

# Widget Developer Guide



Information in this document is subject to change without notice.

No part of this document may be reproduced or transmitted in any form or by any means, for any purpose, without the express written permission of TEMENOS HEADQUARTERS SA.

© 2010 Temenos Headquarters SA - all rights reserved.



## Table of Contents

Document History	4
About this Guide	5
Online Help	5
Licensing and Technical Support	5
Overview	6
<i>Widget Types</i>	6
Using Widgets	6
<i>Adding a Widget</i>	7
Widget Framework	7
Steps to Using a Widget	8
Widget Editor	9
Adding a Widget	10
<i>Widget Types</i>	12
<i>Widget attributes</i>	12
Widget Templates	13
<i>Adding a Header Template</i>	13
<i>Editing a Widget Template</i>	14
Customization	14
<i>Custom attribute specifications</i>	16
Using Customization Values	17
Customization Example	17
Widget Functionality	19
Source Template Content	19
<i>Data Capture Widget</i>	19
<i>Layout Widgets</i>	20
<i>Navigation Widget</i>	21
<i>Handling Hidden Conditions</i>	21
Header Template Content	21
Customization in Template files	22
Dollar Replacement	22
ITEM	23
<i>Datastore</i>	23
<i>General</i>	23
<i>Validation</i>	24
<i>Lists</i>	24
<i>Tables</i>	24
<i>Standard Common Attributes</i>	24
Custom Replacement	25
Styles in Templates	25



Callback Widgets	26
Callback Control Flow	26
Callback Class	26
<i>Example</i>	27
Market Place Widgets	28
What is the Market Place?	28
Market Place Widget Structure	28
Adding a Widget to the Market Place	28
Using a Market Place Widget	28
Widget Layout at Runtime	29
Data Capture Widgets	29
Layout Widgets	29
<i>Sections</i>	29
Columns	30
Tab Panes	30
Example - adding a menu widget	31



## Document History

Author	Version	Date
C Chavasse	5.2	27/11/2013
	5.3	20/04/2015

Comments:






## About this Guide

This guide will explain the use of widgets within an edgeConnect solution. It will explain how to use the Widget Editor to create and modify widgets and include them in your project. It is intended for edgeConnect developers who are familiar with the edgeConnect IDE and edgeConnect Script (see edgeConnect Script Guide for more information). Knowledge of javascript, jquery and css would also be useful.

### Online Help

The online help provides a full reference guide for all the features of the tool and can be used in conjunction with this guide. It is accessed via the Help item on the standard IDE toolbar or by clicking  on the standard button bar. The contents provide a list of all the major sections within the help. The index provides an alphabetical list of all the topics in the help, which is searchable. The search facility can be used to search for all topics related to a keyword you have entered.

### Licensing and Technical Support

Prior to starting any development work described in this guide, it is assumed that the edgeConnect development tool has been set up correctly for your environment. Product installation, setup and registration are described in the Installation Guide.

For technical support, please contact Temenos Product Support Portal at:

**<http://www.temenos.com/contact-us>**

For more information about Temenos Customer support, please see:

**<http://www.temenos.com/services/temenos-customer-support>**.



## Overview

edgeConnect Widgets are complex, customisable UI controls you can use in your solution. Some commonly used widgets are provided with the edgeConnect installation. However, you can also create your own widgets, use third party widgets or customise existing widgets. They are built using the Widget Editor provided in the edgeConnect IDE. You can modify them to be configured by the user either at build or runtime. At runtime, the widget source is used to generate HTML which is then inserted into the destination HTML.

### Widget Types

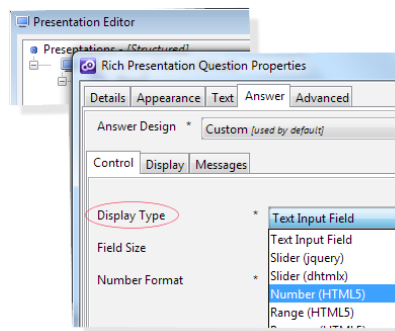
There are three types of widget:

- **Layout** - act as containers for other elements in the page
- **Data capture** - enable input of information by user
- **Navigation** - control movement within the solution.

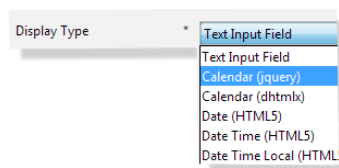
Each widget type has a different function and there are different considerations when creating each of these types, although their general set up and use is similar.

### Using Widgets

You use a widget by specifying it as the display type for the element you wish to apply it to. The display type is set in the Presentation Editor of the edgeConnect IDE (Tools --> Edit Widgets for Project). A drop down list of available display types for an answer is shown below. It includes several widgets which have already been added to the solution.

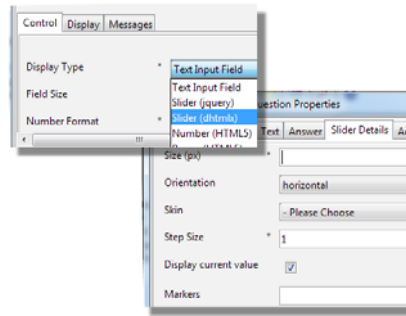


Another example below shows a list of widgets available for a date element. One of the options available is jquery Calendar, a date type widget which is included by default in the edgeConnect IDE as a Product Widget.



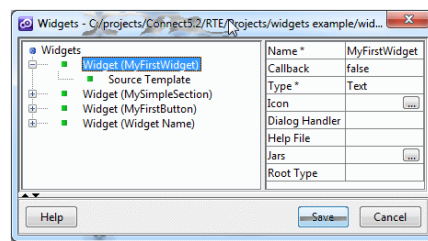
You can only use widgets that have already been added to your project and it must also be of a display type that is the same as the element you wish to use it for e.g. a slider widget can only be used for data capture not for layout.

If the widget has any customisable features, a new tab will appear once you have chosen it as the display type for your control. This new tab will allow you to enter values for the customisable parameters. The example below shows the additional tab displayed if you choose a slider control to enter a numerical value. Customisation allows you to control the size, orientation, step size, etc of the control.



## Adding a Widget

You add a widget to your project by using the Widget Editor tool which is provided as part of the edgeConnect IDE. This allows you to specify the widget files that are required for a widget. It can be a widget you have created or a third party widget you are using.



The widget editor is also used to add or modify the widget content and add any customisable features to it.

## Widget Framework

The details of all widgets used by a project are contained in a single xml file held at the root of the project. This file is called **widgets.xml**. An empty widget.xml file is created by default for each project. The Widget Editor is used to make changes to this file. When you add a widget to a project it is added to this file.

A widget has a number of attributes which describe what it does, how it will appear in the IDE and any related files it uses e.g. help files, jars. Only two attributes are mandatory, name and type.

The functionality of a widget is defined in template files which contain content that will be included in the page of HTML output at runtime. Each widget has one source file and optionally one or many header files. The source template contains the content and any logic required for the widget. The source template files may refer to script files or css files which are held in the web context. These are included in the header template files. These will be included in the header part of the page when it is generated.

A widget may also have user configurable parameters which can be set in the IDE. The xml describes what parameters are available and how they are displayed in the IDE.

An example of the structure of the file is shown below. You can edit widgets.xml directly, however, to avoid errors it is recommended that you use the Widget Editor to make your changes.

```

<!-- JQuery Graph -->
- <widget name="JQPlot Graph" type="Item:jqplotgraph">
  <template type="source" value="/templates/Widgets/jqplot/graph_instance.wgt"/>
  <custom>
    <item name="TRANSACTIONS" type="com.acquire.intelligentforms.ide.dictionaryeditor.PropertyChooser" prompt="Array with Transactions"/>
    <item name="BALANCES" type="com.acquire.intelligentforms.ide.dictionaryeditor.PropertyChooser" prompt="Array with Balances"/>
    <item name="WIDTH" type="javax.swing.JTextField" prompt="Width (with px)"/>
    <item name="HEIGHT" type="javax.swing.JTextField" prompt="Height (with px)"/>
  </custom>
</widget>
<!-- End JQuery Graph -->
<!-- Start JQuery Mobile -->
<!-- Button -->
- <widget name="JQMBUTTON" type="Button" icon="$PROJECTHOME$/images/jqmbutton.png">
  <template type="source" value="/templates/widgets/jquery-mobile/jqmbutton.wgt"/>
  <custom>
    <item name="CORNERS" type="javax.swing.JComboBox" prompt="Corners" typeValue="true^Yes|false^No" tab="Button Details"/>
    <item name="ICON" type="javax.swing.JComboBox" prompt="Icon" typeValue="home^Home|delete^Delete|plus^Plus|minus^Minus|arr
Right|arrow-l^Arrow Left|check^Check|gear^Gear|grid^Grid|star^Star|custom^Custom|refresh^Refresh|forward^Forward|back^Back
  </custom>

```

Attribute

Template file

Customisable Parameters



## Steps to Using a Widget

Creating a widget is carried out as a series of steps. These are shown in the diagram below and described later in this guide. Not all the steps are mandatory and once the widget is created it can be amended at any time to alter the functionality.







