select an Exchange Rate form.



2. Select the Convert button to add a rule to.
3. Click Create Rule and select Valid after change if.



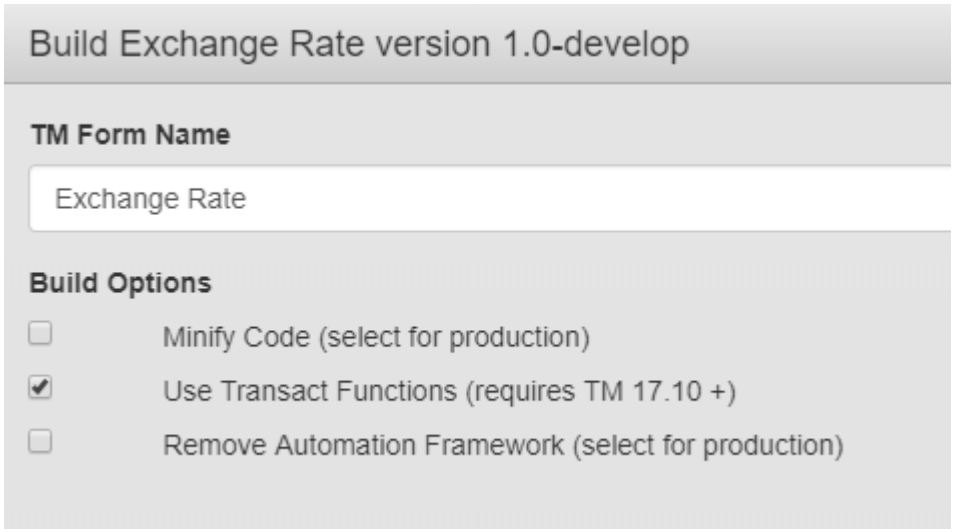
4. Name the rule according to the functionality, in this example it is Click.
   - Enter the following JavaScript code.




```
var params = {from: data.currencyCodeFrom, to: data.currencyCodeTo};
Transact.formFunction('Get Exchange Rate', '0.1.0', 'ratelookup', params).then(func-
tion (response) {
	if (response.data.success && response.data.exchangeRate.rate) {
		data.$exchangeRate = response.data.exchangeRate.rate * 1.0;
		data.$exchangeError = response.data.exchangeRate.error;
		data.conversionRate = -1;
		Form.validate("conversionRate", data);
	}
});
```

5. Click Save.
6. Click Build with Options to see how this rule is displayed when the form is built and rendered. When you initially build the Journey Manager form version in Maestro, it is important that you check the Build Option Use Transact Functions as for this rule to function cor-

rectly, you will be developing with Transact Functions. See Journey SDK - Developer Guide for more details.



7. Check in Journey Manager that the following configuration exists.
    - The Exchange Rate Form is created.
    - It is configured so it can be called from Maestro by service name

- Groovy code is provided.



Visibility Rule Overview

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

A Visibility rule controls visibility scope of components and assets used in a form. Maestro has the following rules defined under the Maestro Visibility Rule group.

- Hidden - hides a component on a form.
- Show if – shows a component on a form if pre-defined conditions are met, otherwise the component is hidden.
- Hide on Receipt - hides a component on a generated receipt but shows it on the form.
- Show on Receipt – shows a component on a generated receipt but hides it on the form.

Create a Hidden Visibility Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The Hidden visibility rule hides the selected component on the form.

A Hidden rule is created automatically when a Show if rule is applied to a component.

To create a hidden rule, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select the component you want to hide.
4. Click Create Rule.
5. Select Hidden.
6. Click Build and Render Form to see how this rule is displayed when the form is built and rendered. Check that the selected component is hidden now.

When a Hidden rule is configured, a Visibility rule item is displayed in the Rules section of the Properties pane for the selected Maestro form. If you want to make any changes to the existing rule, you can do so by selecting this item from the panel. You can also click the X icon to remove the rule from the component.



Create a Hide on Receipt Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

When a transaction is completed by a form user, a receipt is rendered and emailed to the user. This receipt is used as a record for form users. By default, most components are display on both the form, and on the generated receipt. Use the Hide or Show on Receipt rules to hide components on the form and only show them on a receipt, or hide the components on the receipt and only show them on the form.

The Hide on Receipt visibility rule hides the component on a generated receipt.

To create a Hide on Receipt rule, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select the component to hide on the receipt
4. Click Create Rule.
5. Select Hide on Receipt.
6. Type a JavaScript that implements the rule.
7. Click Save.
8. Click Build and Render Form to see how this rule is displayed when the form is built and rendered. Check that the selected component is hidden on a generated receipt.

When a Hide on Receipt rule is configured, a Visibility rule item is displayed in the Rules section of the Properties panel for the selected Maestro form. If you want to make any changes to the existing rule, you can do so by selecting this item from the panel. You can also click the X icon to remove the rule from the component.



Hide or Show on Receipt Overview

Use the Hide on Receipt or Show on Receipt to control which components form users see on a generated receipt.

Create a Show on Receipt Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

The show on receipt visibility rule shows the component on a generated receipt. The component is still displayed in the Editor window, however, when you preview or build the Journey Manager form version, the component is not displayed.

When a transaction is completed by a form user, a receipt is rendered and emailed to the user. This receipt is used as a record for form users. By default, most components are display on both the form, and on the generated receipt. Use the Hide or Show on Receipt rules to hide components on the form and only show them on a receipt, or hide the components on the receipt and only show them on the form.

The show on receipt rule should be used when you have components that require no user input but provide further information to the user. For example, you may want to display a list of items that the user is required be collect in addition to submitting the form and printing a receipt.

To create a Show on Receipt rule, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select the component to hide on the receipt
4. Click Create Rule.
5. Select Show on Receipt.
6. Enter JavaScript that implements the rule.
7. Click Save.
8. Click Build and Render Form to see how this rule is displayed when the form is built and rendered. Check that the selected component is shown on a generated receipt.

When a Show on Receipt rule is configured, a Visibility rule item will be displayed in the Rules section of the Properties pane for the selected Maestro form. If you want to make any changes to the existing rule, you can do so by selecting this item from the panel. You can also click the X icon to remove the rule from the component.

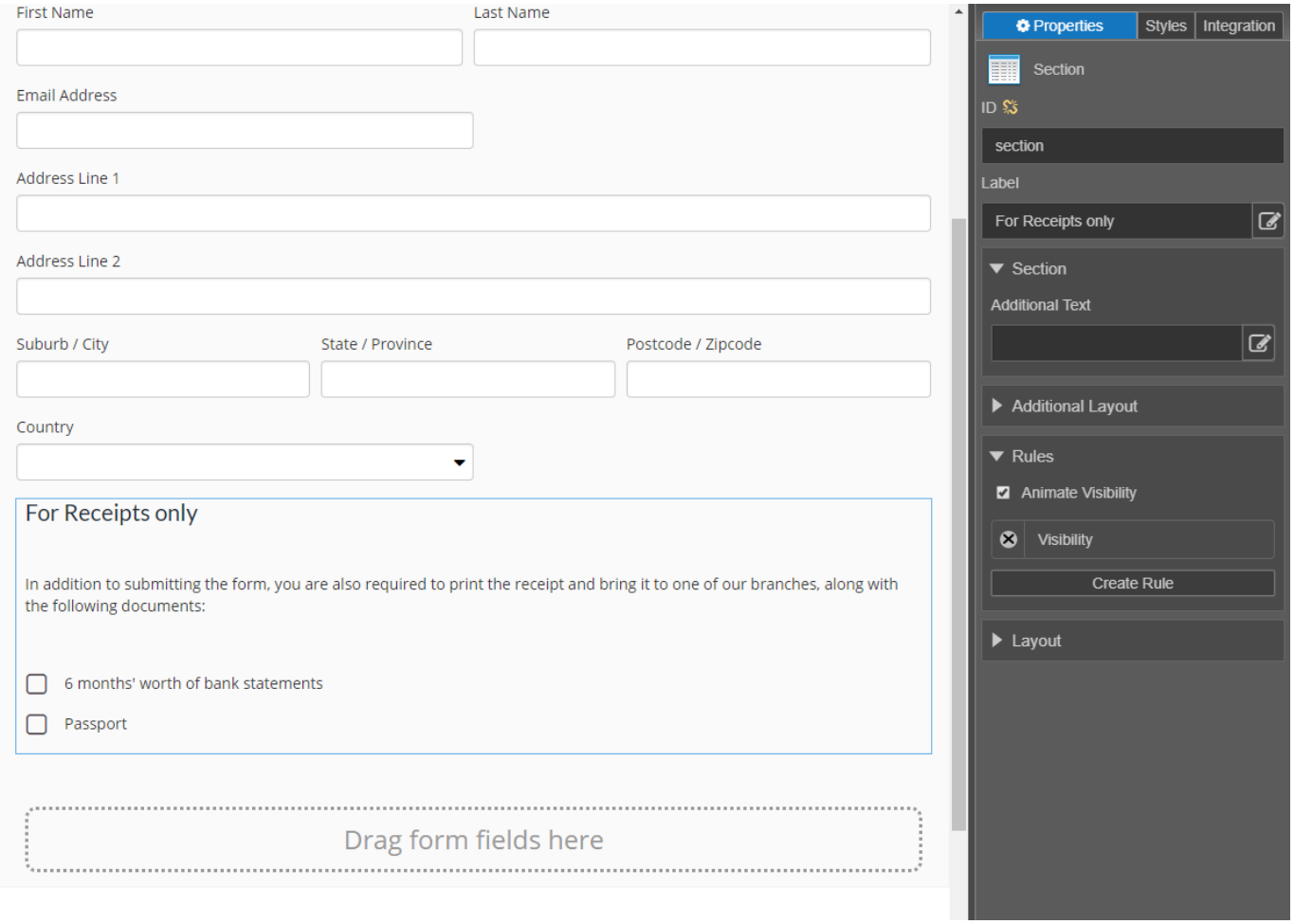
NOTE

Check out the Show on Receipt example.

Show on Receipt Example

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

This example uses the Section component For Receipts only, which is configured to show only on a generated receipt and is hidden on the rendered form.

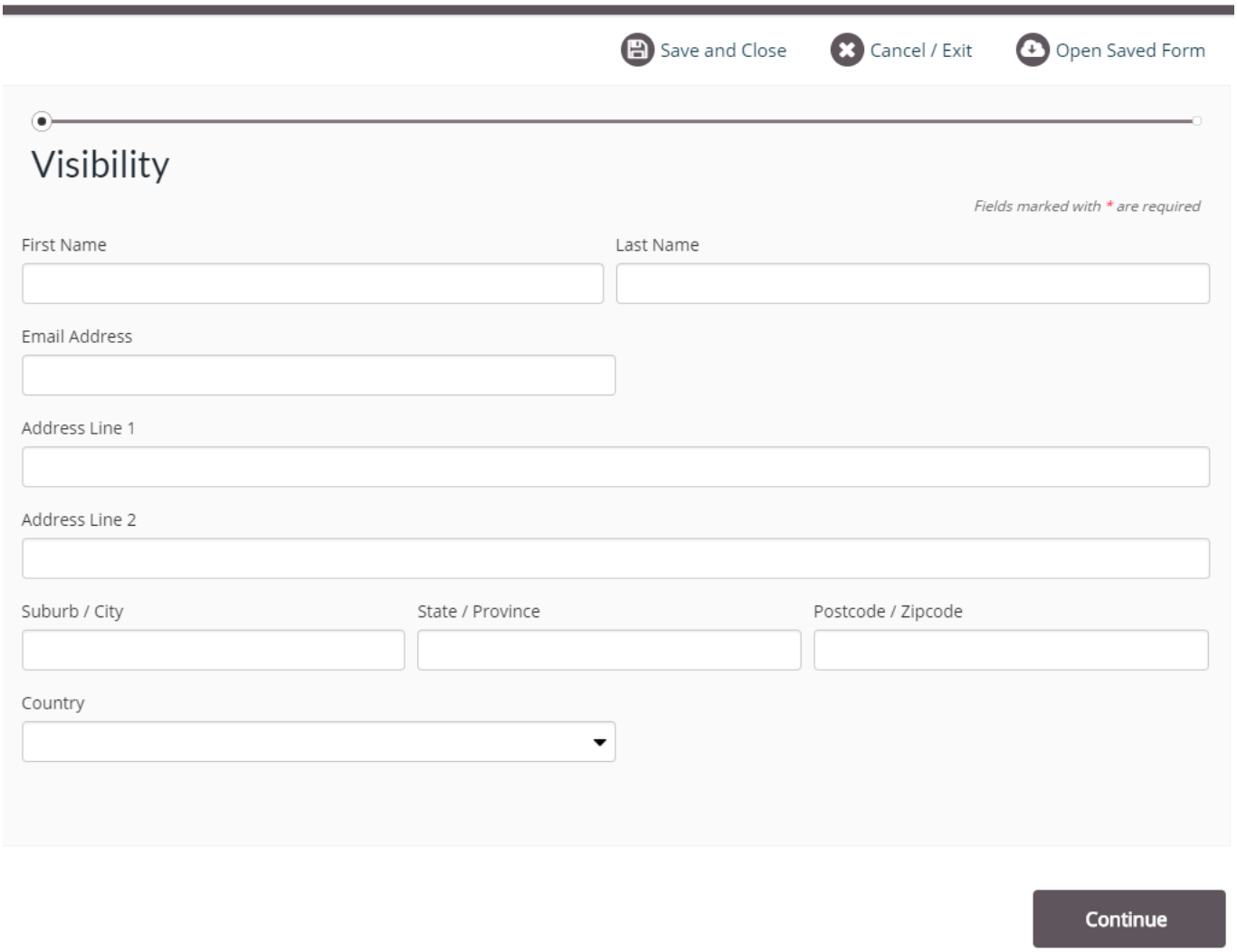


The Section component in the above screenshot has been configured with a Show on Receipt Visibility rule.

The screenshot below displays the script that is applied to the section component.



When the form is saved, built and rendered. The form will NOT show the Section component named For receipts only. The screenshot below displays the built and rendered form.


Save and Close
Cancel / Exit
Open Saved Form
Visibility
Fields marked with * are required
First Name
Last Name
Email Address
Address Line 1
Address Line 2
Suburb / City
State / Province
Postcode / Zipcode
Country
Continue

Create a Show If Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

Use the Show if rule to show a component only when certain conditions have been met, for example provided answers or selections based on previous components. The Show if rule allows you to show components based on the value of other components.

To create a Show if rule, follow these steps:

1. Open a Maestro form.
2. Click the Properties tab.
3. Select the component to show or hide.
4. Click Create Rule.
5. Select Show if.
6. Type a JavaScript that implements the rule or right-click the component that defines whether the selected component is visible or hidden.
7. Select the correct expression.
8. Click Save.
9. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.

NOTE

Check out the Show If rule examples.

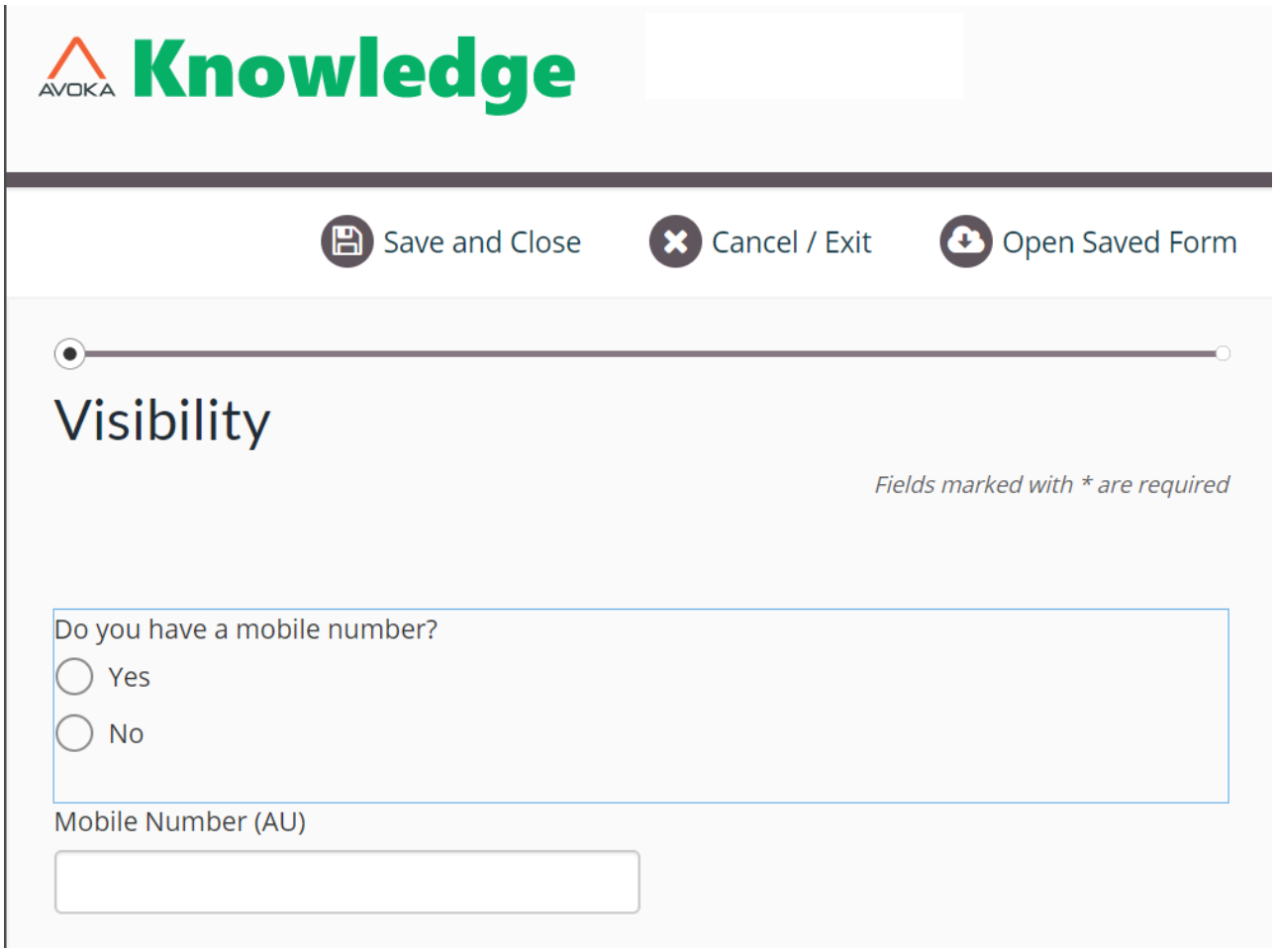
Show If Rule Examples

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

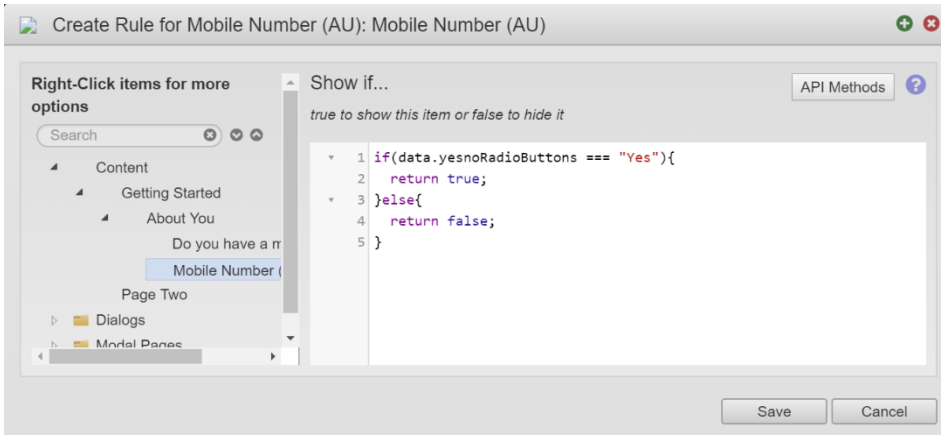
This example demonstrates a Show if to show a Mobile Number (AUS) component only when the answer to the radio button question is Yes. If a form user selects Yes, indicating that they have a mobile number, then a mandatory text field is displayed so the user can enter a mobile phone number.

