1. Composer User Guide (Composer v4.3) . . . 2
1.1 Composer User Guide: Table of Contents (Composer v4.3) . . . 3
1.2 Getting Started in Composer (Composer v4.3) . . . 4
1.3 The Composer Workspace (Composer v4.3) . . . 8
1.4 Composer Form Revisions (Composer v4.3) . . . 11
1.5 Form Designer (Composer v4.3) . . . 19
1.5.1 Assistants (Composer v4.3) . . . 25
1.5.2 Predefined Blocks (Composer v4.3) . . . 32
1.5.3 Tables and Repeats (Composer v4.3) . . . 39
1.5.4 Templates (Composer v4.3) . . . 44
1.6 Edit Properties (Composer v4.3) . . . 50
1.7 Preview and Publishing (Composer v4.3) . . . 74
1.8 Navigation (Composer v4.3) . . . 76
1.9 Layout (Composer v4.3) . . . 88
1.10 Submissions and Attachments (Composer v4.3) . . . 96
1.11 Dynamic Data (Composer v4.3) . . . 100
1.12 TransactField Mobile App (Composer v4.3) . . . 109
1.13 Multi-channel (Composer v4.3) . . . 114
1.14 Form Sharing (Composer v4.3) . . . 117
1.15 Composer Form Parts (Composer v4.3) . . . 123
1.16 Usability and Accessibility (Composer v4.3) . . . 132
1.17 Privacy (Composer v4.3) . . . 133
1.18 Validation (Composer v4.3) . . . 134
1.19 Analytics (Composer v4.3) . . . 143
1.20 Field Types (Composer v4.3) . . . 148
1.21 Repeats (Composer v4.3) . . . 155
1.22 Advanced Scripting (Composer v4.3) . . . 156
1.23 Optimizing Performance (Composer v4.3) . . . 167
1.24 Business Rules (Composer v4.3) . . . 173
1.25 Library Advanced Features (Composer v4.3) . . . 177
1.25.1 Stylesheets (Composer v4.3) . . . 191
1.26 Form Designer Advanced Features (Composer v4.3) . . . 194
1.27 Access & Security (Composer v4.3) . . . 197

Avoka Transact

Composer User Guide (Composer v4.3)

Aim of this Guide

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

Notation

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

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

Composer User Guide: Table of Contents (Composer v4.3)

Getting Started in Composer (Composer v4.3)

Describes how to login into Composer and the various user roles and the hierarchy of documents (or "forms") into Organizations and Projects. More information on Access Control is contained in the Composer Administration Guide.

Login Page

The composer login page presents the user with a standard user id/password dialogue and several links to assist the user in getting started. Each of the links is described below.

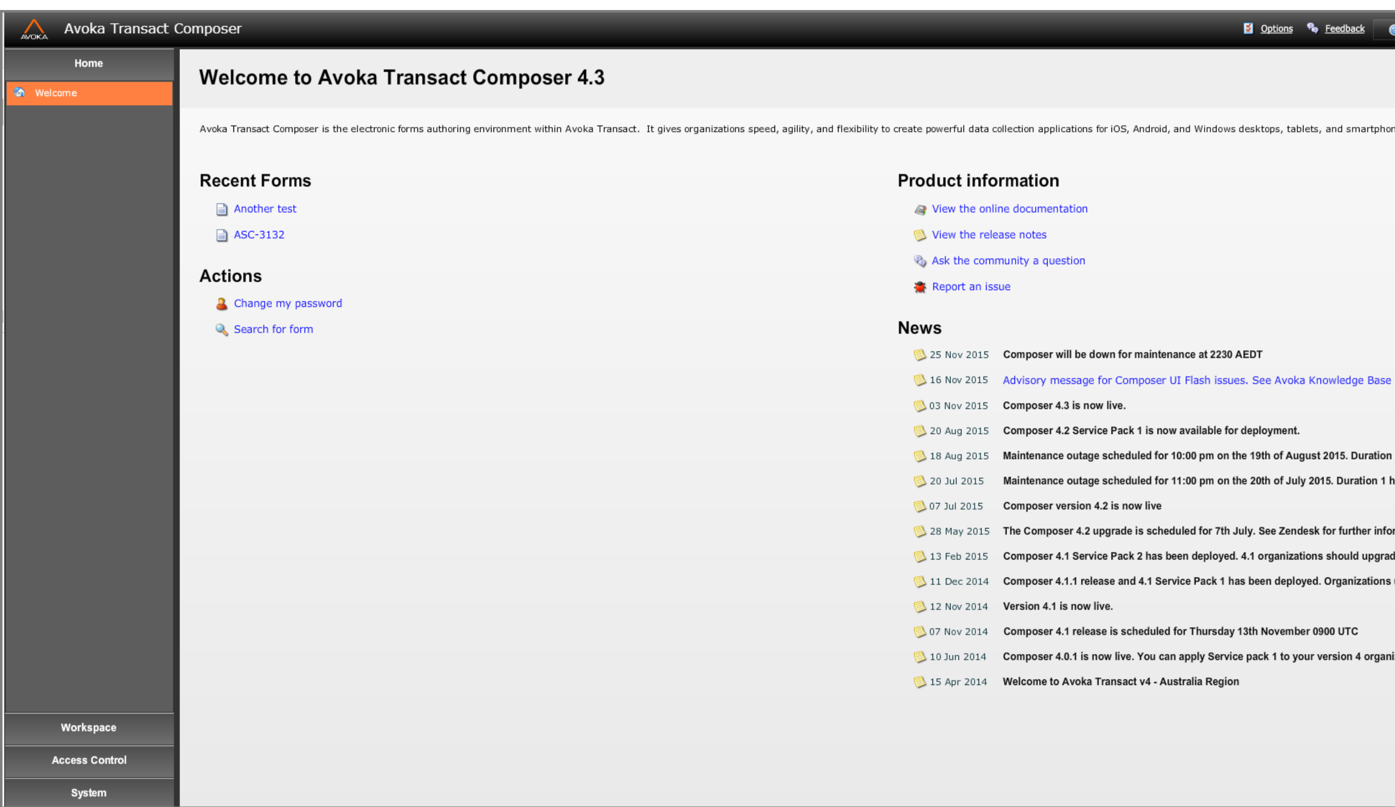
Request password reset function.
By selecting this function, the email address specified in the User name field will be sent an email.

The email body will contain a link with a token

When selected a change password dialog will be shown where the user can reset their password.

The Welcome Page

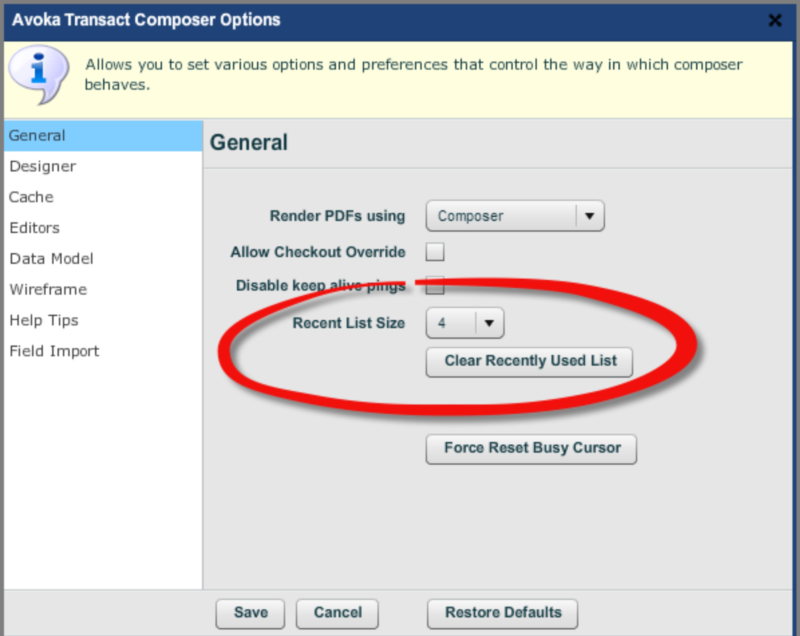
The main area of the Welcome page is divided into sections described below. These are intended to make it easy for users to navigate around the product.



Note that the screenshot is that of an administrator and may contain options not seen by other users.

Recent Forms

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



Actions

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

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

Account	Organization	Project	Form Name	Description	Modified Time
Avoka RnD	Development 50	Selenium Tests	Calc Test		Sun Nov 22 05:41:35 GM
Avoka RnD	Development 50	Unit Tests	Calc Test		Sun Nov 22 05:41:35 GM
Avoka RnD	Development 42	Unit Tests	Calc Test		Sun Dec 6 05:44:19 GM
Avoka RnD	Development 42	Selenium Tests	Calc Test		Sun Dec 6 05:44:18 GM
Avoka RnD	Development 40	Selenium Tests	Calc Test		Sun Dec 6 05:44:26 GM
Avoka RnD	Development 40	Unit Tests	Calc Test		Sun Dec 6 05:44:26 GM
Avoka RnD	Development 36	Unit Tests	Calc Test		Sun Dec 6 05:44:33 GM
Avoka RnD	Development 36	Selenium Tests	Calc Test		Sun Dec 6 05:44:33 GM
Avoka RnD	Development 41	Unit Tests	Calc Test		Sun Dec 6 05:44:38 GM
Avoka RnD	Development 41	Selenium Tests	Calc Test		Sun Dec 6 05:44:38 GM
Avoka RnD	Development 33	Selenium Tests	Calc Test		Sun Dec 6 05:44:08 GM
Avoka RnD	Development 33	Unit Tests	Calc Test		Sun Dec 6 05:44:08 GM
Avoka RnD	Development 43	Selenium Tests	Calc Test		Sun Dec 6 05:44:47 GM
Avoka RnD	Development 43	Unit Tests	Calc Test		Sun Dec 6 05:44:47 GM
Avoka RnD	Development 34	Unit Tests	Calc Test		Sun Dec 6 05:44:52 GM
Avoka RnD	Development 34	Selenium Tests	Calc Test		Sun Dec 6 05:44:52 GM

Product Information

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

News

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

Home Menu Choices

Workspace

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

Access Control

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

System

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

Top of Page Links

Options

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

Category	Function	Default	Description
General			**These are General Configuration Options**
	Render PDFs using	Composer	Options: Composer, Adobe LiveCycle, XDP
	Allow Checkout Override	Off	Setting that will ignore the form checkout status on Open/Preview only. Save/Publish will honor the checkout state.
	Disable keep alive pings	Off	By default Composer pings the server to avoid the session being closed.
	Recent List Size	4	The number of Recently Accessed Forms retained on the Welcome Screen
	Clear Recently Used List	-	Clears the List
	Force Reset Busy Cursor	-	Occasionally the Cursor can remain as busy. This will reset the cursor back to its original state.
Designer			**Options to Configure the designer screens**
	Show Internal Fields	Off	By default internal fields are hidden on form open. This will override that setting.
	Show Names on Structure Tree	Off	By default the field label is shown in the tree. This setting will switch to showing the Internal Name of the Field.
	Enable Parts on Standard Projects	Off	This will enable the Form Parts functionality in standard projects rather than requiring a Parallel Development Project (version 4.2 or later).
	Expand Transparent Parts	Off	This makes the contents of imported transparent parts visible in the structure tree (version 4.2 or later).
Cache			**Allows the manipulation of the local Cache size. This can assist with performance of the UI**
	Organization Cache Size	3	How many Organizations can be cached
	Current Cache Contents	-	Show the Organizations that are in the Cache
	Clear Cache	-	Clears the current organization cache
	Show Load Times	Off	This setting will display a pop up with the form load time in milliseconds.
Editors			**Configuration settings when the editor dialogs are opened.**
	Show Advanced Properties	Off	By selecting this option the advanced properties of the field are shown
	Default Units	Form Units	Allows selection of default units in the editor
	Hide Script Editor Tree at startup	Off	Forces the script editor tree to NOT be built at startup.
Data Model			
	Link Data Model Trees	On	Links tree selection in data model view.
Wireframe			**Controls how the wireframe looks and works.**
	Default Zoom Factor	110%	Changes the default wireframe zoom. This can be adjusted when the form is open.
	Show Measurement Tooltips	Off	Shows approximate measurements of a field dimensions as a tooltip in the wireframe
	Freeze Wireframe	Off	Will set off wireframe redraws
	Use embedded fonts	On	Controls whether the wireframe will use its embedded font
	Auto Refresh Problems/Dependencies	On	When selected the wireframe will auto refresh Problems and dependencies automatically
	Check Errors Before Preview	Off	
	Show Wireframe on Form Open	On	This option can restrict the wireframe from being shown at form open. This can assist with opening times on large forms.
	Show Empty Block Border	On	Setting it on will show a border around an empty block. This makes it easy to drop in items.
Help Tips			**Controls whether Help Tips are displayed**
Field Import			**Settings used with the Field Import Function**
	Show Field Import Dialog	On	
	Create New Field Instead of Updating Existing Field	On	

Feedback

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

Online Documentation

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

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

Logout

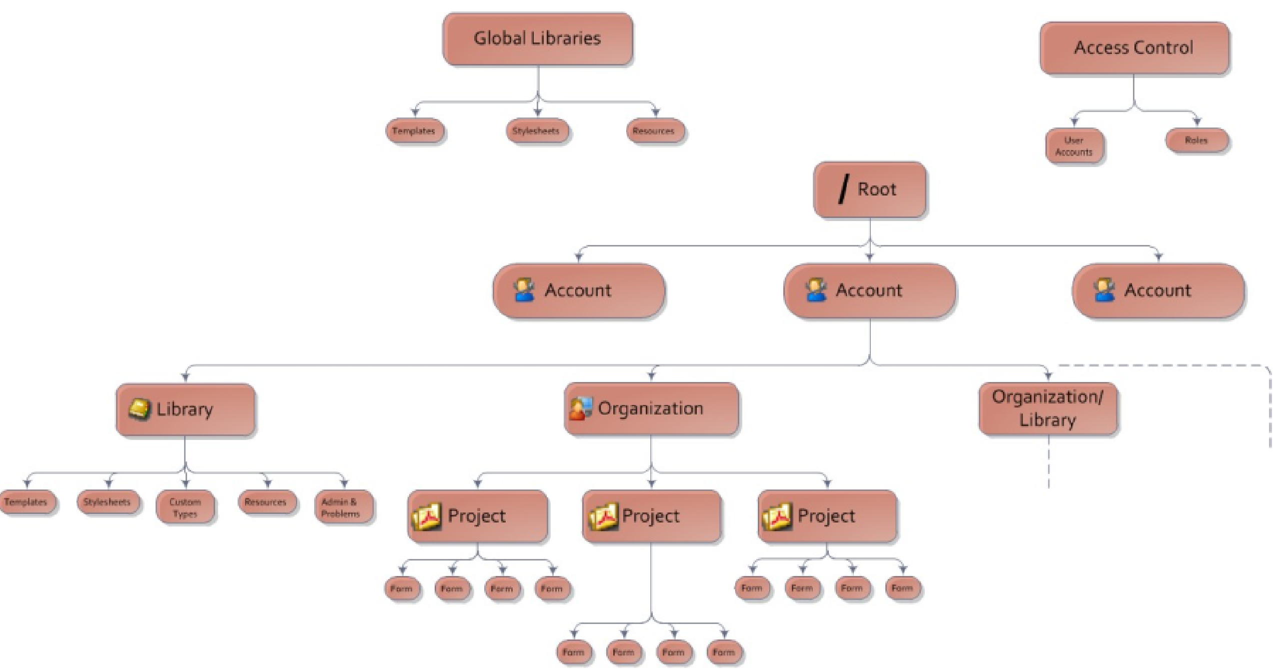
Logs the user out.

The Composer Workspace (Composer v4.3)

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

The Hierarchy and Environment

Workspace Hierarchy



The Workspace divides into "Accounts". Accounts can have several "Organizations". Each Organization has several "Projects". Each Project has its set of forms. In this model, forms are not dynamically shared between projects but can be between Organizations.
When designers require personal workspaces, they do this at the Project level.
All forms can be saved to version control and can be rolled back to a previously versioned revision, if required. If designers delete forms, they are actually not deleted from the Composer database and can be recovered.
If a form is open in Composer by one designer, other designers are locked out. There is a form lock override available however this needs to be used with caution to ensure changes are not lost.
Each Account can have a number of Libraries which are dynamically shared at the Organization and Project levels. Administrators are free to allocate whole Libraries as they see fit, or the components of Libraries.

Merits of this Structure

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

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

The Composer Hierarchy

Accounts

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

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

Organizations

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

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

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

Projects

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

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

Asset	Description

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

Creating a Form

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

Here are the steps to the New Form Wizard:

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

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

Composer Form Revisions (Composer v4.3)

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

It offers the ability to:

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

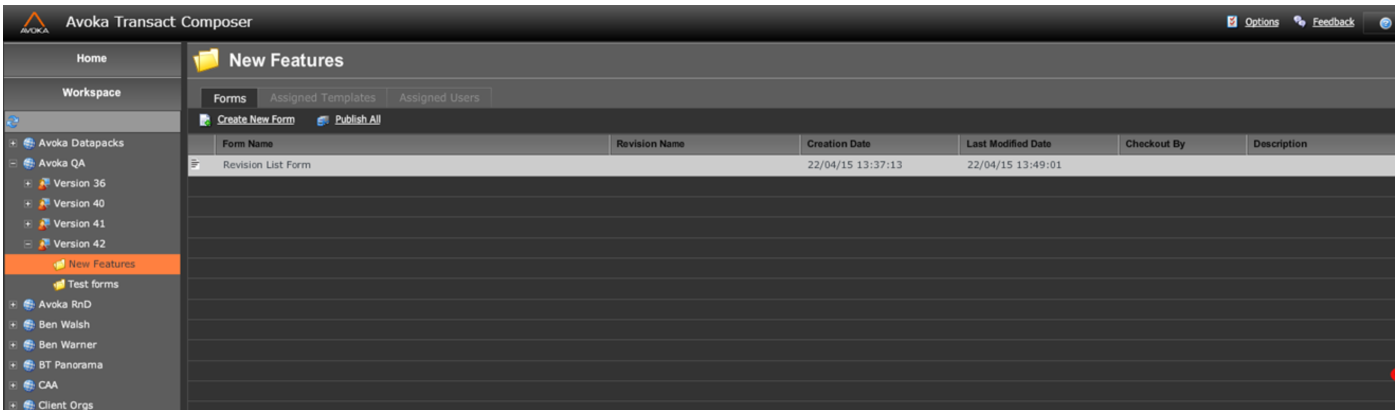
Manage Revisions Functionality

Accessing the Form Revision Dialog

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

Menu

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



Screen 1 - Form List Screen showing Form Menu items.

Manage Form Revisions Dialog

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

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

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

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

Manage Form Revisions [Revision List Form]

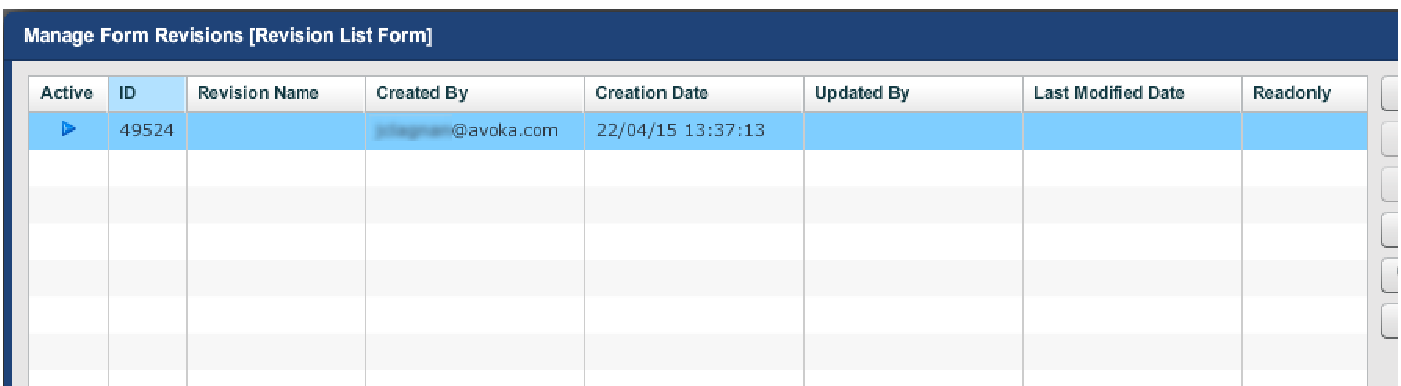
Active	ID	Revision Name	Created By	Creation Date	Updated By	Last Modified Date	Readonly
▶	49524		@avoka.com	22/04/15 13:37:13			

Close

Screen 2 - Revision Dialog

Form Revision Options

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



Screen 3 - Revision Dialog: Options highlighted

Create

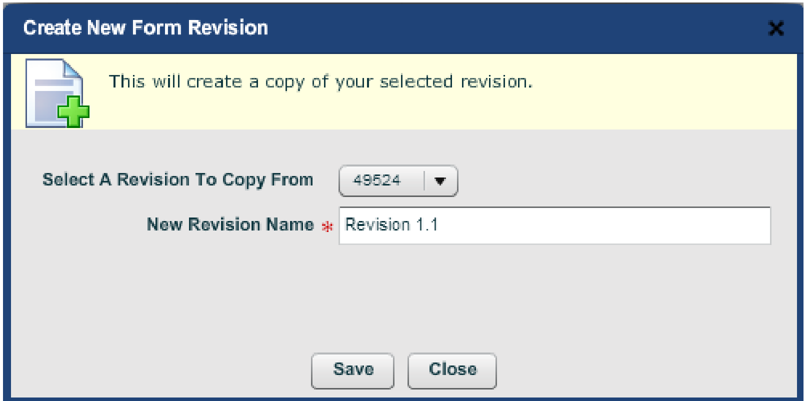
The create option allows a user to create a new form revision based on any of the form revisions in the list.

Option allows:

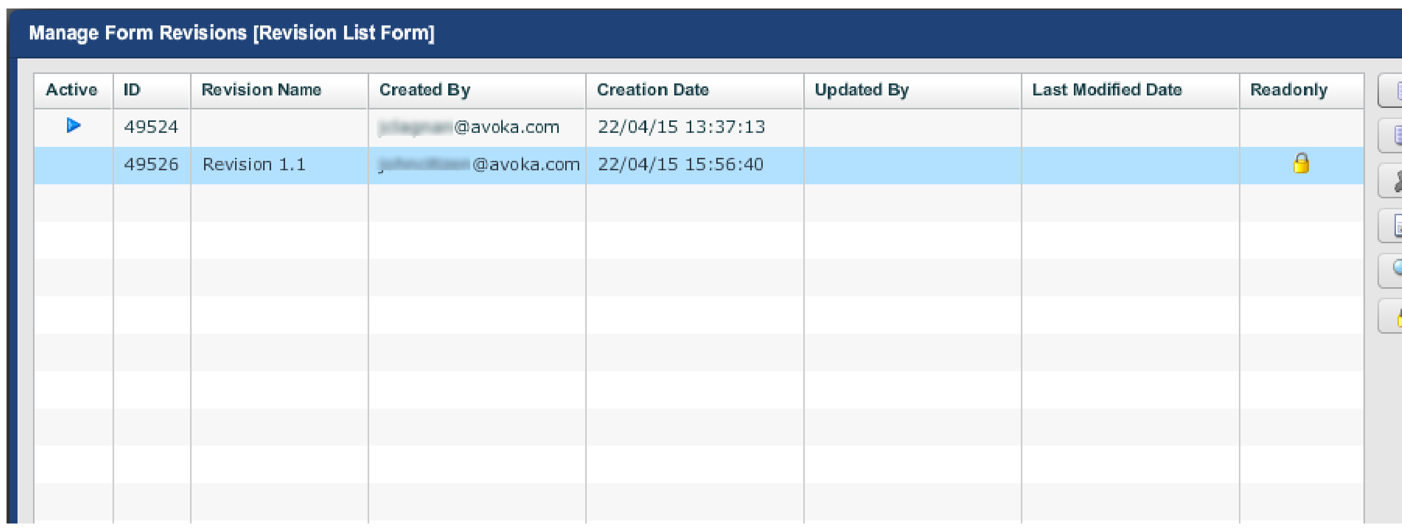
- a user to specify an existing revision ID to copy the newly created revision from. The dropdown list will show all available revision ids for the form.
- Enter a name for the revision. This is a mandatory freetext field that can be used for versioning or informational purposes.
- The newly created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

Create New Form Revision
This will create a copy of your selected revision.
Select A Revision To Copy From 49524
New Revision Name *
Save Close

Screen 4 - Create Revision Dialog



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



Manage Form Revisions [Revision List Form]

Active	ID	Revision Name	Created By	Creation Date	Updated By	Last Modified Date	Readonly
▶	49524		@avoka.com	22/04/15 13:37:13			
	49526	Revision 1.1	@avoka.com	22/04/15 15:56:40			🔒

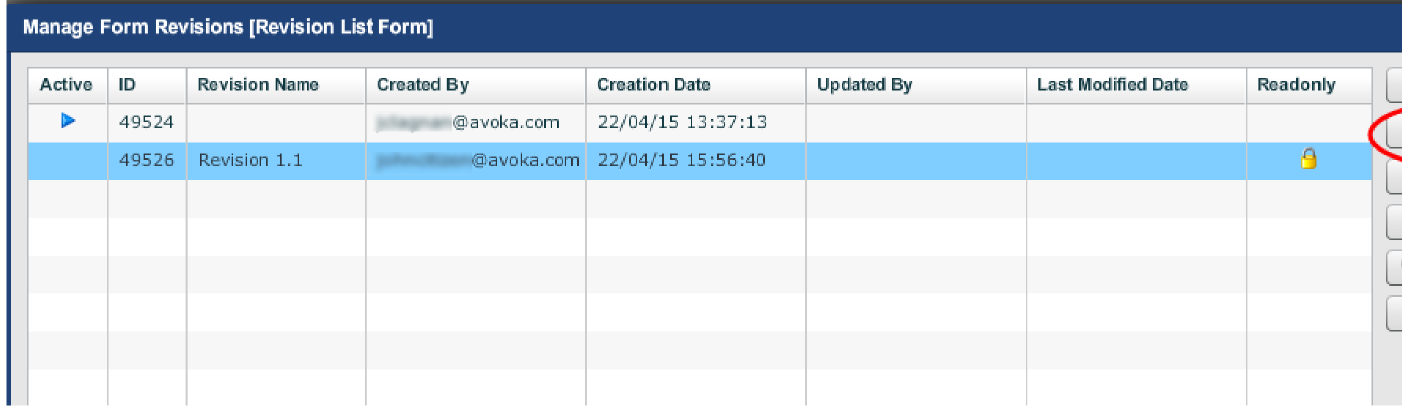
Screen 6 - Revision Dialog with newly created revision name. The new revision is read-only and NOT active.

Make Active

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

Option Summary:

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



Screen 7 - Revision Dialog showing make active option.

Manage Form Revisions [Revision List Form]

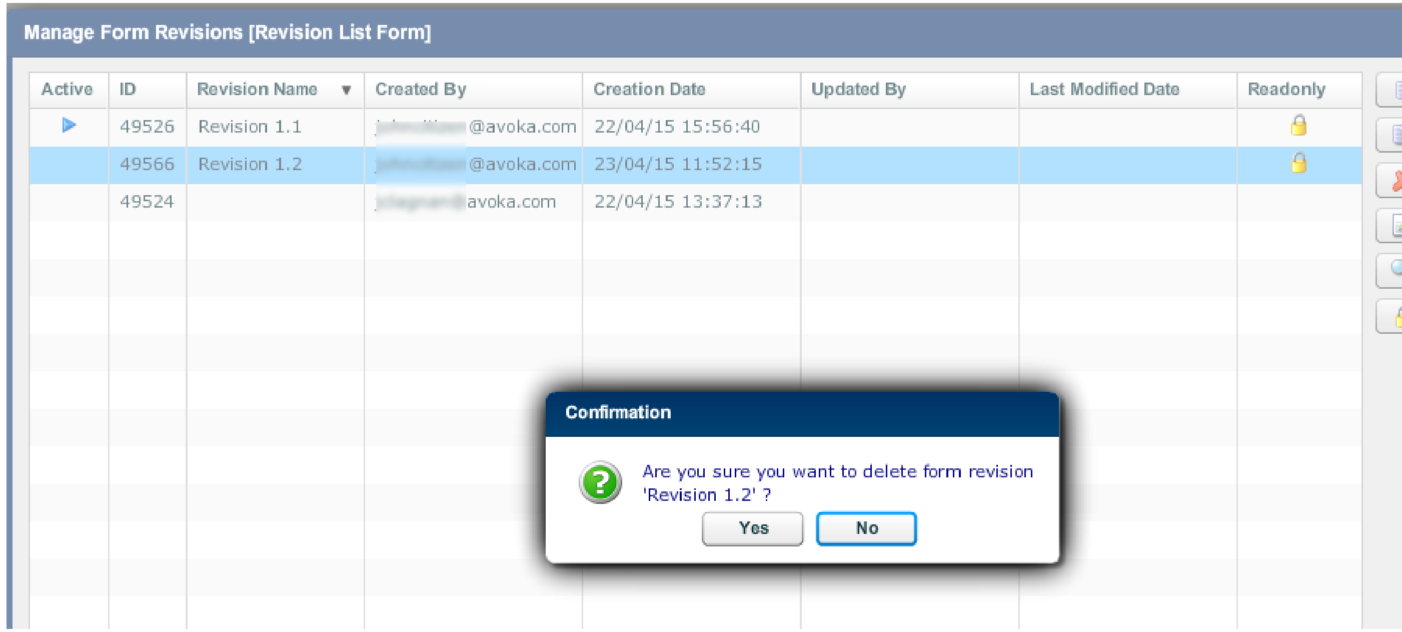
Active	ID	Revision Name	Created By	Creation Date	Updated By	Last Modified Date	Readonly
▶	49526	Revision 1.1	@avoka.com	22/04/15 15:56:40			🔒
	49524		@avoka.com	22/04/15 13:37:13			

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

Delete

This option will delete a revision from the list.

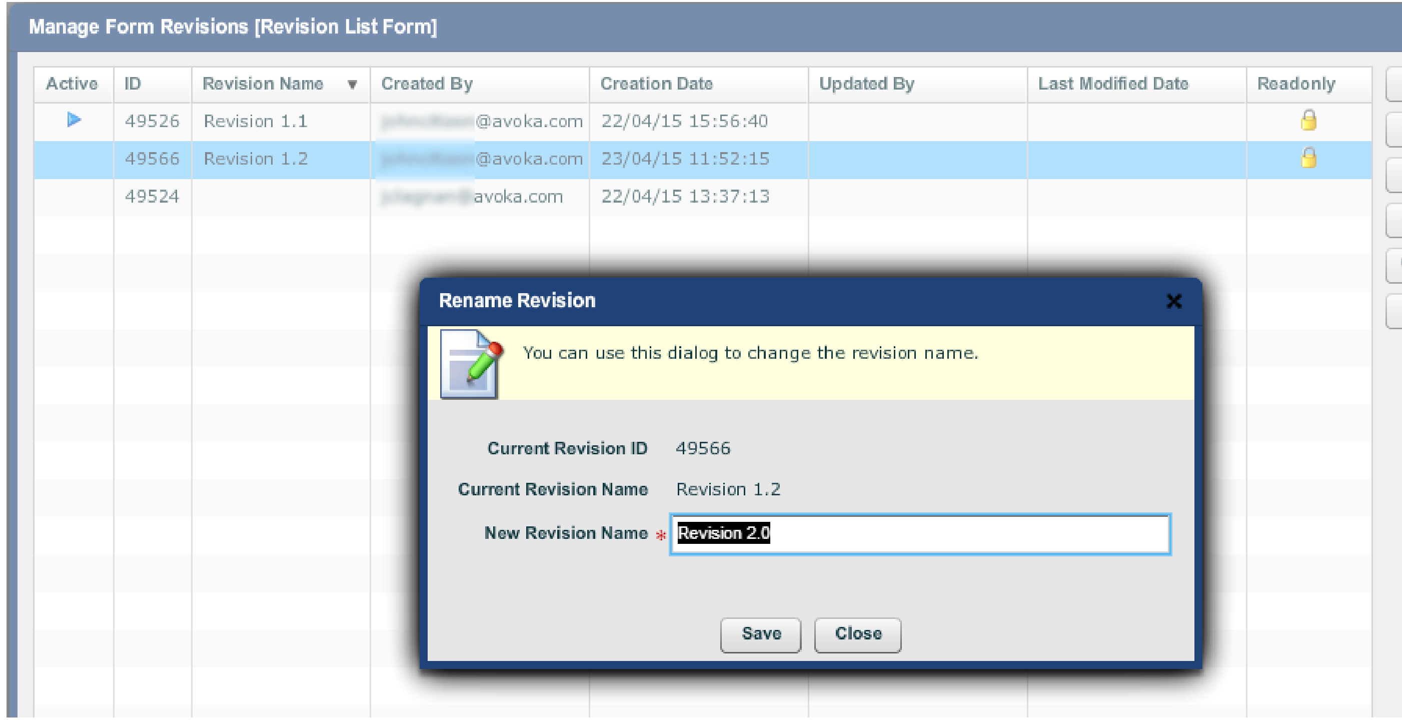
Active revisions can not be deleted. The delete button will be disabled.



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

Rename

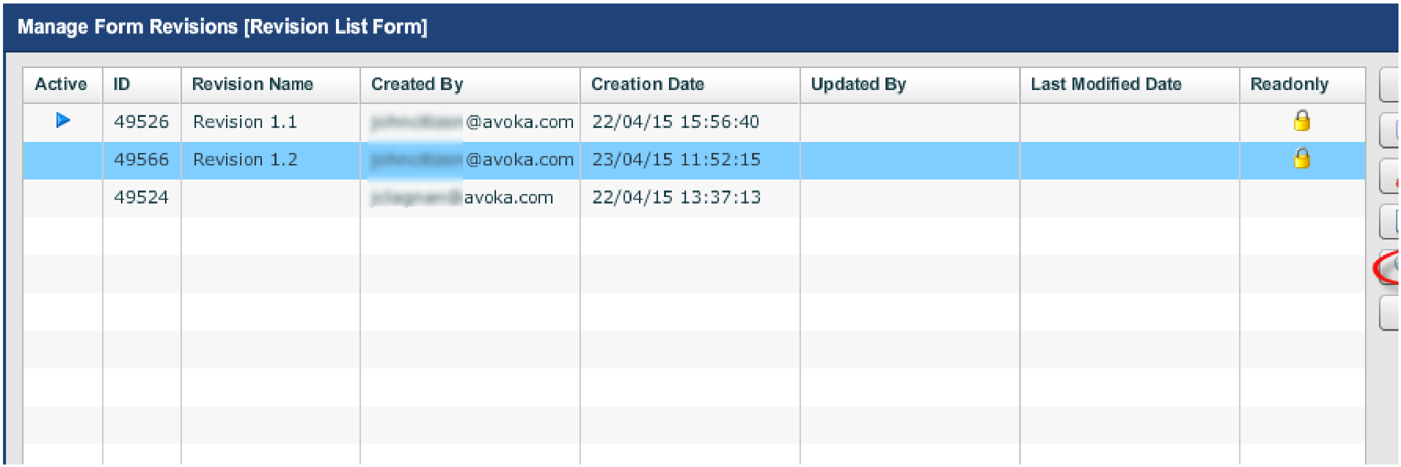
This option allows you to modify the Revision Name text. The revision name is NOT unique and is simply free text.



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

View XML

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



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

View Form Revision XML [49566-Revision 1.2]


```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<formdescriptor stylesheet="Styling-Maguire-Default,Navigation-TopGroups" template="Template-Maguire">
    <content>
        <field name="GettingStarted" type="Section-Standard-Level1">
            <setproperty name="section.heading" value="Getting Started"/>
            <field name="_outerArea" styles="${section.outer.stylename}" type="Block-SectionOuter">
                <field name="_contentArea" styles="${section.content.stylename}" type="Block-SectionContent">
                    <field name="Intro" type="Block-Maguire-Advice-Info">
                        <setproperty name="include.image.left" value="false"/>
                    </field>
                    <field name="AboutYou" type="Section-Standard-Level2">
                        <setproperty name="section.heading" value="About You"/>
                        <setproperty name="section.help.text" value="Section help goes here. Utilising the inbuilt help on level 2 sections to give some cont
                    </field>
                </field>
            </field>
        </field>
    </content>
</formdescriptor>
```


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

Read-Only Option

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

Manage Form Revisions [Revision List Form]

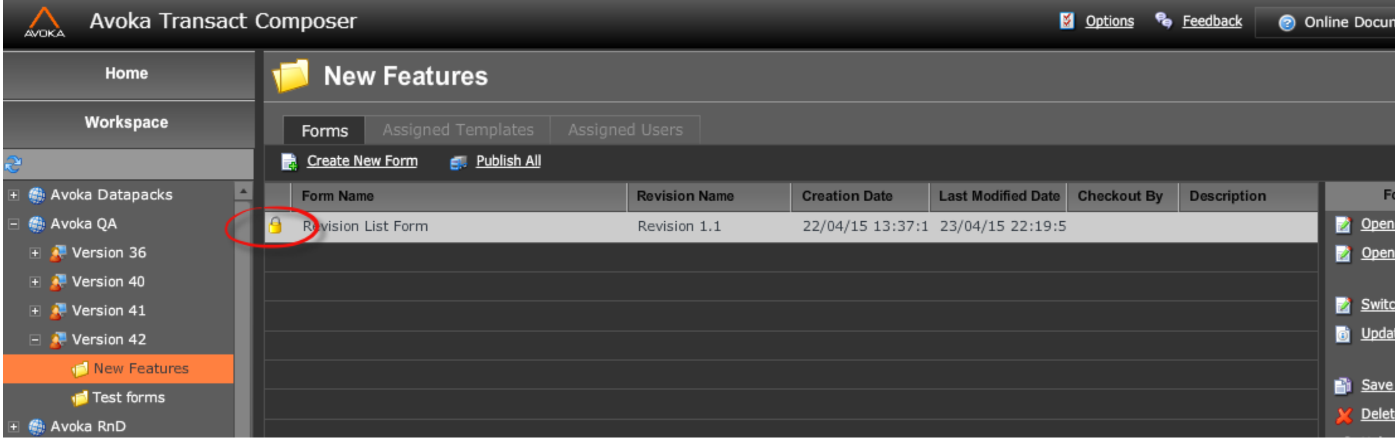
Active	ID	Revision Name	Created By	Creation Date	Updated By	Last Modified Date	Readonly
▶	49526	Revision 1.1	[illegible]@avoka.com	22/04/15 15:56:40			🔒
	49566	Revision 1.2	[illegible]@avoka.com	23/04/15 11:52:15			🔒
	49524		[illegible]avoka.com	22/04/15 13:37:13			

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

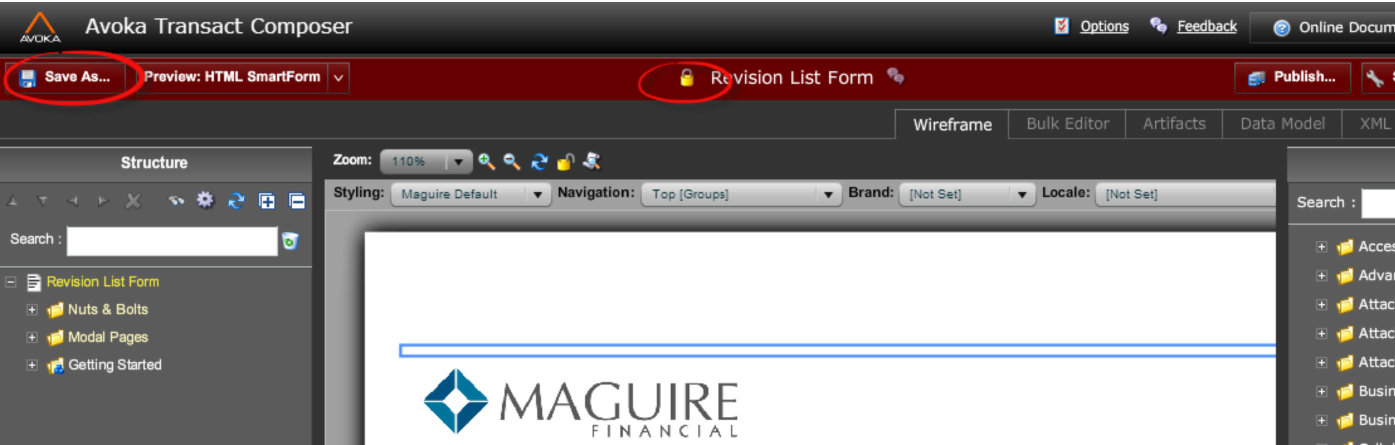
Manage Form Revisions [Revision List Form]

Active	ID	Revision Name	Created By	Creation Date	Updated By	Last Modified Date	Readonly
▶	49526	Revision 1.1	[illegible]@avoka.com	22/04/15 15:56:40			🔒
	49566	Revision 1.2	[illegible]@avoka.com	23/04/15 11:52:15			🔒
	49524		[illegible]avoka.com	22/04/15 13:37:13			

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



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



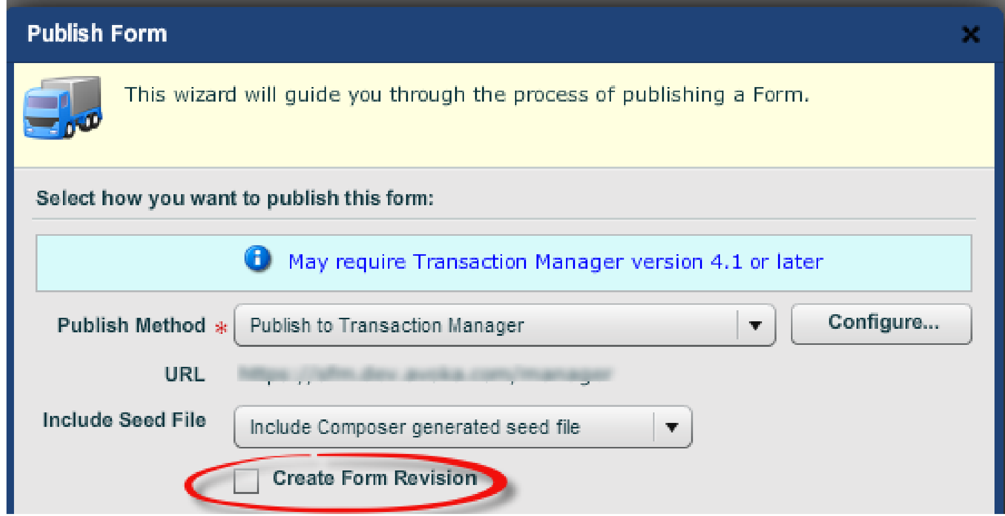
Screen 16 - A Read-only form when opened has the "Save" Button removed, A lock symbol next to the form name and the Form editor header colored red.

Creating a Revision at Form Publish

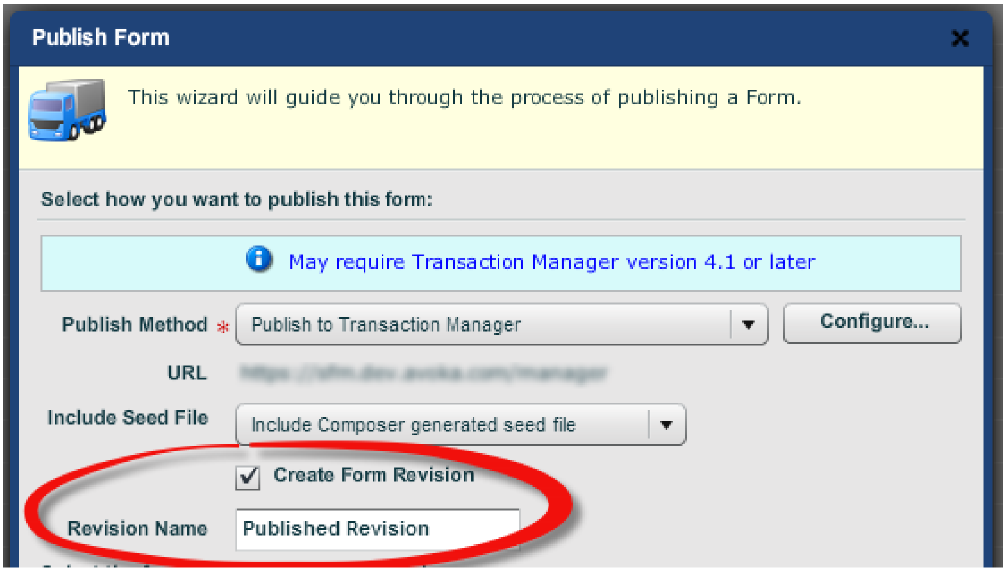
An option has been added to the publish dialog that will create a form revision at the time of publish.

Parameters:

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



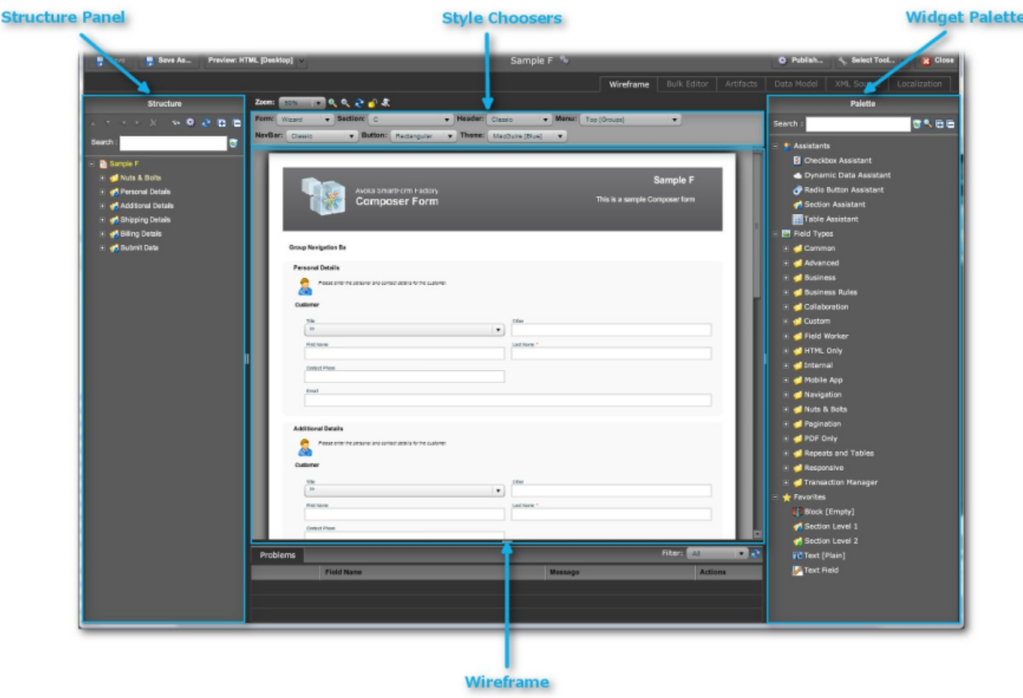
Screen 17 - The publish screen showing the option to create a form revision. It is off by default.



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

Form Designer (Composer v4.3)

The Wireframe View



The Wireframe tab of Form Designer is the heart of Composer. Not only are forms built from elements from the Widget Palette, but the whole logical structure and interactions of the form are built up in the designer.
Composer functions as an agile and capable tool, where complex forms are built and deployed, on a range of devices, in a matter of days or weeks.
This user guide emphasizes the automated aspects of form construction in the early pages. Sections below the Advanced Topics section of the manual deal with more complex functionality. These come later in the guide to act as a reference and not to interrupt the narrative.

Structure and Preview

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

Dragging and Dropping on the wireframe.

The Wireframe is a representation of the visible elements. Because each element contains a structure, it is difficult to drop or move elements to the correct place on the element's structure tree. For this reason we recommend that **widgets are dropped onto the Structure Panel**.
The other tabs ("Bulk Editor", etc.) are more technical, extremely powerful and not of immediate concern. See Form Designer Advanced Features .

Widget Palette

Overview of Widgets

The "widget" is the icon you drag from this palette onto the form's Structure Panel. (See below.)
The resultant object that appears on the form is called the "field". Other synonymous terms are: "node", "form element", "element" and even "widget". We may deplore the lack of common terminology, but in a technology that is rapidly evolving, regional and corporate difference in vocabulary are inevitable.
This has happened with browser and internet terminology in general: look at the various terms employed by different enterprises and teams for what should be standard topics like "query strings" (also incorrectly called "parameters"), "value pairs" (also called "tokens", "attribute-value pairs, "key-value pairs and so forth).
We will try to be consistent and use "widget" in the Palette and "field" on the form.

Organization of the Palette

The palette has 3 major classifications:

Assistants

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

Field Types

collections of widgets organized into convenient groupings

Favorites

which you, the designer, can populate with your own choice of widgets
The Field Types contain a mixture of simple widgets — like Text Fields — and of compound widgets, made up of a set of widgets organized into some sort of functional structure.
The classification scheme used in Field Types is arbitrary and subject to change. Please spend a few minutes to look through the these types to get a bit of a feeling as to what they contain. The "Search" field at the top of the palette assists you to locate a widget without having to hunt and peck through the Field Types. Suggestions will appear in the palette as you type in your search item.

How to Use Widgets to Create Fields on Forms

You simply drag and drop a widget into the Structure Panel. You can also drop widgets into the Wireframe, but Avoka Technologies strongly recommends that you do not drop widgets onto the Wireframe. The reason for this is: the form's structure actually has many hidden fields and structural elements which can only be seen in the Structure Panel _bookmark30.
Sometimes the new field will appear in the Wireframe; sometimes it completely disappears. And even when it does appear, all may not be well internally.

Widget Types

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

Simple Widgets

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

Blocks

Block are containers of multiple fields. Blocks serve 2 main functions:
To group fields together visually in a colored panel, with a common label and so on To group fields together logically into functional units.
Containers can contain other containers in a hierarchical structure. There is no practical limit to the levels of nesting supported.
For your convenience, there are a number of Predefined Blocks in the palette. They are invaluable for building complex functions on your forms with little more effort than drag and drop.

Sections

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

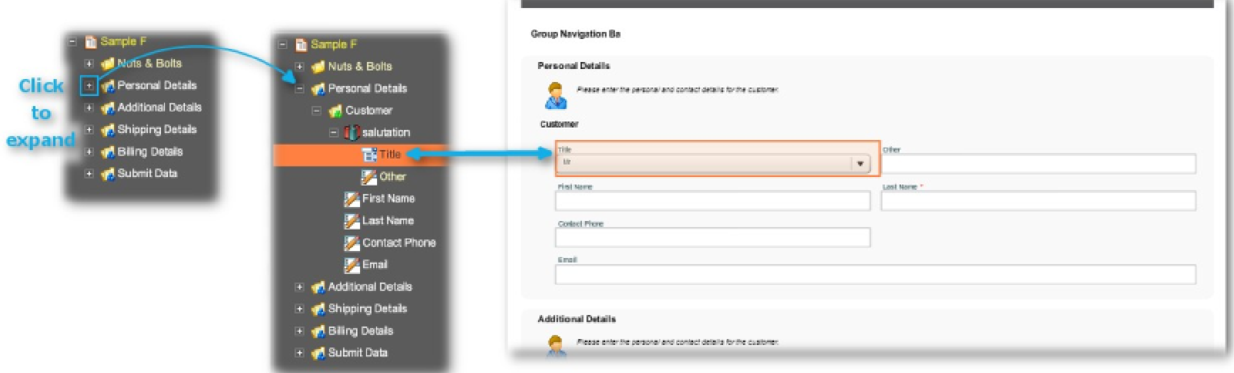
For More on Widgets

Please see:
Field Types
the technical reference to the many types of fields, Assistants
Predefined Blocks

Structure Panel

Standard Mode

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



Showing the relationship between the Structure Panel and the Wireframe

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

Advanced Mode

Standard Mode vs Advanced

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

Nuts & Bolts vs Content

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

Composer Framework (Advanced Mode Only)

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

How to Place a New Field into the Structure

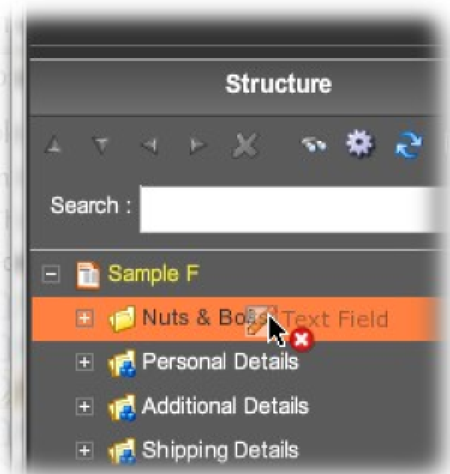
Drag and Drop

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

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

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

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



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

Shortcut for Power Users

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

Menu Bar

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

Layout

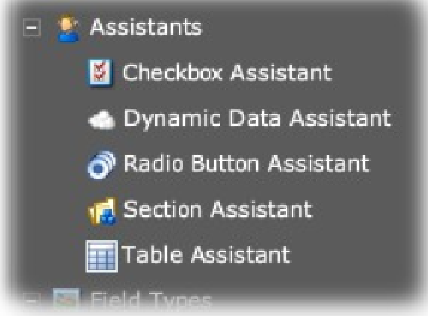
So far, we have shown how to add fields to the structure, but said nothing on layout: where fields will sit in relation to each other, either vertically or horizontally, what will be the distance between fields and how they position on the form.
This omission is deliberate. Layout is dealt with here in this guide and layout is automated in Composer as much as possible.
Why? Wouldn't forms look better if the designer sweats over positioning each element and nudging fields on the web page to get its position perfect? Well, the answer is that forms in a web-browser cannot have all their elements as tightly controlled as with traditional hard-copy forms, as the web page runs the gamut of the ranks of the many web browsers now in common use. Also, these forms will be viewed on different devices: on Android phones and tablets; on iPhones, iPads, iPod Touches, on Windows mobile devices and desktops, on
desktop monitors of a range of sizes, via mobile apps, via HTML and PDF.
In such a babble of viewing environments, what was painfully adjusted for one circumstance will look lousy in another. And you cannot dictate which modern browsers or devices you would prefer your customers not to use. Gone are the days of wagging the stern finger and stating "Best viewed in such and such an ancient browser and not the modern one you are using." If users have migrated over to modern browsers so as to be able to engage in social media, do not try to stem the tide with your web forms.
So, these considerations dictate that layout is held in reserve for later discussion. For the present, we encourage you to experiment with widgets in the Palette, and to see what happens when you drop these onto a form.

Assistants (Composer v4.3)

Assistants are located at the top of the Widget Palette and this is the reason they are discussed first. The Field Types in the palette are a mixture of many different types, see Field Types, placing any of them on the form is the same process already covered above . This is also true of Assistants, and they make a good starting point as the resultant fields on the form are so useful, and even rewarding, for the beginner
Assistants are widgets that facilitate the creation of complex structures. Another important set of widgets, that also simplifies complex form structures, is Pred efined Blocks .
We will discuss individual widgets in more detail later in this manual . But because Assistants and Predefined Blocks are so powerful, we will now be building examples using these two types. Designing complex forms using individual widgets is more the domain of Composer's power users.

Adding Assistants to a Form

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



Assistants

Checkboxes

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

Using the Checkbox Assistant

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

Add A Checkbox Group

This wizard will guide you through the process of creating a checkbox group.

Title: Checkbox Group

Name *: Generate Name

CheckboxGroup

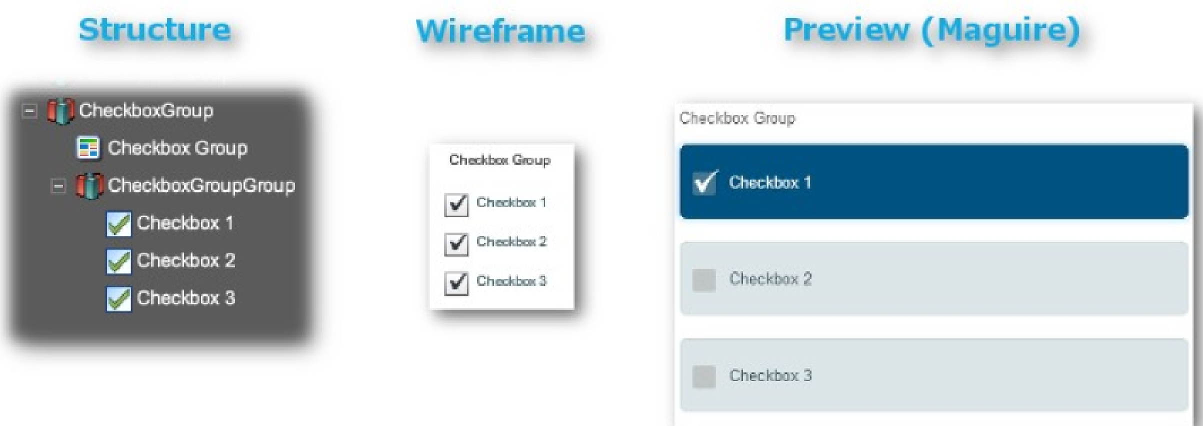
Checkbox

Label	Initial Selected	Create Content Block
Checkbox 1	No	false
Checkbox 2	No	false
Checkbox 3	No	false

Add... Delete Edit...

Layout Manager: Flow - Top to Bottom

< Back Next > Cancel Finish

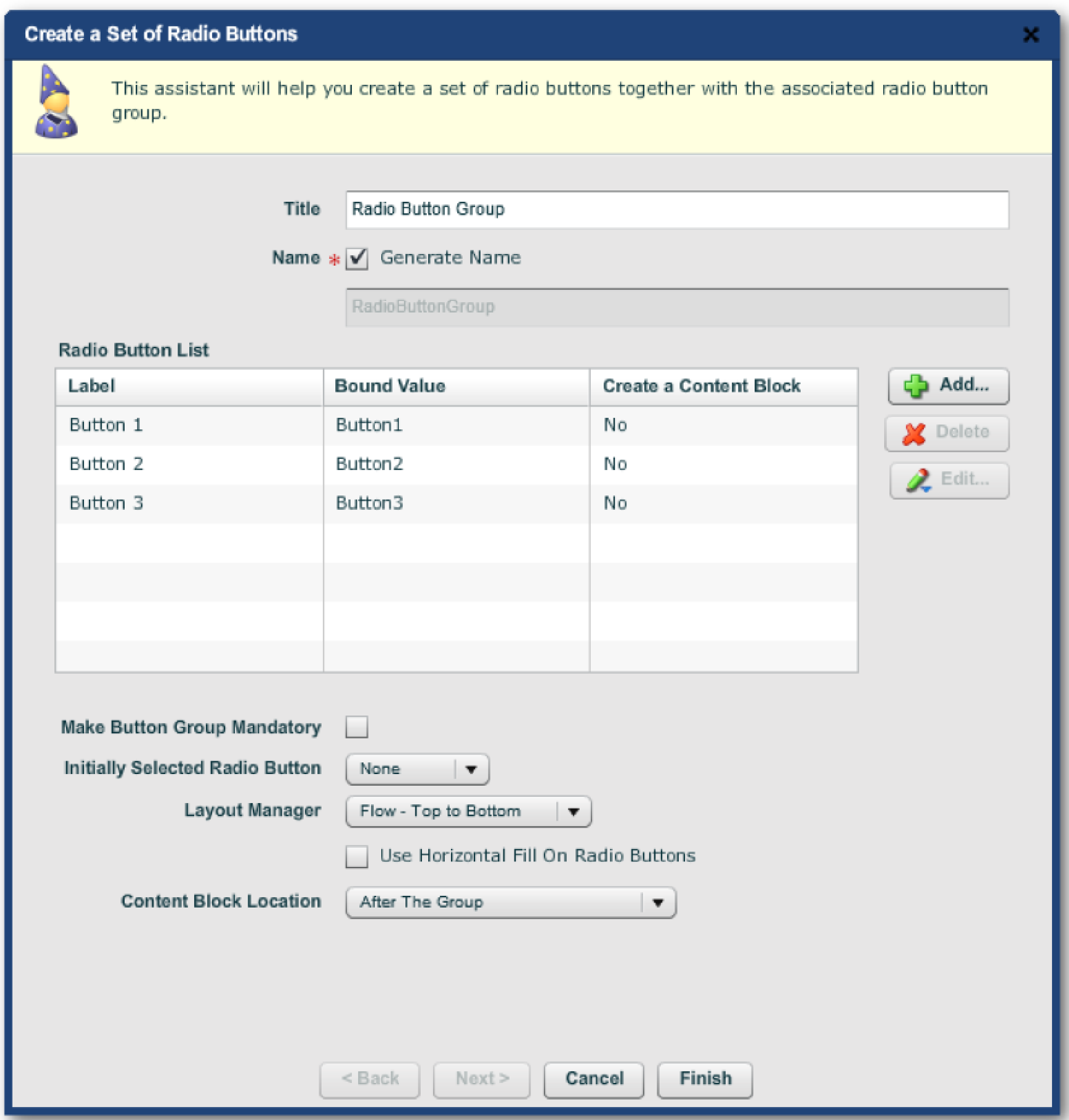


The resultant struture

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

Radio Buttons

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



The "Add a Radio Button Group" wizard dialog

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

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

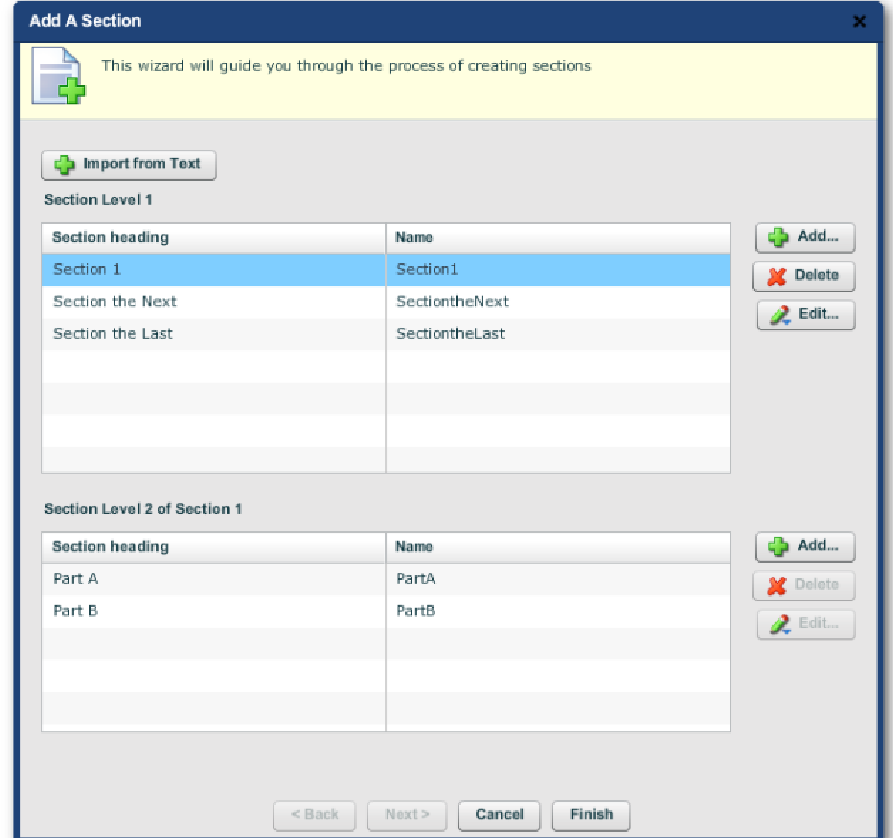
The buttons now render on the form (as seen in Preview) as graphics with a more contemporary look than the standard desktop browser widgets.
Note: the button group now appears in the structure below the buttons. This done to allow the inline validation message to appear correctly on the form.

Dynamic Data

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

Sections

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



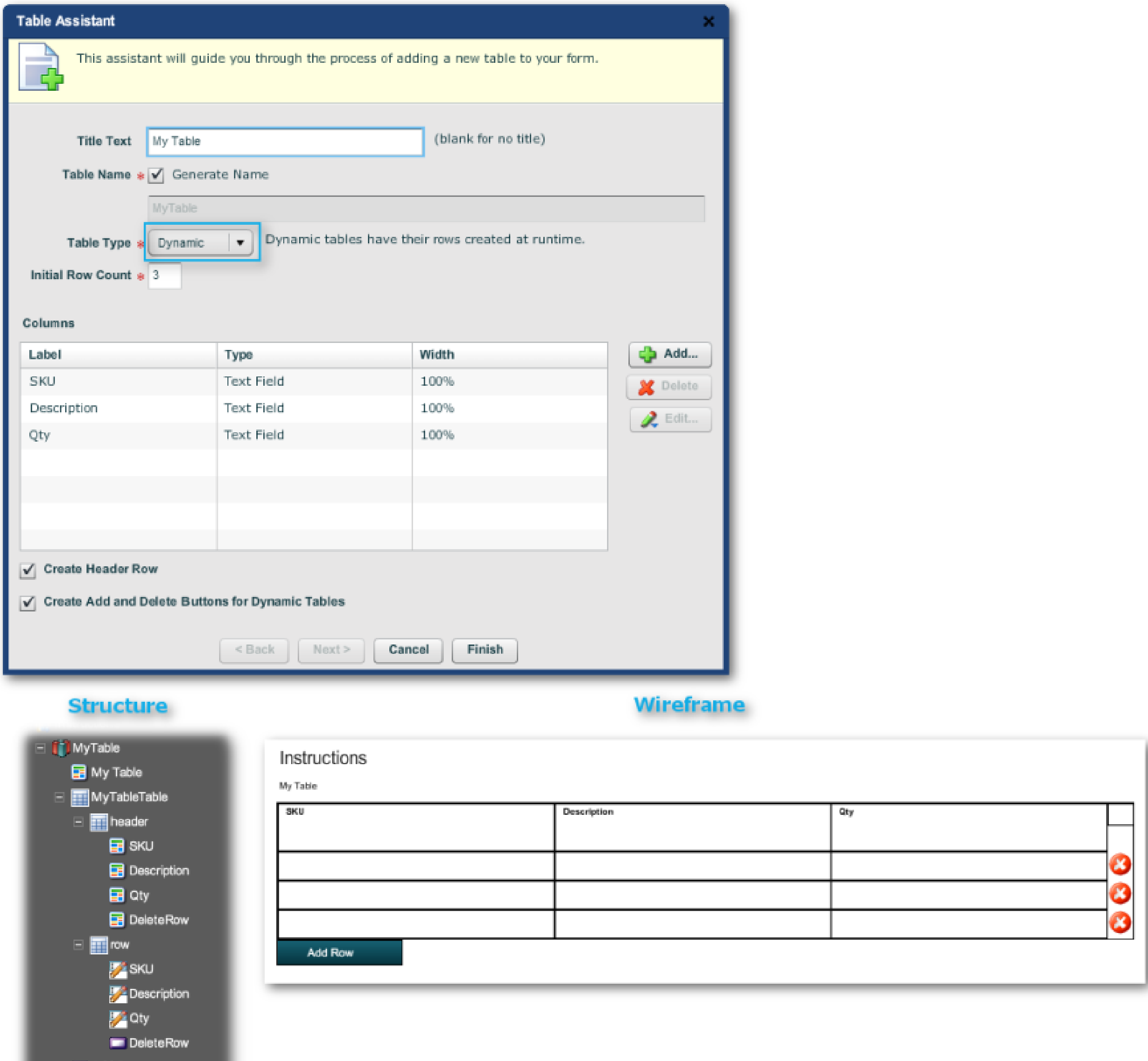
Structure
Section 1
Part A
Part B
Section the Next
Section the Last

The Section Assistant wizard

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

Tables

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



Content Blocks

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

Wireframe

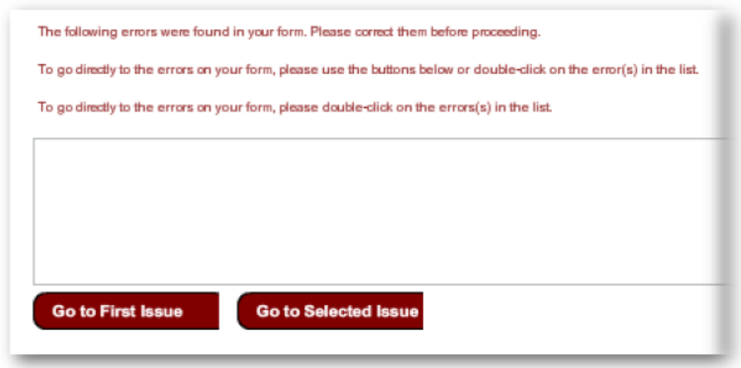
Radio Button Group
Button 1
Congratulations! You have selected this button!
Button 2
Button 3

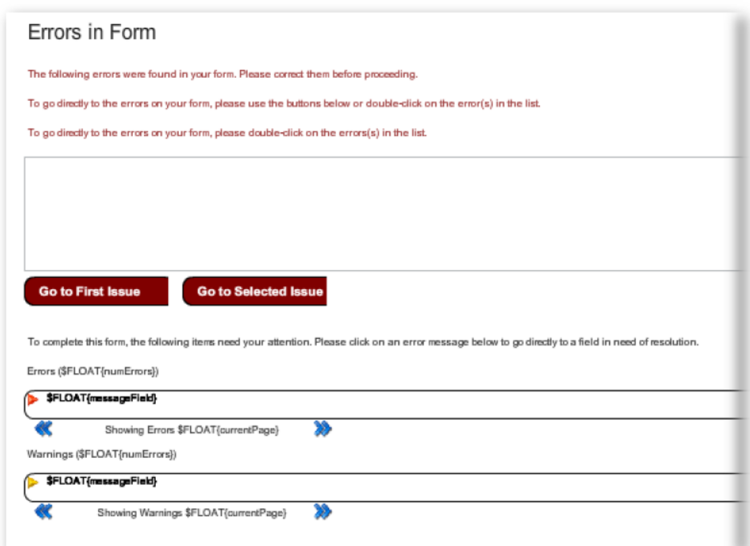
You can reverse this behavior by going to the Edit Properties dialog, "Rules -> Visibility Rule -> Edit" and changing the script from
return {Button1};
to
return !{Button1};

You can also manipulate other behaviors by editing widgets in the group, but take care: you do not want to break the structure.
Note: in Maguire template forms, content blocks now appear at the bottom of button groups, **not between buttons**.

Predefined Blocks (Composer v4.3)

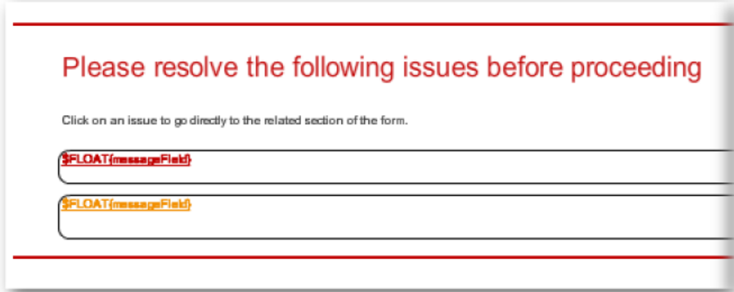
Many of the standard Composer widgets are implemented as prefabricated blocks. You utilize them as you would assistants: drop one into the form's structure and take advantage of the resulting complex structure that adds functionality to your form.
These include various Address blocks, a block to display errors, the standard form header, Sections, the Signature block, and various blocks for use in Transaction Manager.
You cannot modify the predefined blocks in the pallet, only the predefined blocks that you yourself created in the Pallet. But once a predifined block has been added to the form, you can modify that instance and that instance alone.
Here is the current range of predefined blocks generally made available for most organizations in the pallet. You

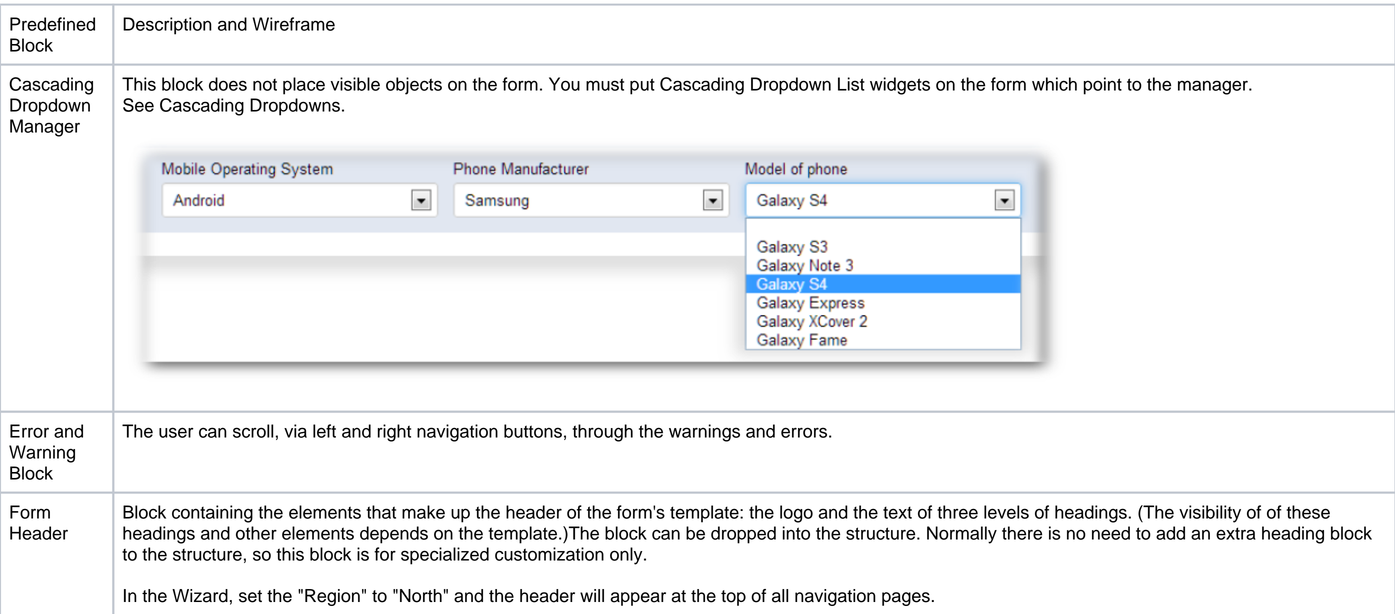




may not see all of these, due to the Composer configuration of your Organization.

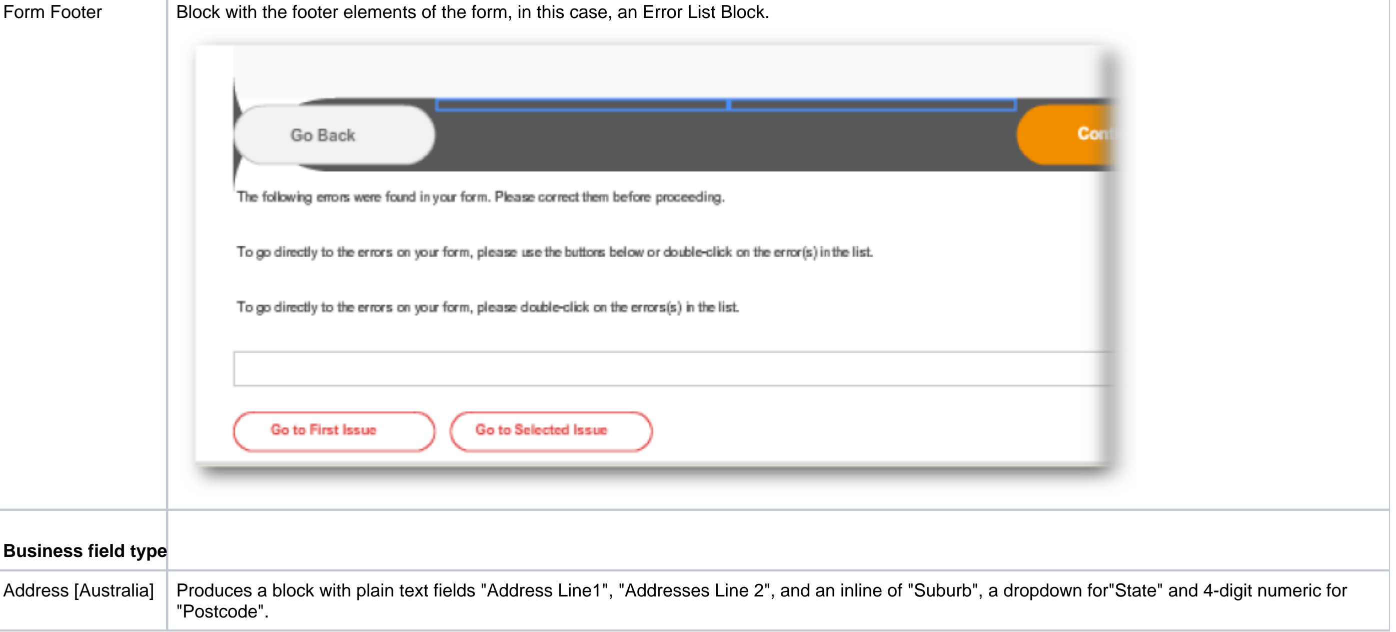
Predefined Block	Description and Wireframe
Common field type	
Fieldset	Used to improve accessibility for forms that have a number of scattered radio buttons, where these do not fall into one convenient block nor have the one controller. Poorly structured collections of radio buttons can confuse screen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

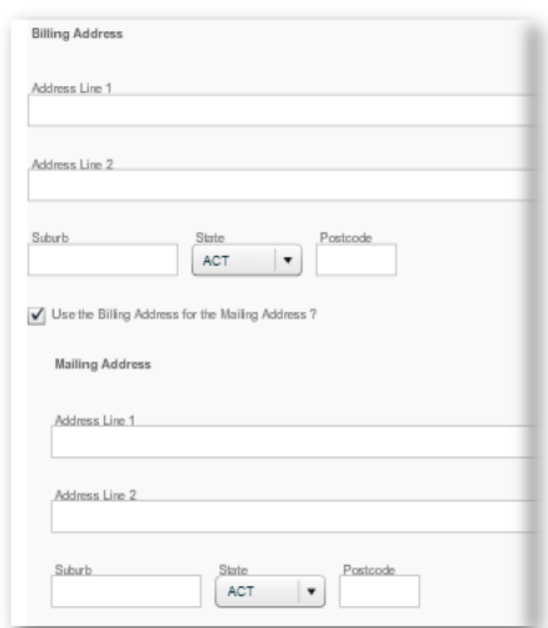


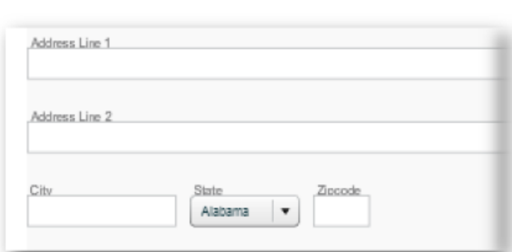


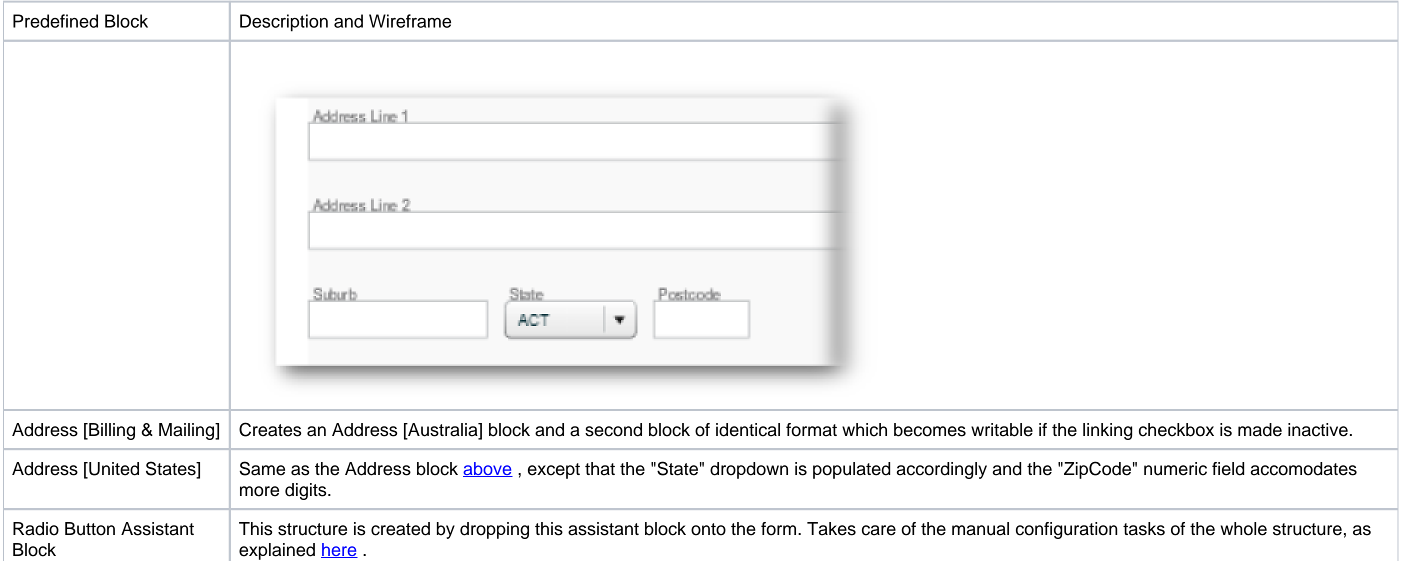
Predefined Block	Description and Wireframe
Cascading Dropdown Manager	This block does not place visible objects on the form. You must put Cascading Dropdown List widgets on the form which point to the manager. See Cascading Dropdowns.
Error and Warning Block	The user can scroll, via left and right navigation buttons, through the warnings and errors.
Form Header	Block containing the elements that make up the header of the form's template: the logo and the text of three levels of headings. (The visibility of of these headings and other elements depends on the template.)The block can be dropped into the structure. Normally there is no need to add an extra heading block to the structure, so this block is for specialized customization only. In the Wizard, set the "Region" to "North" and the header will appear at the top of all navigation pages.

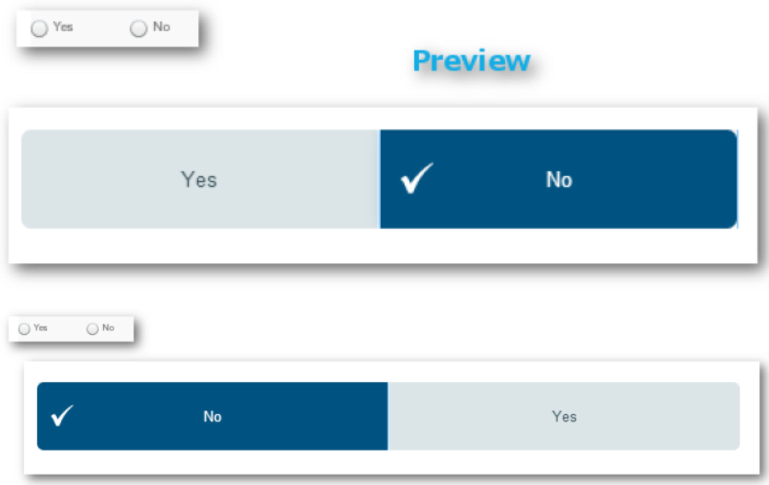
Predefined Block	Description and Wireframe

Form Footer	Block with the footer elements of the form, in this case, an Error List Block. 
Business field type	
Address [Australia]	Produces a block with plain text fields "Address Line1", "Addresses Line 2", and an inline of "Suburb", a dropdown for"State" and 4-digit numeric for "Postcode".