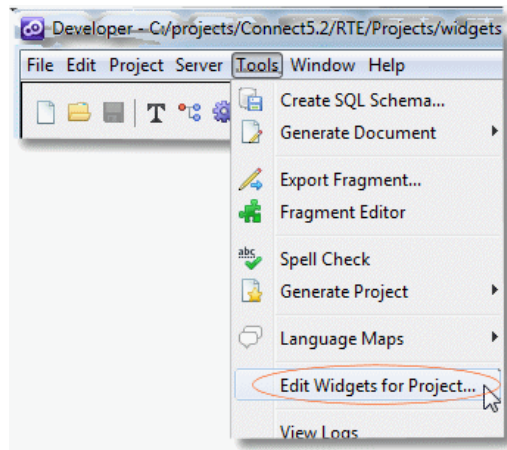
## Widget Editor

The Widget Editor is a tool available within the edgeConnect IDE which is used to create and add widgets to your Project. This must be done before you can use a widget in your project. It enables you to make changes to the widgets.xml file which contains the specification for each widget used in the project.

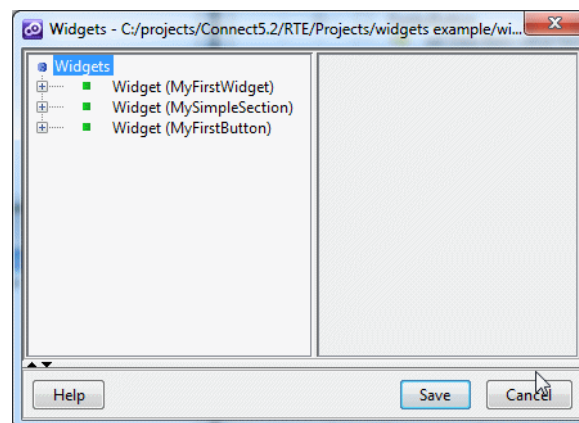
An empty widgets.xml file is created when you open a new project. You can edit this file directly, but it is recommended that you use the Widget Editor to avoid errors.

### Using the Widget Editor

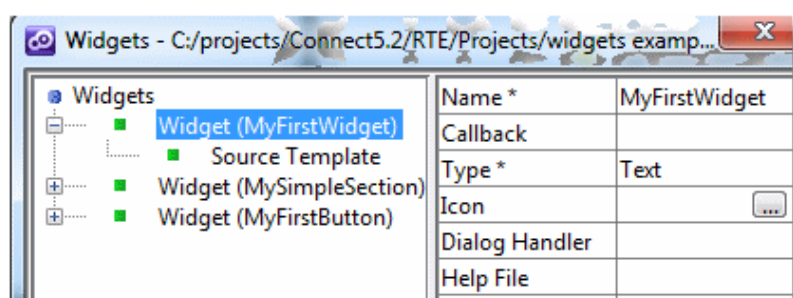
- 1 On the edgeConnect IDE Menu bar, click **Tools**, click **Edit Widgets for Project**



The Widget Editor will open. Any widgets already added to the project will be displayed below the widget root in the left hand pane.



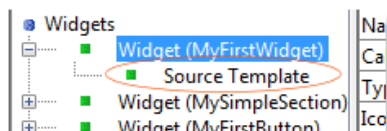
- 2 Click **Widget** to display widget details in right hand pane



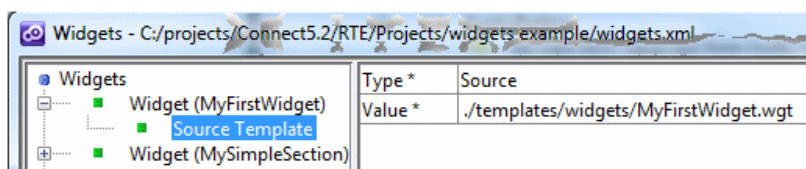


Edit the attribute details by typing or selecting the appropriate value in the right hand column of the attribute details pane.

- Expand the **widget** to show any templates it uses



- Click the **widget template** to display the template details in the right hand pane



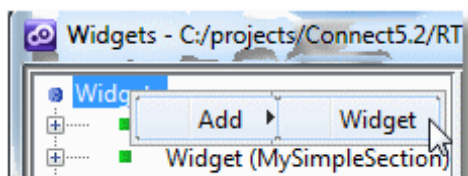
- Click **Save** to save and changes or **Cancel**

The widget editor window will close.

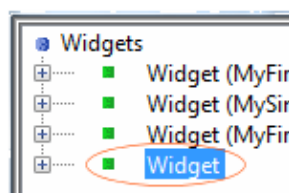
## Adding a Widget

### To Add a Widget to a Project

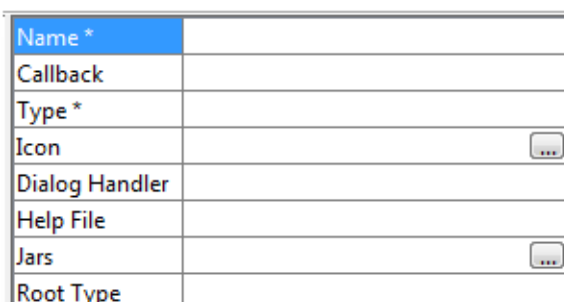
- Open the Widget Editor
- Right click on the **widget root icon**
- Click **Add**, click **Widget**



A new widget is added to the bottom of the list of existing widgets in the Project.



An empty list of attributes is added to the right hand pane.

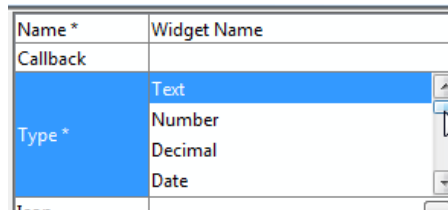




#### 4 Enter widget **Name**

The widget name is used in the drop down list of display types in the Presentation Editor when you select a display type for an element.

#### 5 Double click **Type** field. Select type from drop down list



The type specifies the type of element the widget can be applied to.

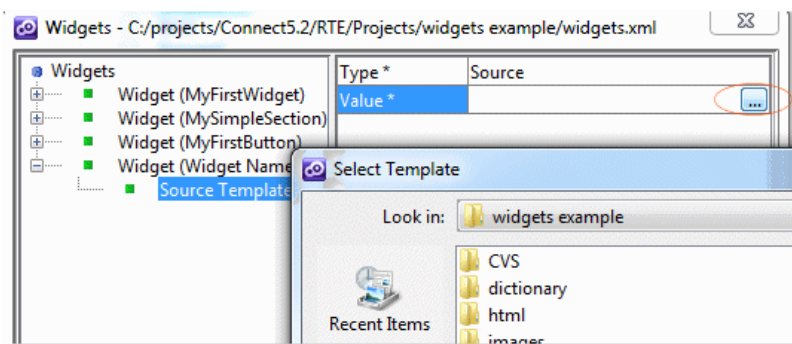
**Name** and **Type** attributes are mandatory. The other optional attributes will be described later in the document.

#### 6 Add any other attributes

#### 7 Expand widget and click on **Source Template**

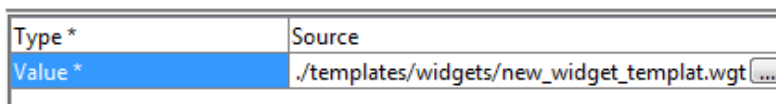
#### 8 Click elipsis for Value attribute

The Template file chooser is displayed.



#### 9 Select the template widget (.wgt) file if it already exists.

If the widget template file does not exist, go to the location you wish to create the file in and type the file name. Click Select to create a new file.



#### 10 Click **Save**

A new widget will be added to the Project.



If you wish, you can see the new widget has been added to the widgets.xml file in the Project root.

```

1  <?xml version="1.0" encoding="UTF-8"?><widgets>
2  <widget callback="false" name="MyFirstWidget" type="Text">
3    <template type="source" value="./templates/widgets/MyFirstWidget.wgt"/>
4  </widget>
5  <widget name="MySimpleSection" type="Section">
6    <template type="source" value="./templates/widgets/MySimpleSection.wgt"/>
7  </widget>
8  <widget name="MyFirstButton" type="Button">
9    <template type="source" value="./templates/widgets/MyFirstButton.wgt"/>
10 </widget>
11 <widget name="Widget Name" type="Text">
12   <template type="source" value="./templates/widgets/new_widget_templat.wgt"/>
13 </widget>
14 </widgets>

```

## Widget Types

The type of the widget roughly corresponds to the types available for data input, layout and navigation within edgeConnect. They define the types that the widget can be used with. The type must be from the lists shown below depending on whether it is a data input, ayout or navigation widget.

Data Input	Layout	Navigation
Text	Format	Button
Number	Column	Menu
Decimal	Tab	Progress Bar
Date		Breadcrumb
List		
MultiSelect List		

The type is used to determine if the widget can be used as a display type for an element in the solution. The display type is specified in the Presentation editor. It is possible to have more than one type for a widget. In this case a comma separated list of types should be used e.g. type="Number, Decimal"

## Widget attributes

Other widget attributes can be specified when adding a widget, but only Name and Type are mandatory. A list of attributes available and what they are used for is shown below.

Widget Attribute	Description	Possible Values
Name	Name of the widget	String values
Callback	Indicates whether widget is a callback widget	True or false
Type	Widget type	Choose from drop down list of types
Icon	The icon displayed in the IDE when the widget is used	Icon filename
Dialog handler		Dialog handler name
Help file	Help file to display for the widget	Help filename
Jars	Jar required for the widget	Jar file name
Root type	The name of the root type if you want to extend an existing type	



## Widget Templates

There are two types of widget template files:

- Source - mandatory
- Header - optional

**Source** template files contain the content which will be used to generate the HTML for the widget which is, in turn, added to the page at runtime. An empty Source template is added to a widget by default when it is created. Adding content to the Source template is required for the widget to function correctly. The type of content required is described later in this document.