To create the Show If rule, follow these steps:

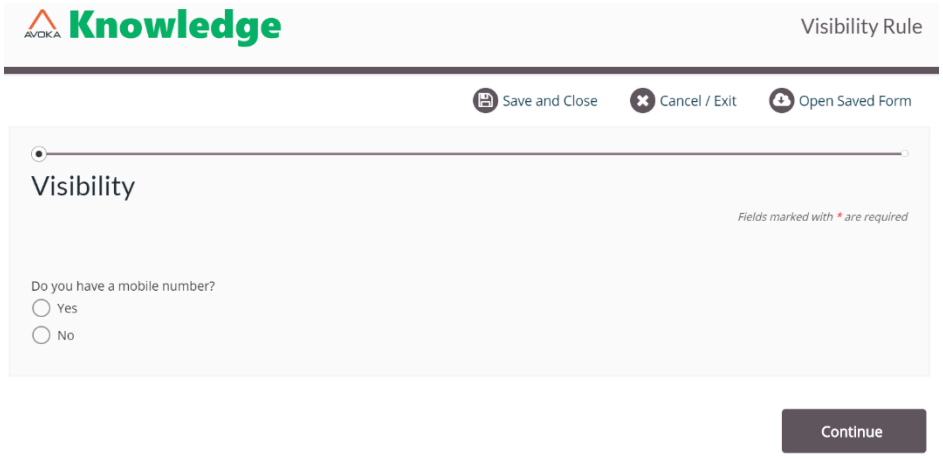
1. Open a Maestro form.
2. Drag a radio button group component to a form.
3. Drag a Mobile Number component to a form.



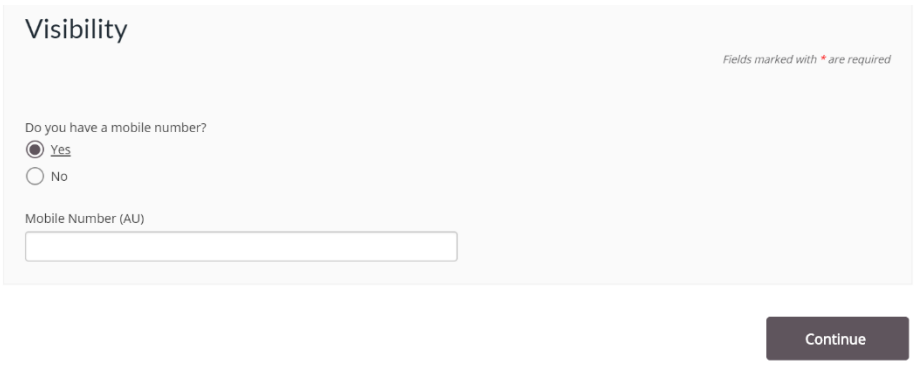
4. Select the component to show only when the Yes option is selected.
5. Click Create Rule.
6. Select Show if.



7. Type a JavaScript that implements the rule similar to the code shown above. If "Yes" is selected in the radio button group, the script will return true and the mobile number component will show in the form, otherwise the script returns false and the mobile number component is not shown on the form.
8. Click Save.
9. Click Build and Render Form to see how this rule is displayed when the form is built and rendered.



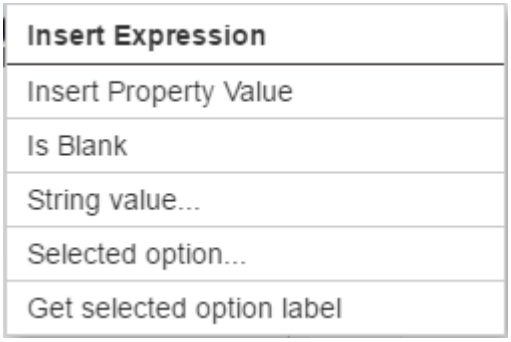
10. If Yes is selected, the Mobile Number component is shown.



Insert Expression in a Visibility Rule

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

You can insert a pre-defined expression into a rule. The options available from the right-click menu are based on the selected component type and may not be relevant to the specific rule you are creating.



The available expressions are:

- Insert Property Value - inserts the value of the selected property, which can be: options, arrangement, popover help text, etc. It's not applicable to the Show if rule.
- Is Blank - creates a Show if rule that displays the hidden component if the selected component is blank.
- Selected option - creates a Show if rule based on one of the options created for the selected component. Use it for radio button groups or dropdown lists.
- Get selected option label - pulls the label from the selected component. It's not applicable to the Show if rule.

Translation

MaestroThe UI design product. | Form Builder | 21.11 This feature was updated in 21.11.

Journey Maestro allows you to configure Maestro forms to support various languages using one of the following methods:

- The Translation functionality of the Maestro editor
- Translation CSV files | 18.05 This feature was introduced in 18.05.

Translation CSV files provide form users and translators with more flexibility when working with and implementing multiple languages in applications. Translations can be configured as one language or multiple languages at a time.

Maestro gives you the ability to specify the source of a translation CSV file using the `Translation.fetch()` function so that it can be stored externally to the form. A service or URL can be used to load the translation, which allows you to maintain translations outside the form without needing to republish it to get updates. | 21.11 This feature was introduced in 21.11.

Maestro comes with the Select Language component, which allows form users to select a language that they want to use on a form.

Next, learn how to configure a new language.

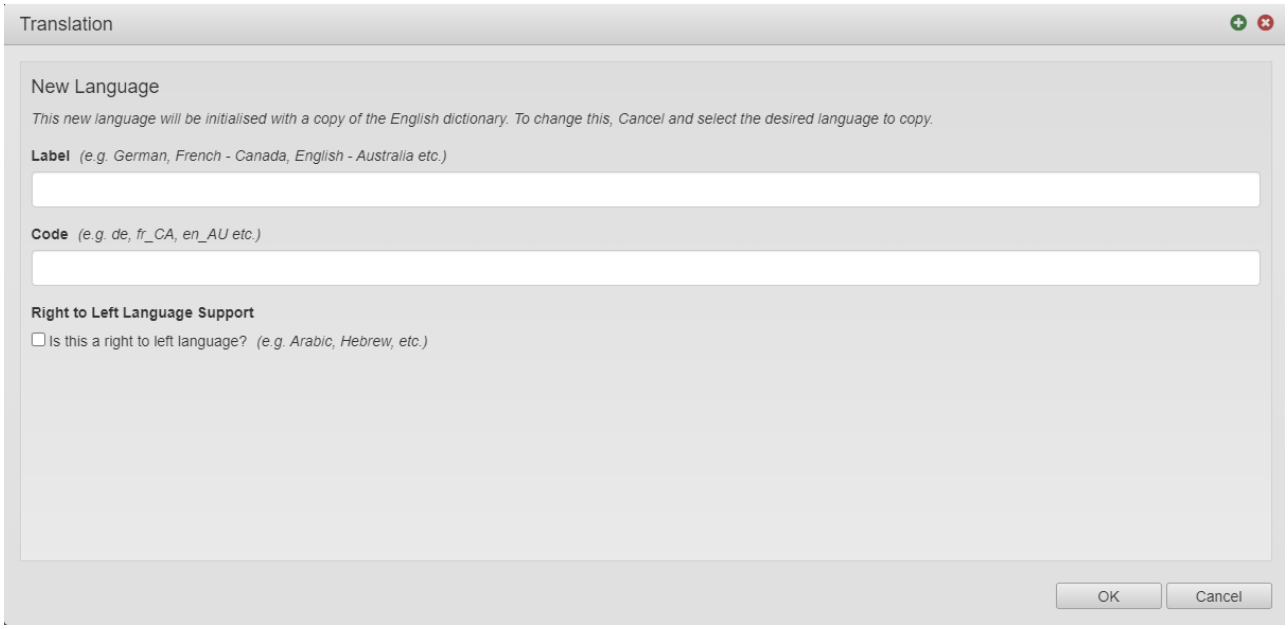
Configure a New Language

MaestroThe UI design product. | Form Builder | 21.11 This feature was updated in 21.11.

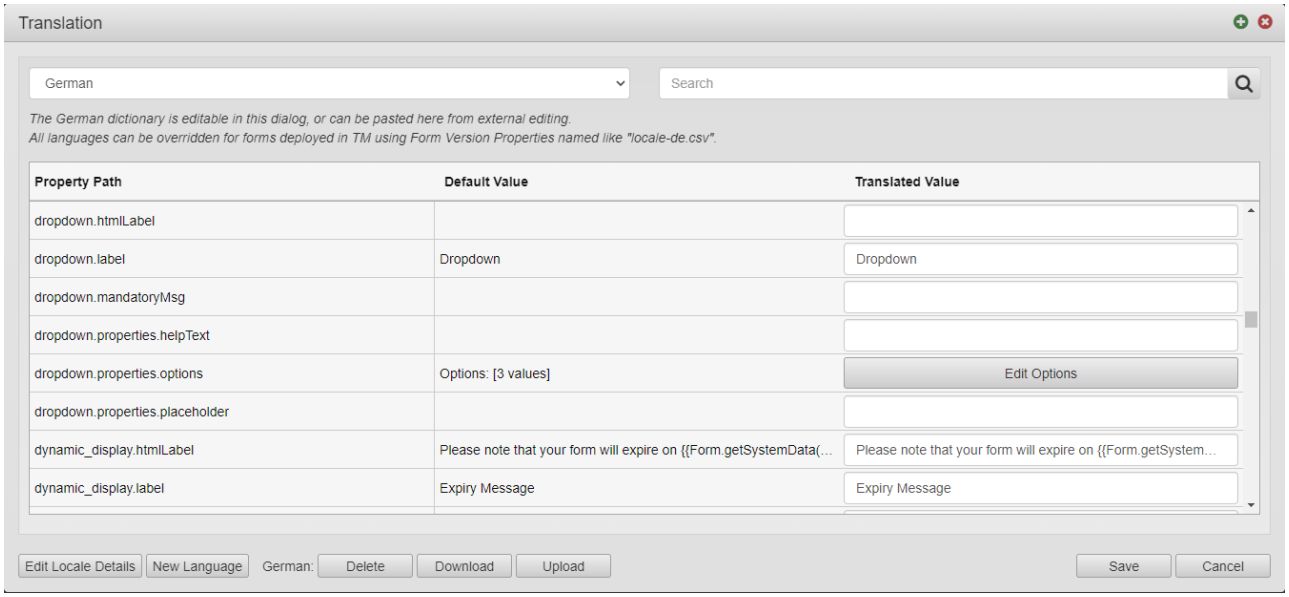
Maestro allows you to translate form's elements into different languages, including right to left languages.

To add a new language:

1. Click New Language to add a language.



2. Enter a language name in the Label field, for example, German.
3. Enter a language code, which is an identifier for the new language, in the Code field, for example, de.
4. Select the Is this a right to left language checkbox to add a right to left language, for example, Arabic.
5. Click OK to add a language.

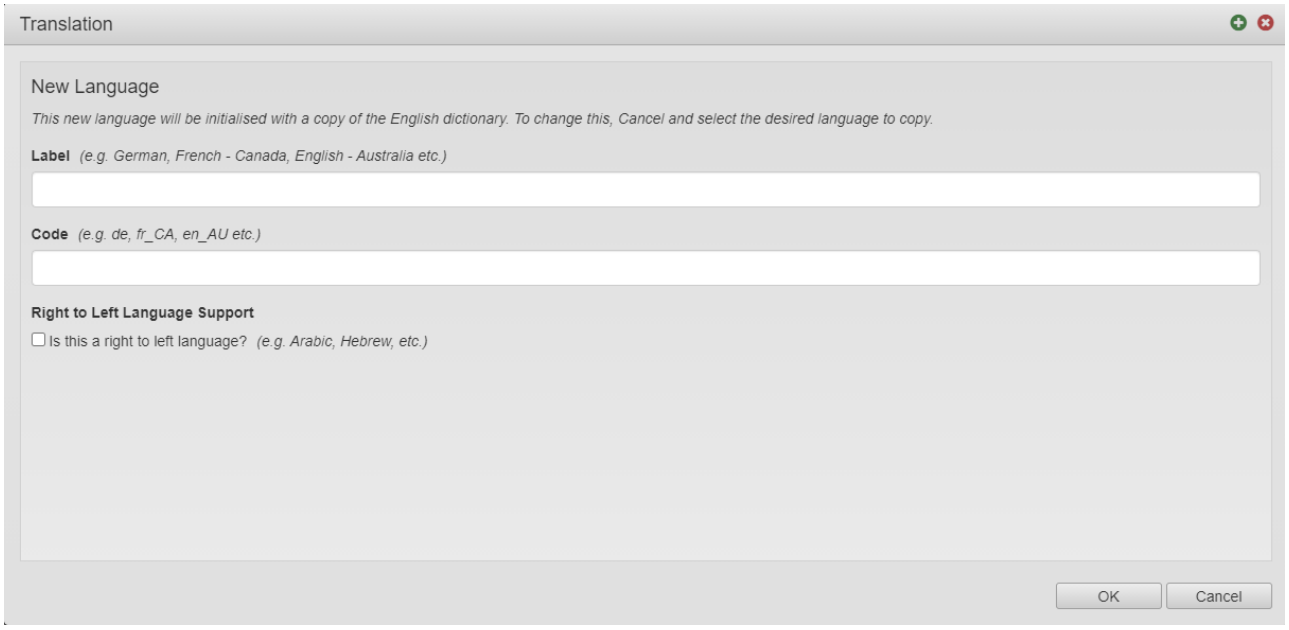


The window now displays an additional column - Translated Value - which you can use when the German language is selected.

6. Enter translated values of the form's elements into the Translated Value column and click Save when you are done.

> NOTE
>
> You can use the search functionality to quickly find the Property Path you want to translate.

7. Click Edit Locale Details to update the name and locale code of the default language.



8. Click Delete to remove the translation for the new language.

   NOTE

   This option is shown only when a non-default language is selected.

9. Click Download to download the translation in the new language as a CSV file on your local PC.

   NOTE

   This option is shown only when a non-default language is selected.

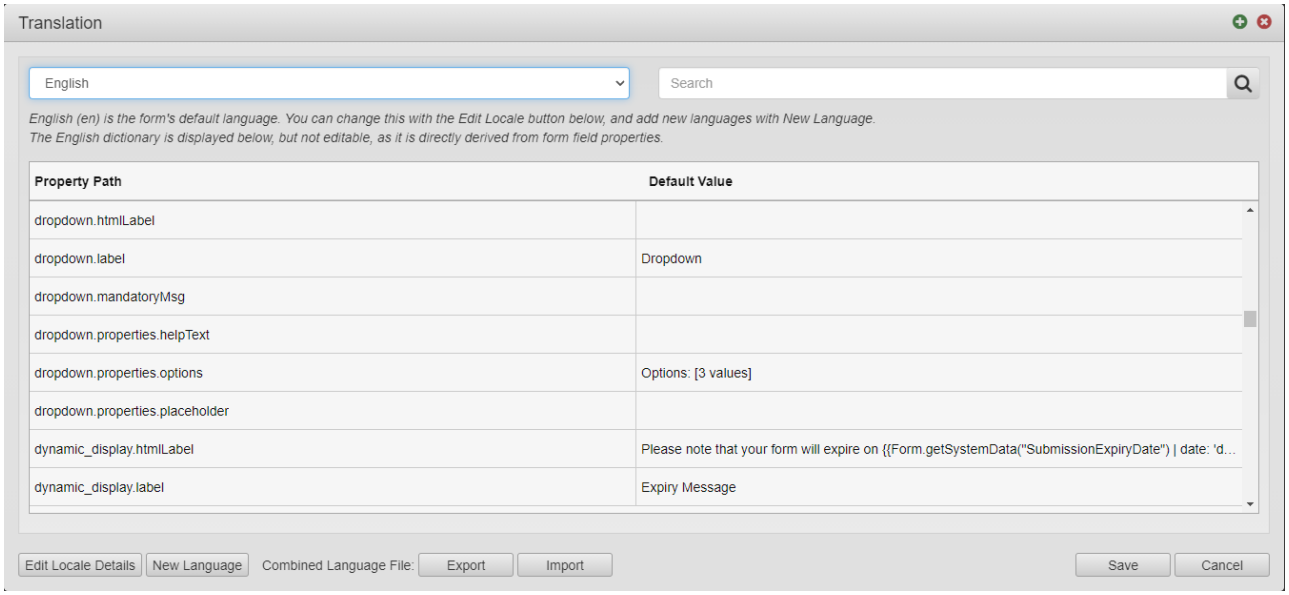
10. Click Upload to upload the translation in the new language as a CSV file from your local PC.

    NOTE

    This option is shown only when a non-default language is selected.

To manage a language:

1. Open a form in the Maestro editor and click Translation on the Toolbar.



The window lists keys for objects within the form, in the Property Path column, that are mapped to labels based on the selected language, the Default Value. The window opens with the default language selected, for example, English.

> NOTE
>
> You are not able to edit the default language in the Translation window, as it is directly derived from the form's component properties.

> NOTE
>
> If you have a component selected when you click Translation, the list is scrolled to the selected Property Path. The scroll position remains as you switch between languages.

2. Select a language from the dropdown list to switch between multiple languages, if more than one language is configured.
3. Type a part of a key name, which are listed in the Property Path column, and click Search to list keys and values based on the search criteria.

4. Click Export to export form's translation as a CSV file.

   NOTE

   This option is shown only when a new language is added.

5. Click Import to import form's translation as a CSV file.

   NOTE

   This option is shown only when a new language is added.

Now, when you have set up the new language, you can configure the form with the Select Language component. The message is shown using a `Form.showProgress` call, for example, `Form.showProgress("Loading...")`. You can right click the string in the rule and turn it into a translatable key. For the save, cancel and submit calls you can change this value on the root form item and it would also be a translatable key.

Since Maestro 21.11, you have ability to specify the source of a translation CSV so that it can be stored externally to the form. A service or URL can be used to load the translation. This allows users to maintain their translations outside the form without needing to republish the form to get updates. You can use the `Translation.fetch()` function to load the translation for the specified language code. If the translation loads, the interface will refresh with the localized terminology. If the translation doesn't load, an error will appear in the JavaScript Console. A simple way to implement it in a form is to add a button that calls `Translation.fetch("myLanguageCode", false, "/web-plugin/resources/mynewCSC.csv")`.This translates the form based on a translation file that's at an external URL. In this example, it's in a portal resource on the Manager server, but it could be anywhere.

Next, learn how to mark for translation.

Translate a String in a JavaScript

MaestroThe UI design product. | Form Builder | 17.10 This feature was updated in 17.10.

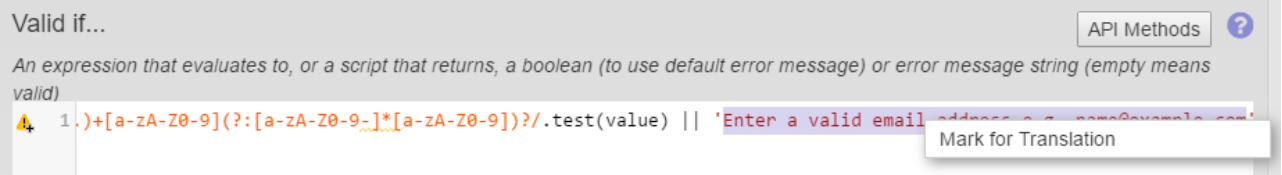
Maestro allows you to translated strings of text used in forms by JavaScripts[1], for example, in business rules.

NOTE

You must configure a new language for a Maestro form before you can translate the string.