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



Predefined Block	Description and Wireframe
	Structure RadioButtonGroup Radio Button Group _layoutBlock Button 1 Button 2 Button 3 Button Group Wireframe Radio Button Group Button 1 Button 2 Button 3 Preview (Maguire) Radio Button Group Button 1 Button 2 Button 3
Radio Button List [Yes /No]	A block of two radio buttons, a "Yes" and a "No".
Radio Button [No/Yes]	Button order reversed from the above.
Signature [DocuSign]	Invokes a Transaction Manager service to provide the DocuSign signing workflow before delivering the submission to another TM service. Sign here Name * Email *
Title Block	Produces an in-line dropdown ("Mr" "Mrs" "Miss" "Ms" "Other") and a text field to specify "Other".

Wireframe

Locate

Latitude: $FLOAT{ffLatitude}

Longitude: $FLOAT{ffLongitude}

Preview

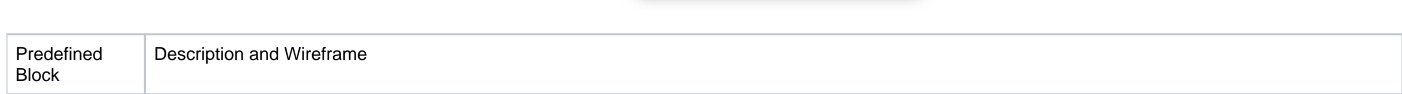
Locate

Latitude: -33.7981122

Longitude: 151.2871184

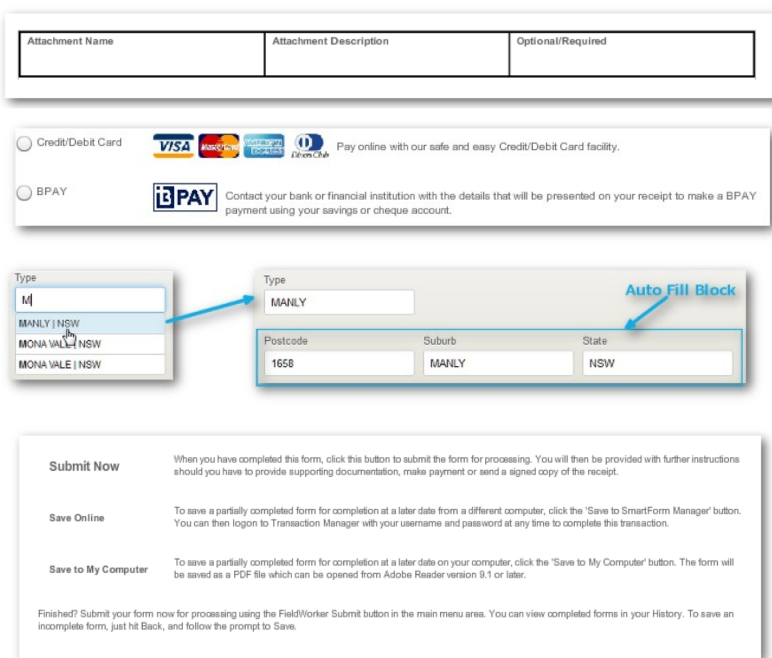
Barcode Value

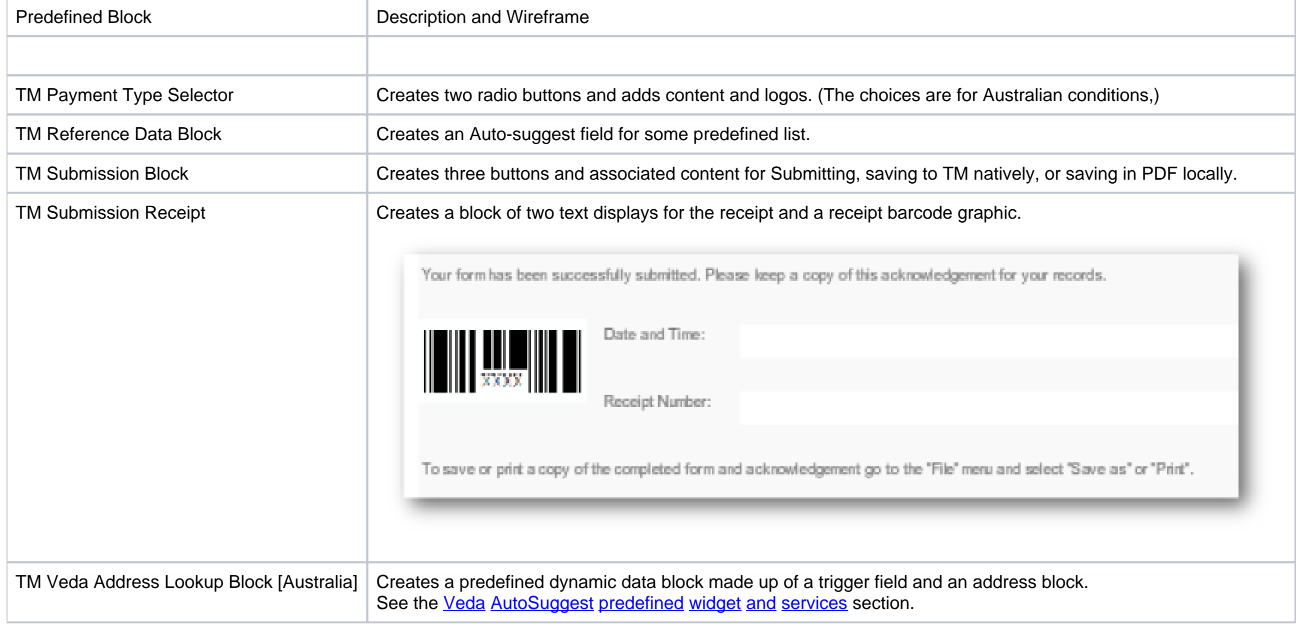
Scan



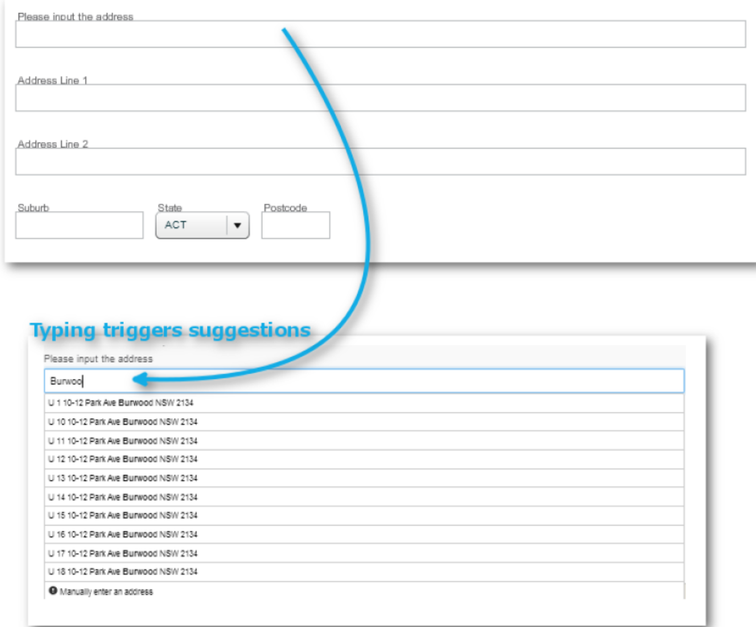
Predefined Block	Description and Wireframe

Business Rules field type	
Business Rule - Error	These are invisible on the form, but control the behaviour for the form. See the Business Rules section of this manual. These field types cover several widgets or widget groups, or even the whole form in some behaviours. Some aspects of these resemble the "Rules" tab of the "Edit Properties" dialog of individual widgets, but the rules tab applies only to the one widget.
Business Rule - General Purpose	
Business Rule - Warning	
Mandatory Block	
Mandatory Checkbox Block	
Collaboration	
Please see Collaboration .	
Mobile App (Field Worker) field type	
Geolocation Block	Also works on the desktop. When the button is tapped or clicked, it loads two floating fields for latitude and longitude.
Barcode Block	Take a picture of a barcode with a mobile device's internal camera using the "Scan" button on the form. The barcode is decoded, and its value populates the field below the camera button .
PDF field type	
Process Management Field Block	See the section on Field Types .
Smart Attachment Field Block	Adds a table to a PDF form for the user to upload an attachment. HTML forms handle attachments differently, see Submissions and Attachments .
Transaction Manager field type	
TM Payment Table	Creates a display table of payments, including GST (Australia's Goods & Services Tax) and Price.



Predefined Block	Description and Wireframe
TM Payment Type Selector	Creates two radio buttons and adds content and logos. (The choices are for Australian conditions,)
TM Reference Data Block	Creates an Auto-suggest field for some predefined list.
TM Submission Block	Creates three buttons and associated content for Submitting, saving to TM natively, or saving in PDF locally.
TM Submission Receipt	Creates a block of two text displays for the receipt and a receipt barcode graphic. 
TM Veda Address Lookup Block [Australia]	Creates a predefined dynamic data block made up of a trigger field and an address block. See the Veda AutoSuggest predefined widget and services section.

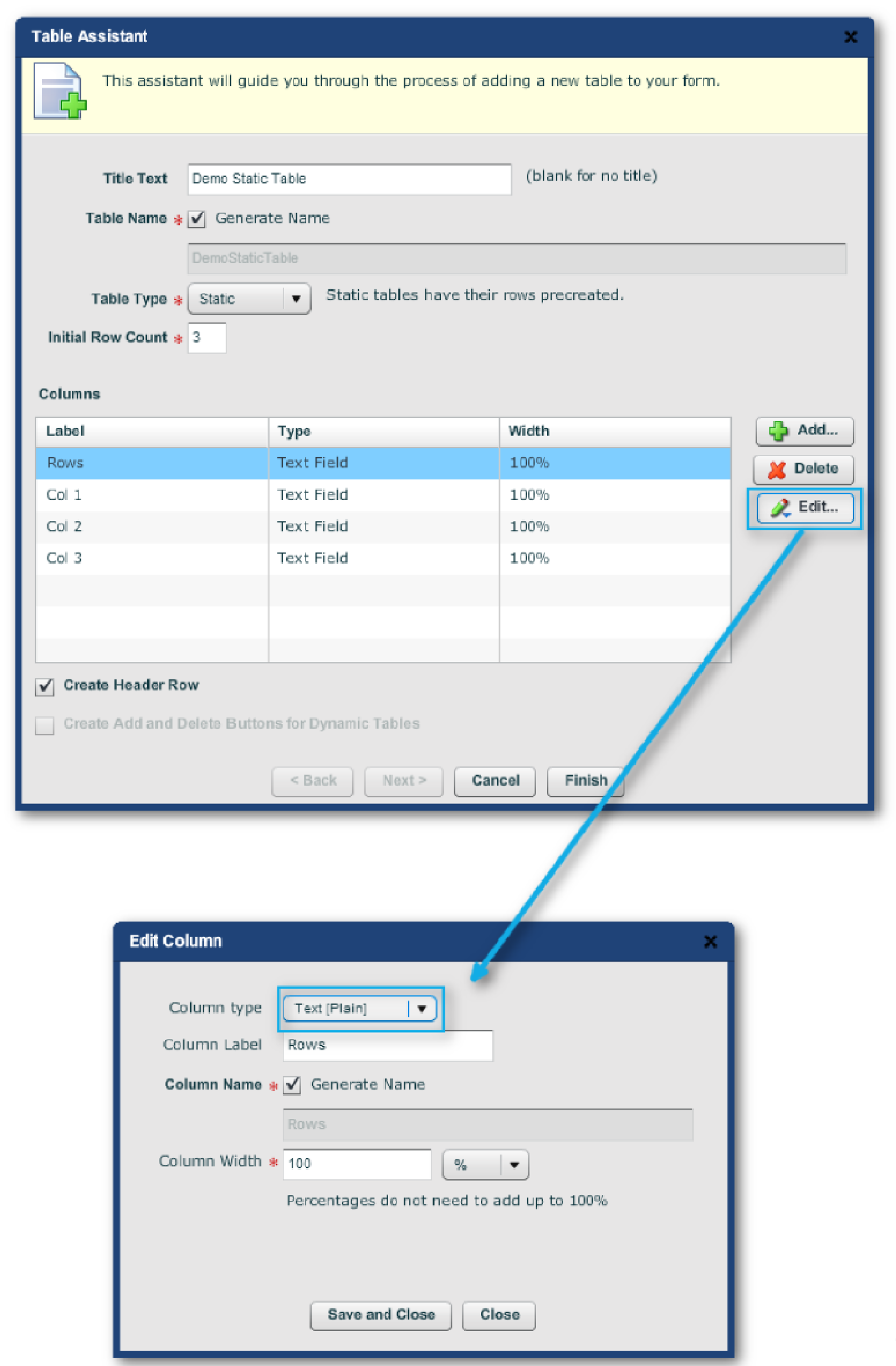
Predefined Block	Description and Wireframe


Please input the address
Address Line 1
Address Line 2
Suburb
State
ACT
Postcode
Typing triggers suggestions
Please input the address
Burwoo
U 1 10-12 Park Ave Burwood NSW 2134
U 10 10-12 Park Ave Burwood NSW 2134
U 11 10-12 Park Ave Burwood NSW 2134
U 12 10-12 Park Ave Burwood NSW 2134
U 13 10-12 Park Ave Burwood NSW 2134
U 14 10-12 Park Ave Burwood NSW 2134
U 15 10-12 Park Ave Burwood NSW 2134
U 16 10-12 Park Ave Burwood NSW 2134
U 17 10-12 Park Ave Burwood NSW 2134
U 18 10-12 Park Ave Burwood NSW 2134
Manually enter an address

Tables and Repeats (Composer v4.3)

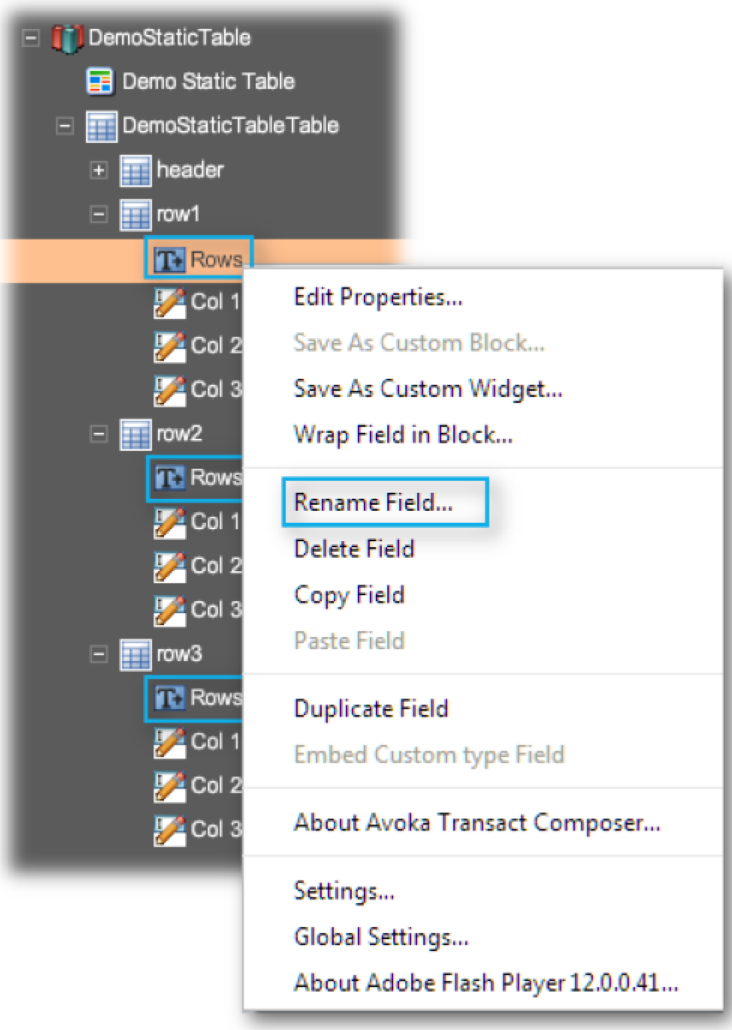
Static vs Dynamic

When you include a table or repeating block into your form, you only design one row of structure, and specify as many rows as you want to display initially. Tables usually have a heading row (or rows). Tables also can be static (i.e. a fixed array of fields for users to fill in) or dynamic, where users can add or delete rows as they go.
A repeat is similar to a table, but instead of being limited to the height of a row, you can make the repeating unit as large as you want, with no restrictions on how the fields lay out — unlike the single line limitation of a table.
Use the Table Assistant to create the table. Creating a static table with headings in each row on the LH side involves a few more steps. Use the Tables Assistant Wizard as follows:



The Table Assistant produces this form structure, although the LH captions for each row have to be altered — by right-clicking on each of the Plain Text

fields in each of the static rows and selecting "Rename Field...".



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

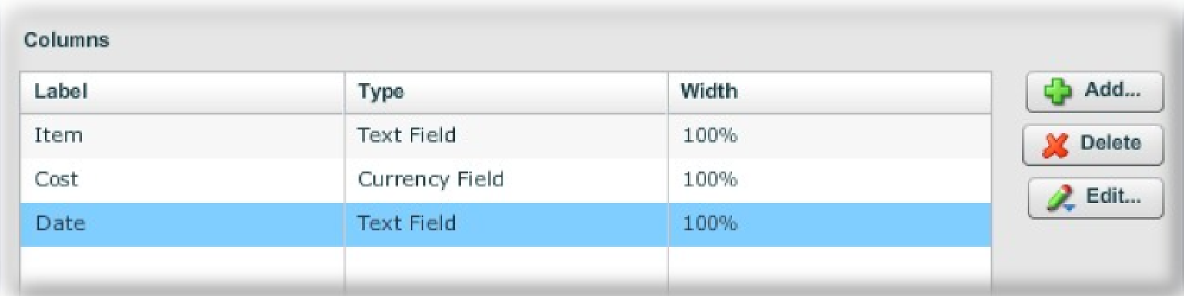
Editing Table Structure

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

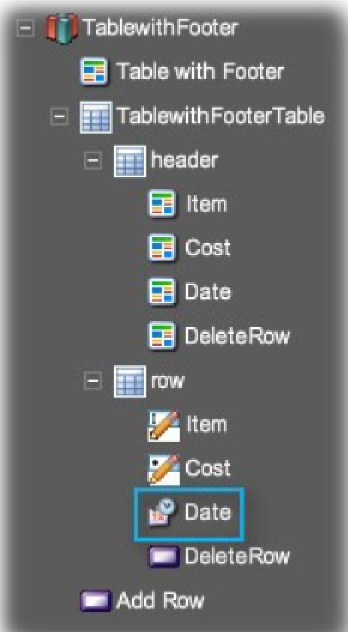
in the structure using the blue arrows at the top of the panel. The result looks as follows:
This flexibility extends across the entire table structure, though be careful to preserve the structural integrity of the table.

Adding a Footer

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

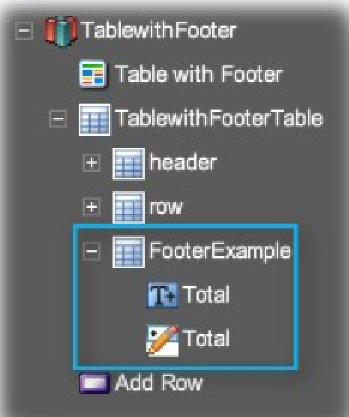


We then manipulate the Structure Panel to delete the text field (by right-clicking on it in the Structure) and drop in a Date widget into the Row container into the correct position (using the <ctrl> key or by using the blue up/ down buttons).

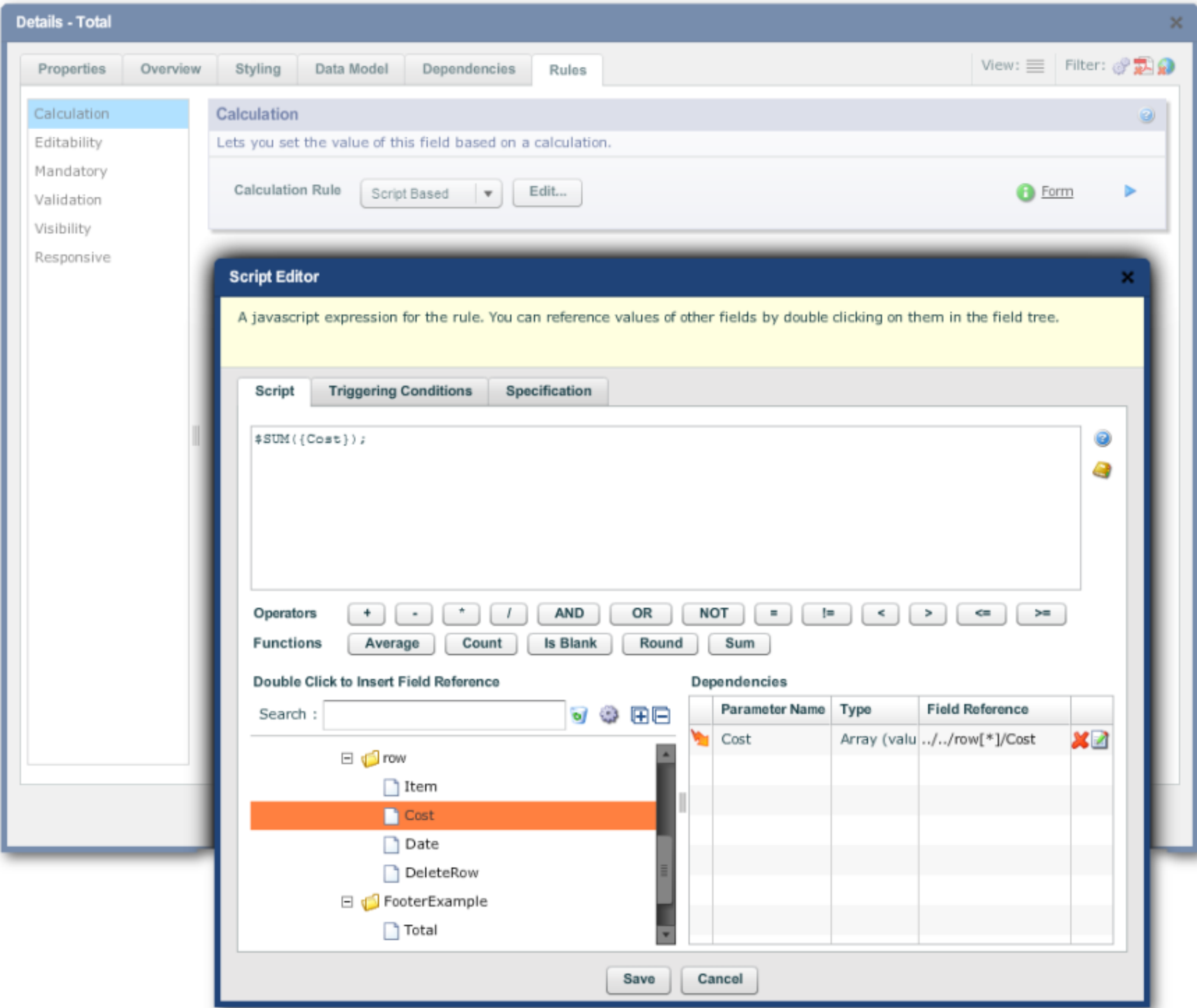


Drop a "Table Footer" widget from the palette onto the table field in the Structure Panel and add appropriate elements from the palette. Here we add only two elements: a Plain text and a Currency. Because the table has four columns per row, we adjust the column widths of these two for the sake of neatness: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

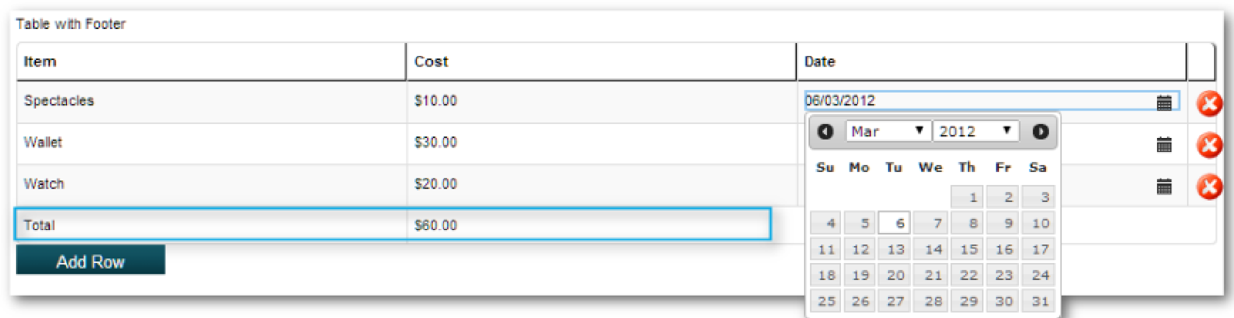
The new structure looks like this:



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



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



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

Repeats

There is no Assistant for repeats. This means that the you have to build up the structure in the Structure Panel yourself by dropping in the appropriate widgets and blocks and using the Edit Properties of the fields to script them. See Repeats in Advanced Topics .
Hint: a good way to start building repeats is to mimic the structure of tables, with blocks for headers, for rows and footers.

Radio Buttons

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

Templates (Composer v4.3)

Overview

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

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

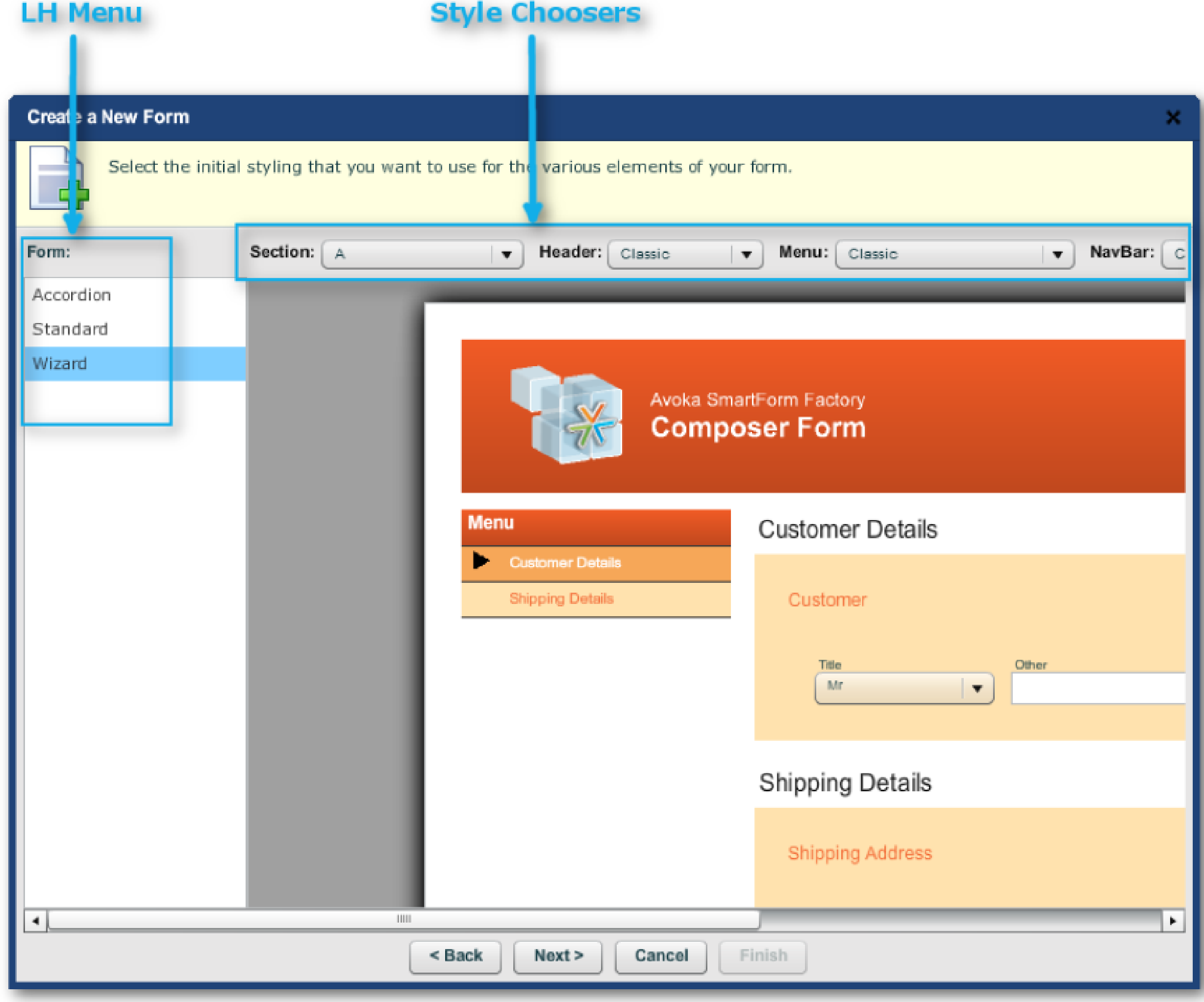
Other chrome in Form Designer, such as the Style Choosers inherited elements in the form's structure panel
This is not a complete list.
If you or your organization are new to Avoka Transact, it would be a good idea to make use of Avoka's support teams to create templates suited to your corporate needs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

Out of the Box

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

Creating a New Form

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



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

Basic Navigation Types

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

Standard Navigation

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

Accordion Navigation

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

Wizard Navigation

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

strictly in sequence

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

LH Lists

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

menu selection

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

restricted menu selection

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

Combinations of the Above

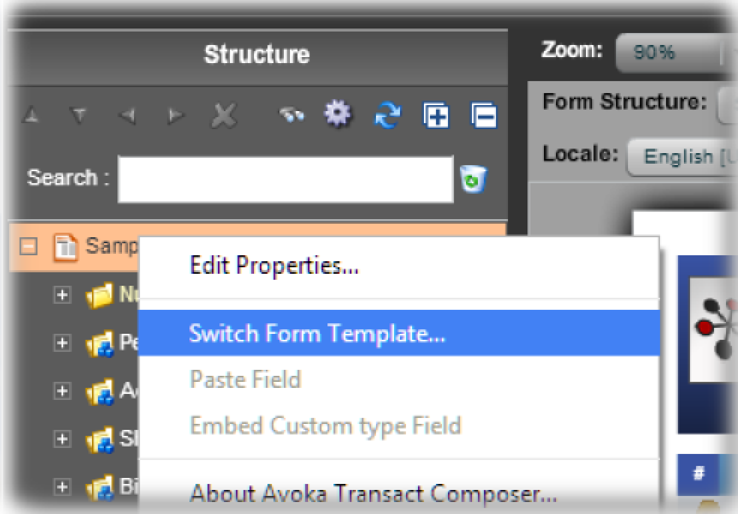
Composer 4 allows different parts of the form to make use of the different Basic Navigation Types. Now, some sections can have Wizard-style behavior while several other sections can be on one web page.

Styling

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

How to Change the Template

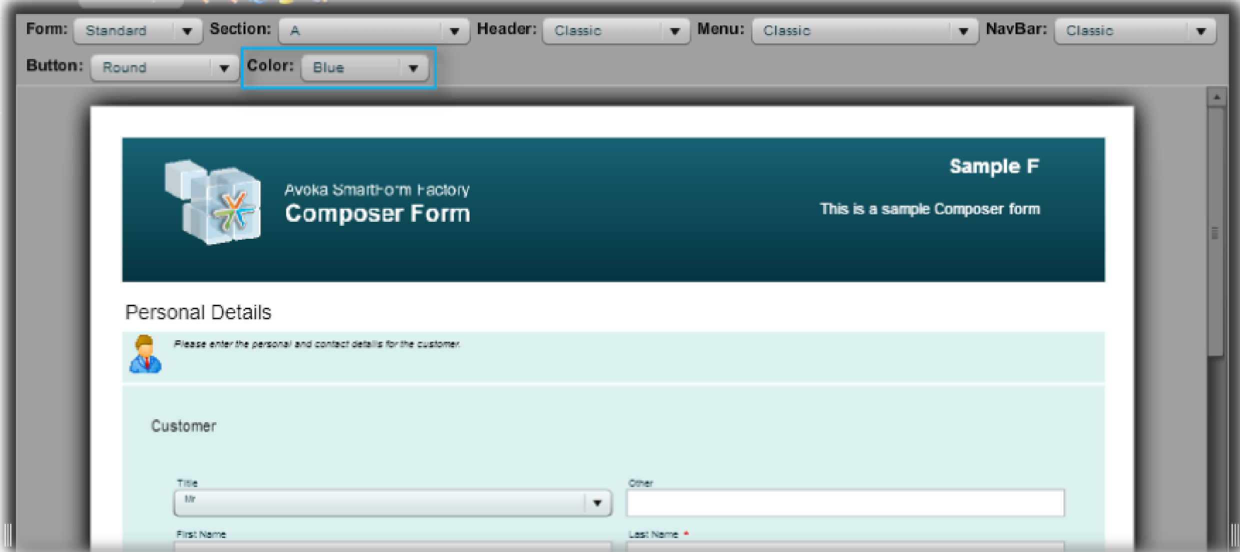
You can easily change the template of a form. Just right-click on the top element of the Structure Panel's tree and select "Switch from Template...." The only coices are templates that have been assigned to the organization (see Library Advanced Features).
Be aware that some layout elements of the form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



Right-clicking on the top of the Structure Panel tree

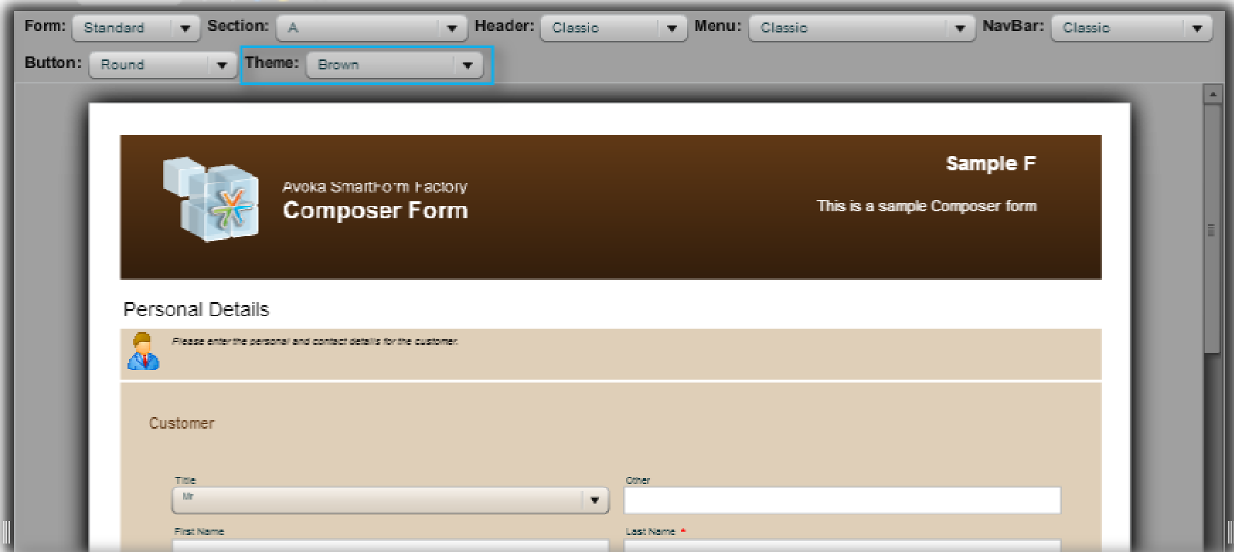
Samples of Templates and Styles

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



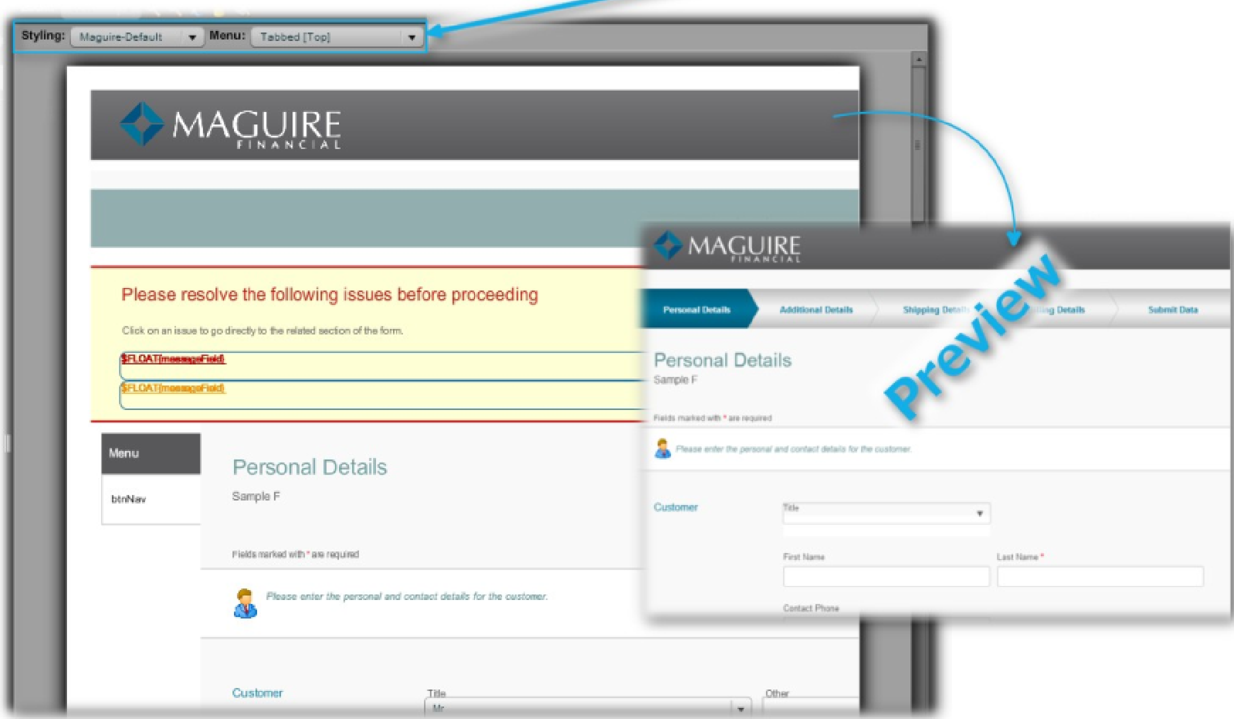
A basic template, and how choice of color or theme changes the appearance.

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



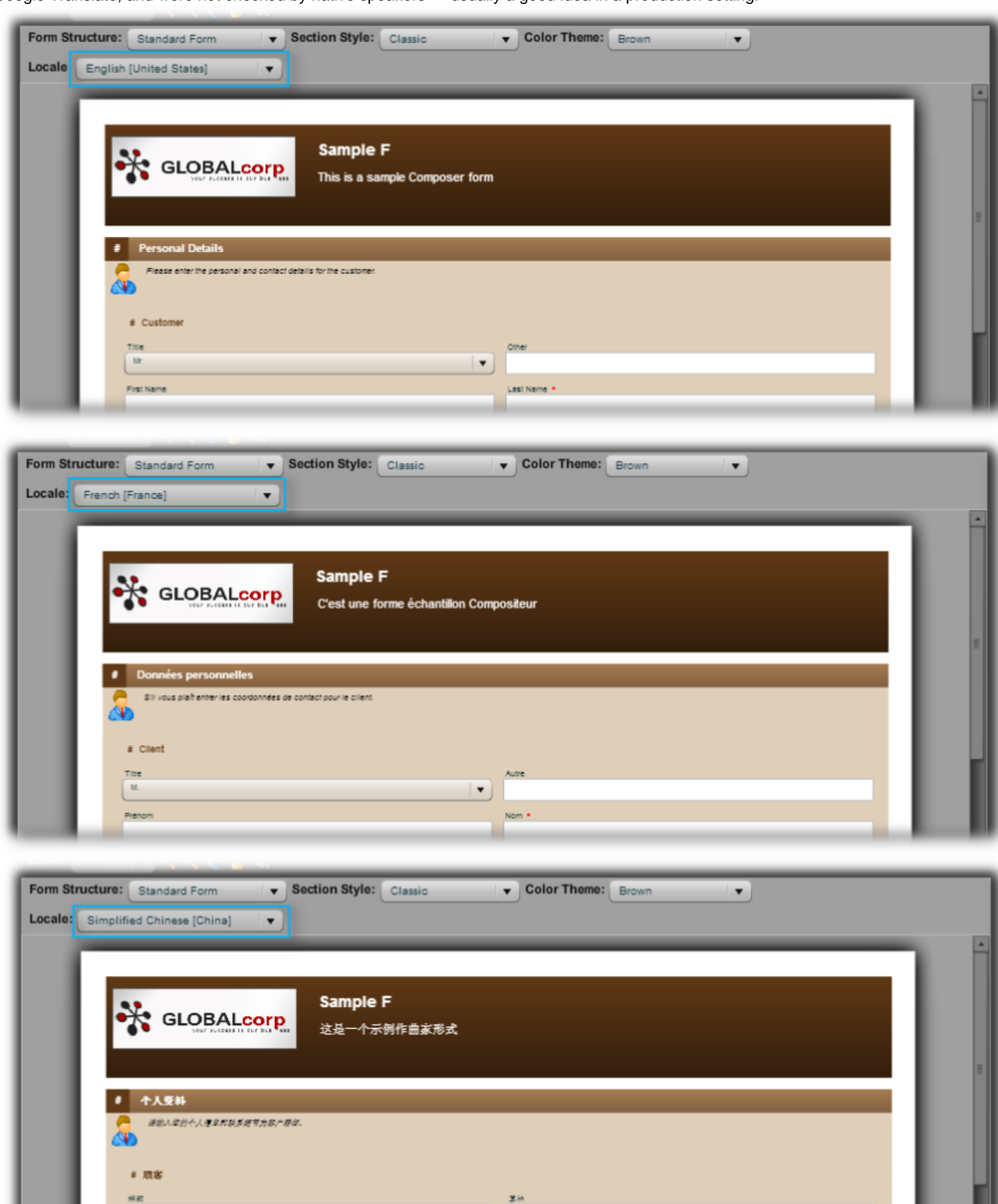
Same form, different color scheme

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



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

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



Edit Properties (Composer v4.3)

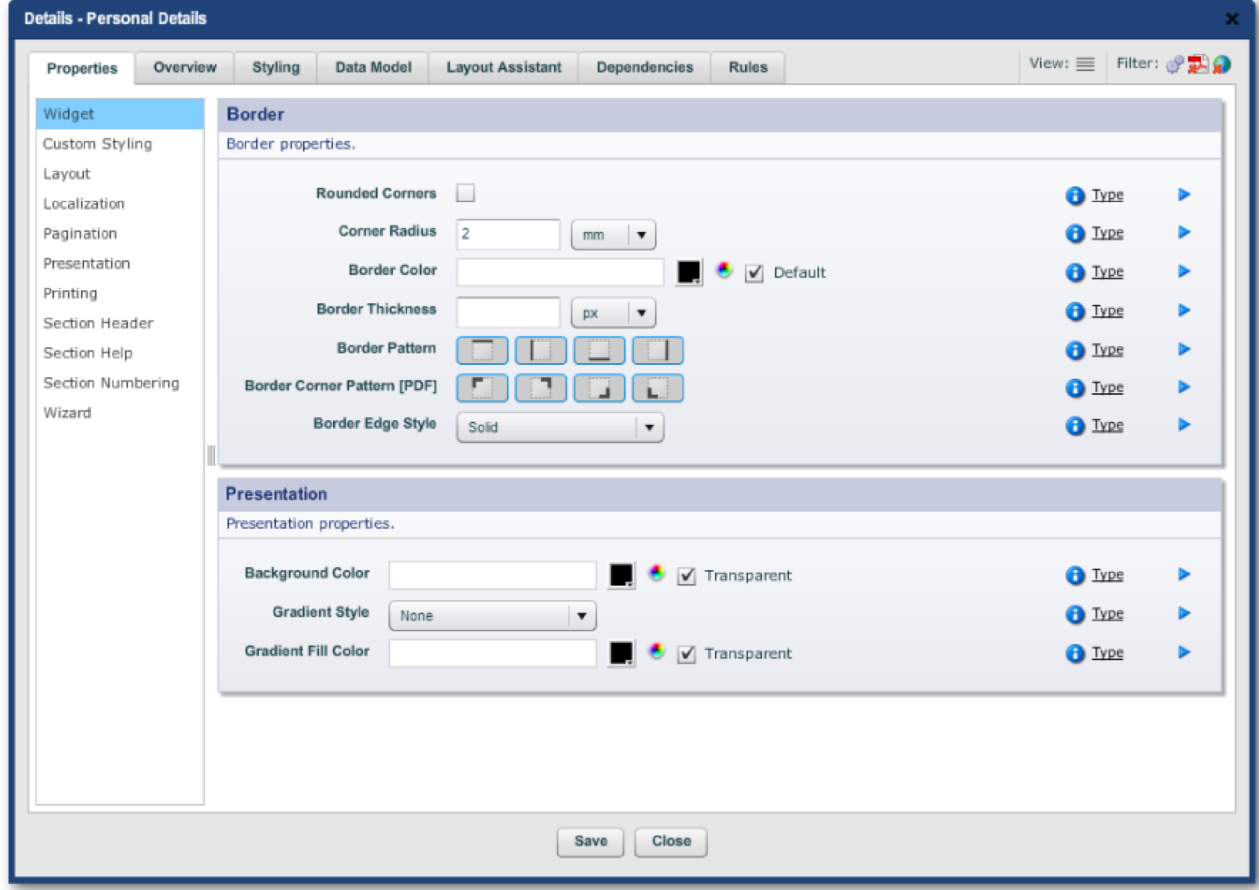
The Edit Properties dialog is the control for every element and field in the form's structure. It has many parameters, which in turn are grouped into tabs, sub menus and even sub tabs.
Though the dialog is very powerful, but its settings do not need altering in most cases. That said, Composer's ability to do scripting and to fine tune the data model of the form through Edit Properties — and have these debugged and quickly up and running — is vital for successful form design. The improved layout of the Edit Properties dialog also facilitates making these adjustments to the form.
Composer Version 4's redesigned Edit Properties has a number of new features, including a layout tab for configuring the layout managers of a block, with a small wireframe for viewing the effects of changes on the block. Also, for the first time in Composer, you can view a field's dependencies and triggers.

Invoking the Edit Properties Dialog

Double click on an item in the Structure Panel. The different types of structural elements have different items in their Edit Properties dialog. Broadly, here are the different types of elements:
The form itself (at the top of the Structure Panel tree)
Sections (a special kind of container that controls navigation through the form) Containers and blocks (which can be expanded by clicking on the "+")
Simple Widgets (which do not expand)

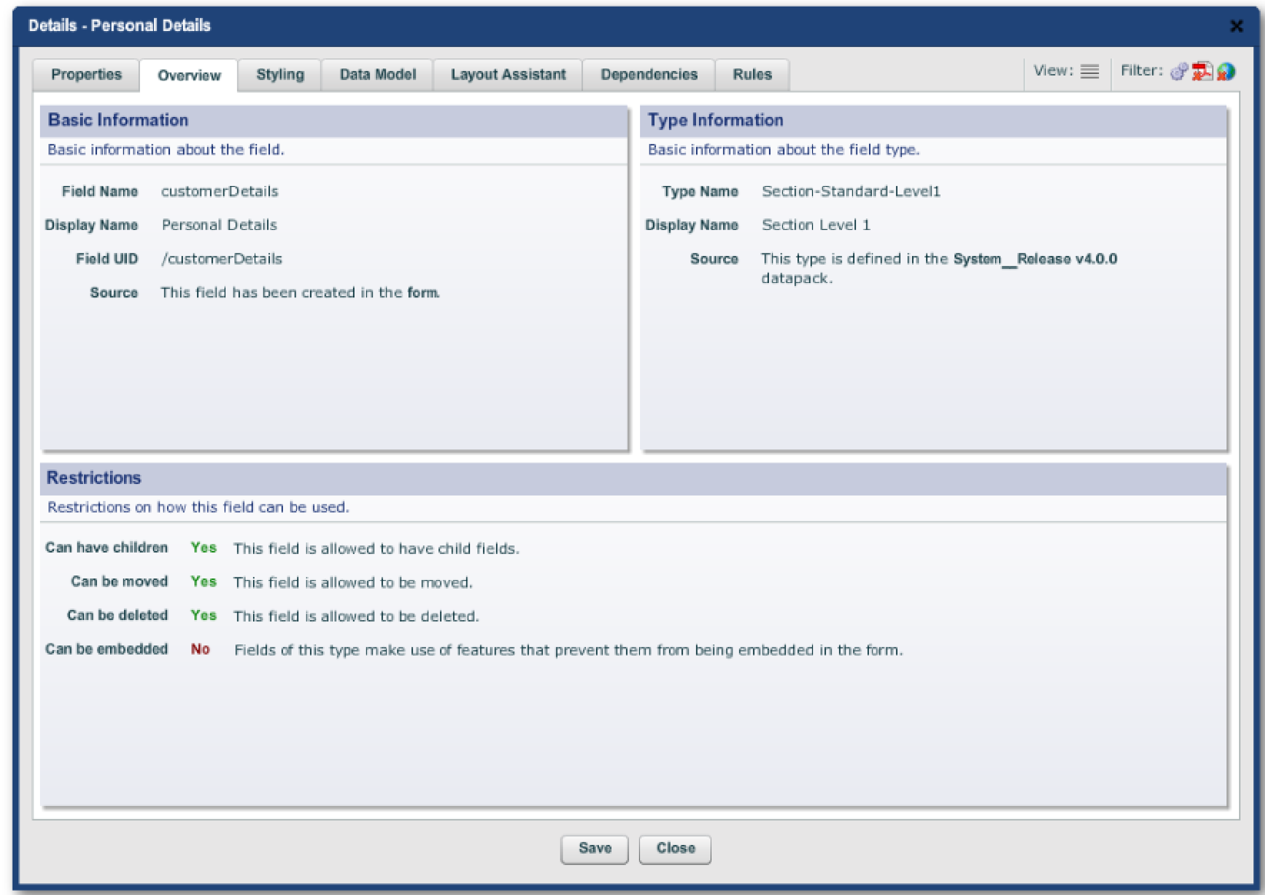
Structure

Version 4's Edit Properties dialog has a new layout for better grouping of the many settings of the properties for fields and containers on the form. The dialog now features a row of tabs along its top. Which of these are visible depends on which of the various structural elements was chosen for the "Edit Properties" dialog.
Properties Overview Styling Data Model
Layout Assistant (for blocks and containers only) Dependencies
Rules
There are a few special tabs that appear for the Edit Properties dialog for the form itself at the top of the Structural Panel tree:
Palette (for the form element only at the top of the Structure Panel) Theme (for the form element only)
Policies (for the form element only) Settings (for the form element only) Localization (for the form element only)
However the properties displayed in these special tabs can be accessed elsewhere in the tree, usually in the Nuts and Bolts. There, the properties appear under the "Properties" tab in the same panels that appear under these special tabs.
"Properties" and "Rules" have Left Hand menus. Each menu item displays a different set of panels to its right. Each of these panels contain their own set of editable properties.



Edit Properties dialog for a tab with a Left Hand menu

"Overview", "Styling", "Data Model" and "Dependencies" have just panels containing properties.



Most of these properties are read only. "Data Model" has editable properties. Two of the panels in the "Styling" tab are editable: "Stylesets" (which gives access to stylesheet preset options) and "Named Styles" (where you can explicitly associate stylesheets with the object).
The "Layout Control" tab, which is visible for blocks and sections only, has its own unique layout, see Field Layout Control .
See Scripting and Dependencies on how to add scripts to the form — which is done through Edit Properties.

Notation Convention Used

In the rest of this guide, we will use the following notation to show where a property, script or configuration is located in "Edit Properties":
Tab -> LH Menu item -> Panel -> Property -> value
So, in the Overview tab shown above, for example:
Overview -> Basic Information -> Field Name -> customerDetails

The Filter and View Buttons

The "Edit Properties" dialog has a similar dual mode as the Structure Panel : Standard and Advanced. The cog icon to switch between the two modes is in the top right-hand corner in the "Filter" mini panel. "Advanced" displays more properties, LH Menu items and tabs.
The other two icons toggle on or off the display of PDF-only or HTML-only properties.
The "Hamburger" View icon toggles between the normal Edit Properties view of tabs, menus and panels and the "Property Grid": a combined list of all properties listed in alphabetical order . Use the Property Grid to find a
property if you cannot locate it in any of the panels, or to inspect the raw and displayed values of properties.

The Properties Tab

The Properties tab for any structural element usually has a LH Menu. The items displayed, though vary from element to element. The items will be listed and defined in the Reference Section near the end of this guide.
These LH Menu items are only groupings of property panels (see The New Dialog topic). The same property panels may also appear elsewhere in the same Edit Properties dialog for the same element, for example the panels in Properties -> Layout will also be found at Layout Assistant -> Properties in the same dialog.
So, the Properties tab's LH Menu usually has many possible entries. The following is intended to be a broad guide so that you can find your way around this tab — and, by implication, the rest of the Edit Properties structure.
Here, we exclude Nuts & Bolts. Indeed, some of the "Nuts & Bolts" items have Property tabs with empty LH Menus
The following table is a rule of thumb only, and we include it here for general guidance. Advanced Mode items are so marked.

Structural Element Type	LH Menu Item	Panels (an indication only)
Form	Form	Form Panel (Holds the settings entered in the original wizard when the form was created .)

Section	Widget (Advanced)	Border Presentation (Borders, background colors and gradient)
	Custom Styling (Advanced)	HTML Custom Styling (HTML forms only. Background image, its repetition, custom HTML classes)
	Layout	Layout Manager Layout Constraints Sizing (i.e. the layout fill: "Horizontal" or "None")
	Localizatio n (Advanced)	Localization Properties
	Pagination (Advanced)	Pagination control (PDF only)
	Presentatio n (Advanced)	Presentation (Alignment, Drop Shadow, etc.)
	Printing (Advanced)	Printing (PDF only. Printing properties)
	Section Header	Header
	Section Help	Help
	Section Number	Numbering
	Wizard	Wizard (does this section belong to the form's wizard navigation scheme)
Block	General	General (Labels)
	Widget (Advanced)	Border Presentation (Borders, background colors and gradient)
	Custom	HTML Custom Styling (HTML forms only. Background image, its repetition, custom

Structural Element Type	LH Menu Item	Panels (an indication only)
	Styling (Advanced)	HTML classes)
	Layout	Layout Manager Layout Constraints Sizing (i.e. the layout fill: "Horizontal" or "None")
	Localizatio n (Advanced)	Localization Properties
	Pagination (Advanced)	Pagination control (PDF only)
	Presentatio n (Advanced)	Presentation (Alignment, Drop Shadow, etc.)
	Printing (Advanced)	Printing (PDF only. Printing Properties)
Simple Widget	Caption	Caption Content Caption Presentation
	Field (Advanced)	Field Border Field Presentation (These give the border, background and so forth of the area into which users type their data)
	Widget (Advanced)	Widget Border Widget Presentation (The border, background and so forth for the widget)
	Accessibilit y	Accessibility (Tool tip, custom screen reader text, precedence and so forth)
	Custom Styling (Advanced)	HTML Custom Styling (Custom HTML Classes and whether Boostrap styling applies) PDF Comb Styling
	Data	Data (various properties, many of them Advanced Mode only, including display format, etc.)
	Layout	Layout Constraints Sizing
	Localizatio n (Advanced)	Localization
	Printing (Advanced)	Printing (PDF only)

The Other Tabs

In The Properties Tab , we gave 4 types of structural elements in the panel. There are actually 5 main types:
The form itself (at the top of the structural panel's tree)
Read only elements (mainly found in Nuts & Bolts elements)

Sections Blocks

Simple Widgets
The New Dialog lists the range of other tabs. The current topic briefly deals with two tabs associated with all the elements:
Overview
Styling
Neither has a LH Menu.

Tab	Panels	Editabl e?	Properties
Overview	Basic Information	No	Field Name Display Name Field UID (the unique identifier of the element) Source (Whether created directly in the form or not)

	Type Information	No	Type Name Display Name Source
	Restrictions	No	Can have children? ("yes" or "no") Can be moved? Can be deleted? Can be embeded?
Styling (Not available in the form element at the top of the tree) (see Stylesheets) **Note**: None of these properties are covered in Properties -> Custom Styling (HTML only)	Stylesets (qv)	No	Lists any stylesets associated with the element
	Typed Styles	No	Lists stylesheets assigned to the element according to its type.
	Named Styles	Yes	Can explicitly add or remove stylesheets to the element and set the order of precedence.

Field Layout Control

This topic covers the mechanics of layout. See here , for a discussion on the aesthetics and design of form layout from. Also see another discussion on Usabil ity and Accessibility .

Problems with explicit position layouts

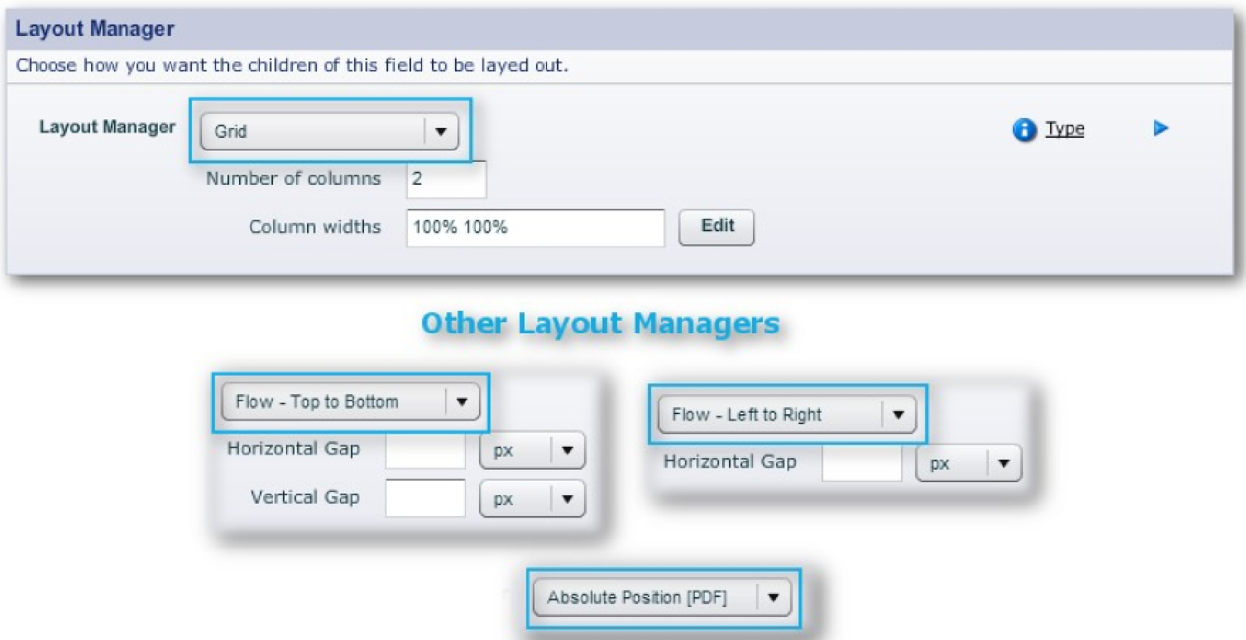
When designers build a form using simple tools, they tend to drag a widget into a specific position on the form
— known as absolute or x-y positioning — then re-position and re-size it using their mouse. This works for simple forms only, not when dealing with multiple device types, nor with maintaining existing forms. Explicit positioning:
Cannot ensure that forms flow correctly on different technologies, for example on the desktop, tablets or smartphones.
Cannot ensure all fields have the same styling and sizing. Cannot ensure that all fields line up nicely.
Cannot ensure proper placement on the high resolution devices. Doing this manually is very time consuming and the results are inconsistent.
Cannot save designers from filling in parameters manually into dialogs. For example, "all standard text fields are 1.5in wide, a caption of 1.2in, and placed 0.7 in in from the left margin". The designer then needs to know these magic numbers, and apply them consistently and enter them for every single field on the form.
Makes a big job out of inserting a new field vertically between two existing fields.
Cannot automate changing all the fields on the form to have, for example, all the field captions left or top- aligned, or changing all the fonts or background colors.
Cannot automate changing the width of the form, or its margins and so forth. Instead, every field has to be repositioned.

Layout Managers and Hints

For these reasons, Composer automates the detailed layout of fields. All you do is specify the order of the fields and, optionally, a few extra "hints" and let Composer do the rest.
Hints may be such things as:
Keep this field on the same line as the previous one
Make this field as wide as possible (within the bounds of the space that it has available on the page) Spread these two fields across the page, in the ratio 40% to 60%.
Lay out these 12 fields equally spaced in a 4 column grid

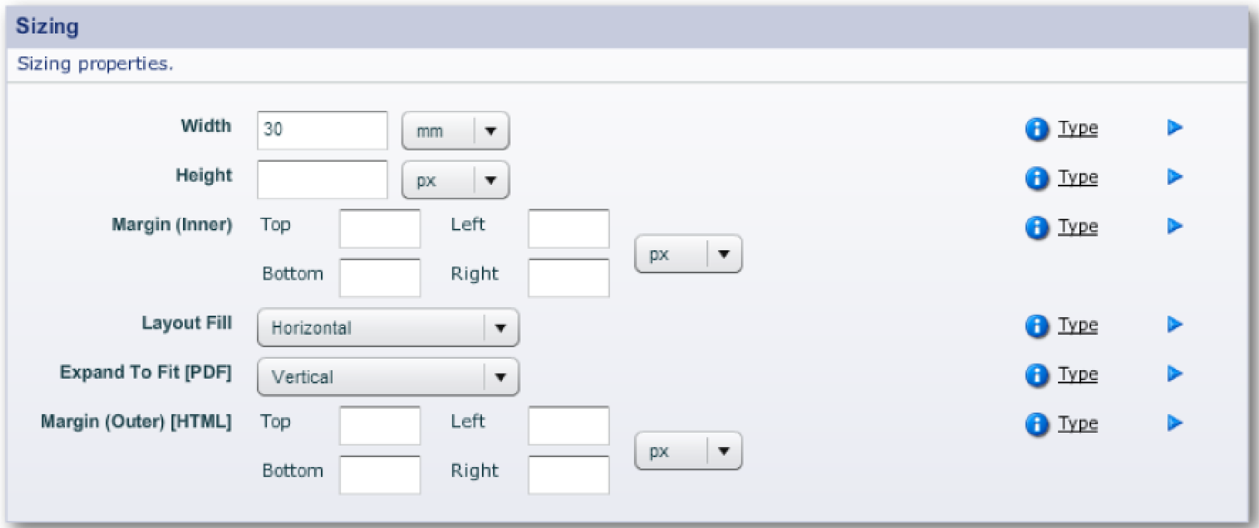
The Layout Panels

The default settings for the Composer layout managers are set in the template for the form. The following documents how to tweak the layout of elements on the form when the default positioning of an element needs it.
You make layout adjustment through 3 panels in Edit Properties. They are:
The **Layout Manager** panel

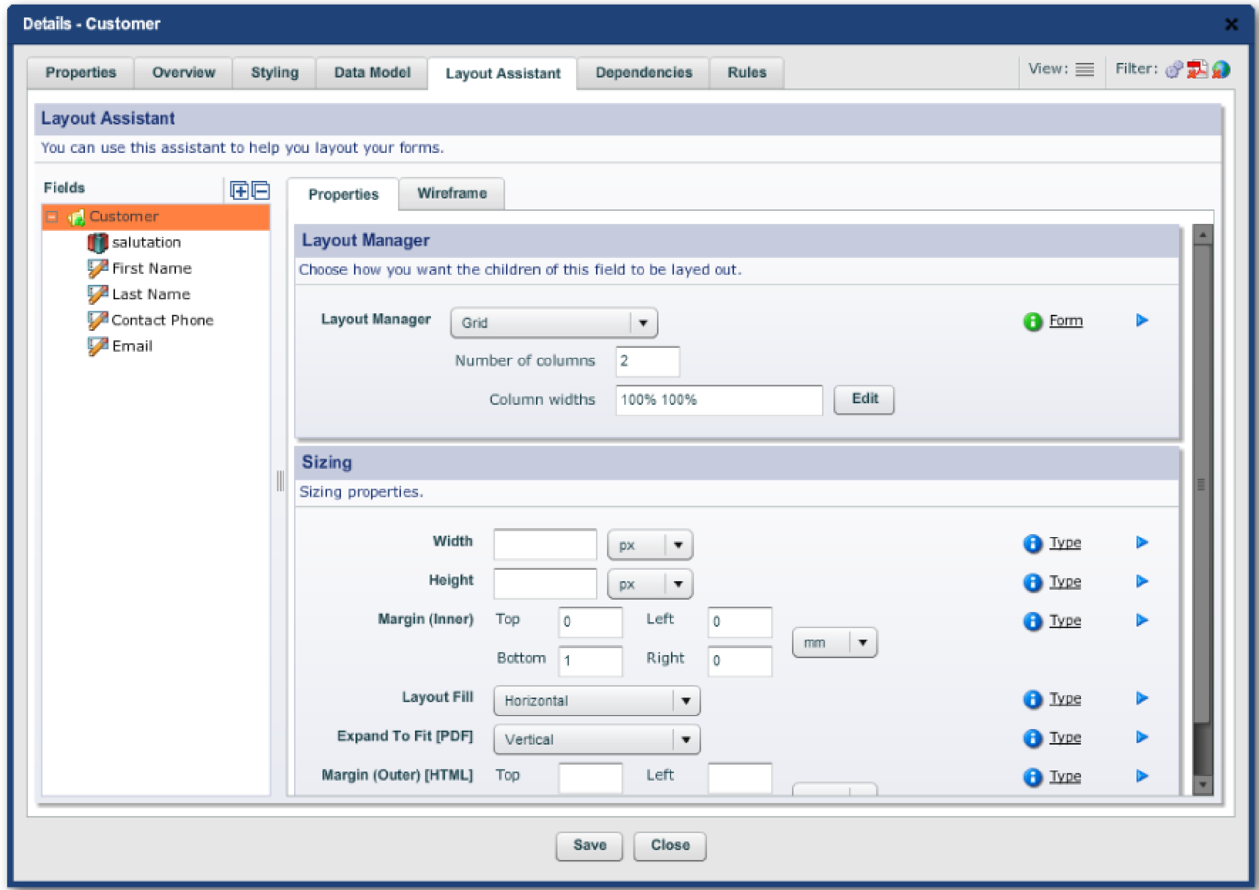


Layout Constraints

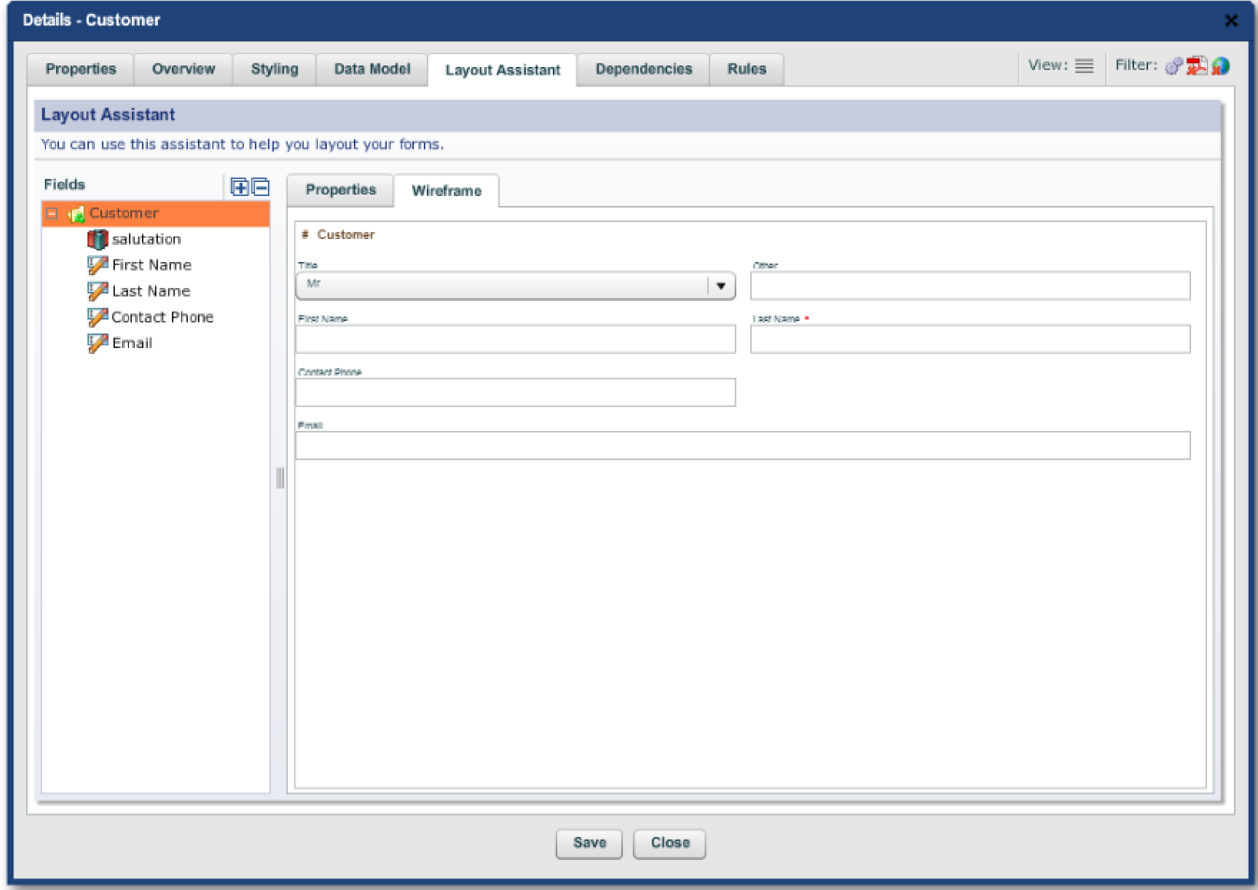
Sizing



You access these panels in "Edit Properties" through "Properties -> Layout" or in the "Layout Assistant" tab. (That tab, though, does not contain the "Sizing" panel.)



Note the mini Structure Panel (marked "Fields") in the LH Menu of the tab. And, there is also a mini wireframe available on the RH side of the tab in which you can test the effect of tweaking the values in the "Layout Manager" and "Sizing" panels.



The Effect of the Different Layout Managers

Flow — Top to Bottom

A lot of forms are very simple - they just contain a series of fields going down the page. This is also the best layout for the narrower screens on mobile devices.
To achieved this type of layout, you can use the Top to Bottom flow. Fields in a Vertical Flow simply go below each other down the page. The Vertical Flow is always the default manager on all sections and blocks.
There are a few options or "layout hints" that you can specify with the Vertical Flow layout manager to achieve higher levels of usability. These include:

Property	Description
Keep on Same Line (Layout Constraints Panel)	If "Keep on Same Line" is checked, this field will be kept on the same line as the previous field. You can set this on as many fields as will fit on the line. This technique is mostly used to group related field together, such as First Name and Family Name, or State and Zip Code.
Layout Fill (Sizing Panel)	If Layout Fill is set to "Horizontal", then this particular field will expand to fit all the available space on the line. This can be used to give your users as much space as possible to fill out longer fields, without having to give your fields explicit widths. If you have multiple fields on the same line with Layout Fill set as "Horizontal", each of them will then share the extra available space equally.

The Top to Bottom flow does not change the width of any of your fields (unless you select Layout Fill). Your field width will be:
The "natural" width of the field as defined in your organization stylesheet; or
The style-set that you associated with this field when you dragged it into the form . Typical style-sets include widths like "Tiny", "Small", "Medium" and "Large", but this will vary depending on how your organization's stylesheets have been set up. If you want to change the style set after you've created the field, you can change it in "Styling -> Stylesets -> Field Size".
If you select Layout Fill, the layout manager will try to use as much of the line's space as possible and take into account the size of the screen or page, margins, borders and gutter areas, space allocated to menus, and the other fields on the same line.
Note: If you specify too many fields on the same line, this can create layout problems, particularly on smaller screens. The behavior of a line with too many fields will depend on the Responsive Layout rules.

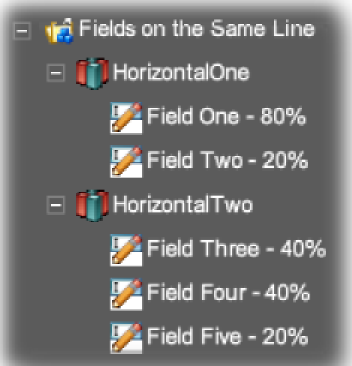
Flow — Left to Right

The Top to Bottom Flow is suitable for many simple types of forms; you often need, however, more fine-grained control over how fields spread out across the page. This is where the Left to Right Flow layout manager, comes in.
Horizontal Layout allows you to lay out several fields across the page with very fine-grained control over the size of each field by specifying the percent width of each field.
Note: While we generally talk about percentages, those percentages need not to add up to 100: Composer will adjust accordingly.
The beauty of using percentages is: should you need to add another field onto the same line, move a field, re- size a field, or switch from portrait to landscape, all the other fields will automatically adjust to accommodate the changes. Form maintenance then becomes easy. The fields will also retain their proportions on the range of mobile devices.
Left to Right Flows are always used in conjunction with Top to Bottom. The Top to Bottom controls each line of fields; and Left to Right within each line.

Multiple fields per line

The simplest way to get multiple fields on the same line is to use a Top to Bottom Flow and the "Keep on Same Line" property ("Properties -> Layout -> Layout Constraints"). This, however, gives you little control over the sizing of the fields.
For more control, use a Left to Right flow inside a Top to Bottom to finely control the widths of each of the fields in the single line. Using even multiples of a base size will result in fields on subsequent lines being lined up to each other. This is shown in the screenshot below.

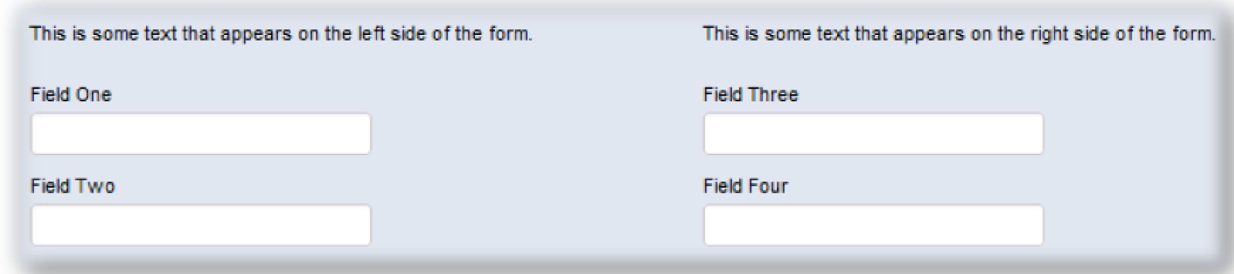
Horizontal Layout



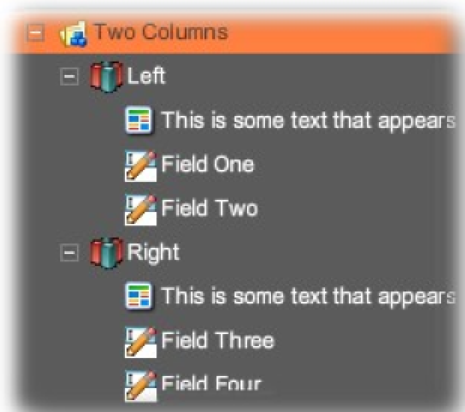
Horizontal Layout in the Hierarchy

Two-column forms

You can create a two column layout by using two blocks laid out within a horizontal flow layout, with fields inside them laid out horizontally.
This is shown in the screen-shot below:



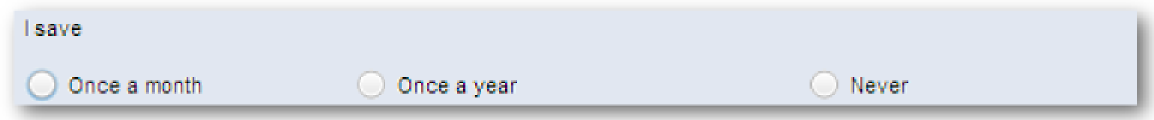
Two Column Layout



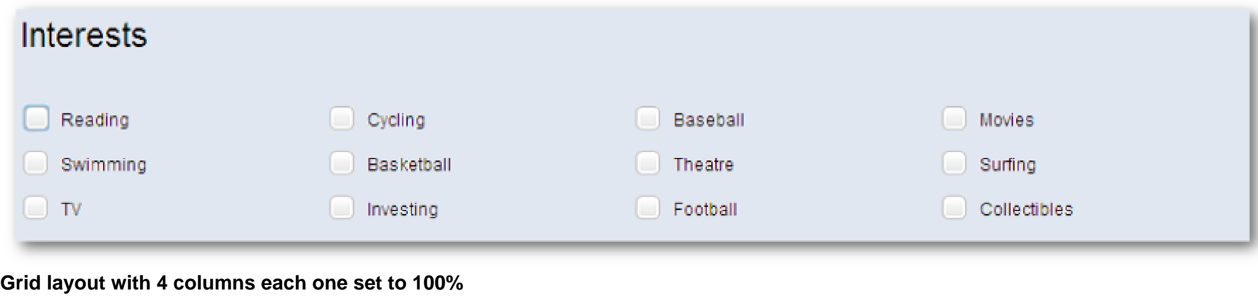
Two Column Layout in the Hierarchy
Note: You can use the Margin property to provide extra spacing between the two columns, or use another smaller block between the two.

Grid Layout

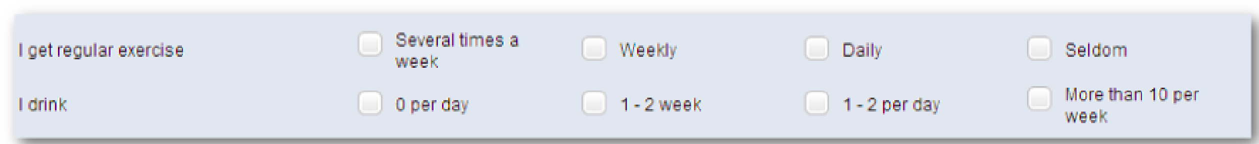
Often, you have to lay out a large number of fields and, for an elegant layout, all these have to align correctly. The Grid layout manager, which is specified on blocks or sections, can achieve this.
By default, each column in the grid has equal size, but you can also specify the width of each column by defining a percentage of the space available, or a fixed size in mm or inches. For example, in a 3 column grid, you may specify the column widths as "100% 60mm 100%". As in the example, these percentages need not add up to 100.



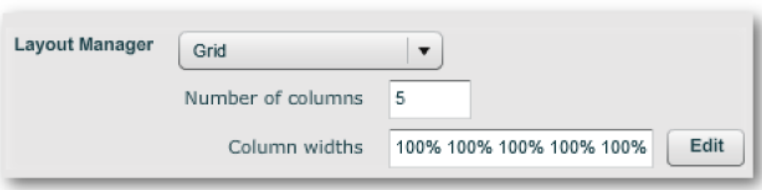
Grid layout with 3 radio buttons respectively 100%, 60mm and 100%



Grid layout with 4 columns each one set to 100%

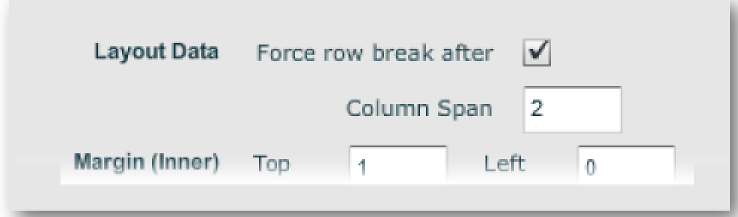


Grid layout having a text field set to 150% and each checkbox set to 100%
Grids can also be used as part of a hybrid layout. For example, a grid could contain one or more blocks which in turn could contain widgets, or grids could be arranged within a vertical or horizontal flow layout.
Each of the objects added inside the grid can specify hints that provide additional control over how they are laid out.
Go to next line Span



Property editor for the parent block

In the example below, I begin by using as I have 5 checkboxes.
Because 'More than once a week' is such as long caption, I select the 'Force a row break after' that checkbox and make it "span 2 columns".

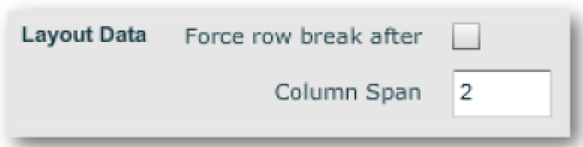


Tweaking the element with "Edit Properties -> Layout -> Layout Data"

Result:

Grid layout where 3rd column spans 2 columns and the 4th column is forced onto the next line
Tip: Grid layout is particularly useful for laying out multiple checkboxes and radio buttons. Even if your radio buttons all appear on one line, it's still advisable to place them in a grid layout. Decide on a standard grid size for all radio buttons, and always use that grid size for all radio button groups, regardless of how many radio buttons you have. If you do this, all your radio buttons in your entire form will all line up nicely.
Tip: If you have a radio button with a longer than usual caption, simply set the span to 2 so that it uses up two grid cells.

Grid layout where each of 5 columns set to 100% (Note: the caption on the 4th column)



Property editor for 'More than 2 times a month'

Grid layout where 4th column spans 2 columns to prevent caption wrapping

Fillers, Springs and Page Breaks

There are a few special objects in Composer that deal specifically with laying forms out.

Object	Description
Filler	A filler is a simple invisible object that simply takes up space, creating "white-space" in your form. White-space is generally considered a good thing by form designers, since it creates visual separation and grouping between objects, and makes a form easier to read and complete. You can use a filler to separate two other objects either vertically or horizontally. For example, you can put a filler after a heading to create some space between the heading and the first field. Fillers can sometimes be useful, but they are difficult to style - if you want to be able to control the look and feel of your form by changing the stylesheets, you're generally better off using properties on the individual objects that can be more easily styled, such as margins.
Spring	A spring is exactly what it sounds — it's an object that tries to expand as big as it can. For example, if you want a layout with one field on the left of the page, and another field on the right of the page, simply use two fields in a horizontal layout and place a spring between them. Under the covers, a spring is really just a simple invisible object with its width set to 100%. Springs only work with other objects on a single line.
Page Break	Page Breaks only apply to page-oriented formats such as PDF. By default, Composer builds forms that just flow down the page. Fields will automatically "roll over" to the next page when there is not enough room on the current page. Usually, this works fine, but occasionally, you may want to explicitly start on a new page. In order to do this, simply drag a Page Break object into the tree, and your form will immediately skip to the next page.

There are three other properties in Composer that affect page breaks in PDF forms, in the "Properties -> Pagination -> Pagination Control" panel:

Property	Description
Allow Page Breaks	Containers such as blocks and sections allow you to specify whether they should allow page breaks within themselves or not. If you turn "Allow Page Breaks" off, then the entire section will always stay on the same page - if even one field of the section would flow over to the next page, the entire section will be moved to the following page. If you turn "Allow Page Breaks" on, then if one of the fields in the section exceeds the current page, just that field and the ones that follow it will move to the next page. Most stylesheets have Allow Page Breaks turned on for two reasons. Firstly, disallowing page breaks can create large blocks of white space at the end of pages, confusing the user. Secondly, in dynamic forms (forms that grow and shrink), making a very simple change on one page could cause a following section to exceed the current page, and when the entire section is moved to the next page, it will seem to "disappear". Users find this very confusing. You should also be very careful to ensure that any section that disallows page breaks is no bigger than a single page, otherwise Reader will not know how to lay it out.