Source template files may refer to other script or css files held in the web context. These files are added to the widget as **Header** templates, their content is included in the header part of the page at runtime. If you use the same widget multiple times on a page, the header template will only be included in the header section of the page once. If you use script in your widget, you should aim to put as much as possible in a header template to reduce the size of the page at runtime.

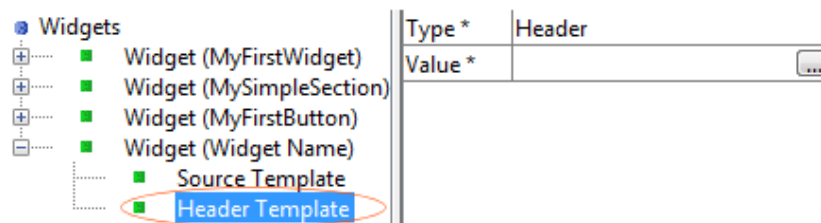
### Adding a Header Template

If a header template is required, it must be explicitly added to the widget as they are optional and not added by default.

#### To Add a Header Template to a Widget

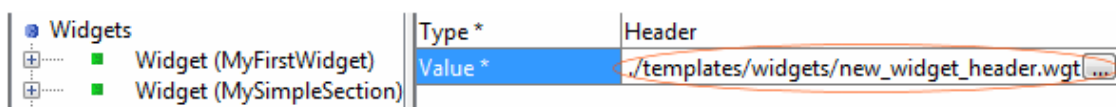
- 1 Open the Widget Editor
- 2 Right click on the **widget**
- 3 Click **Add**, click **Template**

An empty header template will be added to the widget.



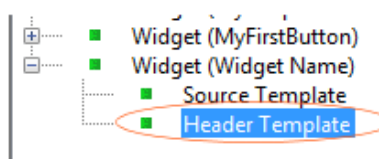
- 4 Click the arrow for the **Value** attribute
- 5 Click **Select file** if template already exists. Click **Select** to choose the file.

If the widget template file does not exist type the file name.



- 7 Click **Save**

The header template is added to the widget.



You can add multiple header templates if required.



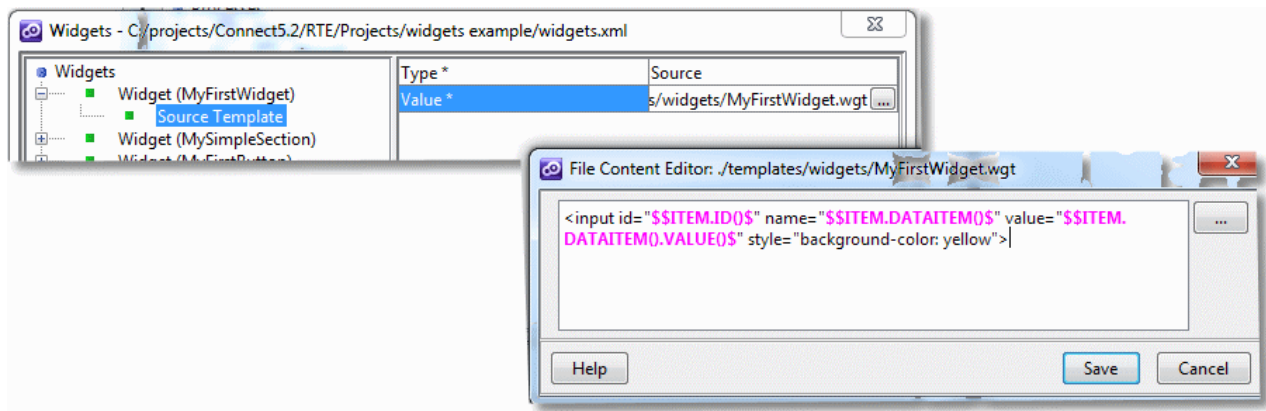
## Editing a Widget Template

The Widget Editor allows you to add or amend the content of a widget file.

### To Add Widget File Content

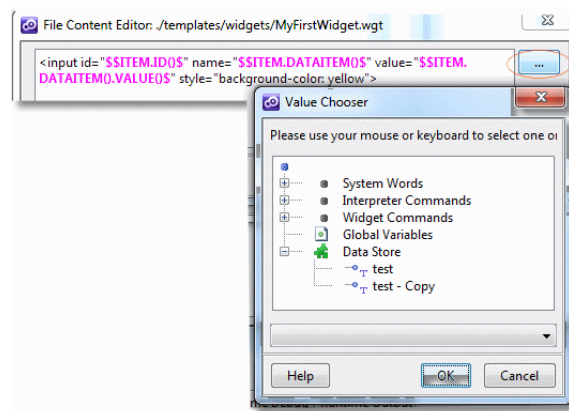
- 1 Open the **Widget Editor**
- 2 Expand the **Widget**
- 3 Click on the **Template** you wish to edit
- 4 Click on the **arrow icon** to the right of the Template value attribute
- 5 Click **Edit file**

The File Content Editor is displayed allowing you to edit the Template file.



- 5 Make any changes required

Use the elipsis button to add any System words, Widget commands, Data store names, etc to the template.



- 6 Click **Save**

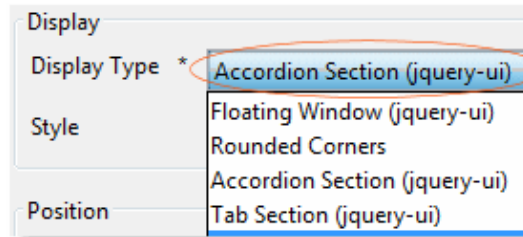
Any changes made are saved to the widgets.xml file. If you do not wish to save the changes, click Cancel.

## Customization

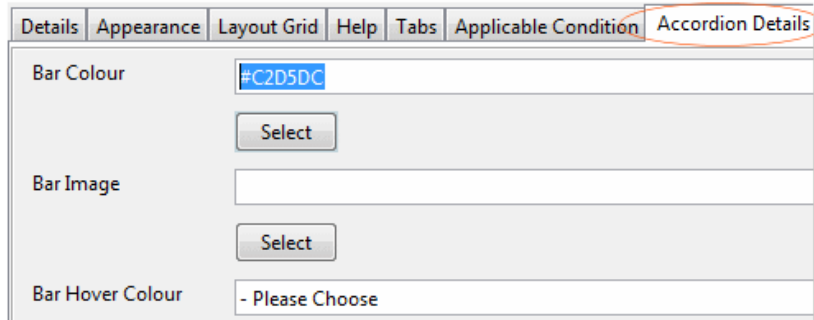
Customization allows some features of the widget to be specified by the User when they use it in their project. When the user chooses a widget display type for their element, if there is any customization available, a new tab will be displayed enabling them to customise the widget.



For example, if you choose a display type of Accordion for your section in the Presentation editor,



then a new tab called Accordion details will be displayed.



This tab allows you to specify various colours and images which are used for parts of the accordion widget. Other types of customization are available depending on what is appropriate for the type of widget you are using. You can add as many customization features as you like to the widget, but there is a payoff in the build and test time for the widget.

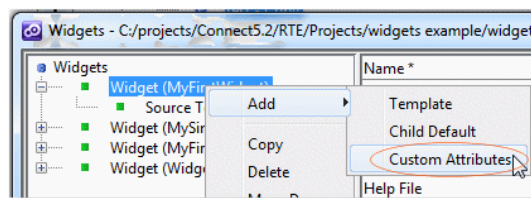
The values of the customisation attributes are updated in the templates at runtime if they are specified, otherwise default values are used.

Custom attributes are contained within <custom> tags within the widget.xml file.

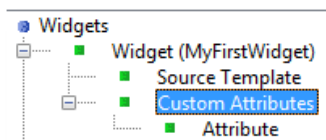
### To Add Customization Attributes

The customization attributes are used to build up a panel of questions for the user to fill in. They allow you to make features of the widget available to customize via the IDE. They also make the widget more reusable. For these values to be effective, you must then use them in the widget source template.

- 1 Open the **Widget Editor**
- 2 Right click on the **Widget**
- 3 Click **Add**, click **Custom Attributes**



A new Custom Attributes node is added to the widget structure. Any custom attributes are added to this node. An empty Attribute node is created when the Custom Attributes tab is first created.

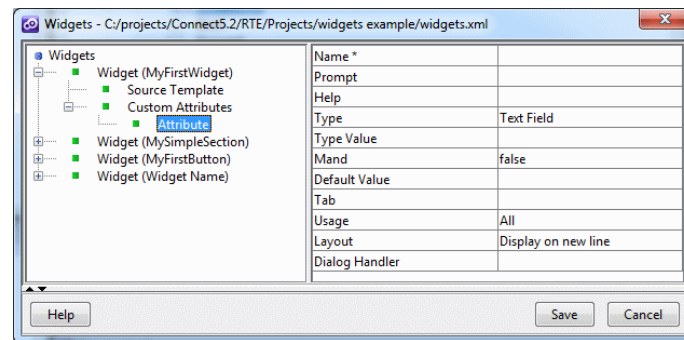






#### 4 Expand **Custom Attributes** and click on **Attribute**

A blank Custom attribute is displayed.



#### 5 Enter a value for **Name**

Name is the only mandatory attribute. It is used to refer to the attribute in the widget template.

#### 6 Enter any other attributes required

#### 7 Click **Save**

The widget.xml file is updated with the customization specified.

If you wish to specify additional customization, right click on the Customization Attribute node, click Add and click Attribute. This will add another blank attribute. As many customization attributes as required can be added in this way. These values can then be used in the widget template file to affect the widget behaviour.