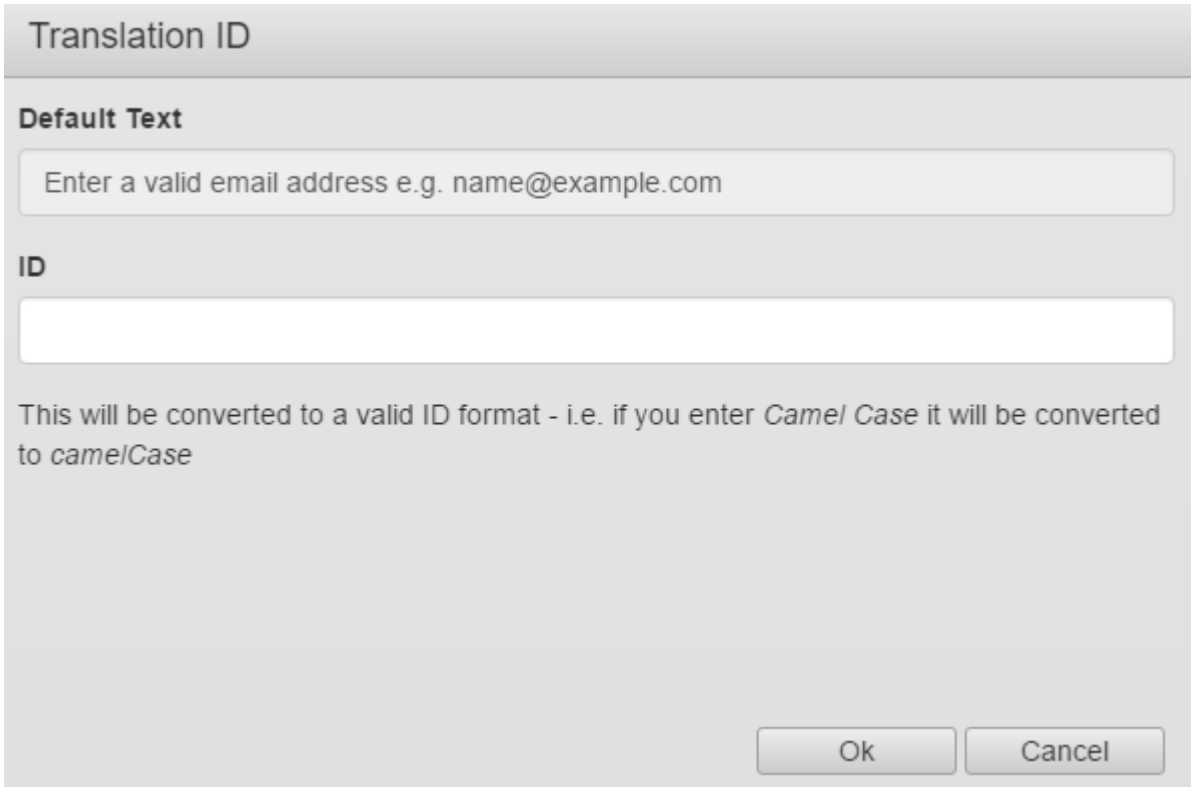
To translate a string:

1. Select a string.
2. Right click it and select Mark for Translation. This creates a translation ID for that string.

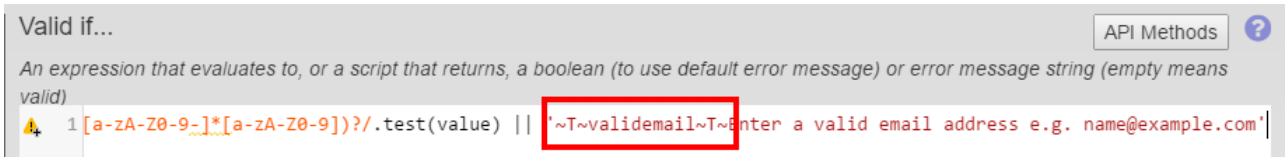


3. Enter a Translation ID for the selected string.

[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.



4. Click OK to create a Translation ID, which you can then find in the Translation window and enter a translation value.



   Now, you can see a token at the beginning of the string. In the code, you can still see the default value for your string, and you can see that it has been marked with a token based on the Translation ID that you provided.

5. Open the Translation window and select the language, which you want to use to edit a value, from the dropdown list. With the language selected you can find the new property key based on the Translation ID that was created. You will see the default value in the middle column and can enter the translation value in the field provided. Notice the translation value displays the default value, however, you can change that value with the correct translation.

Next, learn how to use translation CSV files.

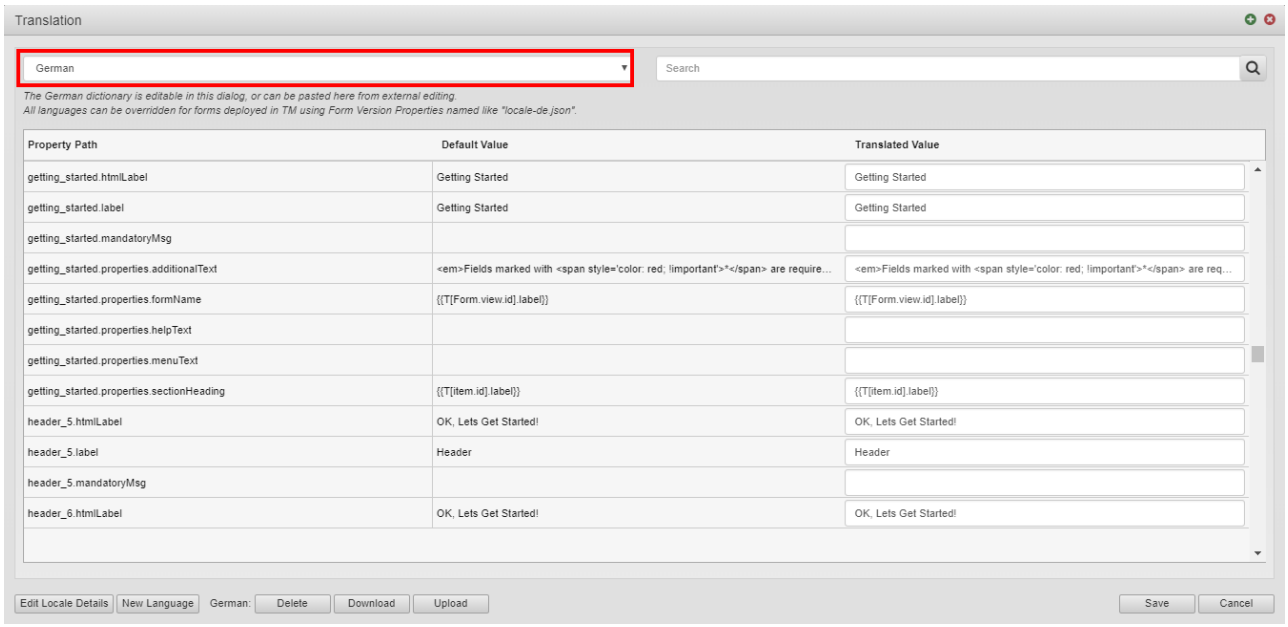
Translation CSV Files with One Language

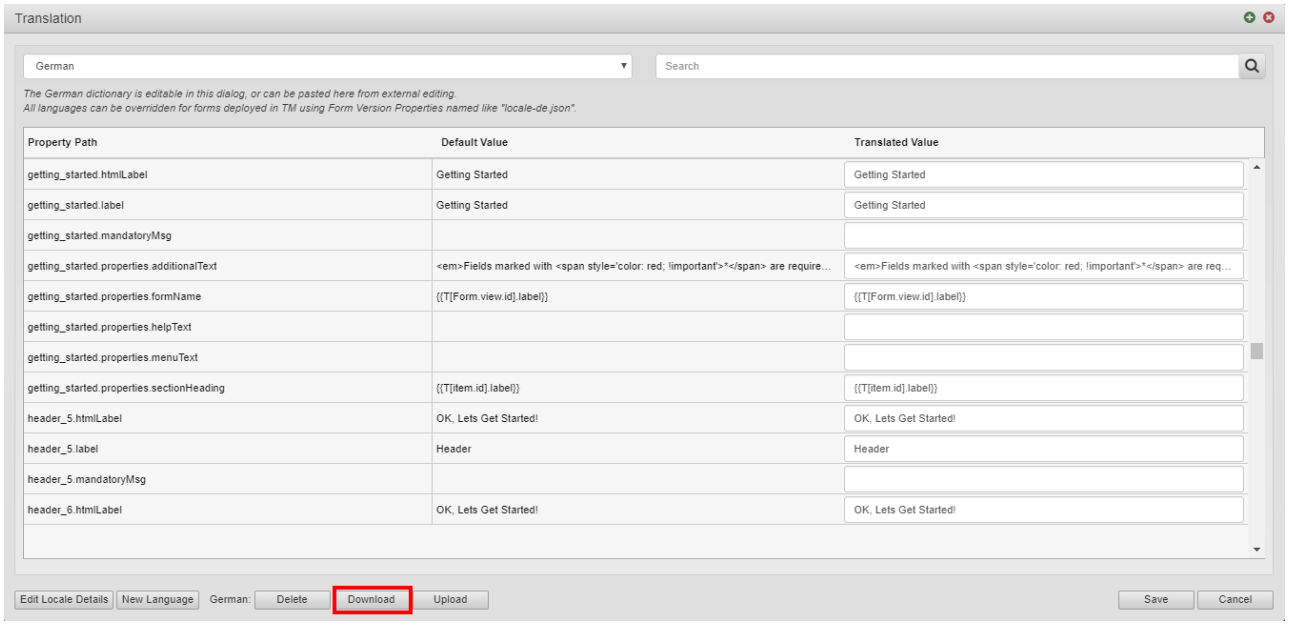
MaestroThe UI design product. | Form Builder | 18.05 This feature was introduced in 18.05.

Journey Maestro allows you to edit translation files containing only one language by downloading the CSV translation file, editing it, and uploading it back into the Maestro form.

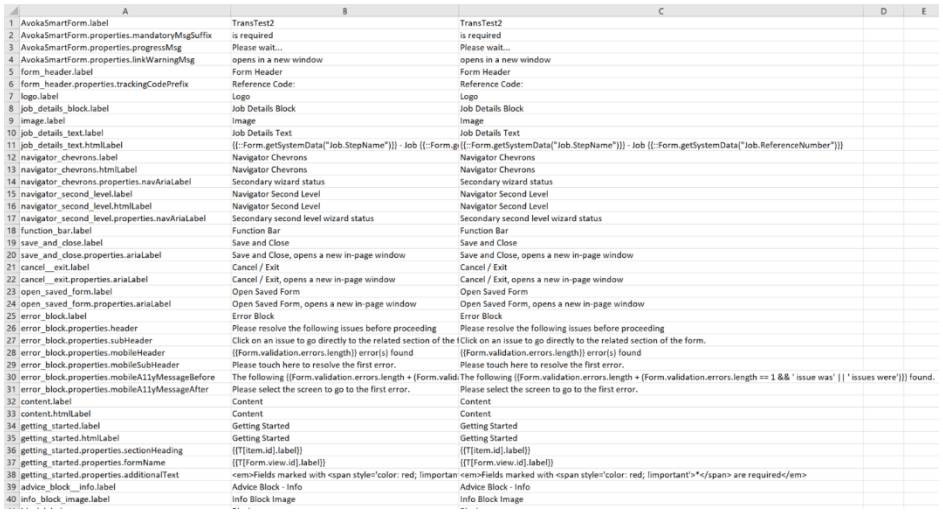
To download, edit and upload a translation CSV file with one language:

1. Open a form with multiple languages configured.
2. Select Translation.
3. Select a language using the language dropdown menu and click Download. The languages in this list are established from the languages configured in Maestro. Clicking Download will download the German CSV translation file (normally named locale-xx.csv though the name of the file is arbitrary) to your computer.





4. Open the downloaded CSV file in any CSV editing tool, for example, MS Excel, as shown below.



	A	B	C	D	E
1	AvokaSmartForm.label	TransTest2	TransTest2		
2	AvokaSmartForm.properties.mandatoryMsgSuffix	is required	is required		
3	AvokaSmartForm.properties.progressMsg	Please wait...	Please wait...		
4	AvokaSmartForm.properties.linkWarningMsg	opens in a new window	opens in a new window		
5	form_header.label	Form Header	Form Header		
6	form_header.properties.trackingCodePrefix	Reference Code:	Reference Code:		
7	logo.label	Logo	Logo		
8	job_details_block.label	Job Details Block	Job Details Block		
9	image.label	Image	Image		
10	job_details_text.label	Job Details Text	Job Details Text		
11	job_details_text.htmlLabel	{{::Form.getSystemData("Job.StepName")}} - Job {{::Form.g	{{::Form.getSystemData("Job.StepName")}} - Job {{::Form.getSystemData("Job.ReferenceNumber")}}		
12	navigator_chevrons.label	Navigator Chevrons	Navigator Chevrons		
13	navigator_chevrons.htmlLabel	Navigator Chevrons	Navigator Chevrons		
14	navigator_chevrons.properties.navAriaLabel	Secondary wizard status	Secondary wizard status		
15	navigator_second_level.label	Navigator Second Level	Navigator Second Level		
16	navigator_second_level.htmlLabel	Navigator Second Level	Navigator Second Level		
17	navigator_second_level.properties.navAriaLabel	Secondary second level wizard status	Secondary second level wizard status		
18	function_bar.label	Function Bar	Function Bar		
19	save_and_close.label	Save and Close	Save and Close		
20	save_and_close.properties.ariaLabel	Save and Close, opens a new in-page window	Save and Close, opens a new in-page window		
21	cancel__exit.label	Cancel / Exit	Cancel / Exit		
22	cancel__exit.properties.ariaLabel	Cancel / Exit, opens a new in-page window	Cancel / Exit, opens a new in-page window		
23	open_saved_form.label	Open Saved Form	Open Saved Form		
24	open_saved_form.properties.ariaLabel	Open Saved Form, opens a new in-page window	Open Saved Form, opens a new in-page window		
25	error_block.label	Error Block	Error Block		
26	error_block.properties.header	Please resolve the following issues before proceeding	Please resolve the following issues before proceeding		
27	error_block.properties.subHeader	Click on an issue to go directly to the related section of the	Click on an issue to go directly to the related section of the form.		
28	error_block.properties.mobileHeader	{{Form.validation.errors.length}} error(s) found	{{Form.validation.errors.length}} error(s) found		
29	error_block.properties.mobileSubHeader	Please touch here to resolve the first error.	Please touch here to resolve the first error.		
30	error_block.properties.mobileA11yMessageBefore	The following {{Form.validation.errors.length + (Form.valid	The following {{Form.validation.errors.length + (Form.validation.errors.length == 1 && ' issue was' || ' issues were')}} found.		
31	error_block.properties.mobileA11yMessageAfter	Please select the screen to go to the first error.	Please select the screen to go to the first error.		
32	content.label	Content	Content		
33	content.htmlLabel	Content	Content		
34	getting_started.label	Getting Started	Getting Started		
35	getting_started.htmlLabel	Getting Started	Getting Started		
36	getting_started.properties.sectionHeading	{{T[item.id].label}}	{{T[item.id].label}}		
37	getting_started.properties.formName	{{T[Form.view.id].label}}	{{T[Form.view.id].label}}		
38	getting_started.properties.additionalText	Fields marked with <span style='color: red; importan	Fields marked with * are required		
39	advice_block__info.label	Advice Block - Info	Advice Block - Info		
40	info_block_image.label	Info Block Image	Info Block Image		

The first column represents the labels of the form, the second column contains the base language and the third column is where the translated language will display. If the translated language has not been entered yet, then the base language will appear as a placeholder in the third column and should be updated with the new language.

5. Edit the CSV file by entering translated values into the third column. For example, Hello translated to German is Hallo.

NOTE

Maestro does not support multiple sheets in a single workbook, so when you edit the CSV file, it is recommended that you DO NOT create/use multiple sheets.

6. Save the CSV file. The name of the file is not important but the file type (CSV) is.

NOTE

If your translation (language) uses non-ASCII characters, you need to "Save as" a UTF-8 CSV File.

NOTE

ASCII characters include the digits 0 to 9, lowercase letter a to z, uppercase letters A to Z, and punctuation symbols (e.g. %,$,*,&). More information on ASCII can be found at https://www.computerhope.com/jargon/a/ascii.htm

7. Navigate back to the Translation Window (in Maestro) and click Upload.
8. Navigate to the location of the saved CSV file on your computer and select it as an upload option.
   Selecting the CSV file will upload it to the opened form.
9. Click Save to update the changes. It is recommended that you click Save to ensure your changes are saved in the form.

Next, learn about translation CSV files for multiple languages.

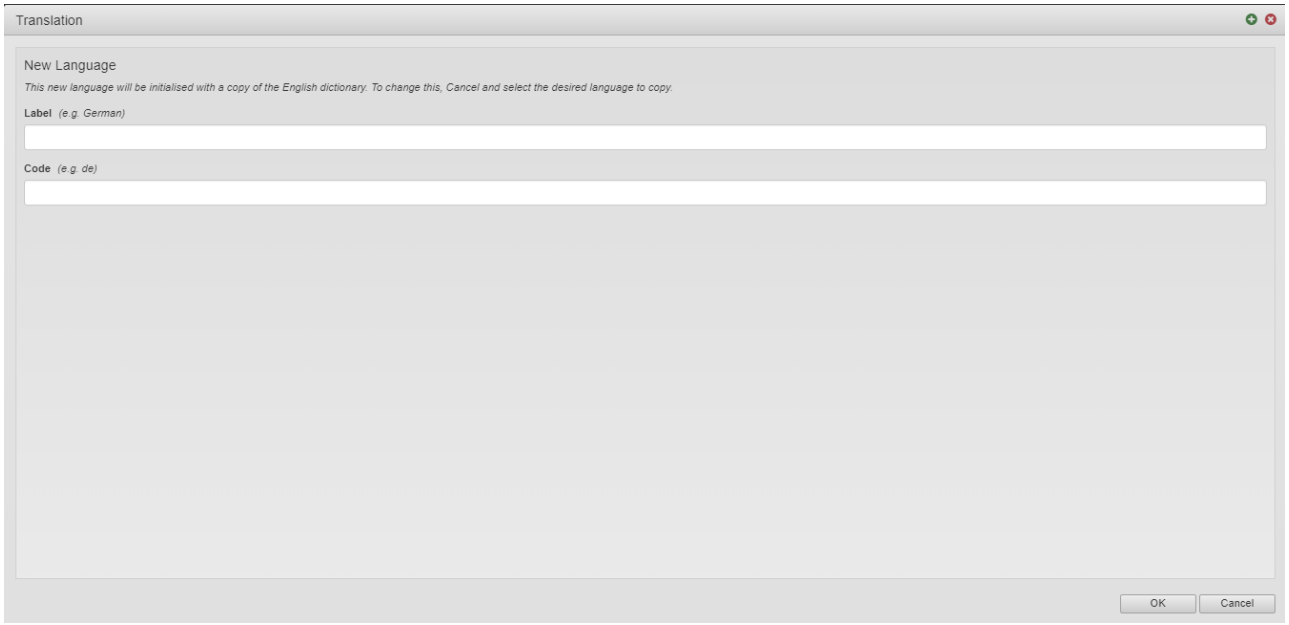
Translation CSV Files with Multiple Languages

MaestroThe UI design product. | Form Builder | 18.05 This feature was introduced in 18.05.