Force Page Break	This property is associated with a block, and will force this section or block to start on a new page. This is equivalent to inserting an explicit Page Break object into your form.
Block Keep With Policy	Choice between: "None", "Previous" or "Next". In other words, do you want to keep this block on the same PDF with the next block or the previous block, or or you don't care either way.

Field Styling

The form's template sets the style defaults for its fields, and so any global changes should be made to the template itself . That is an advanced topic, not covered here.
Local changes can affect only a single field or whole containers or sections. There are 2 types of these changes:
Changes through stylesheets Manual overrides to stylesheets
Manual overrides are easy to understand but are time consuming and as clumsy as explicit position layouts.

Using Stylesheets

In the Workspace Hierarchy , Stylesheets live in Libraries . Libraries are associated with Projects or the higher levels of the hierarchy. See Library Advanced Features .
You can also create *ad hoc* stylesheets (that is, those not contained in a library) at the Organization level through the Stylesheet tab. See Stylesheets for more.
To do either, you need to have the necessary access levels.
Given that you now have suitably modified stylesheets, you can make use of these through the "Styling" tab of "Edit Properties" of the element. This tab is available for all elements except the form element at the top of the Structure Panel tree.
The tabs has 3 panels:

Stylesets

where designers can choose a style from a set of them as defined in the stylesheet, The most commonest example is "Field Size", which often appears in the wizard when a widget is dropped onto a form's Structure Panel.

Typed Styles

a read-only field that displays the style types associated with the field. These types are defined and assigned in the stylesheet. See Stylesheets .

Named Styles.

These 2 kinds of styles are explained in detail later, For now, suffice it to say that:
typed styles are based on field hierarchy or field function, Their allocation is automatic and based on the field's place in the form's structure
Named styles are allocated manually, and their functions are more local, such as for setting a background color or making a field transparent, and so on.

Details - Other

Properties | Overview | Styling | Data Model | Dependencies | Rules

View: Filter:

Stylesets

A styleset is a predefined group of related named styles. While a field can have multiple stylsets, only one named style from each styleset should be selected.

Field Size [Not Set]

Typed Styles

Typed styles are applied based on the type hiearchy of the field. For this field typed styles will be applied in the order they appear in the list below.

Type-Bindable
Type-BindingLeaf
Type-DataSupport
Type-VisibilitySupport
Type-EditableSupport
Type-CalculationSupport
Type-ValidationSupport
Type-LayoutParticipant
Type-ResponsiveSupport
Type-Canvas

Named Styles

A named style is a style that is applied only to fields that it has been explicitly added to. Named styles are applied in the order they appear in the list but after any typed styles.

Add... | Remove | Move Up | Move Down

Select a Named Style to Add

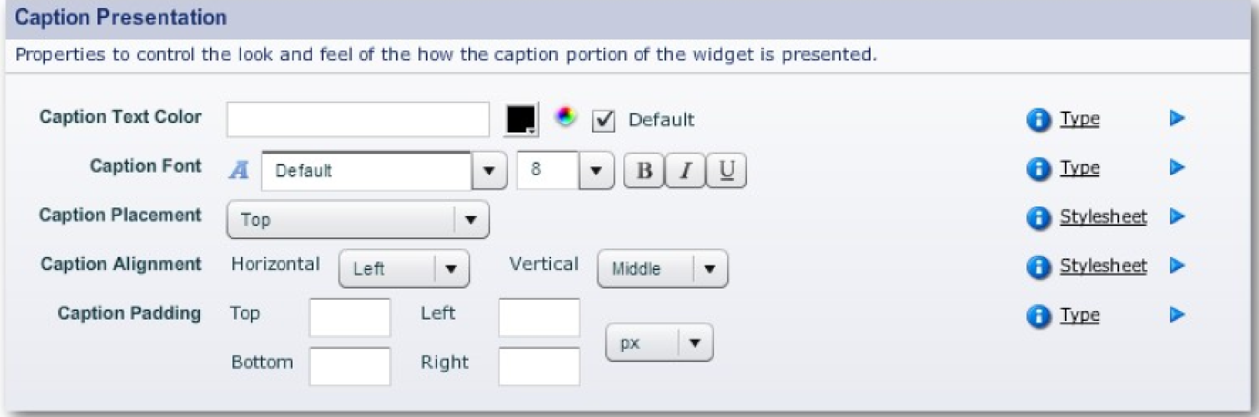
Filter: Only styles for current field type — Hide Internal Styles

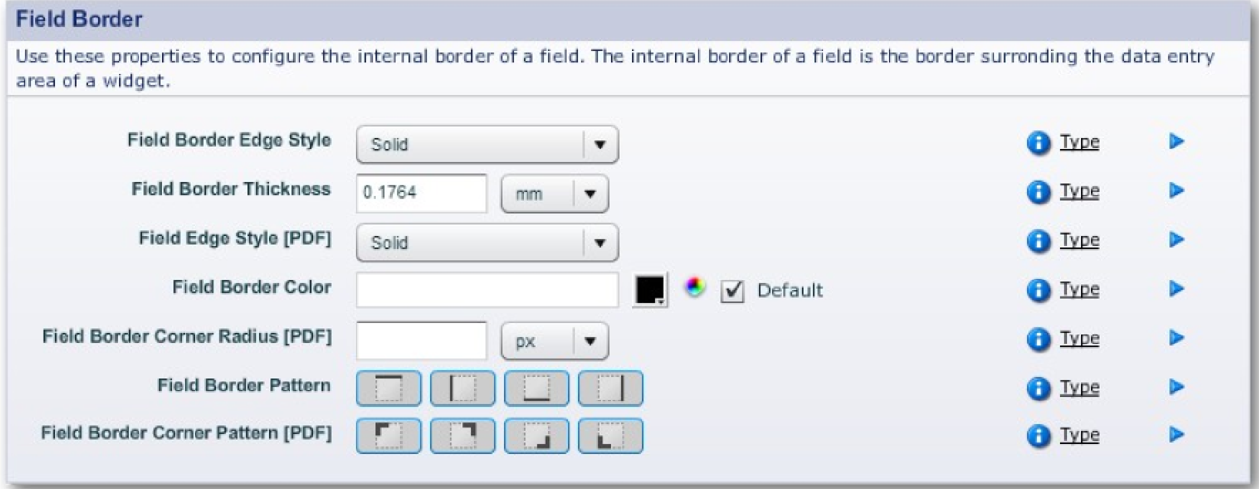
Named Style	Description	Targeted Types
style.transparent.field	Sets the background color and b	Field
styleset.size.default	Resets a field to use its default s	Field-TextField,Field-DropDownL
styleset.size.fill	Sets a field to fill its parent block	Field-TextField,Field-DropDownL
styleset.size.large	Sets a field to use large caption	Field-TextField,Field-DropDownL
styleset.size.medium	Sets a field to use medium capti	Field-TextField,Field-DropDownL
styleset.size.small	Sets a field to use small caption	Field-TextField,Field-DropDownL
styleset.size.tiny	Sets a field to use tiny caption a	Field-TextField,Field-DropDownL

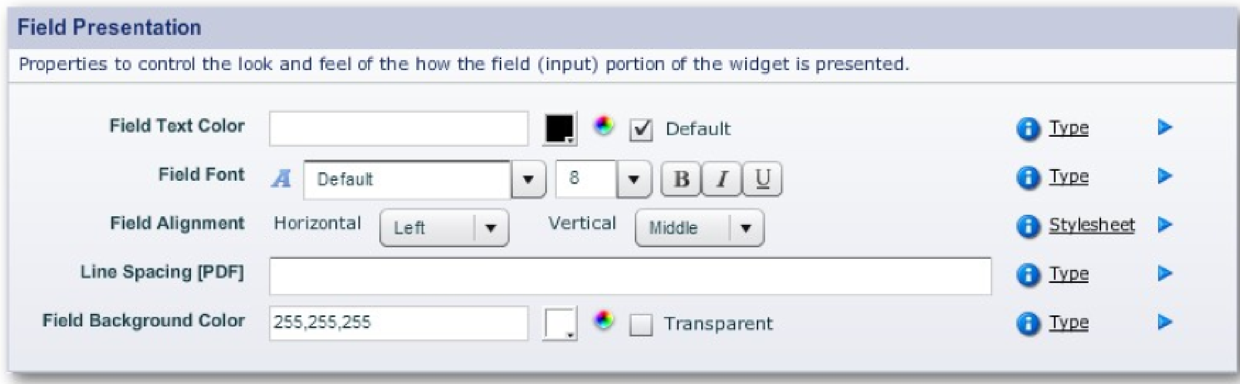
Select | Cancel

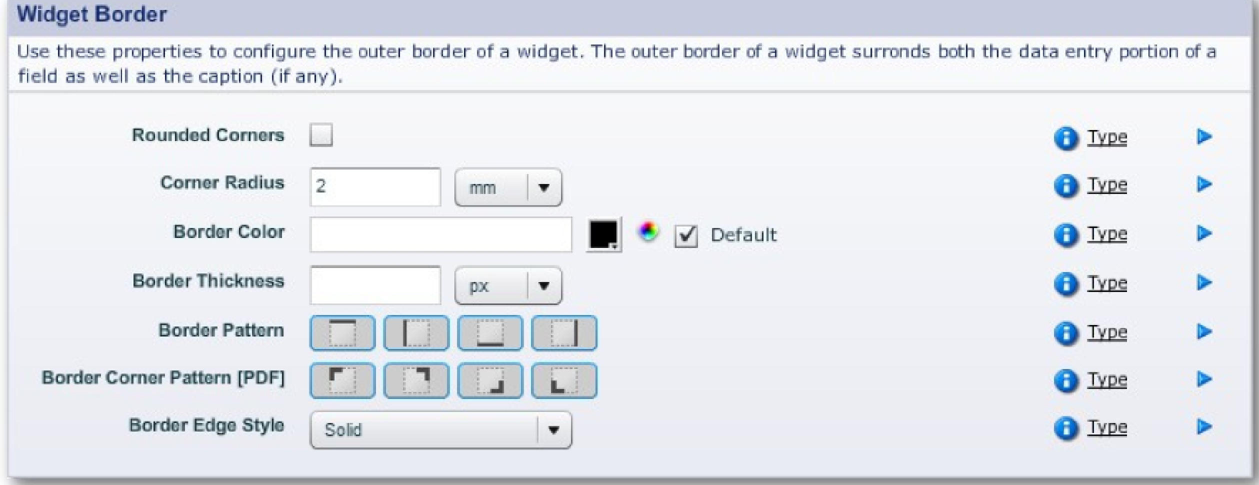
Manual Overrides

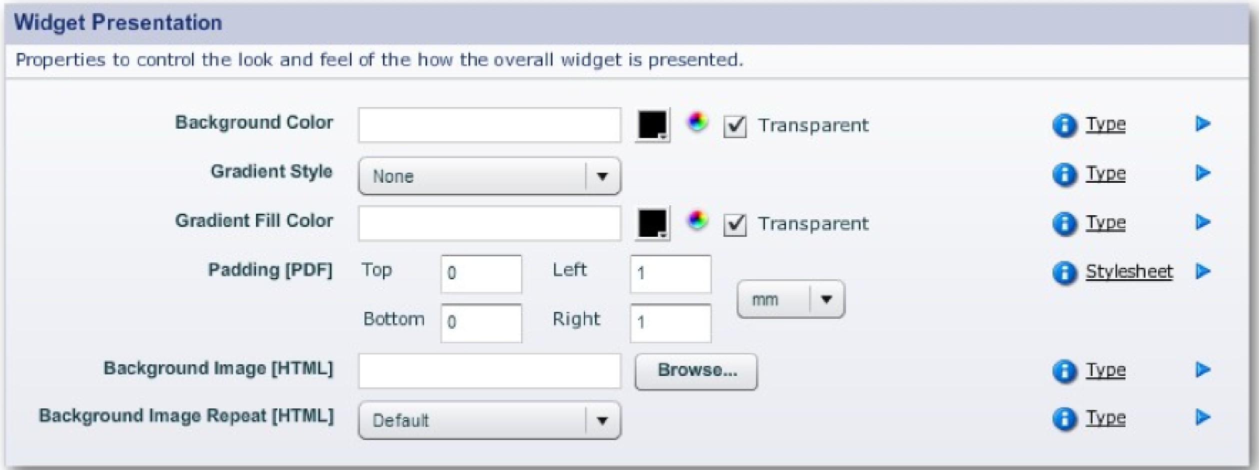
These should be one-off overrides. (If you find that you are continually applying manuals throughout the form, you should be changing the template instead.)
The relevant panels are the "Presentation" and "Border" panels:
"Properties -> Caption -> Caption Presentation" "Properties -> Field -> Field Border" "Properties -> Field -> Field Presentation" "Properties -> Widget -> Widget Border" "Properties -> Widget -> Widget Presentation"




Field Border
Use these properties to configure the internal border of a field. The internal border of a field is the border surronding the data entry area of a widget.
Field Border Edge Style
Solid
Field Border Thickness
0.1764
mm
Field Edge Style [PDF]
Solid
Field Border Color
Default
Field Border Corner Radius [PDF]
px
Field Border Pattern
Field Border Corner Pattern [PDF]
Type


Field Presentation
Properties to control the look and feel of the how the field (input) portion of the widget is presented.
Field Text Color
Default
Field Font
Default
8
Field Alignment
Horizontal
Left
Vertical
Middle
Line Spacing [PDF]
Field Background Color
255,255,255
Transparent
Type
Stylesheet


Widget Border
Use these properties to configure the outer border of a widget. The outer border of a widget surronds both the data entry portion of a field as well as the caption (if any).
Rounded Corners
Corner Radius
2
mm
Border Color
Default
Border Thickness
px
Border Pattern
Border Corner Pattern [PDF]
Border Edge Style
Solid
Type

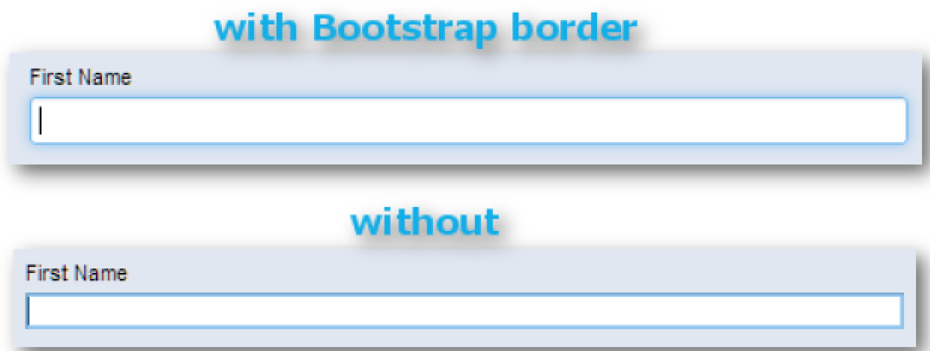


Custom Styling

This technique stands apart from the other styling adjustments.There is a panel for HTML-only settings, and another for PDF-only comb-fields.

2 sample "HTML Custom Styling" panels

Custom Classes means CSS Classes from a CSS stylesheet (a quite different entity form a Composer stylesheet) that is not under Composer's environment or control. One scenario is: the Custom Classes belong to a corporate CSS stylesheet. If so, adding these CSS classes via this panel can result in issues at rendering time, and are not an ideal solution.
"Bootstrap border styling" refers to Twitter's Bootstrap, a popular framework for implementing responsive layout behavior. The effect of having this border styling turned on is rather subtle:



A selected text field in preview

The Background Image parameters use a bitmap image file, .png, .jpg and so on, as "wallpaper" inside the object. The "Repeat" dropdown menu can be set to "Default", "Horizontal", "Vertical", "Both" or "None".

Field Data

Overview

A **writable** field's data is dealt with in 2 ways in Composer:

Data properties

which is about the data contents of the field, but considered in isolation from the rest of the form, with the exception of radio buttons that form part of the one group (as they are, after all, just the one data object with several form fields working together)

Data Model

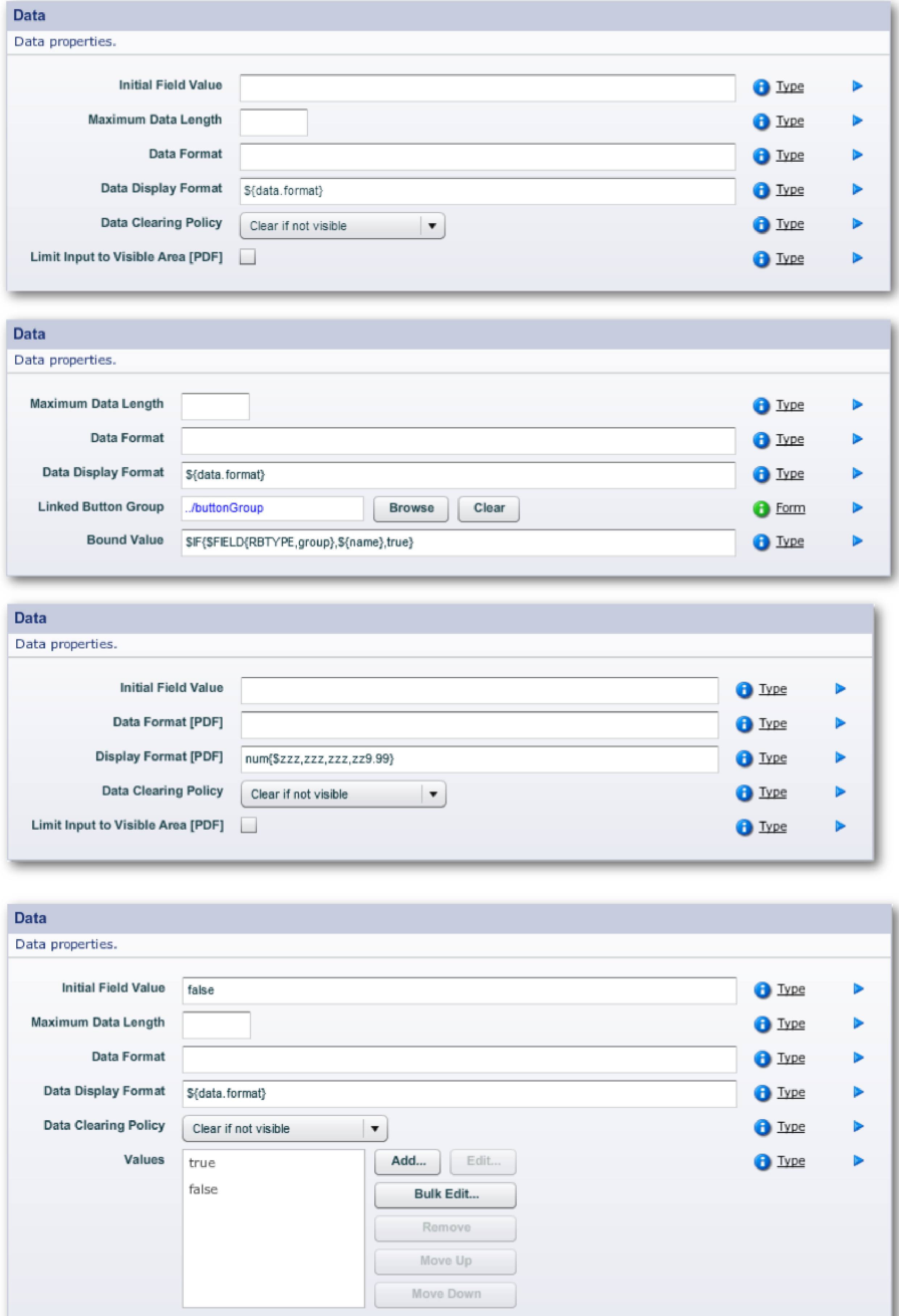
which is how the relationship of the field to the data structures of the form's outputs

Data Properties

These are contained in "Edit Properties" in "Properties -> Data". There are a number of panels, depending on the type of field:
Data (always present in writable fields)
Dropdown Data (in addition to the above for a dropdown menu field) Decimal Options (when the field is a decimal number)

Data Panel

The properties shown in both the Advanced and Standard Modes of the panel vary with the nature of the field. Here are some typical examples:



Some brief notes on the panel's fields (Advanced Mode):

Initial Field Value

Use this as an *ad hoc* way of repopulating the field. If you need to repopulate a number of fields from a database lookup or as a general configuration of the form, see here for an example of how this is done . **Maximum Data Length**
This prevents the user from entering more characters into the field. If you want to do more than merely having the field stop working once the character limit is reached — such as warnings, in-line messages and other helpful cues to asssist users to get to submission — please see Validation .

Data Format

This is the **raw data** sent from the form and as stored in Transaction Manager. This property is often specified for PDF (in XFA data description specification). The "${data.format}" is explained in Scripting and Dependencies .

Data Display Format

This can be different from the raw format, especially so with dates. See Data Types .

Data Clearing Policy

An important factor in Privacy .

Values

Appears with fields that have several states, such as Boolean fields (see Field Types). These states can be named and given an order of precedence. The "Blulk Edit" button enables you to import a list of state names from a text file.

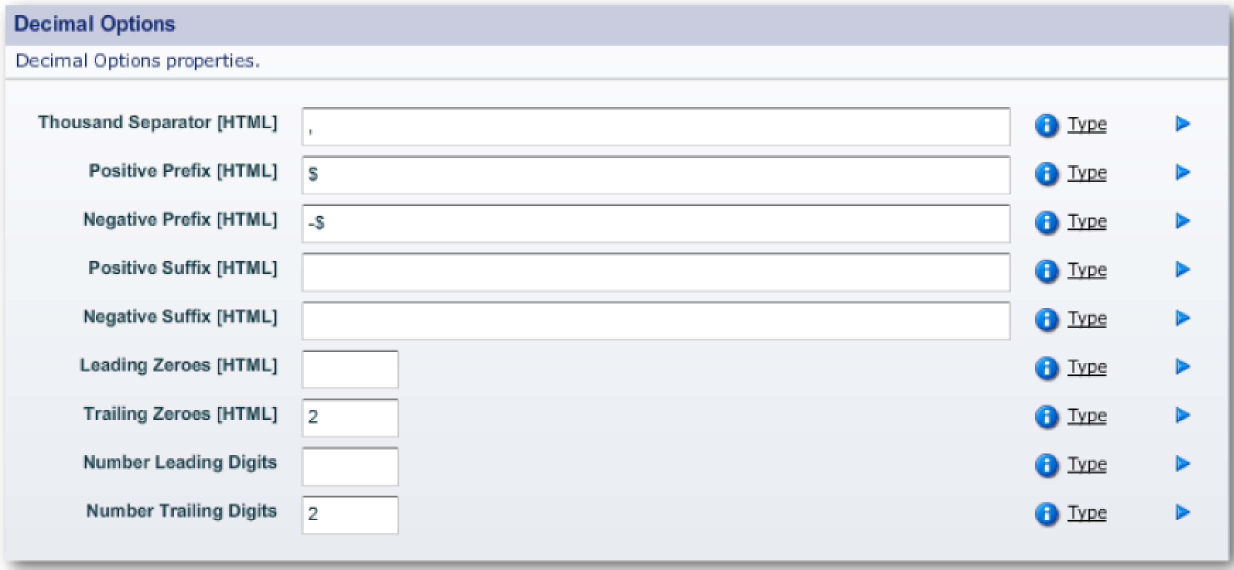
Dropdown Data Panel

You use this panel to edit a dropdown menu field's values and displayed values. The displayed values appear in the menu on the form. For example, the form could, say, display "Northern Territories" in the dropdown; but when users select that value, the form passes the value "NT" to the TM server. The "${values}" entry means that all the values equal the displayed values. See Scripting and Dependencies .

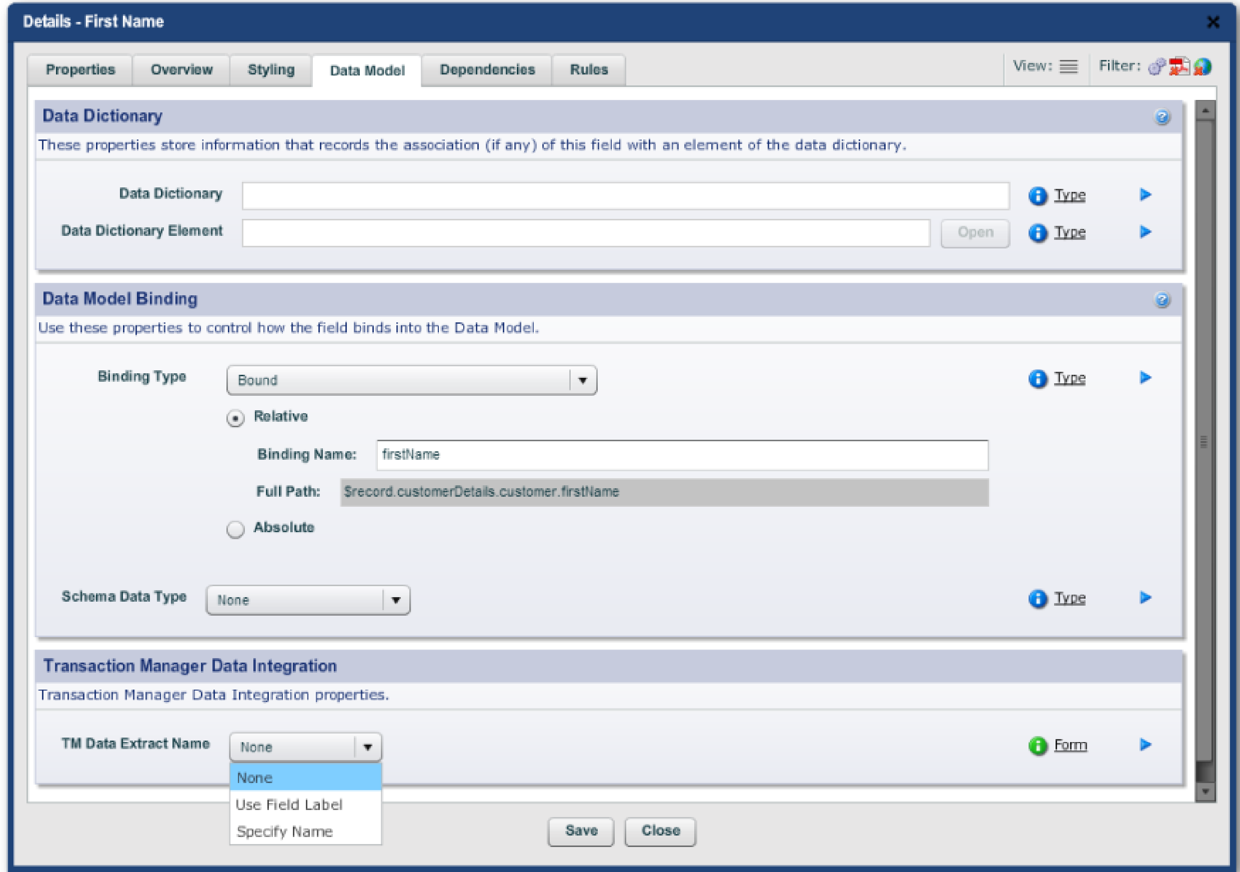
Dropdown Data Panel (Advanced Mode)

Decimal Options Panel

This panel appears for writable numeric fields and are self-explanatory. In the example below, the field is a currency field and so the positive and negative prefixes have a dollar sign.
The two fields for leading and trailing digits have two properties each for HTML forms and unspecified form types. The leading and trailing values should be the same for both form types.



The Data Model Tab



Scripting

Overview of Scripting

A major advantage of using the Avoka Transact environment — rather than writing your forms from scratch in HTML, Javascript and other scripting languages and writing the server-side process — is that the bulk of the form's implementation is taken care of for you.
But this means that you still have enormous scope to specify how the form behaves and to inject scripts into the form. In this topic, we will discuss the general principals for scripting.
These are the ways to script in Composer:

- Where available there is a **Rule Editor**

This performs simple logic operations. This editor is rather limited.

- A point-and-click **Script Editor** window.

This is for Javascript scripts. The editor is far more capable, but requires a knowledge of Javascript.

- Writing a script yourself in a text editor

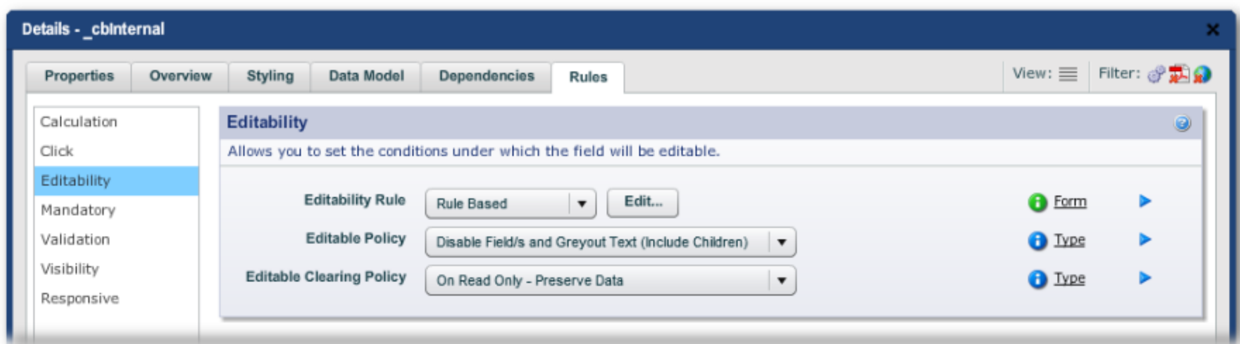
The covered in the advanced topic Scripting. These scripts can be complex.

The Composer Scripting Language

In either case, the language used is JavaScript, but with slightly modified function calls, documented here .

The Rules Tab

Scripts and Rules go in the Rules Tab. This has LH Menu items and depending, on the nature of the field, can have from none to 7. Here is an example of a Rules tab with a full LH Menu:



Within each panel there are several types of properties:
"Rule" properties "Policies" "Messages" "Actions"
and some checkboxes in, for example, "Mandatory" rules
and "Responsive Rulesets" which are exclusive to the Responsive Rules.

"Rule" Properties

This type of property has a dropdown menu:
Script Based (always available in all "Rules" properties) Rule Based (available in only some "Rules" properties
Always (available in "Editable Rules" and "Mandatory Rules") Never (available in "Editable Rules" and "Mandatory Rules") No Validation (only for "Validation Rules")
Regular Expression Pattern (only for "Validation Rules") for pasting in a regular expression for validation Fixed Length (only for "Validation Rules")
The "...based" items bring up an "Edit..." button, which takes you to the rules editor or script editor..

"Policies"

These are always dropdown menus and the choices are restricted to only those that appear in the menus.
The types of policy are:

Editable Policy

i.e. whether the field (or fields) and its children are writable or read-only
Determines what happens when the field is not editable according to the rule in the Edibility panel. Options:

-
    - Disable and gray out text
    - Disable
    - Gray out

Editable Clearing Policy

for more, see Privacy .

Visibility Policy

- When hidden exclude from layout

(i.e. the hidden fields disappear entirely)

- When hidden don't exclude from layout (i.e. leave space for the hidden fields)

- When hidden gray out **Initial Visibility (Policy)** {}* see below .
- Visible
- Invisible

(i.e. the invisible fields are not excluded from the layout)

- Hidden

(i.e. the invisible fields do not appear and their allocated space in the form does not appear as a blank area)

Validation Policy

(see Validation as there is rather more to this than just the following 2 options)

- Show error dialog
- Not show error dialog

Note: The intent Initial Visibility settings is meant to speed up the initial rendering of the form. If the form is taking a long time to render, visibility rules and visibility policies may be the issues. You can sidestep these by setting the Initial Visibility policy for the relevant fields so that the form will render first up without having to test these rules and policies.

Messages

These are text fields, whose contents are displayed when the rule type — usually "Mandatory" or "Validation" — fails (though you also have the option of displaying the message if the rule passes instead).

Actions

These are exclusive to "Click" LH Menu item. The Click panel only has the "Edit..." button for providing a script. The Rules editor is not available. The script will run only when the field (either a button or a link) is clicked on.

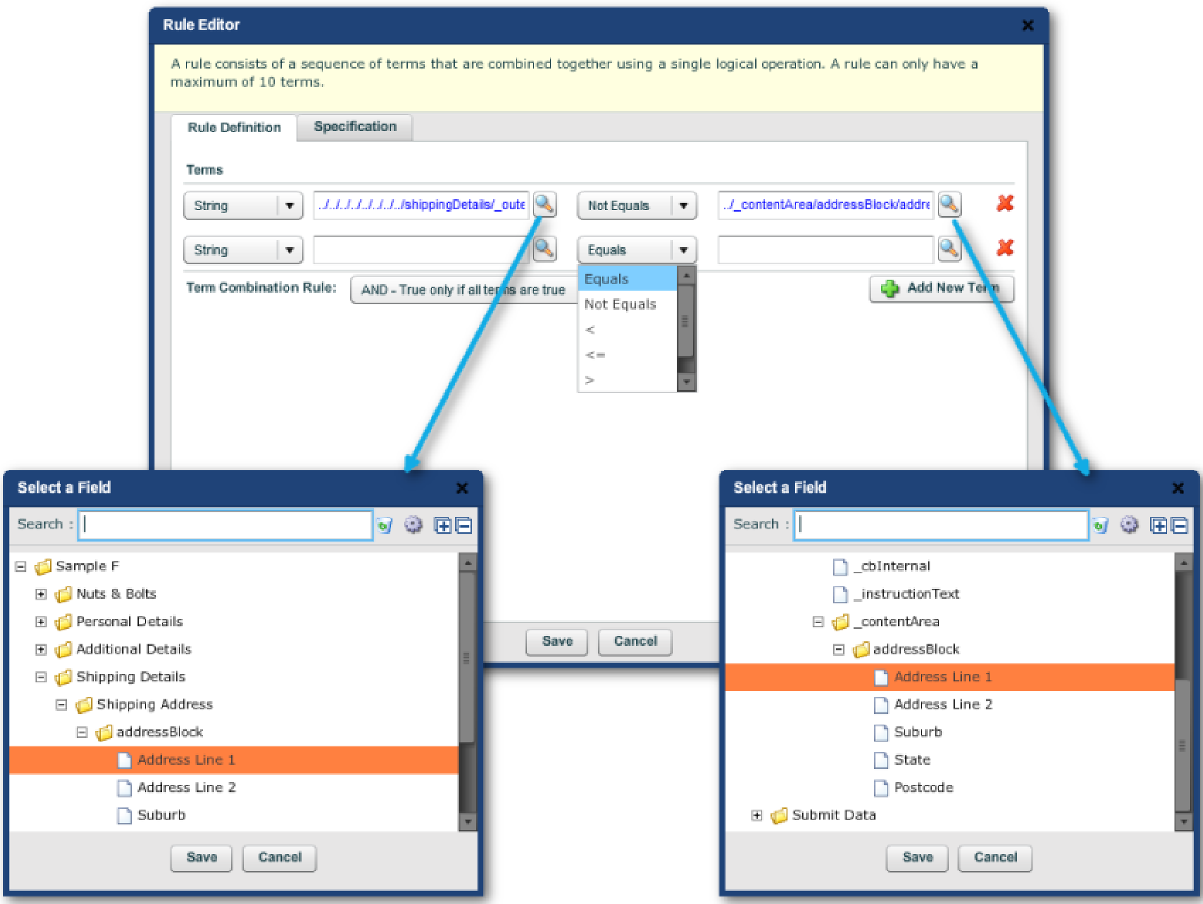
Checkboxes

The checkboxes in the Rules tab panels are for simple either-or options, such as whether the mandatory asterisk will display for mandatory fields.

Responsive Rulesets

These apply only to Responsive Layout settings for the field. See Responsive Layout for more on this.

The Rule Editor



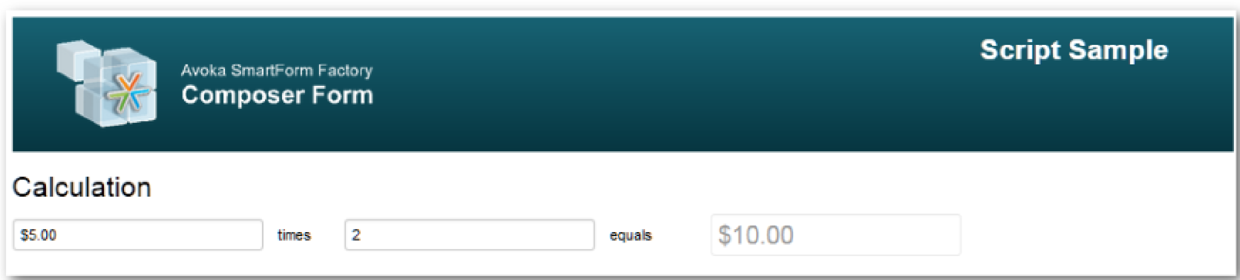
You build up a rule with the Rule Editor a "term" (or line in the editor) at a time with a maximum of 10 terms. Each term compares 2 fields on the form, which you select using the two magnifying icons, each of which brings up a "Select a field" dialog containing a mini Structure Panel, as shown.
On the left-most column of each term, there is a dropdown in which you select whether the two fields in the term are strings, numerical or Boolean. Between the two fields is the relationship between the two: "Equals", "Not Equals", "<", "<=", ">" or ">=".
Add more terms as required and then select the relationship between all the terms. You can only nominate one relationship for all the terms. They are: "AND", meaning that the rule is true only if all the terms are true "OR" where the rule is true only if all the terms are false.
The **Specification tab** is a text area where, if you
If you know something of Javascript, you can do far more with the Script Editor.

The Script Editor

Using this editor requires some knowledge of JavaScript. In this section of the guide, we will only be using very simple sample scripts. The script editor facilitates integration of your JavaScript code with the fields on the form.
The script editor can accomplish much more than the Rule Editor;
Calculations such as summing the values of fields together, or multiplications, averages and so on Write more complex scripts involving "if" statements, loops and so on
In fact, virtually anything the JavaScript is capable of.

Just to be clear, though the scripting language of Composer is JavaScript with some minor differences. One of these allows Composer's JavaScript to use the fields in the form as variables. This makes Composer's scripting capabilities very powerful indeed.
More ground on this will be covered in Scripting in the Advanced Topics. For now, we will be content to just demonstrate how you use the editor to integrate a script into the form.
To make the editor work, you need some working knowledge of JavaScript. We have already given an example of a total in the footer of a table. That example is a sum of repeating elements. The following is a simple multiplication of two non-repeating fields, just to illustrate how scripting works in Composer on a simple level.
Here is a very simple form where we take the first writable field "Amount1" (a dollar value currency field) and multiply it by the second writeable field "Amount2" and display it in "total" a read-only currency field to the right.
We made a few tweaks to "Amount2" and "total" and the "times" and "equals" texts to get the layout as shown below in Edit Properties for each of these fields:
For all, "Properties -> Layout -> Layout Constraints -> Keep on same line (checked)"
For "total", we made it read-only by "Rules -> Editability -> Editablity Rule -> Never Editable"
We could have saved ourselves the trouble of changing each widget by creating a containing block of the 3 widgets and the plain text objects "times" and "equals" and then setting the Layout Constraints.

So, to see the calculation in action, we need to preview the form. Here is what it looks like after inputting a couple of arbitrary values into Amount1 and Amount 2



The JavaScript in the example is only one line, which is the reason the line doesn't end in a semi-colon (";").

The Specification Tab

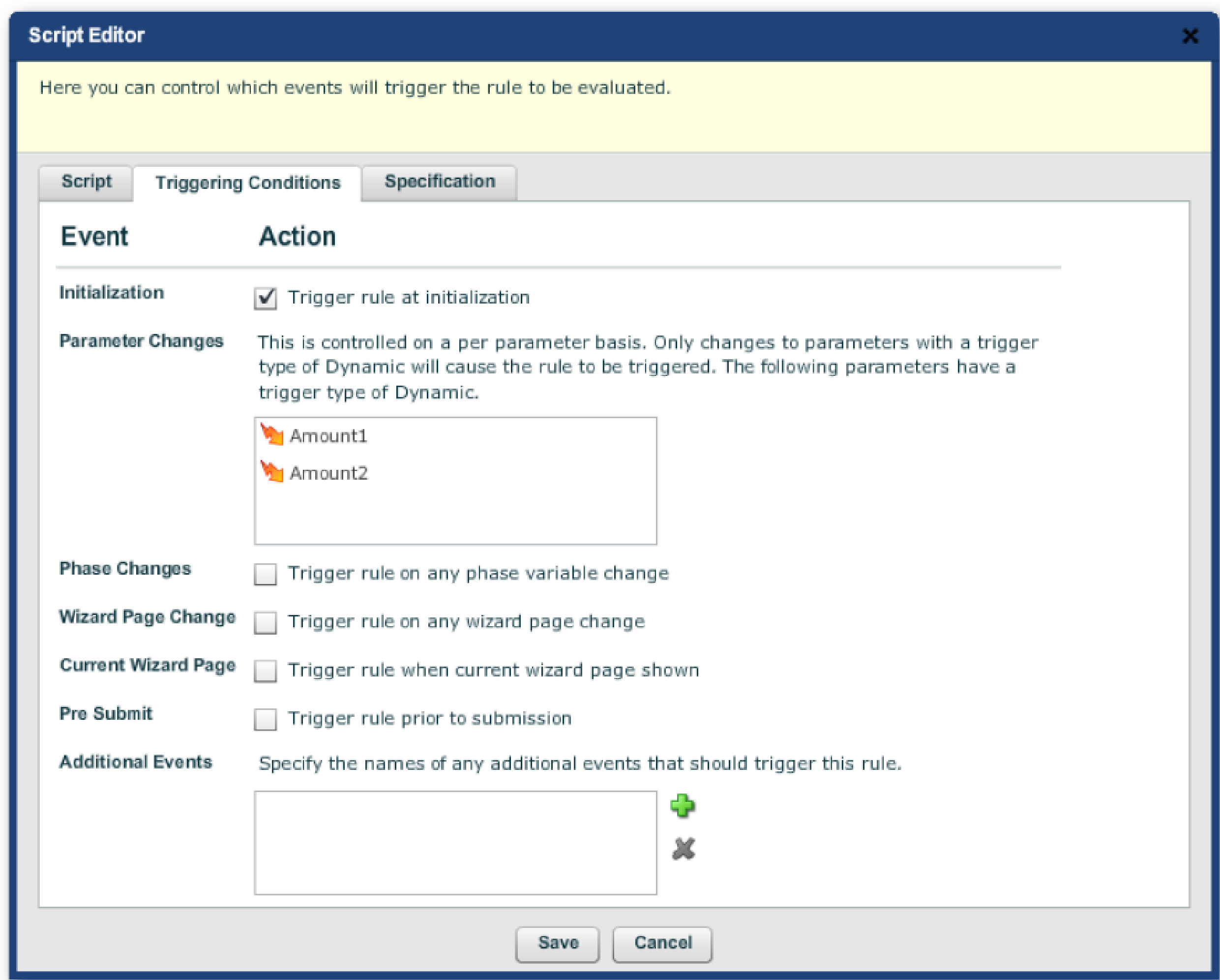
If want someone else on your team to do the scripting or work the Rules Editor, use the Specification tab to describe the behavior you require. That description will then attach to the field or block.

Dependencies

Overview

When a user loads a form, fills in its various sections, uploads any attachments and eventually submits it, the form has triggered many events, which Composer groups into "phases", covered in more detail in Business Rules and Advanced Scripting .
Some of these triggers are the dependencies between fields created by scripting. In the above simple example, the "total" field listens to "Amount1" and "Amount2" and when "total" sees that the other 2 now have values, it will calculate and display the value of the calculation. The user does not have to click on a special button to get the total, the form does not have to be reloaded, nor does the user's browser (in this case) have to send anything to the server in this phase. The calculation happens autonomously.
When a form is complex, though, having very many dependencies can create performance issues. Composer has many strategies to tune such forms. One of them is to tune the triggering in the form so that it does not grind to a halt every time the user changes a value in any of the fields. In Composer 4, for the first time, you can inspect triggers and dependencies on the form.
When you paste script into the text area of the Script Editor, remember that you still have to associate the fields on the form, using the same point-and-click method shown above and setting the field references.

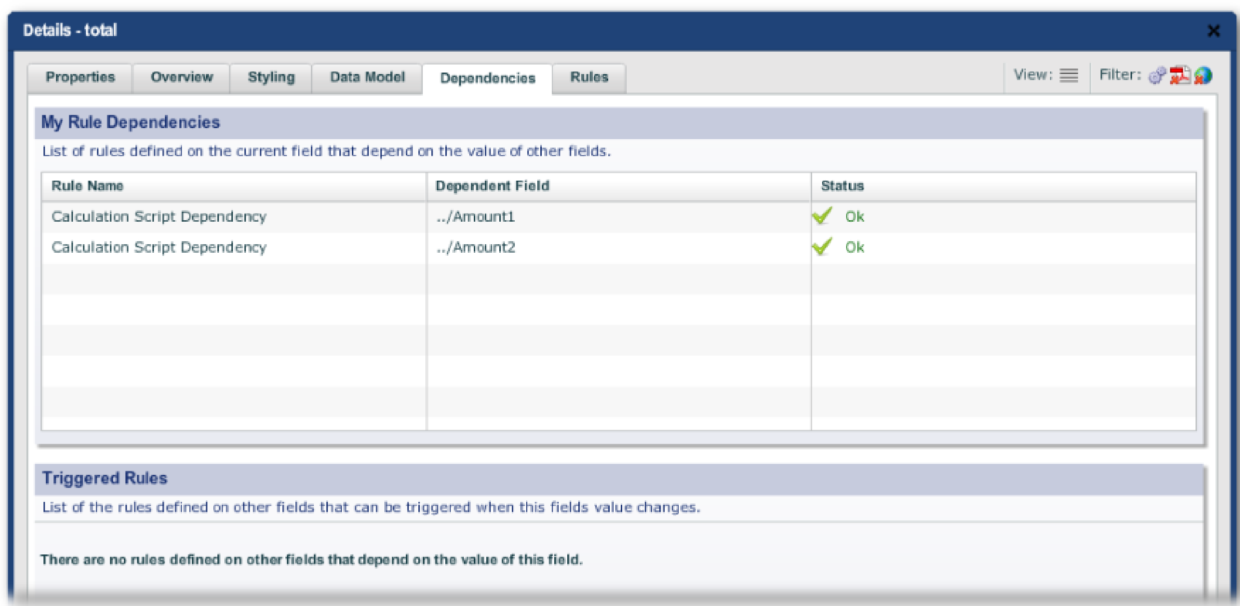
Inspecting Triggers



The triggering conditions of the above simple demo script are listed in the "Triggering Conditions" tab of the Script Editor. We will discuss the advanced settings of the radio buttons in the lower half of the tab later in Advanced Scripting .

Additional events will be covered in the Business Rules widget discussion.

Inspecting Dependencies



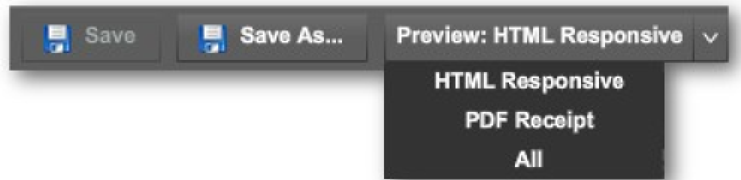
Use "Edit Properties -> Dependencies" to inspect the **dependencies** feeding into the script (in the "My Rule Dependencies: panel) and the **outgoing triggers** in the "Triggered Rules" panel.

Preview and Publishing (Composer v4.3)

Already we have seen, with the simple examples above, that the wireframe's representation of the form does not image all the features of the form as it will appear to the end user, for example here ,
So, while you are designing the form, you can do a preflight of it using Composer's Preview function.
And as with the appearance of the form, the Preview function does not fully reflect the behavior of the form in the real world with regards to attachments or in the Mobile App or in other parts of the form's workflow. To see these, you publish the form to Avoka Transaction Manager (or "TM").

Renditions

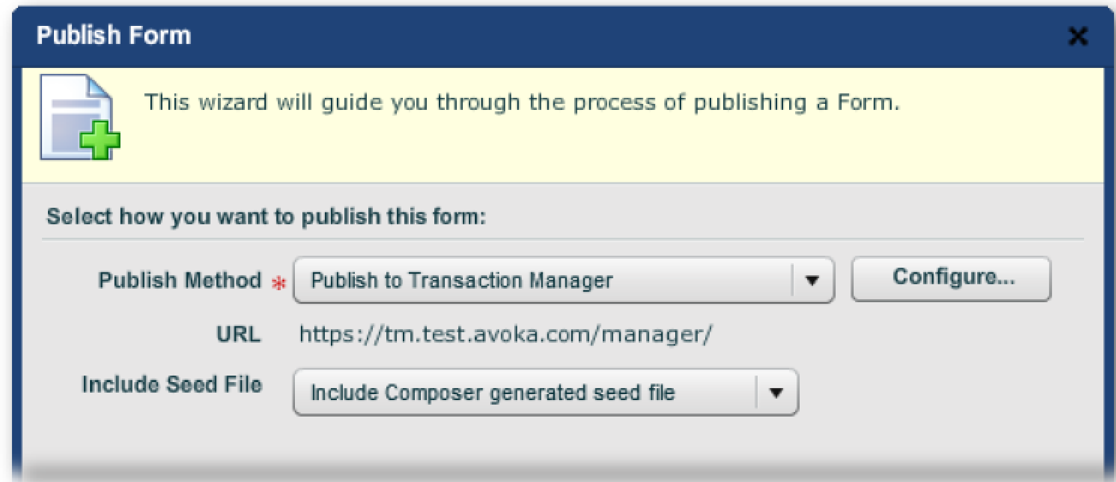
In Composer you define a form — a data collection experience once, if you will — and have it rendered to look and function differently on different devices and environments.
Composer currently supports two core types of form technology (sometimes called "flavors"):
HTML forms using the HTML5 and CSS standards PDF forms using the Adobe XFA standard.
Thus the form often can be rendered in HTML5 and PDF. These days, the general practice is to have users fill out the form in HTML5 and, after submission, get a PDF rendering of the form in the email receipt. The Preview function in Composer has several rendition modes:
HTML Responsive. which uses HTML5 to render how the form will appear in the web browsers of mobile and desktop system.
PDF Receipt, uses the XFA standard and is non-interactive.
The Preview dropdown menu is at the top left of the Form Designer .



(Some older templates give five choices, but now in Composer 4 all the HTML choices have responsive behavior . This is configurable on the form object in the structure panel, "Edit Properties -> Policies -> HTML Generation -> Responsive -> Form Generation (HTML)" dropdown.)
To preview in HTML5, just select **HTML Responsive**. In modern browsers , the preview will render in a new tab. If the form does not render at all, you may have to change the your browser's opup behavior. For example, in Chrome, you can make the Composer hostname URL an exception to popups.
If you need to resize the preview to simulate mobile devices, just pull the tab out from the tab bar and, if necessary, take the resulting new window out of the full size mode. (Do this with the middle button on the top right of the frame in Windows; or with the green candy button on the top left of the Window in Mac OS X.) See the Responsive Layout section of this guide for more on mobile device simulation.
For PDF preview, you must set the browser's preferences to use Adobe Acrobat or its plugin. The reason for this is: all modern browsers have their own PDF renderers and these do not work with XFA.

Publishing to TM

There are good reasons to publish your form to TM while the form is still in pre-production:
The behavior of the form in preview does not accommodate attachments Preview does not show the look of the form in the context of the portal Preview only simulates how the form will look on a mobile device
And the same applies for how the form will look in the Mobile App
You cannot test the behavior of Camera widgets and other use of the sensors of mobile devices
GPS locating is not available in Preview. GPS is available only in those mobile devices that support it. Without GPS, geolocation functions work only from IP address mapping, which is approximate only.
Publishing to TM is simple, and the wizard is accessed through the "Publish" button in the Form Designer or at the Forms Level of the U I.



You have the choice of publishing the form directly to TM (via the URL shown below) or of generating a zip file and importing that manually to TM. Publishing directly is the more convenient.
The layout of a tablet form may also change based on the size of the tablet, and whether it's in landscape or portrait mode. This is not controlled directly by the skin, but is controlled within the form itself. Please see Responsive Layout .
Please consult the Transaction Manager Administration Guide on how to then publish the form to a web portal (or have a colleague with the required access do it for you). You then will be able to access the test form:
through a nominated portal, or
via the Mobile App, configured to point to a nominated mobile portal (see the Transact Mobile App Guide on how to point your instance of the Mobile App to that portal), or
through a URL created by TM in an approved mobile device web browser (see Delivery to Mobile Devices).

Navigation (Composer v4.3)

Introduction

The navigation methods to guide users through a form are of the utmost importance and deserve much thought when designing new business processes and the web forms that embody these.
In an ideal world, navigation through a form should be as convenient, intuitive and frictionless as possible. Achieving this is goal will always be a balance between the needs of the business and trying to provide the user as enjoyable an experience as possible.
So here, "Navigation" means more in interactive forms than providing "Previous" and "Next" buttons at the bottom of the page. It means more than providing menus and breadcrumbs. It means more than restricting the user's access to the form's pages to a fixed sequence. It means more than allowing the user to roam through the form at random. It means more than tailoring the form's layout to the range of devices that will be used to access it. It means more than providing menus, be they down the left hand side, along the top or, in a touch environment, brought up by swiping to the side.
Whatever the business function of a form, all forms ultimately must aim at having the user complete them through to submission. If the user abandons the form, that means a lost opportunity. And we can now see how many users access the form and how many of them advance to the submission stage and how many abandon the form (see Analytics).

The Chrome of Navigation

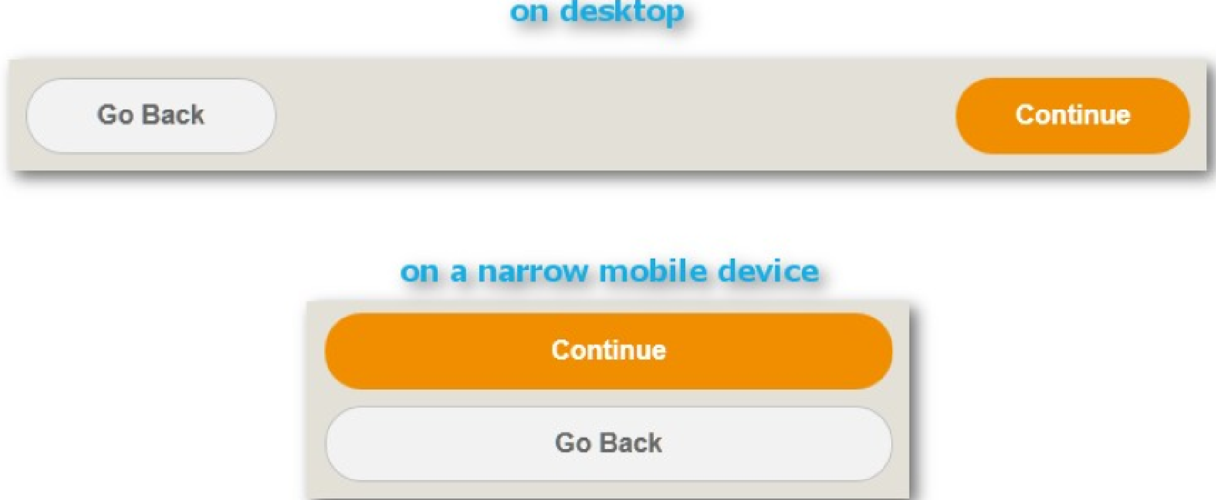
" Chrome is the visual design elements that give users information about or commands to operate on the screen's content (as opposed to being part of that content). These design elements are provided by the underlying system — whether it be an operating system, a website, or an application — and surround the user's data."
— Jakob Nielsen
So, chrome informs users with clues about how to use the form. It is also not part of the form's data — the point of creating the form in the first place — and takes up valuable screen real estate. The smaller the screen, the greater the cost of chrome in terms of space.
But having only minimal chrome, adds friction to the form in that the user will have fewer options to do such useful actions as going back several sections to correct some entry.

The Simplest Chrome: None

At least by this we mean that the form does not introduce its own chrome. However, the usual chrome that comes with the operating system and browser combination is present, and this chrome will not be perfect, even in such a mature use case as web browsing.
So, this means the designer is limited a single-page form, and the only way to navigate the form is to scroll the form in the browser's window. While that does not sound so bad at first, it is very limiting and, where there is a lot of information to be entered onto the form, leads to vertical clutter. It also leads to users getting confused if they have to make revisions. In order to find where to make the revision, users have to remember how far up or down they have to scroll to find the fields subject for change, or — even worse, use the browser's "Find" function. Thus, even with this, the simplest navigation scheme, there is friction, especially if the form is long and detailed.
And the designer cannot control how users move through the form. Some form elements, Sections for example, can collapse and expand. Other elements on the form can switch from being invisible to visible. But users can still skip to the end of the form, in anticipation of the "Submit" button being there, only to be disappointed when that button does not work because they have failed to fill in all the mandatory fields, or have entered invalid information somewhere up the form. See Validation for more.

Button Navigation

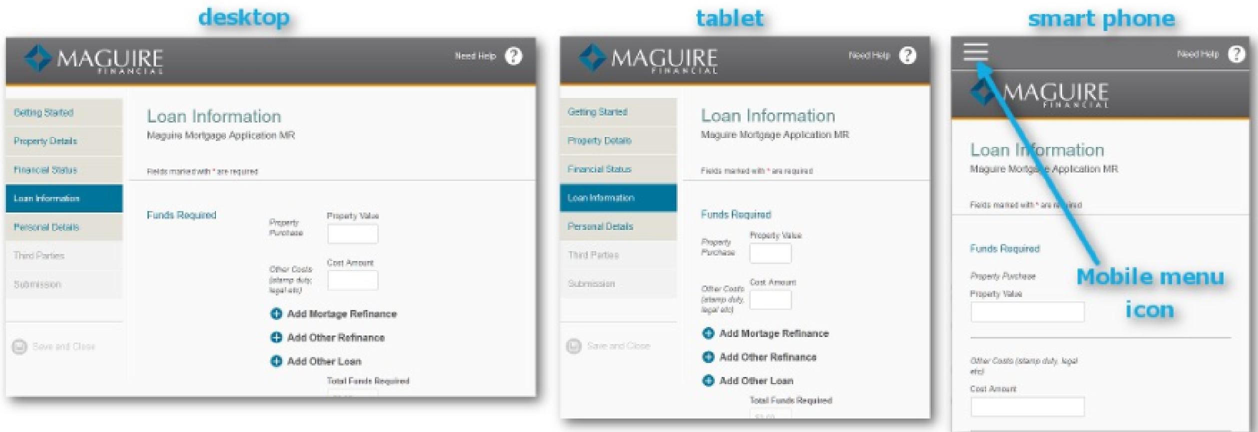
This is slightly more helpful. But users can only step forward and back. This simple chrome uses up the bottom of the page (and may also be duplicated at the top, though many designers resist doing this). This does not amount to much space on the desktop, but in mobile — and especially on narrow smartphones in portrait orientation — that space becomes a large part of the available viewing area. And even more so, when the form is responsive and the two buttons spread out to accommodate the target areas needed for the touch interfaces.



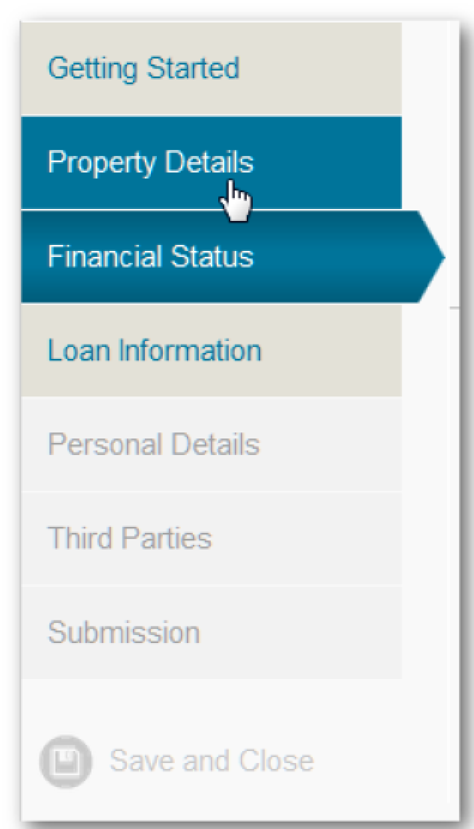
Such a simple concept, though, can suddenly become fraught. The above example has a simple color scheme: orange to move forward and white to go back. But what if users move back and then forget where they have been? Do we indicate whether users have viewed a page through a wider selection of colors? Probably not, because more than two colors have no intrinsic meaning. Users cannot experiment to discover the meanings assigned to colors (such as "already viewed", "yet to be viewed", "not allowed to view", "the last page" and "the first page") except by moving through the pages with the two buttons. On the desktop, where the viewing area is the largest of any viewing environment. the vertical space occupied by the two buttons is small. But on a narrow mobile device (such as a smartphone in portrait mode), the responsive layout function of the form spreads the buttons out so that their tap area is large enough to be useful and stacks the 2 on top of each other. Both are done to minimize user frustration — a vital consideration in mobile.
Note: in the following examples, all mobile screen shots are either from Preview or in the mobile device's browser , not in the TransactField mobile app .

Left Hand Menus

These are a tried-and-true web design element.



The left hand menu shown above has 3 states (indicated by 3 combinations of text color and background color):
Current item, here "Loan Information" Other items, access allowed
Other items, access not yet allowed
The color coding scheme becomes obvious to users through their interactions with the menu. It does not need to be spelled out.
In terms of navigation and user experience, the left hand menu is a vast improvement on 2 button navigation:
Users get clues as to how much more work they have to do before they get to their goal: being able to submit the form.
They can see what is in store, from the labels of the menu items. They get to see where they have been.
If allowed (see Navigation Choices below), they can skip several pages.
They can move several pages back in the form, without getting confused or forgetting where they were. So what are the down sides, given the menu's superiority over the 2 button scheme?
The space below the left hand menu is dead space. There are a few reasons why this dead space exists, mainly due to past limitations of HTML, and such reasons do not matter much any more. But the convention still exists and some usability gurus advocate it. The dead space is hardly an issue on the desktop. In fact, in the above example, the desktop has another level of deadspace (under the subheading "Funds Required"). The rendering on tablet-sized screens eliminates this second level of whitespace.
The menu buttons on mobile have to be large enough to prevent user frustration. But this makes the left hand menu unsuitable for very narrow devices such as smartphones in portrait orientation. In the above example, the left hand menu disappears entirely, replaced by a typical mobile piece of chrome: the mobile menu icon, familiarly called "the hamburger icon".
Another variation on the left hand menu chrome:



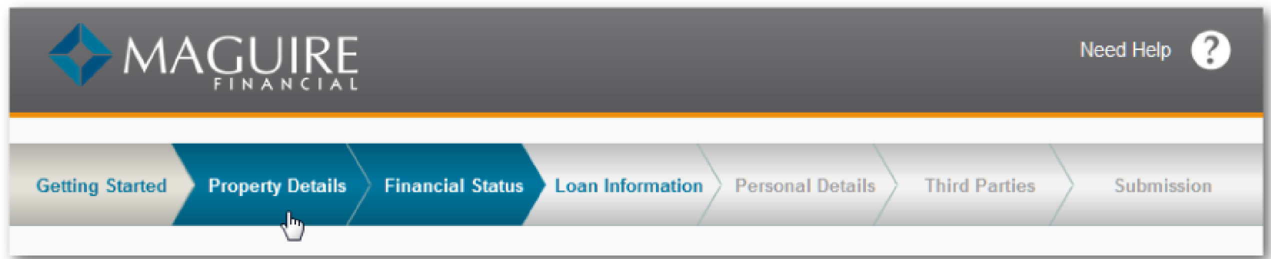
This example has another state added to the 3 already encountered in the previous example: the hover state (as seen in "Property Details" where the user's mouse cursor is hovering over the menu item). Here, "hovering" means that users move their mouse cursor over an object but do not click on their mouse's button to select the object.

LH Menu Variants (Unrecommended)

The left hand menu can have variants, but most of these have, in the light of experience, led down blind alleys.
Right-justified captions. Not recommended.

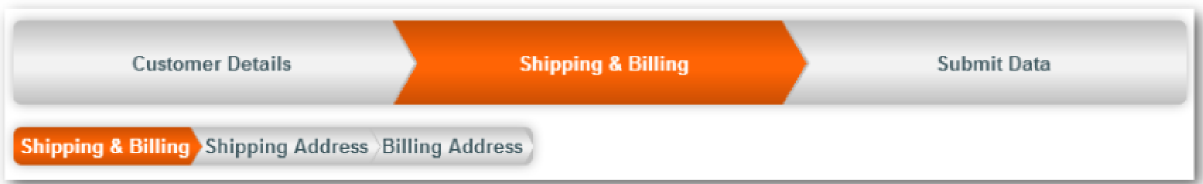
Fly out sub-menus. These have died a quiet death on the web as they demand too much manual dexterity from users.

Top Menus



Top menus are now fashionable and with good reason. They work well on the desktop and with tablets and other wide mobile devices. They also eliminate the dead space of Left Hand menus. This example shows a 5- state top menu: note the differences backgrounds between "Getting Started" (i.e. already visited) and "Loan Information" (available but not yet visited). "Property Details" shows the hover state.
Hover does not exist in a mobile and touch environment. For mobile, hovering is no longer an option.
However, they do not work for narrow mobile devices. And you should keep the captions brief, as the captions begin to wrap for narrower screens. The solution is the same as the above example: the hamburger icon.

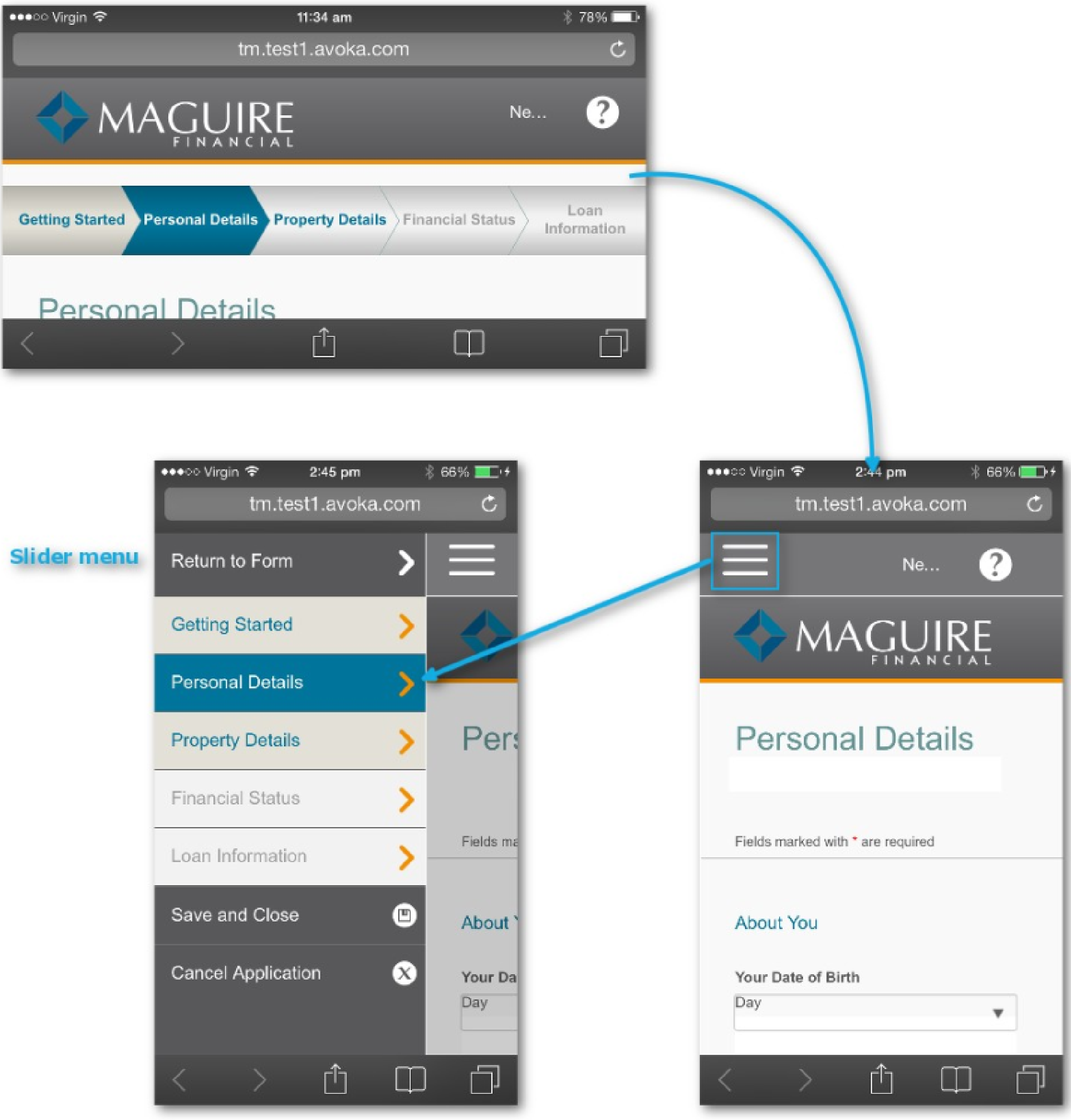
2-Level Menus



Avoka Technologies has given much thought to providing second-level menus to their forms. But we have rejected a large number of the existing solutions, Mega Menus, i.e. big 2-dimensional dropdown menus. These worked well on the desktop, but do not work in mobile due to their dependency on hovering (see Top Menus above).
Flyouts. Again, these are hover-dependent and therefore not an option for mobile.
The example above shows our current practise: "Shipping & Billing" has two subdivisions, shown in the chevrons below the main, topmost menu. Our tests show that these work well, both on the desktop and in mobile.

Mobile Menu Chrome

This is the mobile solution to all the above navigation chrome, aside from the 2-button convention. The Mobile Menu Button is also known among developers as "the hamburger" icon.



Here on an iPhone 5s, the top menu appears in landscape; the Mobile Menu Button in portrait. Tapping on the Mobile Menu Icon causes the menu to slide out from the left. The menu has only 4 states because it lacks hover.

Navigation Choices

Chrome is only part of the story. There are several more degrees of freedom to form navigation.

Unconstrained vs Constrained

Unconstrained here means that users can select any item on the menu at their merest whim. This certainly removes friction but there is a possible down side: the lack of discipline imposed upon users could be a factor in their abandoning the form.
Constrained here means that users can only make a restricted set of menu choices at any one time in filling-in process. To encourage users to understand what is happening and where they stand in their data entry, the subtle coloring of the menu items is tacitly employed — without any tedious explanation. Users who "get it"
will be happy; users who don't will also be happy in that they have not been burdened with the task of ploughing through some ma-rigmarole that is of absolutely no interest to them.

Visible vs Hidden Navigation Choices

Hiding parts of the form is best done to convenience users, not to conceal aspects of the form because they are "unworthy" for whatever arbitrary reason.
So, you may have already noted that the headers of Sections in Composer forms have much that is hidden: the help text, for example. These hidden passages of the form are there to help users who may be struggling and need some assistance. Confident users do not want to see these, as wading through long, unneeded hand- holding text is off-putting and adds for them friction. But not having help for others will add friction for **them**.
Another use case of hiding parts of the form as with accordion wizards, where the sections of the form can expand or collapse (instead of being rendered as new pages). However, do bare in mind that users may not fill in some sections due to inattention — for example not realizing that they should expand some section.
Applying validation will catch this mistake, but you as a designer forfeit the trust of users for letting them down and leading their navigation through the form astray.
Hiding some parts of the form as a result of users choosing some option or another could also be something that eliminates friction for some users and therefore be a good thing.

Validation, Errors and Navigation

There are good reasons for having validation and not allowing users to submit until one or more criteria are met. There are also good reasons why users get fed up with it. Remember: the more users successfully submit a form, the better. In that respect, the web is a popularity contest. The tension between creating friction or gaining the trust of users is not a paradox (a problem to be solved); it is a type of dilemma, where there several options, all correct in themselves, but where you as a designer have to decide what solution to adopt as a judgment call.
Error blocks and validation messages are good to have, but remember the friction they create and that they will win you no popularity.

Abandonment

Abandoned forms are a fact of life in online interactions. We have done our best to give you the tools to achieve reasonable user navigation and hopefully to reduce the percentage of abandoned forms.
To that end, we have introduced to Composer 4 templates capable of detecting where abandonment occurs in the form filling process. This data is then accessed through TM. Note though that there are privacy implications.

Portal Chrome

Avoka Technologies strongly recommends that forms do not appear embedded in the portal page of your enterprise. One common problem is that the portal itself is not responsive and thus the form contained within it will not display properly on a mobile device. Instead, we recommend that:
the form take up the whole browser window (similar in effect to the a href tag's parameter
target="_parent" in html)
is not be rendered in a separate window or tab (as seen, for example, with target="_blank")
so, as a corollary, there should not be breadcrumbs on the form, especially those that lead back to the portal's home landi

Sections

Introduction

In previous versions of Composer, navigation was implemented in forms through the wizard settings of the form and the section fields in the Structure Panel. Sections could be organized into three levels. Each section had an elaborate header which included collapsible Help, and various other user aids; sections also had footers.
The content was between the two.

So, using Sections to divide the form brings the following advantages:
Sections have chrome associated with them, such as headers, footers, collapsible help and inclusion in the navigation menus.
So, all this associated chrome will be consistent throughout the form. Changes to the chrome can also be (if desired) applied globally
Constrained and Unconstrained menu navigation is controlled through Sections
Section styling is professionally designed, including margins, fonts, and colors, to provide an attractive form Section styling is controlled via stylesheets. Change the stylesheet to subtly or radically change the look of your form.

In Composer 4, the navigation menus (shown above) are more powerful than in older version. To accomodate the new menu functions, there is a break from the older way of setting section levels (as embodied in the Section Assistant shown below) and the new way. Also, now there are many more "Edit Properties" settings for sections that control these new menus. The older section levels and the older style of menus are still present in the product to retain compatibility with legacy designs.
So, we will first discuss the older navigation functions through Wizard settings. Then, we will move on to the new ways of configuring navigation.
The headers and footers of sections contain many important elements of sections, such as the numbering of sections, the help and more.

Maguire Template

Some of the section navigation behaviors are embodied in the template. This advanced topic is covered in Templates below.
The mobile menu is a feature of MaGuire (see Templates for an example of this template).

Adding a Section

The following applies equally to all templates, including Maguire.
A Section is really a specialized type of block, which uses features of blocks, including having children and hiding sub-blocks within it.
A convenient way of creating sections is to use the Section Assistant and its wizard.

? Unknown Attachment

The Section Assistant wizard.

Section Levels

In the older versions of Composer, there are three levels of sections provided in the Palette: Levels 1, 2 and 3. This scheme is retained in Composer 4 for compatibility, but we will discuss below the newer way of organising Sections.
The style of each level can be controlled independently.
These older style of Sections Levels must always be correctly nested within each other — Level 1 at the top, with Level 2 indented one level below, and Level 3 indented inside Level 2. If you nest your sections incorrectly, then your form may look strange. If this happens, simply nest the sections correctly.
There is no requirement to use Sections or all the available section levels, although most forms use at least one Section Level 1.
In previous versions of Composer, Level 1 sections are the main wizard pages. This has changed with Composer 4.

Maguire's Use of Sections

Instead of using Section Levels, Composer 4 now uses Section Groups to organize the menu items into a 2- level navigation scheme.

Maguire Template use of Level 2 Sections

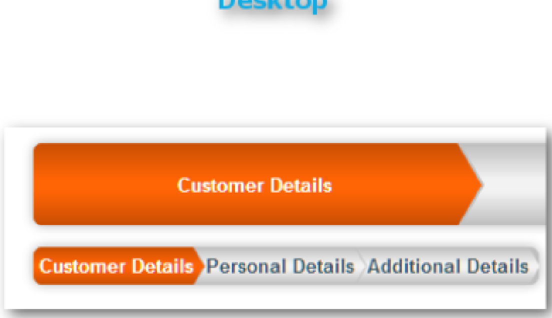
We recommend that each Level 1 section in the Maguire template have **at least 1** Level 2 section. This is **not** for inclusion in the navigation menu; the Maguire template makes use of Level 2 sections for optimum layout. The content of each Level 1 section should be distributed among several Level 2 sections. T

Wizards

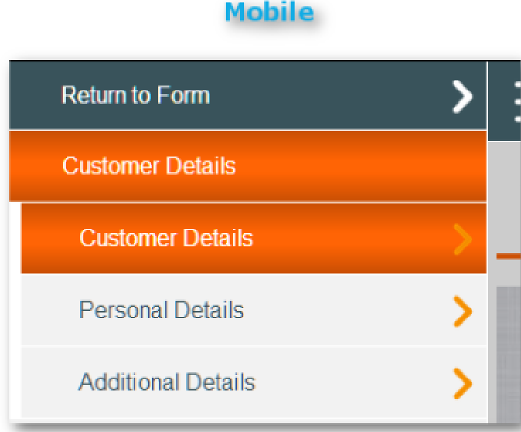
In older Composer versions, the whole form was set to be a Wizard, a Wizard with a menu, or not a Wizard at
all,
Now, individual sections can set to act as Wizard pages in the Wizard Panel. And, independently, a section can also be set to become a menu item or not.
"Edit Properties -> Properties -> Wizard -> Wizard -> Act as wizard page (checkbox)"
"Edit Properties -> Properties -> Wizard -> Wizard -> Create Wizard Menu Entry (checkbox)" "Properties -> Wizard -> Wizard -> Add to Navigation Menu (checkbox)"
So, if you opt for having a page not act as a wizard page, it will be appended under the previous page; if you still make it create a wizard entry and add to the navigation menu, the section will still appear in the menu, despite not being a separate page.

2-Level Navigation

Desktop



Mobile



MaGuire template top 2-level menu

Now, all Sections in the form menu are Section Level 1 (see Adding a Sectiom), irrespective of whether they are top level items or second level items in the navigation menu. A section's menu level is now determined by the property
"Properties -> Wizard -> Wizard -> Section Group Name"
Top Level menu items now belong to their default group (i.e. "${section.heading}")
Second Level Items now are grouped according to their top level menu item's name. With the menu shown above:
"Customer Details" belongs to the section group "${section.heading}"
"Personal Details" & "Additional Details" both are made to belong to section group "Customer Details".
Also, they are both have the setting "Properties -> Wizard -> Wizard -> Add to Navigation Menu (checkbox)" as checked.
Note: Though 2-level navigation is not exclusive to Maguire, the Chevron format of the top menu (shown immediately above) and the slider ,mobile menu are Maguire features.

The Navigation Menu in Composer

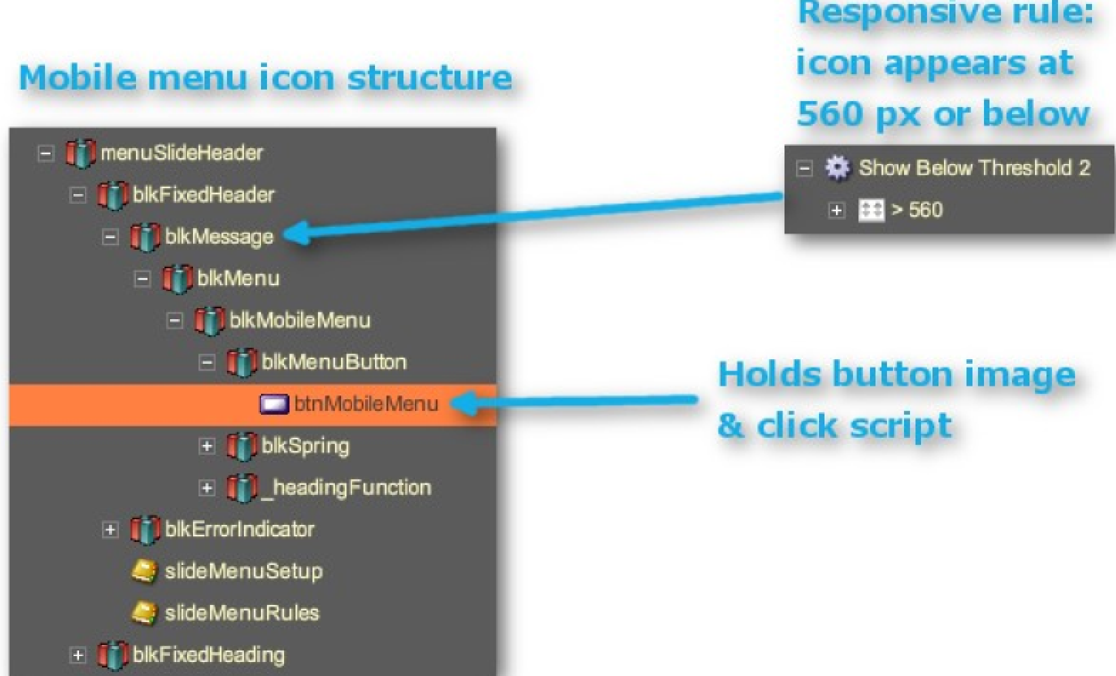
Features common to all templates

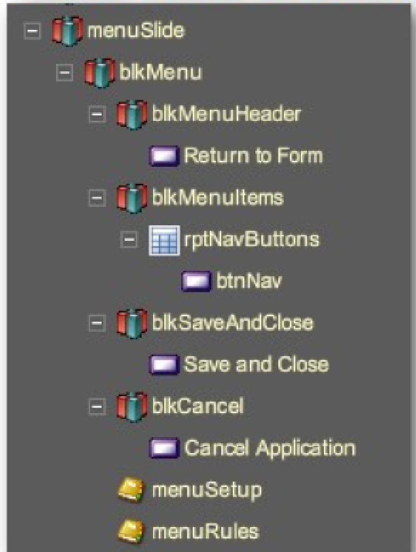
Properties to configure the appearance of navigation menu states (discussed in LH Menus above)
On the form object at the top of the structure panel tree, "Edit Properties -> Theme -> Menu ->" and then these panels:

- "Menu Entry"
- "Menu Entry - Active"
- "Menu Entry - Disabled"
- "Menu Entry - Enabled"
- "Menu Entry - Hover"
- "Menu Entry - Visited"

Maguire Exclusive Features

Mobile menu navigation for narrow mobile devices
Illustration of the structure behind the mobile menu seen in Navigation .