Be aware that the amount of flexibility afforded by adding customization comes at the expense of development time for implementing and testing the customizable parameters for the widget.

### Custom attribute specifications

All the possible values for specifying and displaying a custom attribute within the IDE are given below.

Attribute	Description	Function	Possible Values	Mandatory
name	The name of the attribute.	Used as a reference to attribute within RTE	Alpha characters only (i.e. a-z and A-Z)	✓
prompt	The customization question text	Appears in the Widget Customization tab in the edgeConnect IDE	Any text	✓
type	The customization answer field type e.g. Colour chooser, textfield, etc	Specific input field for the customization value displayed on the customization tab in the IDE	Drop down list of possible values e.g. PropertyChooser, Textfield, ComboBox	✓
typeValue	The values associated with the answer type	Specifies the types of values which can be entered in the answer field	Possible examples: NUMERIC 2 - 2 digit textfield A B C - values for a combo box	*
mand	Is the answer required		true or false (default if not specified)	*
defaultValue	The default value for the attribute	Used at runtime if no value is specified in IDE	Any text	*
tab	The name of the tab to display the attribute on		Will default to "Widget Details" if not specified	*
usage	How the attribute should be used	Useful for language map	Drop down list of possible values	✓
layout	How the attribute should be displayed in the IDE tab		Drop down list of possible values e.g. display on new line	✓
dialogHandler	The name of any dialog handler class to be used when dialog is displayed			*





## Using Customization Values

The customization values which are entered by the user in the IDE when the solutions is built can be used in the Source template file for the widget. At runtime, the reference to the attribute is substituted for the user entered value.

### Customization Example

The basic data capture widget that we have used previously in this document has a yellow background. We could customize the background colour by allowing the user to specify it in the IDE. This is done by adding a background colour custom attribute to the widget. We use the custom attribute in the source template and the value entered by the user for this attribute is substituted at runtime.

#### To Add the Customization Attribute

- 1 Open the **Widget Editor**
- 2 Right click on MyFirstWidget
- 3 Click Add, click Custom Attributes
- 4 Expand **Custom Attributes** and click on **Attribute**
- 5 Enter the attributes as shown below

Widget (MyFirstWidget)	Name *	BACKGROUND_COLOUR
Source Template	Prompt	
Custom Attributes	Help	
Attribute (BACKGROUND_COLOUR)	Type	Text Field
Widget (MySimpleSection)	Type Value	
Widget (MyFirstButton)	Mand	false
Widget (Progress bar)	Default Value	
	Tab	
	Usage	All
	Layout	Display on new line
	Dialog Handler	

- 6 Click **Save**

#### To Use the Custom Attribute in the Source Template

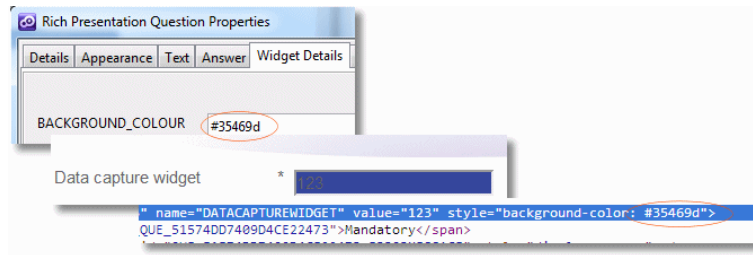
- 1 Open the **Widget Editor**
- 2 Expand the widget and click on the Source template
- 3 Click the **down arrow** on the template value attribute to edit the source template
- 4 Click **Edit File**
- 5 Replace the hardcoded background colour (yellow) with the custom attribute (ITEM.BACKGROUND\_COLOUR)

```
JEQ$" style="background-color: $$ITEM.BACKGROUND_COLOUR$">|
```

It is important to spell the attribute name in exactly the same way as it was specified in the custom attribute, otherwise it will not be recognised by the interpreter and the value will appear as blank in the generated HTML.

- 6 Click **Save** to save the attribute changes
- 7 Click **Save** to close the Widget Editor

When you run the project, you will notice that the background colour of the widget will be the one which you have specified in the IDE. This is because the value is substituted at runtime.



The above example is a simple illustration of how values can be replaced at runtime by user specified values thereby adding complexity to your widget.



## Widget Functionality

The actual Widget functionality is contained in Widget **template files**. They contain the HTML which will be included in the page when it is displayed at runtime. They may also contain some logic, specified in scripting language, or refer to script or css files. When the edgeConnect RTE encounters a widget while processing the Presentation tree at runtime, any scripting and replacement is carried out by the edgeConnect interpreter and the resulting HTML is then inserted into the page which is being generated. The edgeConnect RTE then moves onto the next item in the Presentation tree. This functionality can be used to add complexity to the basic widget.

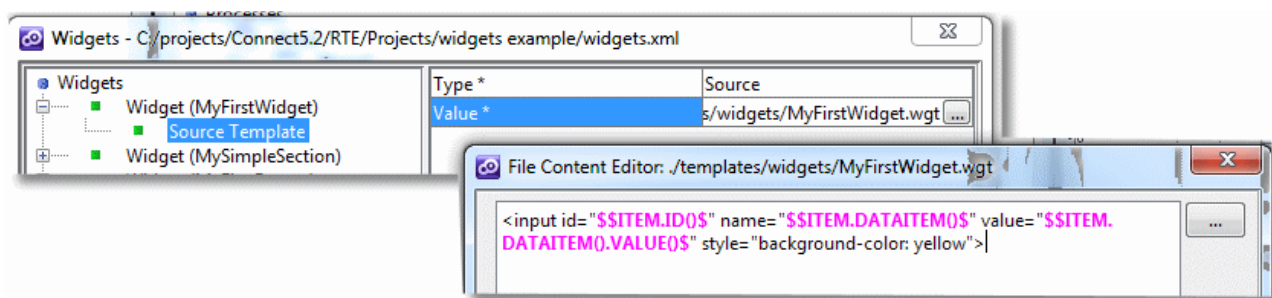
## Source Template Content

Each different type of widget has a different type of function and interacts in a different way with the page of HTML in which it exists. Therefore each different type of widget template file has a slightly different structure.

In addition, the edgeConnect RTE (Runtime environment) needs to be able to identify the widget. You can do this by giving all widgets an ID. To do this you use the keyword ITEM and the function ID(). ITEM refers to the current item in the presentation tree which is being processed. The interpreter will replace ITEM.ID() with the unique ID of the current item being processed in the Presentation tree.

## Data Capture Widget

A data capture widget exists within an <input> tag, therefore any Source template must contain this tag. An example of a basic widget is shown below.



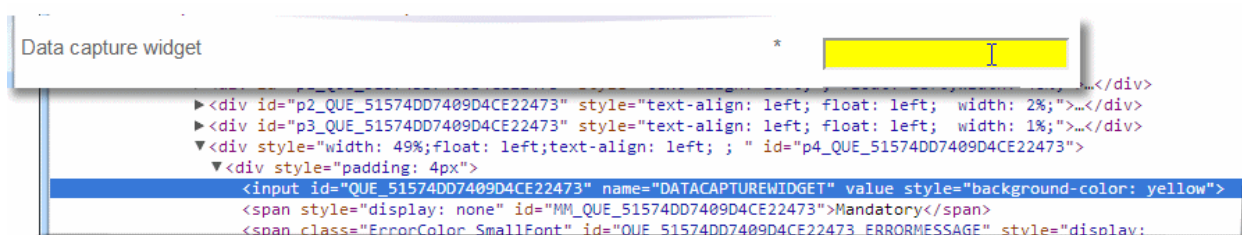
## Data capture attributes

Attribute	Description	Value
id	Unique id of the widget	ITEM.ID()
name	Name of the widget used to pass the value back to the RTE	ITEM.DATAITEM()
value	Value of the widget passed back to the RTE	ITEM.DATAITEM().VALUE()

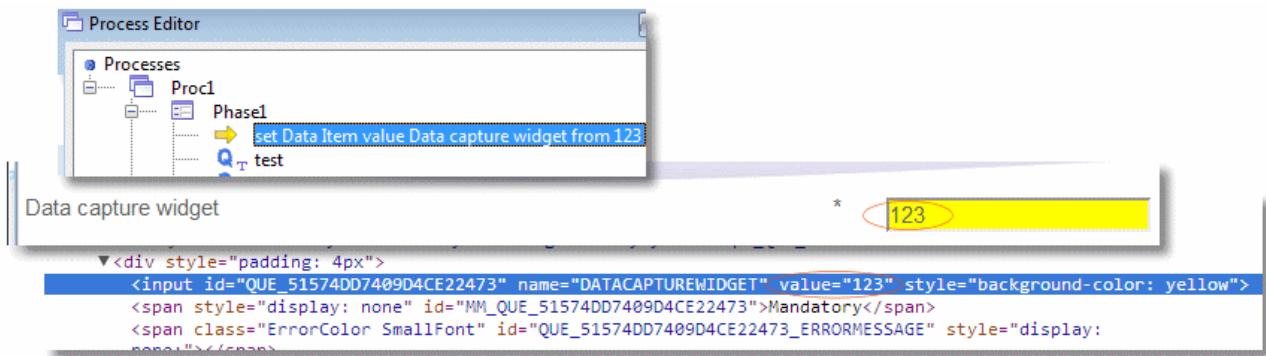
## HTML generated at runtime

The attributes of the data capture widget are used by the RTE to pass the value of the widget back to the server for further processing once the user moves off the page.

The widget generated and its corresponding HTML is shown below.