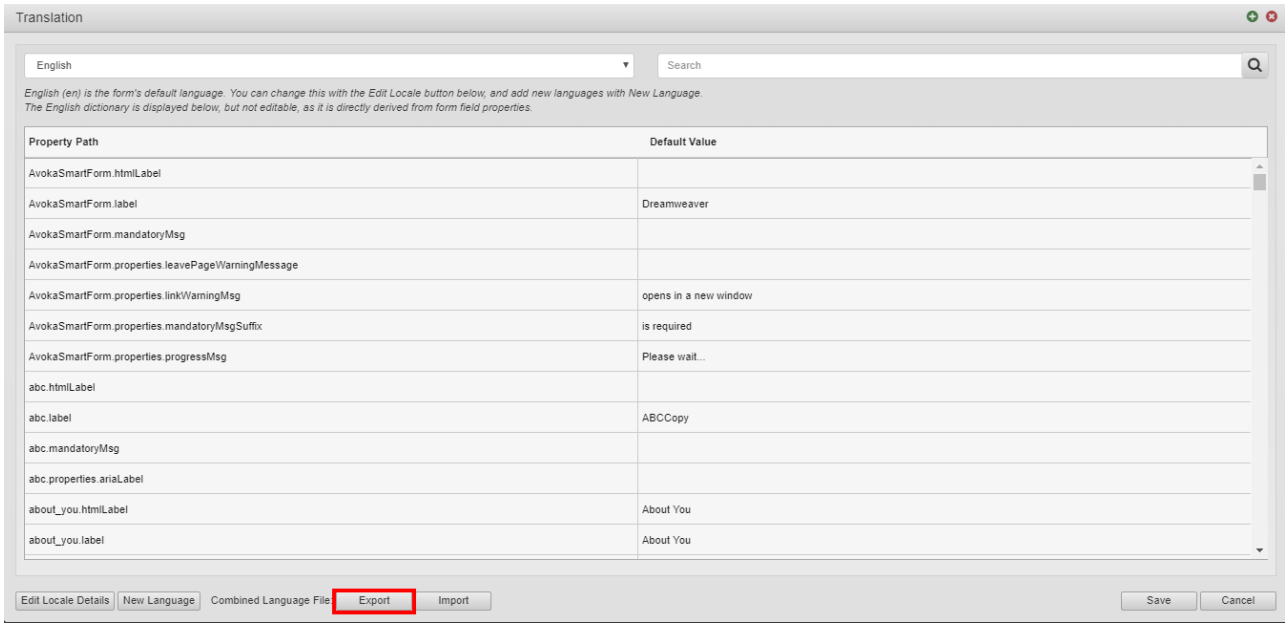
Journey Maestro allows you to edit translation files containing not only one but multiple languages by downloading the CSV translation file, editing it, and uploading it back into the Maestro form.

To export, edit and upload the CSV file with all configured languages:

1. Open a form with multiple languages configured.
2. Select Translation.
3. Click New Language if you want to add new languages to the CSV file.



4. Enter a Label and Code for the new language, for example, Hindi (Label) and hi (Code).
5. Click OK.
6. Click Export from the Combined Language Section of the window.



7. Open the CSV file in any CSV file editor tool, such as MS Excel, as shown below.



Each column contains a different language. A blank column will be created for the newly created language.

8. Edit the CSV file to contain the translations for the newly created language or edit any of the languages that have already been configured.

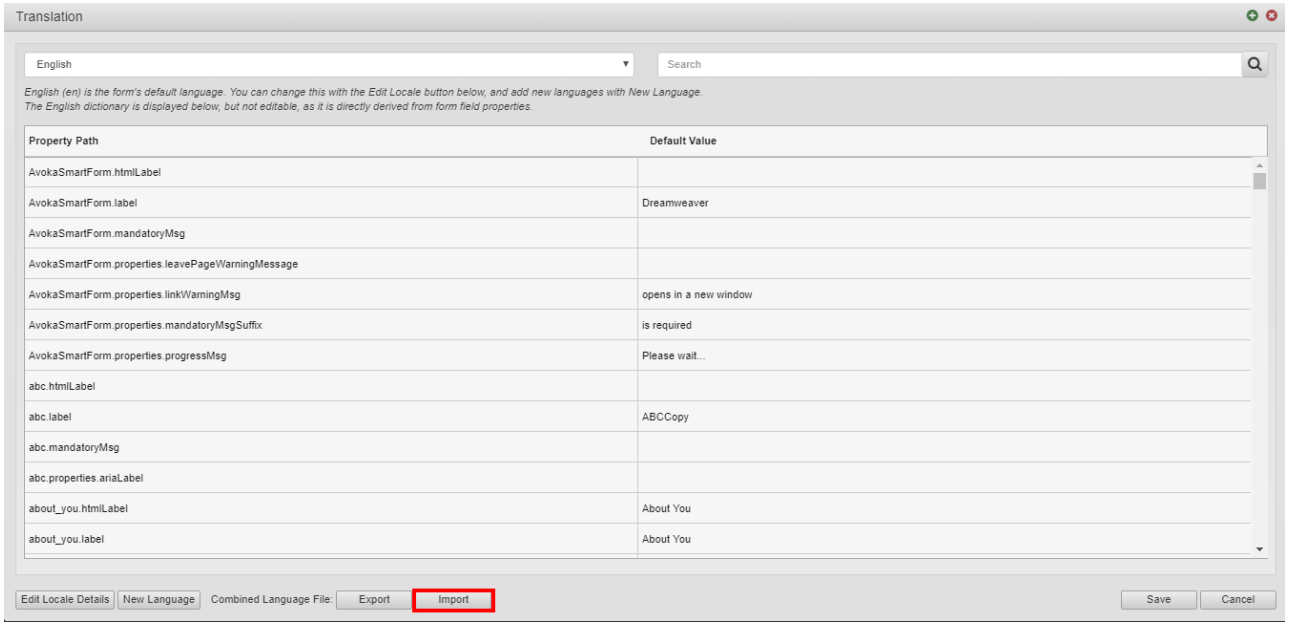
> NOTE
>
> Maestro currently does not support multiple sheets on a single spreadsheet so when you edit the CSV file, it is recommended that you DO NOT create/use multiple sheets in a single CSV file.

9. Save the CSV file. The name of the file is not important but the file type (CSV) is.

> NOTE
>
> If your translation uses non-ASCII characters, you need to "Save as" a UTF-8 CSV File.

> NOTE
>
> ASCII characters include the digits 0 to 9, lowercase letter a to z, uppercase letters A to Z, and punctuation symbols (e.g. %,$,*,&). More information on ASCII can be found at https://www.computerhope.com/jargon/a/ascii.htm

10. Click Import from the Combined Language File option.



11. Select the edited CSV file from your computer.
12. Save your form. Once you save and build the form, the changes you made to the translation CSV file will be reflected in the form.

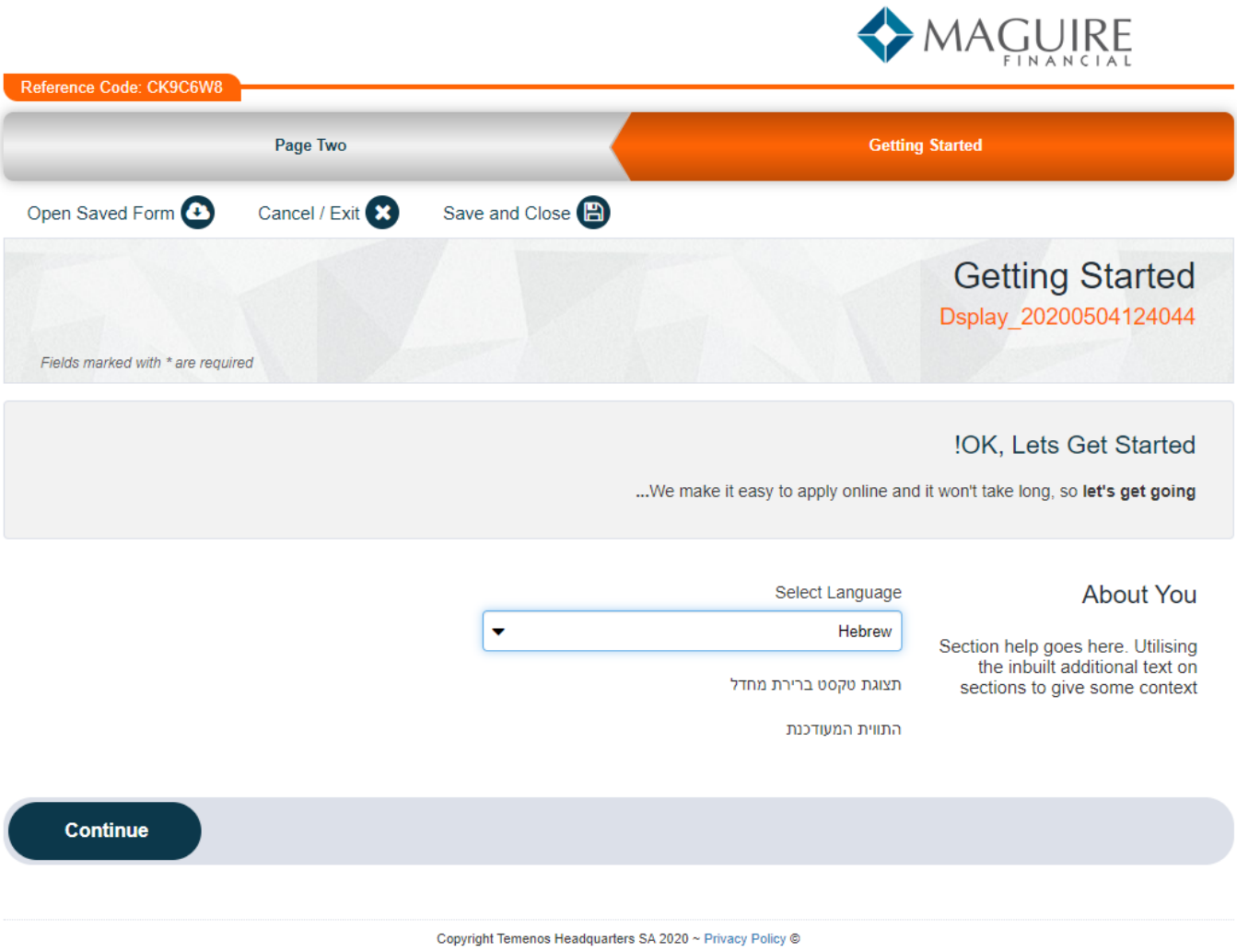
Next, learn about translation CSV files for individual languages.

Maestro Right-to-Left Language Support

MaestroThe UI design product. | Form Builder Template Designer | 20.11 This feature was introduced in 20.11.

The Maestro translation feature has been enhanced to now support right-to-left languages, such as Hebrew and Arabic. You can use this feature to translate a single form into any language that helps open up new markets especially in the EMEA region as well as support existing clients wishing to reach a wider customer base.

A sample Maestro form supporting right to left languages is shown below. Select Hebrew language from the Select Language dropdown list to translate and render the form for the right to left language.



Next, learn how to set up a new language in Maestro.

Entity Overview

MaestroThe UI design product. | Form Builder Platform Developer | 17.10 This feature was introduced in 17.10.

Maestro uses entities to model the data in a form by grouping components in a way you need to implement your business requirements. This provides several advantages, including more meaningful field identifiers that make for clearer business logic code and support refactoring across component and template definition changes in a more resilient, deterministic fashion. The entities also provide a streamlined way of marshalling data objects between client and server code.

Entities define a domain model of a form and they are connected to properties of components and other entities. The entities of the domain model are stored in the form manifest as a JSON[1] object containing a set of name and value pairs, each representing a real-world concept, such as a street name. The example of JSON definition, stored in a model-schema.json file is presented below:

```
{
	"title": "Model",
	"type": "object",
	"properties": {
		"applicant": {
			"type": "object",
			"properties": {
				"firstName": {
					"type": "string"
				},
				"lastName": {
					"type": "string"
				}
			}
		}
	}
}
```

[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.

The name in each pair identifies an entity while the value is either empty (default) or another object which has its own name and value pairs for child entities. The example of an entity in JSON format is shown below:

```
{
        applicant:
                firstName: "//Applicant/FirstName",
                lastName: "//Applicant/LastName"
}
```

You can create and manage entities in the Maestro editor using the Model pane or a component's ID field in the Properties pane.

The Maestro editor knows that a component is an entity by checking for an underscore in a component ID field. If you add a prefix ending with an underscore to the component ID, Maestro adds the entity and the data item (property) to the domain model. For example, the prefix `applicant1_` applied to the `firstName` component results in the component ID `applicant1_firstName`, which identifies that the `firstName` component belongs to the `applicant1` entity.

Any existing component IDs, including those with an underscore, will not cause entities to be created. An entity is only created using the Model pane or when a user modifies a component ID.

Next, learn how to create an entity.

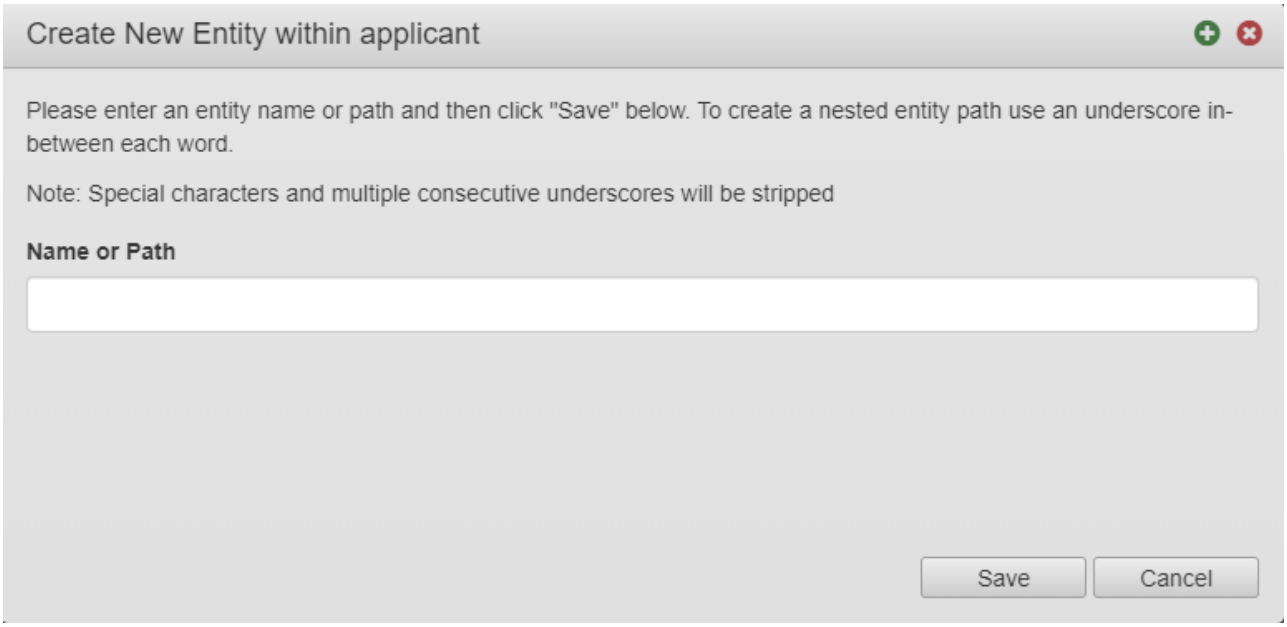
Create an Entity

MaestroThe UI design product. | Form Builder Platform Developer | 17.10 This feature was introduced in 17.10.

Maestro allows you to create new entities and map them to form's fields. This provides logical groupings of fields regardless of their physical position in the form. You can create a new entity using either the Model pane or the Properties pane. This is described below.

To create a new entity in the Model pane:

1. Open a form in the Maestro editor.
2. Click the Model pane and expand the Entities root element.
3. Select an existing entity or select the Entities root element, if there are no entities available, and right-click on it.
4. Select Add Entity from the dropdown menu to add a new entity to the selected element.

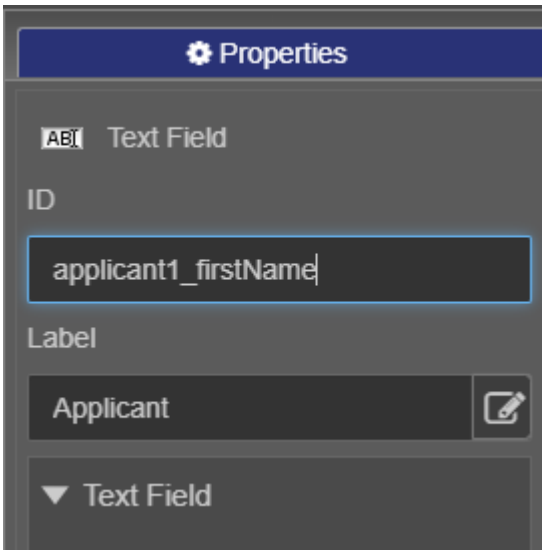


5. Enter an entity name or a path in the Name or Path field. You can create a nested entity by adding underscores in the entity name, for example, `applicant_name`. Maestro recognizes the underscores as entity separators and creates all the required entities for your nested entity.
6. Click Save to update the changes.

To create a new entity in the Properties pane:

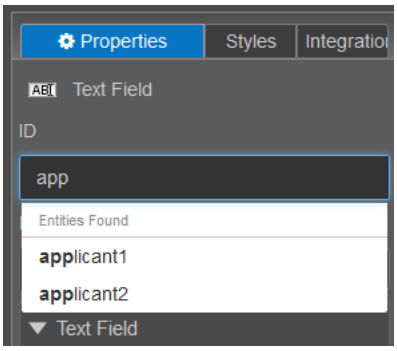
1. Open a form in the Maestro editor.
2. Select a field of a component and click the Properties pane.

3. Enter an entity name in the ID property field using the underscore "_" to logically separate a component ID name, for example `applicant1_firstName`.

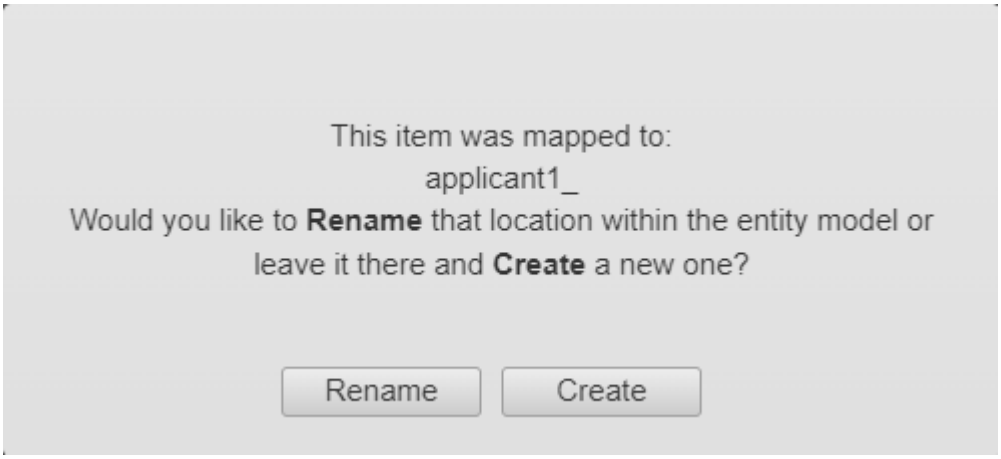


> NOTE
>
> You can have many levels of entity's hierarchy. The easiest way is to include as many underscores in the name as you need, for example, `application_applicant_clientProfile_userName`.

To speed up the entity creation process, Maestro looks up all existing entities as you type a new entity name in the ID field, so you can select a name you need from the list.



4. Click Enter. Maestro displays a message informing you that a new entity will be created. For example, the prefix `applicant1` applied to the `firstName` component results in the component ID `applicant1_firstName`, which tells Maestro that the `firstName` component belongs to the applicant1 entity.



5. Click Save to update the changes.

Next, learn how to delete an entity.

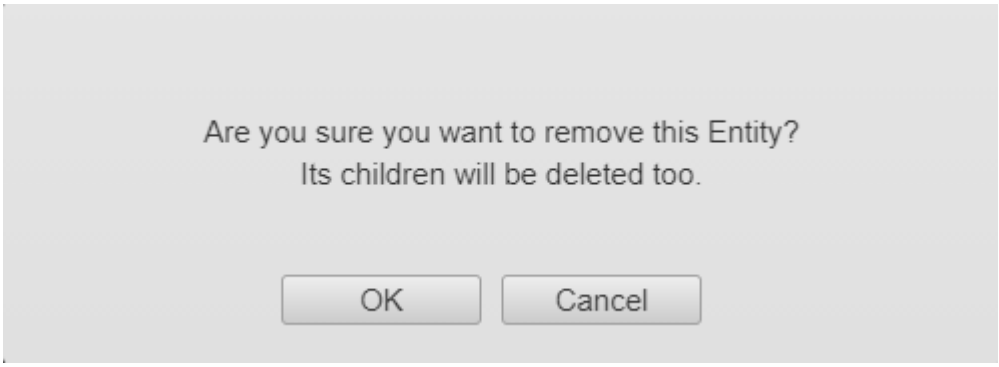
Delete an Entity

MaestroThe UI design product. | Form Builder Platform Developer | 17.10 This feature was introduced in 17.10.