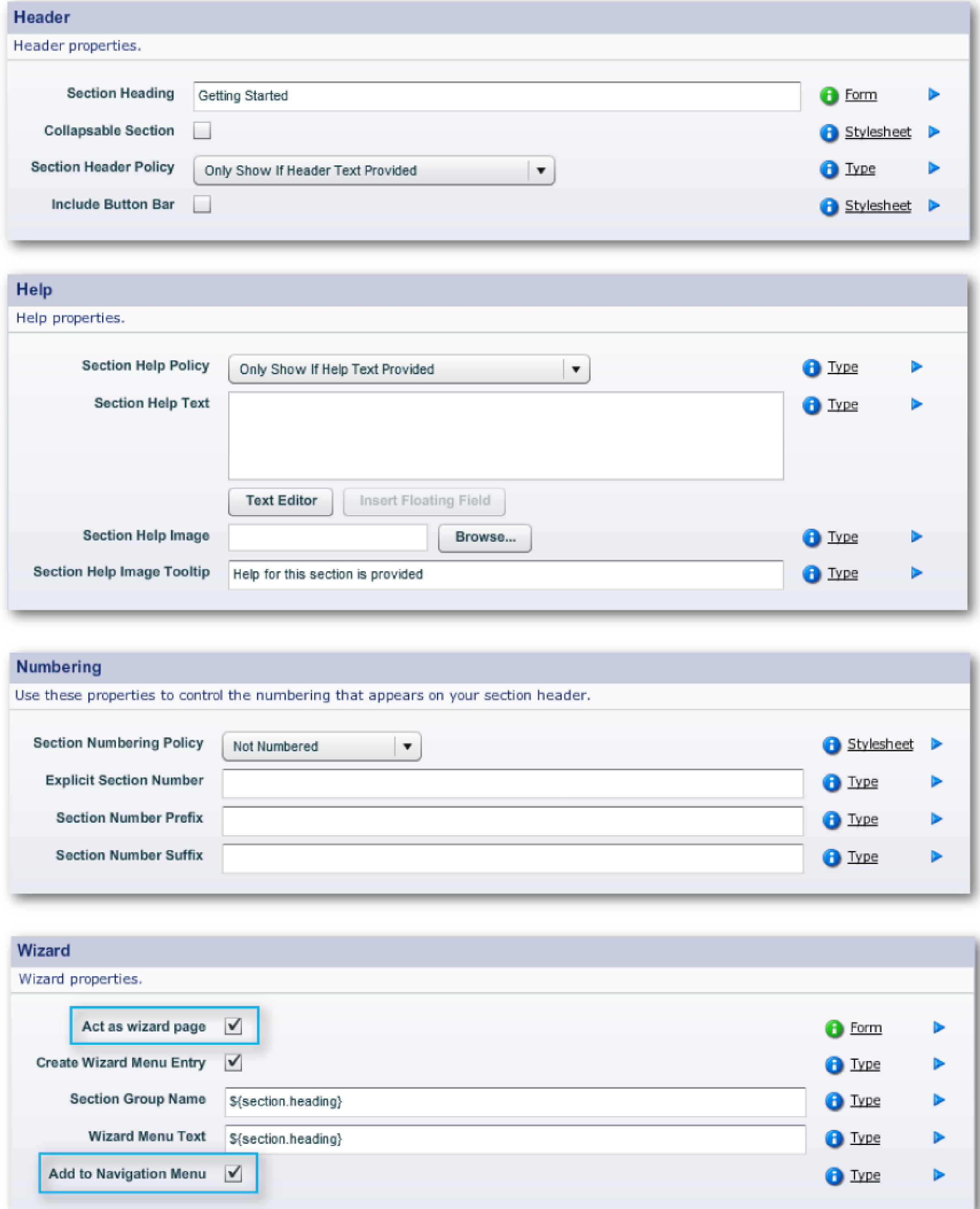


The Maguire navigation menu does not render in wireframe
To view it, you must preview the form, where you can also see the different responsive layouts of the menu.

Headers and Footers

The Simple Approach

Composer 4, and the older versions of Composer, use the "Edit Properties" of the section objects in the structure panel to configure the sections. This approach is useful for prototyping and proof and concept. It will even work for mobile devices, despite its limitations.
You use 4 panels to control the section objects with the simple approach:
Properties -> Section Header -> Header Properties -> Section Help -> Help
Properties -> Section Numbering -> Numbering
Note: these numbers only appear in form menus controlled by the Wizard panel. They do not appear in Maguire-style navigation menus.
Properties -> Wizard -> Wizard
Note: this panel controls whether the section appears in the (Maguire-style) navigation menu and whether it behaves as a wizard page. In Composer 4, forms can now have some Level 1 sections behaving as wizard pages, while other Level 1 sections may not. In earlier versions of Composer, the whole form either was a wizard or not.

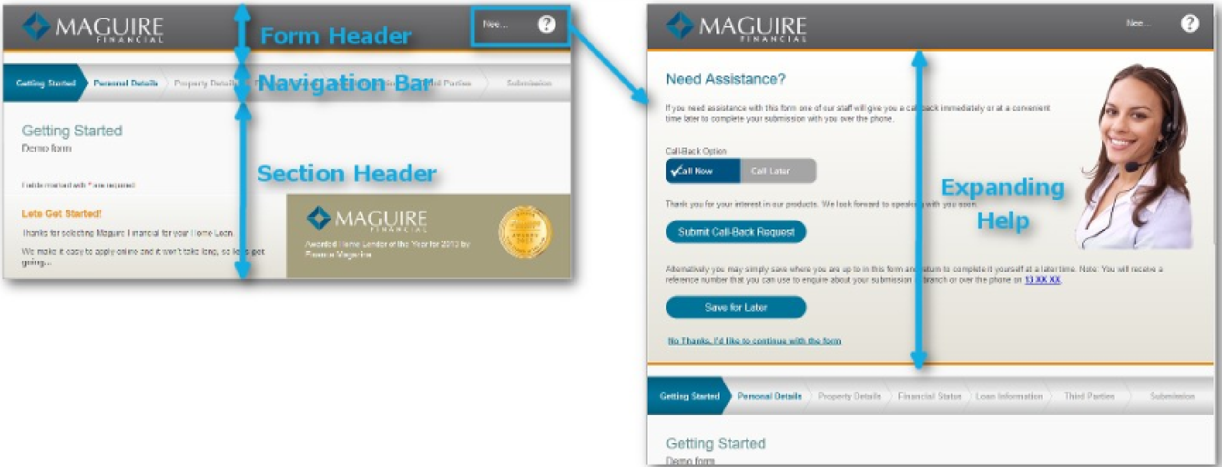


The 2 properties highlighted in the above image of the Wizard Panel support the Maguire-style Navigation. Make sure they are checked when you create a new section in a Maguire-style template.

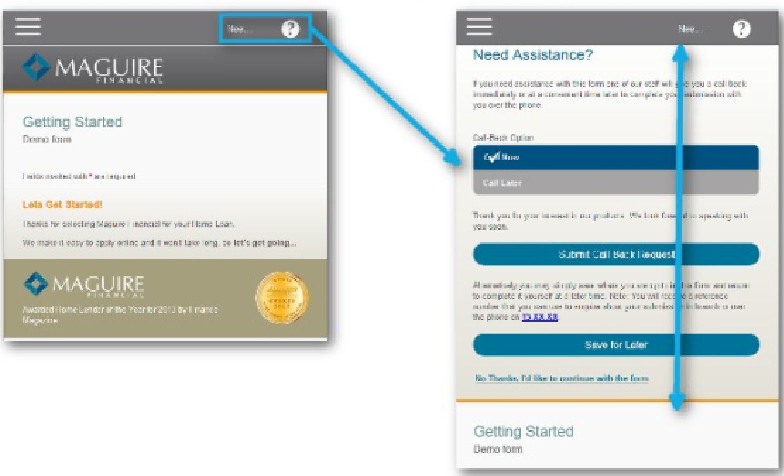
Maguire-style Navigation

The whole approach in Composer 4 and the Maguire style of template is diffewrent from that of the earlier versions of Composer.
The earlier approach was to have the help passages of the form associated with each section of the form. Now, the Maguire style has an overall form header with global help content across the whole form, as part of Composer's responsive layout navigation strategy.

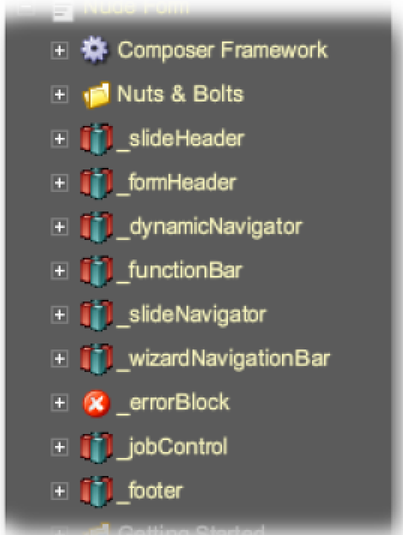
Desktop



Narrow Mobile Device



All these features are controlled in the Structure Panel through the following blocks and their contained objects, most of which are visible only in the Advanced Mode of the Structure Panel..



These blocks expand into a large number of contained elements and it is beyond the scope of the this guide to document every aspect of this structure. By all means, explore these blocks to see how the Maguire navigation structure is built up and edited.

Footers

Here, we mention footers only in passing. In the Structure of the form, footer blocks come under the Advanced Mode. There is a global footer block for the form, which is defined in the block's "Edit Properties -> Layout -> Layout Constraints -> Layout Data -> South". Also, each Section has a footer block defined in the Advanced Mode Structure as *Some Section* -> Outer Area -> Footer Area.
In the above screen grab , the global footer of the Maguire-style sample is "Transact Footer".

Recolor and Hide on Print

Re-color is a separate facility to the concept of a separate Receipt renditions (see Preview and Publishing) for printing. In general, we recommend the use of Receipt renditions rather than Re-color on Print.
Most of Avoka's standard Section styles make use of colored backgrounds, with fields having white backgrounds. Studies show that this is best practice in terms of look and usability. (It is also possible to create style-sheets with a more "plain" appearance - please contact Avoka if you would like to explore this option for your organization's forms.)
However; many organizations and individuals still utilize black and white printers, or wish to conserve the amount of ink used for printing forms. Large "swashes" of color do not satisfy this goal, and also do not fax very well. Therefore, most of our sections have the ability to "Re-color on Print", found in "Edit Properties -> Properties -> Printing".
"Re-color on Print" means that when the user clicks the print button in Adobe Reader, the form will "morph" to a more printer-friendly black-on-white look, and then morph back again once printing is complete. This behavior can be controlled via a property on the sections. Note that some widgets (such as Images) will not observe re- color on print, and will print in color if available.
Re-color on print only applies to PDF forms and PDF receipts.
A related property allows particular objects to be hidden when the form is printed, "Edit Properties ->:Properties
-> Printing -> Visibility on Print [PDF]". You can explicitly set this property on blocks if you want their contents to be hidden on print.

Layout (Composer v4.3)

Just as the concept of form navigation has changed now that users will be interacting with your forms on a variety of devices, where formerly you could have been certain that they would only be using a desktop computer or laptop. Indeed, we now group laptops with desk-bound computers and call them collectively "the desktop". The important distinction is that "the desktop" is now characterized by a large monitor and a pointing device (either a mouse or a trackpad).
Mobile devices have restricted screen space and the user interacts with them through a touch interface. See Devices vs Desktop for more.
As a result, Legacy Layouts now are a thing of the past. What worked in hard copy will probably no longer work on a smartphone.
This early parts of this section of the Composer 4 Guide explores these issues a little. The section ends with Responsive Layout .
Also, forms these days will be consumed more in HTML rather than as PDF. User expect this. In fact waiting for a PDF form to load, which often take much longer than the HTML version of a form, is more unwanted friction for the user.

Legacy Layouts

Overview

There are several different factors to consider when laying out a form. We encourage you to think carefully about your goals, rather than trying to lay out a form to match an existing paper form.

Goal	Suggestions
Correct form layout on multiple devices, with differing widths and screen real estate, or switching from portrait to landscape mode.	Keep your layout as simple as possible. Avoid fixed width fields Use vertical layout managers. Use Layout-Fill. Avoid fixed width fields.
Useability, as much as possible, with grouping of related fields together.	Use vertical layout managers, with keep-on-same-line to keep related fields on the same line. For more control, use horizontal layout managers for each line, within an overall vertical layout.
Both PDF and HTML.	In general, design first for HTML, and then preview in PDF and adjust as necessary.
Fitting the form onto a single page (no matter how big the form is).	Composer is not designed to make use of every bit of whitespace. Usability studies show that whitespace in a form greatly improves readablity and usability. Disable all interactive features, including hide/show logic. Expand multi-line text fields, section help, expandable tables and repeats. There is no cost for an electronic forms to have multiple pages. In fact, multiple pages save money by increasing usability. If a form is ultimately going to press, consider having two different form designs: one optimized for interactivity, the other for receipt or print.
Mimicking an existing paper form.	Use horizontal layouts, grid layouts, and fixed width fields as necessary. Use caption left where necessary. Make the desktop HTML a fixed page width. (This is the default for desktop HTML.) Don't try to make your form identical to the paper form. Remember, you want a better user experience for your users, and that means having design changes.

Goal	Suggestions
Replicating an existing paper form as a "pixel perfect" copy.	You cannot create pixel-perfect copies of existing forms with Composer. If you require pixel-perfect forms, you should consider creating a separate receipt form. o **Note:** Transaction Manager can be configured to support a separate form for the receipt and for the interactive form.

Keep it simple

Keep your layout as simple as possible. More complex layouts become extremely challenging. Keep it simple, unless you have very specific requirements.

Let Composer do the work

Composer's Layout Managers do the fields alignment and uniform sizing, based on your organizational template and style-sheets. Better to let Composer do the work, rather than struggling with Composer for the sake of a particular look.

Use Percentages, not Fixed Width layouts

You don't know how big your user's screen will be. Try not to make too many assumptions about widths, and let Composer do the work of layout out your form.

Use Field Styling

You can use Advanced mode in the property editor to explicitly set field widths in pixels, mm, or inches, known as "magic numbers". Sometimes this may be necessary, but this introduces two problems:
Every form designer needs to remember these "magic numbers" to achieve consistency.
If you need to change some aspect of the layout, you need to modify each field independently.
Composer allows you to define standard fixed width field sizes such as Small, Medium and Large in your stylesheets (see Field Styling). The advantage of these named sizes is that there is nothing to remember, and if you need to change them, you can simply adjust the style-sheet. A good rule of thumb is that these sizes should all be multiples of each other so that combinations of smaller and larger fields all line up.

Checkboxes and Radio Buttons

It can be very useful to use a Grid Layout for groups of checkboxes and radio buttons. Use the same number of grid cells for every set of radio buttons or checkboxes on your form, so that they all line up correctly. Use spans for captions that are very long. We recommend 4 to simplify responsive layout, although this is your preference. Please see the Grid Layout topic on this for details.
Please note that the grid layout is not ideal if you want your form to lay out correctly on narrow screens like smartphones.

Units and usage tips

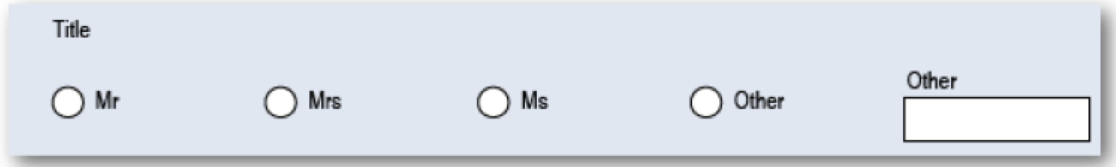
Sometimes it is necessary to put units or other tips on a form. For example "miles" for a measurement of distance, or "MM/DD/YY" as a tip for the format for a date field. You can use a Plain Text or Rich Text field for this purpose, but often its difficult to get the text to line up correctly with the field itself, particularly with Caption Top fields.
Composer has a special Label Field that always ensures that the label is aligned with the Text Field that can be used for this purpose.

Caption Top or Caption Left

We generally recommend Caption Top, for a number of reasons. This is summarized here: {+}http://+ www.avoka.com/composerblog/2011/10/form-design-tip-caption-top/

Mixing Checkboxes or Radio Buttons with Textfields on a line

If you mix Text Fields with Radio Buttons or Checkboxes on the same line, you often get results where the resulting layout looks bad. This is because Radio Buttons and Checkboxes, by default, have their caption on the same line, while Text Fields (for most Composer stylesheets) have Caption Top.



There are a number of simple ways of solving this problem.
Remove the caption by clearing the Label value, or by checking the advanced property Properties -> Caption
-> Caption Content -> Never Show Caption. This effectively associates the caption on the Other radio button as the implicit caption for the text field.
Set the "Properties -> Caption -> Caption Presentation -> Caption Placement" on the Text Field to "Left". Place the Other Text Field on the next line.

Boxy Layout

Many paper forms designed for the US Government use a style known as "Boxy". Here is an example of this style:

Form 1040 Department of the Treasury—Internal Revenue Service (99)
U.S. Individual Income Tax Return 2011 OMB No. 1545-0074 IRS Use Only—Do not write or staple in this space.
For the year Jan. 1–Dec. 31, 2011, or other tax year beginning , 2011, ending , 20
See separate instructions.
Your first name and initial | Last name | Your social security number
If a joint return, spouse's first name and initial | Last name | Spouse's social security number
Home address (number and street). If you have a P.O. box, see instructions. | Apt. no. | Make sure the SSN(s) above and on line 6c are correct.
City, town or post office, state, and ZIP code. If you have a foreign address, also complete spaces below (see instructions).
Presidential Election Campaign
Check here if you, or your spouse if filing jointly, want $3 to go to this fund. Checking a box below will not change your tax or refund. You Spouse
Foreign country name | Foreign province/county | Foreign postal code
Filing Status
Check only one box.
1 Single
2 Married filing jointly (even if only one had income)
3 Married filing separately. Enter spouse's SSN above and full name here. ▶
4 Head of household (with qualifying person). (See instructions.) If the qualifying person is a child but not your dependent, enter this child's name here. ▶
5 Qualifying widow(er) with dependent child

An example of a Boxy layout

This layout was designed around half a century ago, when it had some advantages for form-designers who created these forms using simple typographic tools, and the printers who printed them. It has serious disadvantages in the modern world, which include:
The layout is often very cramped, and is designed to cram as many fields on the form as possible to reduce printing and handling costs. There is no need for electronic forms to have this restriction, and the value of white space within a form to aid in usability are well documented. Usability studies have shown that this approach creates a large number of problems, including customers having difficulty understanding the form, missing data, and difficulty for those reviewing the form to find the information they need.
Inline help needs to be very concise, and so help is often either terse and difficult to understand, or provided in a separate document. Electronic forms do not suffer from this restriction.
The layout is extremely difficult to maintain. The addition of a single field or changing of even a single word on the form can result in large portions of the form needing to be re-designed and re-positioned.

The fields themselves are difficult to locate on the form, as they don't really stand out from the surrounding text.
Even for handwritten forms, the caption text gets in the way of the area for writing, and often leads to cramped writing which is more difficult to read.
There is no possibility of using hide/show logic or repeating sections in the form, because the entire form is so tightly structured. The 1040 form only allows for four dependents, and has special instructions for applicants with more than four dependents.
Getting fields to line up correctly can be very challenging, particularly in the presence of column-spans and row-spans.
This format is often unsuitable for capturing electronically and feeding into backend systems, because often multiple logical fields are compressed into a single line. For example, in the above example the "city, town or post office, state and zip code" are all combined into a single field. This is easy for a human to read, but quite difficult to split into separate fields to upload into a back-end database.
This style is particularly unsuitable for re-flowing onto different screen sizes, because the fields have very fixed widths and positions, and any attempt to reflow them causes the entire form to loose its visual integrity.
There is generally little consistency within these forms, because they are completely hand created, and often tweaked many times over a long period. The sample above uses text of various different fonts, different techniques for displaying help text, and different styles of heading.
The result of trying to maintain electronic forms in the Boxy style is:
Confused and unhappy users due to low usability
Unsatisfactory forms on tablets or smartphones
Frustrated form developers and high maintenance costs.
It is possible to create Boxy style forms in Composer by creating fields with outer borders, and ensuring that at least one field on each line is set to Horizontal Fill (or other techniques), but, this is a challenge. We generally recommend that it be avoided.
We suggest the following approach:
Use one of Composer's existing stylesheets as the basis for your own organizational stylesheet. This ensures that your forms look and function well on all types of devices including tablets, and on multiple screen sizes, as well as allowing you to make use of advanced features such as hide/show logic, repeating sections, and intuitively laid out forms.
If there is a need to produce a printed or electronic form that is exactly the same as an existing paper form, import the existing form into LiveCycle Designer as artwork, add overlay fields, and use this form as the receipt in Transaction Manager. You will need to bind the fields in the artwork form to the same XML Schema that is used and produced by Composer, and then all the fields will simply flow across from the interactive form to the receipt. The artwork form is generally quite easy and quick to implement, because the layout already exists, and there is no need to add any interactive features since it will only be used for output. You may need to make certain restrictions in your interactive form such as limiting the number of rows in certain cases.

Layout Issues

Measurements

There are several different ways to specify measurements in Composer. There is also some "tension" between the different ways to specify measurements:
PDF is primarily a physical page-oriented technology, and the most natural way to specify measurements is in physical units, such as inches or mm.
HTML is primarily a screen-oriented technology, and the most natural way to specify measurements is in screen units, such as pixels for desktop devices.
Mobile devices (particularly iOS) prefer to measure in points (pt).
In responsive layout, the unit of measurement is the px , but this no longer means "pixels" as it did on the desktop.
Composer's Desktop style HTML rendering does attempt to mimic the page-oriented layout of PDF, whereas tablet and mobile rendering largely dispense with the page-orientation.
Composer allows you to enter measurements in any one of these units, and will convert these units on your behalf. Conversion from screen units to page units assumes a particular DPI, and mixing these units may result in slight discrepancies.
Measurements can be changed within a form for field widths and for margins. Most other measurements should be left alone, as they are balanced for use within the stylesheet.
Composer also allows you to specify measurements in percentages or weights if a layout manager is being used — in this case, the actual measurement will be calculated by Composer based on the relative sizes of several different components.
Images height and width are usually specified in pixels, based on the actual pixel size of the image. If you resize an image, best results are often obtained by using integral fractions of the original image size.

Recommendations

In general, you should avoid using explicit widths as far as possible. The main reasons for this are:
Often you don't know what size screen your users are going to be viewing your form on, and you should let Composer take care of sizing your fields to fit the screen. Use a layout manager rather than explicit widths. Widths should be specified in the stylesheet, not in the form. This way, if you need to change anything, you can do it in a single place (the stylesheet) rather than having to do it in every single field.
If you do use explicit widths, we recommend:
Most of Composer's in-built stylesheets use page units in mm. Unless you have a specific reason to do otherwise, it's generally recommended to use mm.
If you are building forms primarily for HTML, and require very explicit control over sizes, specify pixels (px). Note that the meaning of "px" has changed with the advent of high resolution mobile devices. See the relevant passage in Responsive Layout .
If you do need to apply particular sizes (e.g. widths) to your fields, consider using a named style-sheet entry, or a layout manager — this is more maintainable and flexible.

Devices vs Desktop

The parameters that define mobile, touch and desktop are changing rapidly as the computing environment is in a state of flux. Time was when "mobile" was synonymous with "touch". Then Apple introduced "Multi-touch" trackpads to its full range of laptops, with the result that classic touch gestures, such as spreading two fingers to zoom in, moved to the desktop. Next, Windows 8 came along and laptops began to have touch screens for an even more direct experience of manipulating objects with the fingers. It is now commonplace for users to interact with commerce sites with their fingers on tablets.

Though we have already (in Navigation) made the point that mobile devices have different requirements from the desktop, those distinctions are becoming more of a spectrum rather than discrete steps.
For that reason, we would like to show you following video presentation. It lasts about an hour, and is worth your time.
http://vimeo.com/64202295
Video from Luke Wrobelewski http://www.lukew.com/presos/preso.asp?31

Delivery to Mobile Devices

Two Modes of Delivery

Mode 1: The Web Browser

As we have already said, the computing environment is now in a state of flux. In the past, when all web browsing took place on the desktop, there was a period — rightly or wrongly referred to as "browser wars" — when web design was complicated by the different and undocumented ways that the various browser of the time would render the same web page in quite drastically different ways. This came at the worst possible time for the web, when HTML standards were also in transition to the adoption of Cascading Stylesheets (CSS), an attempt to separate style from content.
The end result was the concept of the so-called "modern browser", which means one that supports the latest iterations of the CSS standard.
However a new skirmish has broken out, now on mobile. The current state of play is that there are 3 broad types of operating systems in play:
Apple's iOS operating system, which runs iPhones, iPad and iPod touches,
iOS usually updates annually and, because there is little friction impeding this, most iOS devices rapidly upgrade to the latest version of the operating system
Android, originally developed by Android Inc, which was bought out by Google.
There are a number of manufacturers who produce Android phones, most notably Samsung.
Generally, few Android phones update after purchase to the latest Android release, the reasons for which are many and too complex to discuss here.
Also there are a number of manufacturers in China who have "forked" Android and whose devices run no longer on a Google-sanctioned version of Android.
Microsoft's Windows Phone operating system.
This operates on mobile-devices only. It does not have the market share of the above two. Microsoft's Windows 8.
Microsoft has been pursuing a "Windows everywhere" policy and has, with its latest iteration of Windows, intended that Windows 8 runs on both the desktop and on larger mobile devices such as tablets.
To further complicate this, there are two flavors of Windows 8 on Microsoft's own brand of tablets: the Surface. One runs on Intel central processors; the other (the Surface RT) runs on completely different central processors. The two flavors are not compatible or interchangeable and the Surface RT requires a different version of Windows.
And if that were not enough, every copy of Windows 8 ships with two different browsers: Internet Explorer for the familiar user interface of Windows and Internet Explorer running on the Modern user interface.
This is only the beginning. The different mobile operating systems also have different browsers, usually a default browser that ships with the phone and others that on some or all of the phone models and manufacturers running that operating system

Operating System	Common versions	Default Browser	Other Browsers
iOS	7, 6	Safari	Firefox, Chrome
Android	4.4 (KitKat)	Chrome	Firefox
	4.1 to 4,3 (Jelly Bean)	"Android Stock Browser"	Firefox, Chrome

Operating System	Common versions	Default Browser	Other Browsers
	Previous versions		
Windows Phone	8, 7	Internet Explorer Mobile	?
Windows 8	8.0, and soon to be released 8.1 (64- and 32- bit versions)	IE and Modern Interface IE	Firefox, Chrome on the standard Windows desktop. IE only on the Modern desktop
Windows 8 RT	8.0 (32-bit)	IE for RT and Modern Interface IE for RT	IE only for the two types of desktop.

It is no surprise that all of these have their quirks and incompatibilities. And, the mobile hardware has also evolved rapidly, so that web pages that failed on many of the older mobiles in the past now run quite comfortably on the newer.

Mode 2: Via Native App

As mobile users know all too well, the other means of delivery is through mobile apps, as these generally offer speed and stability in a given mobile operating system. At least, that is the theory.
Avoka offers the TransactField App , where you can publish your forms to the mobile app without having to redesign the form to run in the app. There are even some widgets that work solely in the app and not the browser.
This mobile app is available for iOS, Android and Windows Mobile Phone.

Which Mode is Superior?

The answer is: it depends. If you need to make use of an app-only function, the choice is obvious. If users are coming to the form from a website, the choice is equally obvious.
Just remember, too, that installing an app (which is usually done through an online store, for instance the Apple Store) is a further source of friction for users.

Responsive Layout

Note: Internet Explorer 8 **does not support responsive behavior** because IE 8 does not support modern CSS frameworks. Responsive Design as a practice only began in 2010.

Introduction

Responsive Web Design is a technique for web pages to render properly both on the desktop and mobile devices. A responsive page can be served to desktop browsers or to mobile device browsers, such as smart phones or tablets, and will render optimally on all of them without the need to detect ("sniff") on what device the form is being rendered, nor the need to serve different versions of the page.
It follows that all forms should now incorporate responsive elements and you should now assume that many users will be accessing all forms from mobile devices.
As we have already said, there are 2 modes of delivery to mobile .
The responsive behavior of a form is controlled by the template associated with the page. There are a number of settings in Nuts & Bolts under "Nuts & Bolts -> Responsive Rules" and these are tuned to the Maguire-style of template.
The Maguire Responsive Behavior depends on 3 thresholds values of the width of the device measured in "px".
This no longer means a number of physical pixels on the device's screen, because many modern devices now have higher resolutions than the monitors of the previous century, yet the expectation is that pages should
display the same way as they did before. Also there are devices that have high resolution models, "Retina displays" and the like. Again, the expectation is that objects should be rendered the same size on these different device models. Objects cannot be rendered smaller on the higher resolution models because of the touch-based interaction with the user: smaller objects frustrate users as they attempt to accurately hit touch targets.
So the portrait mode width of an iPad (for both Retina and standard resolution models) has been standardized as 768 px and an iPhone as 400 px.
An Operator in a Ruleset determines when the responsive rule set kicks in. The determining factor is the width of the device's browser, measured in "px" (also called "css px" in other documentation), which no longer means a number of physical pixels on the device's screen, because a number of modern devices now have higher resolutions than the monitors of the previous century, yet the expectation is that pages should display the same way as they did before. Also there are devices that have high resolution models, "Retina displays" and the like. Again, the expectation is that objects should be rendered the same size on these different device models. Objects cannot be rendered smaller on the higher resolution models because of the touch-based interaction with the user: smaller objects present difficulties for the fingers of users to accurately hit touch targets.
So, the value of the width of any device in px is the value that the device itself reports back to the engine handling CSS rendering in its browser and in apps running on the device.
So, here, for comparison, are the widths that apply to Apple mobile devices:

Device	Portrait width in px	Landscape width in px
iPhone 4, 4s	320	480
iPhone 5, 5c, 5s, late model iPod Touches	320	568
iPads (both Retina and non- retina displays)	768	1024

Composer use of Thresholds

To manage the complex sets of responsive behaviors shown in Navigation above, Composer 4 and the Maguire- style of template have 3 levels of thresholds, set in "Form Options -> Generation Options -> Edit Properties -> Properties -> HTML Generation" in the Responsive panel. The default values are:

Threshold Level	Triggering width (in px)
1	768
2	560
3	400

Previewing Responsive Behavior

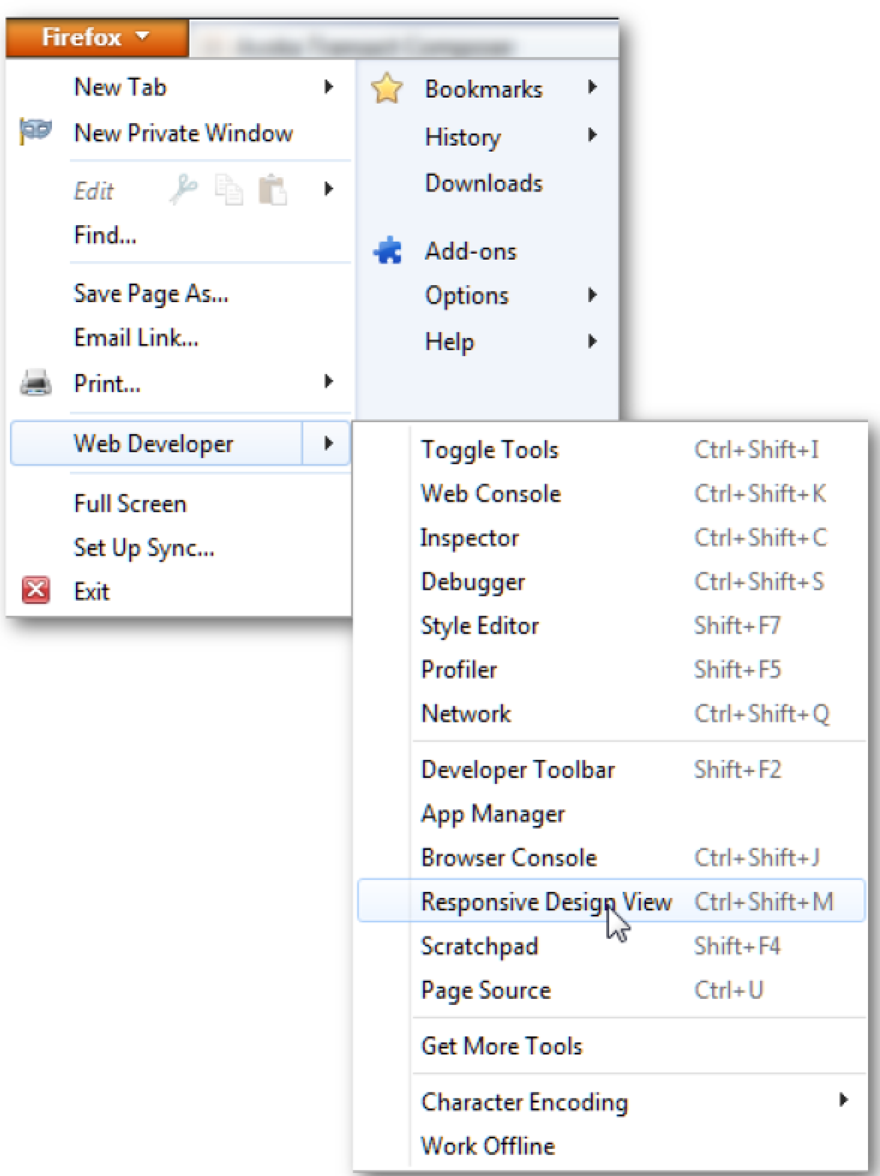
This explained in Renditions . The default setting for Composer is to have only 2 preview settings, "HTML [Responsive]" and "PDF". If you want, you can separate out the HTML preview modes into Desktop and Responsive through "Form Options -> Generation Options -> Edit Properties -> Properties -> HTML Generation
-> Responsive -> Responsive Mode [HTML]". There is really no need to do this in Composer 4; better adjust the width of the browser windowing displaying the HTML Preview and watch the effects as you move through the thresholds.
To test responsive behavior, select HTML preview in the dropdown menu, top left of the Form Designer:

? Unknown Attachment

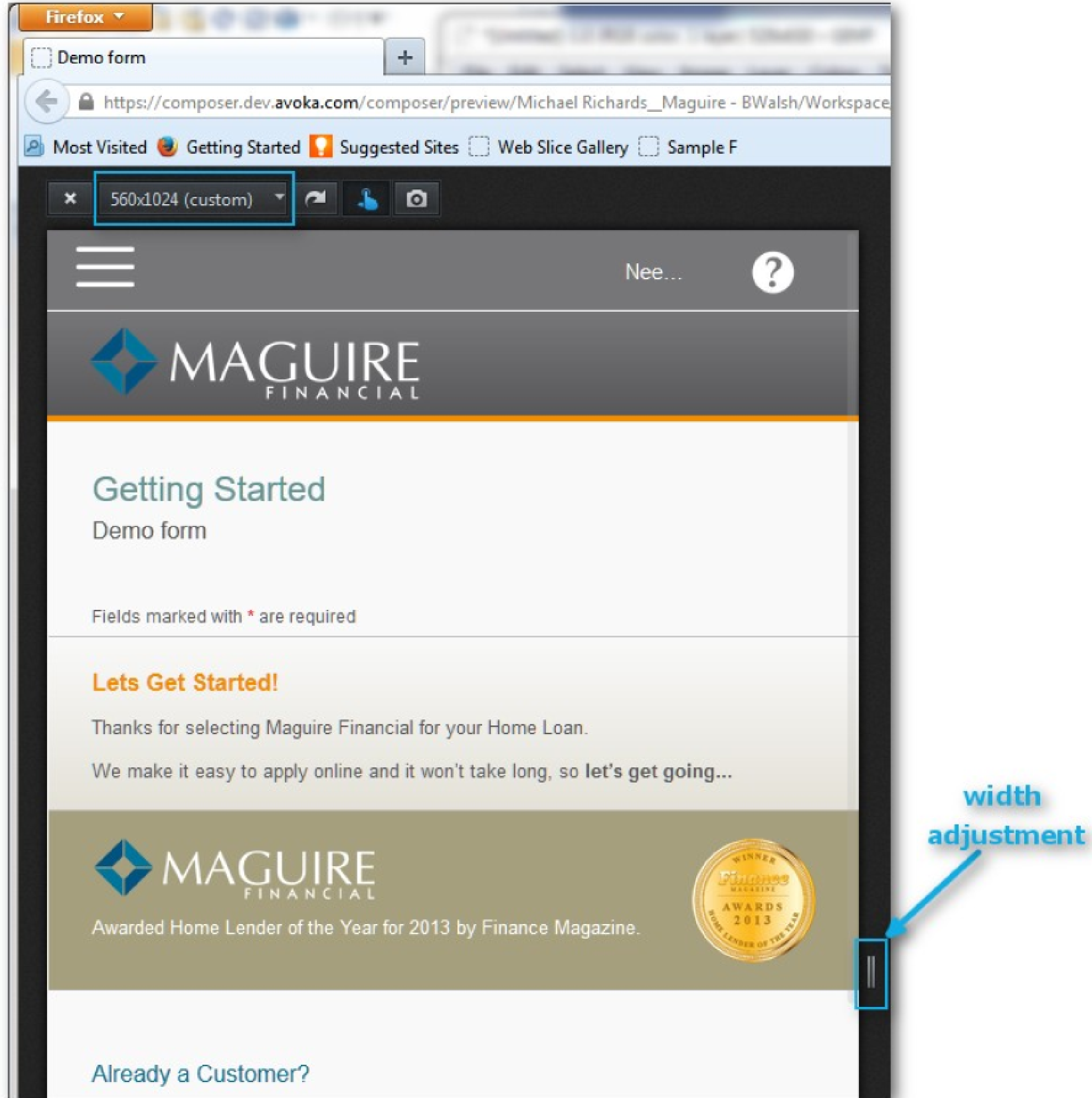
The preview will usually show in a new tab on your browser. To test responsive behavior, pull the tab out of its containing browser window, and make the new window re-sizable (done using the window control icons on the top right of the window in Microsoft Windows, or the top left candy-colored icons in OS X). As you narrow the window, you pass through the thresholds and doing so triggers the next stage of responsive layout where the layout of the form changes accordingly. The changes are quite dramatic:

? Unknown Attachment

Hint: Firefox has a convenient feature for viewing responsive behaviors. Other browsers do not have such a ready-made facility. You access it in **Windows** from the orange dropdown menu on the top left of the browser window. in **OS X** from "Tools -> Web Developer -> Responsive Design View" in the top menu.



This results in a read-out of the width of the rendered page:



Grid Layout Caveats

Many grid layouts do work in Responsive forms, but not all.
Grids with more than 4 columns may look unsatisfactory above threshold 3. You must preview your form above and below the various threshold widths.
All grid layouts are fine below threshold 3.
Adjust the "Size" settings or the number of columns in the grid for a workable solution.

Actions

In Composer, the Nuts & Bolts Responsive Rules (which, unlike previous versions of Composer, now depend on the threshold levels) are now read only. You cannot alter the Responsive Rules in the Structure Panel.
You can, though, point various elements in your form to whichever of these rules is appropriate for that form element (through the "Edit Properties -> Rules -> Responsive -> Responsive Ruleset [HTML]" dropdown menu).

Wrap

The "wrap" action takes inline fields and, when responsive behavior kicks in, places each of these on separate lines.

Wrap and fill

Inline fields are placed on separate lines and all fields stretch across the width of the display.
Note: tables **must** be set to "wrap and fill" or they will not lay out properly in the narrower thresholds.

Fill

The "fill" action should only be used in Wizards with left-hand menus. For the effect, see the iPad rendering o f the example Wizard above. Otherwise, the "fill" action only has meaning for responsive behavior for objects that require special scripting.

Hide

Some elements should not be shown on mobile devices as their presence is inappropriate for the layout on that device.

Hide (no media support)

Some elements should not be shown on mobile devices, due to their fixed space requirements or their relevancy.

Show

The opposite of "hide". In other words, the visibility of the form objects allocated to the show responsive rulesets is determined by the device's position among the 3 thresholds.

In Line Validation Methods

See Validation . "In-line validation" means that users receive messages as they proceed filling out the form, rather than receiving a message when they reach the end of the page or form. In-line validation is of the utmost importance on mobile devices, as they reduce user friction if their entry in a field does not pass validation. The last thing they would want is to be told at the end of filling in a multipage form that they have to reverse back through the form, especially in a touch environment where doing so can be difficult and frustrating.

Tool tips

Touch interfaces have a few differences with the desktop mouse-dependent user environment. In touch there is no concept for hovering over an object or text. Instead, users can hold their fingers over some text or object, and other functions of the user interface come into play, such as the "magnifying glass" cursor control for moving through text, or activating the copy and paste controls. Tool tips have no meaning at all in the touch- driven world of mobile. You must therefore not rely on tool tips to guide users through the filling process.

Submissions and Attachments (Composer v4.3)

So far, we have discussed how to use the Form Designer , Navigation , and Layout , all of which are meant to lead to submission. When users fail to complete submission of the form, the form is said to be abandoned. This is a most unwanted result and a waste of time for everyone.
Users are often asked to upload files, called attachments, as supporting documentation. Most attachments are generated outside of the form, and can be scans, Word documents, PDFs, JPEGs, PNGs and so forth.
However, there are form elements capable of generating attachments from within the form, for example: scribble pads for users to make their signature camera widgets for uses to take pictures with a mobile device's in-built camera barcode-reading widgets for use with a device's camera.
Defining attachments (i.e. files uploaded with the form) used to done exclusively in Transaction Manager. Now these can also be defined in Composer.
The Preview function of Composer does not include user interactions when they upload attachments; for that, you will have to publish to Transaction Manager
.

File Uploading

Defining Attachments in Composer

There are two attachment widget types. You add all of these to the one section on the form, here called "Attachments". It makes sense to have these in the last section of the form which usually contains the "Submit" button.

TM Attachment Table

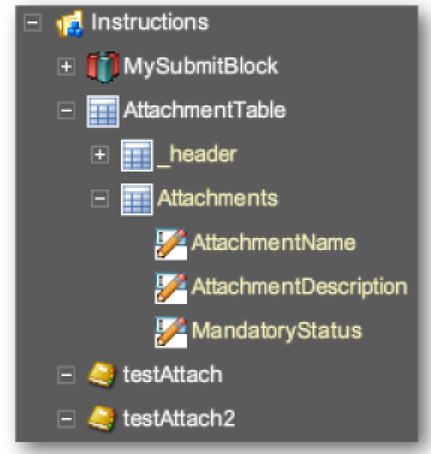
Add one only of these, and its rows will automatically be populated with the details of the attachments.

TM Attachment Rule

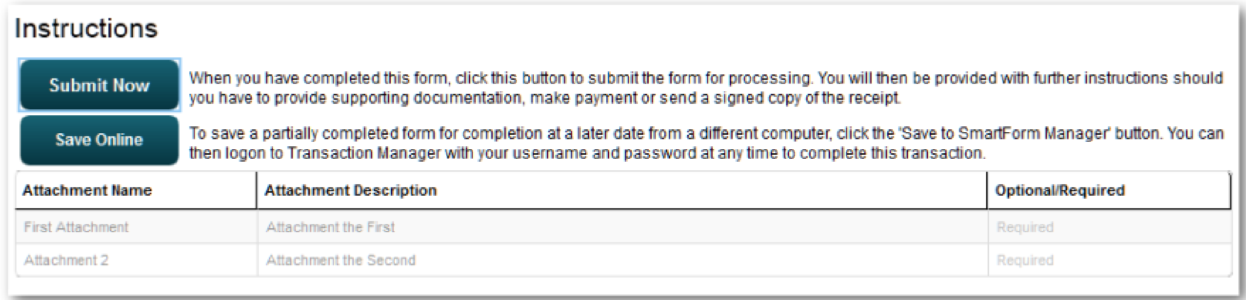
Add one of these per attachment at the same structural level as the TM Attachment Table. Use the "Create a New Field" wizard, which pops up on adding the rule to the form, to give it an "Internal Name". You can also further specify each attachment's properties in the rule's "Edit Properties" dialog.

TM Submission Block

You also need submit buttons, so we have added a TM Submssion Block (as "MySubmitBlock") to the section. This block adds several submission buttons to the form, which give users the options to "Submit Now", "Save Online" or "Save to My Computer:"
Here is the resulting structure of adding a "TM Attachment Table" and two "TM Attachment Rule" widgets to a section containing a "TM Submission Block".

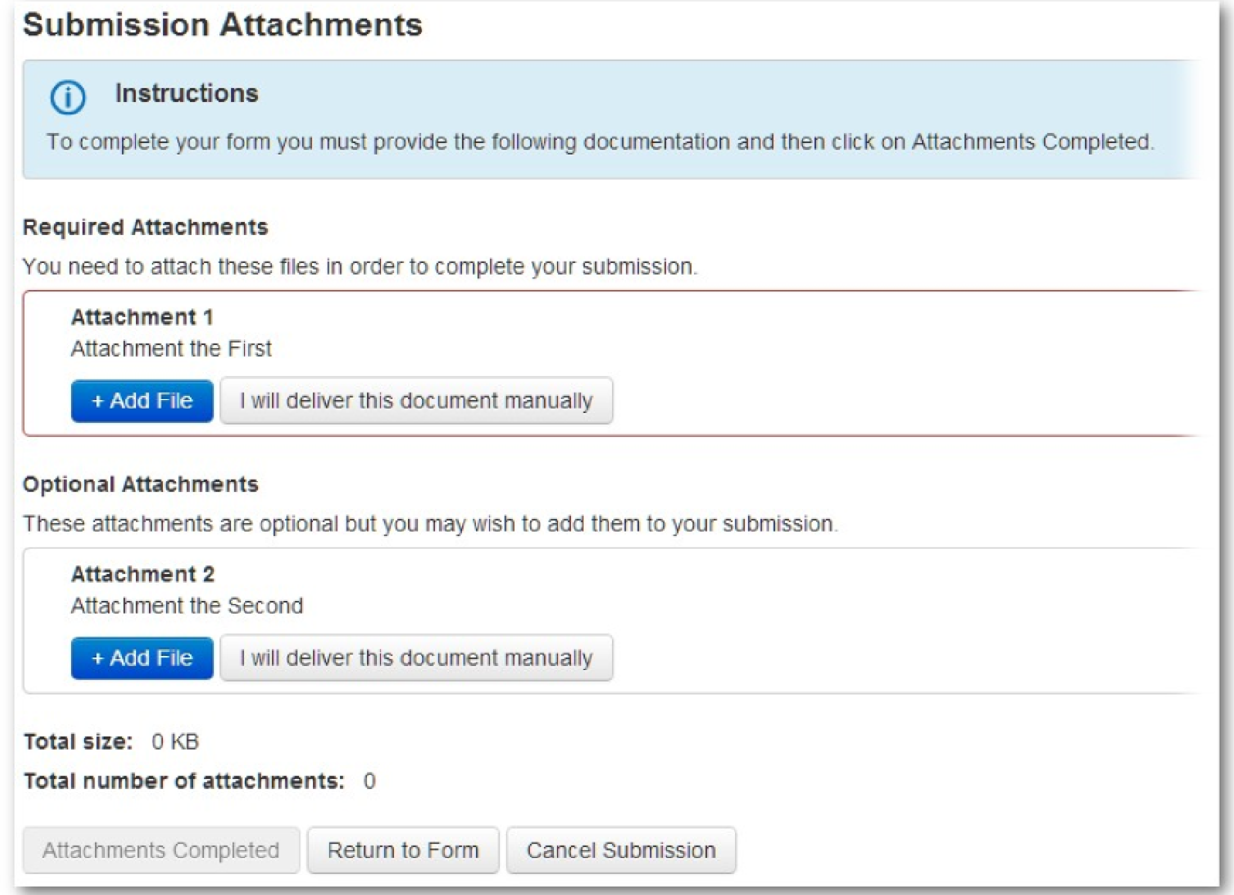


You can then preview the rendition of the form, where the Attachment Table gets populated according the the Attachment Rules:



How this structure renders in preview.

The table informs users that they will be requested to supply these attachment after clicking on the "Submit" button. This behaviour cannot be previewed in Composer: to see this, the form must be published to Transaction Manager. In our example, the form was published without further modification in Composer; all that was done in TM was to assign the form to a portal and to turn off its "Test" mode. The resulting screen after the "Submit" is:



Resuling "Submission Attachments" page.

Delivery of Attachments

The "I will deliver this document manually" buttons appear on the "Submission Attachments" by default for each attachment. To remove them, set in Composer the "Edit Properties -> Data -> Submit Method" for each rule to "Electronic".

Email Acknowledgements

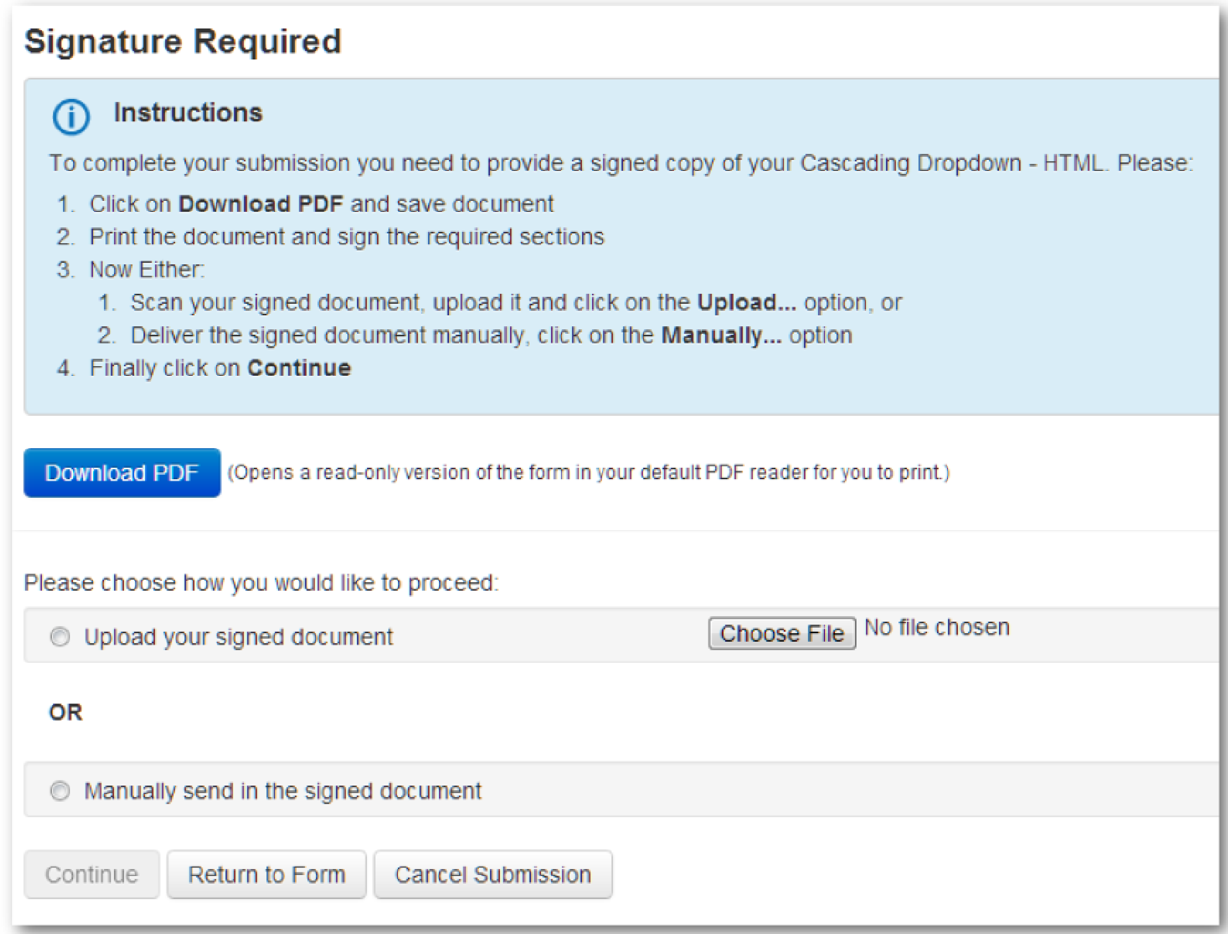
If you require an email to be sent to the user as an acknowledgment, receipt or any other business reason, this must be set-up in Transaction Manager. There is currently no way to specify emails through Composer. Transaction Manager, though, has a great deal of functionality and control of email messages, Please refer to the Transaction Manager Administration Guide.

Signatures

There is a range of signature widgets:
Signature [Wet] and TM Signature [Wet]
Dropping these on to the form has the same effect. Pre-submission, the form presents the user the message, contained in "Edit Properties -> General -> Instruction Set". The default is:

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

After users have clicked "Submit" they are presented with:

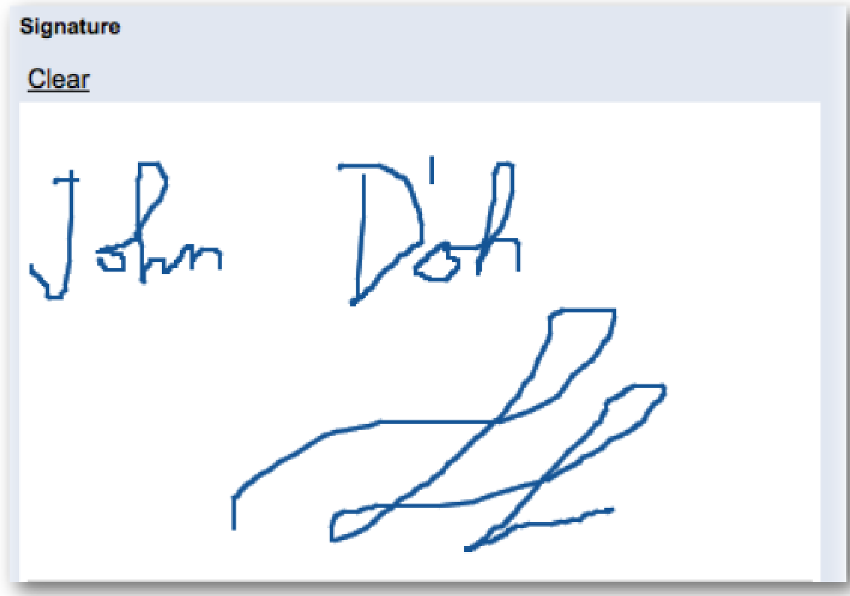


Wet signature form after submission.

DocuSign Signature
Creates a block to handle the DocuSign eSignature process. The users fill in their names and email addresses and DocuSign verifies the users via a two-step process. **Note:** use of the service incurs a service charge.

Signature Pad Widget [HTML]

The widget's "Edit Properties -> Properties -> General -> General -> Signature Type" can be set to "Draw It", "Type It", and "Choose at Runtime". "Draw It" gives users a signature canvas to sign with their finger or mouse (see below). "Type It" presents users with a text field.



The signature canvas.

Signature Pad Popup [HTML]
Similar to the signature pad above, except that the pad appears in a separate lightbox superimposed over the form. On "Save", the lightbox vanishes and the signature shows as a thumbnail.

Notes

Use a true Digital Signature if users have a certificate or smartcard available.
Allow the end-user to click a "Sign" button, and specify their username (and the date) if they don't have a digital signature.
In HTML, just use their username and the date.
If in Adobe Reader, and the form is not "reader extended", just use their username and date.
The name and date from a true digital signature will also populate the form's username and date, so in HTML even if users re-use the form, we still have a record of the signer.
When either the true Digital Signature or the Signature field is signed, an invisible checkbox is checked. This will allow standard editability rules to be defined so that the form can be locked down after signing.

Dynamic Data (Composer v4.3)

Overview

In Composer, Dynamic Data has two meanings:
Dynamically changing the information displayed on a form without reloading the page Accessing a third party web service for data for this dynamic display

What is Dynamic Data?

Dynamic Data is an Avoka Transact capability that allows a form to communicate back to the Transaction Manager Server while the form is still being filled in and even display new data on the fly, without the page being reloaded or the form being submitted. This allows an additional level of functionality to be built into forms.
Dynamic Data is a general purpose communication mechanism and can be used for many reasons. Common use cases are:
Sending some data from the form to the server (such as a user id) and retrieving some information to populate fields in the form (such as their address).
Sending some data from the form to the server (such as a user id) and retrieving some information to populate a pick-list in the form (such as a list of the subjects that the user is enrolled in). The pick-list can be in the form of a drop-down list, table, or other visual element.
Sending some data from the form to the server to perform a server-side calculation. Sending some data from the form to the server to perform a server-side validation.
Dynamic Data requires collaboration between the form developer, and a server-side developer who defines a Dynamic Data service in Transaction Manager. Please refer to the Transaction Manager document for more information on this.
Composer provides a number of different facilities for using Dynamic Data:
Widgets that have pre-built Dynamic Data support, such as the "Lookup" Button. An assistant that builds lookup-style Dynamic Data capabilities .

Dynamic Data in Action

We recommended you initiate Dynamic Data invocations by a direct user actions, like a button click, not from implicit actions, such as exiting a field. Direct actions set user expectations that "something time-consuming should now happen, and may even fail." Internet performance and reliability is beyond our control, so using a button sets end user expectations that some action is about to occur.
However, it is also possible to use Dynamic Data to perform "predictive" lookups, such as the list of completion hints seen while typing in a Google search . The AutoSuggest example below shows these predictive lookups using the Veda Geolocation service.

The Dynamic Data Assistant

By dropping the Dyanmic Data Assistant onto the form, and filling in a few details, you can populate your form with a dynamic data fields in a few steps by merely filling in the "Add A Dynamic Data" dialog.
In this example, the Transaction Manager service definition is "ACMEPOST", which in real life would access some webservice from a third party provider. Here, there is only one input field, "Suburb" and the TM service returns multiple sets of 3 fields each.
The "Result Block Type" selector toward the bottom of the dialog gives 3 options:

Standard Block

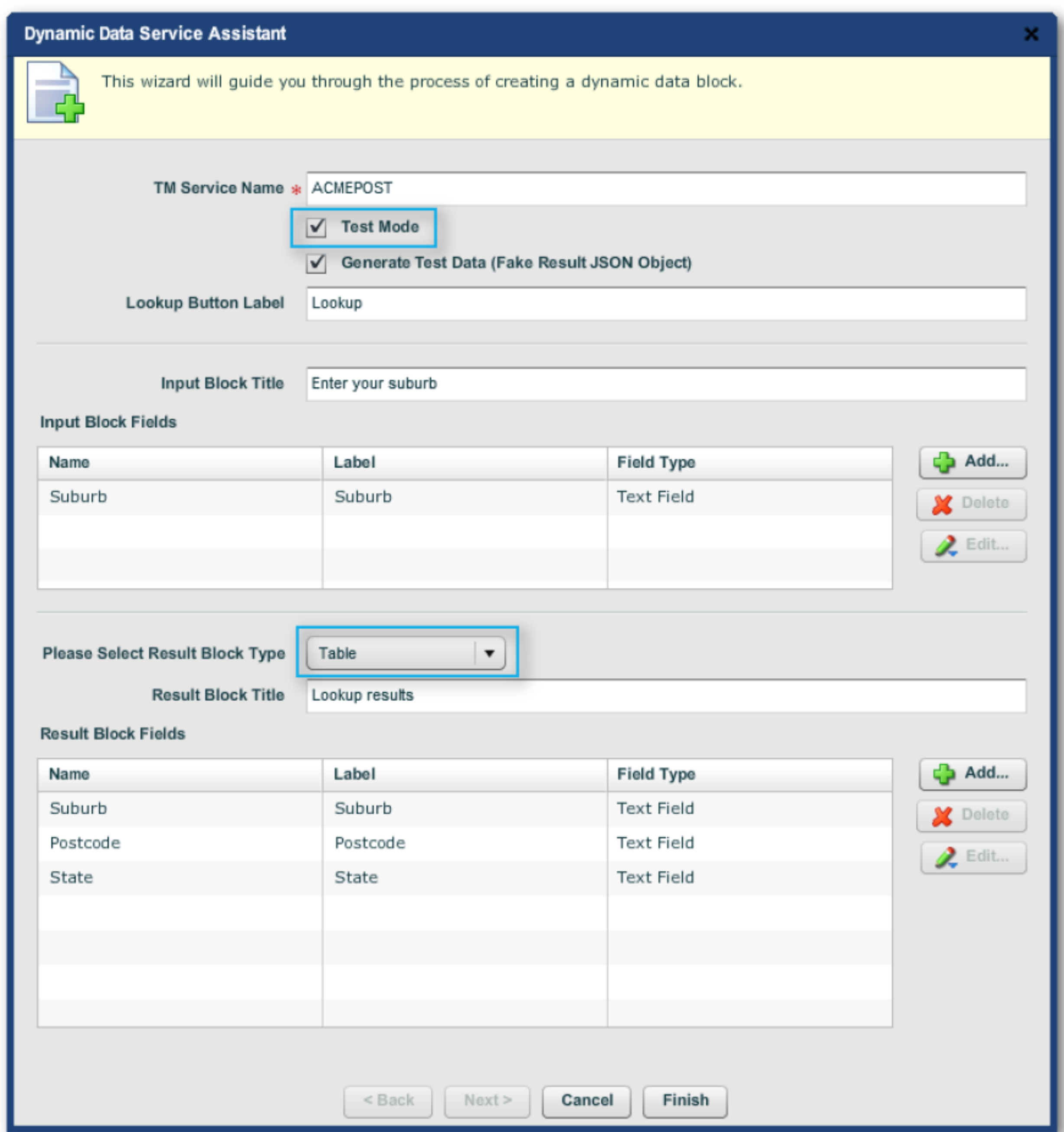
which will display only one set of returned data

Repeatable Block

which will display multiple sets of data in the more flexible format than a table structure

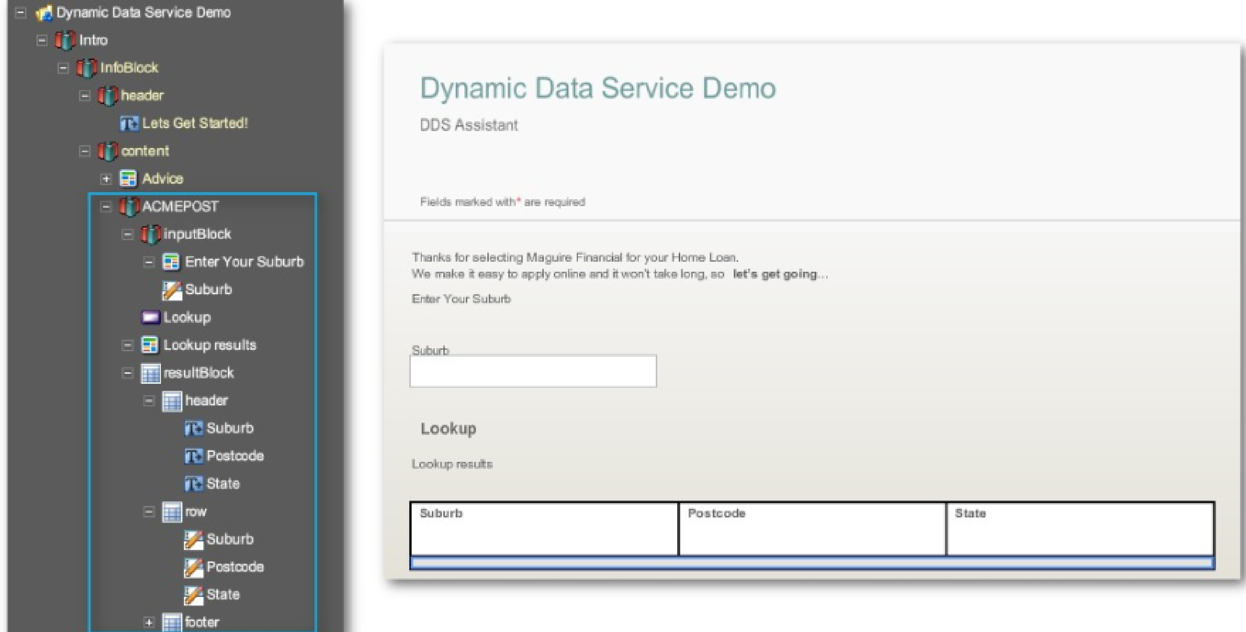
Table

as selected in this example. Each set of returned data will populate a row of the table.
We have also activated the "Test Mode" checkbox. The assistant will then provide a set of fake data automatically. You can also provide your own fake data, as explained below. Of course, for the form to work from real data, the service must already be defined in TM.



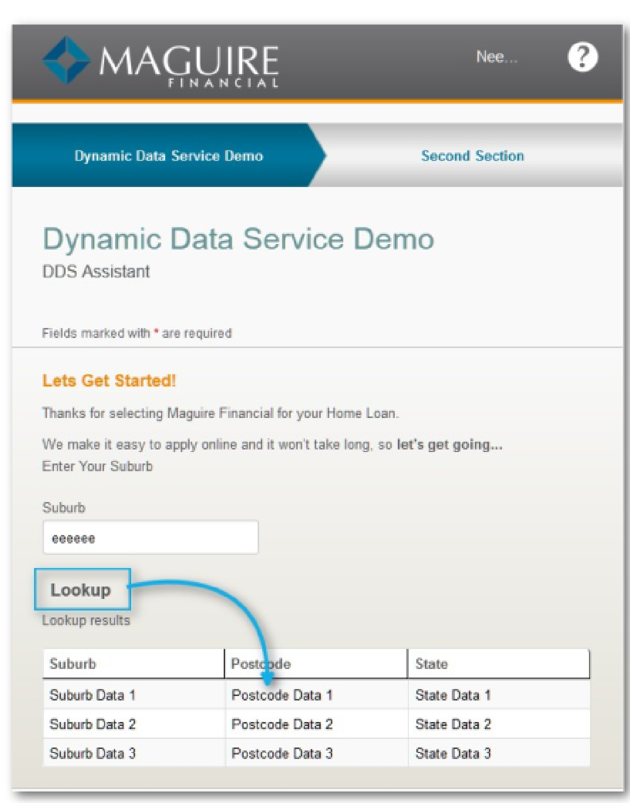
The Dynamic Data Assistant wizard dialog.

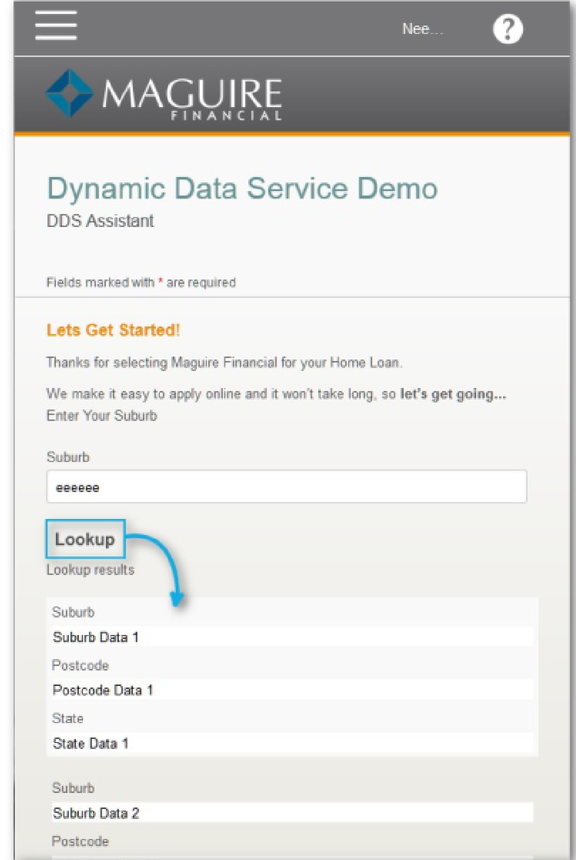
This dialog results in this wireframe and structure:



Dynamic Data Assistant creates the following strutcure and wireframe.

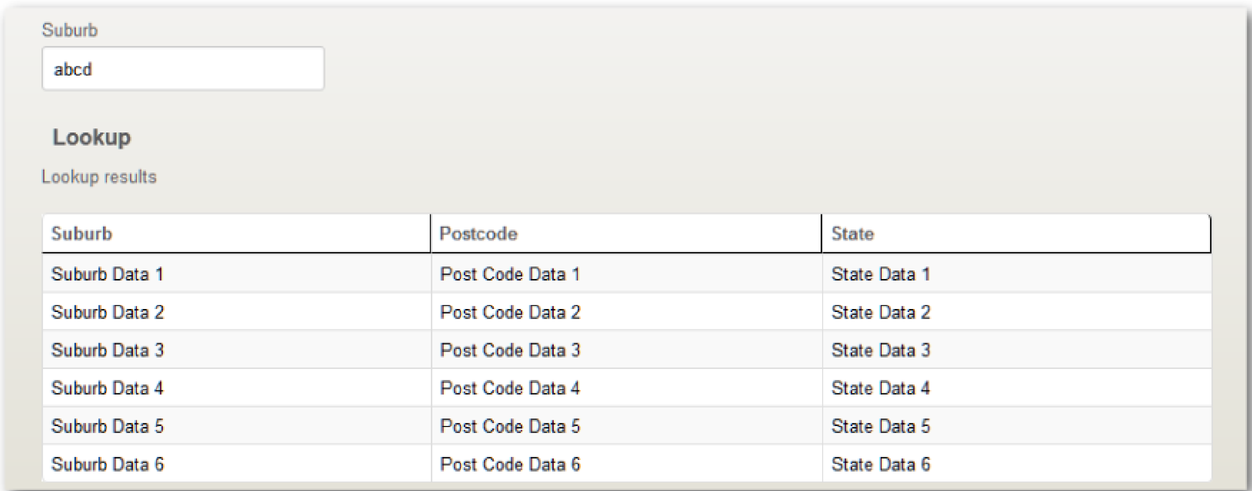
Previewing the form in HTML and clicking on the "Lookup" button results in the following display of the default fake data generated by the assistant.





Preview of the form with fake data generated by the Dynamic Data Assistant, desktop vs narrow mobile

The fake data belongs to the structure's button, here called "Lookup". Going to the button's "Edit Properties -> Properties -> Data -> Dynamic Data Testing" exposes a text field called "Test Data (Fake Result JSON Object)". If you wish, paste your own JSON data here. If the "Result Block Type" is either "Repeatable Block" or "Table", all the JSON fake data will display; if not, only the first set of the fake data will display.

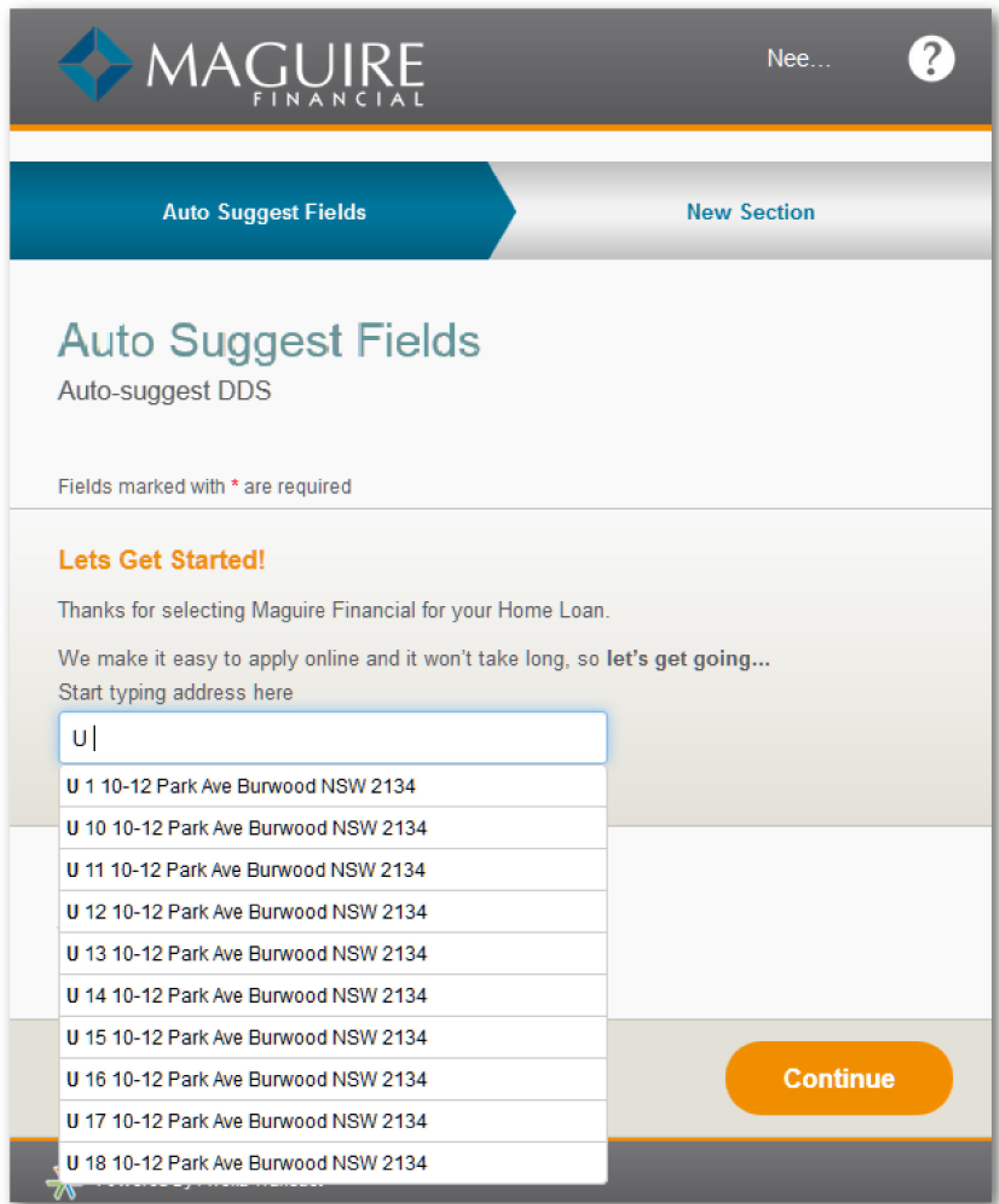


When the service is returning an unmanageable number of hits, consider using pagination .

AutoSuggest

Overview

An auto-suggest field is a text field (and its associated control widget) where the rest of contents of the field are suggested to the user in a popup below the field, and the user selects the correct information. The suggestions are obtained from a dynamic data service (DDS). A common use case for auto-suggest is entering a household address with suggestions coming from a geodata DDS.
Here is how the field looks and behaves:

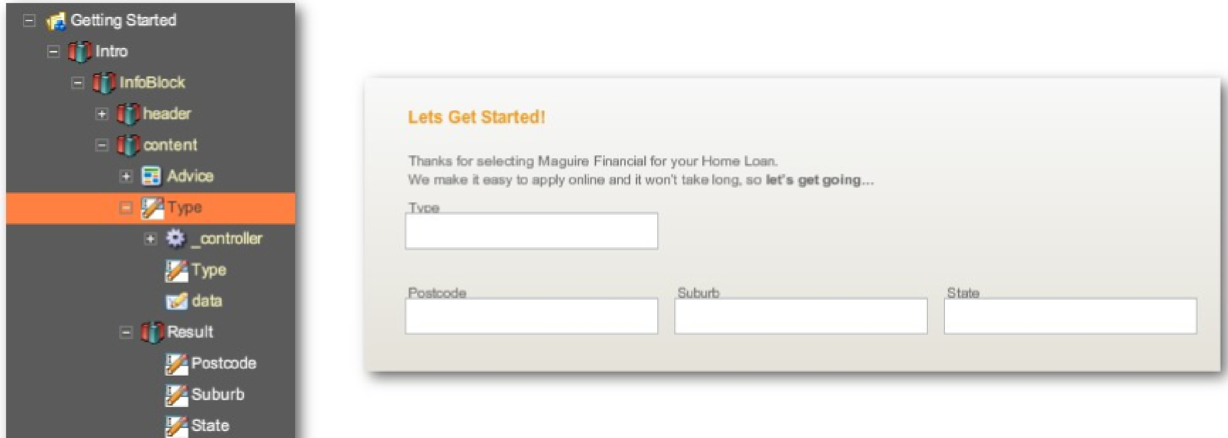


For the autoSuggest widget to work, the widget configuration has to correspond to the Groovy scripts of the service in the "Service Definitions" page of Avoka Transaction Manager. In other words, unlike most widgets in Composer, the necessary service creation and scripting must already exist in Transaction Manager before the widget can be added to the form.
The following discusses:
Building an autoSuggest form element from scratch, with a service we have defined as ACME POST in Transaction Manager
Using predefined widgets and scripts that ship with both Composer and Transaction Manager for the VEDA GeoCoder webservice
For further detail on the configuration parameters, refer to the notes below .

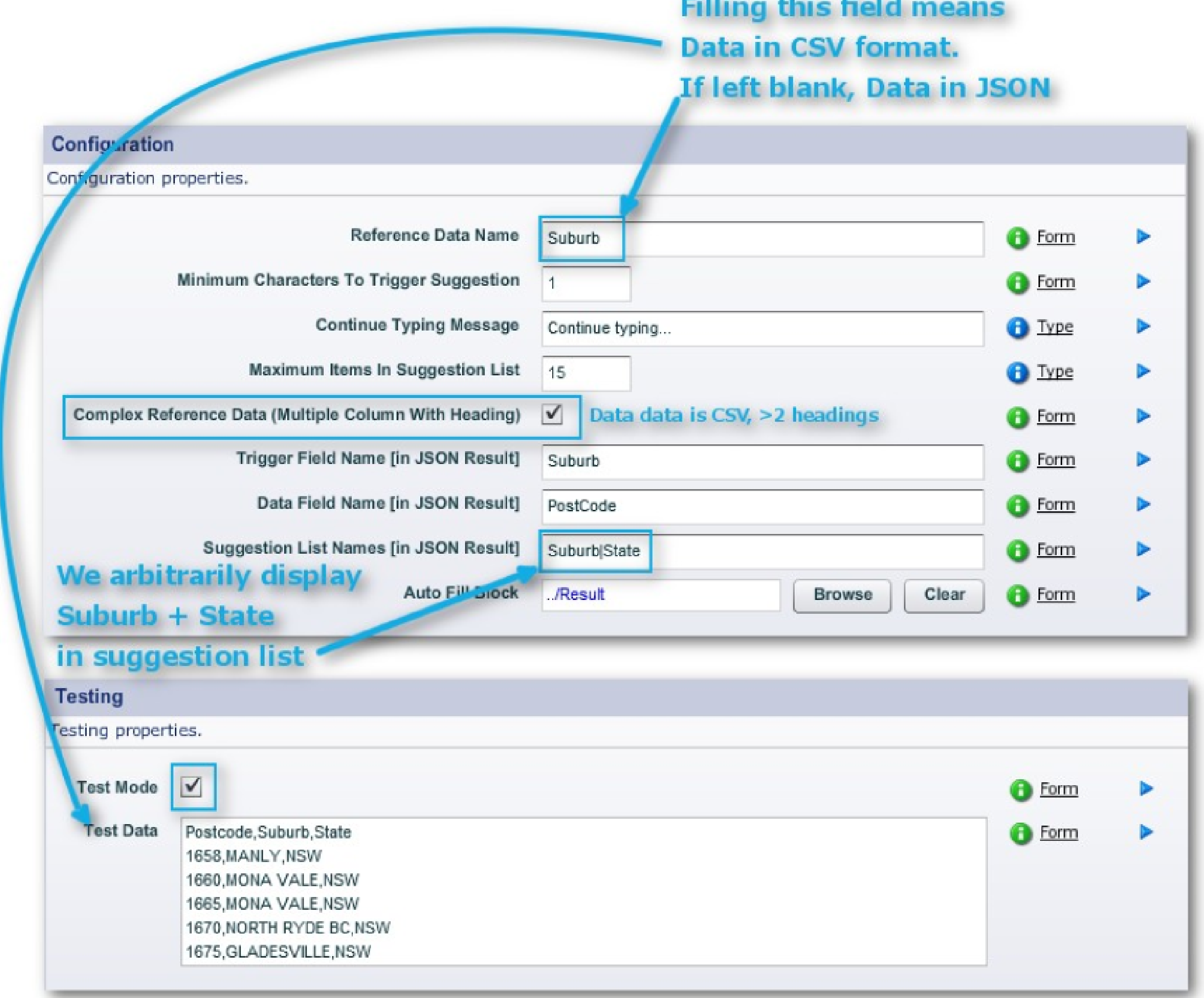
Reference Data Lookup Block

CSV Data With Headings

Another widget, the TM Reference Data Lookup Block, also provides lookup data. The aim of this widget is to give a convenient way to provide an auto suggest function with no programming. The reference data can be an Avoka Transact dynamic data service defined in TM (see the Transaction Manager Administration Guide), or from prepopulated data (see here for an example of how you introduce prepopulated data onto a form using a Data Field widget). In this example, we use test data.

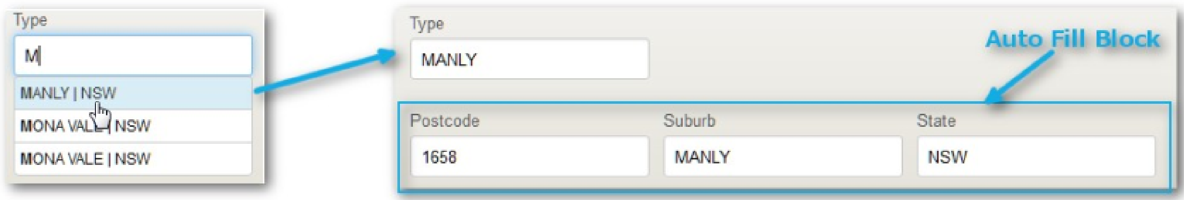


The only change we made to the structure built from the block was to build the "Result" block (3 in-line text fields named according to the table headings of the CVS test data) and point to it in the "Configuration" panel. Note the "Complex Reference Data" checkbox: it is activated because the data is a table with more than 2 column headings.



Panels in "Properties -> Data" for the "Type" node (a TM Reference Data Lookup block)

Which results in the following behavior:



CSV Data Without Headings

You can have only two columns of data, that is only one pair of data points per row separated by a single comma. Make sure the Complex Reference Data checkbox is unchecked.
So, as there are now no headings, the convention is that the left data point or column is refered to as "display". The right data point or column is called "data". The default parameters of the Configuration Panel is to have the left column, "display", as the "Trigger Field Name"; the right is the "Data Field Name".

Notes on Auto Suggestion _controller Data Configuration Parameters

The following tables describes some of the "TM Auto Suggestion Controller" parameters and gives the default abstracted values provided off-the-shelf in the "TM Veda Address Lookup Block [Australia] -> _controller". If you want to change these values, you should do this in the TM Veda Address Lookup Block [Australia], not in its children.
Note: you still nominate to use test data in the "TM Veda Address Lookup [Australia] -> _controller -> Data -> Test Mode" checkbox.

AutoSuggest _controller -> Properties -> Data -> various panels	Parameter Description	VEDA Controller Abstracted Value
Dynamic Data Service Name	Points to the DDS	$../{dds.service.name
Minimum Characters to trigger auto suggest	The number of keystrokes before the auto-suggest popup appears	$../{dds.min.chars}
Continue Typing Message	A message displayed in grey in the trigger field while the user has still not made a selection from the suggestions presented	$../{continue.message}
Not in List Message	Displayed at the bottom of the suggestions if the typed input in the trigger field is not an exact match to one of the suggestions presented. The user clicks on the message to leave the trigger field. What then happens is determined by the "Clear Trigger Field When Not Matched" checkbox	$../{notInList.message}
DDS Fail Message	Displayed if the service times out. This is usually left blank.	[blank]
Clear Trigger Field When Not Matched checkbox	If the user leaves the trigger field, and the entry does not match any of the suggestions, the field becomes empty. Otherwise, the un-matching entry will remain. The "Not in List Message" will be displayed.	[checked]
Second DDS Service Name	Enables another layer of dependent data to be accessed from a DDS service and receive more auto suggestions in a secondary field. It follows that there are secondary input fields, and that test data can also be pasted into another Test Data field.	$../{dds.detail.name}
Timeout	specifies number of milliseconds before the DDS access fails. This value is usually left as "0".	0
Delay of Triggering DDS (in milliseconds)	sets the delay before the user input triggers a query on the DDS. If this is set too short, there will be usability issues.	200

"Hide Standard Address Block" in VEDA Block Widget

This is a special checkbox in "TM Veda Address Lookup Block [Australia] -> Properties -> Data -> Veda Configuration": the "Hide Standard Address Block". Its behavior is as follows:
When **checked:**

- Initally only the trigger field is visible and the address block invisible,
- If the user selects one of the suggestions, the filled-in trigger field only is visible
- If the user selects "My address is not listed", the address block becomes visible and the focus moves to "Address Line 1".