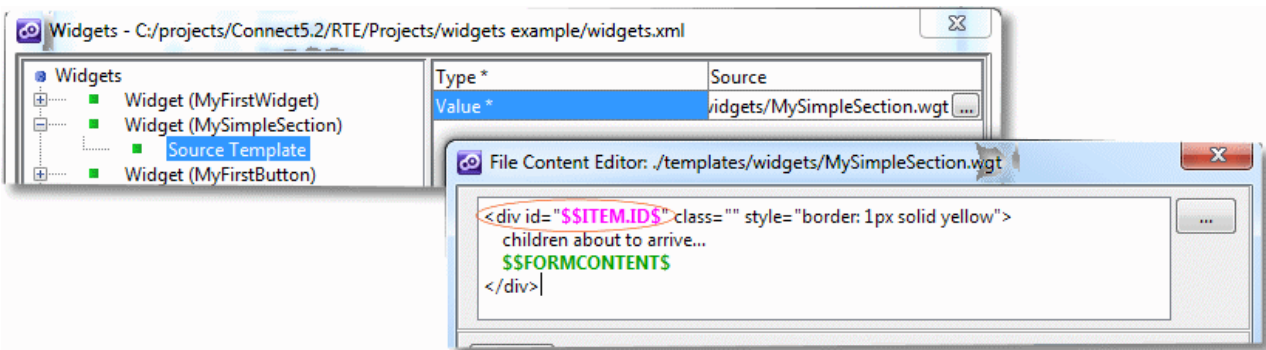
You can see that ID and name have been replaced by the corresponding run time values. The Value attribute will remain empty until a value is entered in the field by the user. However, if the field is prepopulated, for example by using a pre-phase rule, as in the example below, the value attribute is substituted and the data capture widget displays the value.



## Layout Widgets

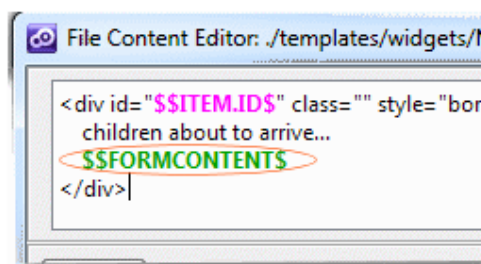
### <div> tag

Layout widgets are typically contained within a <div> tag. The value of the id attribute of the tag is \$\$ITEM.ID()\$. The interpreter will replace this value at runtime in order to uniquely identify the widget.



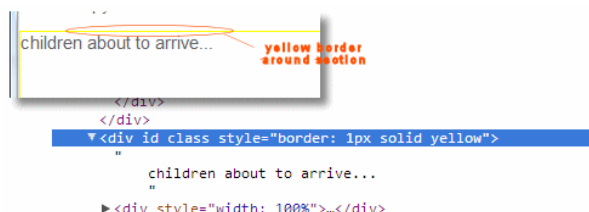
### \$\$FORMCONTENTS\$

Layout widgets act as containers for other elements in the page. Once the HTML for the widget is inserted in the page, the RTE needs to know where to insert the HTML for any children it contains. This is done by including \$\$FORMCONTENTS\$ to instruct the edgeConnect RTE to add children at this point once it has finished processing the widget.



### HTML generated at runtime

This widget will generate the following HTML at runtime which produces a section with a yellow border containing child elements below the text "children about to arrive..".





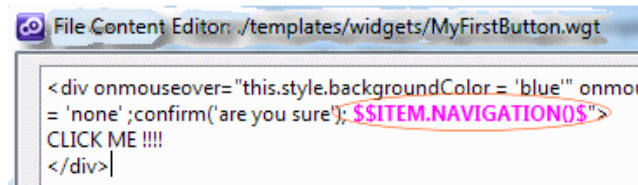
## Navigation Widget

A navigation widget enables a user to move to other parts of the solution. It may be a:

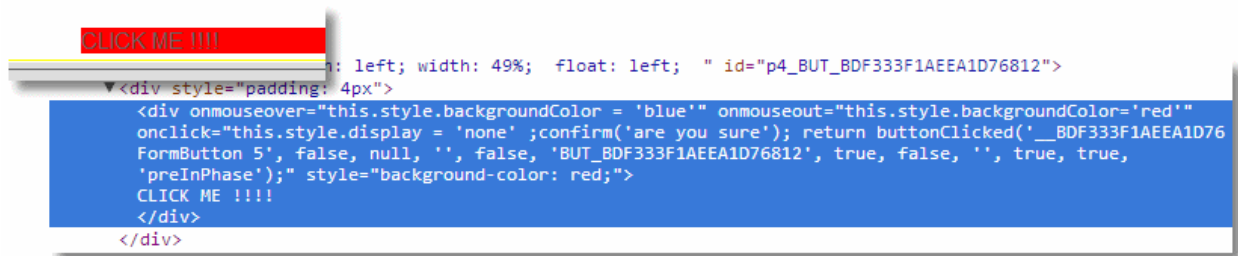
- button
- menu
- tree

### \$\$ITEM.NAVIGATION()\$\$

The \$\$ITEM.NAVIGATION()\$\$ function must be included for any navigation widget.



This will be replaced at runtime with the correct code required to handle the navigation as shown below.



## Handling Hidden Conditions

To test if a widget is hidden or not, you can use the ITEM.IS\_HIDDEN() test. This returns "true" or "false". E.g. \$\$%IF ITEM.IS\_HIDDEN() == "true"\$ the item is hidden.

It is important that any hidden data capture widgets are disabled so that the value is not sent back to the server. This will result in an error. In order to do this, you need to include the following:

```
<input ...

  $$%IF ITEM.IS_HIDDEN() == "true"$

    disabled="disabled"

  $$%ENDIF$

....>
```

## Header Template Content

You need to include any external javascript libraries/files e.g. jquery and external css files in the header template. You should also add javascript/css used by the widget. You should try to include any script required by the widget in the header file if possible as each header is only included once in the page regardless of how many times the widget is used in the page.





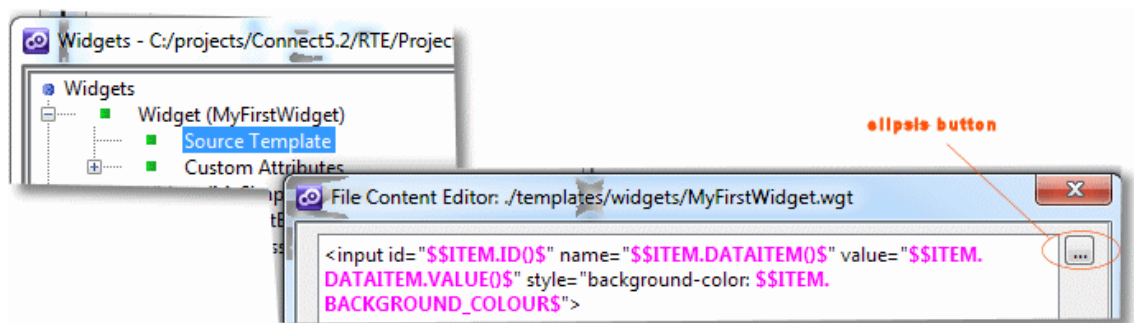
## Customization in Template files

Using replacement and scripting within the widget template file allows more powerful widget customization to be achieved. This uses runtime values to affect the way the widget works.

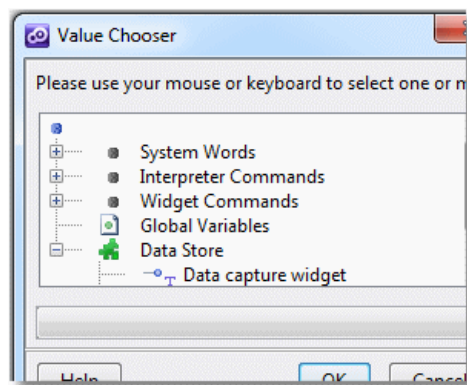
The `$$...$` syntax available within edgeConnect, which allows parameters within the dollars to be replaced with the runtime value, can be used in the template. This is known as “dollar replacement”. Customization, using user entered values, can be achieved by replacing the reference to the value at runtime with the actual value. Further complexity can be added by using scripting commands. These are built-in functions which allow logical processing to be carried out at runtime.

## Dollar Replacement

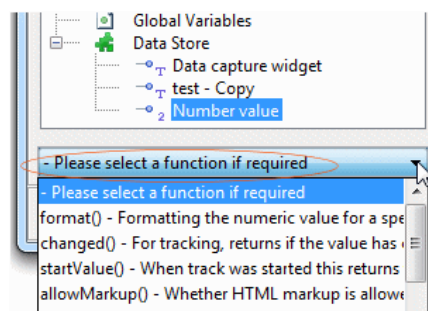
If you edit the widget source template within the Widget Editor tool, you will notice that there is an ellipsis button to the right of the edit pane.



Clicking this button will open the value chooser.



This displays all the system words which are available for you to use in the template file. These will be replaced with the actual values at runtime. If you choose a value which has additional functions associated with it, these will be displayed in the dropdown list below the options. For example, if you choose a number type data item, you can opt to format the number using the `format()` function.





## ITEM

One of the key concepts to understand when using the replacement values is that of ITEM. The ITEM keyword refers to the current item in the presentation tree which is being processed. All the attributes defined for that item are available along with attributes of the corresponding item in the Process. It is also possible to use the data item (if applicable) for questions.

The tables below show the attributes available for the ITEM keyword. These will be replaced with the current value at runtime. The attributes are divided into functional categories.