Maestro allows you to delete entities and entities' properties so you can clean up your data model.

To delete an entity in the Model pane:

1. Open a form in the Maestro editor.
2. Click the Model pane and expand the Entities root element.
3. Find an existing entity and right-click on it.
4. Select Delete Entity from the dropdown menu to remove the entity and all children as well. Click OK to confirm.



NOTE

Maestro will not delete any form fields that are mapped to the entity.

5. Select Delete Property from the dropdown menu to remove a property of the entity. Click OK to confirm.

NOTE

Maestro will not delete any form fields that are mapped to the entity.

Next, learn how to apply an entity path of a component to its children.

Apply an Entity Path of a Component to its Children

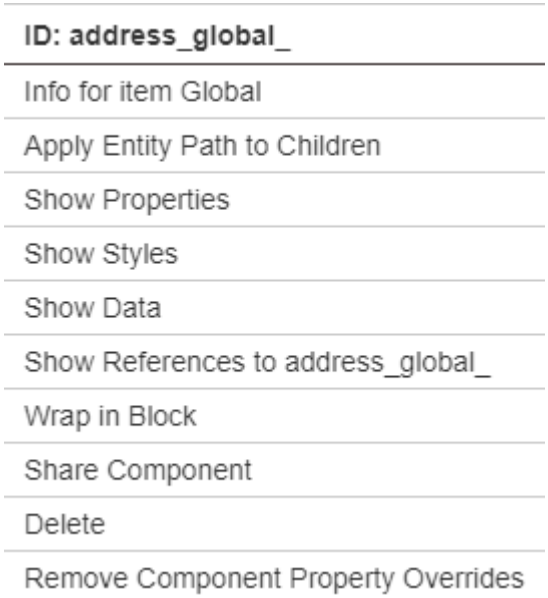
MaestroThe UI design product. | Form Builder ▪ Platform Developer | 17.10 This feature was introduced in 17.10.

Some components, such as an address component, are made up of multiple components, which we call children components. Maestro allows you to apply the same entity path to all children components that make up such a component. For example, if an address component is dragged onto a form and an `address` entity is created, the street, suburb, state and postcode children components, which make up the parent component, can be associated with the address entity.

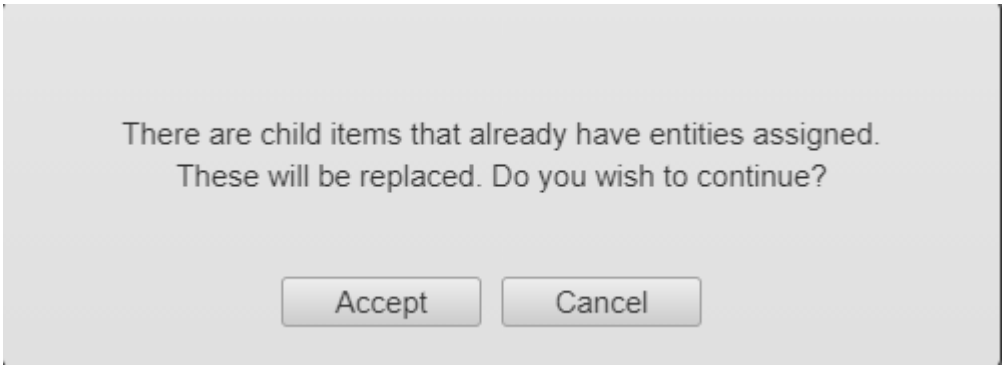
For an entity path to be applied to the children of a component when an entity is created, you must end the entity name with an underscore; for example, applicant_address_. This underscore at the end tells Maestro where it fits in the domain model configured for the form.

To apply a component's entity path to its children components:

1. Open a form in the Maestro editor.
2. Select a component that contains multiple children components.
3. Create an entity with a trailing underscore, for example, `applicant_address_`.
4. Select the component in the View pane and right-click on it.
5. Select Apply Entity Path to Children from the dropdown menu.



6. Click Accept to confirm the changes.



7. Open the Model pane to verify the new entities have appeared in the domain model.

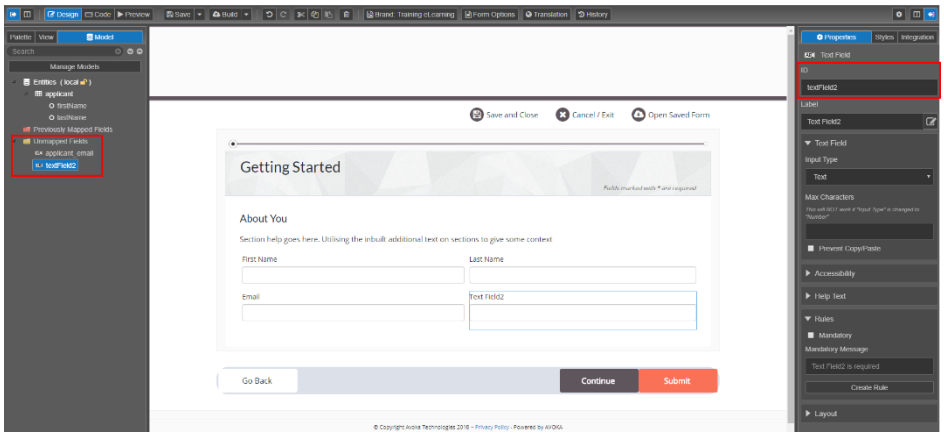
Next, learn how to map a component to an entity property.

Map a Component to an Entity Property

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 18.05 This feature was introduced in 18.05.

As you drag and drop components on a form in the Maestro editor, Maestro populates them with the default data that has a flat XML[1] structure, in the same order as the visual layout. Using entities provides you with control over the structure of the form data, allowing fields to form logical groupings regardless of their physical position in the form. Additionally, fields can be mapped to a pre-defined entity model, guaranteeing the structure of submitted data.

When you add a new component, identified by its ID, to a form without mapping it to an entity or an entity property, it appears in the Unmapped Fields folder of the Model pane. There it stays until it is mapped to an entity. This is illustrated below.



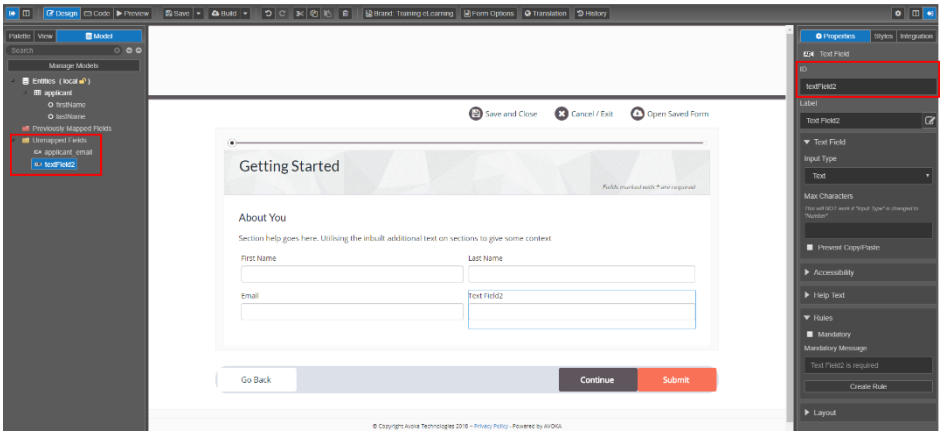
When entity's properties are created, they are not saved within the data of the form, instead, they are saved in the domain model, which holds all data. This means that even if you delete the component from the form, the entity property are still accessible to the form so you can continue mapping other components to it.

To map a component to an entity property:

1. Open a form in the Maestro editor.
2. Select a component on the form or select it from the Unmapped Fields folder.

[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.

3. Select the Properties pane and update a component ID in the ID field.



Use the underscore "_" between an entity name and the name of the property, belonging to this entity, that you want to create.

For example, if you want to create an entity property called `firstName` for an entity called `applicant`, you must type `applicant_firstName` as a ID of the component. Maestro will create a corresponding entity property, using the underscore as a delimiter, to map it to the component model and also populates the label for the selected component.

> NOTE
>
> If you already have entities defined, select the field first and then double-click the entity's property you want to map it to.

4. Click Rename to confirm the change of ID name.

> NOTE
>
> If you delete the text field, the `firstName` entity property is still available to be mapped to another component. See the explanation above.

> NOTE
>
> You can use Data Mapper to load an existing XML file with data definition and then map form fields to submission data.

Map a Repeating Block Component

Our forms often use repeating block components. It's crucial that a repeat element is mapped to an entity so you need to map it to your submission data model, which can be an existing XML structure. The process is very similar to that of described above, with a few additional steps.

Let's assume you need to configure a repeating block template so a submitted form produces the following XML file:

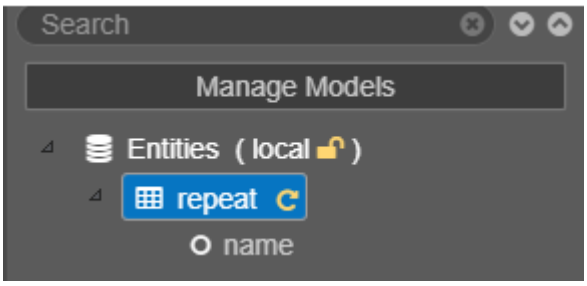
```
<applicant>
    <repeatingFields>
        <individual>
            <name>John</name>
            <age>40</age>
        </individual>
    </repeatingFields>
    <repeatingFields>
        <individual>
            <name>Mark</name>
            <age>47</age>
      </individual>
    </repeatingFields>
</applicant>
```

To map a repeating block component to create a the above XML structure:

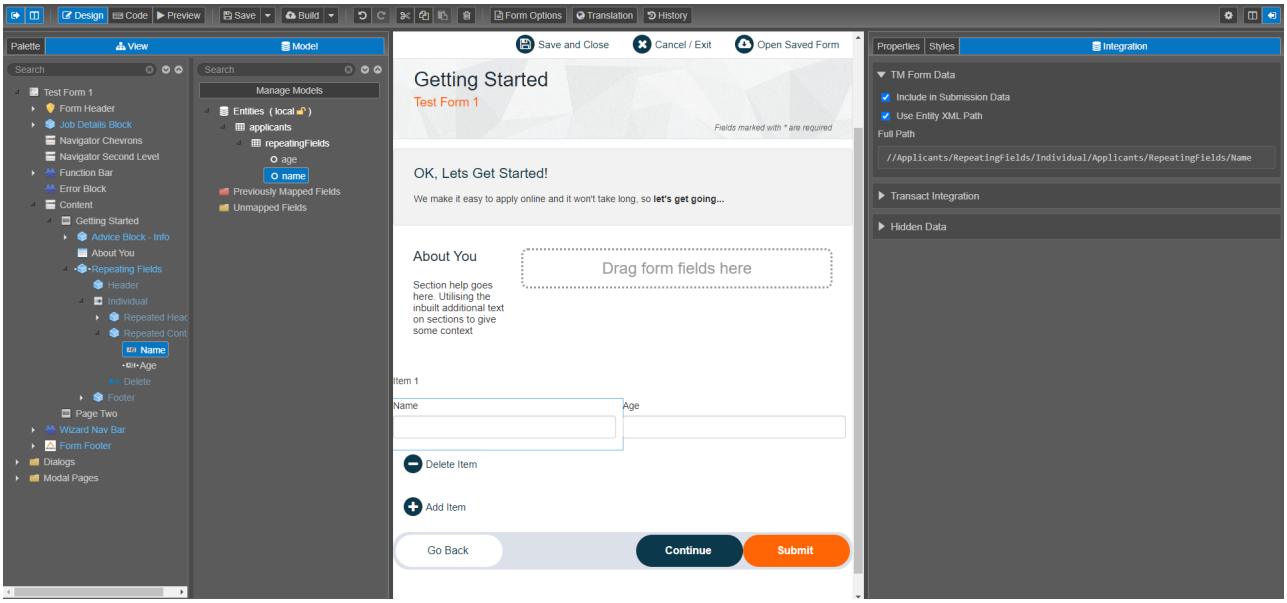
1. Select a repeating block template and drag it onto a form.
2. Select it and update its name in the ID field according to your requirements. For example, `applicant_repeatingFields_`.

   > NOTE
   >
   > The name must end with the underscore "_", so Maestro knows it's a repeating pattern.

3. Open the Model pane to verify a new entry has appeared under Entities. Check the yellow arrow in the entity tree to verify that the element is mapped correctly.



4. Select the Content element of the repeating block template and update it as needed, for example, `Individual`.
5. Open the Palette pane and drop a Text Field component onto the repeating block template component.
6. Change the Text Field component ID to `applicant_repeatingFields_name`. Maestro will prefill this field for you so you only need to update the last part of the name after "_".
7. Click Rename to confirm the name change.
8. Do same for another text field, that is called `applicant_repeatingFields_age`.
9. Select the Integration tab to verify the XML path is correct in Full Path field.



> NOTE
>
> If the path is incorrect, clear the Use Entity XML Path checkbox and select it again.

10. Save the form and click Preview.
11. Fill in the form and click Submit.
12. Click Show form XML to view the resulting XML structure of the repeating block component and compare it with your required XML structure.

13. Click Design to continue working with your form.

Next, learn about the domain model.

Domain Model Overview

MaestroThe UI design product. | Form Builder Platform Developer | 18.05 This feature was introduced in 18.05.

Maestro comes with the Domain Model framework that allows you to define the conceptual structure of the information domain independently of how they are used in any visual user interface. The Domain Model is made up of entities that represent real-world concepts. You can share the Domain Model so it is available across multiple forms. A shared domain model can be used in global forms – for example, with collaboration jobs – and thus it can re-use existing elements including those created by other developers.

Having domain models provide a number of benefits for you to design and develop forms, including:

- Helps focus on what the form does and how it does it
- Allows form design and development to support an object-oriented approach
- Provides an alternative view on form design and development when data structure is used rather than form field elements
- Provides Groovy[1] developers with the ability to use domain models for data mapping
- Supports re-use by allowing multiple forms to consume the same data, for example, collaboration jobs
- Binds form fields to application data by importing JSON[2] files
- Retrieves application data as an object using an Maestro API
- Makes it easier to understand the logic behind the form by looking at its domain model structure
- Simplifies creation of complex rules by using entity mapping and relationships between form fields as objects
- Helps to use long field names, which are difficult to read otherwise, by using underscores to separate entity names from component and property names

---

[1]Groovy is a powerful scripting language which runs on the Java Virtual Machine. Across theTemenos Journey Manager 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]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 you create a new form in Maestro, a single domain model is created and populated with entities. New entities are added as you work on the form. A domain model file is added as a form resource, for example, `model-schema-json`, which is stored in the Resources folder of a form version. You can check the file in the Maestro Dashboard.

Maestro also allows you to create domain models manually. Once the model is established and configured, you can share it across multiple forms. Domain models can also be imported from other sources and shared using a library. You can manage domain models in the Model pane.

The domain model is based on a JSON schema, which is stored in a separate JSON file, so each domain model has its own JSON file. Maestro uses a limited denormalized JSON schema subset to create a domain model, so you will not be able to use all the features available from JSON schema; in particular, the ability to create a type `$ref`, for example, an applicant address type.

NOTE

Prior to Maestro 18.05, models were stored as part of a Maestro form.

NOTE

If a Maestro 17.10 form, with entities configured, is imported into a Maestro 18.05 environment, all entities will be extracted and automatically moved to a domain model file.

Next, learn how to create a domain model.

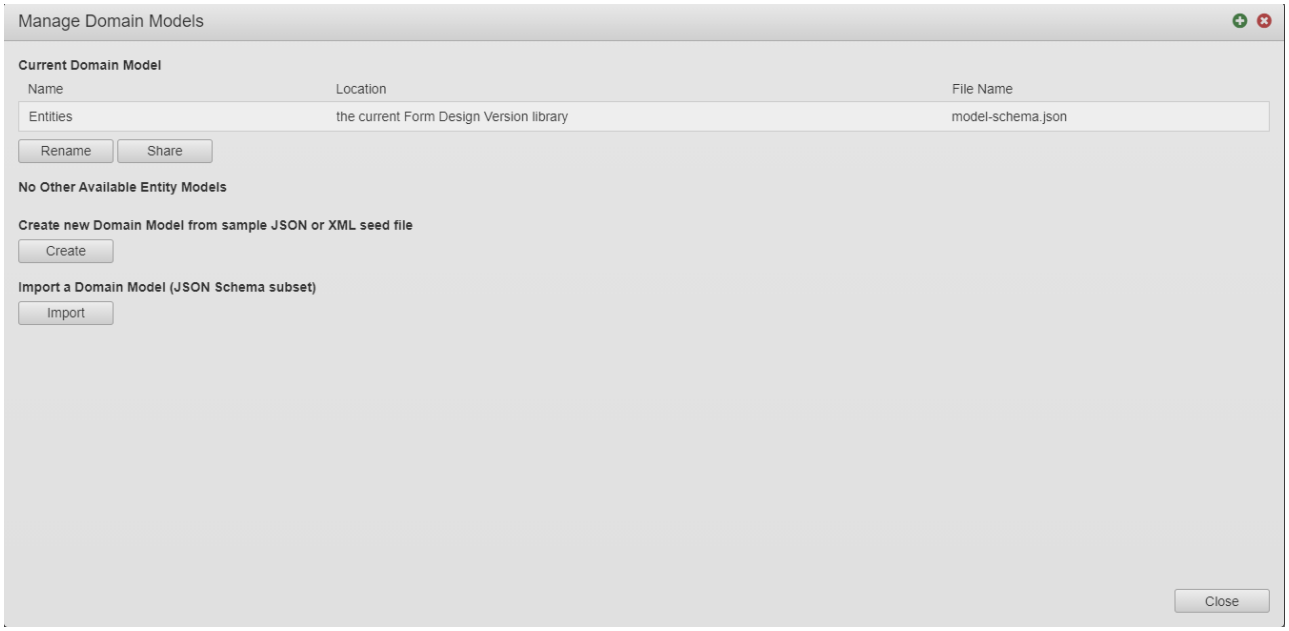
Create a Domain Model

MaestroThe UI design product. | Form Builder Platform Developer | 18.05 This feature was introduced in 18.05.

Maestro creates a domain model automatically for every new form, but it also allows you to create a new domain model based on a JSON[1] or an XML[2] seed files manually.