When **not checked**:

- Initially, both the trigger field and the address block are visible
- Initially only the trigger field is writable
- If the user selects "My address is not listed", the address block becomes writable and the focus moves to "Address Line 1".

In **both cases**:

- The trigger field will be cleared, regardless of the setting "_controller -> Clear Trigger Field When Not Matched".

Sequence Numbers

Sequence numbers are enabled by these lines at the end of the service definition Groovy script in Transaction Manager:
// See if there is a queryString holding a sequence number def value = request.getParameter("seq")
if (value == null || "".equals(value)) {
// The request parameter 'seq' was not present in the query string OR
// The request parameter 'seq' was present in the query string but has no value return result
} else {
// add the sequence number back on the result
result += new JSONSerializer().toJSON("{'seq' : '" + value + "'}") return result
}
If there have been a number of requests made on the DDS, the responses may not be returned in the same order as the requests. If sequence numbers are being emitted in response, the value of seq is incremented with each response, allowing TM to tell if responses came back out of sequence. TM will then only act on the most recent response, that with highest seq number.

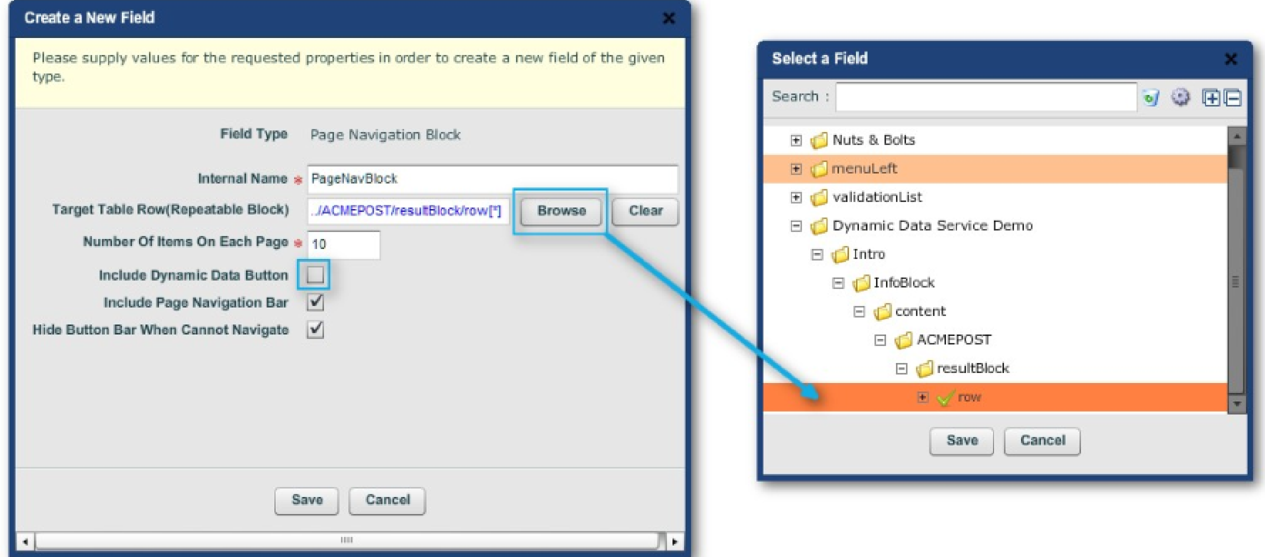
Internet Explorer 8 Issue

End users on IE8 can receive multiple alert dialogs. The workaround for this is setting the "TM Auto Suggestion Controller -> Edit Properties -> Rules -> Click -> Failure Notification" and choose "Do Nothing".

Pagination

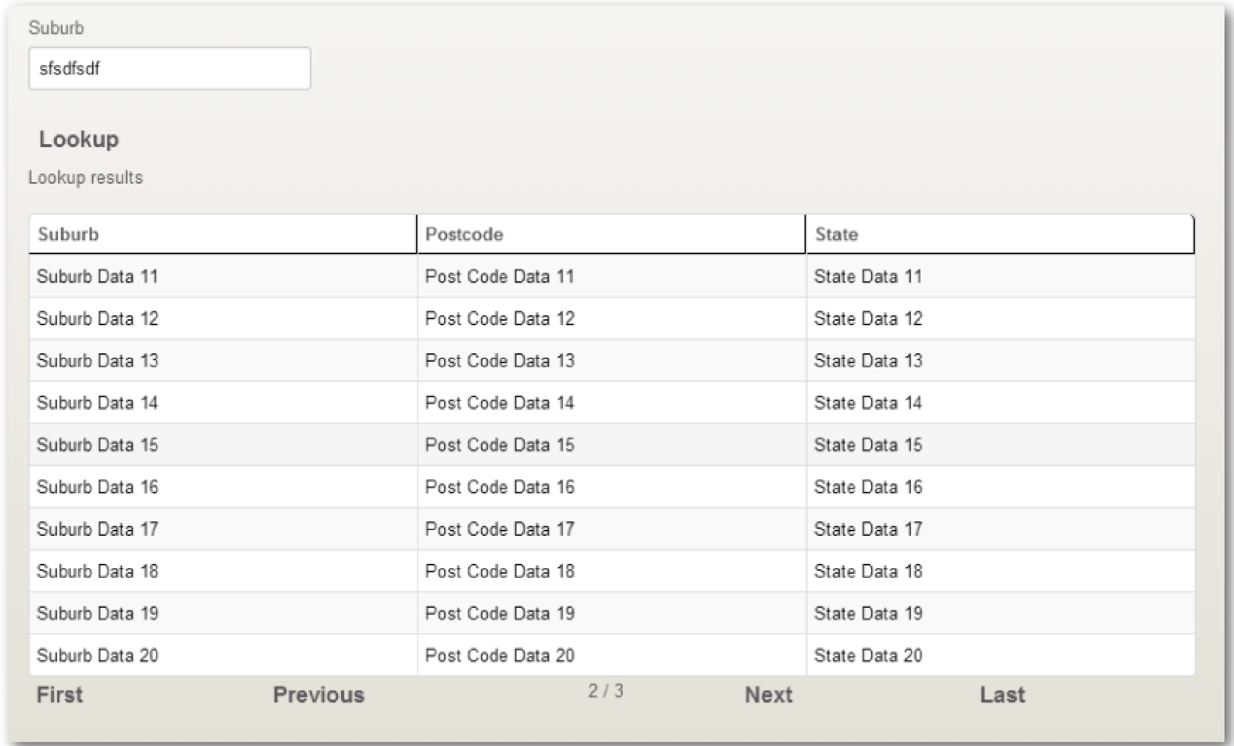
Composer now has a "Pagination Block" widget (currently specifically for Dynamic Data using a "Lookup" button). It is very simple to add to a form using a Dynamic Data Service with a table in the result bock:
just drop in a "Pagination Block" onto the form
point the lookup button of the input block of the dynamic data service to the page controller of the Pagination block
point the page controller of the Pagination block to the lookup button of the input block We illustrate this by adding pagination to the ACMEPOST example .

Add from the Palette, a "Field Types -> Pagination -> Page Pagination Block" element. Fill in the resulting dialog:



Adding a Page Navigation Block

Note: the "Include Dynamic Data Button" in the Page Navigation Block is not activated in this example: there is already a button in the ACMEPOST Dynamic Data block.
And. *vice versa*, you point the Dynamic Data Service back to the page navigation structure, via the ACMEPOST Lookup button's "Edit Properties -> Properties -> Data -> Data -> Page Navigation Controller".
There must be enough rows of fake data in the result table to activate the pagination functionality. If so, the preview looks like this:



2nd page of the page navigation output (using fake data).

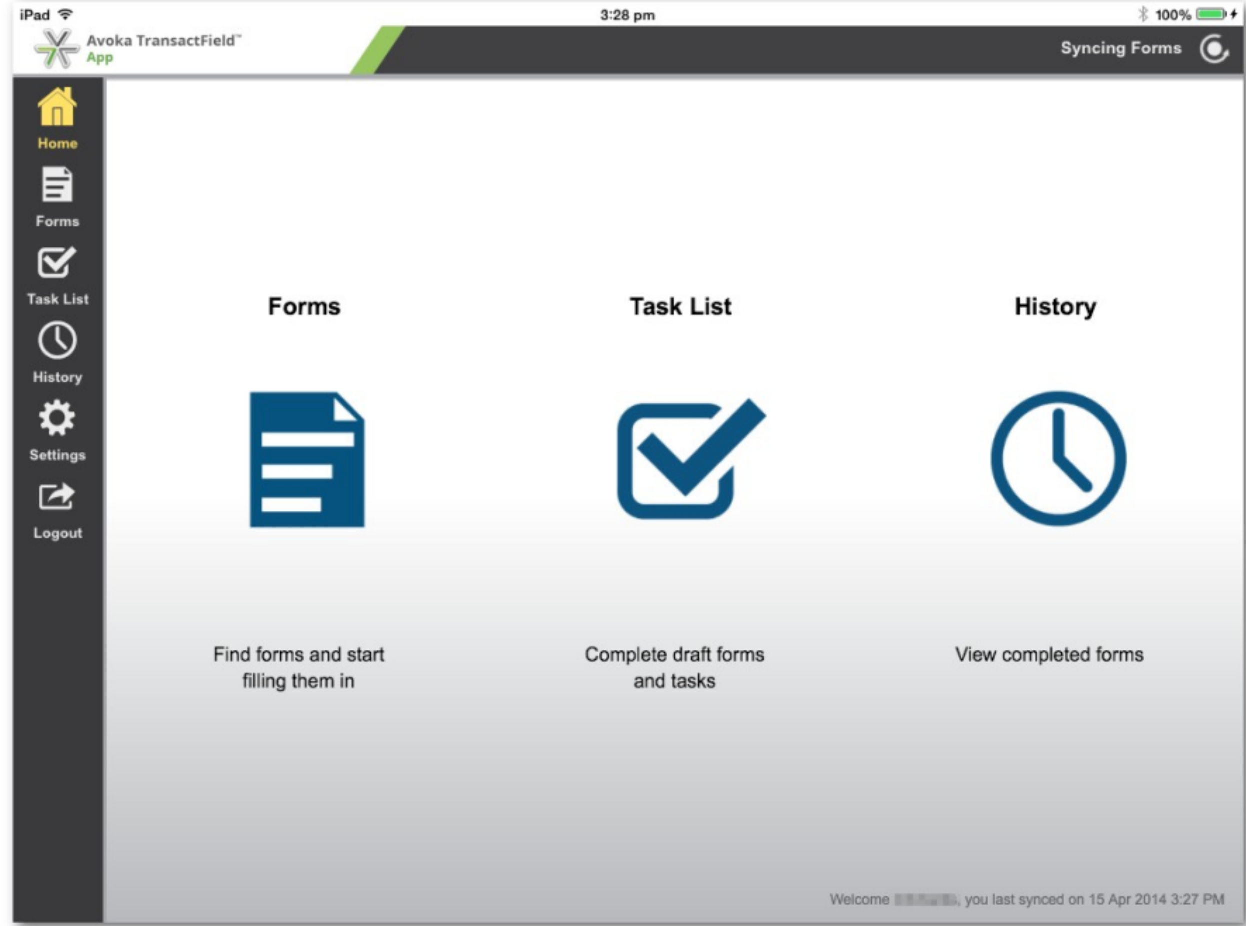
Please refer to the Responsive Layout section about viewing the result table on mobile devices. You should give some thought on how you want the table to appear in the responsive layout thresholds , where each table cell gets its own line instead of being laid out in rows.

TransactField Mobile App (Composer v4.3)

Avoka Technologies offer a couple of ways for users to access your forms on mobile devices:
Through the device's web browser, with Responsive Layout technology
Via a native TransactField App.

About the App

The Avoka TransactField App runs on iPads, iPhones, Android tablets and phones, and Microsoft Windows devices.



The Avoka TransactField App's main user interface

It delivers to these devices intuitive mobile data-capture applications that are built using the Avoka Transact platform. The TarnsactField App client affords the following capabilities to mobile workers:
Allows field workers to take their forms with them out in the field
Gives field workers a task list of forms to complete or approve and a timeframe in which to do these. Synchronizes forms automatically to the device so that they can be filled out online or offline
Holds submitted forms temporarily on the device, completing the submissions when reconnected Enables forms to make use of device capabilities, including camera , GPS , and sign-on-glass Enforces security, including password protection and encrypted data on the device and over the wire Provides full control over which forms go to which field workers through permissions and roles Synchronizes to all apps when a new form or an updated form is deployed.
Please refer to the Transact Mobile App Guide for more on these functions. There is very little to do in
Composer to utilize these: they are taken care of by TM and by Composer's scripting.

How to Publish a Form to the Mobile App

Use the following steps:

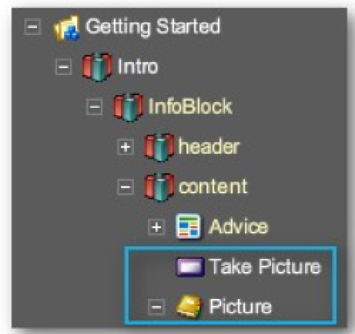
1. Save the form and publish it to TM
2. In TM, save or import the form
3. In TM at "Forms -> Form -> Details", uncheck "Test Mode
4. In TM at "Forms -> Form -> Portal Access", make sure "Mobile App" is one of the "Assigned Portals".

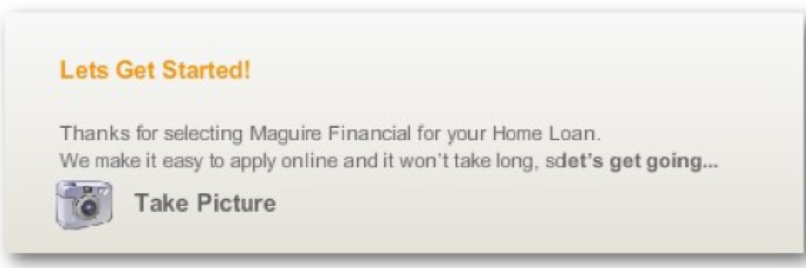
Now, when you open the mobile app on a device (making sure that you are connected to the correct instance of TM) you should — after the synching process is completed — see the form by tapping on the "Forms" icon and be able to select it in the forms list.

Mobile Camera Support

Mobile devices came with in-built cameras, and so users want to be able to integrate this picture-taking capability into forms. Users tap on a form's "Take Picture" button widget and generate an image attachment. This gets uploaded when the form is submitted.
Though the attachment generated by the button is like other attachments, the Button [Take Picture] widget is unique in that:
It only works in the TransactField App
It is not supported on the desktop or in a mobile device's web browsers

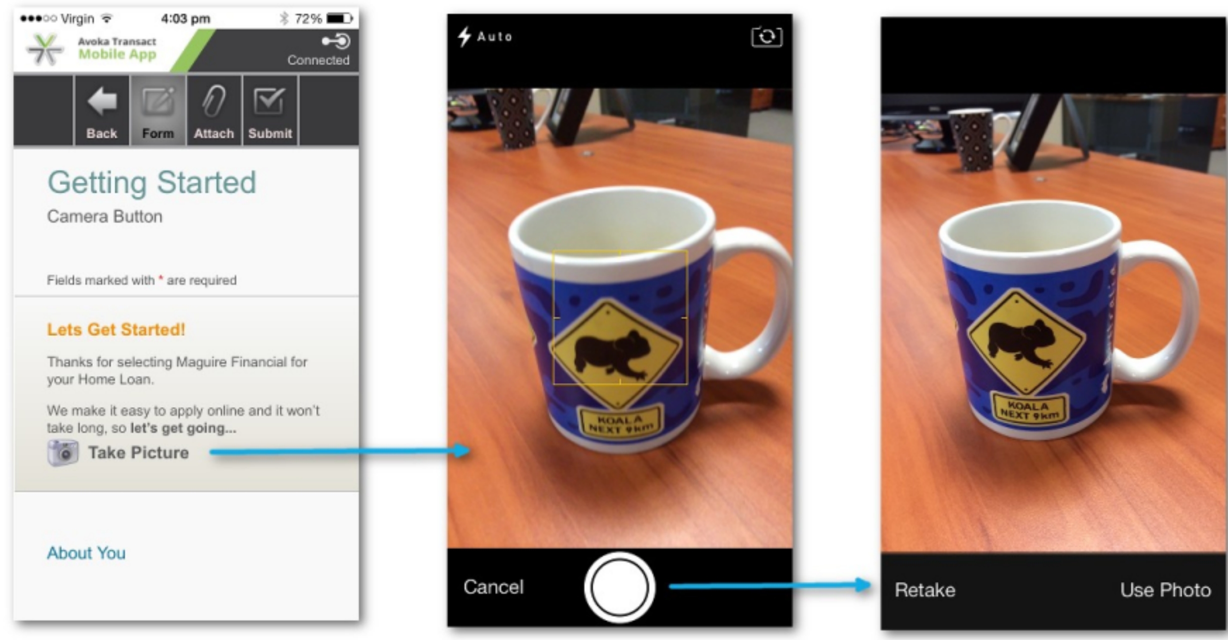
The Standard "Take Picture" Button

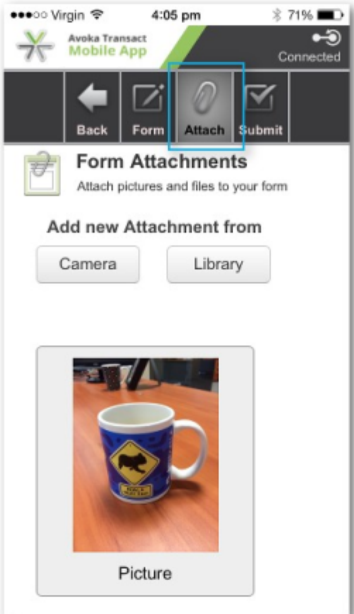




Very simple form with a camera button

The "Button [Take Picture]" widget works only in the Mobile App. Tapping on it takes mobile device users to the in-built camera app. The user gets to take the picture and choose either to retake or use the photo. It is not saved to the device's camera roll. For the sake of appearance, we have set "Take Picture -> Edit Properties -> Styling -> Style Sets -> Button Type -> Push button".
The form also needs a TM Attachment Rule widget to activate the Attach tab in the Mobile App; we have named this element "Picture" in the structure. Without it, no attachment file will be uploaded to Transaction Manager.
The resulting form behaves as follows in the Mobile App (on a smartphone):



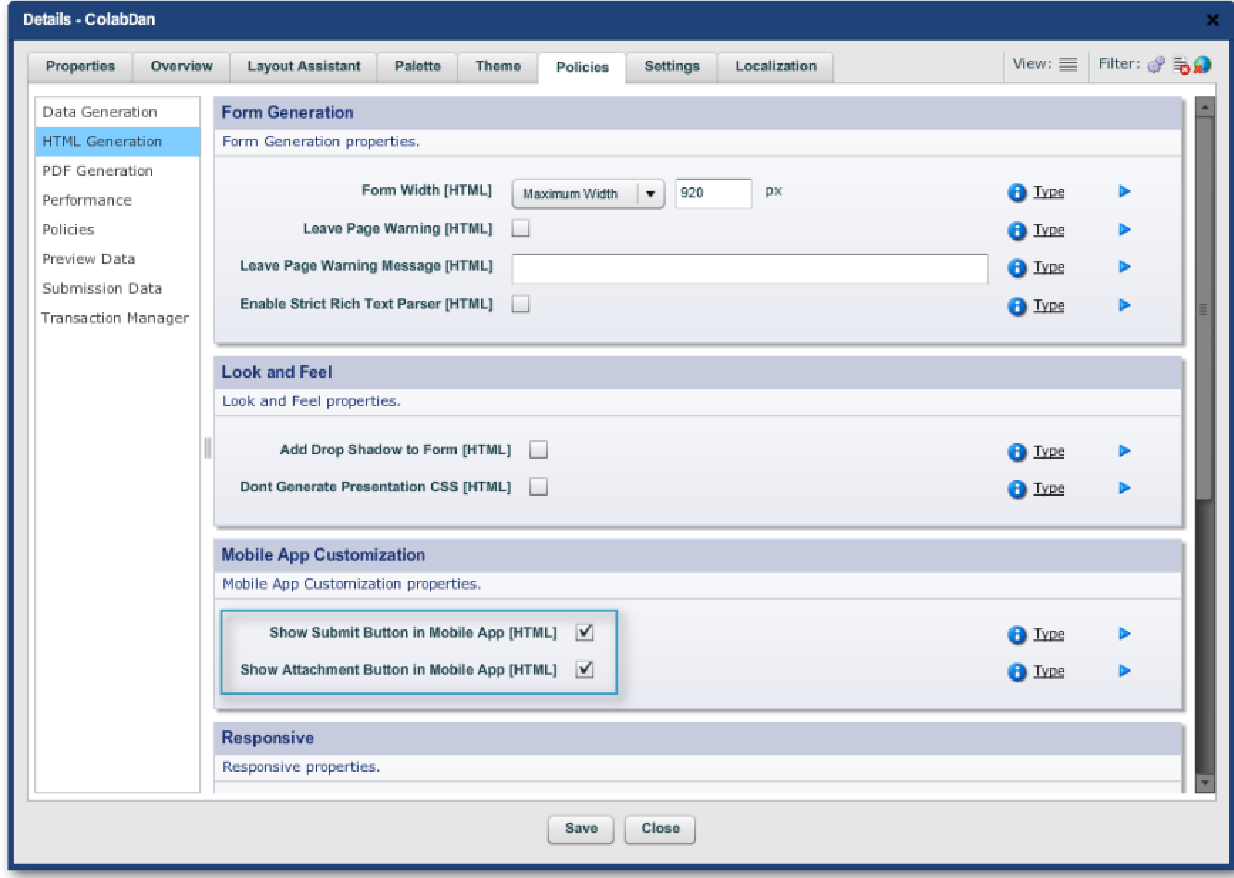


The form in the Mobile App, and the effect of the "Take Picture" buitton

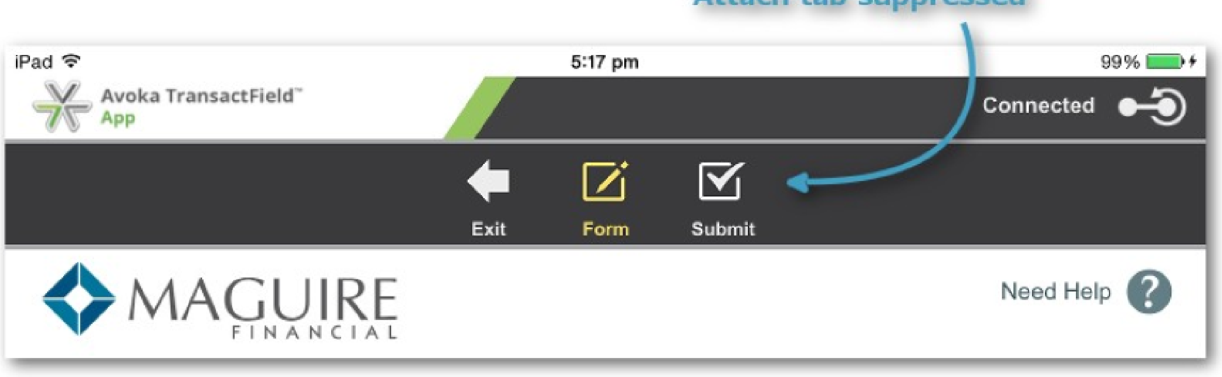
For simple forms like this example, and the user has taken only one photo, the user can go straight to the "Submit" tab and submit.
If the users have opted to "Use Photo" more than once, they have to go to the "Attach" tab and choose which image gets attached.

Suppressing the Attach and Submit Tabs

There are use cases where you will want to have attachments controlled on the form in the TransactField App, rather than in the app's tabs. You can suppress either tab (or both) in Composer's Structure Tree: "<Form> -> Edit Properties -> Policies tab -> HTML Generation -> Mobile App Customization panel".

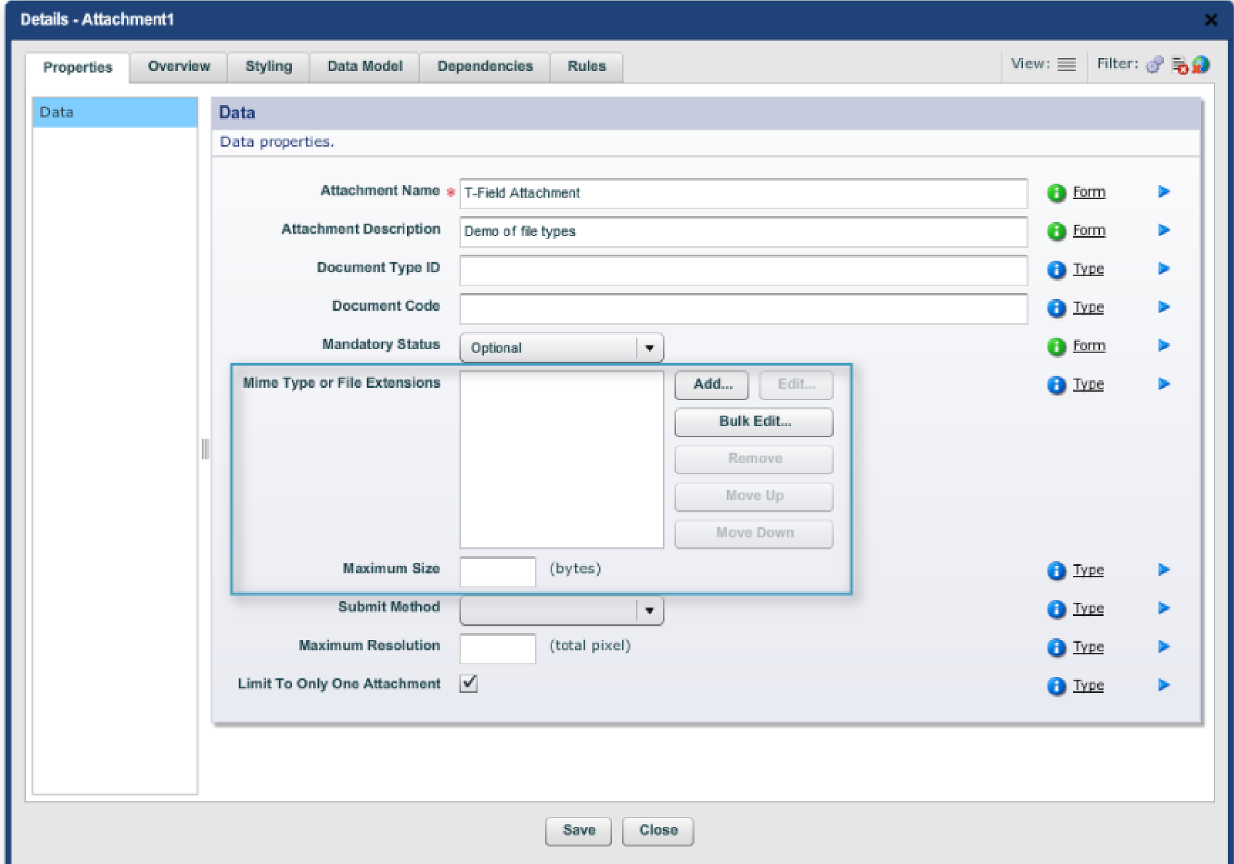


Here is an example of the Attach tab being hidden in the TransactField Mobile app:



Restricting Attachment File Types in the App

You now can control the file formats that the TransactField App can accept for an attachment or attachment. Place a TM Attchament Rule widget on the form and fill out the dialog:



Saving an attachment of the wrong type in the App results in a popup:

Multi-channel (Composer v4.3)

Overview

The Avoka Transact gives you the tools to publish all your forms on desktop and on mobile devices, and yet be useable on all these different form factors. This is more than a convenience — it has become a cost of doing business.
It follows that users will want to move freely between their devices, without needing to complete a complex form on one device in the one session. Users now expect that not only can they resume a form later, but that they can do so on a different device or platform.
The Avoka TransactWeb Multi-Channel Tracking Technology enables end-users to start, pause, continue and complete their applications across channels – from their mobile device to their desktop, the call center, and the branches.
Here is a video of a presentation Avoka made at a finance industry forum. It illustrates the device agnosticism users will appreciate.
{+}http://youtu.be/nNY8MdPw1Qc+

Authenticated Users

With the Maguire Template

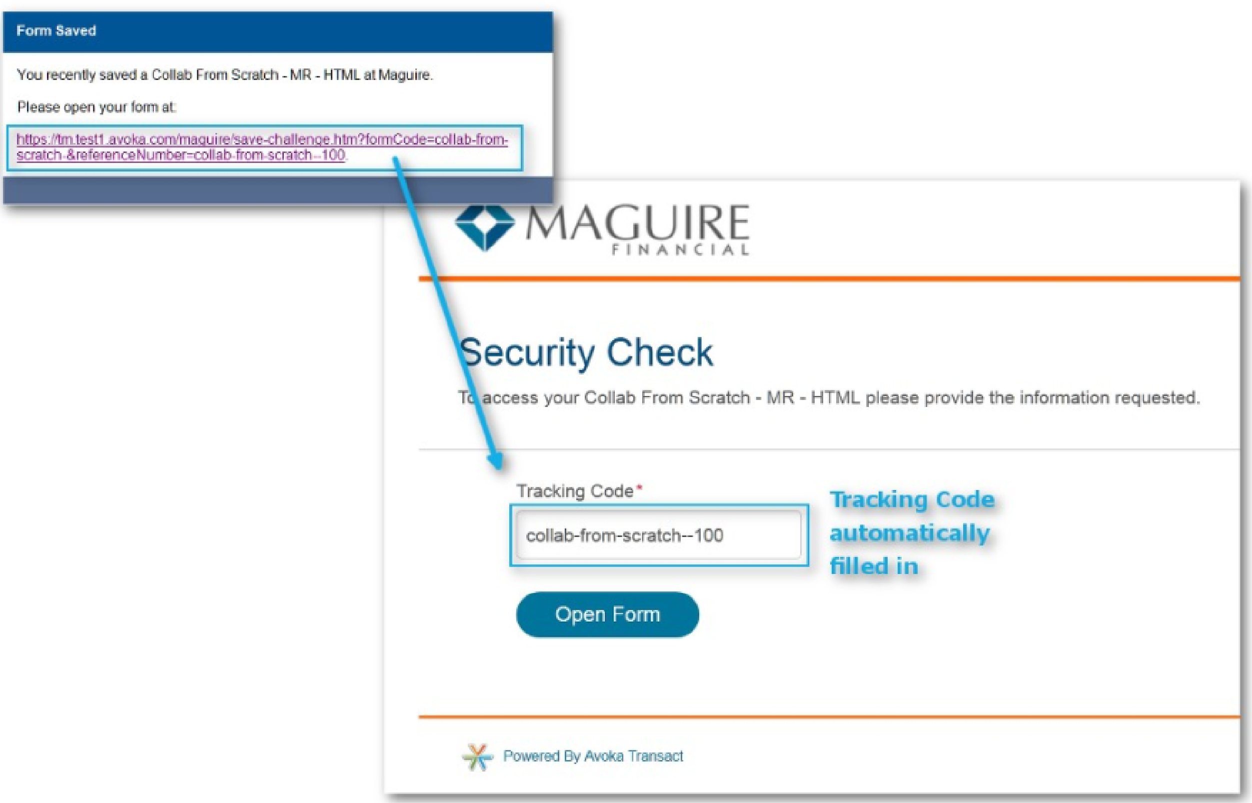
The Maguire template already has this technology embedded as a "Save and Close" button. On the desktop it is at the form header. On narrow mobile devices, it becomes an option in the slider menu .
You can configure TM to prompt users to opt if they want to receive an email (to an address they nominate) when they "Save and Close". They are immediately sent an email giving a link to the portal and their unfinished form. Users click on the link, log into the portal (if required) and continues to fill in the form, which is prepopulated with their data.

With Other Templates

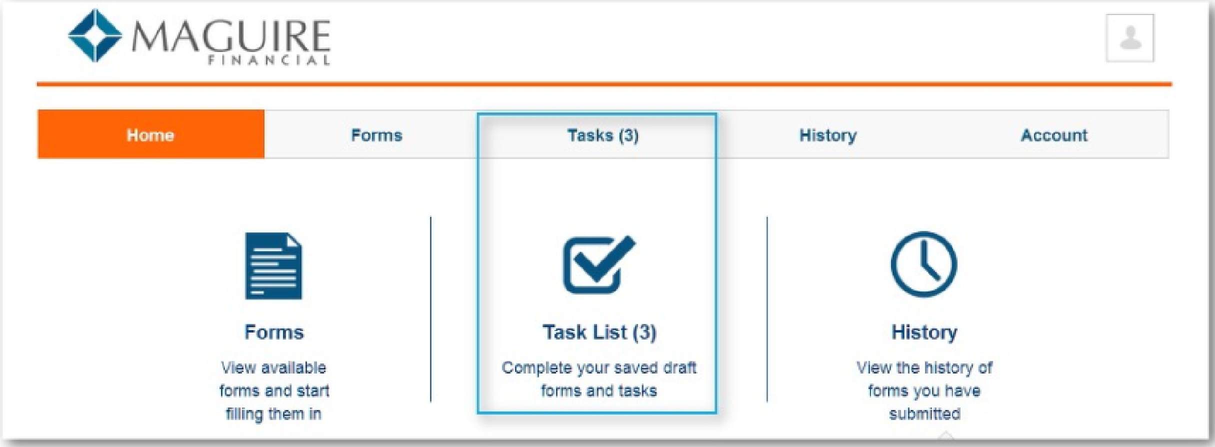
To get enable multi-channel, drop the TM Submission Block onto the form. You can reword the text fields in the block to suit. You can even remove the "Save to my Computer" and text, as saving the form as a PDF is not useful in the context of multi-channel delivery.

The Task List

So, users can resume their forms either by
Clicking on the link in the email. This takes users directly to the form, after first being challenged to supply ID and Password, and not to the portal's authenticated landing page.



Going to the portal's authenticated landing page and accessing the prepopulated form through the Task List.



TransactField App

These Save buttons in both the Maguire template and in the TM Submission Block, have special scripts so that they will work in the TransactField App. Putting the Save and Close buttons on the form will alter the App's Submit tab to accommodate the multi-channel save; the action will no longer be activated by the button on the form.
Of course, the button will appear on the form when the form is rendered on the device's browser (Safari in iOS and Chrome in both Android or iOS). On narrow devices, such as smartphones in portrait orientation, the Save and Close button will be in the Maguire slider menu.

Anonymous Users

Actually, Avoka Transact supports anonymous users, both for multi-channel and for collaboration.

The reasons you would want to do this are:
To reduce user friction
Some users, while happy to provide some personal details in order to request some service or good, are reluctant to create yet another transient user identity that will probably get used only the once
It is a necessary condition for Form Sharing , where several people have to fill in the one form, such as a pair of partners or another party is required to provide some documentation or statement before the form in question can be submitted.
Support for anonymous users is a configuration performed in TM. Please see Form Sharing .

Form Sharing (Composer v4.3)

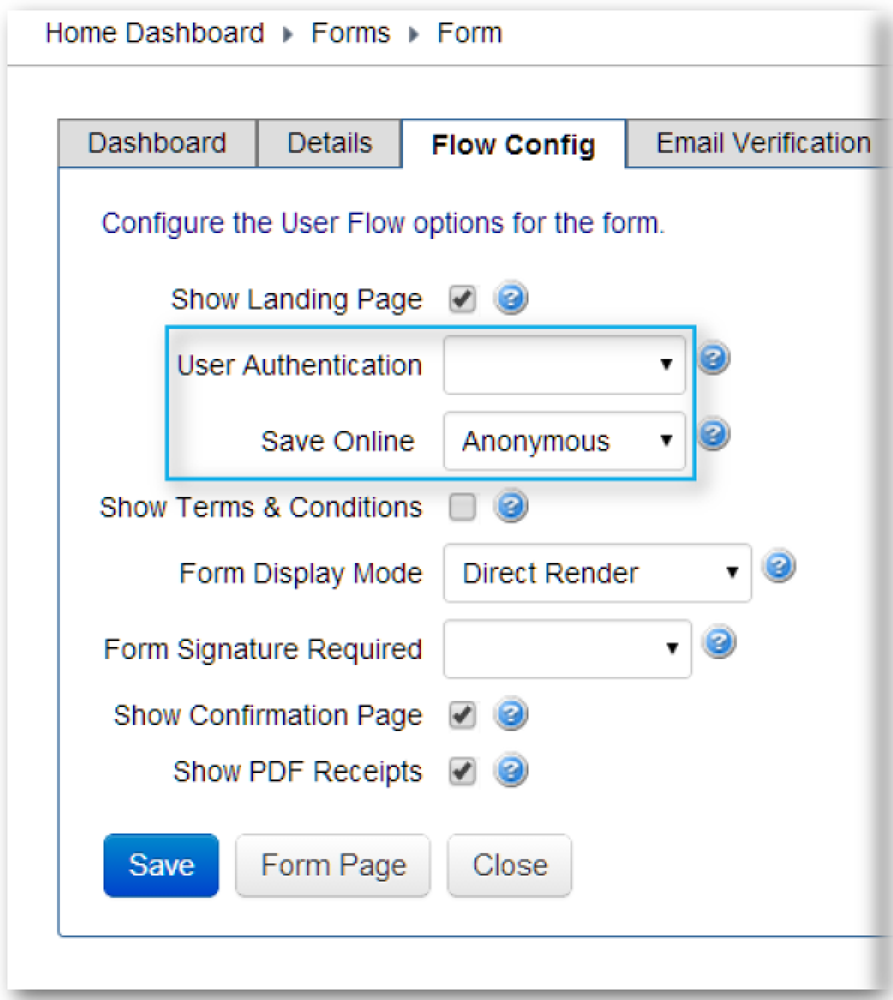
Overview

Just as a form can be shared between the devices of a single user, see the Multi-channel topic above, so too can a form be shared between a number of different parties before it gets submitted.
The kinds of business cases this function supports are:
Sharing the form between applicants and their lawyers Between applicants and their partners
Between various parties filling in the same form where they are not in physical contact.
The sharing involves giving the name, email of the nominated other party, and a text area for notes to the other party; then the user, instead of submitting the form (as would be done of the form were complete), clicks on a dedicated button with some appropriate caption like "Activate Sharing". TM generates an email containing a link. The other party clicks on the link and is taken to the form (prefilled with the other user's data) which can then be edited, files attached and so forth.
The following notes on enabling form sharing assume that you are using the standard form sharing widgets that ship with Composer. We do not discuss configuring the form's behavior for, say, edibility or visibility of its fields. We also are using the default submit button, which — without any extra scripting — allows any of the parties to submit the form. After submission, the form sharing ends.

Minimum Prerequisites for Form Sharing

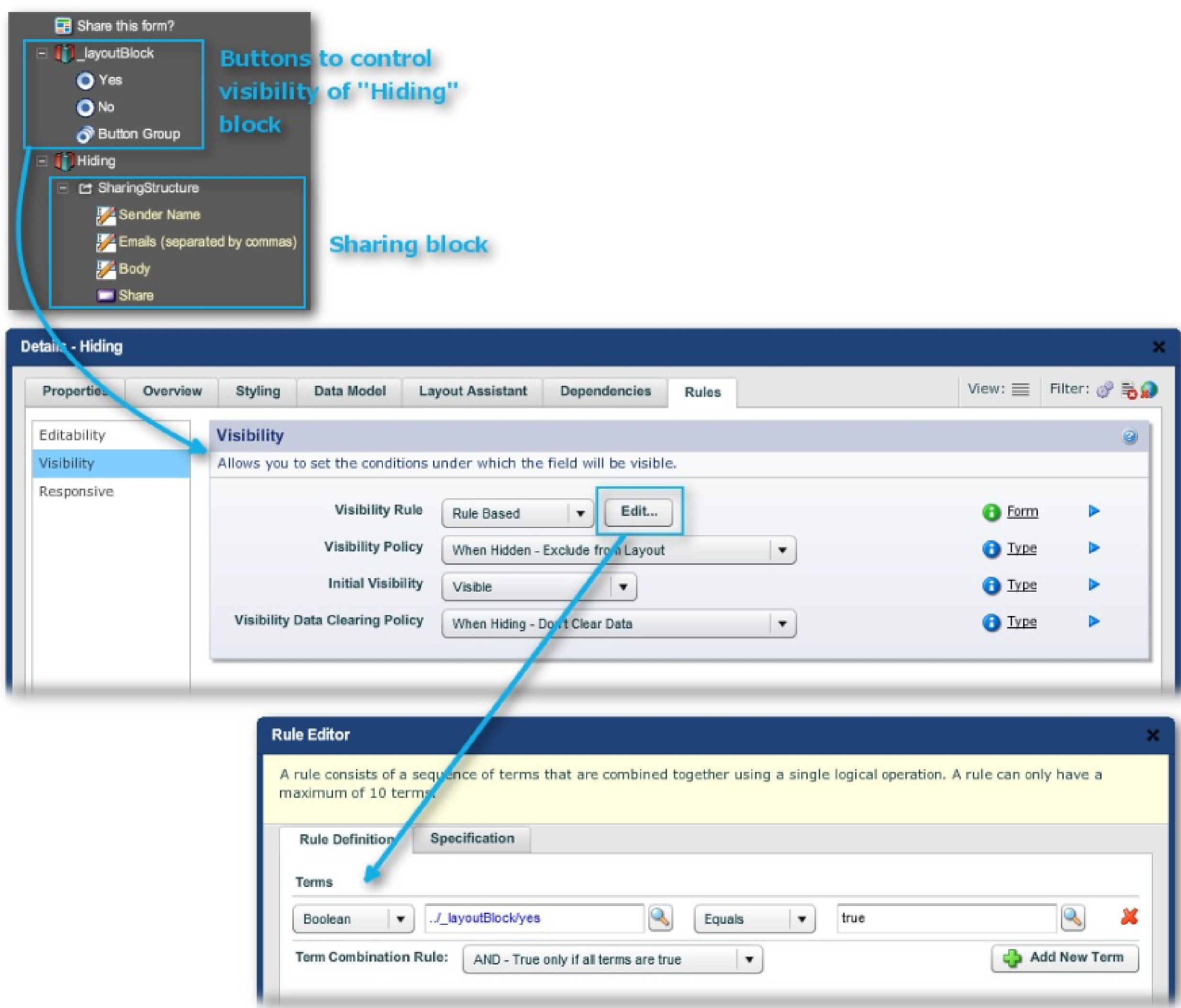
User Authentication Settings

The form's Authentication settings are made in Transaction Manager.

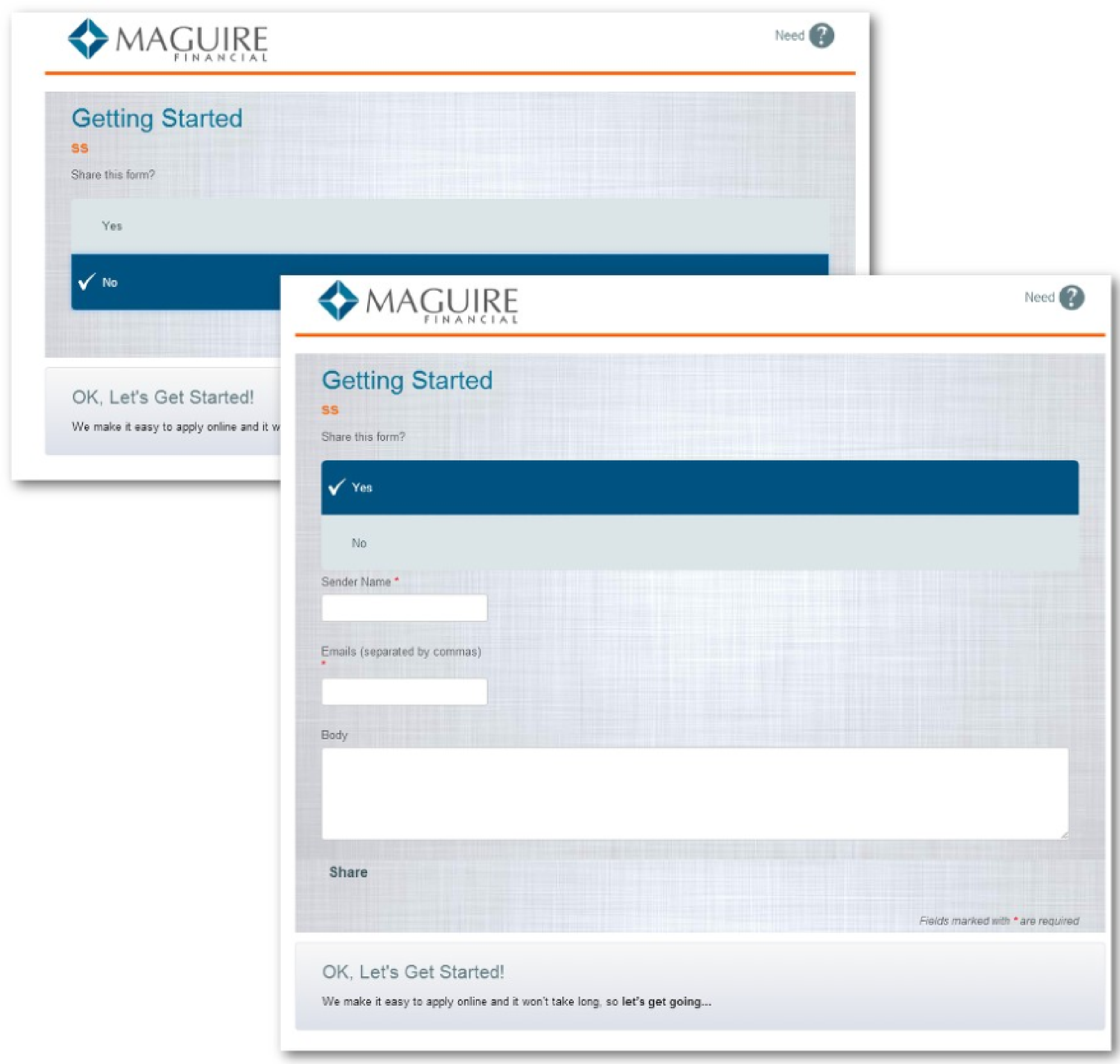


User Authentication has to be Anonymous (or unspecified at the very least). If not, the prefabricated TM Sharing block has scripts that make it invisible to authenticated users.
Form saving must also be anonymous so that the other parties can also save the form when they activate the share button.
There must be mandatory fields to capture the contact details of the original form user, otherwise there will be no connection to that person.

Simple Case using Prefabricated Blocks

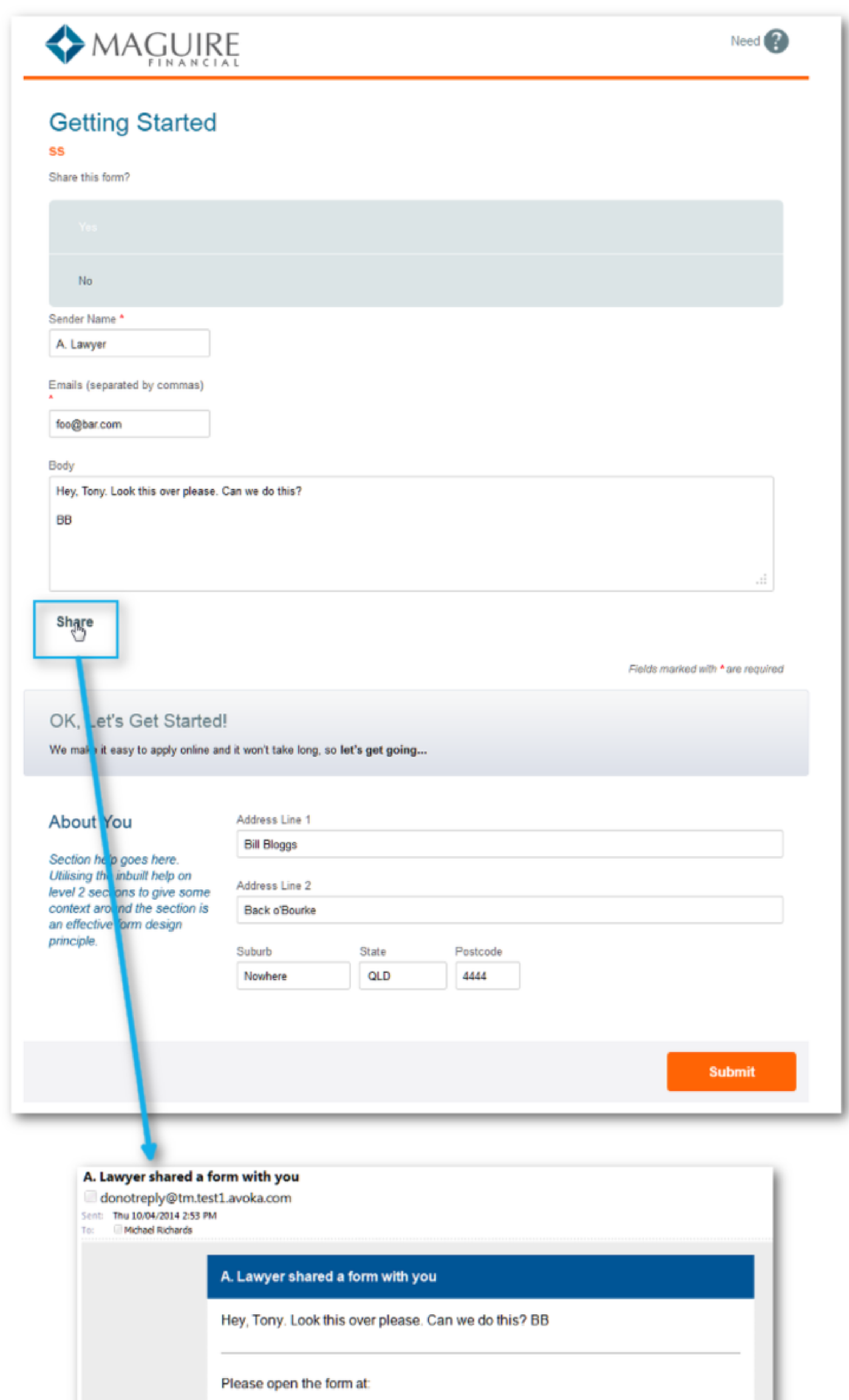


Which results in the following behavior.



The first party opens the form anonymously (using, in TM terminology, the "form friendly URL"). The user is given 3 choices:
Open New Form Open Saved Form Login and Open Form
For this example, we are going to select the first option.

© 2015 Avoka Technologies Pty Ltd 142


MAGUIRE FINANCIAL
Need
Getting Started
SS
Share this form?
Yes
No
Sender Name *
A. Lawyer
Emails (separated by commas) *
foo@bar.com
Body
Hey, Tony. Look this over please. Can we do this?
BB
Share
Fields marked with * are required
OK, Let's Get Started!
We make it easy to apply online and it won't take long, so let's get going...
About You
Section help goes here. Utilising the inbuilt help on level 2 sections to give some context around the section is an effective form design principle.
Address Line 1
Bill Bloggs
Address Line 2
Back o'Bourke
Suburb
Nowhere
State
QLD
Postcode
4444
Submit
A. Lawyer shared a form with you
donotreply@tm.test1.avoka.com
Sent: Thu 10/04/2014 2:53 PM
To: Michael Richards
A. Lawyer shared a form with you
Hey, Tony. Look this over please. Can we do this? BB
Please open the form at:

The second party can then open the form and share it back or submit it.
In TM, you can view the history of the form (when opened, when shared and when submitted) and the contents of the form (in XML).
Note: when you are testing form sharing (using a form served by Transaction Manager, not using Composer's Preview), all of the users cannot be logged into any of the portals. To test form sharing, you may have to initiate a new browser session, one in which you have not logged onto a portal. You can get round this be starting another browser altogether; you can already be logged into the portal in Chrome and test form sharing in a fresh session of Firefox.

Why Hide the Sharing Block?

This prefabricated block is mandatory. The form will not submit if the block is visible, unless the user enters some random data into the fields displaying in-line validation messages.

Sharing in Collaborations

This is possible to do. However, the user authentication of the first step needs to be anonymous. Also the link back to this user is through email alone and the service definition object will need to be modified. See Collaboration on how this feature works in general. Currently the documentation does not include what to do to be able to share the form before the first submission.

Composer Form Parts (Composer v4.3)

Form Parts projects introduce a special type of form known as a part-form. Within a part-form, one or more parts can be created. Inside the part, a form developer can create any content, including entire section level 1's. The parts can then be exported from within the part-form to the organization.

Page Contents

Search

- Overview
- Setting Up a Form Parts Project
- Creating Master Forms and Part-Forms
- Creating Parts within the Part Forms
  - Combining the Parts to Create a Master Form
  - Opaque and Transparent Parts
- Parts and Versioning
- Cross-part rules

Overview

In projects where there are many forms and/or larger and more complex forms, two problems can arise:

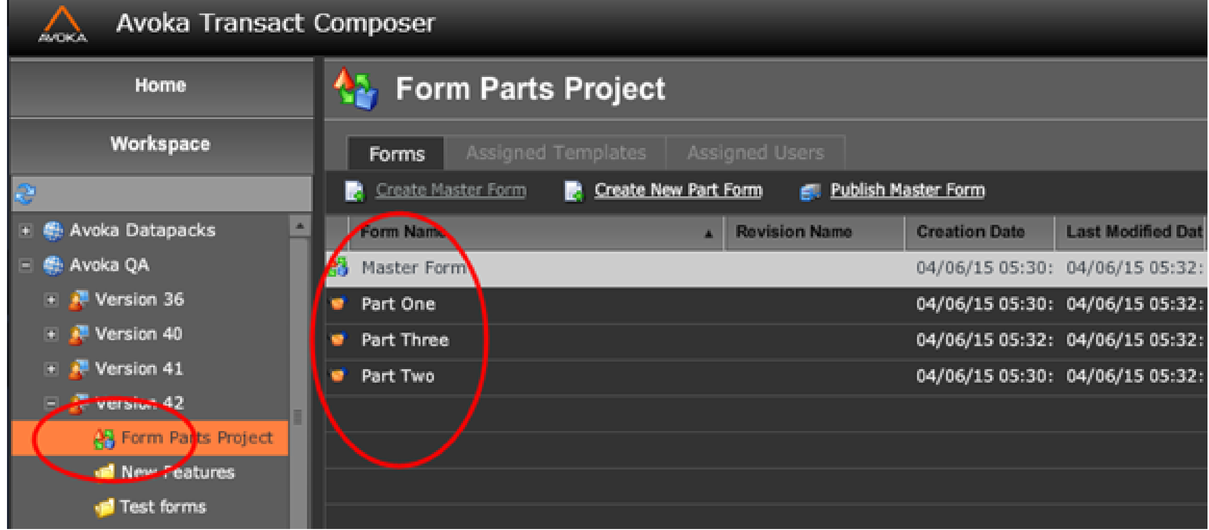
- Composer does not allow two users to work on the same form at the same time.
- Often forms contain large re-usable sections, such as a "Personal Details" section. Blocks can be used to create these sections, however blocks are generally designed for smaller entities, such as an address block. Composer isn't optimized for creating larger blocks, especially those containing sections. The block editor is an advanced tool which is intended for power users, and isn't suitable for editing these types of larger blocks. Instead, it would be preferable to use the regular Composer editing environment for creating these larger blocks.

Form Parts solves both of these problems.

Form Parts projects introduce a special type of form known as a part-form. Within a part-form, one or more parts can be created. Inside the part, a form developer can create any content, including entire section level 1's. The parts can then be exported from within the part-form to the organization.

Another special type of form unique to Form Parts Projects is known as a master form. It is used to include any of the exported parts that have been created in the part-forms. A master form can override properties from the parts if necessary. Business rules can be created either within each part, or in the master form spanning multiple parts.

These new form types are shown below:

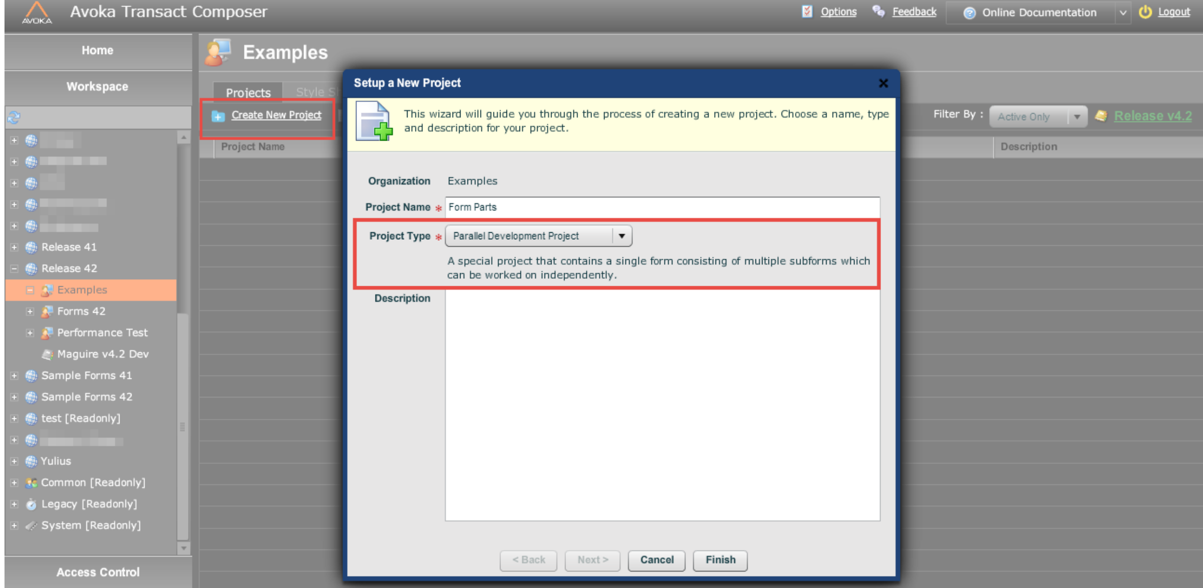


This allows multiple developers to work on different parts of a single form in a controlled and convenient way. It also allows re-usable parts to be created that can be used in multiple master forms.

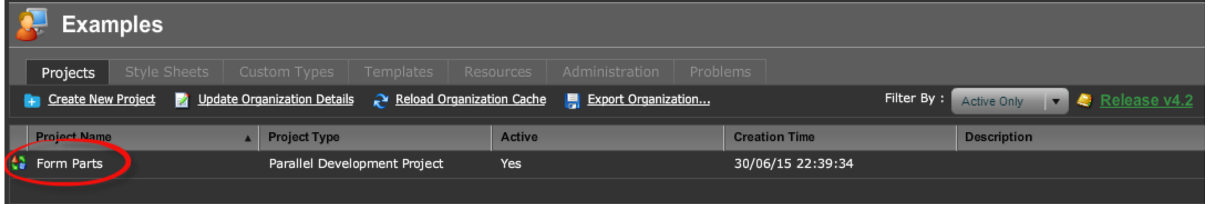
Setting Up a Form Parts Project

Form Parts projects (also known as Parallel Development projects) behave differently to normal Composer projects. For this reason they are distinguished from normal projects with different icons.

Creation of a Form Parts Project is done at the Organization Level by selecting "Create New Project". In this dialog the Project Type dropdown "Parallel Development Project" must be selected.



This creates a Form Parts project

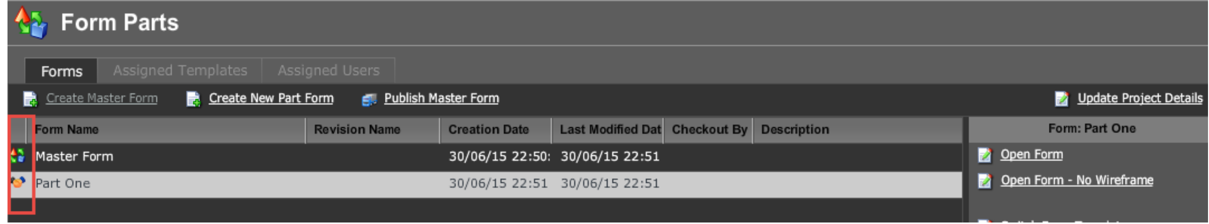


Creating Master Forms and Part-Forms

At the project level you can create the part-forms and master forms by selecting the options.This process is much the same as creating an ordinary Composer form.

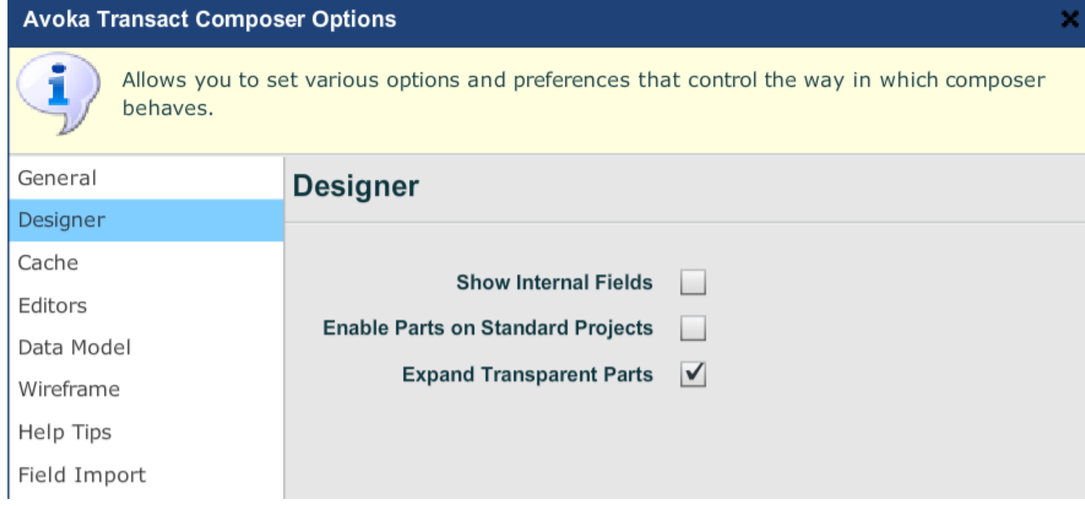
Only one master form can exist in a form part project. Multiple part forms are permitted.

The different form types are distinguished by their icons. The Create Master Form option is now disabled in the screenshot below.



There is nothing special about a Parallel Development project, or the master form or the part forms. These are really just regular project/form, which are tagged in a special way to give them different icons in the Composer user interface. They also enable some special toolbar icons in the project menu, such as Create Master Form and Create New Part Form. You can convert freely between a Parallel Development project and the Standard type of project. You can promote a regular form to a Master form using the right hand icon menu at the project level (but you can only have one master form per Parallel Development project).

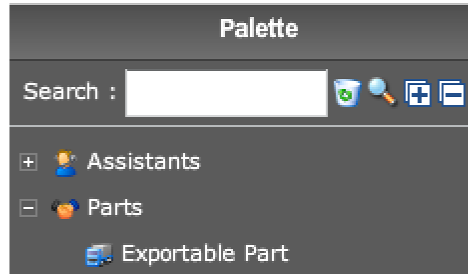
If you wish to use the special parts menu options in a regular project, then you may do so by checking "Enable Parts on Standard Projects" in the Composer Options dialog.



Generally the master form and the parts will all be in a single project. However, a master form in project B can also use parts that are defined in project A. This approach might be useful for the case where multiple forms utilize the same set of common sections, and these sections are defined in a separate common project.

Creating Parts within the Part Forms

The part forms act as containers for the parts. In order to create a part, drag a Exportable Part from the Parts folder in the palette into the top level of your form.

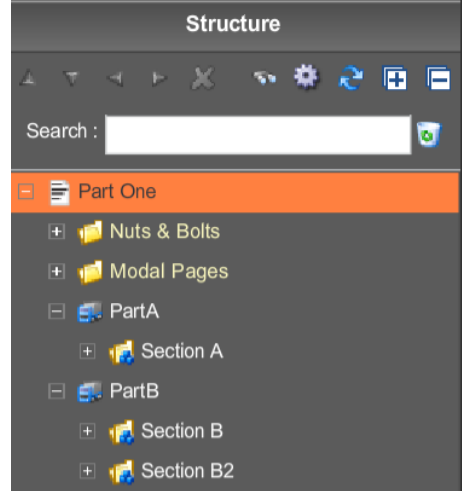


You can have as many parts as you want within each part form.

Parts can be either transparent or opaque. This will be covered later.

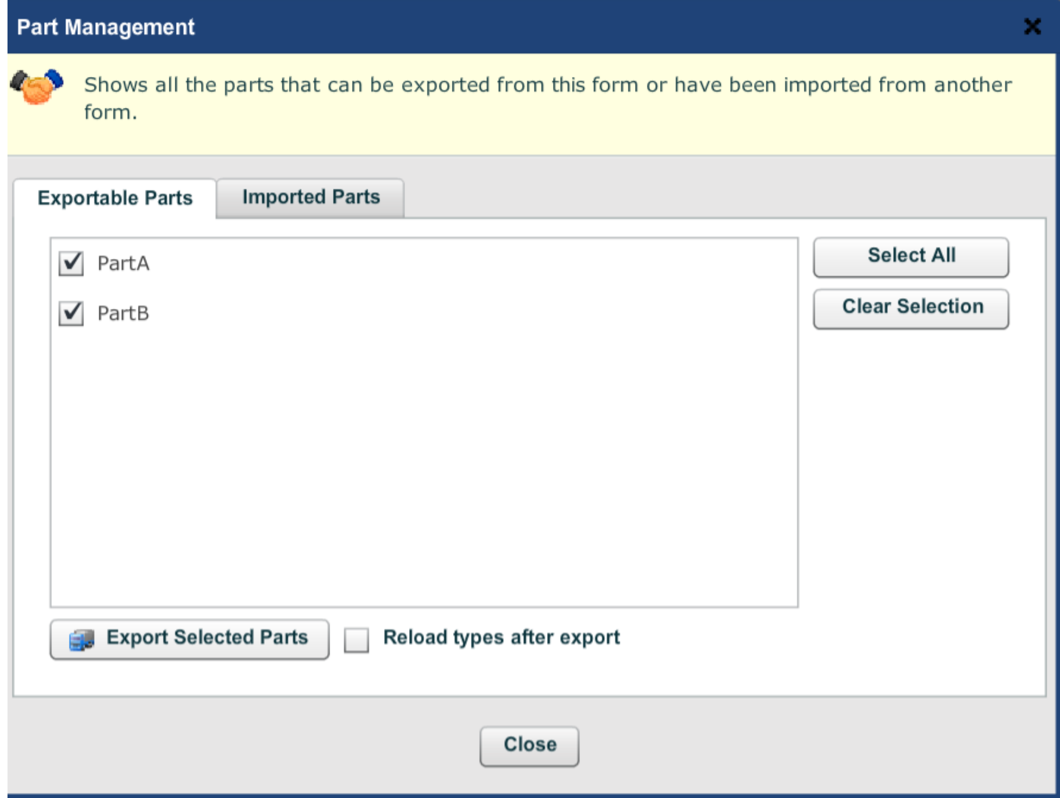
You can then add any content you wish into your Part. In most cases, you will drag a Section Level 1.

The example below shows a Part One form that contains two parts, Part A and Part B. Usually a part form will only contain one part, and a each part will contain one section, but this example illustrates that a part form can contain multiple parts, and a part can contain multiple sections if desired. The sections can contain arbitrary content, including fields, blocks and business rules. However, if your form contains multiple parts, you must not create any dependencies or business rules between the parts. Each of the parts can be developed, and tested in isolation, as it it was a normal form.



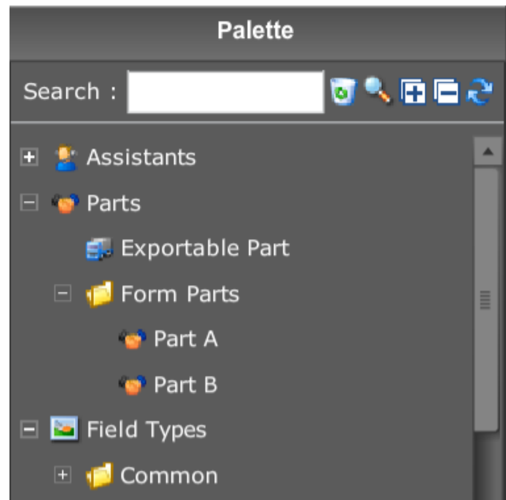
In general, it is recommended that each part form contain only one part, and that each part contains only one section. It's also usually a good idea to name parts and part forms (and even the sections they contain) using a consistent naming scheme, so that the parts and sections can easily be located.

In order to use these parts within a master form, they must first be exported from the part form. In order to export one or more parts, click on the Parts... icon which is visible in the toolbar of all part forms. This will bring up the Part Management dialog shown below.



Your parts are not available to be used in a master form until they have been exported. Click Export Selected Parts in order to export the selected parts. It is not usually necessary to select "Reload types after export" - use this checkbox if you want to verify that the parts have been exported by viewing them in the component palette.

After exporting, the selected parts will appear on the component palette on the right under the Parts node, with a sub-node which is the name of the project, as shown below:



If you use parts extensively, this can result in many projects and parts under the Parts node, which can become confusing. This can also result in slower time to load the types for the organization. If you use a large numberof parts, you should either use a project and part form naming convention that allows you to easily identify the various parts, or you should create multiple organizations to group the various parts and the forms that use them.

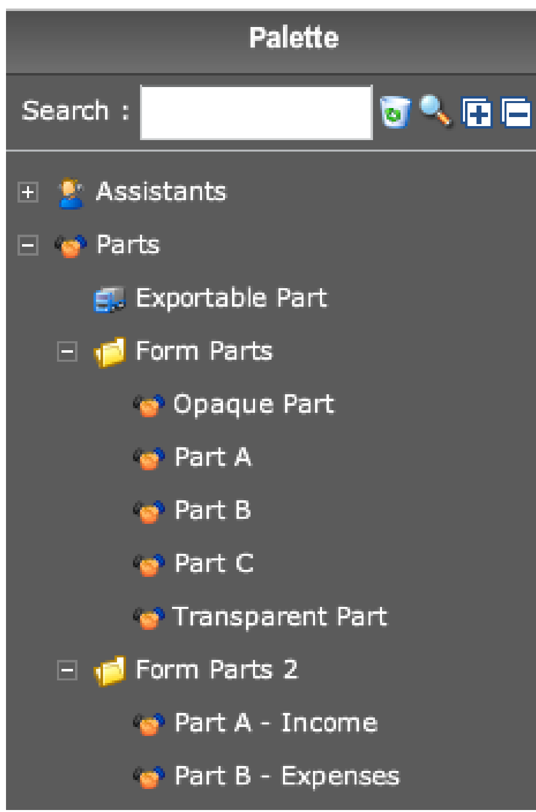
An exported part is actually just a special type of block, similar to a block created using "Save as Custom Block" or the block editor. It is really just the way that the block is created, edited and exported that is different.

Saving the form does not automatically export the parts. And exporting the parts does not automatically save the form. This is important so that you can control when and how you update your exported parts. Each of these operations must be performed separately. This is useful because it allows you to finely control when you make your part available to others. For example, you wouldn't want to export your part if it has a bug, or if you're not finished developing it.

Be careful of renaming a part - the part will be exported under the new name, but the original part will be retained - this may cause confusion. You can manage the parts as in the Custom Types tab at the organization level.

Combining the Parts to Create a Master Form

In the master form, simply drag and drop the various parts that you require from the Parts folder in the palette into your master form. Because parts are generally section level 1's, you should drag the parts into the top level of the form. For parts that contain other types of content, drag them into the appropriate locations.



Parts are grouped by the name of the Project in which they are defined. You can drag as many parts as you want into your master form.

Once you have imported parts into the master form, you may view a summary of what parts have been included by clicking on the Parts,, button and selecting the Imported Parts tab. This tells you not only what parts are being used, but also which part form they are defined in, and when/who exported them.

Part	Exported From	Exported By	Exported At
OpaquePart	TransparentOpaque	htreisman@avoka.com	2015-07-30 21:18:15
TransparentPart	TransparentOpaque	htreisman@avoka.com	2015-07-30 21:18:35

Important: it will be important later on when we discuss cross-part business rules that all parts have the same name everywhere. The easiest way to ensure this is to accept the default name for each part when you add it to the master form.

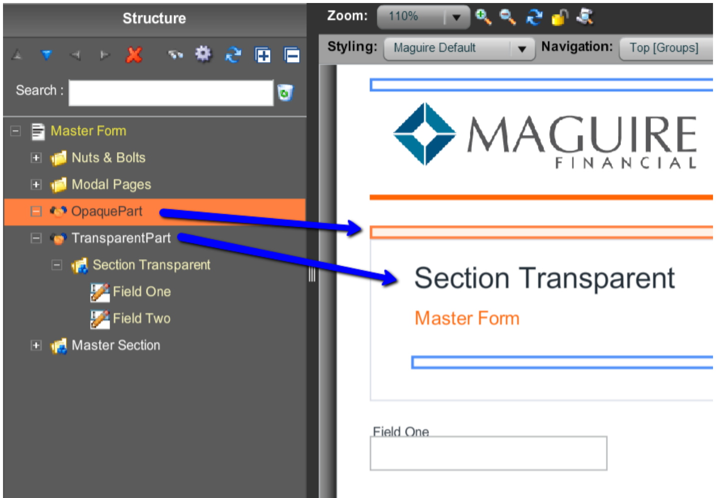
Opaque and Transparent Parts

A transparent part is the default type of part. When imported into a master form, it looks just like a normal section of the form.

An opaque part is treated as a "black box", and is not actually loaded into the Composer designer. An opaque part's contents will not appear within the structure tree or in the wireframe. However, the contents of an opaque part will appear in the form itself, when previewed or published.

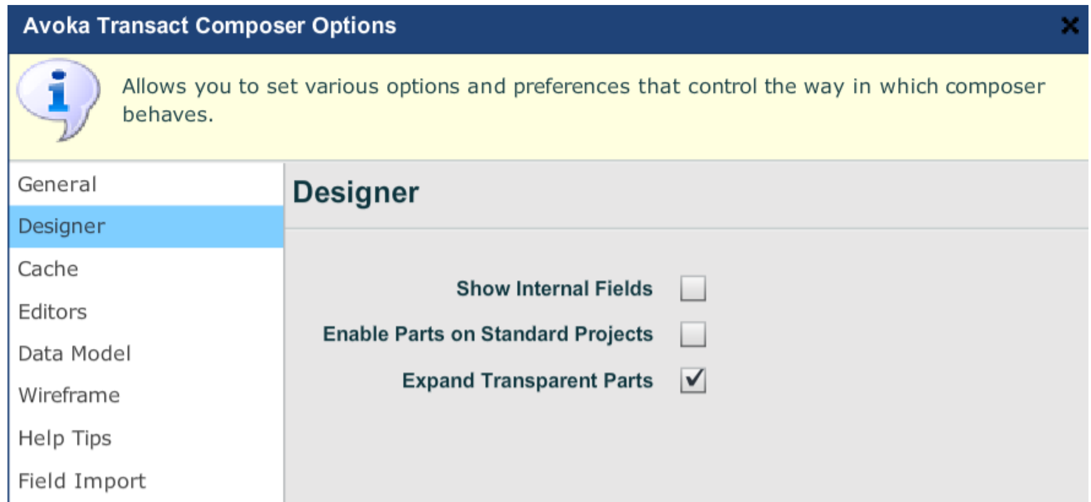
There is no real difference internally between a transparent and opaque part - it is only the Composer editor that treats them differently from a visibility perspective. You can modify whether a part is transparent or opaque in its property dialog.

The difference between an opaque and transparent part when imported into a master form is shown in the picture below.



You can mark your parts as opaque in order to indicate your intention that your part should be used "as-is", and not modified. An opaque part will also mean that the form using the opaque part will load quicker.

By default, transparent parts will not have their contents displayed in the structure tree. This serves as a reminder that the contents of the part are not intended to be modified. In order to see the contents of transparent parts in the structure tree, you need to select "Expand Transparent Parts" in the Composer Options dialog.

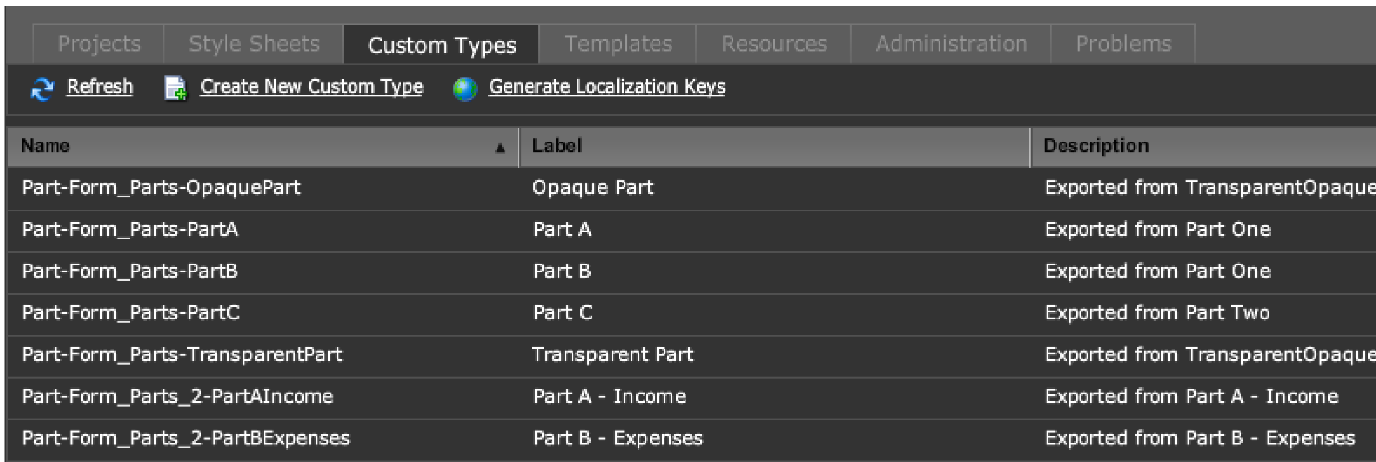


Even if the "Expand Transparent Parts" option is unchecked, the fields in a Transparent part are always available in the rules editor, and so may be used within rules.

Parts and Versioning

Part Forms (the forms that contain the parts) are like any other type of form, and benefit from the usual Composer form versioning capabilities. However, the parts themselves, which are exported from these part forms, and which exist within the component palette, only have a single version - this being the latest version that was exported.

Under the covers, parts are exported as regular Composer blocks, and can be viewed within the Custom Types tab at the Organization level.



Name	Label	Description
Part-Form_Parts-OpaquePart	Opaque Part	Exported from TransparentOpaque
Part-Form_Parts-PartA	Part A	Exported from Part One
Part-Form_Parts-PartB	Part B	Exported from Part One
Part-Form_Parts-PartC	Part C	Exported from Part Two
Part-Form_Parts-TransparentPart	Transparent Part	Exported from TransparentOpaque
Part-Form_Parts_2-PartAIncome	Part A - Income	Exported from Part A - Income
Part-Form_Parts_2-PartBExpenses	Part B - Expenses	Exported from Part B - Expenses

If you wish to revert to a previous version of a part, you must open the version of the part form that contains it, and re-export.

Cross-part rules

When business rules exist entirely within a part, then they are very easy to set up - simply define the rules within each part as normal, and the rules will be imported along with the part into the master form. However, when it is necessary to define rules at the master form level that span multiple parts, then things can get a little more complicated.

There are several different approaches to dealing with cross-part rules. The approach you choose will depend on the sophistication of the form and the business rules, as well as the sophistication of your development team. Three different approaches are discussed below. There are probably other approaches that can be used too, but these are the recommended approaches.

All of these approaches require the development teams to use discipline in the way that they use the parts capabilities. It is possible for a single project to mix and match the various techniques listed below - but this can become confusing, and it recommended that the form developers on a project consciously select a particular approach and stick to it.

Simple - All Cross-part Rules in the Master Form

All the cross-part rules are defined in the master form, as if it were just a single large form. Rules can be set up that include fields in any of the parts, and /or any of the fields that are explicitly placed in the master form. This has the advantage that it is very simple to use and understand.

However, it suffers from some disadvantages:

- All the responsibilities and dependencies are on the master form developer. If there are a large number of cross-part rules, this limits the ability for multiple developers to be able to work independently on these rules.
- When any of the parts are changed, the master form developer needs to fix everything in the master form. In some cases, it might be preferable for each part developer to manage their own cross-part rules.

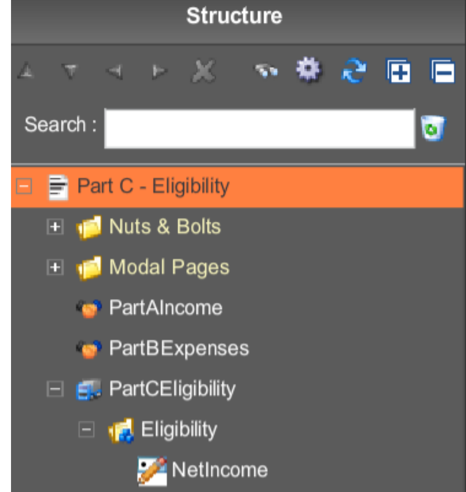
Normal - Each Part References Other Parts it Requires.

This is a slightly more sophisticated approach, and encourages each part developer to define and maintain the rules in their own parts independently.

Let's consider the following example:

- Part A - Income Section. This section has a numeric field named TotalIncome.
- Part B - Expenses Section. This section has a numeric field named TotalExpenses.
- Part C - Eligibility Section. This section contains a calculation for a field NetIncome = TotalIncome-TotalExpenses, which is in turn used to determine whether a customer is eligible for a loan.

The developers of Parts A and B develop their parts as usual. The developer of Part C, because there is a dependency on fields outside of his part, will add these parts into the part form. The Structure tree for Part C is shown below:



Note that:

- The above diagram is the structure for Part C, NOT for the master form. The master form will include parts A, B and C, but will not directly contain any business rules.
- Part C is defined and modified within this part form. But parts A and B are defined elsewhere, and simply imported so that they can be referenced.

The developer of Part C will include the calculation rule into Part C. When Part C is exported, it will contain a business rule that references Parts A and B.

When Parts A, B and C (and any other additional parts) are included in the master form, the business rule that exists in part C will reference fields in other parts. However, since these parts also exist within the master form (as imported parts), the references will be correctly resolved, and the business rule will work correctly. (This is why it was stated earlier that each part should always be given the same name wherever it it used.)

Tips:

- Within the Part C form, only add cross-part business rules to Part C (not Parts A or B). Because we are only exporting Part C, any business rules you add to parts A or B will never be exported.

- When editing a part form, use Transparent parts, but uncheck "Expand Transparent Parts" in the Composer Options screen. This will display the imported parts without any contents, as shown above. This will serve as a reminder to not add any business rules to these parts. The business rule editors will still allow you to create rules in Part C that include field from Parts A and B.
- You can preview and test your form as if it were a complete form.
- You can set the wireframe to only display Part C, using the Wireframe Options button. That way your form will load faster, and also you won't be tempted to accidentally add rules to parts that aren't defined in this form.
- If one of the other editors makes a change to part A or B, you may need to adjust rule business rules. However, you can usually detect this using the Problems pane.
- When working in a single form, Composer will automatically refactor your business rules if you move or rename a dependent field. However, this will not be done when the dependent field is modified in a different part. It is therefore advisable to either limit the number of changes made, or to communicate well between developers when these changes are made. The Problems pane will notify you of any errors.
- Generally when using this approach, the master form just contains a series of parts, and has no content or business rules of its own. It is possible to mix the approaches, but this is not recommended, as it can get very confusing.