### Datastore

Attribute	Description
<b>\$\$ITEM.DATAITEM()</b> \$	The data item for a question
<b>\$\$ITEM.DATAITEM().VALUE()</b> \$	The run time value for a data item
<b>\$\$ITEM.DATAGROUP()</b> \$	The processed data group name
<b>\$\$ITEM.DATAGROUP().LASTINSTANCE()</b> \$	The last populated instance for the current item's data group.
<b>\$\$ITEM.DISPLAY_VALUE()</b> \$	Formatted version of data item value

### General

Attribute	Description
<b>\$\$ITEM.ID()</b> \$	The edgeConnect generated unique id for the item - this gets used quite a lot, and should be used in any script where a unique reference is required, e.g. a javascript variable that may be used by more than one occurrence of a widget should use this id in its name to avoid clashes.
<b>\$\$ITEM.READONLY()</b> \$	A boolean indicator identifying if the question is marked as read only. Returns true if it is read only.
<b>\$\$ITEM.READONLYSTYLE()</b> \$	The style of the read only question. Read only questions can be displayed as just text, or as a disabled component. This returns 'Text' for text and '???' for a disabled component.
<b>\$\$ITEM.STYLES().xxx</b> \$	The current style for the item. xxx must be one of the available styles for the item, and is case sensitive - e.g. <code>ErrorStyle</code> . So to get the error style, it would be <code>\$\$ITEM.STYLES().ErrorStyle</code> . Other available style names can be found in the .ect file for the project.





## Validation

Attribute	Description
\$\$ITEM.MINVALUE() \$\$ITEM.MAXVALUE()	The min/max value for a numeric/decimal item (taken from process, if data store validation has been overridden, otherwise from data item or its custom data type)
\$\$ITEM.MINLENGTH() \$\$ITEM.MAXLENGTH()	The min/max length for a text item (taken from process, if data store validation has been overridden, otherwise from data item or its custom data type)
\$\$ITEM.VALIDATION()	This is the onchange script that edgeConnect would have inserted for a standard question. Therefore widget values can be validated by edgeConnect methods
\$\$ITEM.CHECKHIDDEN()	This inserts the check hidden trigger code into the widget

## Lists

Attribute	Description
\$\$ITEM.LISTLENGTH()	The number of items in the list
\$\$ITEM.DATAITEM().LISTITEM().VALUE()	When iterating through the list of items, returns the display value for the current list item
\$\$ITEM.DATAITEM().LISTITEM().KEY()	When iterating through the list of items, returns the key for the current list item
\$\$ITEM.DATAITEM().LISTITEM().GROUPVALUE()	When iterating through the list of items, returns the group value for the current list item
\$\$ITEM.DATAITEM().LISTITEM().SELECTEDINDEX()	The integer index of the currently selected value from the list

## Tables

Attribute	Description
\$\$ITEM.NUMBER_OF_ROWS()	If the current item is a table, this returns the total number of rows in the table
\$\$ITEM.NUMBER_OF_CHILDREN()	If the current item is a table, this returns the number of child items within the table.
\$\$TABLE_CHILD.xxx\$	An attribute from a child item within a table. This is used in conjunction with the FOREACH syntax - see section 3.
\$\$TABLE_CHILD.IS_LAST()	A Boolean to indicate if the child item is the last child in the table. Returns "Y" if child is last. Used in conjunction with FOREACH syntax.

## Standard Common Attributes

Note that there are no brackets after the attributes.

Attribute	Description
\$\$ITEM.QuestionText\$	The question text for a question
\$\$ITEM.TabName\$	The name of the tab - this is the text that is displayed in tab

Note that it is possible to access any standard edgeConnect attribute if you use the syntax ITEM.AttrName. This is case sensitive. Attribute names can be found in the .jpf file.





## Custom Replacement

If the standard replacement syntax is not sufficient, edgeConnect will look to see if you have provided your own custom replacement class. This will only be called if all the above checks have not produced an output value.

In order to write a custom replace class, it must adhere to the following:

- the class must be called ItemReplacer
- the class must be in the package com.edgeipk
- the class must implement the interface com.acquire.intelligentforms.presentation.interpreter.ItemReplacer
- the interface has one method, which obviously must be declared, called replace, which takes the arguments FormSession (the session info) and String (the input String), and must return a String (the result of the replacement)

## Styles in Templates

You can define styles that your widget will use within the template. The advantage of this is that you can make these styles dynamic by incorporating customization attributes which the user has specified using the IDE. Any styles defined must be contained within style tags as shown below.

```
<style type="text/css">

    styles definitions

</style>
```

Styles are written in CSS. You can use them to overwrite styles that are provided with a third party widget. The example below is from the Slider widget supplied with edgeConnect. It overrides styles that are supplied with the jquery-ui slider widget which is used. You can see that scripting is also contained within the style definitions to use a slider image if one has been specified in the IDE.

```
<style type="text/css">
    #$$ITEM.ID()$.ui-widget {font-family: inherit;}

</style>
```

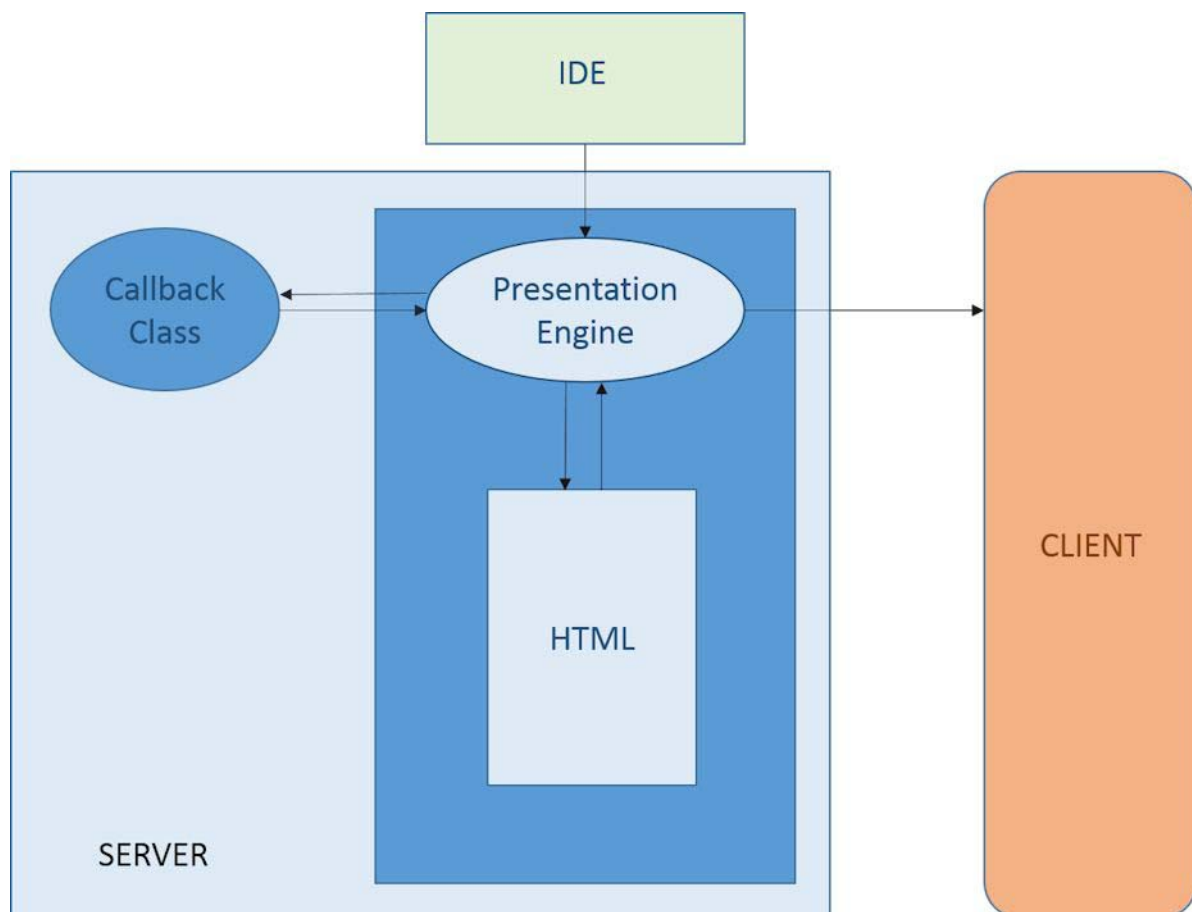


## Callback Widgets

A callback widget is one which exhibits callback behaviour. This enables the widget to alter the HTML content, generated by the Presentation Engine, on the server. This allows you to alter the way the widget behaves by making changes to the standard HTML generated for the widget. Any widget type, i.e. data input, layout or navigation, can be a callback widget. Setting the callback attribute to true will enable the widget to exhibit callback behaviour.

### Callback Control Flow

The callback widget is specified as the display type for the element and its characteristics are defined in the IDE. When this is passed to the Presentation Engine, the HTML for the widget is generated. If a callback widget is encountered as the page is generated, control will immediately pass to the callback class for the widget which will inject the required HTML into the page at the point the widget appears.



Once the callback class is completed, control is passed back to the Presentation Engine and the rest of the HTML for the page is generated.

### Callback Class

Every callback widget must have a corresponding callback class. This will specify what HTML will be generated for the widget. A callback widget must:

- extend the interface `IWidgetCallback`
- implement the method `updateContent`

The class needs to exist somewhere on the classpath.



The update Content method contains the business logic for the widget. It is passed the following parameters:

- session
- presentation entity e.g. button, textfield, section, etc.
- result - the HTML page generated so far
- startLoc - the start point for the entity in the page

When you define a callback widget it only has one Template file. The name of the template file is the name of the callback class. The type attribute is not used, so can be excluded or defined as source or header.