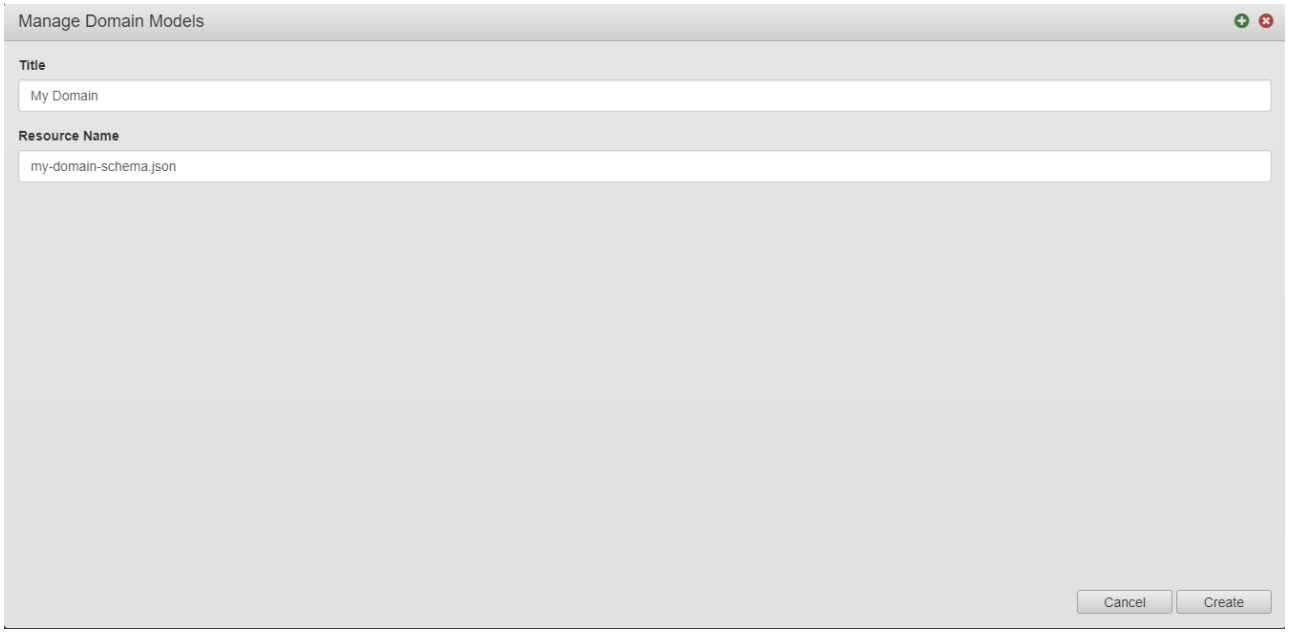
To create a domain model:

1. Open a form in the Model pane and select Manage Models.

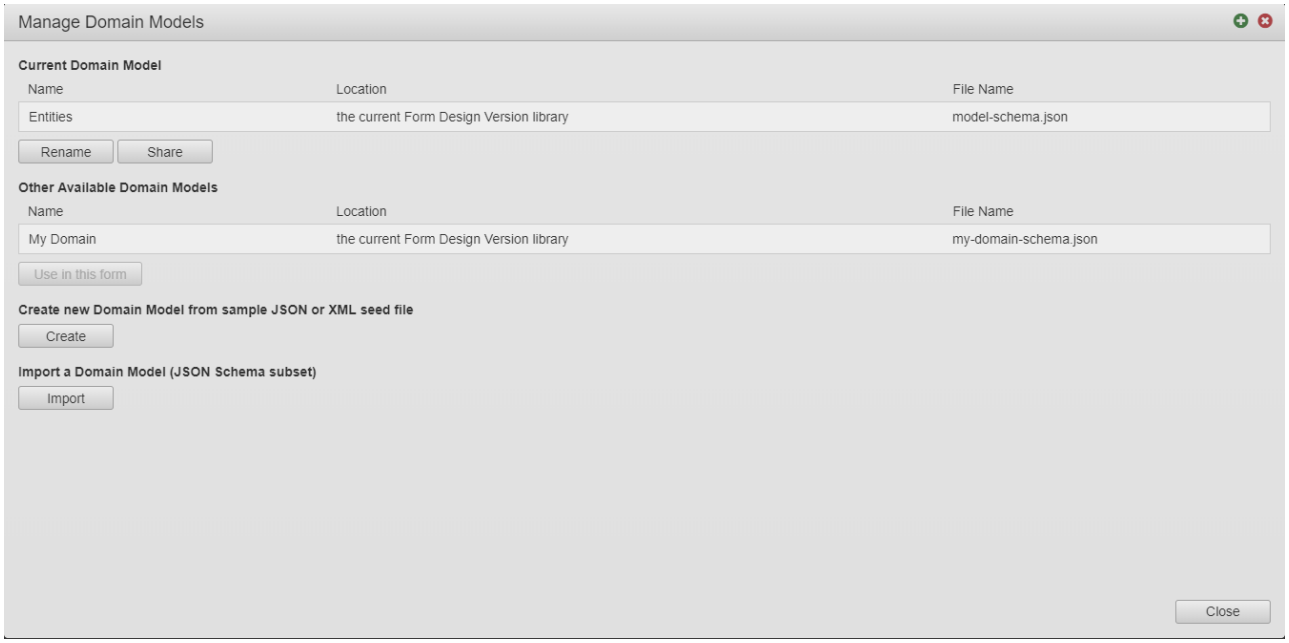


2. Click Create.

[1]JSON (JavaScript Object Notation) is a syntax for storing and exchanging data and is used in Maestro to store forms. If a Form Builder wants to transfer their form to another system, they can do so by using the Download JSON file button.
[2]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. Enter a new name in the Title field.
4. Update a new unique file name including a file extension, such as JSON or XML, in the Resource Name field. Maestro populates the Resource Name field automatically based on the title by adding `-schema.json`.
5. Click Create.
6. Choose a JSON or XML file to import from your PC. Maestro converts all imported domain models to the JSON format.
7. Review the new domain model.



8. Select other domain models from the Other Available Models list and click Use in this form to change your domain model.
9. Click Create to create another data model.
10. Click Import to import a domain model.
11. Click Close to go back to the Model pane where you can check the imported domain model. By default, the domain model remains local (unshared) to the form, which is depicted with the open lock. For more information, see how to share a domain model.

Next, learn how to import a domain model.

Import a Domain Model

MaestroThe UI design product. | Form Builder Platform Developer | 18.05 This feature was introduced in 18.05.

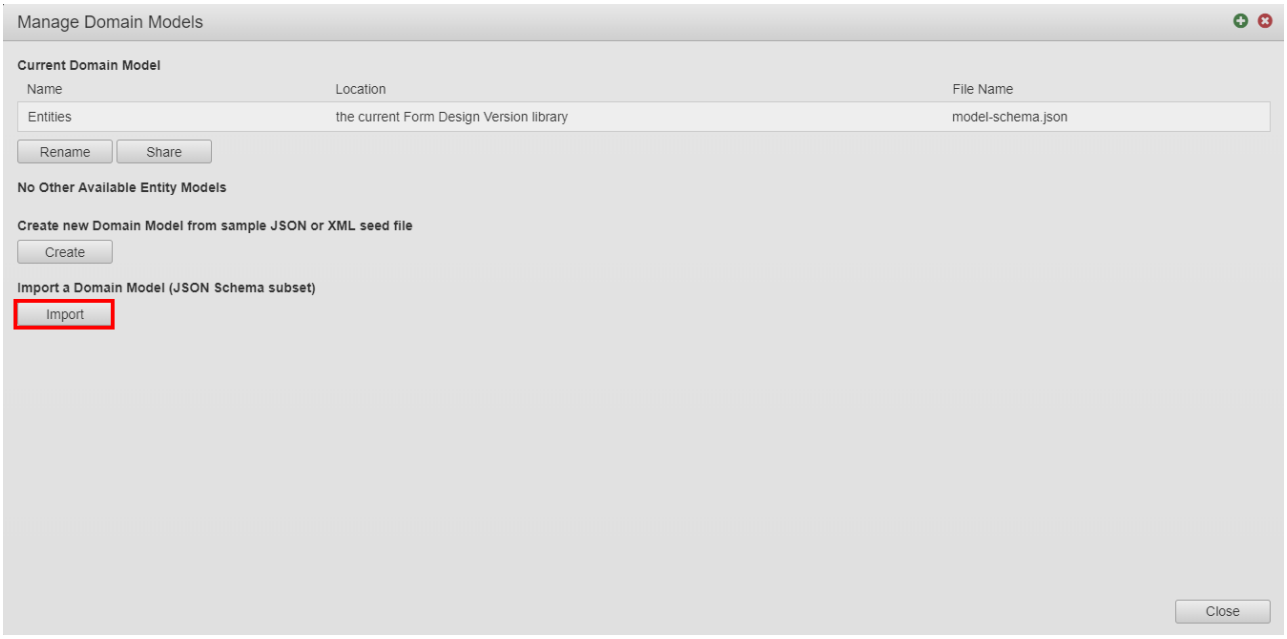
Maestro allows you to import a domain model into a form. Usually, you export a domain model from a form using the Export all Resources option. Then, you edit it, for example, by adding a new entity, and, finally, import the domain model back into Maestro.

NOTE

Maestro supports a limited subset of JSON[1] schema.

To import a domain model based on a JSON schema subset:

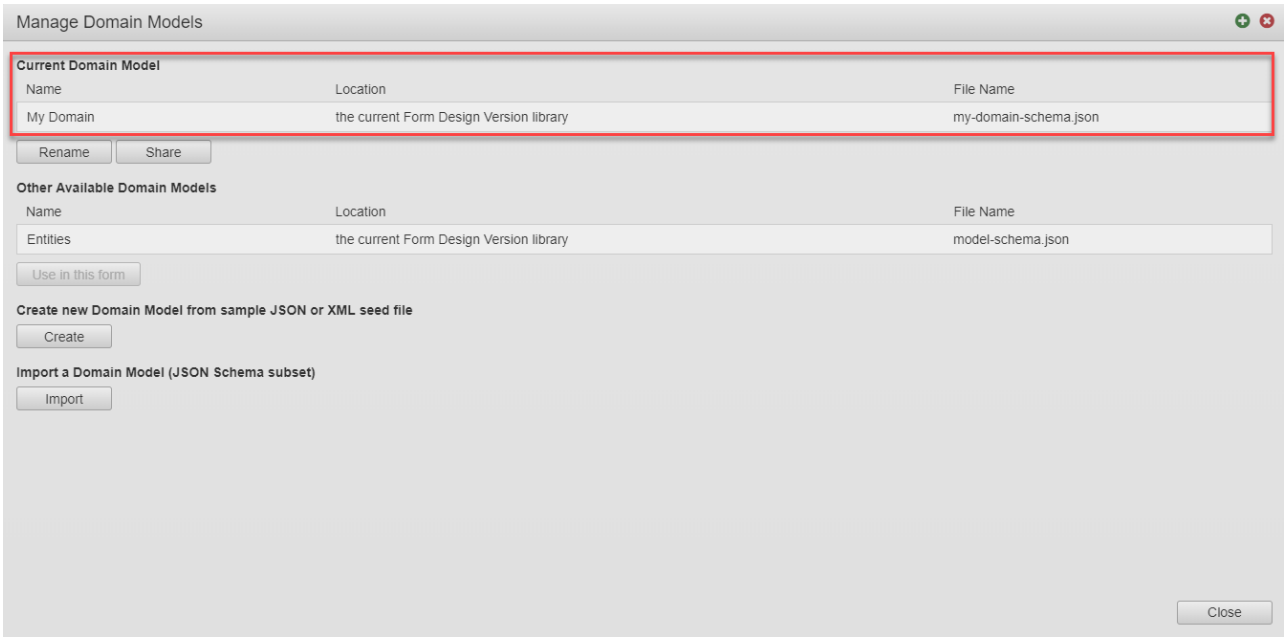
1. Open a form in the Model pane and select Manage Models.
2. Click Import.



3. Select the JSON Schema file then click Open.

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

4. Check the JSON schema file is imported as the Current Domain Model.



5. Click Close.

Next, learn how to share a domain model.

Share a Domain Model

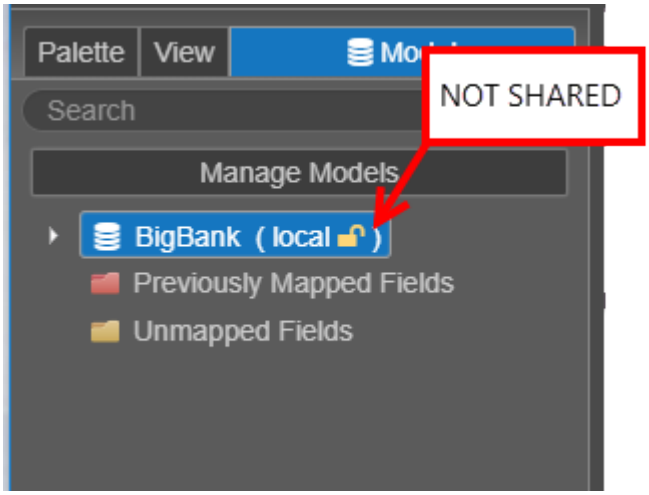
MaestroThe UI design product. | Form Builder ▪ Platform Developer | 18.05 This feature was introduced in 18.05.

Maestro allows you to share a domain model so you can use it across different forms. A shared domain model can be used in global forms, for example, with collaboration jobs, so you can reuse existing form elements including those created by other developers.

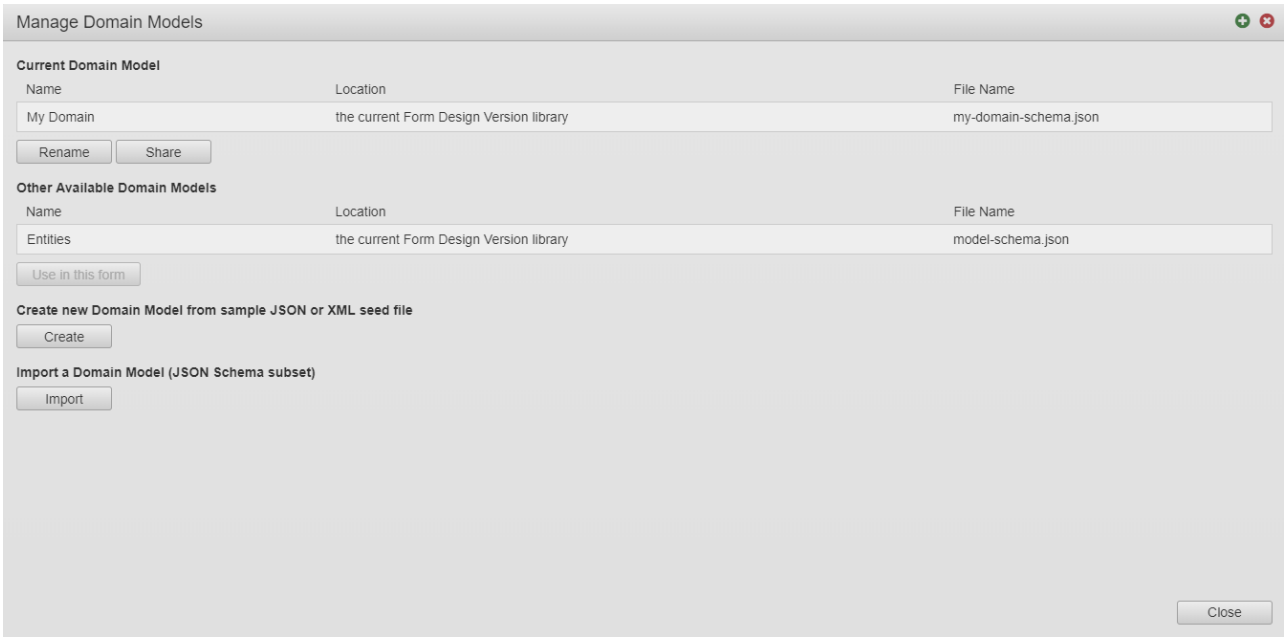
> NOTE
>
> To share a domain model, your Manager user account must be assigned a role that has the necessary permissions; for example, the **Maestro Administrator** role in its default configuration.

To share a domain model:

1. Open a form in the Model pane. If the domain model has not been shared, the unlocked padlock icon is displayed. You can share the domain model now.



2. Select Manage Models.



> NOTE
>
> If any other domain models are available to the form, they will be displayed in this dialog under Other Available Domain Models.

3. Click Share.



4. Provide a domain model name in the Title filed.

5. Provide a domain model resource file name in the Resource Name field.
6. Select a library, which you want to share the domain model to, from the Library dropdown list.

   NOTE

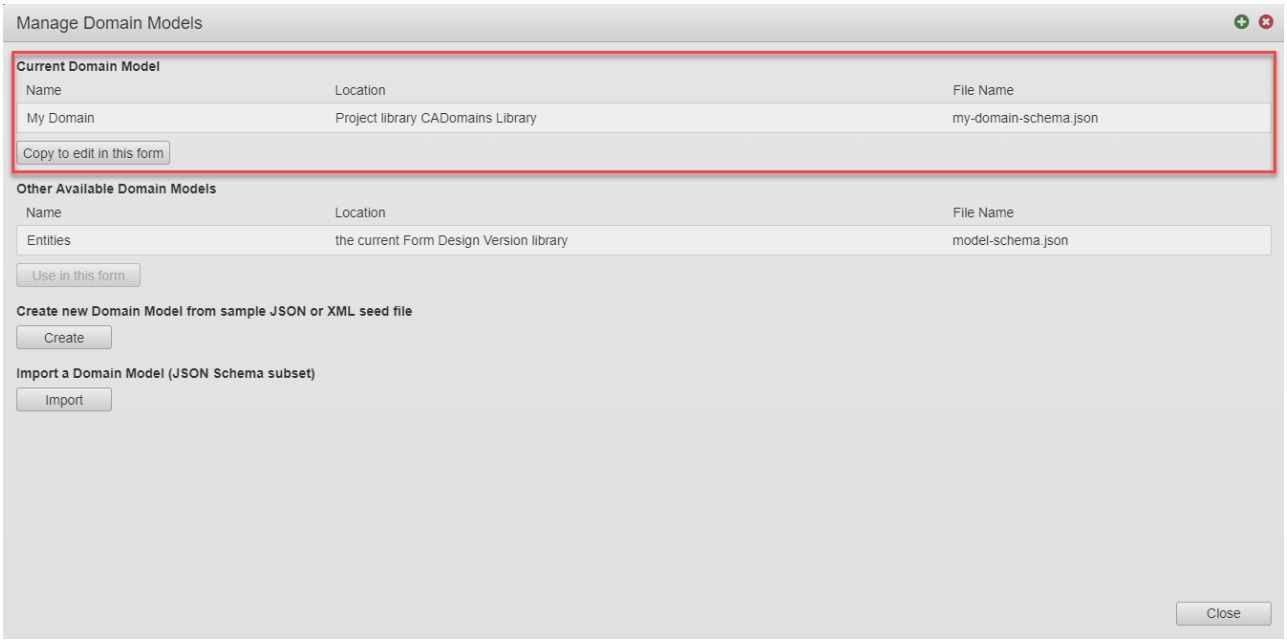
   All forms with access to that library will have access to the this shared domain model.

7. Select the Keep local copy for further editing checkbox to store a local copy of the domain model. This is the default option.

   NOTE

   If you do not want to keep a local copy of the Domain Model, clear the Keep local copy for further editing checkbox.

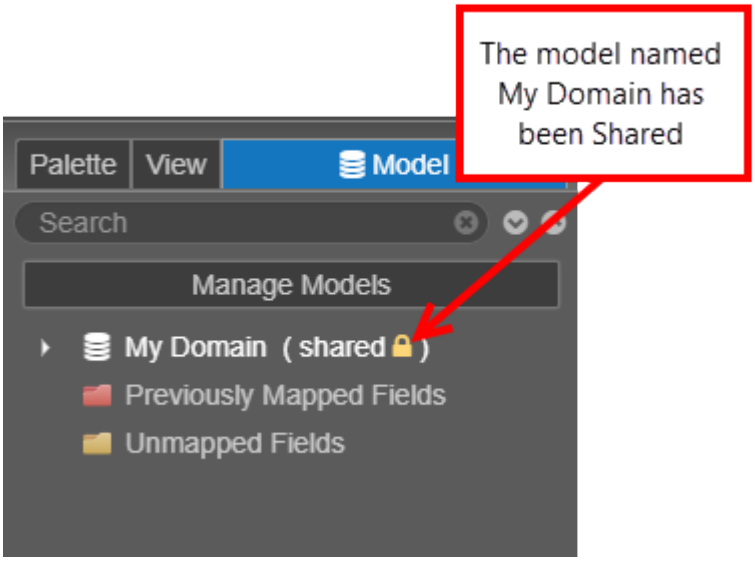
8. Click Share.
9. Check the library that the domain model has been shared to.



   It also displays any other domain models that are available to the selected form. You can select any of the available domain models and share them to a library.

10. Click Copy to edit in this form to copy the domain model and edit it in your form.

11. Click Close. Now, as the domain model has been shared, the locked padlock icon is displayed.



When a domain model has been shared, the lock icon next to the domain model's name will change from open (unlocked) to closed (locked). Any new entities created in a shared locked domain model will not be mapped or shared with other forms and libraries.

When sharing a domain model, it is important to note that the local copy of the domain model takes precedence over the shared domain model. Therefore, it is recommended that when you share a domain model and choose to keep a local copy to edit, you should rename the local copy of your domain model to avoid any confusion about which domain model you are using.