This approach does have a number of disadvantages in certain situations:

- Part C has a dependency on Parts A and B. It will fail if used in a form that does not have Parts A and B, and cannot be used independently of them.
- This approach may not work very well when a particular form has business rules that reference a large number of different parts - you may be better off using the Simple approach in this case.

Despite these minor disadvantages, this is the approach that is generally recommended.

Advanced - Model/View/Controller

This is an advanced approach to cross-part rules, using a well-known pattern from software engineering, the Model/View/Controller pattern. In this approach, rather than defining business rules directly between the fields of one part and those of another, an intermediate data model is constructed, which serves as the communication mechanism between the parts. In summary:

- Model = a special "data model" part containing only data fields. (A data field is a special invisible field available in the palette.)
- View = the regular parts containing sections.
- Controller = the cross-part business rules (these can be defined in a number of places, as described below).

This approach is a little more complicated than the preceding ones, and requires some discipline and rigor in the development process. However, it has the advantages of:

- An explicit definition of what data is shared between the parts.
- A software "contract" about how this data is shared and used.
- There are no direct dependencies between any part and any other part - there is only a dependency between each part and the shared data model. This means that each of the part forms only need to include the model part, and nothing else.

When creating the model, there are some interesting choices about what data elements to create in the model, and where to place the rules (or controller). For example, if we had a simple calculation of "C = A + B", where A, B and C are each in a different part, then we could have any of the following:

- A and B in the model. C exists only as a real field, and contains the calculation.
- A, B and C in the model. The business rule exists on data element C within the model. The real field C has a calculation equaling the data field C.
- Other choices are possible.

One simple approach is to only include in the model those fields that are needed for business rules in other parts, and only to include business rules in the parts. However, your project team may choose a different approach.

Once the model has been created, it is imported into each of the part forms, and then two steps need to be performed:

1. Create any business rules that are dependent on data fields in the model.
2. "Push" the values of any fields that need to update the values in the model. (It is up to the developers to ensure that each field in the model is only updated by a single real field.)

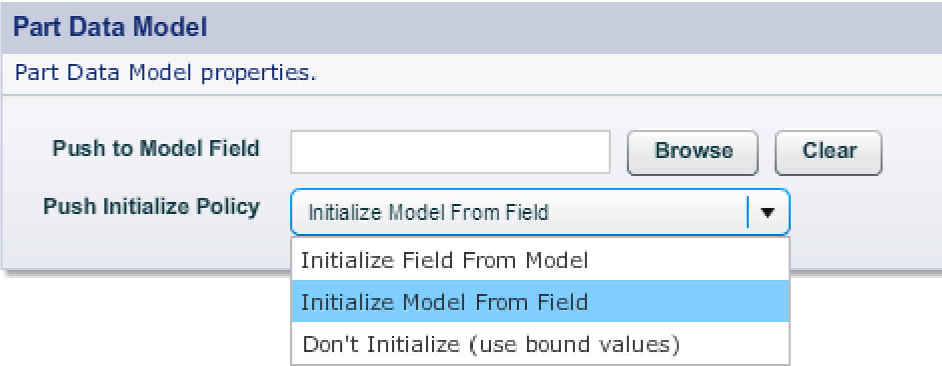
Note that while it is theoretically possible for the model to "pull" the values it needs from the various parts, this breaks one of the fundamental principles of model/view/controller, in that the model has no dependencies on any of the views. Views can reference the model, but the model should not reference the views.

In order to do this, you can use the "Push to Model Field" property of the Part Data Model section on the Properties/Data tab of the properties dialog.

Another interesting decision occurs because you now have two fields (a data field in the model, and a real field in the view), both of which represent the same physical data element. The question becomes: How do we bind these two fields to the XML, and initialize each of the values? Remember that when a Composer form loads, each field will first have its value initialized from the XML data it is bound to, and then any business rules will be fired. The choices are as follows:

1. Bind the real field to initialize it. Initialize the model field from the real field.
2. Bind the model field to initialize it. Initialize the real field from the model field.
3. Bind both fields. No additional initialization is necessary, because they are both bound, and will both be set from the binding values in the XML. This can be a little dangerous, because you are responsible for ensuring that the XML contains the same value in both locations, which in turn ensures that both fields are initialized to the same value. If the XML is inconsistent, you will get strange behavior, with rules not appearing to reflect the data shown. Alternately, you could selected either "Initialize Field from Model" or "Initialize Model from Field", and even though both are bound, as soon as the form loads, the second field will be set to the value of the first one.

You can control which of these approaches using the "Push Initialize Policy" property, as shown below.



The decision outlined above actually occurs any time that you have two different fields that represent the same piece of data, not just in this approach to cross-part rules. The same reasoning, and the same solutions, can be used in these cases also.

When using this approach, it is recommended that you make each of the view parts opaque, and the model part transparent.

Usability and Accessibility (Composer v4.3)

General Statement

The W3C defines Accessibility as:
"Web accessibility means that people with disabilities can use the Web. More specifically, Web accessibility means that people with disabilities can perceive, understand, navigate, and interact with the Web... "
http://www.w3.org/WAI/intro/accessibility.php/
Accessibility is challenging. Users with vision impairment, who depend on Screen Reader applications so that the contents of a web page is read aloud, have to be accommodated. Forms also must cater for partial visual impairment (such as color blindness or poor accuity), cognitive impairment, motor and auditory disabilities.
Composer does a number of things to assist compliance:
Composer generates many of the HTML tags that are required for correct accessibility. For example, all images are generated with an ALT tag so that blind users can have the image read out to them.
Composer test forms are verified by a number of accessibility test tools to ensure that the HTML Composer generates passes these tools' criteria.
Composer's use of stylesheets means that if your first form is accessible, then all subsequent forms are much more likely to be accessible.
However, while Composer tries to do as much as possible, it cannot absolutely verify that your forms are accessible. For example:
While Composer will generate an ALT tag for your images, it cannot verify that the tag's text is actually a good description of the image.
Composer will apply consistent colors and styles, but it cannot verify the the contrast of your foreground text against your background color meets the criteria W3C criteria.
Even so, the accessibility of your forms is ultimately your responsibility.
The full WCAG guidelines can be found here: http://www.w3.org/TR/WCAG/

Our latest portals are largely WCAG compliant, but be aware that individual forms may not be. Some customizations can defeat WCAG compliance.

Note: There is a misconception that the use of JavaScript in a web page makes the page inaccessible. This is completely incorrect. This comes from a perception many years ago, that certain operations in JavaScript confuse screen readers. However, modern screen readers handle JavaScript well, and in fact some JavaScript can make your forms easier for all users, including the disabled. The appropriate use of JavaScript is more of a positive for the disabled users — for example, hiding a section that isn't relevant means that the unsighted user will not have to tab through many irrelevant fields.
Avoka will aim to achieve 100% WCAG v2 AA compliance in Total Validator, and 100% compliance in SiteMorse.
While we do pass automated tests, we also understand that there is a difference between passing tests and being truly accessible. Any additional or specific Accessibility requirements are not necessarily in scope but will be dealt with on a case by case basis.

FAQs

Question:

Are we confident to state that they will be WCAG v2 AA compliant for forms created in Composer?

Short Answer:

No. We will never be able to certify that. But we can make it as easy as possible to achieve AA.

Long answer:

Passing automated tests doesn't guarantee compliance (although it's a good start). Automated tests tell you when you've done something wrong, but they don't tell you when you've failed to do something right. We are in the process of doing further human expert tests on sample forms to improve our compliance. Ask two different "experts" to certify a form, you can very often get two different answers – there is a level of subjectivity in interpreting WCAG. It also depends on the browser, and the screen reader.
Composer is just a tool. It helps you to build forms that work well, but it also allows you to do what you want. You could easily do things in your form, either intentionally or unintentionally, because of the way you build it, or the JavaScript you hand-craft, that breaks accessibility. We can NEVER certify that your forms will be accessible just because our tool starts from a good place. For example, you may put light grey text on a dark grey background and fail contrast requirements. Or name an image inappropriately. It's a bit like asking Microsoft to certify that all documents you produce will be spelled correctly just because Word has a spell checker.
Ultimately, meeting the needs of people with disabilities is about trying to do the right thing. Organizations will be attacked if they appear not to care. Being concerned, but making some mistakes, and learning from them is defensible. Picking the right tool is a good start. Doing some accessibility testing as part of your development process would further support your good intentions.
Ultimately, the only way you can be 100% sure that the forms you build are accessible is to get an accessibility expert to test them.

Tab Order

Tab Order is the order in which your cursor will navigate from one field to another when you press the Tab key. This is particularly important for users who touch-type, who want to avoid taking their hands off the keyboard and onto the mouse in order to move to the next field; and also for visually or physically impaired users, who favor the keyboard over the mouse.
In Composer, you don't have to set the tab order - Composer sets the tab order for you. There is nothing for you to do, and nothing for you to maintain. If you insert a new field, it will simply be added in the correct tab order relative to the fields around it.
If you need to explicitly control the order of tabbing, you can achieve some control by wrapping your fields in blocks. Tabbing will always occur from field to field within a block before moving to fields in subsequent blocks.

Privacy (Composer v4.3)

General Overview

Privacy of information has, for us, 3 major concerns:

1. That only appropriate information is temporarily retained and then submitted
2. That the submission is sent securely to the server
3. That submitted information is retained securely

Only the first of these is the concern of form designers using Composer.

Privacy Concerns with Hidden Fields

Many organizations have strict data privacy policies. A common policy is that the organization should never collect data that the end-user is not aware of.
This can be an issue in the following example:
The end user indicates that they would like to include insurance for a spouse by clicking the "Insure my spouse" checkbox.
The form displays an optional section, where the spouse's details are collected.
The end-user later decides not to insure their spouse, and unchecks the "Insure my spouse" checkbox. The Spouse section is hidden.
The end user is now submitting private information about their spouse, that is not required, but has been captured in fields that are no longer visible. This can be viewed as a privacy breach.
Here are two ways of solving this problem:

Clear Hidden Data on Submit

The standard Composer Submit button has a property named "Clear Hidden Data on Submit". This is checked by default. If this property is selected, the form will automatically clear the values of any fields that are hidden when the form is submitted.
Note: The Submit button also has a property named "Clear Non-editable Data on Submit". This is rarely turned on, but is available if required.

Overriding

Sometimes it is necessary to preserve the data in an invisible field even though the user can't see it. For example, there may be a calculated field that is used purely for internal purposes, or there may be fields that are used to communicate information between the form the Transaction Manager.
You may override the usual Clear Hidden Data on Submit behavior on the Submit button by selecting the Preserve If Not Visible option in the Data Clearing Policy property on the Data tab.
Note: There is a special Composer field named "Data Field". This is just a regular Text Field, but is invisible by default, and has Data Clearing Policy set to Preserve. It is intended to store (and submit) internal data rather than user-visible data.

Clear Data when Hidden

Another option it to clear the data as soon as the field is made invisible. This is controlled by an advanced policy property in the Rules area. In most cases this is set to false. The same policy also exists for Read-only fields, although the need for this is rare.
There are pros and cons to this approach:
Pro: It is quite obvious to the end-user that their hidden data is cleared, because if they re-show the data, it will be cleared.

Con: If the user accidentally hides a field or section they have already entered information into, that information will be lost, to their annoyance.
Note: If a field is hidden implicitly (for example, because it is on a wizard page that is currently hidden), the data will not be cleared.

When to Clear Data

The decision whether to clear data from fields on their first hiding or on submission of the form, is a policy decision for each organization.
In both cases, privacy is maintained.
Clear-on-submit generally provides a better user experience.
Clear-on-hide provides a more obvious demonstration to the end-user that their private data will be removed.
Composer's defaults focus on user experience, and so hidden fields clear on submit by default. You may change these defaults in your organization or in a particular form if desired. Please contact Avoka for details on how to set this up in your style-sheets.

Validation (Composer v4.3)

Overview

A validation rule verifies the data entered into a form. The simplest and most common type of validations are single field validations, which validate the data in a single field. More complex validations use the values of several different fields.
Note: Validation rules fire only once the user has entered data into the field. You may want to combine a Validation Rule with a Mandatory Rule to ensure that the user enters valid data.
Composer provides several types of standard validation types:

Fixed Length

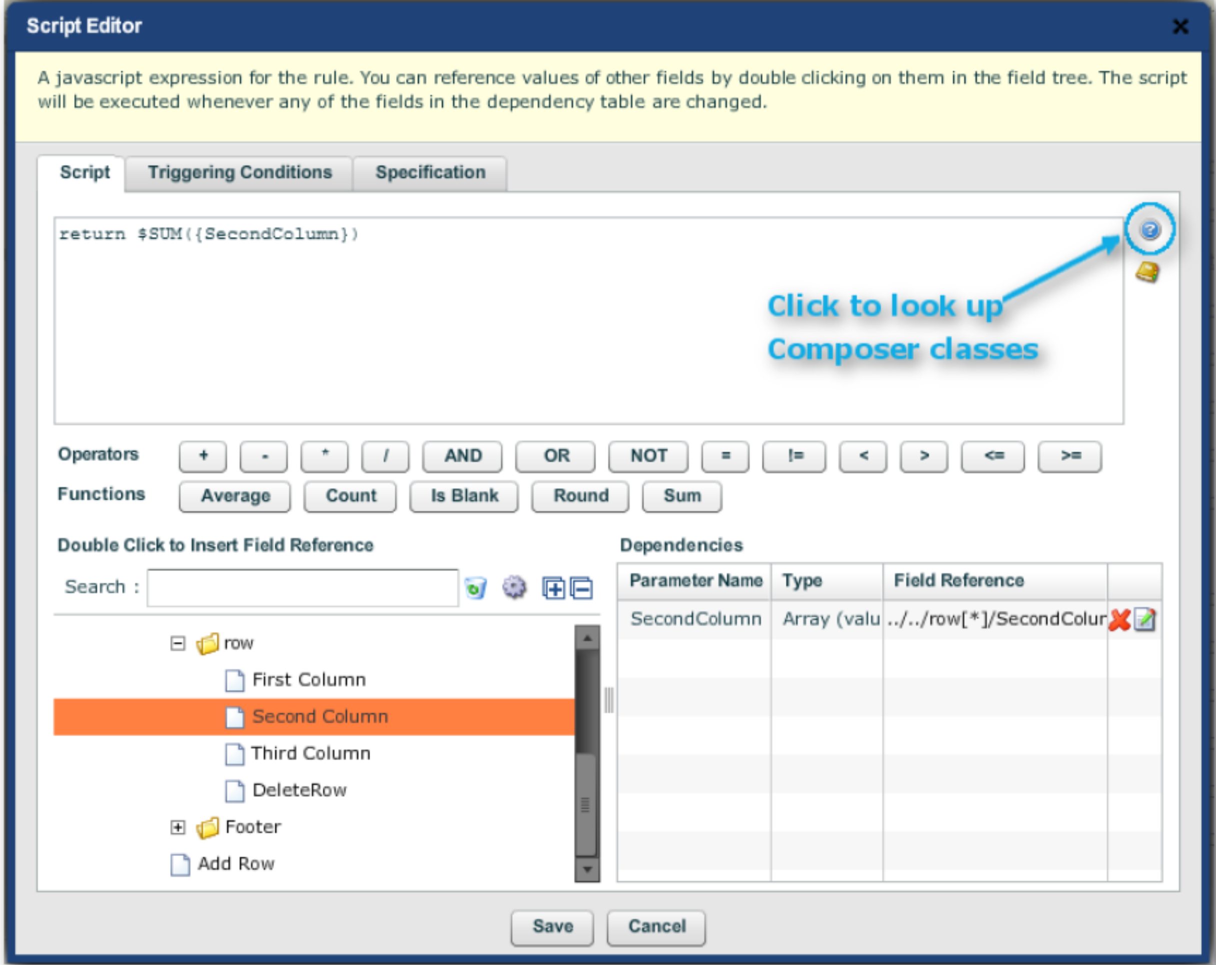
The fixed length rule ensures that the length of the data (ie the number of characters typed) is exactly the value specified. This is usually useful for a customer identification number or similar that is a fixed length.

Regular Expression

A regular expression is a sophisticated, industry-standard way of specifying a validation pattern. An explanation of regular expressions is beyond the scope of this document, but there are numerous books and web sites that you can refer to. You can also use search engines to find standard regular expressions for common types of data. An example of some regular expressions are:
^[a-zA-Z0-9_-\.]@[a-zA-Z0-9_-\.]\.[a-zA-Z]{2,4}$ - email address
^http\://[a-zA-Z0-9-\.]+\.[a-zA-Z]{2,3}(/\S*)?$ - URL A good resource for regular expressions is: {+}http://regexlib.com/+.
Online regular expression editor and evaluator can be found at: {+}http://gskinner.com/RegExr/+ and {+}http://+ www.debuggex.com/.
The download from following site can also be helpful: {+}http://www.weitz.de/regex-coach/+.

Validation Script

You can write a JavaScript function of arbitrary complexity to evaluate the value entered into any field. In order to write a validation script, you will need to have some minimal JavaScript knowledge. If you return false, the validation will fail.
Here is a sample validation script:
var myvalue = sfc.convertToString(sfc.getRawValue(me)); //1 if (myvalue.length < 3) return false; //2
if (myvalue.match("[0-9]")) return false; //3 return true; //4
Line 1: This is the usual way to obtain the value of the current field. The "sfc" functions are standard functions that are provided within the Composer libraries. For a full listing, click the "?" icon in the Script editor.



Icon in Script Editor to access Composer classes listing

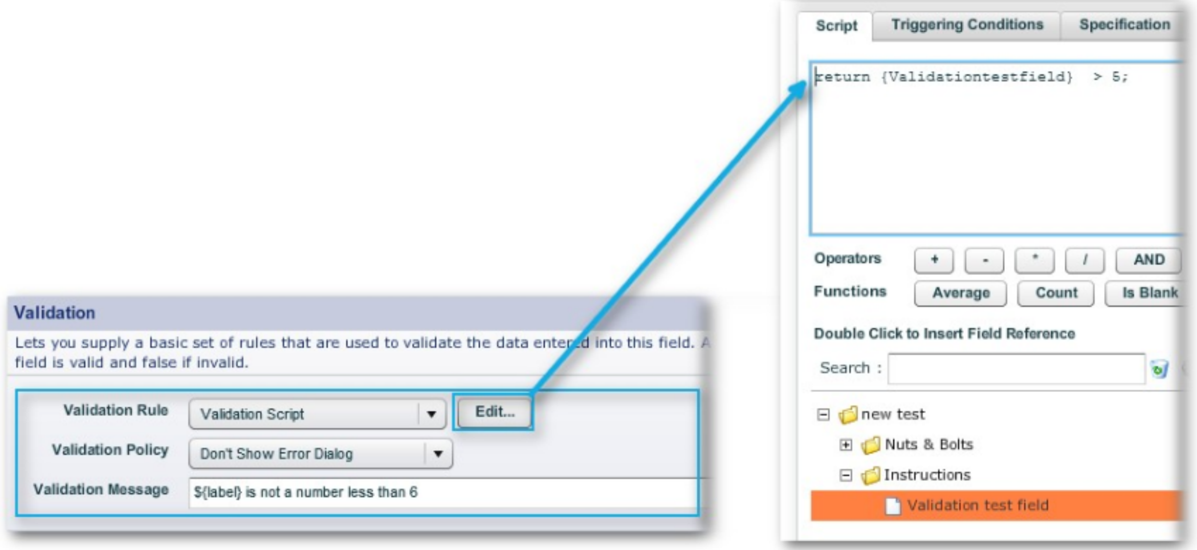
Line 2: This line uses the JavaScript built-in length operator to test whether the field contains at least 2 characters. Returning false causes the validation to fail
Line 3: This line uses a regular expression to test whether there are any numeric characters in the field. If there are any numeric values, the script returns false.
Line 4: If none of the preceding lines return a false, then return true. This will mean that the validation rule passes.

Validation and Hidden fields

Validation rules are always fired, even if the field is invisible, or is hidden because of some other business rule. This is to enable "hidden" validation rules that aren't represented as a widget in the form.
If you have a field that may become hidden, and you want to ensure that the validation rule on the field doesn't fire when the field is hidden, one way to ensure this is to set the "When Hiding - Clear Data" policy. Validation rules don't fire on empty fields.
Note: This is different to the behavior of mandatory fields. Mandatory fields become non-mandatory when hidden.

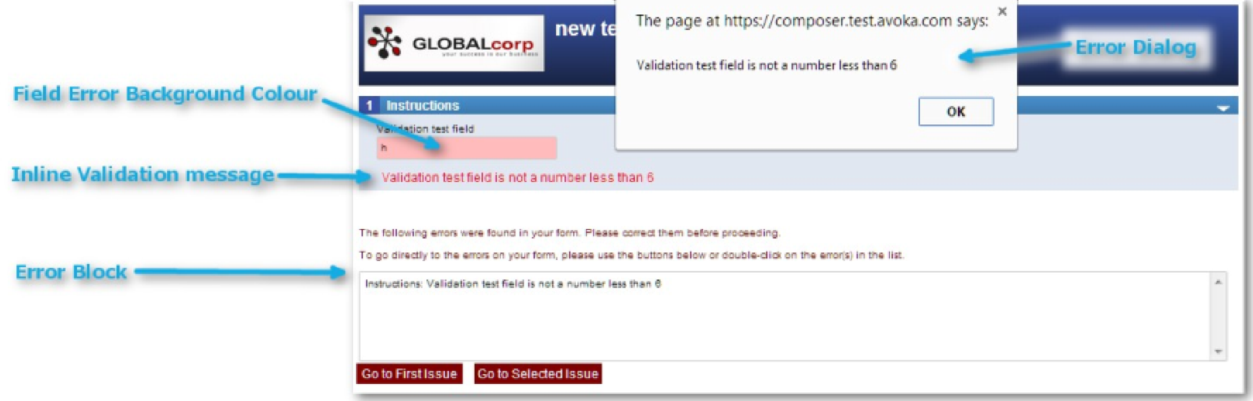
Validation Messages

There are 3 types of validation messages:
In the Error Block of the form. Error blocks do not display on mobile devices.
As a dialog, which displays on the failure of the element's validation. Users have to "OK" to continue and so these are rather annoying and intrusive. Inline validation which is the most convenient of the 3.
All 3 are turned on by default.
Here is a test field set up to have a trivial validation test. The Error Dialog is left on. The Inline Validation is left on the default setting for the form.



Validation settings for a text field

In the Desktop Preview, the form behaves as follows:



Resultant Validation messages on the Desktop

The Error Dialog is intrusive, requiring users to click on the "OK" button. You can turn it off, but only for each widget individually, by "Edit Properties -> Rules -

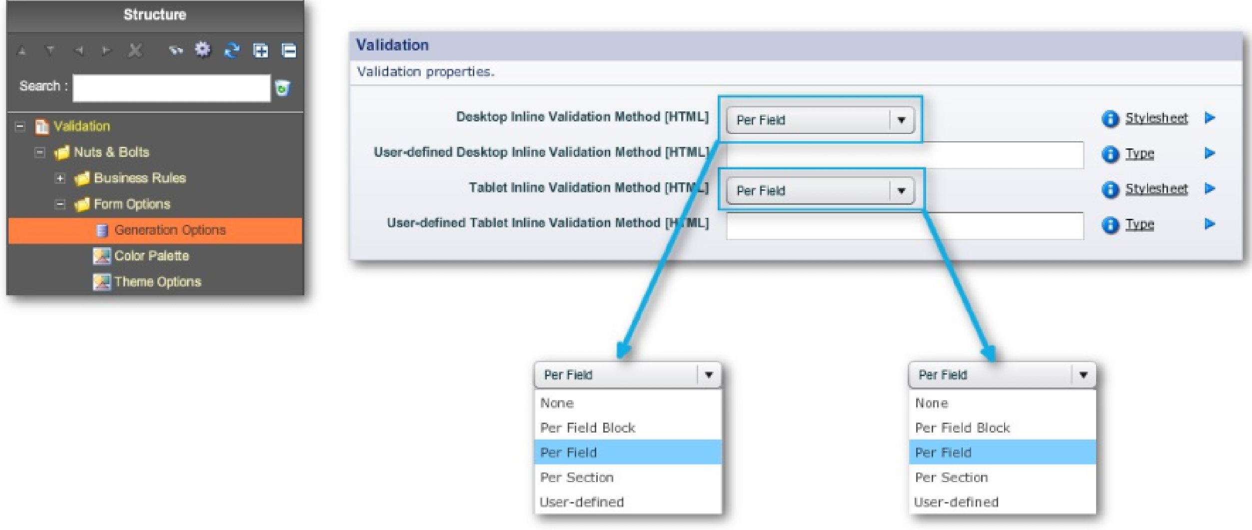
> Validation -> Validation -> Validation Policy -> Don't Show Error Dialog".
The Error Block does not appear on mobile devices.

In Line Validation

Inline Validation is really another instance of Dynamic Data, that is: if a field's validation test fails, and inline validation is turned on, the message to the user is displayed without the form page reloading.
The following examples have the Error Dialog turned off for each widget. The Error Block remains on, though will only be visible for the desktop previews.

Inline Validation Settings

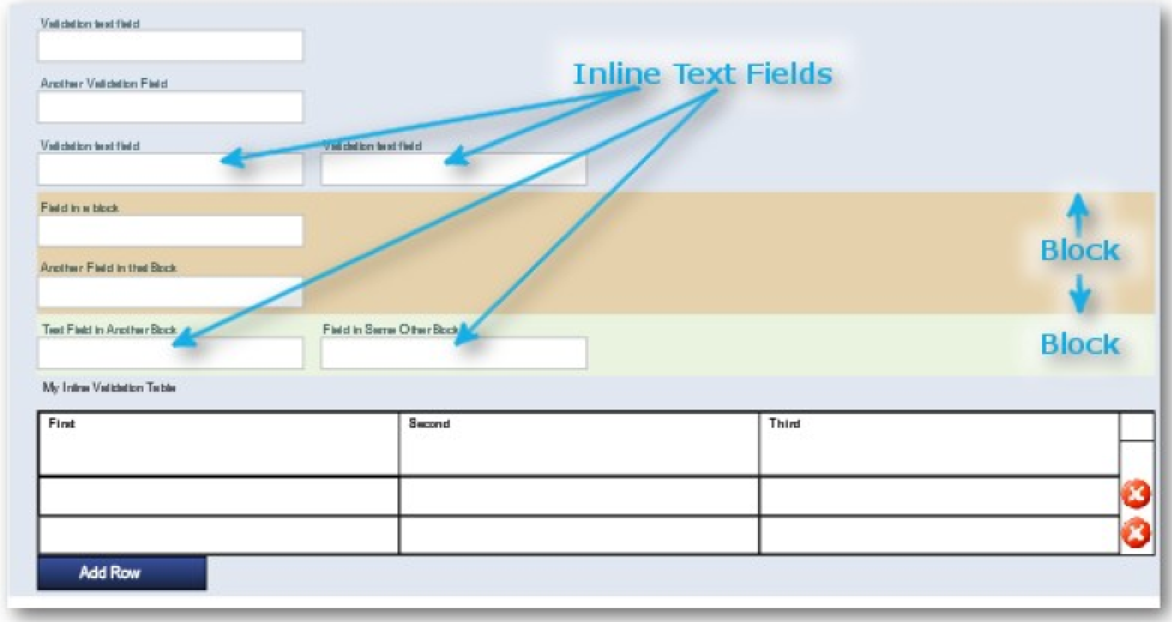
These are found in "Nuts & Bolts -> Form Options -> Generation Options -> Edit Properties -> Properties -> HTML Generation -> Validation", as shown below:



Some of the important form-wide HTML Inline Validation options

Inline Validation Example

The form used here, though very simple, embodies the various inline validation behaviors for the desktop and on mobile devices. Its salient features are:
2 blocks in the first section (delineated by background colors)
Text fields on separate lines or inline (as inline validation behaves differently for these)
A simple table (to show how inline validation methods behave on the desktop and in the various modes of mobile devices).
The Responsive Rules are the defaults, so that on an iPad, Landscape is below Threshold 2; portrait, below Threshold 3.



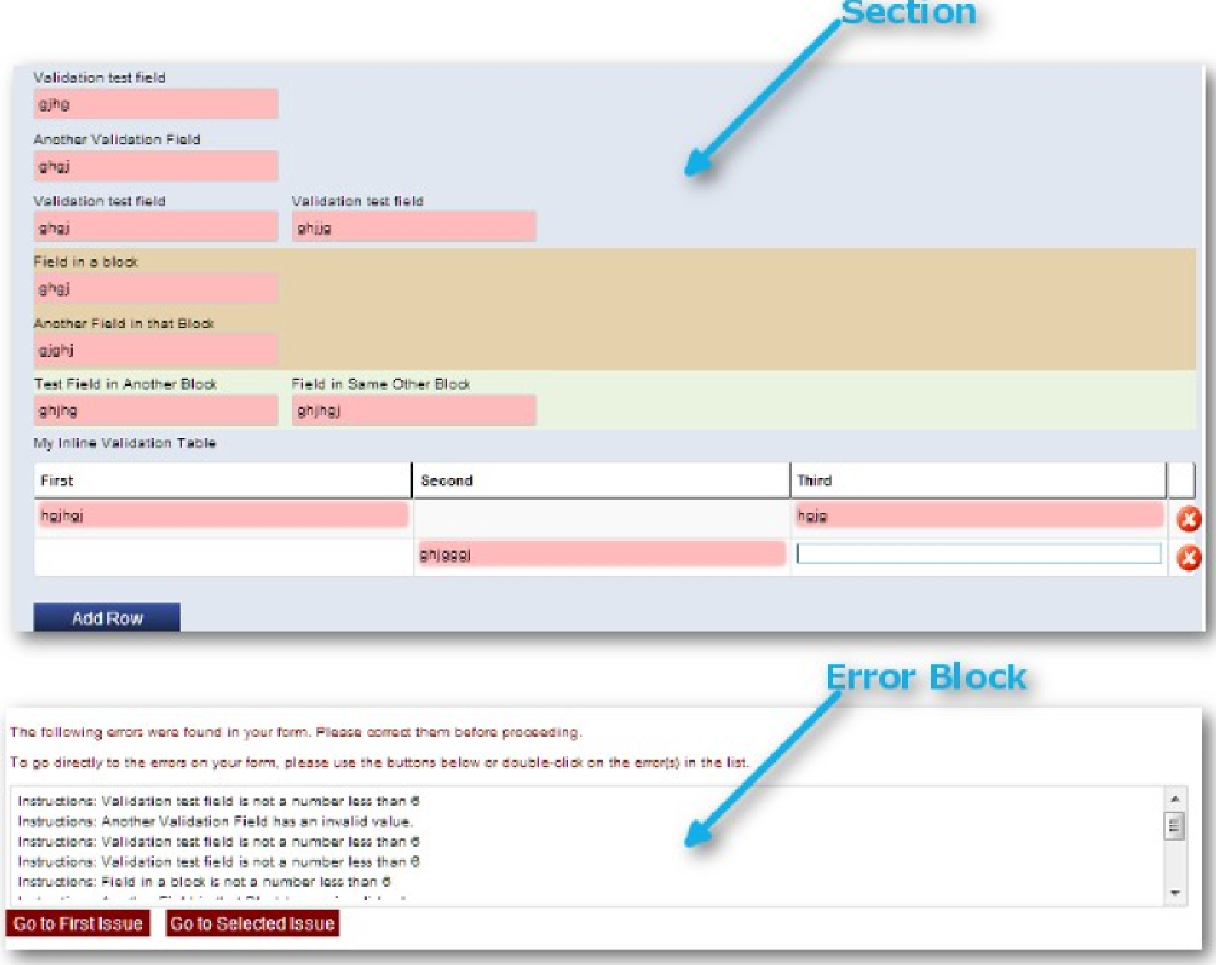
Wireframe of form used to demonstrate Inline Validation

Desktop Inline Validation Method [HTML]

In all cases, we have turned off the Error Dialog for convenience for each element at "Edit Properties -> Rules -> Validation -> Validation -> Validation Rule -> Don't Show Error Message".
All the fields with data entry are set to fail validation in this demo. The text of the Inline validation methods are set in each widget at "Edit Properties -> Rules -> Validation -> Validation Message".
Here are the Preview results for each desktop setting.

None, Desktop

The only messages the user gets is the Error Dialog when turned on at the widget-level, or the clickable Error Block list.

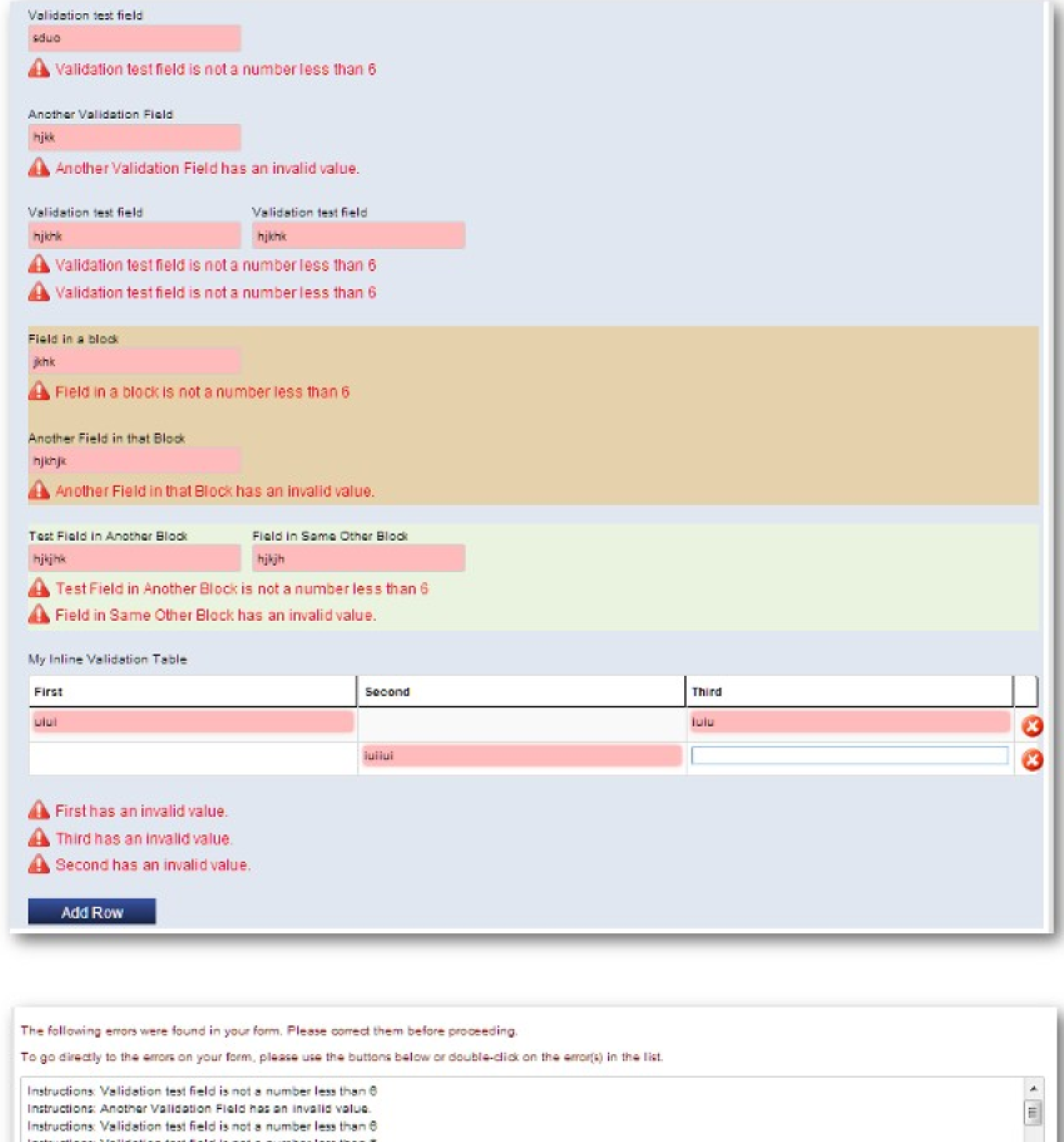


Inline Validation: Desktop, None

Per Field Block, Desktop

Inline Messages are under each field or line of fields
The same behaviour for fields within blocks (see the two blocks marked out with background colours in the example below)

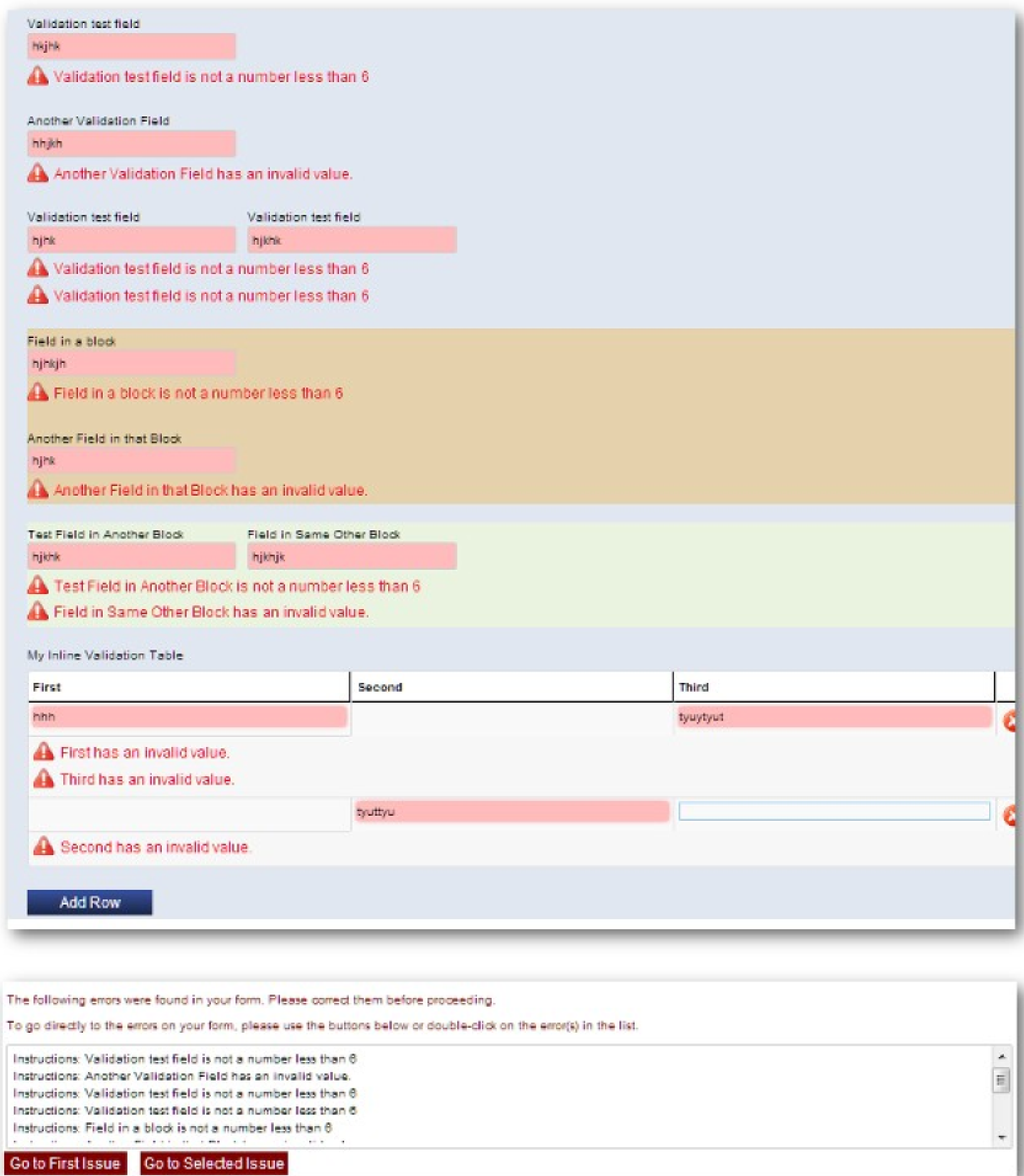
Inline messages for elements in tables get listed in a group below the table but above the standard "Add" button.



Inline Validation: Desktop, Per Field Block

Per Field, Desktop

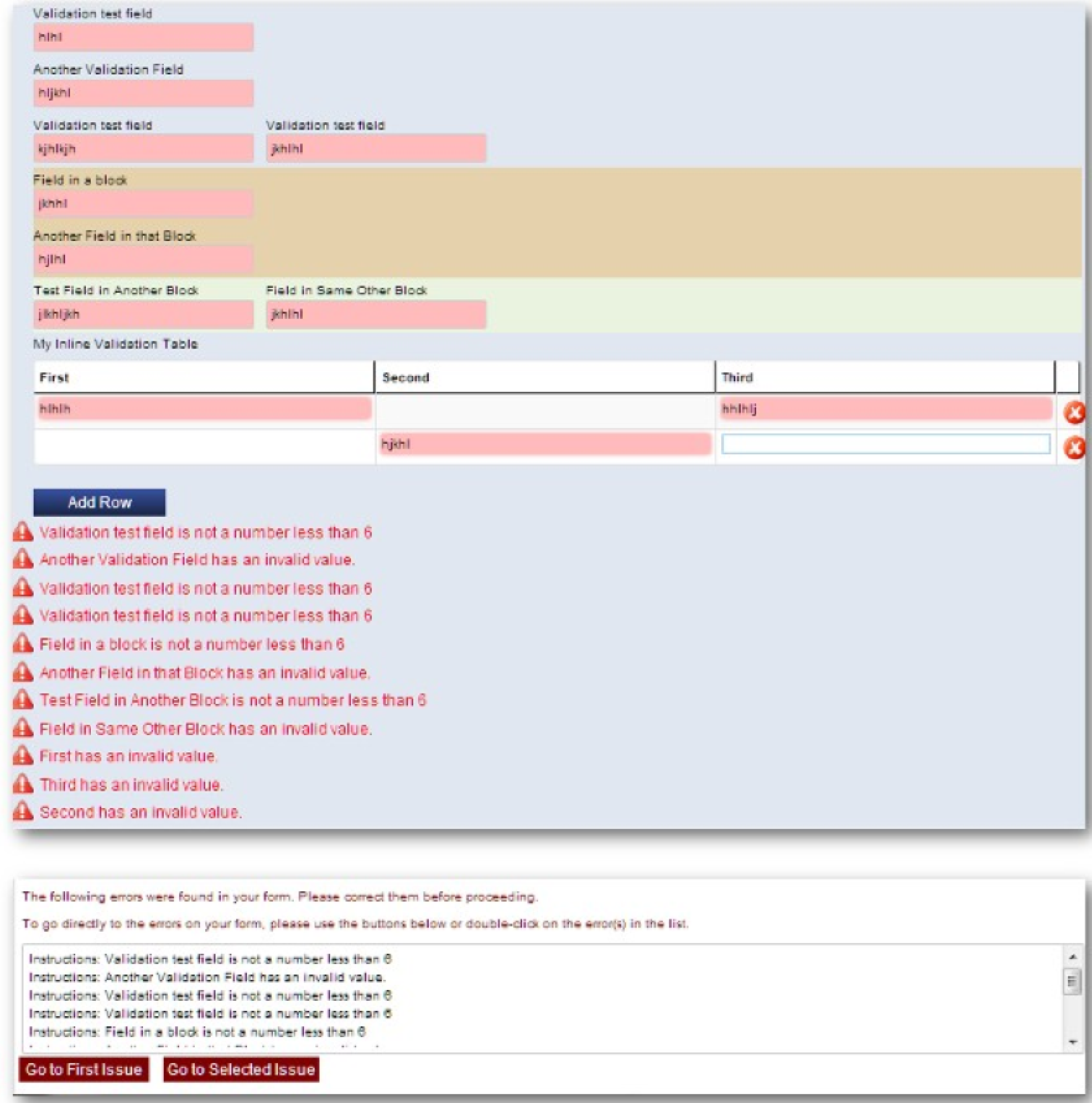
The same beaviour as the "Per Field" setting, except for
Tables, where the messages for each row come in a group under that row.



Inline Validation: Desktop, Per Field

Per Section, Desktop

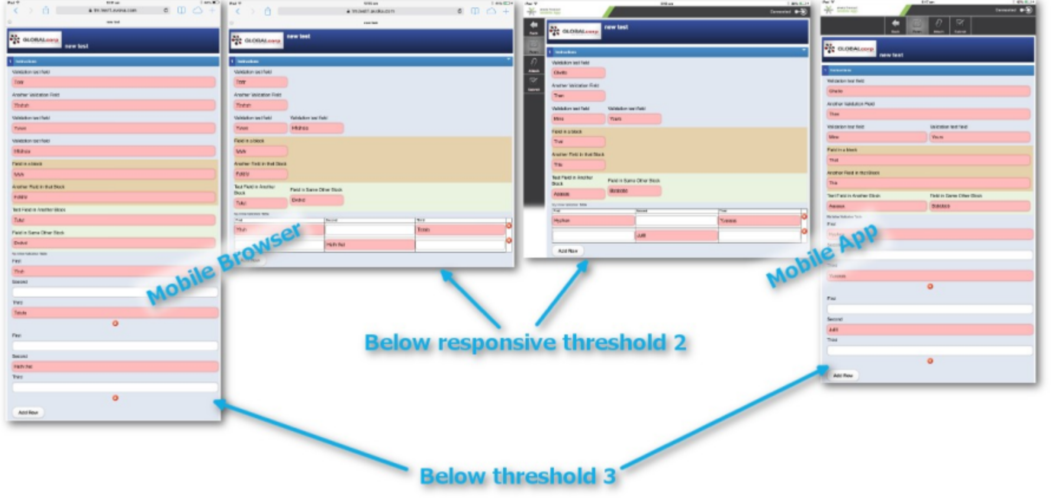
All the Inline Messages in a section come in a group at the end of each section.



Inline Validation: Desktop, Per Section

Mobile Inline Validation Method [HTML]

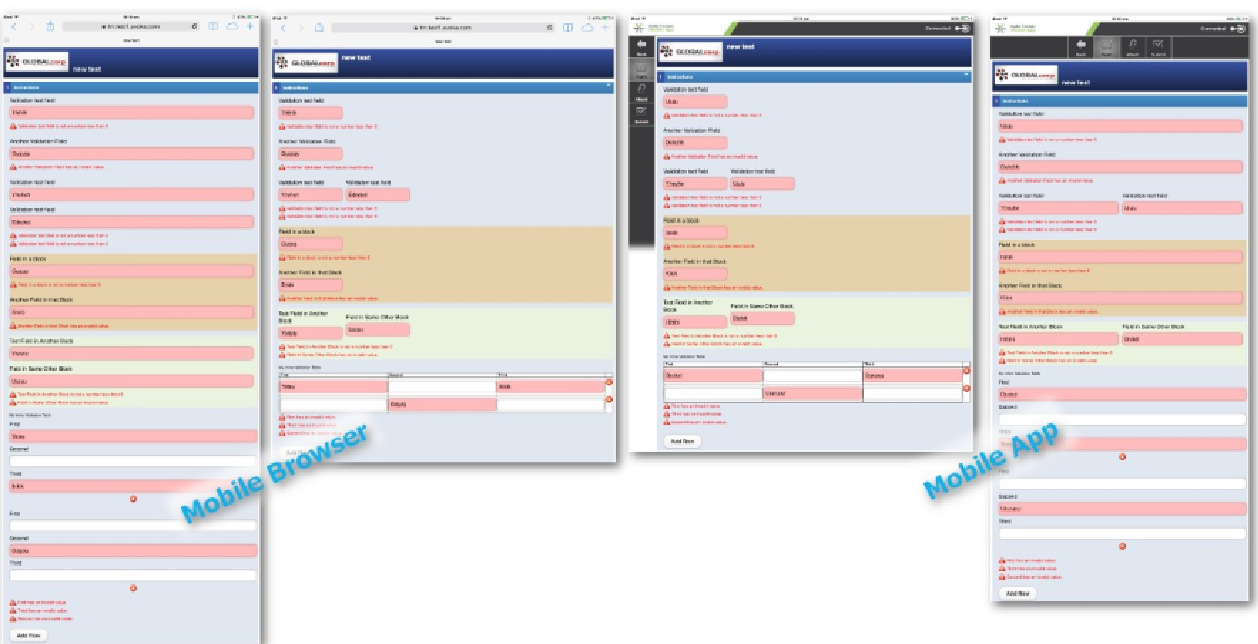
The mobile settings are independent of the desktop settings, and apply equally to forms accessed through viewing the form either in a mobile browser or in the the Mobile App .



None, Mobile

Mobile: Inline Validation method = None

Per Field Block, Mobile

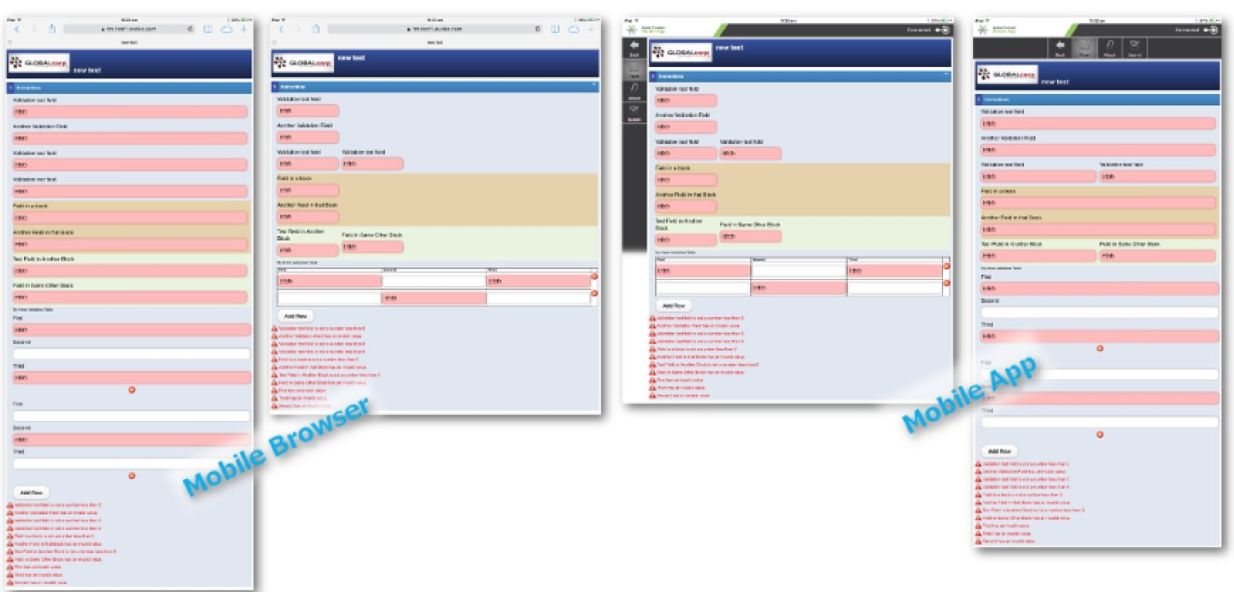


Mobile: Inline Validation method = per Field Block

Per Field, Mobile

Mobile: Inline Validation = per Field

Per Section, Mobile



Mobile: Inline Validation method = per Section

Analytics (Composer v4.3)

Overview

Transaction Manager, the backend of the Transact platforms forms, will obviously log information on form submissions:
Data on the original form request:
o The user's IP address
o the agent string of the browser, from which holds information on the Operating System (Windows, OS X, iOS, Android)
The version number of the system
the device (for example, Windows desktop, iPad, and so on)
the browser brand (i.e. Chrome, Firefox, Safari, Internet Explorer) and version number

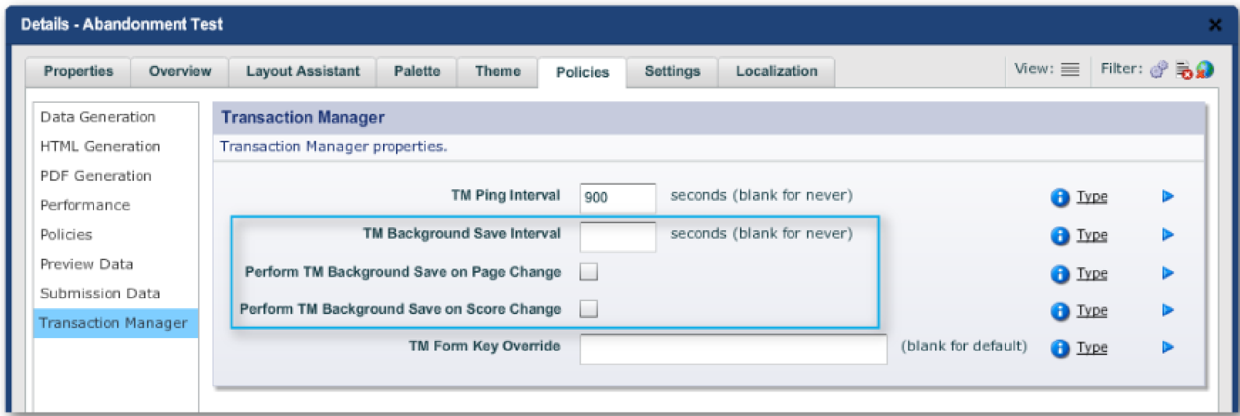
- The referring page (i.e. the page containing the link the user activated to reach the form) and portal
- The time stamp of the original request
- Data and keys on the session, cookies and other technical information Whether the form was saved or abandoned, including
- When the form was submitted (if ever)

The form's rendering time (vital information for optimizing form performance)
TM stores the data on each form request and also consolidates these data and renders them in various graphs and dashboards in TM.

Composer's Role in Analytics

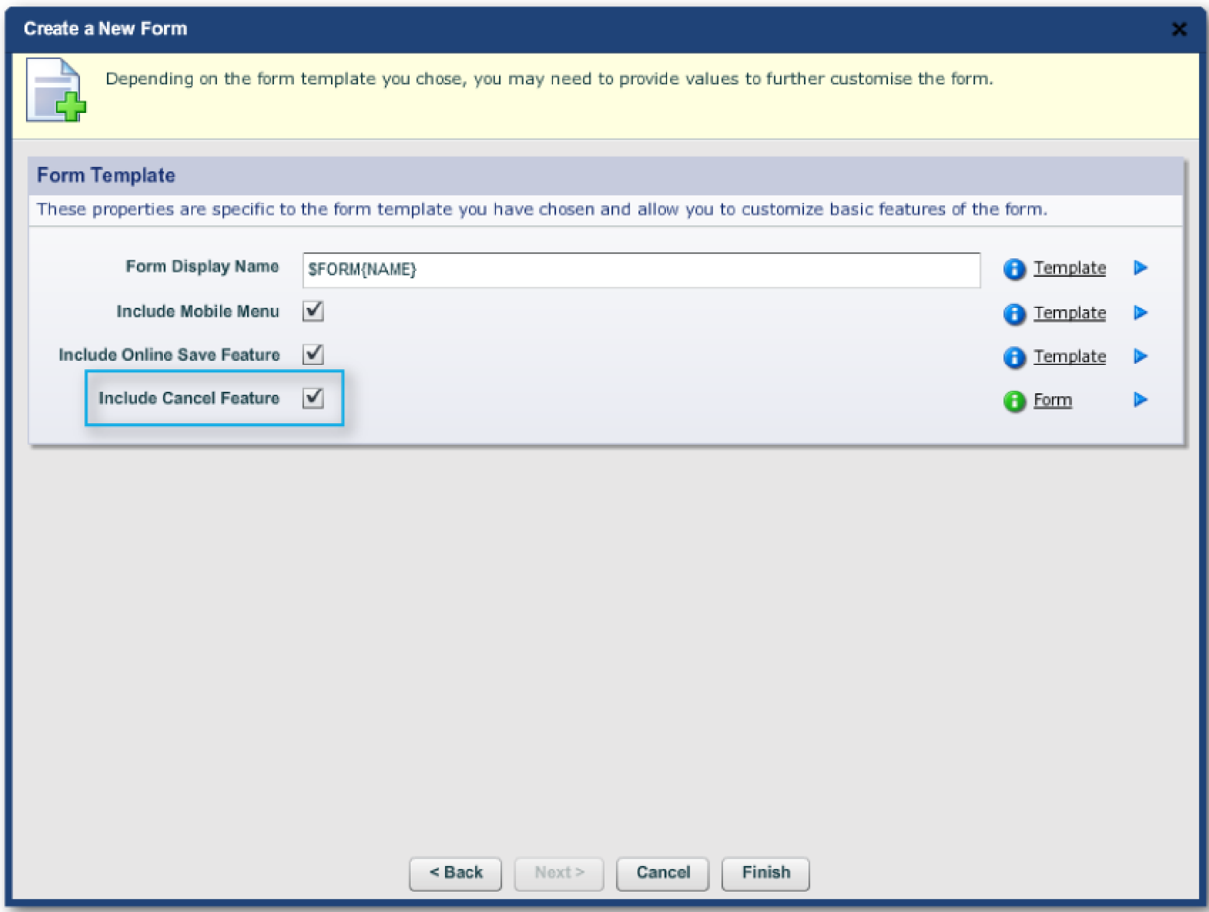
Background Saving

Time was when the data entered onto a form was not saved back to the server until the end user submitted the form. Now, Composer has tools to:
Configure background saving
where data is sent dynamically back to Transaction Manager before the form is submitted.
The data can be either the values entered onto the form or an arbitrary numerical score where you ascribe various numerical values to some or all of the fields on the form
Background saving can be set to take place at regular time intervals, or when the user navigates to another wizard-style page, or when the numerical score changes value
You may not want, for privacy reasons, to retain the actual data entered on the form by an end user who has not yet submitted this data, but still want to have data on where the user reached in the form before abandoning it so that you can modify the form so more users successfully get to the submit stage of entry.
This substantially increases the value of the form to your business. Without such a feedback mechanism, you will not be able to make as many meaningful improvements to the form.
You activate background saving through "Structure -> <Top item of the structure tree> Edit Properties -> Policies tab -> Transaction Manager menu item -> Transaction Manager panel:



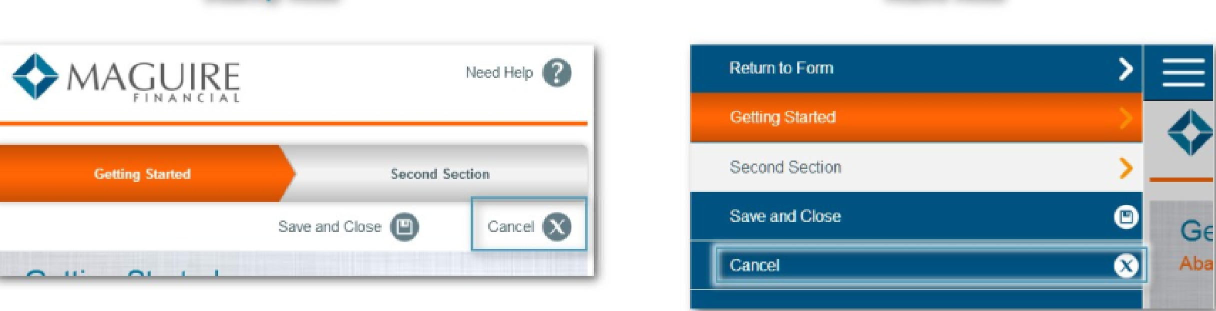
The Cancel Button in Maguire Forms

The Maguire template forms have a Cancel facility. This must either be activated when the new form is created, the Maguire template chosen in the "Create a New Form" Wizard and the "Include Cancel Feature" checked in the last page of the wizard dialog.



The resulting feature is a "Cancel" button. Currently this is **not** visible in the Wireframe, only in the Preview (here in both desktop mode and in triggered mobile responsive view):

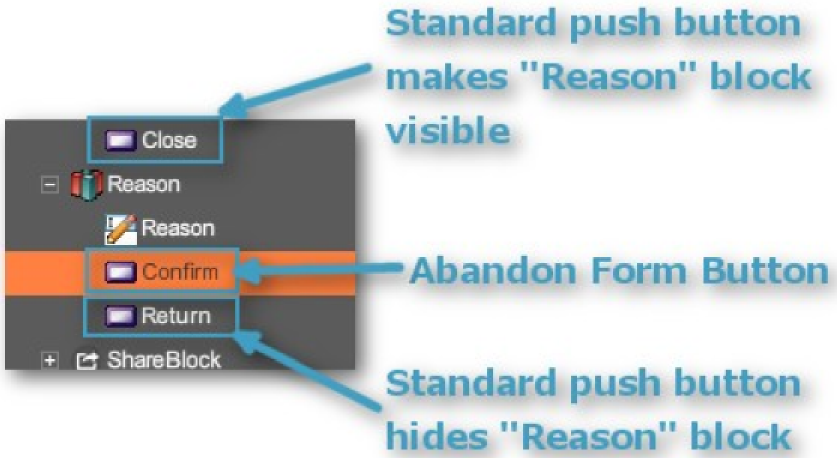
Desktop Mode Mobile Mode



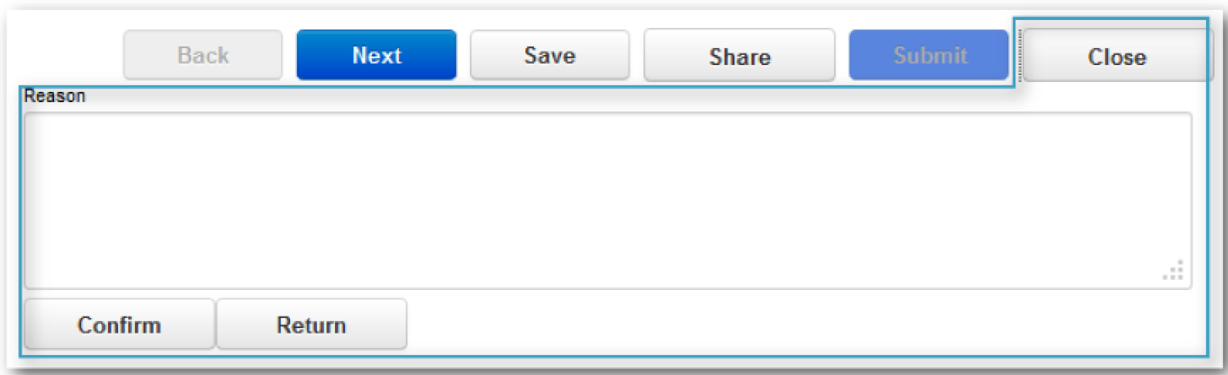
Clicking on "Cancel" results in the form's data being saved to TM and the form's data and analytics being displayed in TM under "Home Dashboard -> Abandoned Transactions".

The Abandon Widget in non-Maguire Forms

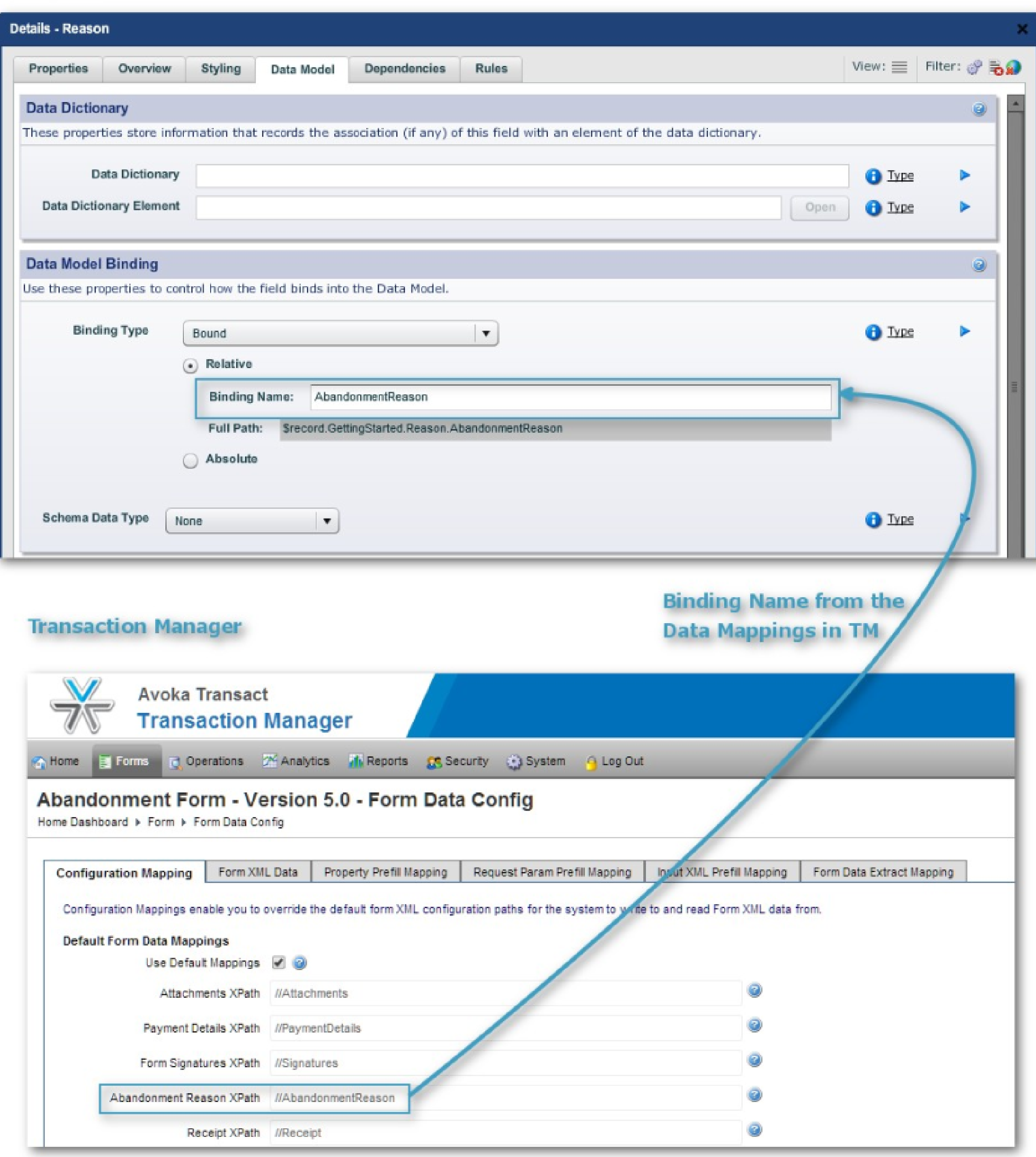
The same function is available in non-Maguire templated forms. Place the "Palette -> Field Types -> Abandon Form Button" on your form. Probably the best places are in the Header and Footers (which are exposed in the Advanced Mode of the Structure Palette as the "Form Header" and "Form Footer" blocks below the Nuts & Bolts block.
You can even add a "Reason" text block for users to give the reasons for abandoning the form. We recommend the following technique to make the text block hidden and then become visible when users click on a "Close" button.



Use Click Action scripts to control the show/hide behavior of the "Close" and "Return" standard buttons. The most basic Click Action scripts to accomplish this (with no refinements whatsoever) are:
For the "Close" button: sfc.updateVisibility({Reason}, true, "exclude"); And the "Return":
sfc.updateVisibility({Reason}, false, "exclude");
On the form, the fields look as follows (when placed into the "Form Footer" block"):



By careful configuration of the "Reason" text area, you can make the contents of this field be listed in the TM "Home -> Operations -> Abandoned Transactions" page in the Abandon Reason column. Use the Data Model Binding panel ("Edit Properties -> Data Model tab -> Data Model Binding panel").



The Binding Name used is one of the Default Form Data Mappings in TM. The double slash "//" just means a wildcard: in other words, the field mapped to AbandonmentReason can be anywhere in the form's structure. If there are more than one such field, TM will use the first it encounters in the tree. You should be setting the mapping unique for best results.

Scoring

We have seen how you can retain data from an abandoned form. However, due to privacy concerns, may not actually want to retain this data, and yet still get some metrics on the abandonded form so as to identify usability issues which could have contributed to its being abandoned in the first place. One way is to score the form to see how far the user managed to get to before abandoning or canceling. Scoring could also be useful in your analysis, even when you retain the incomplete data.
All you need do to set up scoring in Composer is add some simple scripting to "Structure Panel -> <form name> -> Nuts & Bolts -> Transaction Manager Support -> TransactionScore -> Edit Properties -> Rules tab -> Calculation panel":

Details - TransactionScore

Properties | Overview | Styling | Data Model | Dependencies | Rules

View: Filter:

Calculation
Editability
Mandatory
Validation
Visibility
Responsive

Calculation

Lets you set the value of this field based on a calculation.

Calculation Rule: Script Based — Edit... — Form

Script Editor

A javascript expression for the rule. You can reference values of other fields by double clicking on them in the field tree.

Script | Triggering Conditions | Specification

```
score=0;
if ( ! $ISBLANK({FirstName})) {score = score + 10;};
if ( ! $ISBLANK({suburb})) {score = score + 15;};
if ( ! $ISBLANK({state})) {score = score + 5;};
if ( ! $ISBLANK({FirstName})) {score = score + 10;};
if ( ! $ISBLANK({MiddleInitial})) {score = score + 5;};
if ( ! $ISBLANK({LastName})) {score = score + 10;};
return score;
```

Operators: + - * / AND OR NOT = != < > <= >=

Functions: Average Count Is Blank Phase Value Round Collaboration Step Sum

Double Click to Insert Field Reference

Search :

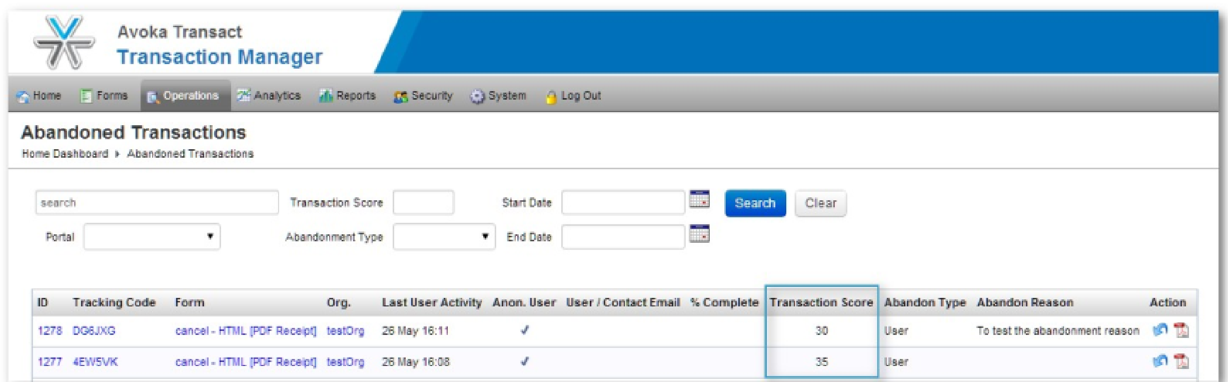
TransactionScore
TemplateVersionNumber
TrackingCode
SFM Attachment Controller
LastAttachmentId
SFM Payment Controller
LastPaymentId

Dependencies

Parameter Name	Type	Field Reference
FirstName	String	../../../GettingStarted
MiddleInitial	String	../../../GettingStarted
LastName	String	../../../GettingStarted
addressLine1	String	../../../Next/_outerAr
suburb	String	../../../Next/_outerAr
state	String	../../../Next/_outerAr

Save Cancel

The above script is a very simple and unsophisticated example. Whatever the script used. it must return an integer, which will, after you have exported the form, then be displayed in Transaction Manager under "Operations -> Abandoned Transactions":



Field Types (Composer v4.3)

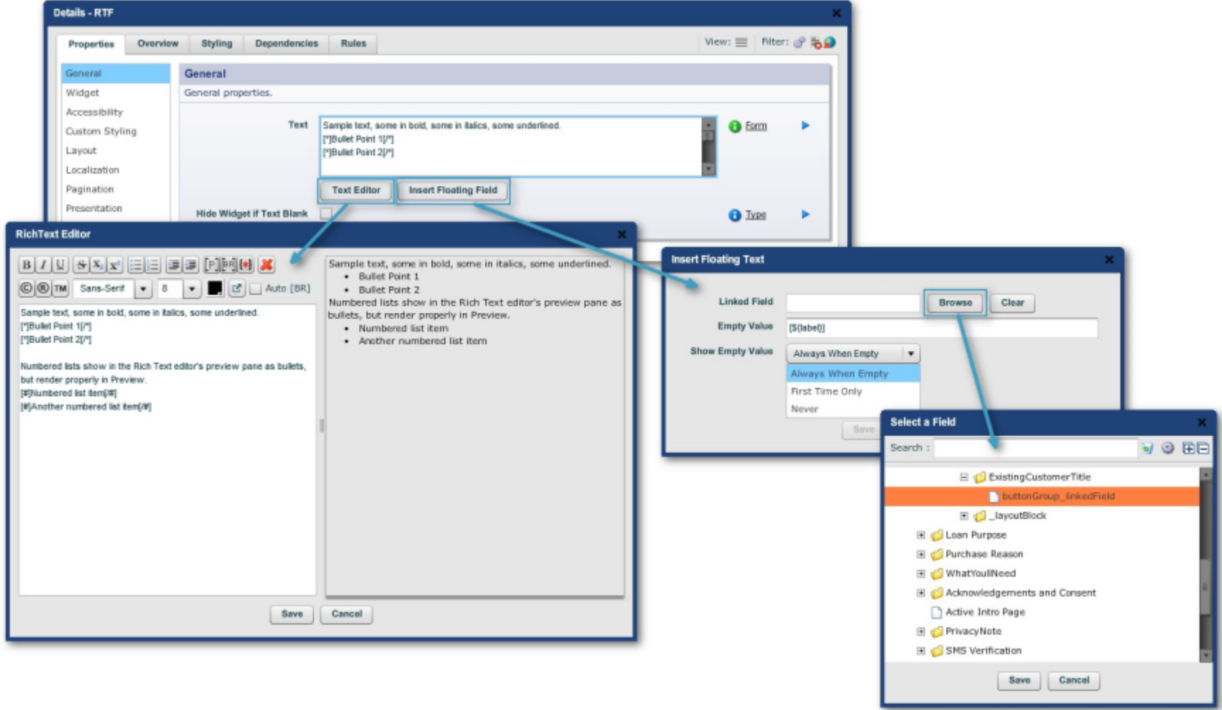
There are a few field types (or "widgets") that require further documentation in this manual. They are:
Rich Test Fields Cascading Dropdowns Prepoplated Dropdowns

Rich Text Fields

A Rich Text is a read-only widget that has styling, such as bolding, italics, fonts, and colors, embeded into the text. Rich Text is often used for explanatory or legal text in forms; many other fields also support Rich Text, such as the Labels for all fields and Section Help Text.
You can embed the values of other fields within a Rich Text Field. **This replaces the legacy widget "Floating Field"** which no longer ships in the Palette.
You control the format of the text by embedding "tags" or "markup" into the text. We will list these below. Tags must be properly opened and closed as a pair. If the pair is incorrect, the text will not display correctly. Rich Text may render slightly differently in the wireframe, PDF and HTML. This is because the underlying implementations on each platform are not the same. Composer attempts to make the text appear as close as possible on all platforms — if you discover discrepancies, please notify us with a sample.

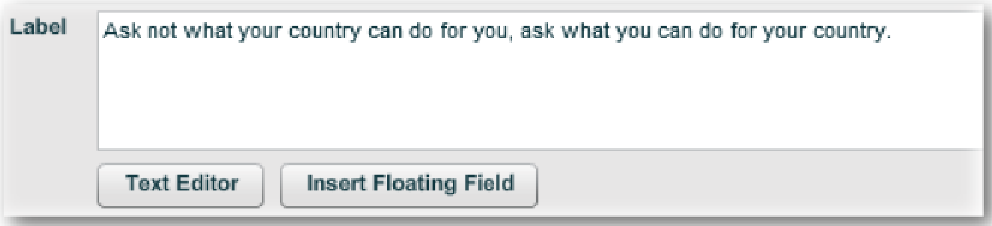
Rich Text Editor

The Rich Text Editor ("Text [Rich}" widget, "Edit Properties -> Properties tab -> General -> General panel -> provides simple ways of embedding rich text tags into your text.



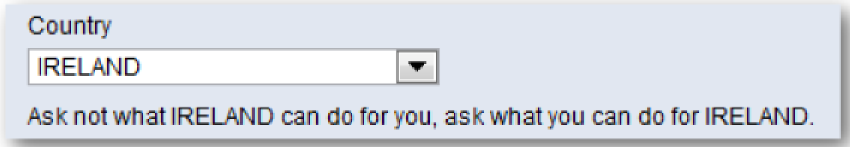
Consider the famous quotation:
Ask not what your country can do for you; ask what you can do for your country.
Instead, use a Country Dropdown List at the top of the form asking for which country. And then depending on the country choosem, the quotation reads:

Ask not what America can do for you, ask what you can do for America.
or
Ask not what Canada can do for you, ask what you can do for Canada.
And so on.
This is quite straight forward to do.
Create the Country Dropdown (or any other field whose value you want to appear in the text.) This field can appear anywhere in the form.
Create a Rich Text field as shown below:



Click on the "Insert Floating Field" button, and point at the Country Dropdown created earlier.
You may also want to update the other two values in the dialog - these control what is displayed in the Floating Field if the Country Dropdown has a blank value.

Copy and paste the floating field reference into the two locations, so that it looks like this:
Ask not what $FLOAT{Country_linkedField} can do for you, ask what you can do for $FLOAT{Country_linkedField}.
Test your form. You should get something that looks like this:



Note that Composer has created a hidden Floating field underneath the Rich Text field. Do not delete this field, as it temporarily stores the value to be injected into the Rich Text, and is linked to the real value. You can modify the formatted value of this field if desired.The Rich Text toolbar icons can be used in two ways:
To apply a tag to an existing piece of text, select the text, and then click the icon.
If you click the icon when no text is selected, the matching tags will be placed into your text, with the cursor positioned for you to start typing.
The Auto [BR] checkbox, if selected, will automatically add a "[BR/]" tag into your text every time you press the Enter key.
Tip: It is strongly recommended not to specify custom fonts, as these may not render correctly on different systems.

Tag Reference

There are two types of tags, those that affect individual characters within the text, and those that turn whole paragraphs into list items (either bullets or numbered items).

Character tags

Tag	Formatting	Sample	Result
[i]..[/i]	Italics	Hello [i]World[/i]	Hello *World*
[b]..[/b]	Bold	Hello [b]World[/b]	Hello **World**
[u]..[/u]	Underline	Hello [u]World[/u]	Hello World
[x]..[/x]	Strike Thru	Hello [x]World[/x]	Hello ~~World~~
[sub][/sub]	Subscript	Hello [sub]World[/sub]	Hello world
[sup][/sup]	Superscript	Hello [sup]World[/sup]	Hello world
[colour:#00FF99][/colour]	Color	[colour:#ff0000]Red[/ colour] and [color:rgb (0,0,255)]blue[/ color]	Red and blue
[font:Arial][/font]	Font type	My font is [font:Myriad Pro]Myriad Pro[/font]	Fonts available depend on the fonts available in your destination system, and vary between PDF and HTML. See separate topic on Fonts.
[fontsize:9pt][/fontsize]	Font size	My font is so [fontsize:16pt]big[/ fontsize]	My font is so big.
[link:][/link]	Link	[link:[http:// www.avoka.com]Avoka[/ link]	Avoka (Links work in HTML and more recent releases of Adobe Reader.)
[copy] [reg] [trade]	Symbol	Copyright [copy] 2010 Avoka Technologies.	Copyright © 2010 Avoka Technologies.

Tag	Formatting	Sample	Result
${options:policy. mandatory .marker}	Mandatory marker. This is used to insert the mandatory marker (usually a red asterisk, but controlled via your stylesheet) into your help text.	All fields marked with ${options:policy. mandatory .marker} are mandatory.	All fields marked with * are mandatory.

Bullets, Lists and Paragraph Tags
Bullets and Lists affect an entire paragraph of text. The bullet character or numbering style is controlled via the properties and the stylesheet for your form.
Extra levels of indent are controlled via a leading dot.

Example	Sample
[]**USA[/]** [.]**CA[/]** [..]**San Francisco[/]** [.]**NY[/]** [..]**New York[/]** []**Australia[/]** [.]**NSW[/]** [..]**Sydney[/]**	USA • CA San Francisco • NY New York Australia • NSW Sydney
[#]USA[/#] [.#]CA[/#] [..#]San Francisco[/#] [.#]NY[/#] [..#]New York[/#] [#]Australia[/#] [.#]NSW[/#] [..#]Sydney[/#]	1. USA a. CA i. San Francisco b. NY i. New York 2. Australia a. NSW i. Sydney
[#reset]	Use this tag to reset the numbering list back to 1. The reset take will take effect in the next rich text field.
[p]Use the paragraph tags to wrap around multiple paragraphs of text. Each paragraph will be spaced from the each other.[/p] [p]This is another paragraph[/p]	Use the paragraph tags to wrap around multiple paragraphs of text. Each paragraph will be spaced from the each other. This is another paragraph
[.p]A paragraph. Notice the spacing[/p] [..p]One level of indent[/p] [...p]Two levels of indent[/p]	A paragraph. Notice the spacing One level of indent Two levels of indent
You can break [br/] text onto [br/]multiple lines without any spacing	You can break text onto multiple lines without any spacing

Auto-numbering is controlled by Sequences in Nuts & Bolts. This is an advanced topic, and usually configured for you in your style-sheet.
Paragraphs are optional, but are useful for longer blocks of rich text.

Radio Buttons

Radio buttons in Composer required a lot of documentation. No longer: Composer 4 now handles them gracefully.

Creating Radio Button Groups Manually

The Widget Palette now contains a "Smart" family of radio button elements:
Smart Radio Button
Smart Radio Button Group
To create a functioning set of radio buttons, you need the Group widget and a smart button widget for each of the buttons to be selected. For each smart button you place on the form, in the wizard dialog, use the Browse button to point to the Linked Button Group. (If you neglect to do this in the wizard dialog, you can point to the group in "Edit Properties -> Properties tab -> Data -> Data panel".)
If you want to pick up the selected value of the button group in, say, an inserted floating field in a Rich Text Field, always point to the button group.
Now you can take a whole button set, place them into a simple block and copy that block to another part of the form. This now works without any fuss.

Mandatory Radio Buttons

It doesn't really make sense to make an individual Radio Button mandatory. As you can see above, you can make the Button Group mandatory - this means that the user will have to select one of the buttons in the group. See Mandatory Block .

Calculating or Using the value of a Radio Button Group

If you need to calculate the value of a Radio Button Group, you need to place the calculation on the Button Group, rather than individual buttons. Please refer to Scripting and Dependencies .
If you need to use the value of a Radio Button in another rule, you can either treat individual buttons similarly to a checkbox (and use Is-Selected in the parameter) or you can refer to the value of the Radio Button Group.

Dropdown Menus

Use the "DropDown List" widget from the Palette.
A Dropdown List has two sets of values in the "Data" tab of the advanced properties dialog of the widget:

Display Values

This is a list of values that are displayed in the user interface.