### Example

This example changes the generated button html to use “onmousedown” rather than the generated “onclick” event. It will replace all instances in the html generated for the button.

```
package com.edgeipk.widgets.callbacks.buttons;

import com.acquire.intelligentforms.FormSession;

import com.acquire.intelligentforms.presentation.engine.elements.IWidgetCallBack;

import com.acquire.intelligentforms.presentation.util.StringBufferAdapter;

import com.acquire.util.StringUtils;

public class MouseDownButton implements IWidgetCallBack
{
    public void updateContent(FormSession p_formSession,
                             PresentationObject p_presObject,
                             StringBuilderAdapter p_result,
                             int p_startLocation)
    {
        while (p_result.indexOf("onclick", p_startLoc) > 0)
        {
            int start = p_result.indexOf("onclick", p_startLoc);
            p_result.replace(start, start + "onclick".length(), "onmousedown");
            p_startLoc = start + 1;
        }
    }
}
```

The xml for this example would appear in widgets.xml as shown below:

```
<widget name="MouseDownButton" type="Button" callback="true">
    <template value="com.edgeipk.widgets.callbacks.buttons"/>
</widget>
```



## Market Place Widgets

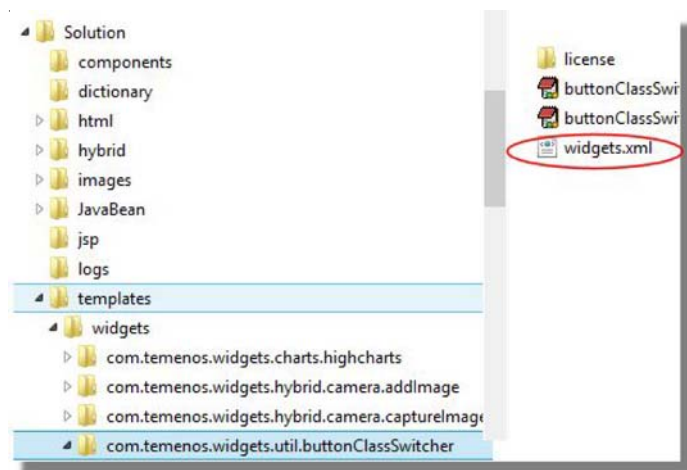
### What is the Market Place?

The Market Place is a Temenos portal for sharing widgets, components, etc. You need to be a Temenos partner to gain access to the Market Place.

### Market Place Widget Structure

In order for widgets to be available in the Market Place, their structure has been altered slightly to simplify sharing. The main change is that the widgets.xml file is contained within the widget itself i.e. each widget has its own widgets.xml file.

The example below shows a Market Place widget which has been included in a project. Note the widgets.xml file contained within the folder.



### Adding a Widget to the Market Place

Once you have built and tested your widget, you will need to zip it up in order to add it to the Market Place. Requests to publish a widget on the Market Place will need to go through Temenos marketing.

### Using a Market Place Widget

Market Place widgets will be available from the Temenos website if you are a subscriber. You will not need a licence to obtain a widget. If a licence is required for the widget itself, it will be contained within the widget and no extra steps will be needed to obtain it. If you have subscribed, Market Place widgets will be available free of charge.

When you download a zipped widget from the market place, with the IDE running, it will automatically be copied to the project, extracted and made available for use straight away.

You can have more than one widget in a MarketPlace widget zip file, for example, there are several facebook features in the Facebook widget zip, but it contains one widgets.xml.



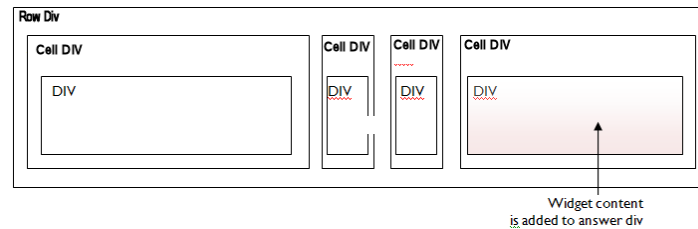
## Widget Layout at Runtime

This section describes how widgets interact with the layout grid. The layout grid is described in more detail in the Presentation Guide.

### Data Capture Widgets

These types of widgets are used as answers for questions.

The example below shows a standard question, with the question text cell (with a div inside), the mandatory cell (with a div inside), a prefix cell (with a div inside) and an answer cell (with an answer div inside).



If two answers are grouped together, the answers are in two separate divs inside a fieldset. The widget content is still placed within the answer div.

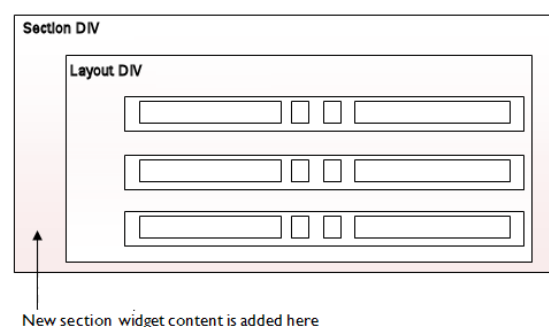
### Layout Widgets

These are slightly more complex, in terms of layout, since Sections, Columns and Tabs all do slightly different things.

- A Section is meant to completely decouple one section from another – there should be no interaction by changing contents in one Section with the layout in another.
- Columns sit within Sections (NB all phases are a Section by default), and are essentially new cells within one of the tables within the Section.
- Tabs create a tab header (the part you click) and a tab pane.

### Sections

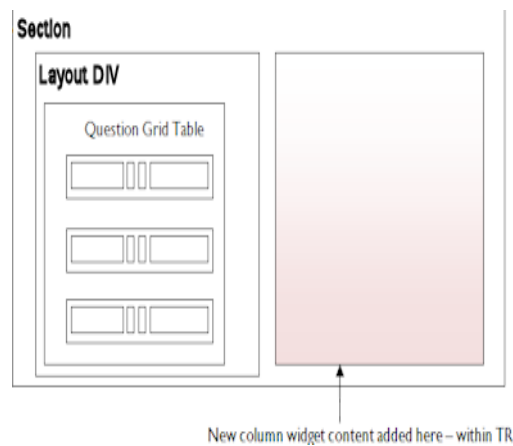
There are two DIVs for a section as shown below.



Note that when a widget has been added, all content is added directly to it.



## Columns



## Tab Panes

These are created in the same location as a new Section.



## Example - adding a menu widget

The example we are using is a menu widget for a drop down horizontal menu as shown below.



A menu is a more complex example as it consists of three different parts:

- the menu itself - contains all the other parts
- menu branches - in this case “for him”, “for her”, etc. There may be more than one level of branches.
- menu leaf - the lowest level of the menu hierarchy e.g. towels as shown above

You will need to add a widget for each different part of the menu.

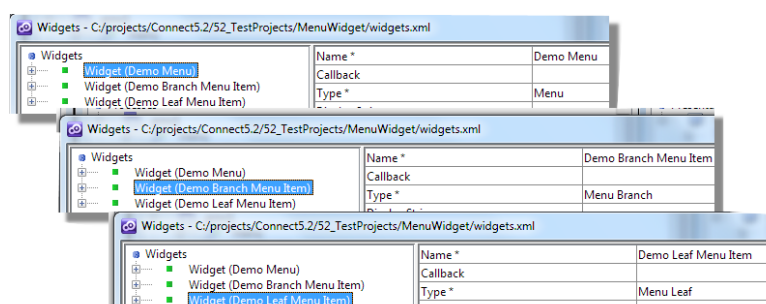
The steps for adding and setting up the widget are as follows:

- Add each Menu type widget to the project
  - Add the widget
  - Create the source template
  - Add any header templates
  - Add any customization attributes
- Use the widget in a project
  - Specify any customization attributes
- Add the widget code to the source template
- Incorporate any customization to the source template

### To Add Menu widget to the Project

- 1 Open the **Widget Editor**
- 2 Right click on the **Widget** root
- 3 Click **Add**, click **Widget**
- 4 Enter **Name** and **Type**

The name and type for each part of the menu is shown



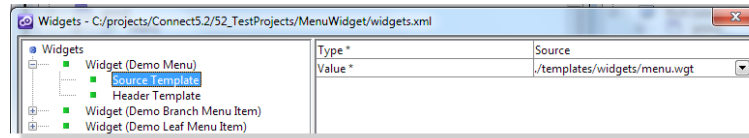


## 5 Click **Source Template**

## 6 Click **down arrow** on value attribute

Navigate to widget source file location and select file or type in the name of a new file for the menu container

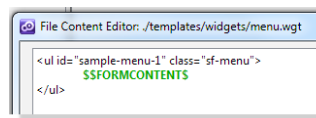
## 7 Click **Select**



In our example, the file name is menu.wgt.

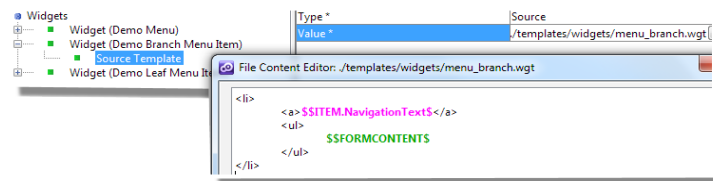
## 8 Click **down arrow** on value attribute, click **Edit File**

Edit the source template to add the following content. This creates the top level menu container and any further content will continue to be added to the page at the `$$FORMCONTENTS` tag.