Next, learn about entities.

Build a TM Form Version

MaestroThe UI design product. | Form Builder | 22.10 This feature was updated in 22.10.

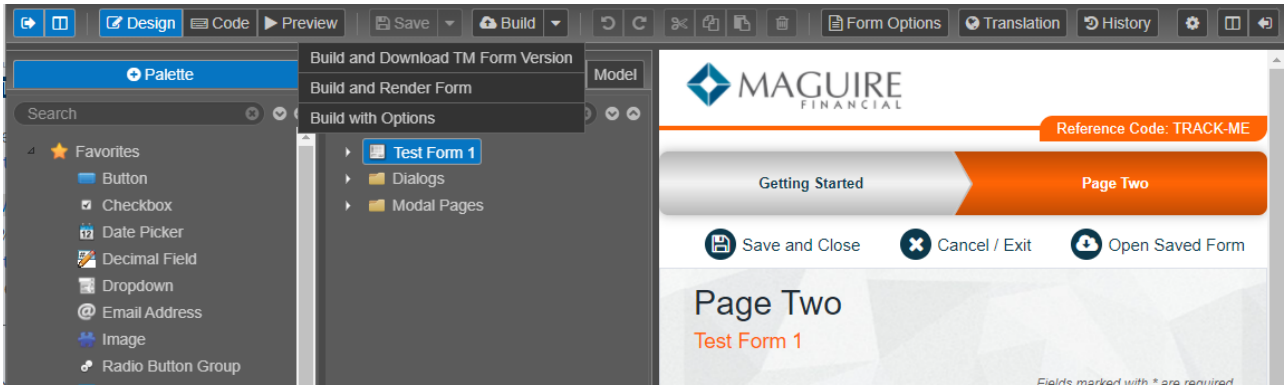
Maestro allows you to build a form as a TM Form Version ZIP file, which includes a FAR file and all related form's assets, so you can easily deploy it in a form space in Journey Manager. You develop and build the form in the same tool - the Maestro editor, which is handy and saves you time.

If the form containing multiple brands, you can build the form using one or many brands available to the form.

If you don't need to deploy a TM Form Version ZIP file right away, you can build and review it first, and import it into Manager later , when you need to. Usually, you import the TM Form Version ZIP file to Test and UAT environments, then you promote a form to a production environment by exporting and importing it using Manager's functionality.

To build the TM Form Version of a Maestro form:

1. Open a form in the Maestro editor and select one of the following options from the Build dropdown menu:



2. Click Build to build a TM Form Version. This doesn't download the TM Form Version or display the rendered form, which is useful when you are re-building and updating a TM Form Version that has already been imported into Journey Manager.
3. Click Build and Download TM Form Version to build a TM Form Version and then immediately download the ZIP file needed to import the Form Version into Manager.
4. Click Build and Render Form to build a TM Form Version and render the form in a new tab. You can download the TM Form Version later from the Management Dashboard. This option is handy when you need to view or test a rendered version of the form.

5. Click Build with Options to apply the build and brand options before building the TM Form Version.



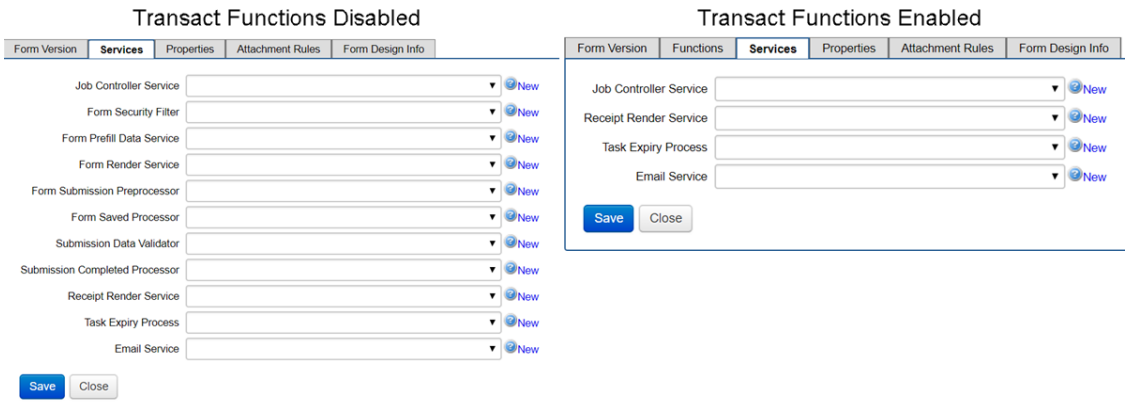
Enter a new TM Form Version name in the TM Form Name field, in case you want to change it from the default name, which is the existing name of the form.

Specify Build Options to build the form according to your requirements:

- Select the Use server side Transact Functions (requires TM 17.10 +) checkbox to use Transact functions to call services in Manager. | 17.10 This feature was introduced in 17.10.

  It's important to notice that enabling Transact functions causes the following:

  1. It limits the number of available services you can configure for your form version, as shown below:



2. It removes a number of `SystemProfile` properties, such as `<TestMode>-false</TestMode>`, from the form and from the submission XML data.

However, enabling Transact functions allows you to configure Transact functions, which were not available before that.

> NOTE
>
> You can see whether the Transact function are available in a form when you configure a form version in Manager.

> NOTE
>
> Transact functions require Journey Manager 17.10 and higher.

- Select the Minify Code checkbox to reduce the size of the form and related JavaScript[1] resources without changing its functionality so they can be sent out to a client browser more efficiently.

[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.

NOTE

Code minification[1] functionality has been enhanced in the following Maestro versions:

- Prior to 18.05 - JavaScript resources are concatenated into the `app.js` and then minified, but not transpiled[2].
- In 18.05 - JavaScript resources are concatenated into the `app.js`, then minified and transpiled. The transpiling support to the rules, the runtime framework and any widget scripts so that users could use ES6[3] constructs in their rules while maintaining IE 11 support as it is still a supported browser. External JavaScript files are the responsibility of the form developer to maintain for minification and IE 11 compatibility.
- Since 19.05 - JavaScript resources are added as they are as separate resources and provided to the form as-is. External JavaScript files that are loaded with just a URL are not processes at all other than to add the script tag that adds them to the form.
- Since 22.10 - a reference to Internet Explorer 11 was removed from this option.

NOTE

JavaScript resources are the object dropped on to a form and a user has selected a JavaScript resource that is in the form or other available library. External JavaScript files are added through the JavaScript object and setting the URL property.

---

[1]Minification is the process of removing all unnecessary characters from the source code of interpreted programming languages or markup languages without changing its functionality.
[2]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.
[3]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.

NOTE

Reducing the code makes it difficult to read it, so you have to consider deselecting the Minify Code checkbox when debugging the form.

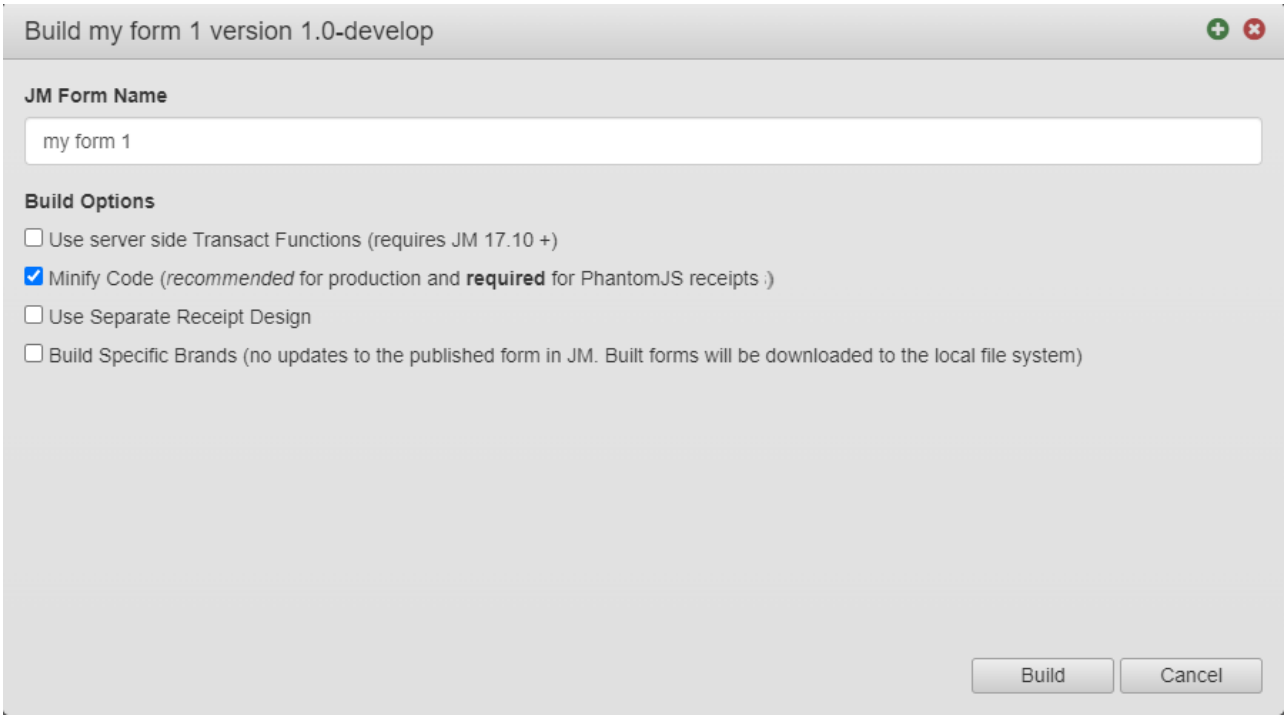
- Select the Use Separate Receipt Design checkbox to use a separate receipt, if a receipt design is available in the form, so all built receipt assets are pulled into the FAR file. This FAR is different as it effectively has two complete Maestro designs - a form and a receipt. There is an XML file in the FAR that tells Manager that the form uses a specific HTML file for the interactive form and a different HTML for the receipt.

  NOTE

  When Maestro builds the form, it doesn't rebuild the receipt form. It simply takes the latest version of the receipt that has been built and checks whether the last update time on the receipt is after the last built time on the receipt or not. If the receipt has been changed since its last built, Maestro shows a warning that the receipt needs to be republished.

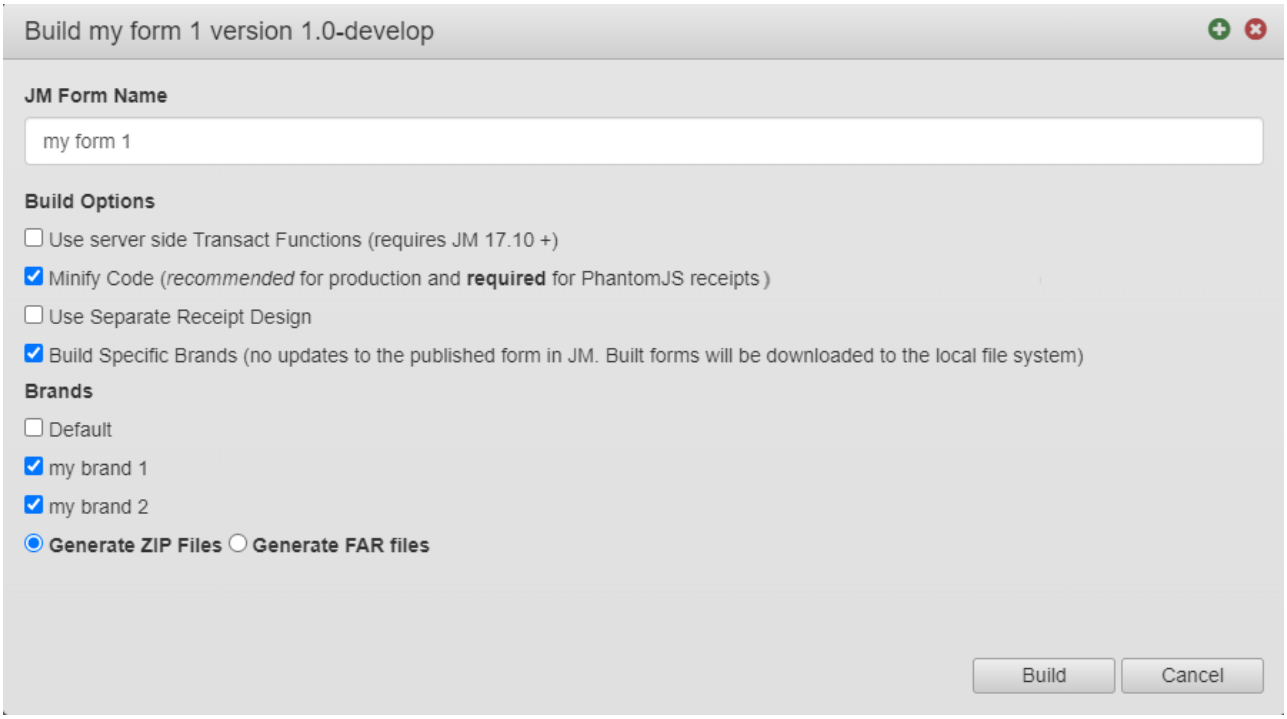
- Select the Remove Automation Framework checkbox to remove the MAF code, which reduces the download size of the form and prevent misuse of the API in production forms. The Maestro Automation Framework (MAF) provides an API which can be used by other tools to automate and investigate the state of a Maestro form. Once a form is ready for production, MAF is not required.

If the form containing multiple brands, Maestro gives you additional options to choose which brand, or all of them, to use to generate a TM Form Version.



Select the Build Specific Brands (no updates to the published form in JM. Built forms will be downloaded to the local file system) checkbox to see the the list of brands available to the form.

> NOTE
>
> You can't select the Build Specific Brands (no updates to the published form in JM. Built forms will be downloaded to the local file system) checkbox if the form has only one brand.

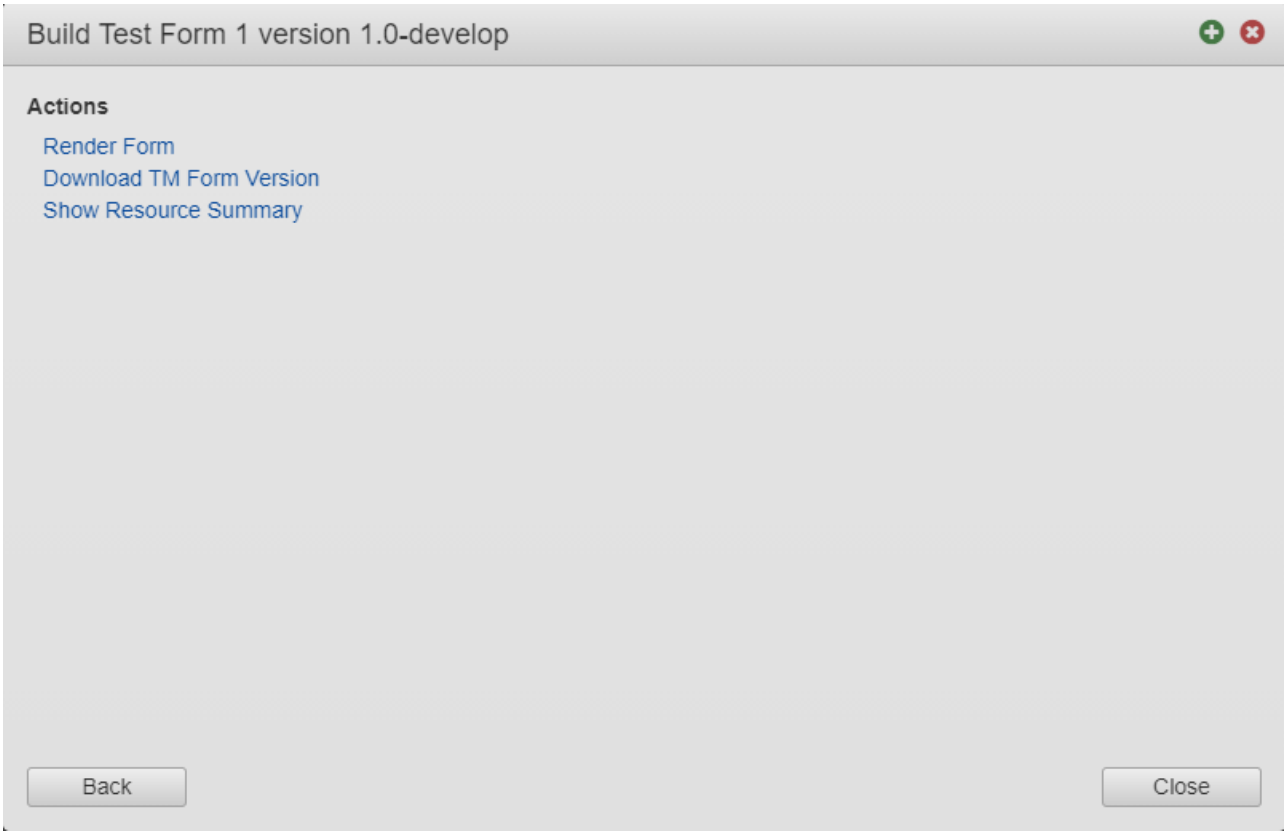


Specify Brands to build the form using specific brand or brands:

- Select one or many brands from the available list of brands defined in your template. The Default brand is always available.
- Select Generate ZIP Files to bundle the form and all relevant resources in a ZIP file, which you can deploy in various Manager environments.
- Select Generate FAR files to bundle the form and all relevant resources in a FAR[1] file, which you can deploy in various Manager environments.

Click Build to build the form. Check the build result and perform some actions as described below.

[1]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.



- Click Render Form to render the form in a new tab. It's similar to the Preview Mode.
- Click Download TM Form Version to download the ZIP file on your local PC.
- Click Show Resource Summary to see resources associated with this form and included in this ZIP file.
- Click Back to go to the previous screen.

Once a TM Form Version is built, you can download it at any time from the Maestro Management Dashboard using the Download JM Form Version option.

When building the TM Form Version, you may encounter some common errors, which you can resolve yourself.

Next, learn how to update TM Form Version.

TM Form Version Build Errors

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

When you build a form using the TM Form Versions functionality, an error can occur. Below are some common errors which you can easily resolve yourself. In other cases, you may need to contact your system administrator or our support team.

A Form with This Name Already Exists

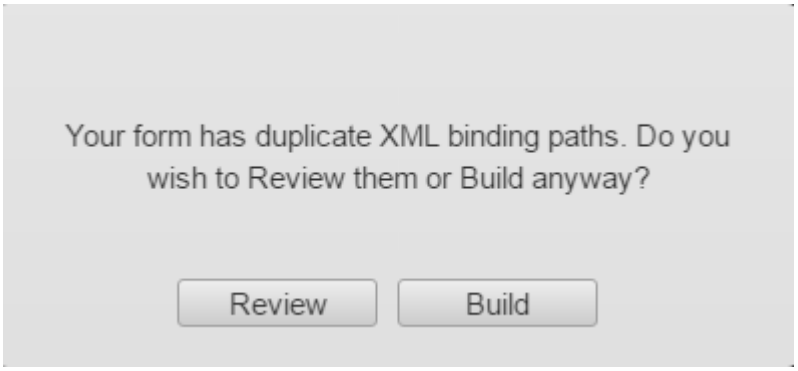
This error tells you that a form with this name already exists in your organization in a Maestro instance. This can happen if you want to move a form (by exporting and then importing it) from one project folder to another and you want to keep the form name the same.



To resolve this error, you should use the Delete JM Form Versions option to delete the TM Form Versions from your Maestro instance.

Duplicate Binding

A Duplicate Binding error occurs when a Form Builder tries to build a TM Form Version with duplicate submission data bindings on the form. Maestro detects that two or more components share the same data binding location and shows the message below.



This is a warning to a developer that users may experience unintended behavior during data entry. Click Build to confirm that you are aware of the duplicate binding and want to proceed. Otherwise, click Review to stop the build and review a list of the duplicate bound components. You can resolve the duplicates and try to build the form again.