Values

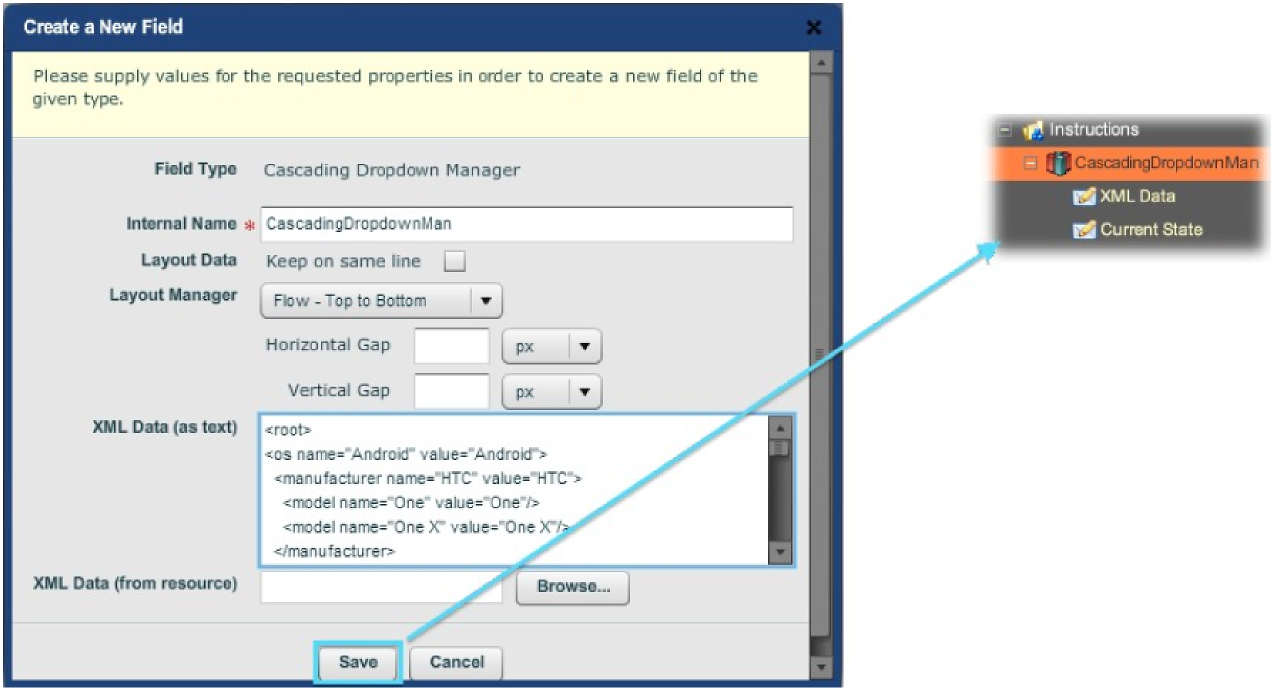
This is a list of underlying raw values the field will pass on to TM when a user selects a display value.
Display Values has a default value of ${values}. This means that by default, the set of Display Values is identical to the Values. This simplifies using Dropdown Lists, because you only need one set of lists.
However, sometimes it's useful to have a different display value versus the underlying value. For example, you may want to display countries in the Dropdown list using their full names such as Australia, United Kingdom and France, but store their underlying values using the two letter country abbreviations such as AU, UK, and FR. In this case, you can simply modify the values of Display Values.
Note:
The number of Display Values and Values must be identical.
You must delete the ${values} formula if you're going to add explicit Display Values.
If you use Composer's translation feature, only the Display Values will be translated. This means that any code that uses the Values will continue to work without modification.
Internally, the Values and Display Values properties are stored using a pipe character "|" as a separator. This means that you cannot use pipe characters in your Values or Display Values.
To access the Display Value in a calculate event, use the following script:
sfc.getFormattedValue(noderef)
In PDF, a Dropdown list can have a blank value even if "blank" is not one of the allowed values. (When a Dropdown list is first displayed, it's value is always blank unless initialized.) In HTML, a Dropdown List is only allowed to have a blank value if blank is one of the values in the Values list. In order to achieve compatibility between the two implementations of Dropdown List, Composer provides a "Automatically Add
Blank Value [HTML]" property which is checked by default. This has the added benefit of making the concept of a Mandatory Dropdown List meaningful.
Composer also provides a number of prepopulated dropdown lists. See Predefined Blocks . These include: Dropdown List [Countries]: A list of countries from around the world.
Dropdown List [Genders]: Contains 'Male' and 'Female'.
Dropdown List [States+Codes,United States]: A list of states found in the United States is displayed in the dropdown list, e.g. Alabama. This is what the form filler sees. However a corresponding two letter country code e.g. AL is stored in the underlying data.
Dropdown List [States,Australia]: A list of Australian states.
Dropdown List [States,United States]: A list of states found in the United States. In this dropdown list, the displayed value and 'stored' value are identical.
Dropdown List [Yes/No]: Contains 'Yes' and 'No' values.

Cascading Dropdowns

The first step is to define the behaviour of the set of cascading dropdowns with XML. Here is an example of three orders of cascading dropdowns: os -> manufacturer -> model.

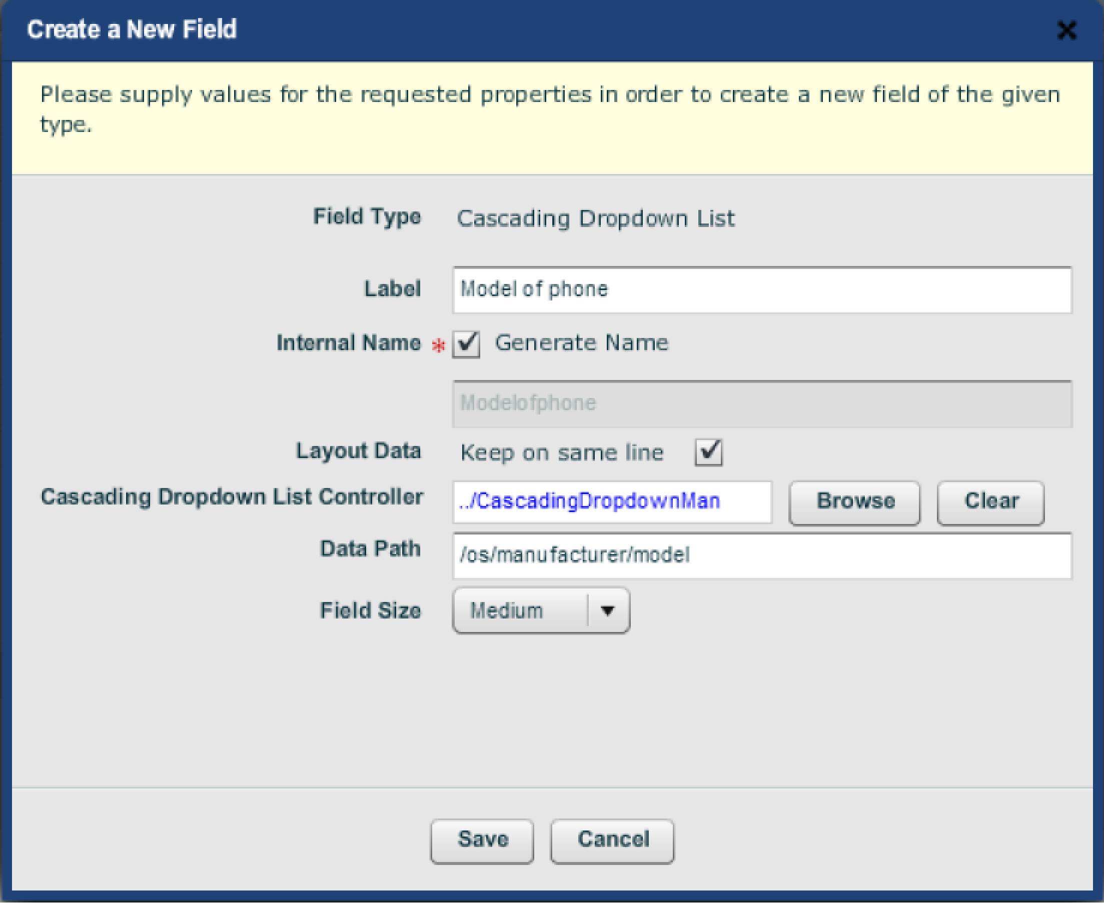
```
<root>
<os name="Android" value="Android">
<manufacturer name="HTC" value="HTC">
<model name="One" value="One"/>
<model name="One X" value="One X"/>
</manufacturer>
<manufacturer name="Samsung" value="Samsung">
<model name="Galaxy S3" value="Galaxy S3"/>
<model name="Galaxy Note 3" value="Galaxy Note 3"/>
<model name="Galaxy S4" value="Galaxy S4"/>
<model name="Galaxy Express" value="Galaxy Express"/>
<model name="Galaxy XCover 2" value="Galaxy XCover 2"/>
<model name="Galaxy Fame" value="Galaxy Fame"/>
</manufacturer>
</os>
<os name="Windows Phone 7" value="Windows Phone 7">
<manufacturer name="Nokia" value="Nokia">
<model name="Lumia" value="Lumia" />
<model name="Lumia 625" value="Lumia 625" />
</manufacturer>
<manufacturer name="Samsung" value="Samsung">
<model name="OMNIA 7" value="OMNIA 7" />
</manufacturer>
</os>
<os name="iOS" value="iOS">
<manufacturer name="Apple" value="Apple">
<model name="iPhone 5" value="iPhone 5"/>
<model name="iPhone 5S" value="iPhone 5S"/>
<model name="iPhone 5C Green" value="iPhone 5C Green"/>
<model name="iPhone 5C Blue" value="iPhone 5C Blue"/>
<model name="iPhone 5C Yellow" value="iPhone 5C Yellow"/>
<model name="iPhone 5C Pink" value="iPhone 5C Pink"/>
<model name="iPhone 5C White" value="iPhone 5C White"/>
</manufacturer>
</os>
</root>
```

Drop a Cascading Dropdown Manager widget onto a section (here, the default "Instructions") in the Structure pane. Paste the XML definition into the XML Data test field.



The Cascading Dropdown Manager predefined block dialog, with the XML pasted into the data textbox. "Save" creates the structure shown.

Drop three "Cascading Dropdown List" widgets into the structure, Here we have put the three widgets onto the section level of the form. Fill in their dialogs, giving the name (as it will appear on the form), point to the "Cascading Dropdown Manager Controller" and give the data path from the root as defined by the XML. Here is the dialog for the last of these lists:

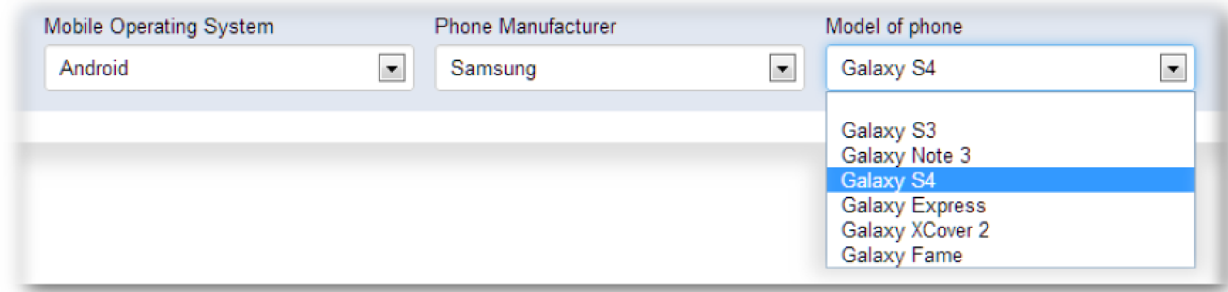


The dialog for creating the "model" level cascading dropdown list.

For convenience we have kept the lower two list widgets on the same line as the "os" top order. The final structure looks as follows:

The final structure on the form.

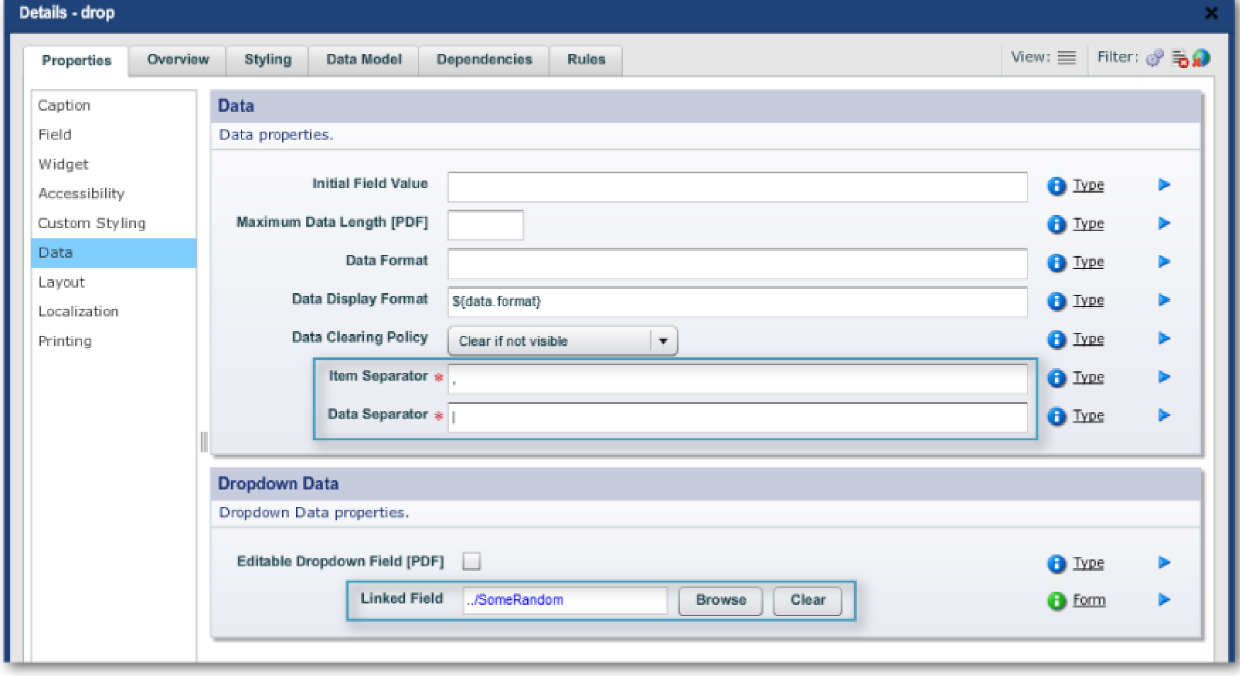
And here is the preview rendition:



Preview of the cascading dropdowns.

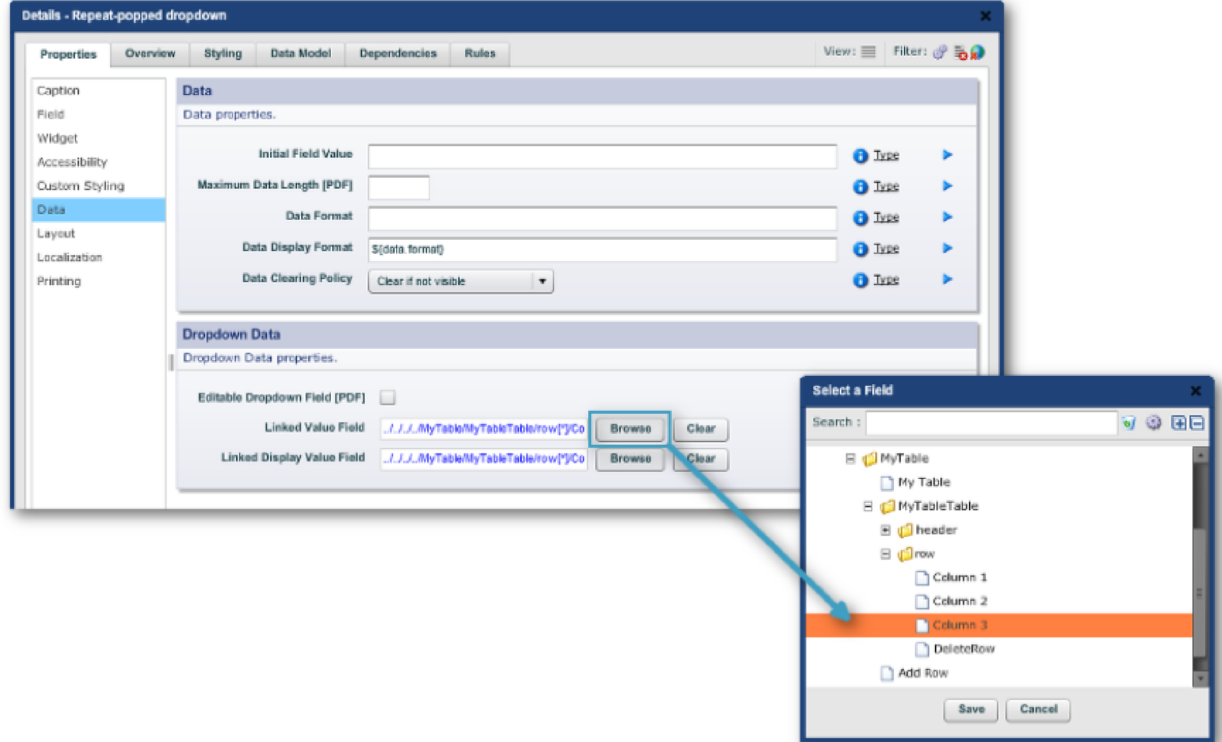
Prefilled Dropdown Lists

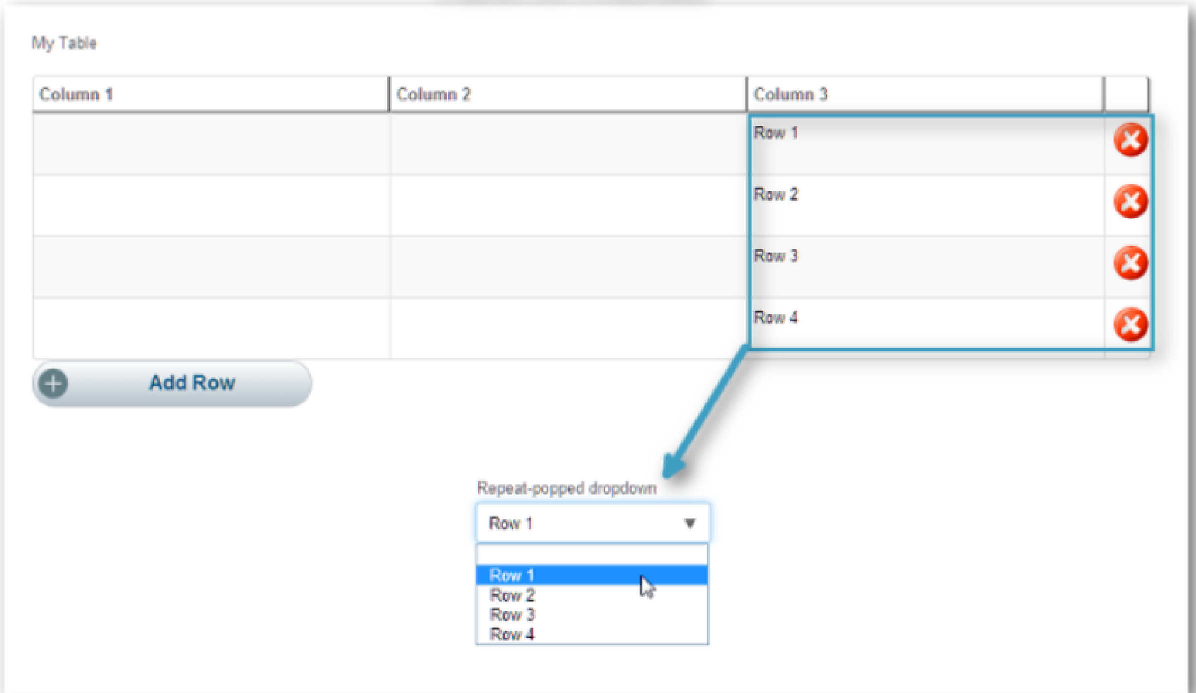
The Populating Dropdown List widget allows a Dropdown Field to be linked (via the Linked Field property) to a data field (created by dropping "Widget Palette -> Internal -> DataField" onto the form's structure. TM is then configured (in TM, "Home Dashboard -> Form -> Form Data Config link -> Property Prefilled Mapping tab -> Edit button") to prepopulate the data field with values. The default is for TM to give the data as a string using the default separators in the Data Field's Edit Properties:



Prefilled Dropdown Lists from Repeatable Areas

The Prepopulating Dropdown List from Repeat widget creates a dropdown menu where the items are taken from a nominated column of a table or a repeat.

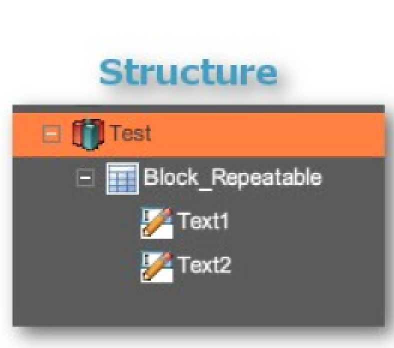


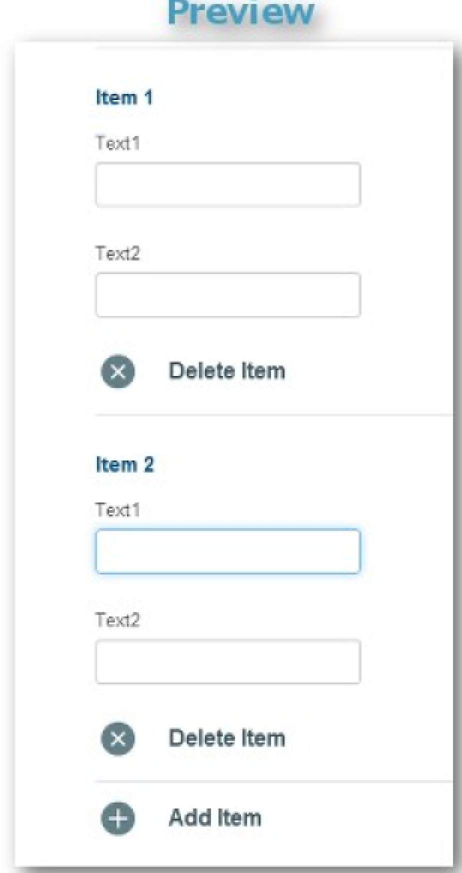


Repeats (Composer v4.3)

Overview

Repeats in earlier versions of Composer used to be complex. You could not copy a a whole repeat structure block to another location on the form without generating error messages and time was you had to use many workarounds to get the new structure work. All these had to be documented for many pages. No longer, with Composer Version 4.
Repeats, then, are a more flexible repeating structure than Tables. Each "Row" in a repeat can be a layout block of fields. That means the row is not confined to a rigid grid-like structure, as with tables.
Otherwise, repeats are easy to construct. The simplest way is to use the predefined block "Repeating Block Template". Into it, insert a "Block [Repeatable]" and populate it with the fields you want repeated. The template gives you Add and Delete buttons for users to dynamically add or delete blocks. You can also use Edit Properties on the block template to change layout and the wording of the buttons.





We encourage you to experiment with the widgets in Repeats and Tables and discover how powerful is this approach.

Advanced Scripting (Composer v4.3)

JavaScript in Composer

Composer uses JavaScript as its scripting language. The only differences are some Composer-related libraries which contain Composer classes to, for example, reference fields in a Composer form or for other Composer integrations.
This section of the Guide focuses on form input scripting. Elsewhere we will cover the XML code used to tweak the format the form's output for some special purpose (such as to accommodate some integration point with the Enterprise's backend).

Syntax

The syntax for Composer's scripts is that of JavaScript. JavaScript is maintained by the Mozilla Foundation. Here are the canonical references:
{+}https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide+ {+}https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference+
Other authoritative references are:
{+}http://www.w3schools.com/js/js_syntax.asp+ {+}http://www.w3schools.com/jsref/default.asp+ {+}http://www.tutorialspoint.com/javascript/javascript_syntax.htm+ {+}http://javascript.info+ {+}http://www.java.com/en/download/faq/java_javascript.xml+

"Me" Instead of "This"

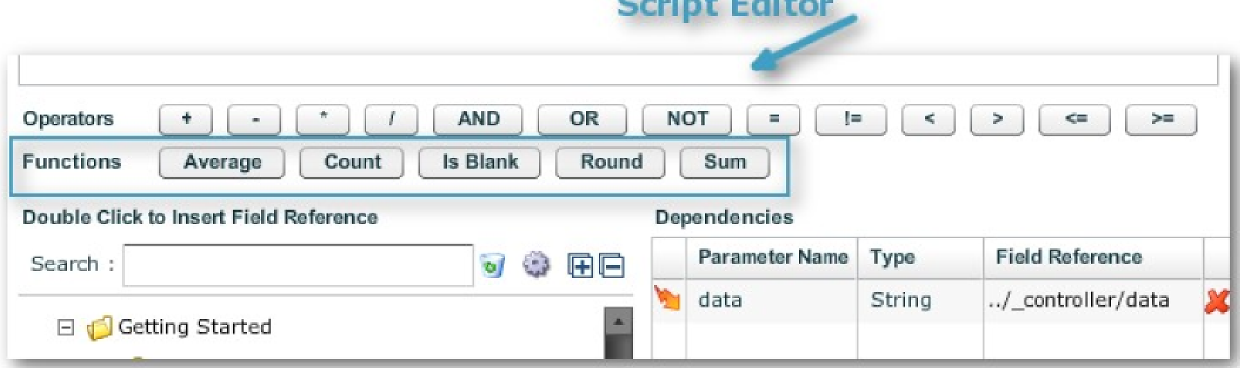
In JavaScript, the "this" object generally refers to the current object receiving the event. Never use the "this" object in Composer scripts.
All Composer JavaScript (for rules, click action scripts, etc.) are embedded in a function, rather than being coded directly under an event. (The reasons for this are beyond this document's scope.) The commonly used "this" object, which refers to the current object, is not available in functions. Instead, Composer provides a "me" object. The me object refers to the object where the business rule is located.
If you have some existing JavaScript code that works correctly in either PDF or HTML, you can generally simply replace every occurrence of "this" with "me" for the same code to work in Composer.

Other Departures from Standard JavaScript

If you want to reuse a legacy script in Composer and want it to work correctly across platforms, you must modify it to use the technology-independent Composer Framework JavaScript library instead of the native libraries. For example, use sfc.getRawValue(me) rather than me.rawValue. See the Avoka Composer Framework link on Composer's Welcome page .
Here, in the context of Transact Composer, "across platforms" specificity means that the sfc calls work for both PDF and HTML.

Classes and Functions

Many of the Composer functions, which are defined in sfcFunctions.js library, have been designed for your convenience — to accomplish a lot with little code. Many of these are embodied in the function buttons in the ScriptEditor .



The function buttons insert the corresponding function code into the Script tab's text window. That code ("$SUM(...)" for example) gets expanded further at runtime (to "sfcFunction.sum(...)" for example
We will also use the terminology "sfc classes" interchangeably with "Composer functions". We will mention here a few of these sfc classes created for you convenience:
sfc.copyValues (srcBlockNodeRef, targetBlockNodeRef);
though note that this does not work recursively, i.e. to blocks within blocks.
Composer also uses conveniently has a number of scripting conventions, such as parameters, which reduce the amounts of code needed to refer to form elements and their values. For more, see the Composer Framework .

Syntax Checking

There are a number of JavaScript tools on the web for checking syntax:
{+}http://www.javascriptlint.com/online_lint.php+
A simple webpage, where you paste your JavaScript fragment into a textarea field and you get back a report of any simple syntax errors. Note that this tool is somewhat limited in scope, but this makes it very useful for quick checks.
Most modern browsers have JavaScript tools {+}https://developers.google.com/chrome-developer-tools/docs/javascript-debugging+ documentation for Chrome's inbuilt checker
{+}https://developer.mozilla.org/en-US/docs/Tools/Debugger+ {+}https://developer.mozilla.org/en/docs/Debugging_JavaScript+ These document Firefox's inbuilt Web Console tool
{+}https://developer.apple.com/library/mac/documentation/AppleApplications/Conceptual/+ Safari_Developer_Guide/Debugger/Debugger.html
The documentation for Apple's Safari on OS X

Composer Framework

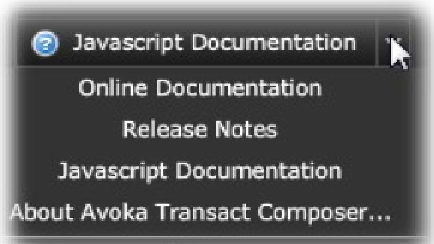
The JavaScript Libraries

Composer has several JavaScript libraries. These are exposed in the Advanced Mode of the Structure Panel ("<Form> -> Composer Framework ->")
formBridge.js (included in PDF forms) htmlFormBridge.js
sfmSupport.js which supports
o sfc.js
This defines the methods in the Composer public API. json.js
for JSON support.

The Public API

Documentation

You can access the API documentation at anytime in Composer, through the Online Documentation dropdown at the top right-hand corner of the Composer User Interface.



Select the "Javascript Documentation" option.
This opens a webpage in a new tab in your browser, listing the methods in the public API. (Avoka does have a set of internal APIs, but these are subject to frequent change. We strongly recommend not using these.)

Calls

The sfc calls are designed to work both for HTML and PDF. This is the reason sfc framework originally came into being.
Most of the sfc methods are static: you just call the method, specify the parameters and the method returns some value. Some typical examples:
sfc.getDaysDifference(startDate, endDate) sfc.convertToString(value) sfc.formatDate(dateObject, dateFormat) sfc.getRawValue(noderef)
Others are dynamic, such as the sfcInstanceManager which operates on a delegate (i.e. an event that is being listened to). You also can make use of Composer's Business Rules widgets to listen to events.
However, Composer provides "parameters" which are convenient constructs to refer to form objects with a minimum of coding.

Using JavaScript's Built-in Functions

You can use the standard JavaScript functions, though you should make preferentially use of any sfc method instead of the standard JavaScript method as a general rule. But there are far more JavaScript functions for you to draw on.
There are many references for these:
https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects http://www.w3schools.com/jsref/jsref_obj_global.asp http://www.tutorialspoint.com/javascript/javascript_builtin_functions.htm
For example:
var n = 12345;
var x = n.toExponential(); // x will be 1.2345e+4

Parameters

In the Script Editor

The above examples of sfc static methods calls have a number of parameters, such as "startDate", "endDate" and so forth. Composer' Script Editor can conveniently refer to fields on the form for these calls.

Refereing to form elements becomes a simple matter of point-and-shoot to elements in the Structure Panel on the bottom-left of the Script Editor. The field is referenced in the JavaScript in the scripting text field by its field name in curly brackets. The Dependencies panel in the dialog gives the Parameter Name (the name within the curly brackets), the data type (about which, more later) and the Field Reference (which is the field's position in the structure, relative to the field for which the script is being created).
So, we could use an sfc call on such a parameter:
sfc.convertToString({Amount1})
You should not merely type the parameter name into some curly brackets in the script: the editor works best if you insert your cursor into the point of the JavaScript and double-click on the parameter listed in "Dependencies".
A useful byproduct of using these Script Editor parameters is that Composer is able to keep track of form elements when they are moved within the form, so that you do not have to re-point to these elements in their new locations.

Edit Parameter Data

Clicking on the Edit icon in the right-hand column of the Dependencies panel (see immediately above) brings up the Edit Parameter Data dialog where you can:
edit the parameter name (i.e. the name that goes into the curly brackets Choose the parameter type via a dropdown:
o string

- Boolean

o number

- Is Selected

o node
o raw value o UID Trigger Type o Static

- Dynamic

We will discuss the Trigger Type further in Optimizing Performance .

Parameter Types

Para meter Type	Explanation
RawV alue	Always returns a String representing the "raw" value of an object. Does no conversion, so it returns dates, booleans and other values as the underlying string that the value is stored as. Uninitialized values may return a null.
String	Always returns a string. Nulls will be converted to empty strings. (This gives you one less thing to "worry" about in your calculations - you don't have to test for both null and empty string.)
Numb er	Always returns a number. It will do its best to convert any parameter to a number. If it cannot convert it, it will return zero.
Boole an	Takes anything that resembles a true value, and return the boolean value "true". This includes "1", "T", "true", "Y", "yes", "on", and is case insensitive. Otherwise it returns the boolean value "false". Blank or Null is false.
IsSele cted	Only used for checkboxes and radio buttons. Returns Boolean true if the checkbox/radio-button is selected.
Node	Returns the internal node object that represents an object. If you point at anything in a repeat from outside the repeat, this returns null (because it's really multiple nodes). If you refer to any object inside a repeat from within the same repeat, it returns the node. Nodes can be passed to internal SFC functions to perform operations on the object itself.
UID	Provides a reference to a node. Similar to a node, but can be used to reference an object in a particular row within a repeat.

Node Reference

There are three ways to refer to a node (or "form element", the terminology we have been using consistently so far):
The Script Editor parameter
the relative path (as seen in the "Field Reference" column value in the Dependencies panel) the Full Path (or "Absolute Path" from the form's root, usually by default "$record")
The field's UID (Unique Identifier)

Obtaining the Fullpath

If you do not avail yourself of the convenience of this point-and-shoot construct, you will have to refer to the form
field using the Data Model tab by double-clicking on the field in either the left- or right-hand panels and copying the "Fullpath" from the Data Model Binding panel in the properties dialog.
For example
$record.GettingStarted.FirstName.data

Obtaining the UID

You could get this from "Edit Properties -> Overview tab -> Basic Information panel -> field UID"
For example:
/GettingStarted/_outerArea/_contentArea/FirstName/data
Or you can use the sfc call: sfc.convertToUid(noderef)

Data Types

Overview

Data Typing is used throughout Information Technology. In Composer, there are two sets of data types:
Composer's own typing, as embodied in parameter data types

- which are used by Composer internally. Some of these are exclusive to Composer:
  - - Node

- UID
- Is Selected JavaScript's typing
- not to be confused with the Composer types, though some do resemble Composer's

The Composer types, as you would expect, were designed to handle data for both HTML and PDF forms.
JavaScript is looser in its data typing, which means that you usually do not explicitly specify some data's type. Circumstances, however, may require you to type data in order to get desired results.

JavaScript Types

JavaScript has less types than Composer. For example, there is no such thing as a Date type. All JavaScript has are:

String

which is a sequence of characters, usually delimmited by straight quote marks for example: "To be or not to be"

Number

and there is no distinction made between inegers or decimals for example: 2, 3.142, 123e5 (scientific notation for 12300000)

Boolean

that is: true or false **JavaScript arrays JavaScript Objects**
for example: JSON objects

Undefined

A variable with no value

Null

An empty variable
Note: undefined is not the same type as null. This is a subtle distinction, but it does have relevance to Composer scripting, as in the a scripting example given in Visibility in Collaboration .

Raw Data

This is a Composer internal data type. All data is held internally in Composer as a string of characters, be it a number, date, string and so on. This can be tricky: for example there is a difference between the integer 2, the floating point value 2 and the string "2". JavaScript handles the calculation of adding an integer to a floating point gracefully, but can mishandle adding an integer to a string.
Dates are a potential source of problems. Please refer to the Dates topic.

Conversions

Because of JavaScript's loose typing, the need to convert form values hardly ever arises for Script Editor parameters. Composer looks after this for you. However, if you are writing your own field references manually, you may have to convert the field's value from its native raw data value. The Public API has a number of methods to do this:
sfc.convertToBoolean(value) sfc.convertToFloat(value) sfc.convertToInteger(value) sfc.convertToString(value)
Converts a number value (or Boolean?) into a Composer string data type. sfc.formatDate(dateObject, dateFormat)
see Dates sfc.getFormattedValue(noderef)
returns the value of the node according to its Composer data type sfc.getRawValue(noderef)
the opposite of the formatted value: retaurns the raw value, irrespective of the Composer data type.

Explicit Conversions

You can convert String values using the public API's functions (listed above) . For example:
var myvalue = sfc.convertToNumber(sfc.getRawValue(me));
This code will convert the raw value of the current field ("me") to a Number.

Implicit Conversion

JavaScript is a loosely typed language, and will attempt to convert data for you. The rules are a little complicated, which is why we suggest that you always try to convert to the appropriate type before you use them.
One simple example is a way to convert any data type into a string - simply concatenate an empty string to it. For example:
var var1 = 18; // var1 is a number
var var2 = var1+""; var2 is a string, because we concatenated "" to it.

Comparisons

Be careful when making comparisons between different data types. JavaScript will attempt to convert them for you, but the results may not always be the expected.
In JavaScript, the "this" object generally refers to the current object receiving the event. Never use the "this" object in Composer scripts.
All Composer JavaScripts are embedded in a function (see Advanced Scripting), rather than being coded directly under an event. (The reasons for this are out-of-scope.) The commonly used "this" object to refer to the current object, is not available in functions. Instead, Composer provides a "me" object. The "me" object refers to the object where the Composer script is located, i.e. the form element whose Edit Properties dialog holds
the script.

If you have some existing JavaScript code that works correctly in either PDF or HTML, you can generally simply replace every occurrence of "this" with "me" for the same code to work in Composer.
Note: If you want the script to work correctly across different technologies, you must also rewrite it to use Composer's technology independent JavaScript library rather than the native libraries. for example, use sfc.getRawValue(me) rather than me.rawValue. See the Avoka Composer Framework.

Dates

Date Formats and Patterns

There are a number of different ways in which data is stored, displayed, edited and validated in Composer forms. These are mostly useful for dates and numeric values, which can be represented in different ways. When you are using Composer, you can specify these different formats by using data patterns, such as DD/ MM/YYYY or $#,###.##.
The different formats are:

Format	Description
Edit Format	This is the format that is used when the user types into the field. For example, 3/27/2012, or 49.99. It is possible to have multiple alternate edit formats - for example, allowing either 2 or 4 digit years.
Display Format	This is the format that is displayed when the user exits the field. It can be more readable than the edit format, and clarifies the interpretation. For example, 27th March 2012 or $49.99
Raw Format	This is the value that is actually stored within the form at runtime. It's the value you get from calling the getRawValue(). This is standardized to make programming easier • it doesn't matter what your formats are used for the form users, you always get a consistent value in any code you write. For dates, this is ISO format, such as 2012- 03-27.
XML Format	This is the format that is used to actually store the data in the XML file. Composer will use this format for pre-population data, and will convert to this format on submission.
Validation Pattern	PDF forms allow a separate validation pattern that is used to ensure that the data is valid. In Composer, the Validation Pattern is always identical to the Edit Pattern.

Usually, you don't need to be aware of these patterns because they are specified in your organizational stylesheets. However, you may need to occasionally override these patterns for particular fields.
In most cases, users will enter dates using pop-up date editors, and so date edit format patterns are not so important. However, some users, especially "power" users and visually impaired users may prefer to enter dates using the keyboard.

Recommendation for Dates

For dates, we recommend that the most useful Data Format is:
North America: date{M/D/YYYY}|date{M/D/YY}
Elsewhere: date{D/M/YYYY}|date{D/M/YY}
These format will:
accept single digit days and months with or without a leading zero (flexible)
accept 2 digit or 4 digit years (flexible)
for editing, will always display days and years without leading zeros (simpler)
for editing, will always display years including the century (no ambiguity) Working with Dates
Composer uses the Java library moment.js (freely distributable under the terms of the MIT license) to handle dates.
The date clases in the Avoka Composer Framework are:
sfc.formatDate(dateObject, dateFormat)
where "dateObject" is is the raw value of the form element, sfc.getRawValue(someFormElement)
sfc.getDaysDifference(startDate, endDate) Where the startDate and endDate are raw values.
sfc.getTodaysDate()
Returns today's date as YYYY-MM-DD
This is not the generation date of the form (which you access in scripts with the property formula
$GENERATION{DATE} instead)
This call returns only the calendar date. To get the time of day and timezone as well, use this native JavaScript call instead, where "x" will be your date object.
var x = new Date();
sfc.isLeapYear(year)
Where "year" is a 4-digit value. Returns true or false.
sfc.isPastDate(date)
Where "date" is the raw value. Returns true or false
sfc.isToday(date)
Where "date" is the raw value.
sfc.parseDate(dateString, dateFormat)
Parses a date according to the given date format. Both parameters are strings. Returns a raw value.

Advanced Date Output Formats

The sfc methods for date handling depend on the Moment.js library. This JavaScript library can output dates in various formats, as specified by this table:

Input	Output
M, MM	Month Number (1 - 12)
MMM, MMMM	Month Name
Q	Quarter (1 - 4) – sets the month to the first month in that quarter

D, DD	Day of month
Do	Ordinal day of month (from 2.6.0)
DDD, DDDD	Day of year
d, dd, ddd, dddd	Day of week (NOTE: these formats only make sense when combined with "ww")
e	Day of week (locale) (NOTE: these formats only make sense when combined with "ww")
E	Day of week (ISO) (NOTE: this format only make sense when combined with "WW")

Input	Output
w, ww	Week of the year (NOTE: combine this format with "gg" or "gggg" instead of "YY" or "YYYY")
W, WW	Week of the year (NOTE: combine this format with "GG" or "GGGG" instead of "YY" or "YYYY")
YY	2 digit year (see below)
YYYY	4 digit year
gg	2 digit week year (if greater than 68 will return 1900's, otherwise 2000's)
gggg	4 digit week year
GG	2 digit week year (ISO) (if greater than 68 will return 1900's, otherwise 2000's)
GGGG	4 digit week year (ISO)
a, A	AM/PM
H, HH	24 hour time
h, hh	12 hour time (use in conjunction with a or A)
m, mm	Minutes
s, ss	Seconds
S	Deciseconds (1/10th of a second)
SS	Centiseconds (1/100th of a second)
SSS	Milliseconds (1/1000th of a second)
Z, ZZ	Timezone offset as +07:00 or +0700
X	Unix timestamp
LT, L, LL, LLL, LLLL	Locale dependent date and time representation

You parse an input date with sfc.parseDate (dateString, dateFormat), using the formats in the Input column of the table immediately below. The output of this call gives you a properly formatted raw date format.
You can use these to build the full input format specification, using other characters as punctuation. For example: YYYY-MM-DD or YYYY/MM/DD or DD/MM/YYYY or MM/DD/YYYY. Internally, Composer's raw date format is YYYY-MM-DD, or for more precision to include time of day, timezones along with the date, it uses the ISO Date Format.
You also can control how a date object displays using sfc.formatDate(dateObject, dateFormat), using the formats displayed in the right-hand "Output" column. As with inputs, you can concatenate these together using characters as punctuation. For example: "MMMM Do, YYYY, HH:mm a, timezone Z" formats as "May 19th, 2014, 13:27 pm, timezone +10:00", where the Z specifies the timezone as the timezone offset from GMT.
To return the precise time and timezone, use
var x= new Date;
and get the formatted time with our sample format with the call
sfc.formatDate(x, "MMMM Do, YYYY, HH:mm a, timezone Z")

Regular Expressions

Overview

A regular expression is a sophisticated, industry-standard way of specifying a validation pattern. An explanation of regular expressions is beyond the scope of this document, but there are numerous books and web sites that you can refer to. You can also use search engines to find standard regular expressions for common types of data. An example of some regular expression patterns are:
^[a-zA-Z0-9_-\.]@[a-zA-Z0-9_-\.]\.[a-zA-Z]{2,4}$ the pattern for email addresses
^http\://[a-zA-Z0-9-\.]+\.[a-zA-Z]{2,3}(/\S*)?$
the pattern for HTTP:// URLs

RegEx References

https://developer.mozilla.org/en/docs/Web/JavaScript/Guide/Regular_Expressions https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/RegExp http://www.regular-expressions.info/javascript.html
http://regex101.com http://regexpal.com
A good resource for regular expressions is: http://regexlib.com/.

Online regular expression editor and evaluator can be found at: http://gskinner.com/RegExr/ and http:// www.debuggex.com/.

How to Use Regular Expressions

There are two ways to use patterns in JavaScript:
var patt = new RegExp(pattern,modifiers); or more simply:
var patt = /pattern/modifiers;
The modifiers (also called "flags" in the documentation) are:

g

global match

i

ignore case

m

multiline; treat beginning and end characters (^ and $) as working over multiple lines (i.e., match the beginning or end of each line (delimited by \n or \r), not only the very beginning or end of the whole input string)
y
sticky; matches only from the index indicated by the lastIndex property of this regular expression in the target string (and does not attempt to match from any later indexes). This allows the match-only-at-start capabilities of the character "^" to effectively be used at any location in a string by changing the value of the lastIndex property.
In other words, in most cases where you are not using a modifier, just use the pattern as follows:
/^[a-zA-Z0-9_-\.]@[a-zA-Z0-9_-\.]\.[a-zA-Z]{2,4}$/
in your JavaScript code.

Using Property Formulae

Overview

The following scripting constructs, the property formulae, only apply while the form is being generated. When generation is complete, the fom is then delivered to the user, either as HTML the browser or as XFA to Acrobat reader (or some other reader) or to a pdf browser internal module or plugin.
These property formulae allow you to query the build environment, to inspect a form field, to find a field in the form hierarchy, perform some string manipulations, and other useful functions.
They do not trigger anything while the form is running and only affect the time it takes to serve a form.

$BUILD

Call	Result
$BUILD(ENVIRONMENT)	Returns the values of the called arguments: ENVIRONMENT VERSION REVISION
$BUILD(VERSION)	
$BUILD(REVISION)	

$CHAR

Call	Result
$CHAR{COMMA}	Returns a comma character ','
$CHAR{DOT}	Returns a dot character '.'
$CHAR{SLASH}	Returns a slash character '/'

$CHAR is a convenience to specify characters, in a text field for example, that would normally are parsed in JavaScript.

$DATA

Call	Result
$DATA(ROOT)	Returns the root element of the form: E.g. "/AvokaSmartForm"

$ENCODE

Call	Result
$ENCODE{SCRIPT,value}	Encodes the value so it can placed in a javascript string literal
$ENCODE{RICHTEXT,value}	Replaces richtext markup with its form specific implementation
$ENCODE{URL,value}	URL encodes the supplied value
$ENCODE{XML,value}	Encodes the value with XML markup entities
$ENCODE{STRIP,value}	Removes richtext markup
$ENCODE{TRIM,value}	Removes cr, lf characters, leading and trailing spaces
$ENCODE{UPPER,value}	Converts the value to uppercase
$ENCODE{LOWER,value}	Converts the value to lowercase
$ENCODE{FNNAME,value}	Converts the value to a valid javascript function name by replacing invalid chars with _

Note: $ENCODE is unique among the property formulae, in that you can specify multiple values, pipe- delimited. For example: "$ENCODE{TRIM|UPPER, value}", will first trim the string (i.e. remove any leading or trailing space characters) offered as the "value" and then convert the characters to upper case. The order of these pipe-separated arguments is important, as changing the order of them my affect the output. Note that these are executed from left to right.

$FIELD

Call	Result
$FIELD{TYPE}	returns the type of the current field

Call	Result
	if any
$FIELD{TYPE,relativeUid}	returns the type of the field at the relative uid
$FIELD{TYPE,relativeUid,type Name}	returns 'true' if the type of the field at the relative uid is an instance of the supplied type name
$FIELD{VALUE}	returns the value of the current property of the current field
$FIELD{VALUE,typeName}	returns the value of the current property as defined in the specified type
$FIELD{RBTYPE}	For radio buttons only, indicates whether the buttons value is controlled by the group or by the button returns 'group' for group controlled buttons (case 1&2) returns 'button' for button controlled buttons (case 3)
$FIELD{RBTYPE,buttonType}	For radio buttons only returns true if the button type (as deternmined by $FIELD{RBTYPE}) matches the supplied type.
$FIELD{INSTANCEMANAGE R}	returns the relative uid of the closest parent instance manager or blank if not found
$FIELD{NESTLEVEL}	returns the number of repeat instances this field is nested inside of
$FIELD{INDEX}	returns the index of the field as a child of its parent
$FIELD{REVERSEINDEX}	returns the reverse index of the field wrt its parent
$FIELD{ISFIRST}	returns true if this field is the first child of its parent
$FIELD{ISLAST}	returns true if this field is the last child of its parent
$FIELD{CHILDCOUNT}	returns the number of child fields
$FIELD{HASCHILDREN}	returns true if the field has at least 1 child
$FIELD{RESOURCE,resName}	returns the contents of the given resource applying formula substitutions with the context of the current field.

$FIND

Call	Result
$FIND{propertyname}	Searches from the current context up the hierachy for a property with the given name

Call	Result
	and returns the value of the first one found. If a property isn't found then a blank string is returned. E.g. $FIND{section.name} returns the section name from within any child field of the section
$FIND{path, propertyname}	Sames as $FIND{propertyname} but the upwards search begins from the field found at the relative path provided. If no field can be found at the supplied path then a blank string is returned. e.g. $FIND{../../someField,label}

$FLOAT

TBA

$FORM

Call	Result
$FORM(NAME)	Returns the form's values of the called argument
$FORM(DESCRIPTION)	
$FORM(RELEASE)	
$FORM(TEMPLATE)	

$GENERATION

Call	Result
$GENERATION(TECHNOLOGY)	
$GENERATION(FLAVOUR)	
$GENERATION(FLAVOR)	
$GENERATION(TARGET)	
$GENERATION(VARIANT)	
$GENERATION(UUID)	
$GENERATION(TIMESTAMP)	
$GENERATION(DATE)	
$GENERATION(DEBUG)	
$GENERATION(DEBUGSCRIPT)	

$IF

TBA

$INFO

Call	Result
$INFO(USER)	Returns the form's info values of the called arguments
$INFO(ACCOUNT)	
$INFO(CLIENT)	
$INFO(ORGANISATION)	
$INFO(ORGANIZATION)	

Call	Result

$INFO(PROJECT)	
$INFO(SERVER)	
$INFO(ENVIRONMENT)	

$LIST

Call	Returns
$LIST{CONTAINS,list,value,sepa rator}	Returns true if the list contains the value
$LIST{INDEXOF,list,value,separ ator}	Returns the 0 based index of the value within the list
$LIST{ITEMAT,list,index,separat or}	Returns the item at 0 based index position within the list
$LIST{SIZE,list,separator}	Returns the size of the list

$LOCALE

Call	Result
$LOCALE(LANGUAGE)	Returns the form's locale values of the called arguments
$INFO(COUNTRY)	
$INFO(CODE)	

$LOOKUP

TBA

$MATH

Call	Result
$MATH{PX,value}	Converts the value to pixels and strips the units
$MATH{IN,value}	Converts the value to inches and strips the units
$MATH{MM,value}	Converts the value to millimetres and strips the units
$MATH{MAX,value,...}	Returns the largest value - if the first value has units then the result is returned in those units
$MATH{MIN,value,...}	Returns the smallest value - if the first value has units then the result is returned in those units
$MATH{SUM,value,...}	Returns the sum of the supplied values - if the first value has units then the result is returned in those units
$MATH{PRODUCT,value,...}	Returns the product of the supplied values (multiplies them together) - if the first value has units then the result is returned in those units
$MATH{HALF,value}	Returns half of the supplied value - if the value has units then the result is returned in those units
$MATH{TWICE,value}	Returns twice of the supplied value - if the value has units then

Call	Result
	the result is returned in those units
$MATH{MARGIN,value}	Converts a composer margin to a technology supported margin

$RULE

Call	Result
$RULE{TRIGGER,rulename)	Returns the trigger width in pixels for the given rule

$SEQUENCE

Call	Result
$SEQUENCE{Sequence argument, Sequence argument1, ..., RESET)	Resets the named sequences.

$STRING

Call	Result
$STRING{CONTAINS,value,patt ern}	Returns true if the value contains the pattern as a substring
$STRING{INDEXOF,value,patter n}	Returns the 0 based index of the pattern within the value or -1 if not found.

$TEST

Call	Result
$TEST{BLANK,value}	Returns true if the supplied value is blank
$TEST{NOTBLANK,value}	Returns true if the supplied value is not blank
$TEST{EQUALS,value1,value2}	Returns true if value1 and value2 are the same
$TEST{NOTEQUALS,value1,valu e2}	Returns true if the value1 and value2 are not the same
$TEST{NOT,value}	Returns true if value is the literal false otherwise returns false
$TEST{AND,value1,value2}	Returns true if value1 and value2 are both the literal true
$TEST{OR,value1,value2}	Returns true if value1 or value2 are the literal true
$TEST{LESSTHAN,value1,value 2}	Returns true if value1 as a number is less than value 2 as a number
$TEST{GREATERTHAN,value1,v alue2}	Returns true if value1 as a number is greater than value 2 as a number

Optimizing Performance (Composer v4.3)

Overview

Forms in Composer are smart: they are dynamic and user actions in filling out the forms trigger events and change the form on-the-fly as users enter data. For example, a user can click on a checkbox and a number of fields become visible as a result. Or users fill in numerical values in a table column and a total is displayed in a field at the foot of the table.
Many forms in the real world elicit a great amount of information from the end users, with the consequence that a number of events ripple through the form as users complete entry into individual fields.
Why are there so many events on these smart forms? They are there to:
ensure that, as a default, all the data and all the dependencies on the form operate with complete integrity all of the time
ensure that all validation rules and scripts have occurred
ensure that any changes made at any time during the filling out process to any fields in the form affect all objects on the form
This design approach means that calculations, scripts and dependencies produce predictable and consistent results and that the data presented to end users is accurate and up-to-date. These take place without the form's designer doing any further work. But, as is often the case with a "one size fits all" approach, there are trade offs in large, complex forms:
an increase in generation time for very large forms an increase in the rendering time of the form
increased lag for the user to move from one field to another in such forms
increased lag when some action takes place, such as navigating through the form, submitting, saving it.

Form Generation

A full discussion of this topic is beyond the scope of this guide. The form's HTML is not statically stored in Transaction Manager and just downloaded to the end user. Instead, a number of processes take place:
the form's dependencies are inspected, resulting in a hierarchy of triggering and firing the form is prepopulated with data while being served
server-side processes to track the form from its being served to its outcome (submission, saving, abandonment, anonymous sharing or progression through a collaboration)
and even all the web server processes to deliver all the assets of the form to the user's device

Form Rendering

This is the time it takes for the device to render the form in either the browser or in the Mobile App. The form is served to either as HTML5 code interspersed with JavaScript. The JavaScript code is not merely the scripts you have placed into some of the form's fields through the Script Editor: there is also much code to integrate the form and your scripts together, as well as the functioning of the other form widgets. All these scripts and HTML have to load and initiate on the device. Most mobile devices are slower than desktop devices — although they are getting faster.
Be aware that the whole form downloads to the device, not just the visible or editable parts of the form. The same is true for all the prepolutation data, all of the form's autosuggest data (as distinct from such data provided by an external dynamic data service) and even mobile device responsive elements normally not visible on the desktop. All of these affect the rendering times, but the good news is that the increase in processing speeds of devices, and the huge performance increases in modern browsers, rendering times are no longer the problem they once were.
In the end, you must always take the user experience into account, and this may lead you to simplify your forms as a general rule.

Speeding up Generation and Rendering

It follows that these two, generation and rendering, depend mostly on the complexity of the form itself. A couple of settings in Composer have a minor influence on form generation (as we will see below). The form rendering times for the most part on the amount of HTML and JavaScript code to be processed. Optimization of rendering really means simplification of the form — just as with any other web page.

Optimization Strategies

Composer now has tools to improve the performance of forms while they are in use. Most forms of medium to low complexity require no optimization at all and run fast enough for users to have no issues. This may not be the case with very complex forms with a great deal of dependencies.
There are a number of possible strategies for tuning complex forms to run faster:
Tuning the dependencies
Tuning the triggers that make scripts fire Tuning the events that make scripts fire
Tuning the observers watching form objects and events Use of the Business Rules widgets on the form

Tuning Dependencies

Inspecting Dependencies

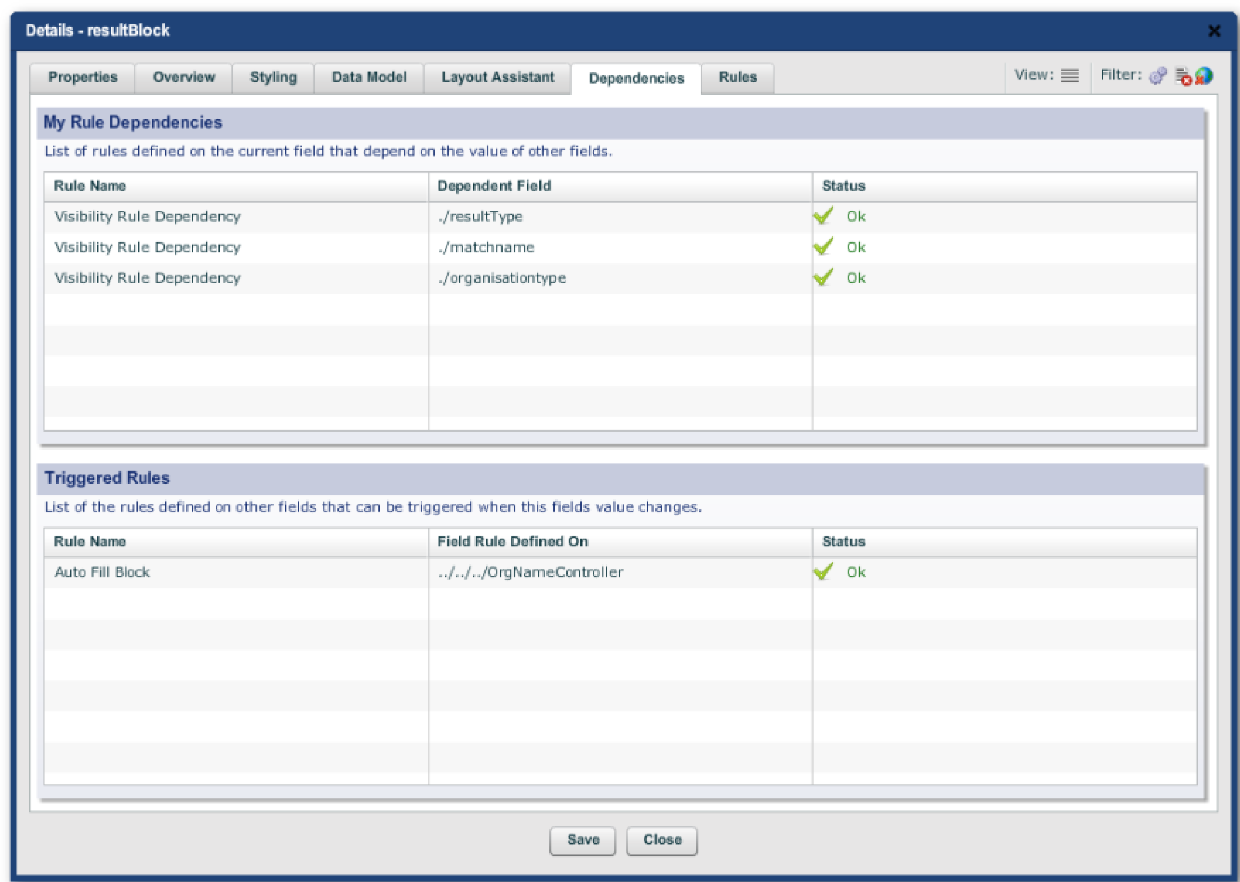
The Edit Properties dialog now allows you to inspect a form element's dependencies through "Edit Properties -
> Dependencies tab". The tab has 2 panels:

My Rule Dependencies

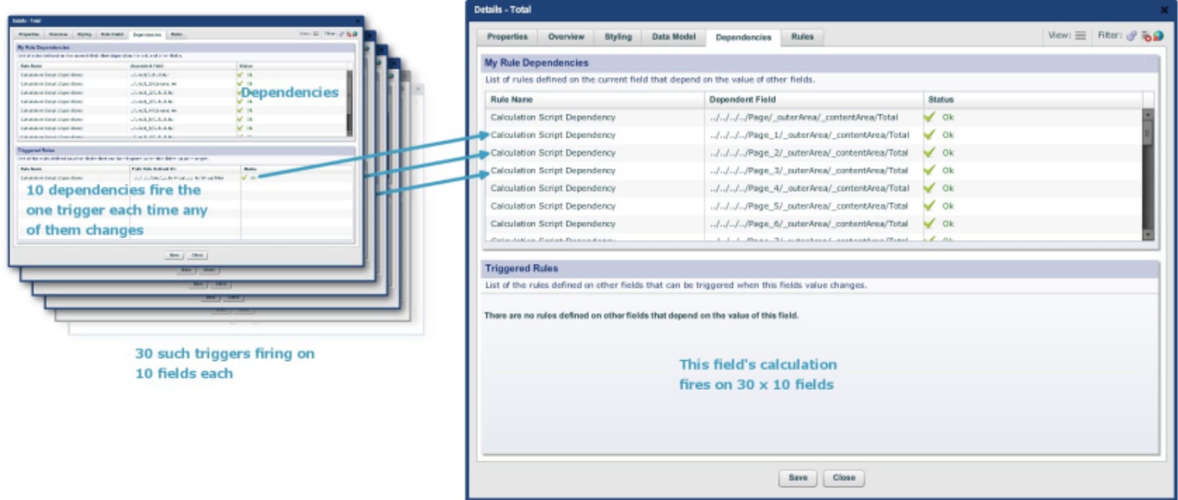
which lists the form elements on which depend this element's the Rules or Scripts. In other words, this panel shows the rules, scripts that fire and affect this element's value upon the change of any of the elements listed.

Triggered Rules

which lists the rules or scripts which fire upon this element's change.

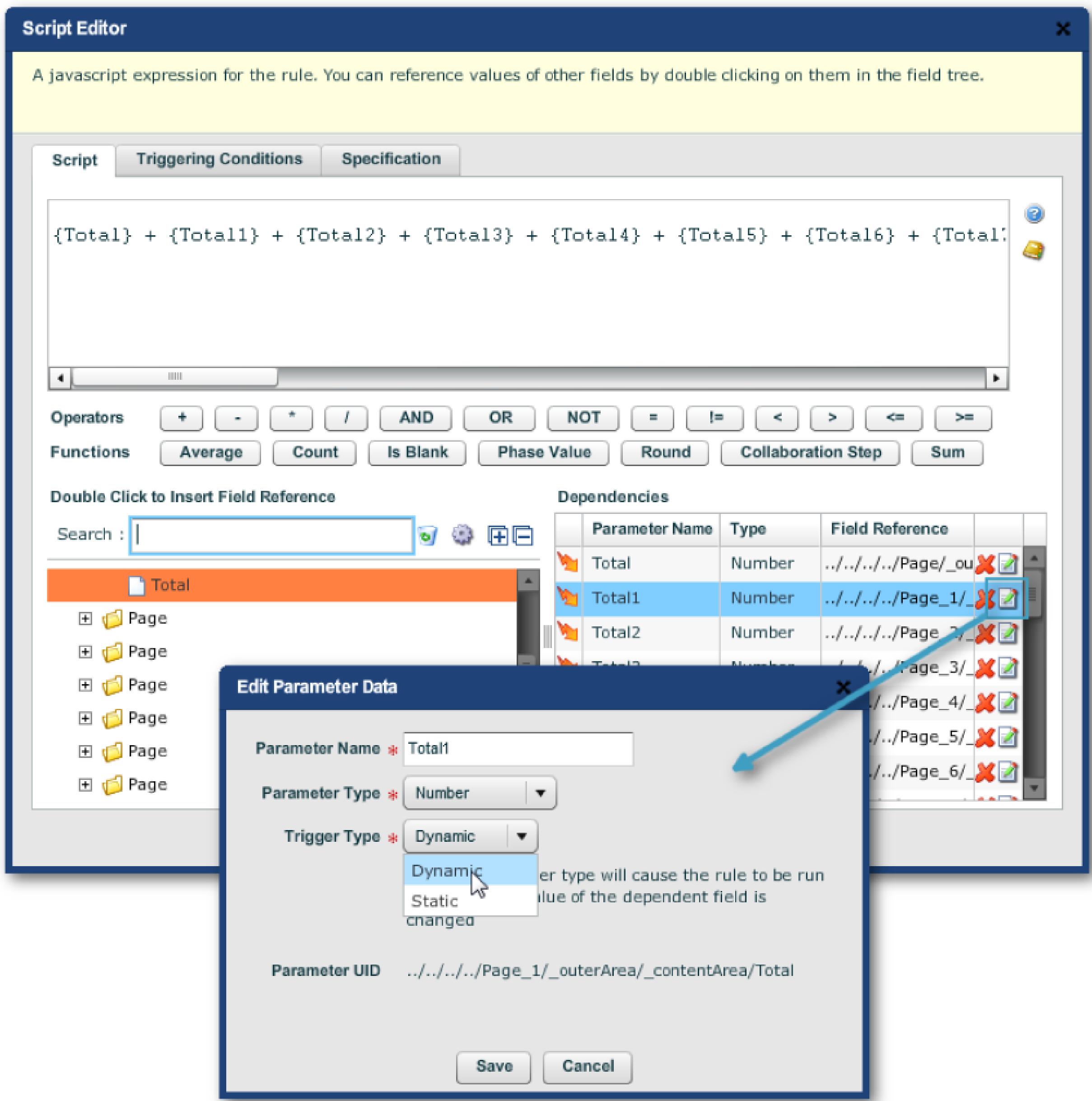


Here is an extreme case, where we have deliberately created so many dependencies on the form that moving from field to field takes more than 10 seconds each time. The form has 30 wizard-style pages. Each page has a static tables of 10 rows, 3 columns each. Each row has a calculation, in the 3rd column, of column 1 multiplied by column 2. At the bottom of each page is a subtotal of all 10 calculation fields. The last page of the form is a total of all 30 subtotals. This means that the total on the last page is triggered by any changes in some 91 times 30 fields, i.e. the field is triggered by changes to any one of 2,730 fields. Let us focus on just the subtotals triggering the total:

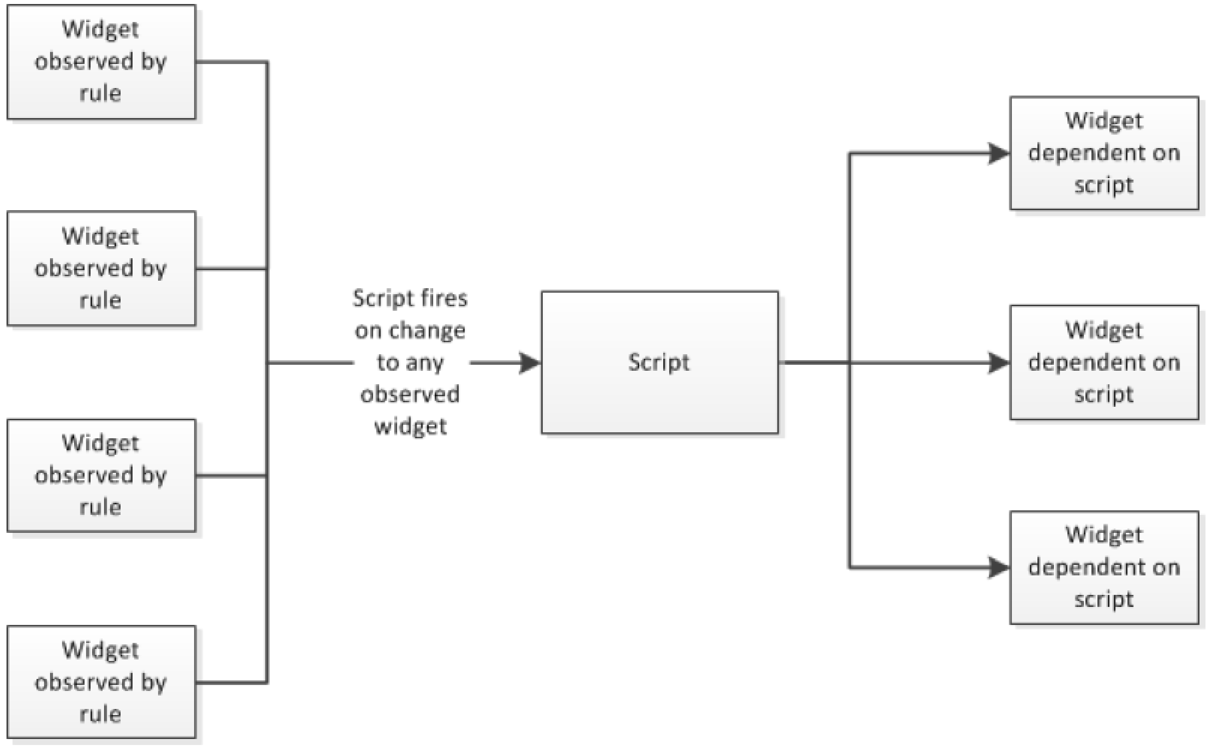


Dynamic vs Static Dependencies

When you write a script in the script editor of the Edit Properties dialog, you reference a form object using a Parameter . This creates a dependency, which duly appears in the My Rule Dependencies panel of the Dependencies tab.
The default setting for these is to have a Dynamic trigger type, as indicated by the presence of the lightning arrow in the left-hand column of the Dependencies table of the Script Editor:

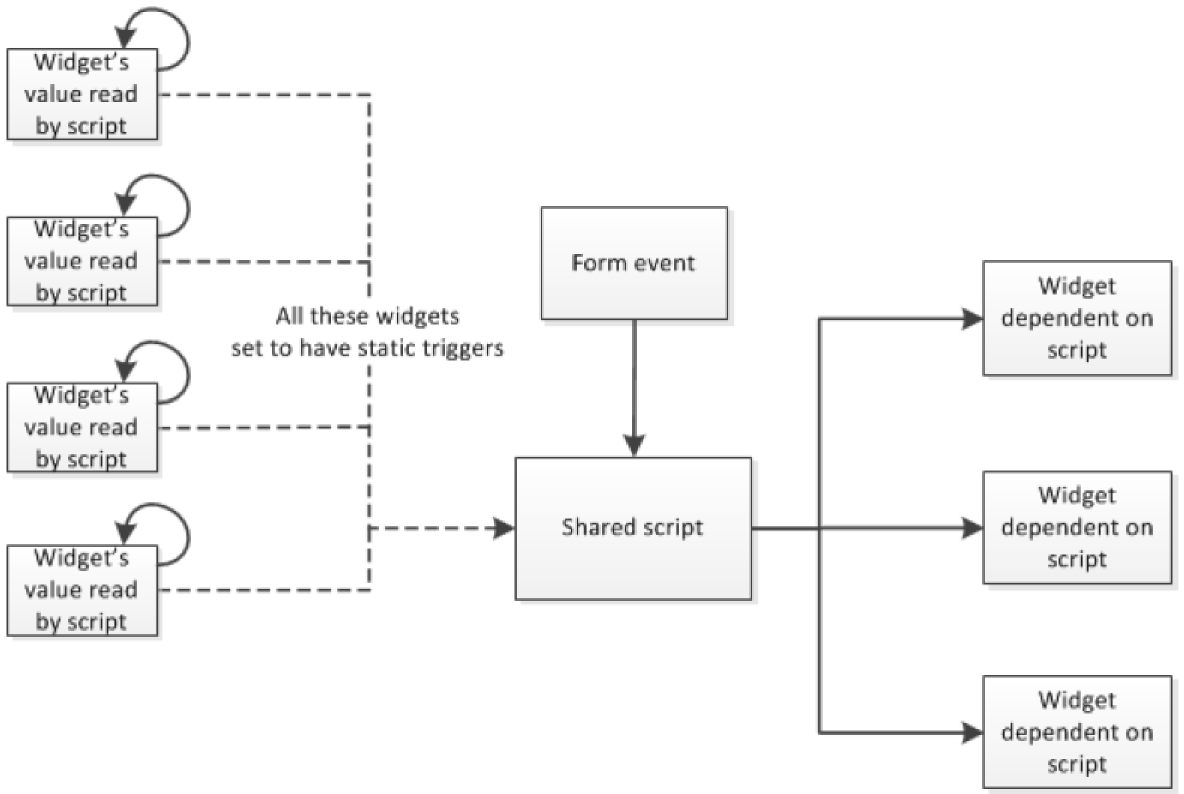


A dynamic trigger type means that the script will fire whenever the field referenced changes. That means the field is always watched and the trigger will always fire on any change to the field.

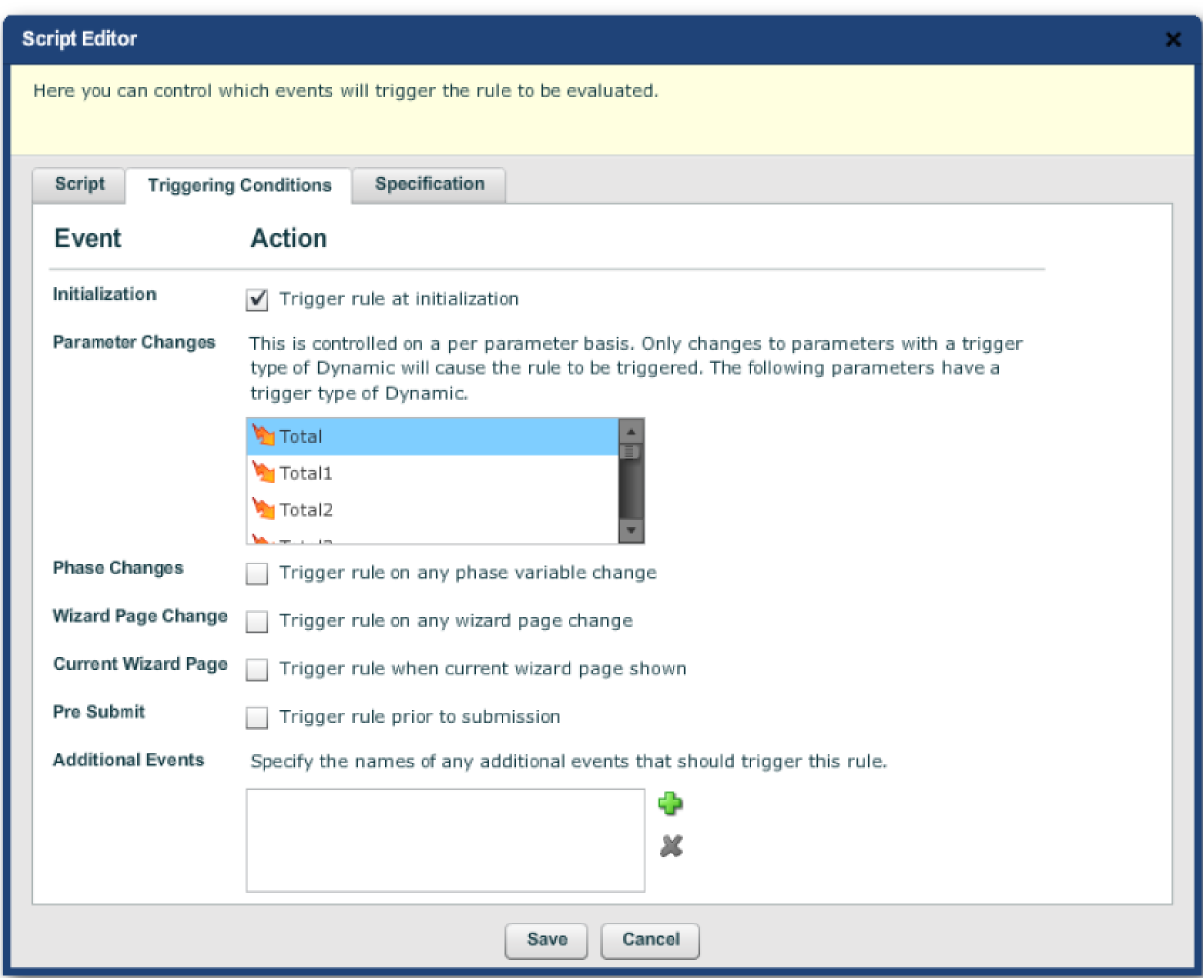


Changing the trigger type to Static means that the widget will no longer be observed. So, if we change all the dependencies in the above example to the Static trigger type, the rule will not fire. That we do want the rule to fire sometimes goes without saying, because we want the total to be the correct value. Which brings us to the Trigger Conditions tab in the Script Editor.

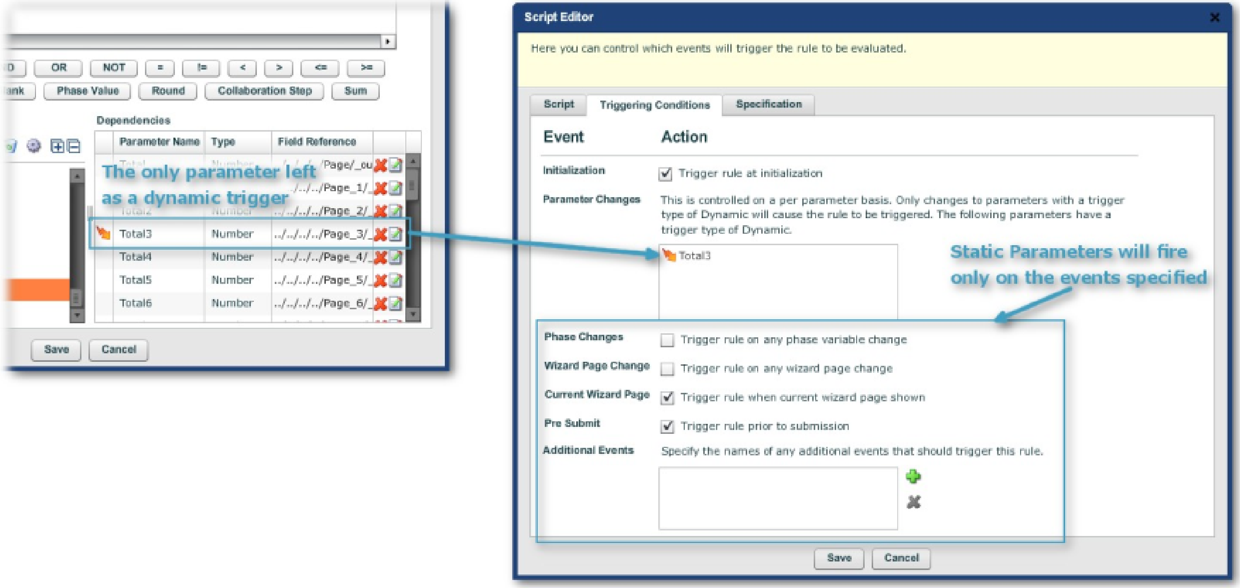
Trigger Conditions



Here, the widgets are only read when particular form events occur. We can specify the events in the Triggering Conditions tab of the Script Editor:



But the problem here is that all the parameters listed in the Parameter Changes list have dynamic triggers. That means the settings below the list will are not reduce the firing events by very much: the rule will fire on the Parameter Changes listed. So, as a demo, we have changed all the parameters but one to static, and set them to fire only when the user moves to the wizard page with the field containing the script or just prior to submission. Of course, in practice, you would most likely, in this case, turn all the trigger types to static to give you complete control over the script firing by eliminating the dynamic triggers.



And, the reverse also applies: you can then set the triggers from the form element with the script to the widgets dependent on the script (the widgets on the right hand side of the diagram with the form event) to also be static through the Script Editor of each of these widgets.
You can see that we can now, by going through the triggering conditions throughout the form, reduce the triggers from the original 2,730 fields to a much smaller number and substantially improve the performance on this form.
You might want to leave a parameter or two still with dynamic triggers: these could be a checkbox or button, whose action is to fire the script. In earlier releases of Composer, there were some work arounds to control the firing of scripts. These included having hidden a checkboxes (to act as a Boolean to make the script fire) or using blocks as a parameter reference (in the absence of static triggering). These are now deprecated.

Trigger at Initialization

You can turn off this option and the form will render in the browser. This would only make rendering perceptively faster if the script is compute intensive. It is not worth doing for the majority of scripts. But note that the setting is not global: it applies only to this script. Also, make sure that you are not creating issues that by not running this script.

Phase Changes

The events listed in the Triggering Conditions tab are self-explanatory. Phase Changes may not be.
A phase is a non-localized variable that holds some value and which can be observed for whatever purpose. It is non-localized because you need not reference it through either a relative or absolute path.
Here follow some brief notes on creating a phase.
Pick up a "Phase" Widget from the palette (use Search to find it) and drop it onto "Nuts & Bolts -> Phases" in the Structure Panel. In the resulting wizard, it is usual to check the "Exclude from Layout Manager" as the widget normally does not appear on the form.
The Phase is configured in the Data tab.

Put any scripts required in the Rules tab, as per usual. In other scripts, you can reference the phase by
pointing to the structure widget as a Parameter or by src.getPhaseValue("phase name") as before.
Make sure you check the Phase Change checkbox in the Triggering Conditions tab of the scripts you want to fire on phase changes.

Additional Events

You can also type in the names of other events. The standard events are listed here in this guide.

Caveats

As you can see, these tools are powerful and require no programming as such.
However, as we have mentioned above , after optimization, you must test the form to see if the scripts yield the correct answers now that changes to field values no longer ripple through the entire form automatically. There are also other undesired effects that you must guard against, such as preventing validation rules from firing and therefore letting invalid data through to submission.

Business Rules (Composer v4.3)

The Rules Properties section of this manual describes how you add scripting and logic to individual widgets. Though some widgets, like blocks, can pass on their rules to their children, this methodology does not allow rules to be assigned to several widgets, especially those not related or which do not share inheritances. What if the widgets are scattered throughout the form and cannot easily be wrapped by some block in the structure panel without including other widgets that you do not want to similarly inherit properties? This problem cannot be solved through "Edit Properties" individually for all the widgets concerned.
The "Business Rule" and "Mandatory" field types solve this problem by being central containers for rules and scripts. Other widgets can now share, for example, a script, by pointing to a business rule widget in the their "Edit Properties" dialog.
There are three types of Business Rule field types:

General Purpose

These contain the logic that control visibility, editability, data clearing, event firing, triggers and event listening.

Error

These generate messages and also prevent submission of the form

Warnings

These also generate messages but do not prevent form submission.

1. Mandatory Checkbox Block

Allows you to enforce the activation of at least one checkbox in the block: leaving all of the boxes unchecked will prevent submission of the form.

1. Mandatory Block

You use this to enforce that **at least one** of the contained form elements are given values.
These rules widgets are intended for advanced users only, though many of their features are no harder to implement than editing some of the properties in the "Rules" tab for each widget. They are a powerful tool for optimizing response times of complex forms

General Purpose

Configuration Settings

The General Purpose Business Rule widget's Rule tab holds the following:

Visibility Rule

Note the difference between **Always Visible**, **Never Visible** and **Script** or **Rule Based**.
That is: the value of an invisible field being observed by the Business Rule could be affecting the outcome of the rule because the field has not been cleared.
See also Clearing Policies .