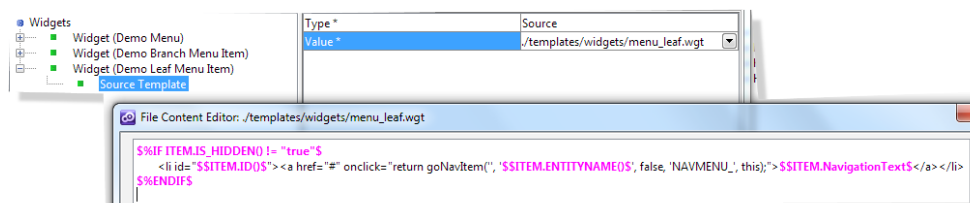


## 9 Click **Save**

## 10 Repeat steps 5 - 9 for the branch and leaf menu widgets. The content to be added for the source file for the menu branch and leaf are shown below.



Note that the leaf item must not be shown if it is hidden, therefore a conditional must be added to handle this.

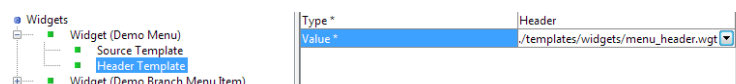


The widget processing is dependent on code supplied by jquery UI. This must be included in the header part of the page and is therefore added as header templates.

## 11 Right click the **widget**, select **Add**, click **Template**

## 12 Right click the **down arrow** for value attribute, click **Select file**

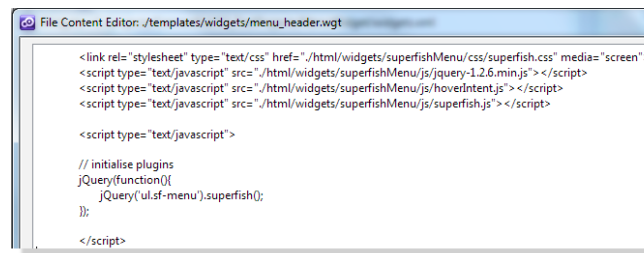
Select the header file required for the Menu widget in their location on your machine or enter the name of a new header file. You only need add a header file for the menu container widget as the other two will use the same files.







The header template content should reference any javascript or css files that the menu widget needs to function.

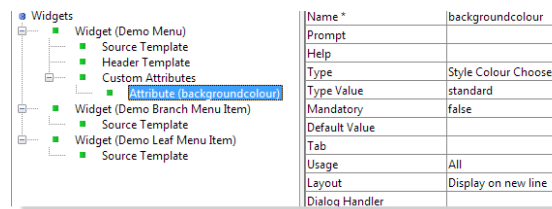


If you would like to allow the user to customise the widget using the IDE, you can add custom attributes at this point. In this example we are going to allow the user to customise the background colour of the menu.

13 Right click the **widget**, select **Add**, click **Custom Attributes**

14 Expand **Custom Attributes**, click **Attribute**

Fill in the custom attribute as shown below.



15 Click **Save**

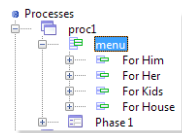
The Menu widget has now been added to the project and you are able to use it within your solution.

### To Use the Menu widgets in a Project

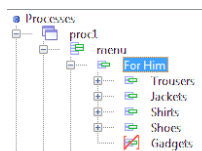
The Menu widgets apply to menu, branch and leaf menu types respectively, therefore you need to add a menu to your project where you would like it to appear. The menu example is a simple menu for clothes and home items. Once the menu is set up, you will then be able to change the display type of each menu constituent to the appropriate widget type in the Presentation editor. The display types are set at the menu level so that you do not need to set a display type for each menu item. This makes it much more efficient to set up and maintain.

1 Add a **menu** to the Process. This will contain all the other menu items.

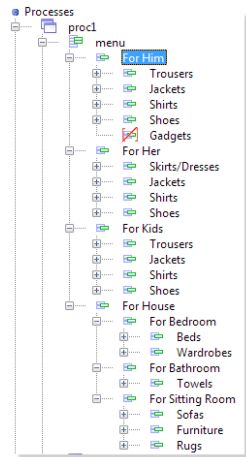
2 Add a **menu item** to the menu for each branch of the menu - For Him, For Her, For Kids, For House



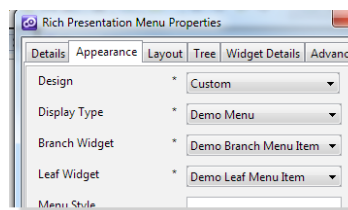
3 Add a **menu item** to the For Him branch for each leaf item - Trousers, Jackets, Shirts, Shoes, Gadgets



- Repeat Step 3 until all the remaining menu items have been added.

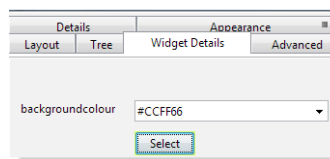


- Open the **Presentation Editor**
- Double click on the **Menu**
- Click **Appearance** tab, select **Display Type**, **Branch Widget** and **Leaf Widget** to be the widgets you would like to use for each part of the menu



Note that you can only select the menu display types here.

- Click **Widget Details** tab
- Enter a background colour if required

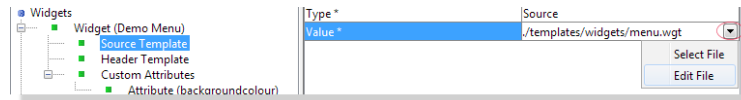


- Click **OK**

### *To Add the Customization to the Source Template*

In this case, the customization allows the user to specify the colour of the menu. The colour is specified by changing the style applied to the menu item in the case of this widget. Therefore in order to use the customization value entered by the user, we must use it as a variable for the style. The style is specified in the source template.

- Open the **Widget Editor**
- Click on the **Source** template of the widget you wish to customise
- Click the **arrow** to the right of the template file name, select **Edit File**



#### 4 Edit the template to add the code shown below.

Alternatively, you can edit the template file in your preferred text editor.

```

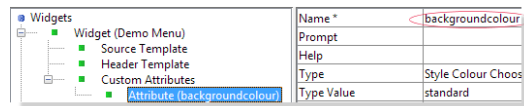
%%IF ITEM.backgroundcolour == ""$
%%SET backgroundColor = transparent$ %%ENDSET$
%%ELSE$
%%SET backgroundColor := ITEM.backgroundcolour$ %%ENDSET$
%%ENDIF$

<style type="text/css">
  $$ITEM.ID()$.sf-menu a {
    background-color: $$!backgroundColor$;
  }
</style>

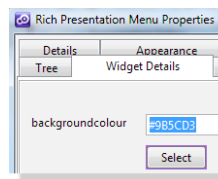
<ul id="sample-menu-1" class="sf-menu">
  $$FORMCONTENT$
</ul>

```

- "backgroundcolour" is the name of the custom attribute for the widget



- The background colour is set in the IDE by the user



- A variable is used for the background colour and this is set if the background colour has been specified, otherwise it is transparent. This is not essential, but makes the style setting less cluttered.

```

%%IF ITEM.backgroundcolour == ""$
%%SET backgroundColor = transparent$ %%ENDSET$
%%ELSE$
%%SET backgroundColor := ITEM.backgroundcolour$ %%ENDSET$
%%ENDIF$

```

ITEM.backgroundColor is the value of the colour set by the user for this widget.

- The colour set is used in the style for the menu item.

```

<style type="text/css">
  $$ITEM.ID()$.sf-menu a {
    background-color: $$!backgroundColor$;
  }
</style>

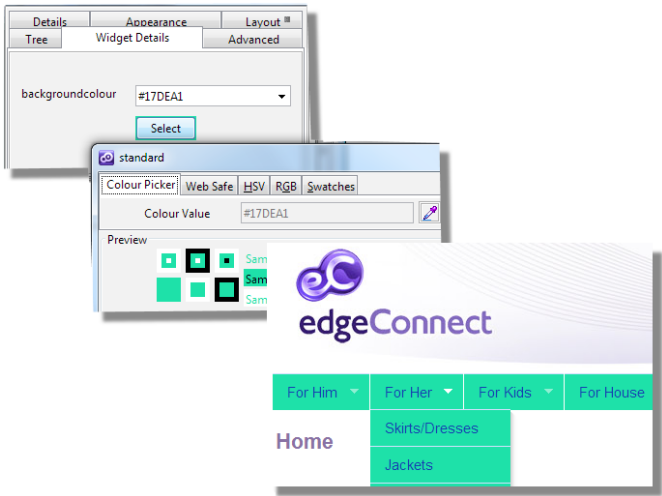
```

- Any styles must be contained within the <style> tag.

#### 5 Click **Save** to save the code changes

#### 6 Click **Save** to close the Widget Editor

An example of the effect of customisation on the menu is shown below when the user chooses a green colour for the menu.





**TEMENOS**  
The Banking Software Company

Temenos Headquarters SA  
18 Place des Philosophes  
CH-1205 Geneva  
Switzerland  
Tel: +41 22 708 1150  
Fax: +41 22 708 1160  
**[www.temenos.com](http://www.temenos.com)**

TEMENOS™, TEMENOS T™, TEMENOS T24™ and T™ are registered trademarks of Temenos Headquarters SA  
The Triple'A Plus™ and WealthManager™ trademarks belong to the Temenos Group of Companies

Information in this document is subject to change without notice.

No part of this document may be reproduced or transmitted in any form or by any means,  
for any purpose, without the express written permission of TEMENOS HEADQUARTERS SA.

© 2010 Temenos Headquarters SA - all rights reserved.