For Composer Users

There are significant differences in the publishing and building process between Maestro and Composer.

As a standalone tool, Composer requires the form designer to choose the Manager server to host the form. When published the designer, or an administrator, is then required to choose the Organization and form update strategy (New Version or Update Existing).

By design, Maestro is a Space associated with a Journey Manager server, so it can only build to the same server. Also, by design, Maestro requires Forms to be arranged in organizations that align with those configured on the server. Finally, the Form Builder controls the versioning of forms in the Maestro Management Dashboard.

Next, learn how to update TM Form Version.

Update a TM Form Version

MaestroThe UI design product. | Form Builder | All versions This feature is related to all versions.

You may want to update a Maestro form after it has been deployed to Journey Manager or migrate it to a new Journey Manager environment. This is to accommodate customer feedback or follow continuous improvement processes within your company.

To update a TM Form Version:

1. Open a form in the Maestro editor and edit it as required.
2. Build a TM Form Version.
3. Log in to the Manager server where you originally uploaded the TM Form Version ZIP file.
4. Select Forms > Forms and find the form that you want to update, which is the form that you just updated in Maestro.
5. Open the form, select a form version and click Update Design from Maestro to automatically update it form the Maestro server.

NOTE FOR ON-PREMISE AND AMAZON VIRTUAL PRIVATE CLOUD (VPC) MANAGER INSTALLATIONS

If the VPC is not managed by Temenos, in order for the updating process to function correctly, the Manager environment needs to send an outbound request over HTTPS to a Maestro host (for example, `maestro.avoka.com.au`). For this reason, it is recommended that before you attempt any update of a form version using Maestro, you first check your firewall rules and ensure that they allow this outbound request.

Next, learn about the SCM tool.

Maestro SCM Overview

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 17.10 This feature was updated in 17.10.

Journey Maestro comes with the Source Code Management (SCM) command line interface[1] (CLI) that enables you to integrate your Maestro projects with external source code management systems, such as Git or SVN. It provides support for collaborative Journey form development using a flexible approach towards version control allowing you to use various source control strategies, such as feature branches, to work on new features or bug fixes simultaneously using the same form code.

Getting Started

To start using Maestro SCM, you need a few things set up first:

1. Maestro SCM requires Java 8 or later installed on your computer. If you don't have Java installed, you need to download a JDK runtime for your operating system, which are available for Linux, Mac and Windows.
2. Obtain a user account to log in to the Maestro server that you will be integrating with.

   NOTE

   The user account must be assigned with the Maestro **SCM Project** role.

3. Download the latest SCM version from our community web site.

   NOTE

   You must be logged into the community web site to be able to download installers.

4. Unpack the archive into a directory on your computer.

Using Maestro SCM

Maestro SCM is similar to Git CLI, so you will get the hang of it fairly quickly.

To start using the SCM client:

---

[1] A command line interface (CLI) is a text-based user interface (UI) used to view and manage computer files.

Windows

Run the `scm.bat` file on your local computer:

```
Microsoft Windows [Version 10.0.17134.407]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\jsmith>scm.bat

<not-connected>/ C:\users\jsmith\
$
```

Linux or Mac

Open a terminal and type the following command:

```
$ java -jar scm.jar

<not-connected>/ \home\
$
```

To list all SCM commands and get help, type `help` at the prompt.

```
$ help
Maestro Source Code Management (SCM) shell, version 19.5.0
Type help 'name' to find out more about a named command:

cd                      Change the current working directory.
ls                      List the files in the current or specified directory.
cat                     Displays the contents of a text file.
start                   Start the specified program or OS shell command.
del                     Delete the specified file or empty directory.
cls                     Clear the screen.
rm                      Delete the specified file or empty directory.
dir                     List the files in the current or specified directory.
pwd                     Print working directory path.
type                    Displays the contents of a text file.
mkdir                   Make a new directory.
scm                     Maestro Source Code Management (SCM) integration commands.
scm connect             Connect to remote Maestro server.
scm design clone        Clone remote design to the local directory.
scm design diff         Compare differences between local design files and remote design.
scm design pull         Pull remote design file differences to local directory.
scm design push         Push local design file differences to remote design.
scm design rm           Remove remote design file.
scm design status       Compare local design files with remote design files.
scm library clone       Clone remote library files to the local directory.
scm project add         Add a new project to Maestro server using local project files.
scm project clone       Clone remote server project to the local directory.
scm project diff        Compare differences between local project and remote project.
scm project delete      Delete remote server project.
scm project list        List remote server projects.
scm project pull        Pull remote project file differences to local project.
scm project push        Push local file differences to remote project.
scm project rm          Remove remote project file.
scm project status      Compare local project files with remote project files.
```

For more information on how to use Maestro SCM, see SCM Quick Start for your Maestro version.

You can migrate your existing Maestro projects to SCM projects to align form development with your existing practices.

Next, learn how to use Maestro SCM to implement a Git Feature Branch workflow.

Best Practices for Maestro SCM Feature Branch Workflow

MaestroThe UI design product. | Form Builder " Platform Developer | 18.11 This feature was introduced in 18.11.

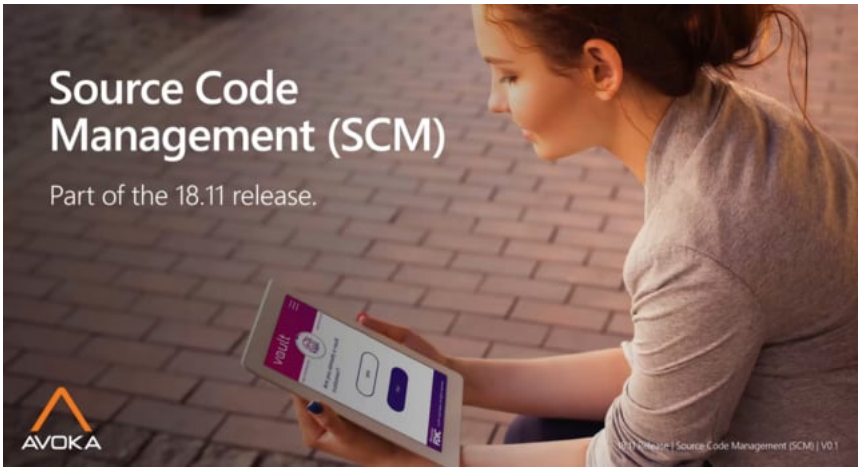
Introduction

Maestro Source Code Management (SCM) provides support for collaborative Journey form development using a flexible approach towards version control. This article provides an example of how to use Maestro SCM to implement a Git Feature Branch workflow.

The Feature Branch workflow scenario starts by creating an SCM project in Maestro and cloning it to your local computer. The Maestro SCM project source code is used to initialize a Git repository which becomes the source of truth for all subsequent development. While form development still occurs in Maestro, all version control activities, such as branching and merging, are performed in Git. In order to follow the Feature Branch workflow, Maestro projects are mapped to Git master and feature branches.

The following diagrams illustrate two perspectives on the relationship between Maestro SCM and Git in a Maestro SCM Feature Branch workflow. The first diagram shows how changes flow between Maestro projects and Git branches using Maestro SCM and Git commands. The second diagram shows how changes move between a project on a Maestro server and a remote Git repository on a Git server via the user's local Git repository using the SCM shell (also known as the SCM client) and a Git client.

These diagrams serve as a useful point of reference for the process described in this article. They are also used in this Maestro SCM v18.11 video to support an introductory explanation of how Maestro SCM is used in a feature branch workflow.



Prerequisites

This article assumes that:

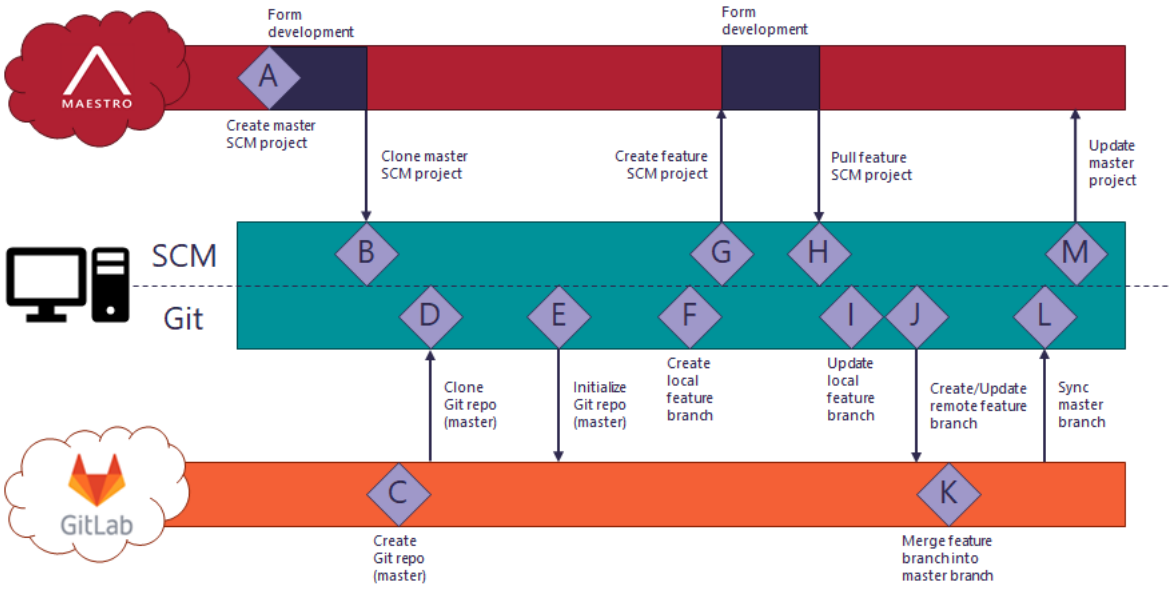
- The Maestro SCM shell is installed on your computer and you know how to start it.
- You know how to create a Maestro form.
- You have access to a Journey Maestro environment.
- You Maestro user account has the SCM Project permission.
- You have a working knowledge of Git.
- You have a Git client installed on your computer.
- You have a working knowledge of (and access to) a Git server, such as GitLab.
- You are using a Windows computer. (The `scm` and `git` commands are operating system agnostic, but some Windows-specific commands and tools are used in this example.)

Maestro SCM Feature Branch Workflow

The Feature Branch workflow is implemented across three environments:

- Maestro.
- GitLab (or another Git server).
- Your local computer.

The following diagram shows that the majority of commands are performed on your local computer using the SCM shell and your preferred Git client, while form development occurs in Maestro, and key Git actions happen on your Git server.



Follow these steps to implement a Maestro SCM Feature Branch workflow. The letter diamonds in the previous diagram are referenced throughout these steps using the format X. Sample `scm` and `git` commands are provided that you can adjust to follow along with the example.

Part 1: Getting Started

1. Create a Maestro SCM project. A

   Login to Maestro and select your organization. Click Create Project to display the Create Project modal. Enter the required details and select SCM as the Project Type.

   > NOTE
   >
   > SCM projects appear in the Navigator pane with a different icon than standard Maestro projects. This new icon shows at a glance that the project supports Maestro SCM.

2. Create a form in your Maestro SCM project.

3. Open a command prompt, start the Maestro SCM shell, and change to the directory that you want to clone your Maestro project into.

   ```
   scm.bat
   cd \c\projects\scm-example
   ```

4. Clone your Maestro SCM project's source code. B

The Maestro SCM command to clone your project's source code can be found on your Maestro project's Project Details page, in the SCM Clone field. The command is of the form `scm project clone <org>/<proj>` where `<org>` is your project's organization and `<proj>` is your project's name.

Enter the SCM Clone command.

```
scm project clone know/scm-example
```

You can verify that your project's source code is on your computer and up-to-date using the `dir` and `scm project status` commands.

```
dir
scm project status
```

5. Create a blank repository on your Git server. C
6. In your Git client, execute the commands to clone the new repository and initialize it with the Maestro project files. These commands can be obtained from GitLab after creating the repository. Use the **Existing folder** commands. D E

```
git init
git remote add origin <path-to-remote-repo>
git add .
git commit -m "Initial commit"
git push -u origin master
```

Part 2: Feature Branches

At this point, the Maestro master project and Git master branch have been created and initialized. We are now ready to create the feature branch and project.

7. In your Git client, create a local feature branch. F

```
git checkout -b "scm-feature"
```

8. In your SCM shell, create a Maestro feature project using the `scm project add` command. G

```
scm project add 'Add a new feature'
```

9. Refresh Maestro to confirm that the feature project is now available. The Project Details page should show that it's of the type SCM Private. Only project owners and administrators have access to projects of this type.

   NOTE

   In the Navigator pane, feature projects are distinguished from the master SCM project by a differently colored icon.

Part 3: Feature Development

At this point, the Maestro feature project and Git feature branch have been created and are ready for development of the new feature. While the feature project is available in Maestro, the Git feature branch only exists on the local computer.

10. In Maestro, open the form in the feature project, make necessary changes to implement the new feature, and save your form.

11. Return to your SCM shell and check the feature project's status:

    ```
    scm project status
    ```

    You'll find differences between your local project and the remote project on your Maestro server.

    Update your local project with the changes from the Maestro server: H

    ```
    scm project pull
    ```

    When you pull the changes, you need to confirm which changed files to accept. You can type `A` to accept all file changes, `N` to reject all file changes, or `Y` to interactively accept or reject each file change.

12. Return to your Git client and check the local repository's status. Git will report that there are modified files that need to be staged for commit. Stage and commit these files, except for `scm.properties` which we'll deal with later. I

```
git status
git add *.jsor
git commit -m "Implemented the new feature"
```

Part 4: Merging Features

At this point, work on the new feature has been completed in Maestro, and the Maestro feature project and local Git feature branch have been updated. Next, we need to merge the feature branch into the master branch and update the master project from the master branch.

13. In your Git client, push the local feature branch to the remote repository, thereby creating a remote feature branch. J

    ```
    git push origin scm-feature
    ```

14. In GitLab, merge the feature branch into the master branch. K

    A good (and sometimes mandatory) practice is to merge the feature branch into the master branch via a merge request. When creating the merge request, you can choose to have GitLab remove the feature branch for you once it is merged. If you don't select this option, you can delete the feature branch manually after it has been merged.

15. In your Git client, checkout the master branch and synchronize your local master branch with the remote master branch. L

    ```
    git checkout master
    git pull
    ```

16. Checking the local Git repository's status reveals that the modified `scm.properties` file remains in the local working directory. As this file contains the project name, we want to restore the version of this file from the original SCM project created in Maestro before we update the master Maestro project. To do this, check the `scm.properties` file out of the master branch.

    ```
    git status
    git checkout scm.properties
    ```

17. With the local working directory updated, we can now update the master Maestro project.

M

In your SCM shell, push the local master Maestro project up to your Maestro server:

```
scm project push
```

Then delete the feature project:

```
scm project delete know/scm-example
```

Part 5: Project Cleanup

At this point, the master Maestro project has been updated with the work carried out in the feature project. You can confirm this in Maestro, then finish up by removing any left-over artifacts of the development process.

18. Refresh Maestro to confirm that the feature project has been deleted.
19. Open your master Maestro project and confirm that the changes made in the feature Maestro project are now applied to the master Maestro project.
20. In your Git client, delete the local feature branch and the local tracking branch for the deleted remote feature branch.

```
git branch -d scm-feature
git fetch --prune
```

The Feature Branching workflow used in this article offers a simple approach to collaborative development, wrapping Maestro SCM commands and processes around a Git feature branching strategy. If, however, you prefer to use a different branching strategy, you can still use the Maestro SCM version control commands to manage version control of your Maestro project assets.

Getting Help

If you need help with Maestro SCM command syntax, type `help` at the prompt to see the following list of available commands.

```
$ help
Maestro Source Code Management (SCM) shell, version 19.5.0Type help 'name' to find out more
about a named command:
cd                  Change the current working directory.
ls                  List the files in the current or specified directory.
cat                 Displays the contents of a text file.
start               Start the specified program or OS shell command.
del                 Delete the specified file or empty directory.
```

Next, check the Maestro SCM Quick Start for your Journey SDK release.

Maestro Offline Build Tool Overview

MaestroThe UI design product. | Form Builder Platform Developer | 21.11 This feature was updated in 21.11.

Journey Maestro comes with the Offline Build Tool, which is a command line interface (CLI) for building Maestro forms. This tool is a faster and more flexible alternative to building forms using the Maestro web user interface and it enables you to:

- Build one or all forms in a project with a single command.
- Create a script to automate the form building process.
- Integrate the form building process with your Continuous Integration / Continuous Deployment (CI/CD) pipeline.
- Adapt Maestro development to your own workflow.

The Offline Build Tool relies on Maestro Source Code Management (SCM) to clone a project and related libraries to your local file system before building forms. This means that non-SCM projects can't be built using this tool. However, this may change in a future Maestro release.

> NOTE
>
> Maestro forms prior to version 19.05 may not build correctly with the Offline Build Tool.

Next, learn more about the Offline Build Tool, which is available for your Maestro version.

Migrate to a SCM Project

MaestroThe UI design product. | Form Builder ▪ Platform Developer | 17.10 This feature was updated in 17.10.

Journey Maestro allows you to create SCM projects so you can use your favorite version control tools to branch project's source code. If you have a MaestroStandard project, once you are ready use Continuous Integration (CI)[1] practices on Maestro code base, which consists of templates, forms, components and libraries, you can convert it to the Maestro SCM Project type.

NOTE

It also works in the opposite direction - if you have a Maestro SCM project, you can migrate it to the Standard project type as the project type doesn't affect the content of forms.

Migration Challenges

The main problem here is that the SCM maestro project doesn't support multiple versions of any asset type in it. There has to be only a single version number of each of forms, components, libraries, and templates in the project. Its assumed that the GIT branch it was pulled from identifies what version of the asset this is (or will be deployed as).

Due to the above we are hitting a big problem in the migration to SCM projects. Generally, there would always be multiple versions of any of the assets in the legacy maestro projects. That's just the way we used to do it. Since the SCM project can have only one version, we need to pick the initial one which will make it to GIT as the first one at the migration time. Picking the right version number of any of the assets is not as of an issue as making sure we pick and migrate only that one while removing (or skipping) all the other ones during the migration process.

Just imagine we are migrating a project with 10 forms where each of them has 30+ versions (same for templates, components, libraries). When creating the SCM project only the most updated version needs to be migrated.

---

[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.

The restriction of a single version only takes us to the next challenge: a new folder structure of the SCM projects. The folder structure of a legacy project downloaded via maestro UI or SCM is different to the folder structure of the scm project type. The version folders were removed decreasing the depth of the folder structure by 1.

Migration Strategies

There are several ways to migrate an existing Maestro standard project to a SCM one. Let's look at them in details.

Recreate the original project

As the heading suggest this approach recommends to create an empty SCM project where all necessary assets from the original one will be manually copied to. The coping is done via Maestro dashboard export and import of each asset one by one.
Pros

- Great opportunity to clean up the project by skipping any litter
- A good opportunity to review the project structure, dependencies between assets, such as forms, components, libraries, and refactor them if possible

Cons

- Too lengthy - can time too much time
- Error prone - easy to miss or copy wrong assets

Converting a legacy project to the SCM project type

If you chose to convert an existing legacy project to the SCM project type as it is a few steps must be executed manually. The steps are listed below:

1. Pick asset's current versions:
    - remove or skip any outdated asset versions and pick only the current one you wish to push into the GIT master branch.
    - this would likely be the latest or highest version of your asset.
2. Delete remaining asset versions. It must be done on forms, components, templates, libraries.
3. Migrate asset's content. Copy all asset current version's content into the asset's root folder. For example, if you picked the version 1.0.0 as the current one: `forms/cc-application/1.0.0/* to forms/cc-application/*`.
4. Update asset's lib-manifest.json. remove the delete version numbers from the `designVersions` node.

5. Update the asset's `form-manifest.json`.
6. Validate. Test library associations. Make sure the SCM project has the same libraries associated as the legacy (original) project. Note that the project version libraries will be on the version 1.0.0 though.

Next, learn how to use Maestro SCM to implement a Git Feature Branch workflow.