Visibility Policy

Initial Visibility
Visibility Data Clearing Policy
Business Rule
Either "None" or "Script Based"

Additional Event LIsteners

See here

Business Rule Trigger

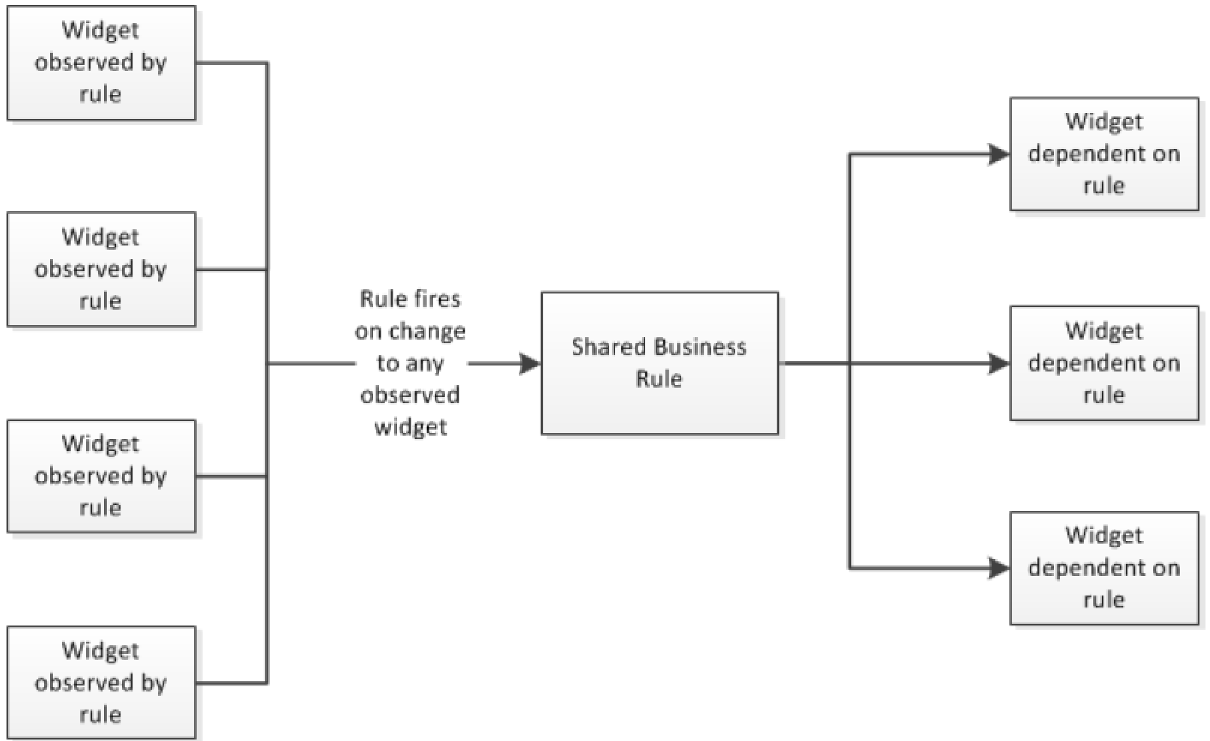
Run Always | Run only when visible.

The Relationship Between Nominated Widgets and the Rule

There are two broad ways for a General Purpose Business Rule widget to fire:

The default

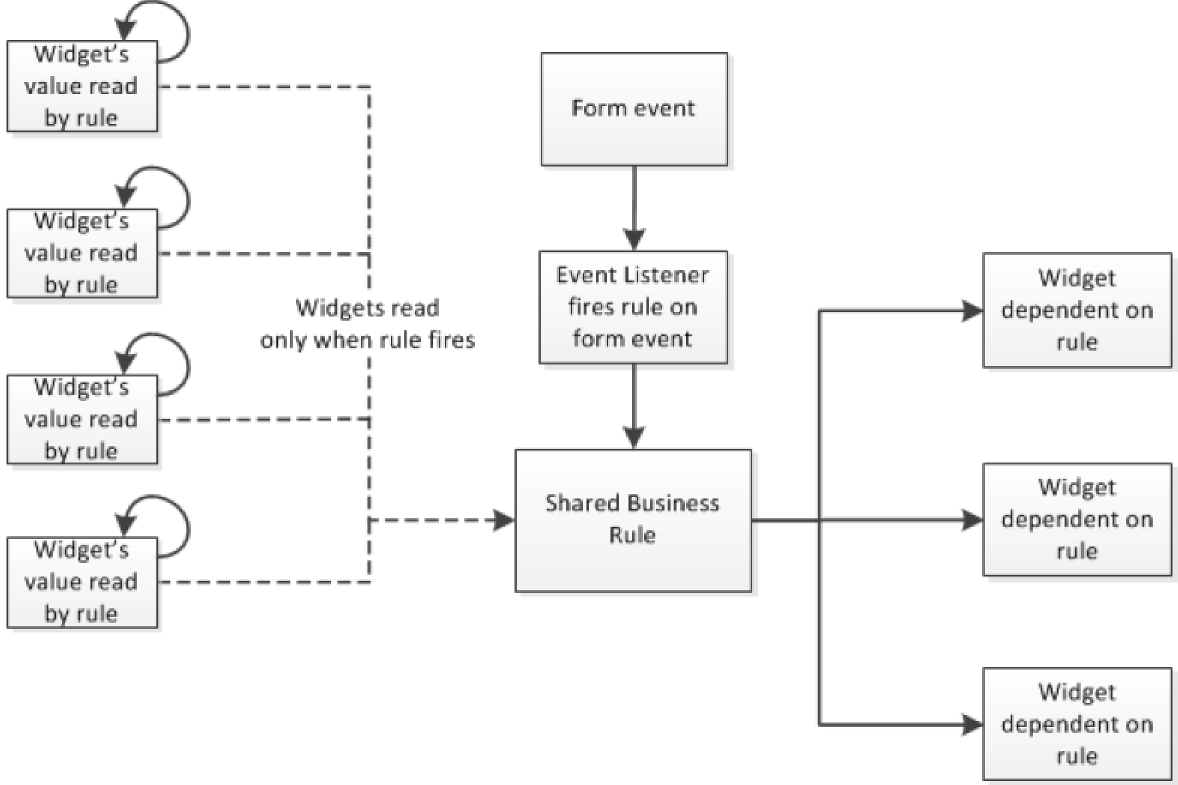
where the rule fires every time one of the nominated widgets changes. In other words, the rule's listener is forever observing these widgets for change and will fire every time one of these changes. This can slow the form down if the scale of all these observed widgets is large enough.



Rule fires on change of any observed widget; as a result, the rule could fire many times and make the form laggy

1. Additional Event Listeners

Here, the widgets linked to the rule can asynchronously change, but the rule will only fire when a form-level event is detected by the listener. The form events are listed, comma separated in the "Additional Event Listeners" field in the "Data" tab of the Business Rule widget.



Rule fires less and only on a nominated form event or events; results in a more responsive form.

The above second case results in the Rule widget firing less frequently than for the first case: every change to any of the observed widgets no longer results in the rule firing every time.

Standard Composer Events

Event	Description
init	The first event to which a widget can listen when the form starts. This event sets the initial data-based state.
postInit	Runs immediately after init. Used to run code when everything is in the initialized state.
change	The only event that does not call everything listening to it. It will look at the source node and look up the list of fields with a dependency on that field.
phase	Will fire when any phase value changes. evt.evtData is the name of the changed phase. Its new value is fetched with sfc.getPhaseValue("<phaseName>"), where <phaseName> is the name of the affected phase.
evtValidationV iew	This event is fired once all current validation updates have been completed and it is ready to update the error lists, etc.
preSubmit	this event fires immediately before the form is submitted (although there is an issue that preSubmit is not currently fired when using a form in Mobile App but this should be resolved in a future release).

Error

Error Business Rules will, on error, prevent the user from submitting the form.
The rule has more settings than the General Purpose business rule properties. The visibility rules in particular can cause confusion if not properly understood.

Focus Field Target

The rule fires when the user focus becomes the nominated field and whenever the value of the field changes. The rule can be set to "Run Only When Visible" (which is the default setting) or "Run Always". The nominated field will display the warning background colour while the rule is in error.

Clearing Policies

See Privacy on the privacy considerations that dictate data clearing policies. Note the distinction between "Clear Hidden Data on Submit" and "Clear Data When Hidden" .
The distinction may seem trivial if we consider only the form's data at the point of user submission. However, when one of form's Business Error Rules is dependent on a field that happens to be hidden, and that field is causing the rule to be in error, that hidden field would prevent the user from submitting the form, even though all the visible data is suitable for submission.
So, this is the reason for the distinction between "Visibility Data Clearing Policy" and "Editable Clearing Policy" in the rule's properties.

Displaying Errors

The default display of form errors is the standard footer of the typical template. The same structure is also available with the Error List Predefined Block . The errors are listed in a clickable list; double-clicking on a list item takes users to the form element needing recitification. You also can write the explanatory text displayed as well as the error messages themselves. You therefore have the tools to assist users though the process and, therefore, minimise the rate of form completion failures.
The Form can display both errors and warnings through the Error and Warning Predefined Block . Users can scroll through wither the errors or the warnings using next and previous links.
The Error Selection Block displays both of these blocks, one on top of the other.
The Edit Properties -> Rules tab of the Error Business Rule can also be set to displaying an error dialog: a popup with the text of the Warning Message of the Rules tab. The same also can be set for warnings.
The use of these various blocks gives you some flexibility in how you want problems with the form data to be explained to users and what to do to remedy these issues.

Warning

These rules behave the same as the Error Business Rules, except that warnings do not prevent the user from submitting the form. We have already discussed the display of warnings above .

Mandatory Checkbox Block

This mandatory block enforces the rule that its containing checkboxes must have at least one member selected before the form can successfully be submitted.

Mandatory Block

This mandatory block enforces the rule that its containing members and groups must have at least one member selected before the form can successfully be submitted.
There are, however, a subtle differences between using this widget or using the button assistant and checking the "Mandatory" box in the creation wizard for the assistant:
The Mandatory Block by default leaves all its contained buttons or checkboxes unchecked or without value. If end users try to submit the form with all these boxes unselected or unfilled, the error message for the block is displayed and all the contained elements get red outlines.
Checking the "Mandatory" box of the button assistant wizard means that at least one of the buttons (the first by default) is selected at form initialization and

there is no way to go to an all-unselected condition.
Several checkbox or buttons blocks can be contained in the mandatory block, all of them unchecked at initialization, and yet only one button of one group must be selected before submission and all the other buttons in the other groups can remain unchecked.

Library Advanced Features (Composer v4.3)

Overview

As we have already seen above in The Hierarchy and Environment , an Account has Libraries and Organizations under it. Also, there are also special libraries called "Releases" and "Service Packs" which we will call here "Compatibility Settings"; but these are assigned to Organizations. Also, the Organization itself has its own library elements attached to itself. Here are the different Libraries and their locations:

Library Type		Location or Association
Compatibility Settings	Releases	Live above the Workplace structure. Each Organization is associated with a particular release.
	Service Packs	Live above the Workplace structure. Each Organization may be associated with particular service packs. A Service Pack is also relevant to a particular release. Any Library that is neither a Release or Service Pack is termed an "ordinary" Library.
Ordinary Libraries	Native Libraries and Style Packs	Style Packs are Libraries packaged by Avoka Technologies. Otherwise, they are ordinary Libraries. "Native" here means that the Libraries live in the Account, and are therefore already shared with the
Library Type		Location or Association
		Account
	Shared Libraries and Shared Style Packs	These are ordinary Libraries that live in other Accounts and which have been explicitly shared with this Account.
The Organization's Internal Library		Lives inside the Organization, not — unlike the other Libraries in the Account — as a separate Library entity under the Account. The elements in the Organization's Internal Library affect all Projects in the Organization regardless of other settings.

Composer Account Administration

If you wish to carry out work on Libraries, you should consult the "Transact Composer Account Administration Guide". This is a handy reference on where to go in the Composer user interface in order to carry out such important Library-related tasks as:
Allocating users at the Organization and Project levels Creating and importing libraries
Inspecting which libraries are available to Organizations Assigning Libraries from other Accounts to an Organization
Maintaining Libraries and their contents (style sheets, templates, custom types and property sets) Ongoing maintenance
Maintaining compatibility through revision maintenance of Organizations
The Administration Guide gives a number of tasks to be completed in order to make Libraries and particular Templates available to Project users. The rest of this topic below discusses the theory on how the stack of inheritance works and what its effects are. The administration tasks and the theory work together and both need to be understood.

How Libraries Relate to Organizations and Projects

There are 2 properties in the behavior of libraries that need to be understood here:
Visibility and

Inheritance

Visibility is the easier to understand. Inheritance then is a consequence of visibility.
Inheritance is extremely important. It determines what templates are available at the form level and the style of each template at the Project level. Please refer to Inheritance Within Forms . especially the diagram showing how the inheritence of library components then affect the behavior of the form's elements.

Visibility

Ordinary Libraries at the Account Level

Ordinary Libraries — those not part of the Compatibility libraries — are visible at the Account Level. The rules for visibility for ordinary Libraries operate at the Account level and are:
Libraries that are live in an Account are visible to all Organizations in the Account. This includes Libraries imported into an Account via a package. ("Workspace -> <Account> -> Import Library or Organization link") The Libraries of an Account can be explicitly shared (or not shared) with other Accounts. (Go to "Workspace
-> <Account> -> <Library> -> Administration tab -> Library Share sub tab".)
Visibility means that the components of the Library, **with the exception of templates**, are shared by all Organizations and their children in the Account. Here, we do not include Administration (the configuration settings for visibility for users and Organizations) and Problems (a log of error messages for the Library).

Compatibility Libraries

Compatibility Libraries contain Releases and Service Packs; these are visible at the Organization Level. They are allocated at the Organization level in the wizard when a new Organization is created ("Workspace -> Create New Organization link"). Compatibility can be changed later (via "Workspace -> Organization -> Change Release action button"), meaning that the Organization will point to a different set of compatibility Libraries.
So, the consequence is that different Organizations in the same Account can be compatible with different Composer versions and service packs.

The Organization's Internal Library

In terms of its type of contents, the internal Library is pretty much an Ordinary Library. But in terms of its effect, it is the most powerful of all the Libraries because it lies above the Search Path settings of the Organization.

Search Path Inclusion at Organization Level

When an Account is visible to an Account, either by virtue of belonging to the Account or by being shared, the contents of the Library still doesn't affect the Organization until it has been included in the Organization's Share Path. You specify the share path in "Workspace -> <Organization> -> Update Organization Details action button -> second screen of the resultant wizard".
Note: the choice of Libraries for search path inclusion is only between Libraries native to, or shared with, the Account.

Templates

Template availability operates at the Project level. Only Templates from visible, search path-included Libraries can marked "Available" to a Project.

Visibility Summary

Note: Visibility has to cascade down from the Account Level. Thus, for a Template to be allowed for a project, its Library has to be shared at the Account Level and put into the Search Path at the Organization Level.

Level	Object	Location	Method
Account	Ordinary Library (Templates, Stylesheets, Custom Types, Resources)	Within Account	Always Visible
		In another Account	Shared with other accounts: "Workspace -> <Account> -> <Library> -> Administration tab -> Library Share sub tab"
Organization	Compatibility Libraries (Releases and Service Packs)	Assigned to Organization through "Release Settings"	In wizard when creating account "Workspace -> <Account> -> Create New Organization link" or when updating Organization "Workspace -> <Account> -> <Organization -> Change Release action

Level	Object	Location	Method
			button"
	Ordinary Library	Anywhere in the	In wizard when creating
	(Templates, Stylesheets,	environment.	account
	Custom Types,	Must be shared with the	"Workspace -> <Account>
	Resources)	Account and included in	-> Create New
		the Search Path	Organization link" and 2nd
			screen of the resulting
			wizard
			or when updating
			Organization
			"Workspace -> <Account>
			-> <Organization ->
			Update Organization
			Details action button", 2nd
			screen of the resulting
			wizard.
	Internal Library		

		Is automatically allocated to every Project in the Organization. Affects all Projects in the Organization and its settings overide all other settings.	Examine and populate this library through "Workspace -> <Account> -> <Organization> -> the tabs Stylesheets to Resources"
Project	Templates	Anywhere in the environment. Containing Library must be shared with the Organization	Allow or Block the template from being used in new forms in the project. "Workspace -> <Account> -> <Organization> -> <Project> -> Block or Allow this template to be used in new forms in this Project action button".

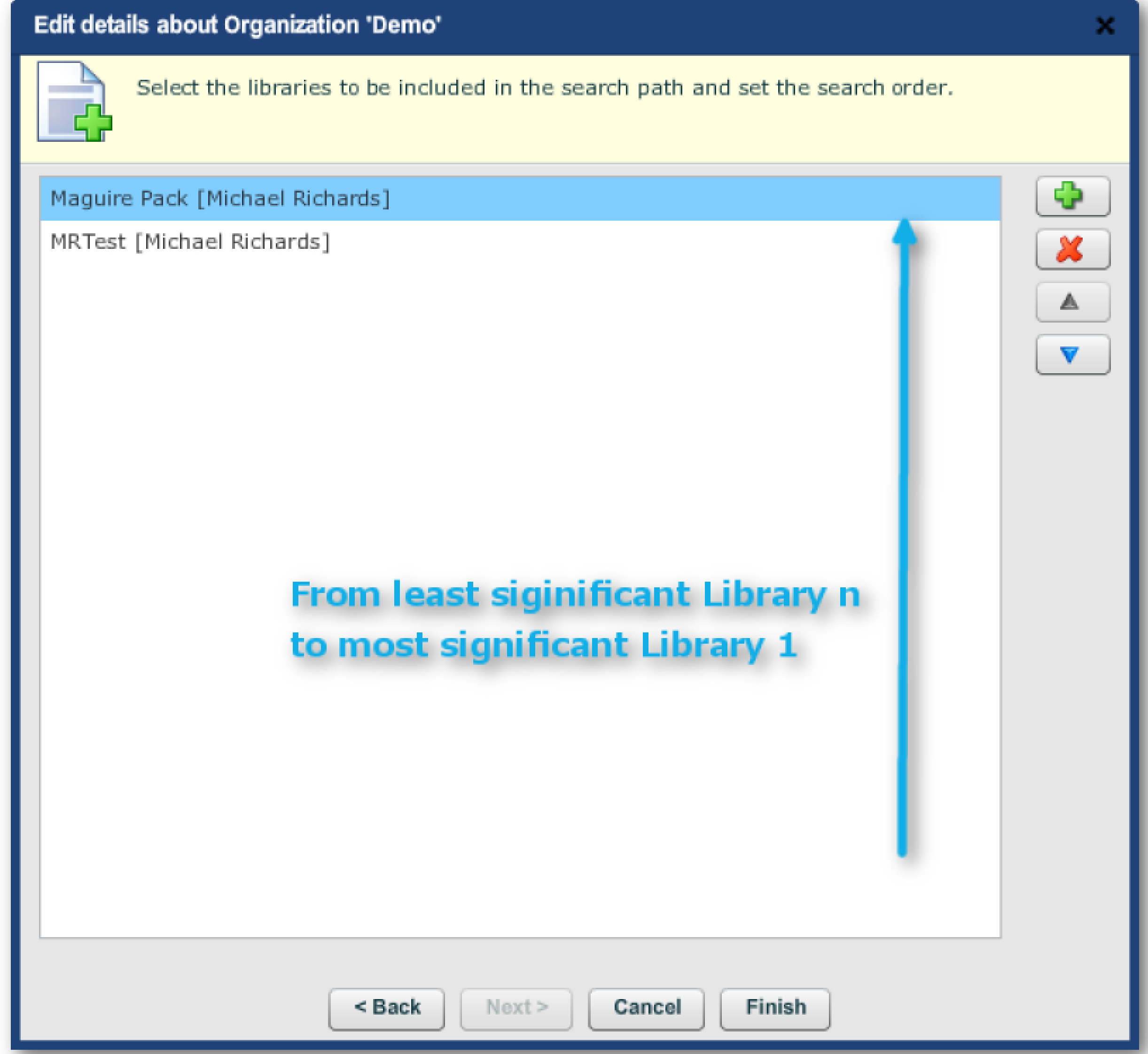
Inheritance

The Cascade

Stylesheets and Custom Objects now live in the Libraries, not in Templates. All Organizations now inherit all the Stylesheets, Custom Objects and Resources contained in the Compatibility Libraries.
This means that all users have access to all the objects in a release and that all the release's widgets will appear in the Form Editor's palette. (In fact, if you have the access, you can open a release and you will find that the widgets in the "Custom Types" tab, though you must check "Include non-editable types" to see them.)
So, there is a cascade of inheritance at the Project Level. which is where the Library visibility settings matter. Forms within the project inherit the Library visibility of their Project. The form inherits all the Stylesheets, Custom Objects and Resource, plus all of the allowed Templates, of all the visible Libraries.

Search Path Order

So, what happens when there is a conflict, for example stylesheets in different libraries with the same name? This is resolved as follows, moving through the stack of libraries from the least significant to number 1 in the Search Order and then to the Internal Library:
Compatibility Library -> Search Order Library n -> .... -> Search Order Library 1 -> Organization's Internal Library
The Internal Library, therefore, lies at the top of the stack.
You set the Search Order in "Workspace -> <Organization> -> Update Organization Details action button -> second screen of the resultant wizard":



Do note that where n > 2, the search order for the other lesser significance libraries may still matter because the element at the top of the stack may not specify every parameter and those that are not specified will be populated according to the order of values encountered in the stack.

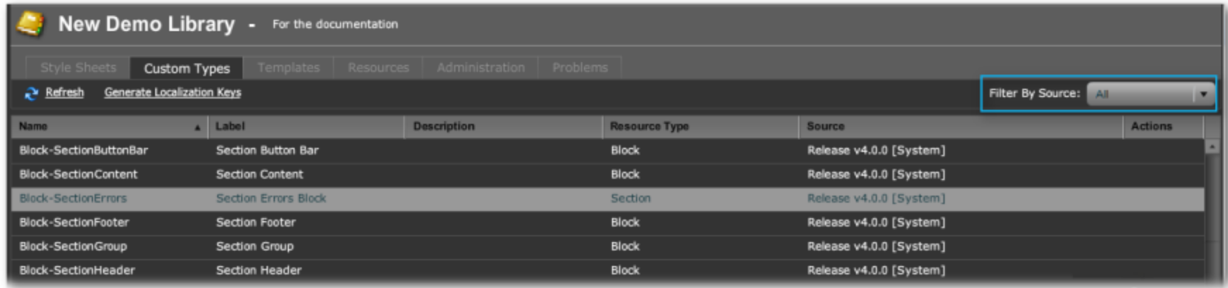
Custom Types

Location: Workspace -> <Organization> -> Custom Types tab or
Location: Workspace -> <Library> -> Custom Types tab
Custom Types refers to all the members of the Widget Palette of the Form Designer. There are 2 types of Custom Types:
Non-editable Custom Types
the standard members of the Widget Palette.
Editable Custom Types which are your saved types.

Localization Keys

used for the Internationalization of forms. We will discuss this elsewhere in this guide .

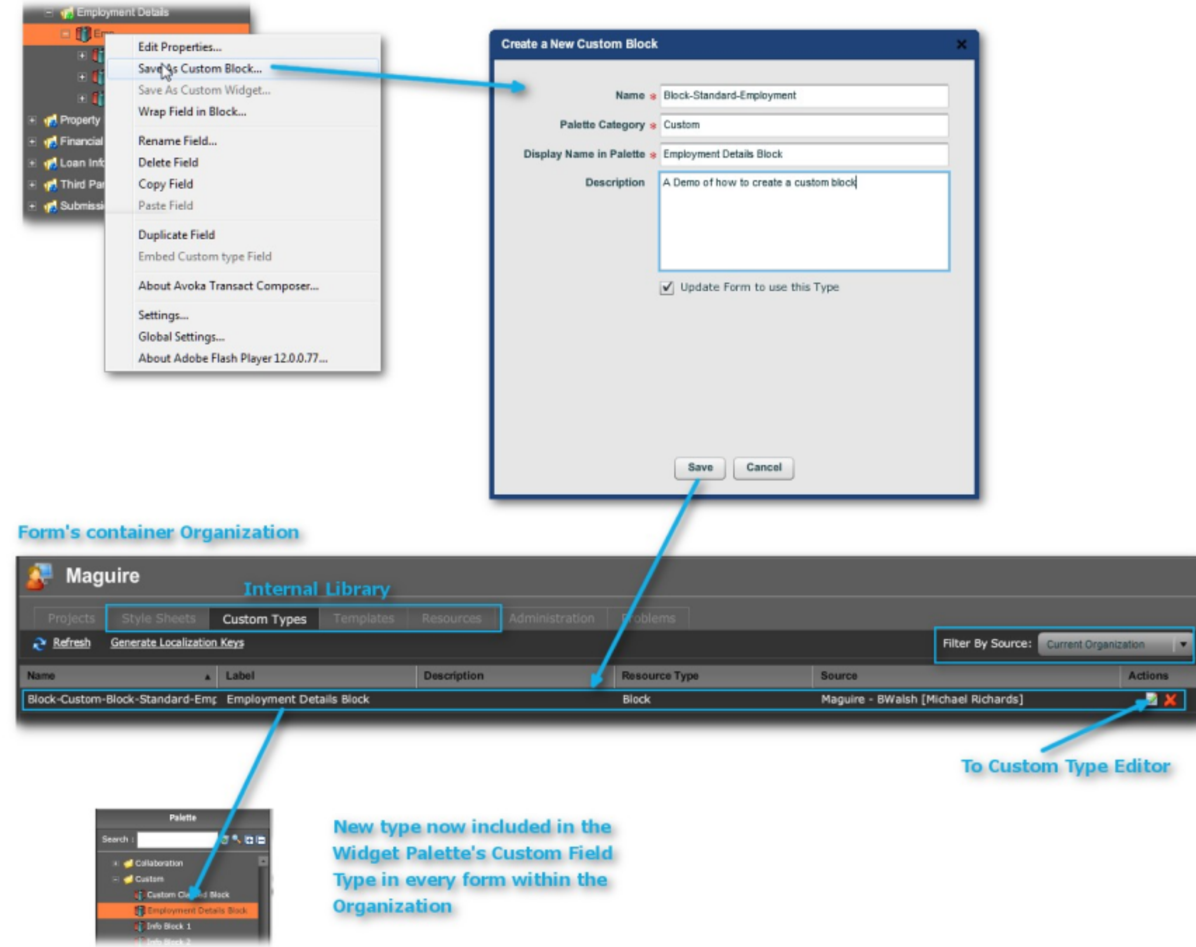
Non-editable



To list the non-editable Custom Types visible for a Library (either Ordinary or internal to an Organization), go to the Custom Types tab of the Library or Organization. Set the "Filter By Source" to "All".
There are no Action items.
All the non-editable Custom Types have been inherited from the stack , so the bulk of them come from the Release and Service Pack.

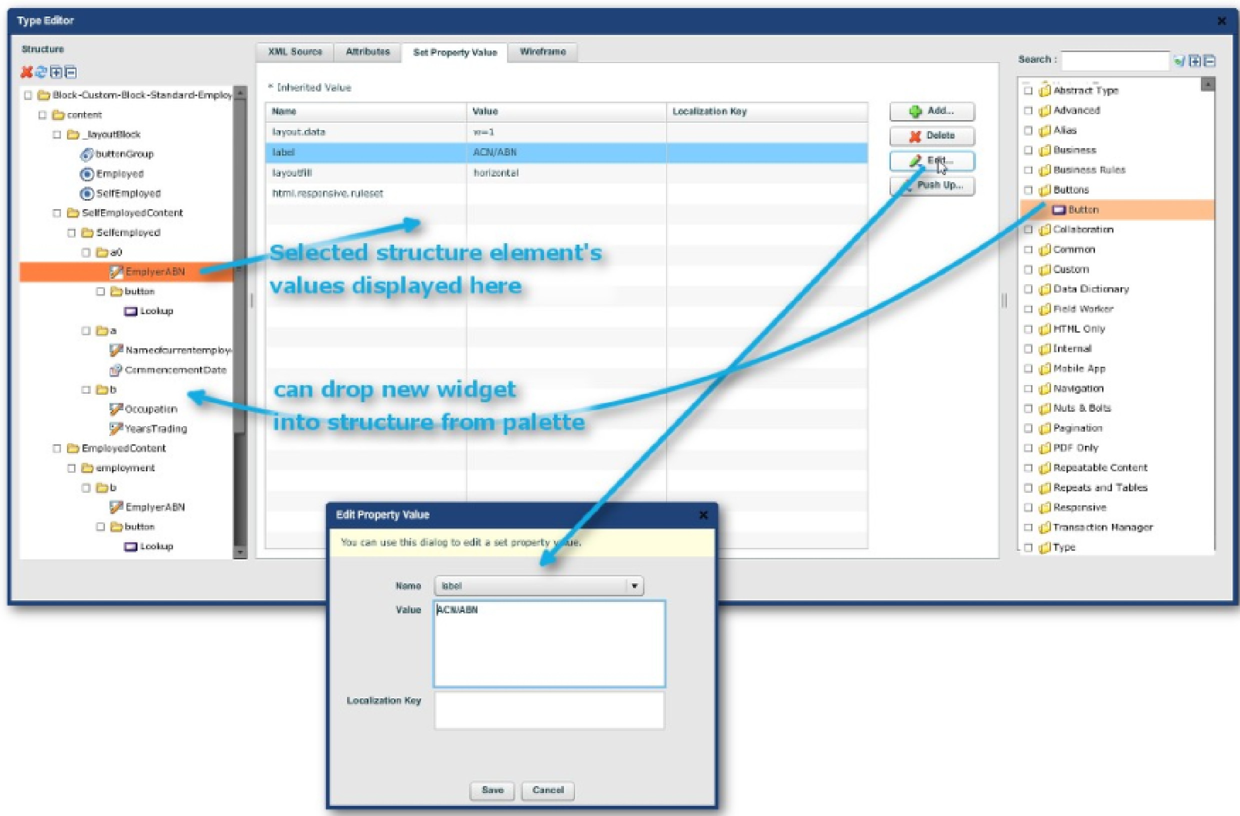
Editable

There is only one way to create a Custom Type: by using "Save As Custom Type/Block" from a form's Structure Panel. The newly-saved type will then appear in the Internal Library of the form's containing Organization. It is not possible, using the User Interface of the Form Designer, to create a Custom Type in one of the Account's Ordinary Libraries.



Because of the position of the Organization's Internal Library in the inheritance stack, your Custom Types will be available to all Projects in the Organization. However, you are not able to export these to other Organizations in the Account or to other Accounts. You can, though, export the whole Organization as a package and import it into another Account and the Custom Types in the imported Organization's Internal Library will also come across.

You can open the editor on editable types:



In the Set Property Values tab, you can edit the values of all the structure elements contained in the Custom Type. Inherited values from the stack are marked with an asterisk ("*").

Edit Property Definition

You can use this dialog to edit a property definition.

Name * binding.type

Value skip

Label Binding Type

Type * bindingtype-container

Visibility normal

Category Data Binding ☐ Category Not In List

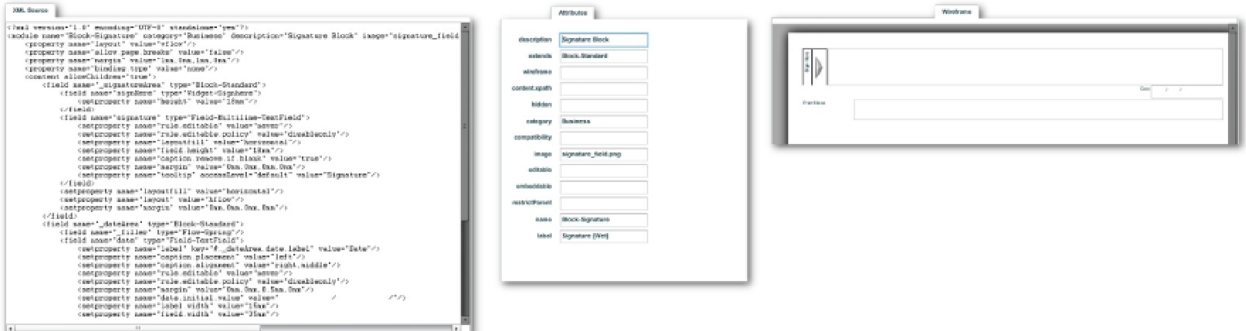
Group Data Model Binding ☐ Group Not In List

Create Pattern Don't show this property when creating the field

☐ Is Localizable

Save Cancel

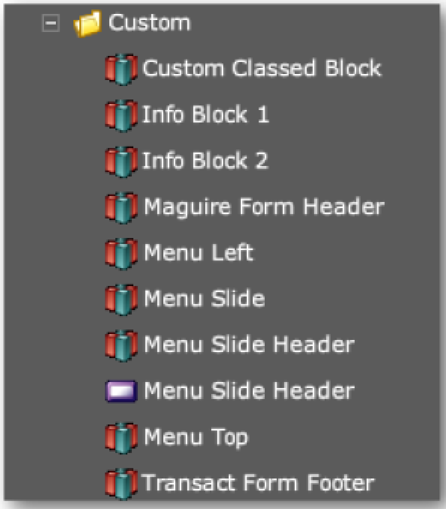
The **Create Pattern** dropdown menu in the Edit Property Definition dialog defines whether:
The property appears or does not appear in the wizard when the custom Type is dropped onto the structure panel of forms
and if it does appear in the wizard. whether the property is marked mandatory or not.] Here are examples of the other tabs in the editor:



Note that the XML Source updates on-the-fly as you add or subtract new objects into the structure or edit attributes and property definitions.

Style Packs

Custom Types play a major role in Style Packs. The Custom Field Type, for Organizations where a Style Pack is part of the stack, will have a rich selection of prebuilt widgets.
The Maguire Style Pack is a good example of this. It has a number of form structures not covered by the usual Nuts & Bolts settings.



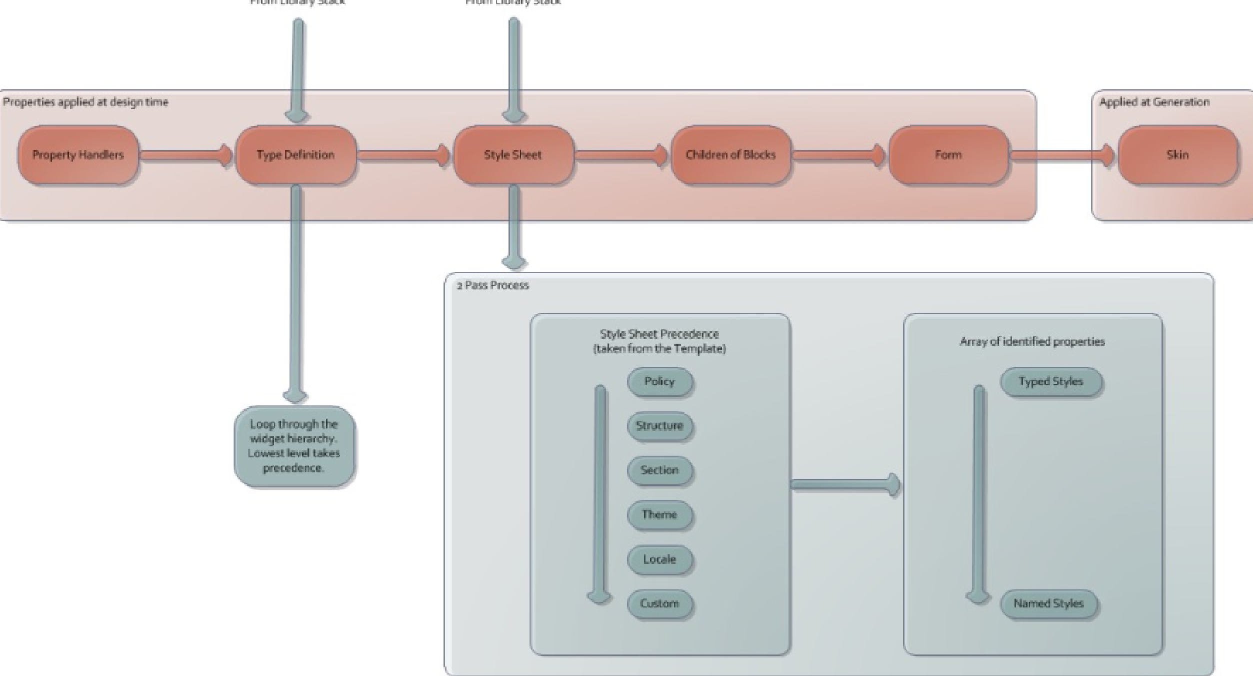
Besides these custom widgets in the Form Designer's Palette, such a style pack also has its own set of Stylesheets, Templates and Resources to supplement the Release. Typically, a release has a very large number of Stylesheets because the set has to cover practically every time of form element, while a Style Pack's set is very much smaller and far less general in scope.

Inheritance Within Forms

Overview

As above , forms inherit their Library elements (Stylesheets, Custom Types, Templates, Resources) through the stack. So, the library elements available to the form has already been established. This includes stylesheets that you have modified to change color-schemes and other components of the general look of your forms.

But then, inheritance now occurs through the form's structure:
The settings of the various Style Choosers
Settings of parent form elements and their children (blocks and their contents, for example) The use of Named Styles (or even manual over-rides



Widget Hierarchy

As we have already seen in Custom Types , some widgets are blocks of widgets and and children blocks. The properties of the simplest elements of these compounds are the ones to take effect. See the next topic immediately below.

Inheritance in blocks

Inheritance in blocks is similar to inheritance for individual widgets, but it is complicated by the fact that blocks contain content. When you drop a block onto a form, its content is also inherited, and can also be overridden, just like regular inherited objects. So you can simply double click on any of the child objects contained in a block, and modify their properties.
However, there are some important differences between inherited children and regular children. For example:

You cannot delete an inherited child (although you can change its visiblity to "Never visible", which achieves a similar result)
You cannot re-order inherited children
You cannot insert another object between two inherited children (although you can append an object at the end of the inherited children)
In order to differentiate between inherited children and regular children in the tree, inherited children are shown in yellow. This helps you to identify why certain operations (such as right click->Delete) are disabled for some objects.
If you make a change to a property of an inherited child object, only that particular property change is stored in your form (or overridden); all other property changes are obtained from the shared block. So if the definition of the block is changed, your form will automatically inherit all of the changes, except the specific properties that you've overridden. This makes maintenance of your form library very simple - you can change blocks in one place, and have the changes automatically reflected in all your forms.

Starter Content

Sometimes when you drag certain types of objects such as sections, tables or even complete forms, you will see a checkbox called "Include Starter Content". Starter Content appears similar to inherited content, but there are some important differences.
Starter Content is not really part of the block itself - it's just additional children that are optionally added to make your life easier and quicker. This is partly to save you time, and partly to show what type of content is appropriate for this type of object. There is no difference between the starter content, and content that you may have manually added yourself.
Therefore, you can modify, delete or move starter content just like you can any other object - none of the restrictions of inherited content apply.

Resources

The Resources tab in a Library is where you store uploaded files. In theory you can upload any kind of file. But, the User Interface of Composer really only allows you to utilize only graphics files in a form.
Other file types commonly found in library resources are JavaScript (.js) and Cascading Stylesheets (.css). Please refer to Library Advanced Features on how to utilize these in forms.
Other file types stored in the Resources tab are XML (for data definitions), XFA files, .zip files, backups and others. These are currently beyond the scope of this guide.

Custom CSS

Composer uses an abstracted stylesheet mechanism in order to allow styling of both HTML and PDF forms. For HTML forms, you can implement additional styling by directly specifying the CSS in the form definition to achieve effects not possible in Composer alone.
Note: CSS is rarely developed in isolation. It is usually specifically designed to add styling to a set of web pages that are structured in a particular way. This has two implications:

1. a. i. In order to use CSS effectively, you must understand the way that Composer generates HTML forms. The easiest way to do this is generally to open a Composer generated form in a browser that enables you to explore the HTML and CSS, such as Firebug (in Firefox), in Google Chrome or Web Inspector in Safari.

1. a. i. It is unlikely that importing an existing CSS file from, for example, your corporate web site, will have much value to be used directly in Composer forms (except in the most trivial of cases). It is more likely that you will extract certain elements from your existing CSS, restructure or refactor them, and then import the modified CSS into Composer.

Note: If you are going to be directly implementing CSS, you must not only understand CSS in general, but also be careful to test your CSS on multiple browsers and devices. The advantage of sticking with regular Composer styles is that we have done all the testing to ensure that changes work correctly in all browsers.

Custom JavaScript Libraries

Sometimes you may need to add custom JavaScript libraries to your project. This is an advanced topic, and we generally recommended that you add custom libraries with the help of Avoka professional services.
You can do this as follows:
Write your JavaScript functions using a text editor, and save as a .js file, such as mycode.js.
Drag the JavaScript Library widget from the palette to the Nuts and Bolts/JavaScript Libraries node. Specify a name.
Click the browse button to upload and select your mycode.js file to be associated with this library.

You may now invoke functions in mycode.js using the following syntax:
library_name.function(a, b);
If you want two different versions of the library for PDF and HTML:
Create two versions of the .js file. Ensure that each library has the same functions, with the same signatures, but with appropriate implementations.
Create a JavaScript Library object for each .js file, but check the checkbox to specify it's used for only HTML or only PDF respectively.
Override the Library Name property (which defaults to the Field Name by default) so that both Libraries have the same Library Name.
This way you will be able to use the same syntax to call functions, and it will automatically have the right version of the library in either the HTML or PDF forms.

Templates

Overview

We have already seen a demonstration of the power of templates in Composer . Composer 4.0 has an editor for creating new templates which now gives you control over some of the template properties.
which Style Choosers will be visible
what options each of these choosers has custom choosers
which form types will be available Custom form types
Layout of form headers and footers
However, the bulk of the work of creating a template involves editing

stylesheets
Using the Template Editor in conjunction with editing the XML source of the template uploading the appropriate resource files to be referenced in the XML source
There are about 120 stylesheets and over 300 Custom Types in the latest release, not to mention those in future service packs. Such a workload is the reason we recommend that Avoka Professional Services be engaged to create templates for you, if what you require is an extensive departure from the templates that ship with the product.
But, it is also true that no one would in reality contemplate a complete re-invention of the wheel: i.e. an overhaul of every stylesheet, property and parameter of the shipping libraries.
That being so, here follows some notes on the creation of template based on moderate intervention into the stylesheets

Stylesheet Modification

Location of the Stylesheet Modifications

The location of these revised stylesheets depends on where you want these modifications to sit in the Library stack .
Will these modifications be made available to A few Projects in an Organization
Or to all projects within one Organization.
To several Projects in more than one Organization, all in the same Account
Or to all Projects in several Organizations, still all in the same Account Or across several Accounts
Ultimately the choice is to have your new stylesheets either in an internal Library for one Organization or to be in a separate library in the Account and make sure that any form you intend to use the template has the necessary visibility of the library elements.

Scope of Stylesheet Modifications

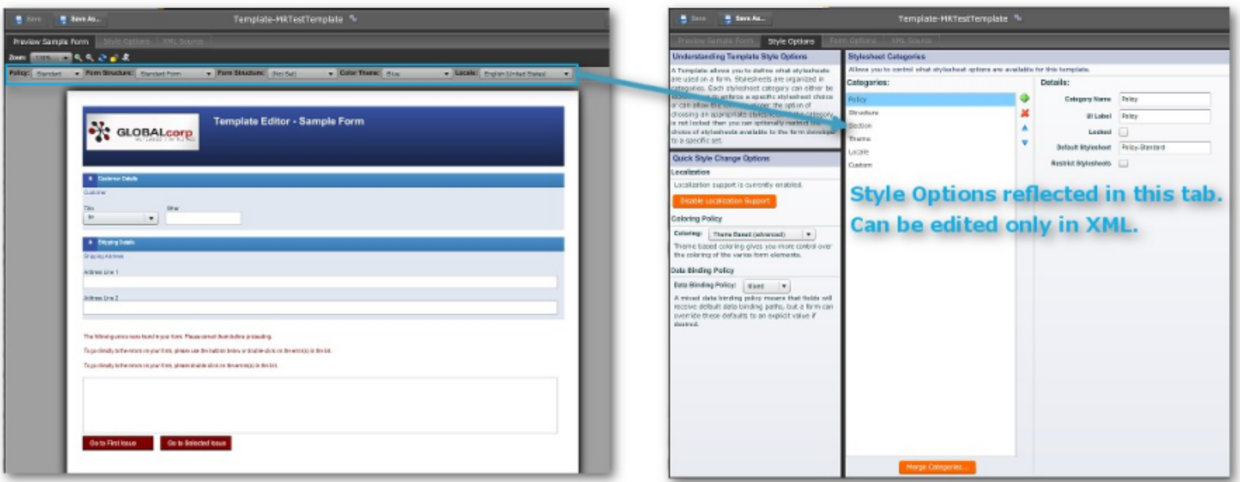
Just be aware that changing stylesheets will also affect the standard-issue templates. If you want the templates shipped with a release to still look and work the same as when they were installed, you should reserve, or create afresh, an Organization to hold the unedited stylesheets that accompanied these standard templates.

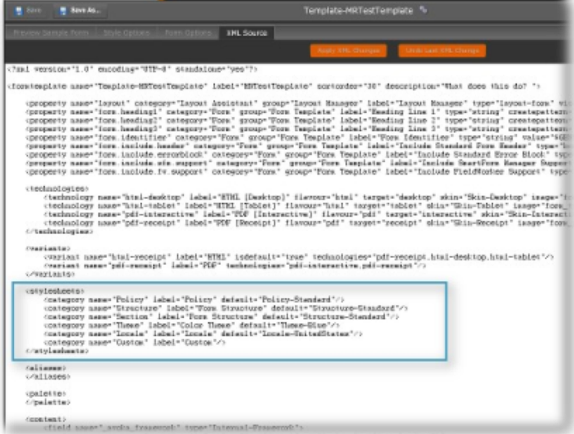
Editing the Elected Stylesheets

We have already described how to do this . If you want your new stylesheet to replace the standard issue stylesheet in the stack, you must give it the same name as the original. After you have saved the new version, you should be able to see it in the "Stylesheet" tab of the Library or Organization you have chosen to contain the modified version. Note the values in the "Source" column for stylesheets with the same name to check that you new version exists. Use the template editor to view if the change is expressing itself in the template and that it is having the intended effect.

Template Editor

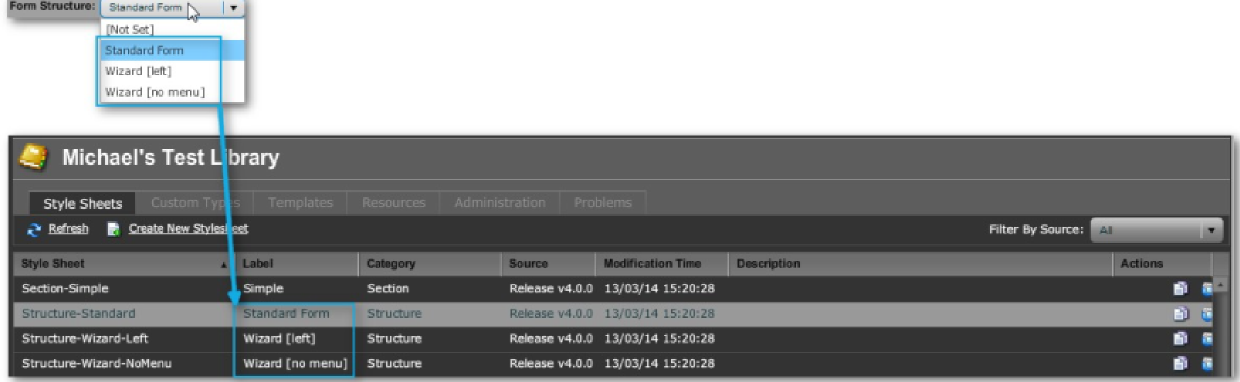
You access the Template editor through "Workspace -> <Account> -> <Organization> or <Library> -> Templates tab -> Edit Template action button". The editor itself has 4 tabs, 3 of which we show immediately below:





Style Options tab

The Style Categories panel of this tab allows for configuration of the Style Chooser dropdowns at the top of the Form Designer's wireframe.
The Categories are, as expected the stylesheet categories and the dropdowns are populated by each stylesheet in the given category.



For the Details panel of a Stylesheet Category, the settings are:

Detail	Explanation
Category Name	Cannot be edited. Derived from the Stylesheet category chosen for a Stylesheet Chooser dropdown
UI Label	The caption appearing next to the stylesheet chooser. For example, we here have "Form Structure" for the Structure category.
Locked (checkbox)	If unchecked, the category's chooser appears in the form wireframes. If checked, the category gets no visible chooser in the wireframe.
Default Stylesheet	The name of the default stylesheet in the chooser menu. Here "Structure-Standard", giving "Standard Form" as the default in the dropdown.
Restrict Stylesheets (checkbox)	If checked the chooser is populated by only a few of the available stylesheets in the category. A textarea appears, in which you list the stylesheets used in the restricted menu..

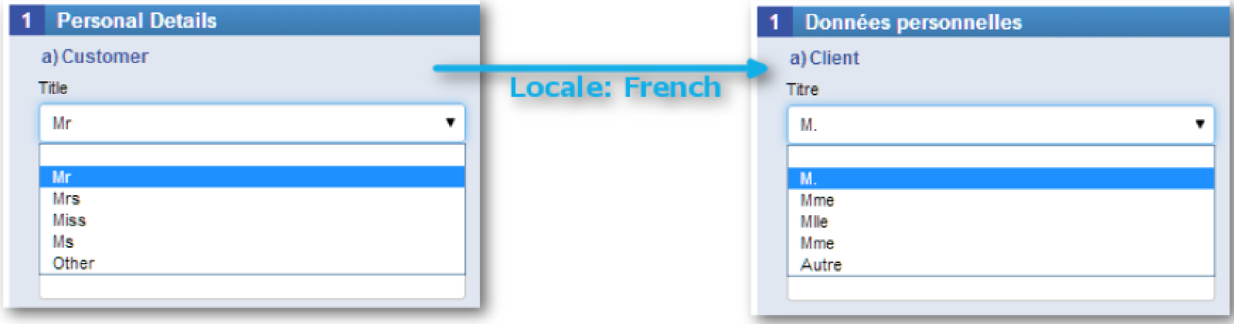
Editing the XML Source of the Template

Currently, though the set of stylesheets inherited to the template from the stack determine the layout and look of the template's content, the actual content of the form is dictated by the form's XML Source. This you can view and edit in the XML Source tab.
Editing this is a job only for the technically competent in this area, otherwise you could do a lot of damage. However, for those who feel up to attempting this, we have included the Template Schema at the back of this guide.

Localization

Overview

Composer's Localization feature allows you to provide your forms in multiple languages. Earlier in this guide, you would have seen a modest demo of localization in action .
For localization to be successful, you must have a native-speaker to edit the localization resources. For a rough draft you can use Composer's auto-translation feature (which uses the Google Translate engine), but if you have ever used Translate to render a foreign language web page, you would know that the results are less than convincing.
But once you have obtained a decent translation of a form, you will have no need to lay out the form anew for new supported languages (or "locales"). Even dropdown menus, and other widgets involving choices, will reconfigure to reflect the language change:

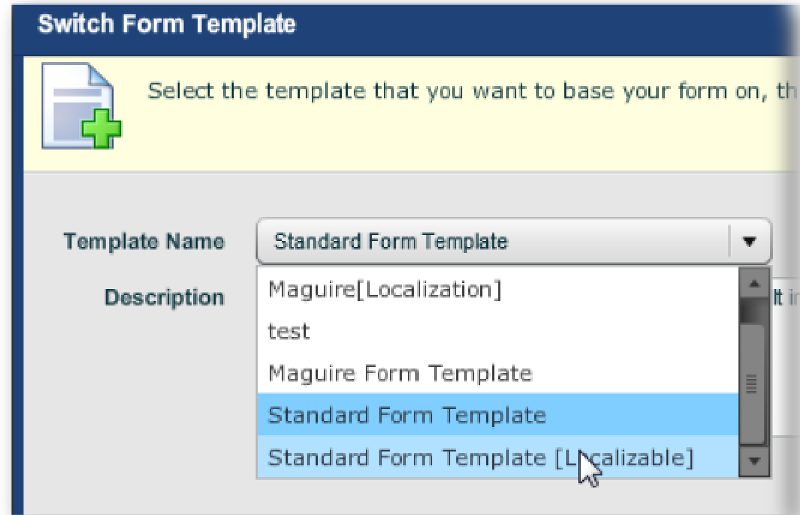


Activating Localization

The template used for a form must support localization. So, either you switch the form's template to one that does support localization, or you activate localization support in the template.

Switching Template

Done in "Workspace -> <Account> -> <Organization> -> <Project> Forms tab -> <Form> -> Switch Form Template button". Usually templates with localization activated indicate this in their title in square brackets.



You then can move to the next step: Setting Target Locales

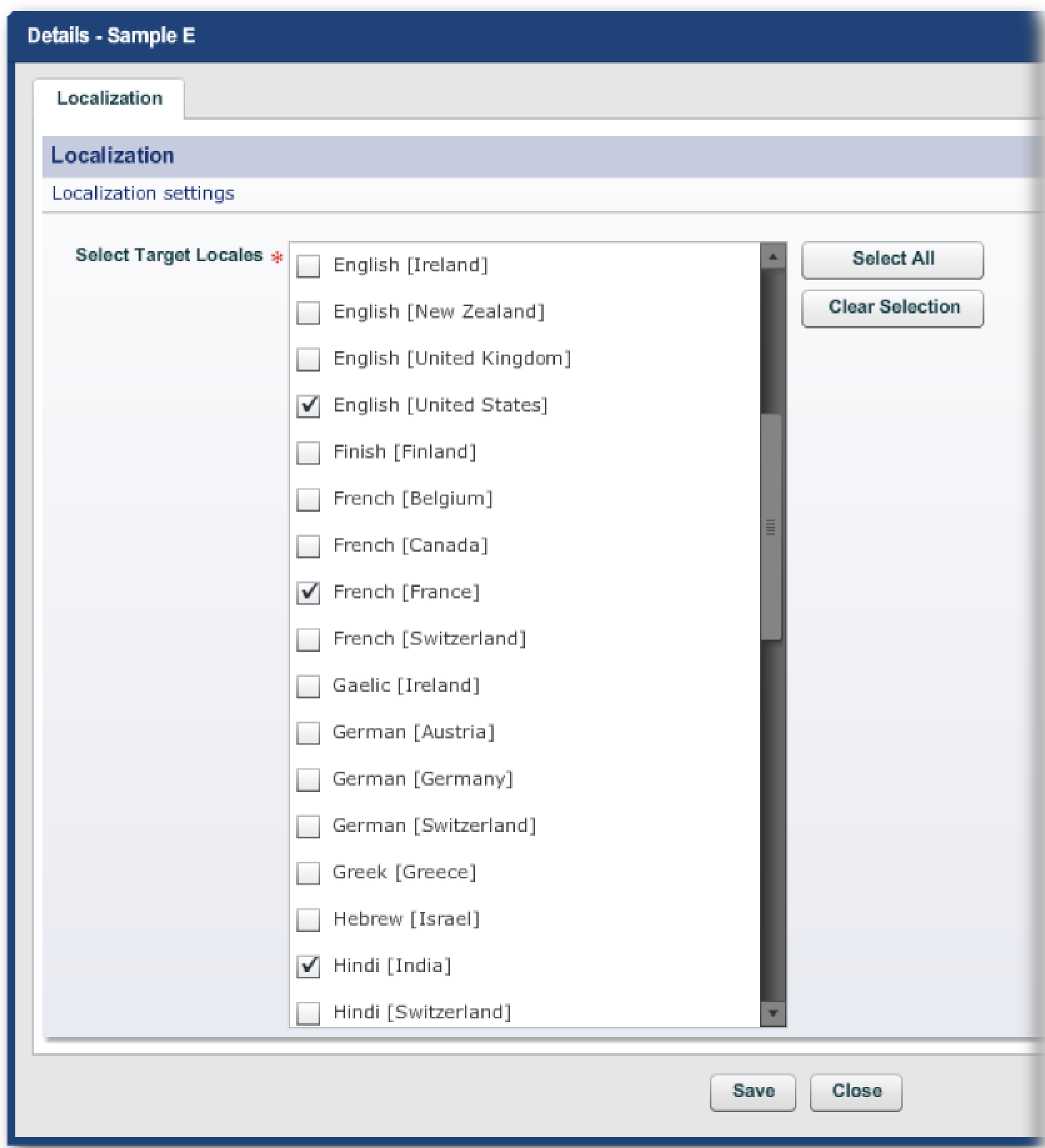
Activating a Template

You can also activate Localization in an existing template. Go to "Workspace -> <Account> -> <Library> or <Organization> -> <Templates> tab -> <template> -> Edit Template action button" which brings up the Template editor .
In the editor, "Style Options tab -> Quick Style Changes Options panel -> Enable Localization Support button".

Setting Target Locales

You have to configure Localization for each form.
Open the form in the Form Designer and go to the Localization tab ("Workspace -> <Account> ->

<Organization> -> <Project> -> <Form> -> Localization tab").
Click on the "Set Target Locales" link. This brings up a dialog for selecting locales.

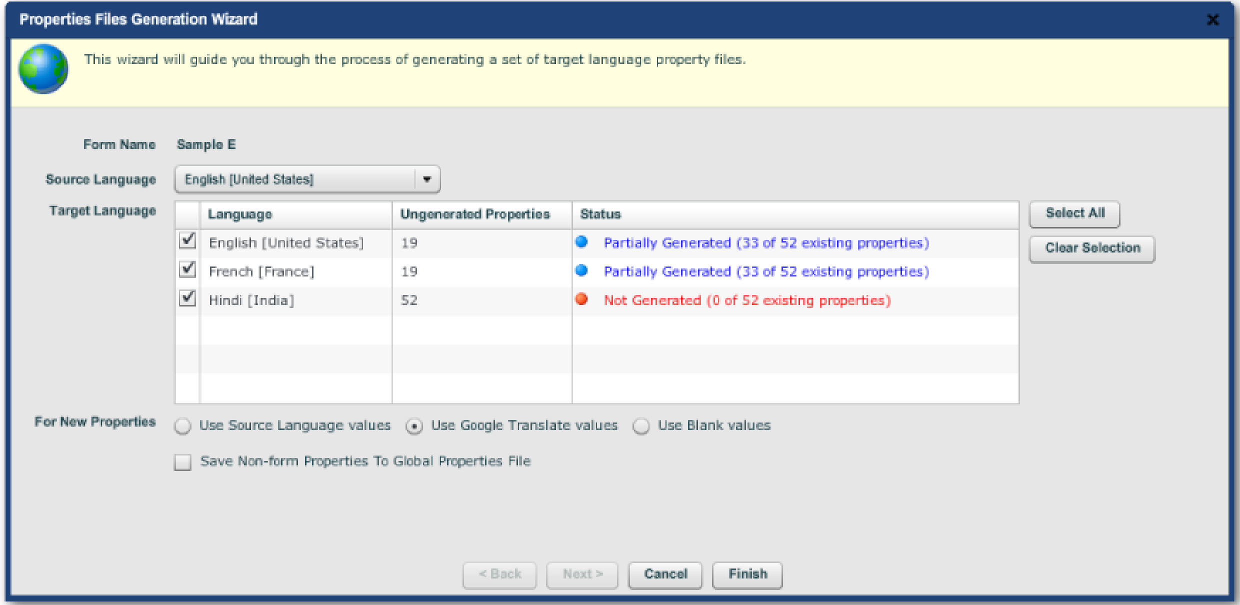


Generating Keys

You then have to generate keys for the form's elements. This is simply done by clicking on Generate Keys link in the form's Localization tab and completing the brief wizard. You can then move on the next step.

Generating the Properties File

Click on the Generate Properties File link in the form's Localization tab.



The "For New Properties" radio buttons and checkbox:

Use Source Language Values

create the new files, but leave the values the same as the original locale and let the (human) translator change the values to the new locale

Use Google Translate Values

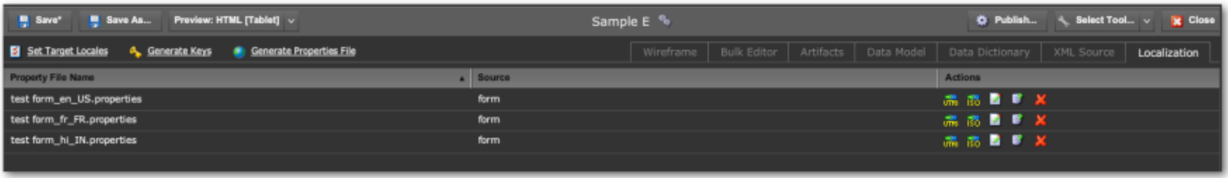
and use these values as a rough draft for the translator

Use Blank Values

let the translator fill in the blank values. This means the translator will have to have access to the Properties file of the original locale of the form.

Save Non-form Properties to Global Properties File

This means that a global properties file is created for each locale at the Organization's internal library resources level. It lists only the template's common properties (such as "Type- Section.section.help.image.tooltip" or "Button-SectionCollapse.tooltip" and so on).
Form properties are the content of the individual form, such as "form.customer.section.heading" or "form.firstName.label". Form properties are objects that begin with "form.".
These property files for form properties will appear in the Localization tab of the form:



Their contents look as follows:

The translators then can edit the translation of the properties. Be careful, though, that the diacritics and glyphs of any script (here Devanagari for the Hindi locale) are handled properly by the text editor used by the translators.

Deployment Options

The Localizations of the form can be deployed to different instances of the parent form in Transaction Manager, or for that matter to different portals.

Stylesheets (Composer v4.3)

Overview

Stylesheets are the building blocks to any template. This also means that there are a great number of these and they have many categories. You are even free to add to the list of categories (which you do in the "Save Stylesheet as..." dilaog shown below).
Stylesheets can be simple where they specify a few properties for a single element type on a form, for example a "rounded button".
They also can be compounded, built out of a set of simple stylesheets.
In Composer, all styles. whether simple or compound, are defined by a stylesheet. These are sometimes called "styles".

Typed Styles

A typed style is a style that applies to all objects of a particular type. For example, by default, all Buttons in your form have the same background color because there is a Typed Style for Button that defines the foreground and background colors for all Buttons. The same applies to all object types.
Note: Composer uses a fully object-oriented type hierarchy. Some styles apply to "base" objects in the type hierarchy. For example, a Text Field is a sub-class of Field. Properties can be styled at either the base-class level (in which case it applies to all objects of that base-type) or at the sub-class level (in which case it applies only to objects of that sub-class). This is the reason for the chains of names separate by
dots. Even if you do not grasp this concept, you will still find styles useful.
Normally, every object you may use in your forms will already have a typed style associated with it, which was created by the person who designed your organizational stylesheet.
There are many typed styles needed for any template, be it designed by Avoka Technologies for your organization or for one of the standard libraries that ship with the product. In either case, a full set of typed styles becomes available to an Organization after a library is assigned to the organization. These are read- only.

Named Styles

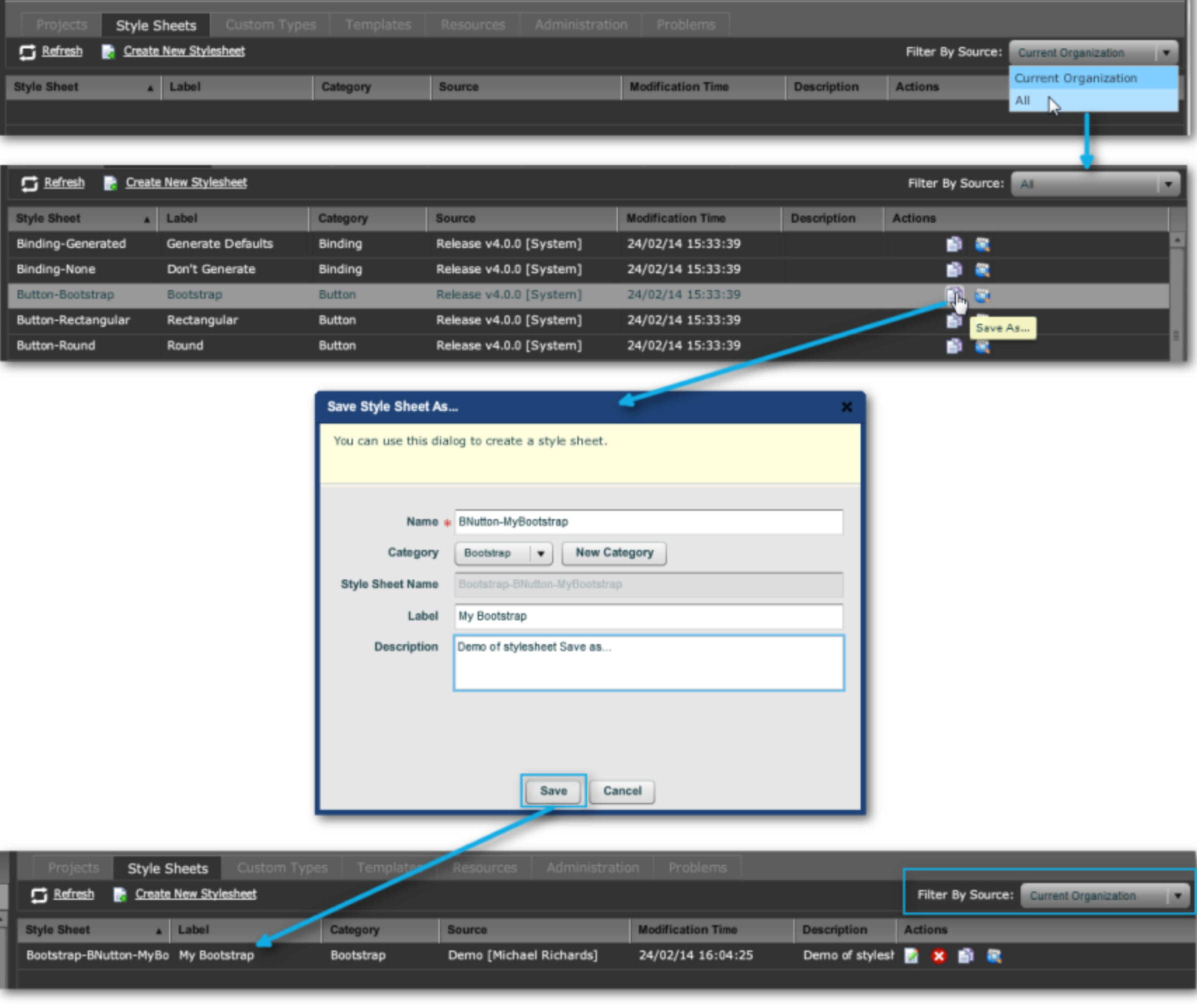
In most cases you want all fields of a particular type to always look the same on all forms. (And that is why we set up typed styles for each field type.) But occasionally, you will want to override some of the styling for a particular type of field. For example, there are two ways to represent a Button on a form — either as a traditional push-button, or as a flat mobile-friendly style of a text-only button. Though they are just two slightly different styles of buttons they have different foreground and background colors, fonts and so on.
You select these named styles for a form object by "Edit Properties -> Styling -> Style sets" . The available stylesets and their choices (for example, "Button Sizes", "Button Shapes" and so forth) are determined by the Named Styles associated with the Typed Style of the object.
Tip: It's generally a much better idea to define and use a named style, rather than explicitly setting "magic values" that you have to remember, such as sizes, colors and other properties. Not only does this mean you don't need to remember the values, but also, if you ever want to change the values, you can do so in one place.
Another example of using named styles is sizing of fields. Rather than explicitly setting widths in inches or mm, you can create named styles such as "large", "medium", and "small".

Stylesheet Categories

Stylesheets can now be categorized (either for convenience or for template choices, as we will see). You can use the already-existing categories or create your own.
Under the covers, you can define any number of stylesheet categories, and apply them in any order. Each category can define any number of styles sheets, each of which will appear in the drop down lists. Avoka has defined a standard set of stylesheet categories that work for many customers. you can define your own using the "Save Stylesheet as...." dialog shown below ,
The standard stylesheet categories that Avoka has defined go from the most general (overall form structure) to the most specific (colours and locale). However, it possible to define other stylesheet categorization schemes for organizations based on their needs. For example, it is possible to define an organization's template with no stylesheet categories at all – all forms would look exactly the same. Or perhaps with a choice of only two stylesheets in a single category. Or on the other extreme, you could potentially define a different stylesheet category for each different type of component.
The standard stylesheets provided by Avoka in the default templates are more to show the capabilities of the system – most organizations will only define a very small number of stylesheet combinations that represent the styles of forms in their organizations. On the other hand, some organizations, particularly those that "white label" their forms, may define a very large number of different stylesheets for each of the sub-organizations that they represent.
You then can, if you want, associate these categories with the Style Choosers of the template .

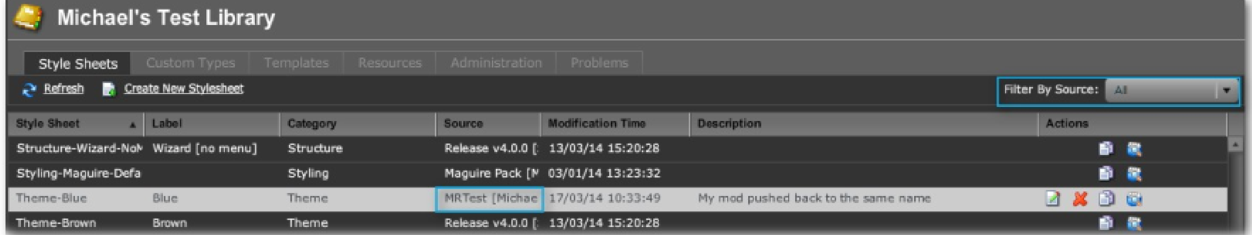
Editing Stylesheets

Location: Worksheet -> <Organization> -> Stylesheets tab or
Location: Worksheet -> <Library> Stylesheets tab
By default, an Organization can access the Account's stylesheets. These are all read-only. You can perform a "Save as..." on each of these so in that you can edit the new versions and also assign them to nominated libraries.

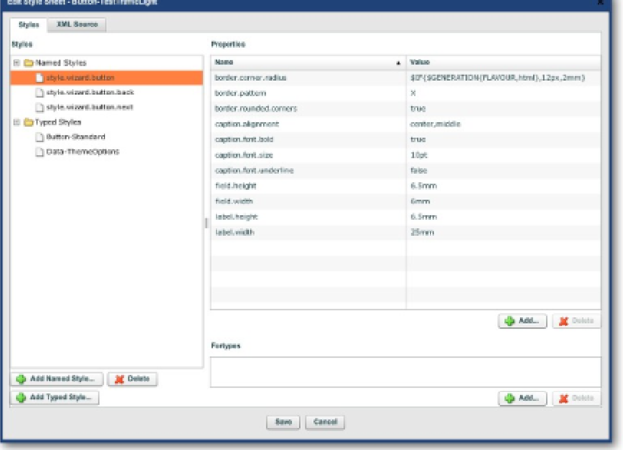


This is the way to populate both Libraries and Organizations with bespoke stylesheets.

So, you can replace a standard-issue stylesheet (from the release, say) with your own in the library by saving the stylesheet with the same name. For example, if I edit "Theme-Blue" and give it the name "Blue" and keep the category "Theme", my modifications are saved as Theme-Blue (but **only** in this particular library) and will affect all form elements that use the "Theme-Blue" stylesheet from this library in the stack. The entry in the Stylesheet tab for the Library shows the source of the style:



Once you have saved a stylesheet, you can edit its properties (double-click on item in the list or use the edit Action):



All of the values of the Properties of the Named Styles and Typed Styles are clickable and editable. You can also add and delete styles and properties and even edit the XML (though do not attempt to do so unless you know what you are doing). The Edit Stylesheet function is therefore very powerful.

Customizing Stylesheets

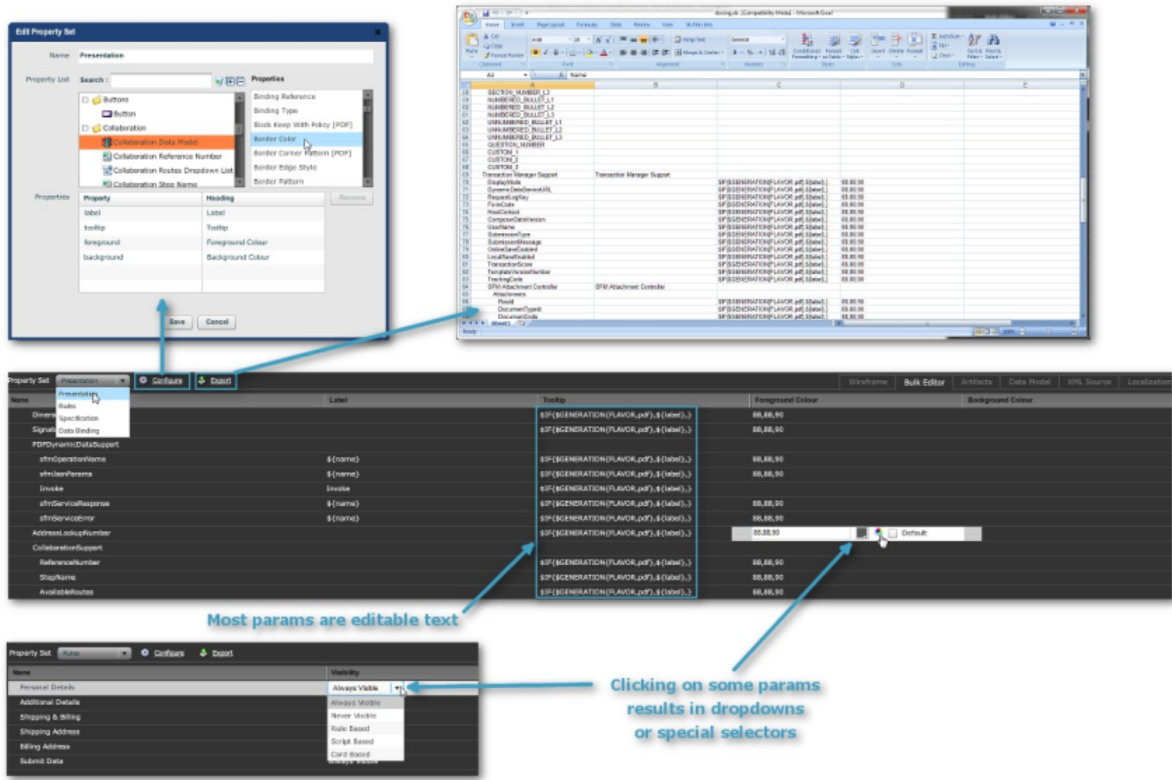
The ability to customize stylesheets is one of the most powerful features in Composer, for several reasons:
You can make global changes to your forms in a single place, and this will be automatically applied to all the forms that use that stylesheet.
You can control the scope of the changes by being careful about which library's stylesheets you change. For example, changes to the internal library of the Organization will affect the whole Organization. You can alter the scope of changes to other libraries by altering the Search Path and the Search Order for Organizations.
Even so, the scope Stylesheet changes seem to naturally affect Organizations. Trying to have Projects in the one Organization pick up different stylesheet versions is complicated and difficult.
Stylesheets hide (or encapsulate) a lot of the advanced properties associated with individual widgets. These advanced properties don't even appear in the Composer by default (you can turn on Advanced Properties in the Property Dialog), which means that you don't have to worry about them. You just get on with the job of designing your form.
The stylesheet embodies your organizational style guide, ensuring all your forms conform to the style guide. Just use the standard stylesheet, and let Composer do the rest. If you organizational style changes, simply change the stylesheet, regenerate all your forms, and you're done.
Tip: While stylesheets and their close relative, Templates, are very important, they are also moderately complex. We strongly suggest that you contact Avoka if you require either a custom stylesheet, or a modification to one of the standard stylesheets. We can either develop the stylesheet for you, or provide some customized training to enable you to make your own stylesheet changes.

Form Designer Advanced Features (Composer v4.3)

Overview

Besides the Wireframe tab of the Form Designer, there are:
The Bulk Editor tab The Artifacts tab The Data Model tab The XML Source tab The Localization tab
These do not provide further functionality than that already given in the Edit Properties dialogs and panels, but instead organize the parameter and settings of the form into tables. Some of these allow parameter settings to be changed and can be more convenient than going in and out of numerous Edit Properties dialogs for the fields and nodes on the form. The tabs provide an alternative and more comprehensive view of the parameter settings of the form than the atomic and fragmented glimpses afforded by going through the properties of a large number of individual form widgets.
There are at present some limitations: you cannot now alter the settings for the same type of parameter on a number of form elements. But, the ability to be able to survey the setting on, for example, all the Composer scripts within the form is a vary valuable facility.

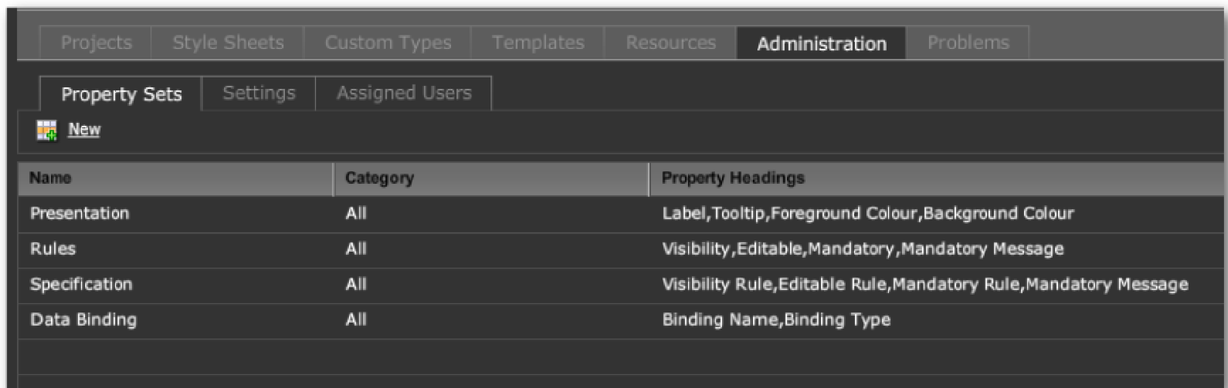
Bulk Editor



In the Rules Property Set listing, you can select "Script Based" under Visibility, Edibility and so on, and then use the "Edit" button that appears in the row to bring up the "Script Editor" , a very convenient way to access this dialog.

Property Sets

The form's parameters are arranged in Property Sets, which belong to the Organization. You edit, and create new, property sets in "Workspace -> <Organization> -> Administration tab -> Property Set tab".



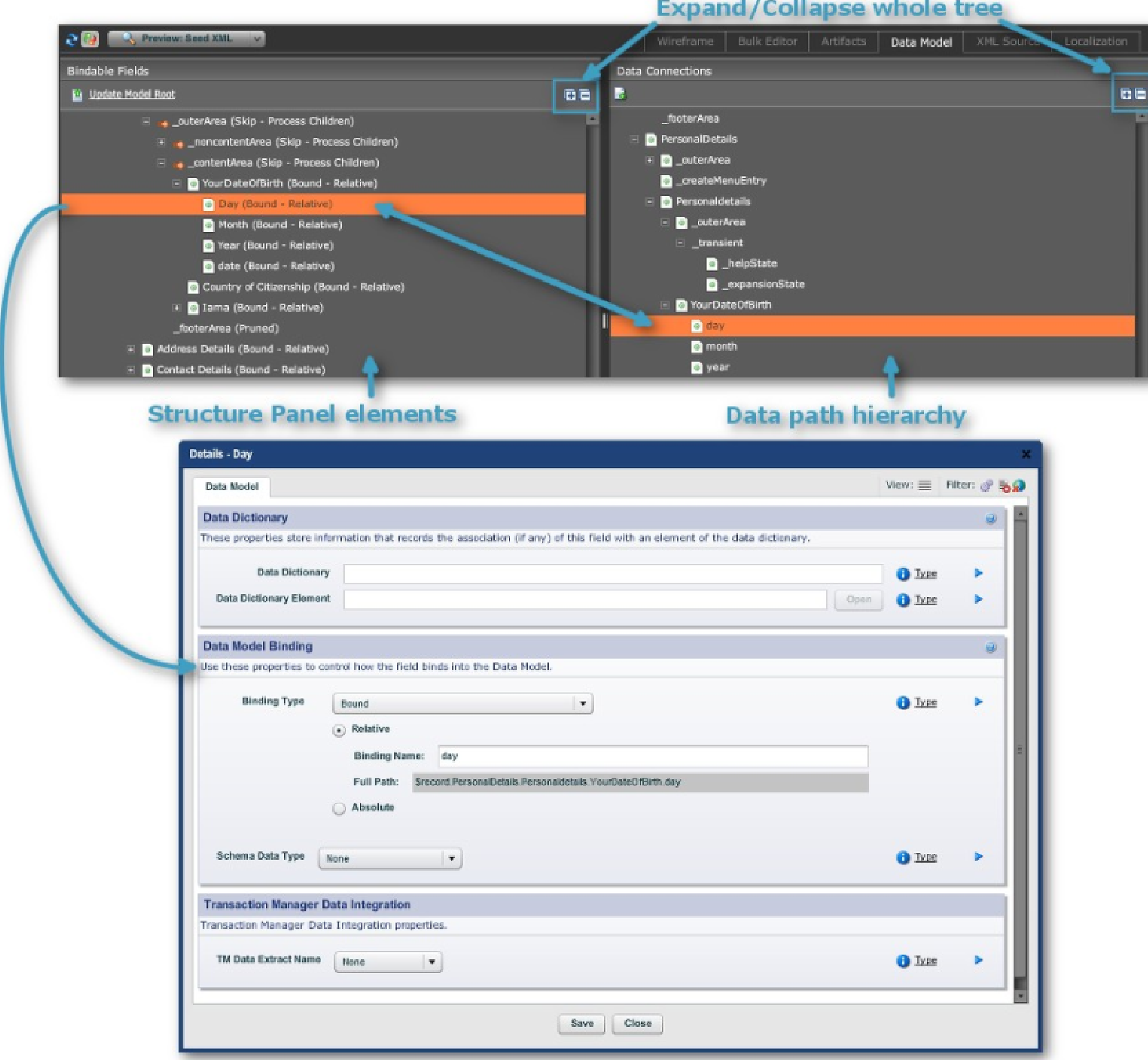
This the only place in the UI that you can create a new set. However, you cannot at present configure the new set. You must configure it in the Bulk Editor of one of the forms in the Organization, using the Edit Property Set dialog from the "Configure" link at the top of the Bulk Editor page. Here you select a property

from the Property List (which just like Widget Palette in the Form Editor) and then double-click on the Property you want in the Property Set. The property then appears in Property list in the lower half of the Edit Properties Set dialog. You can change the wording of the heading of the resulting Property Set column in the Bulk Editor

Artifacts Tab

This lists, and allows you to download, files associated with the form. Here is a partial list of the files:
<form name>.xls
A spreadsheet of all the form elements and their properties, including the Nuts & Botls
<form name>.xml
The form definition in XML.
<form name>_info.properties
A text file giving listing the creation date, time and time zone, and the preview technology .

Data Model Tab



The left-hand panel gives the various bindings (bound, unbound, relative, absolute) of the fields in the structure; the right shows the field's position in the data hierarchy. Select a field in one of these two panels and it will be selected in the other panel.
Double-clicking on a field in the left-hand panel will also bring up a dialog with the Data-related panels to be found in the "Edit Properties" dialog, though conveniently grouped into a "Data Model tab" created for this part of the User Interface.
The hierarchy depicted in the right-hand panel is the data hierarchy. In Composer, the data hierarchy is for the most part a reflection of the form's structure. This means that the data structure of the field bindings within a block is, for better or for worse, similar to the way the fields relate according to layout and block groupings. It is beyond the scope of this document to show how to change these relationships without altering the layout structure of the form itself.

XML Source Tab

This tab is an editable display of the XML definition file of the form. You can (provided you the rights to do so) save you changes, or you can rollback the last edits you have made to the code. Only do this if you are sure of what you doing; otherwise you can do serious damage to the form.
The XML frquently has Character Data ("CDATA") blocks, which are code that is passed straight through the
XML parser without it reading the block (which it would not understand in any case). CDATA here is used, in the main, to pass JavaScript fragments.

Localization Tab

This is explained elsewhere in the guide .
Transact Composer User Guide and Reference Access & Security

Access & Security (Composer v4.3)

Please refer to the Transact Composer Account Administration Guide.