Advanced Grid Widgets Extension

Version 4.2.1

July 05, 2019

Contents

Document Revision History

Revision Date	Description of Change
March 22, 2017	Initial version of document.
April 13, 2017	Fixed incorrect links to specific sections of the document. Added detail to the Samples File section.
May 17, 2017	Updated to include 2.0 functionality and bug fixes.
June 2, 2017	Updated to include know issues.
July 31, 2017	Updated Release Notes section and added more information about the available column renderers.
October 4, 2017	Updated to include 3.0 functionality and bug fixes.
October 13, 2017	Updated to include 3.0.2 functionality and bug fixes.
October 31, 2017	Updated to include new Location column renderer and 3.0.3 bug fixes.
November 20, 2017	Updated to include 3.0.4 functionality and bug fixes.
December 5, 2017	Updated the Prerequisites to require at least ThingWorx 7.4
January 5, 2018	Updated to include the 3.0.5 functionality. 1. Revised the Prerequisites table. 2. Removed the example entities from the document. 3. Revised the Sample File section. 4. Revised the bug fixes section. 5. Revised the MinRowHeight Property Description. 6. Revised the Cell Editing Options section.
March 2, 2018	Updated to include 3.0.6 functionality and bug fixes.
June 06, 2018	Updated to include 4.0.0 functionality and bug fixes.
July 06, 2018	Updated to include 4.0.1 bug fixes.
August 31, 2018	Updated to include 4.0.2 bug fix.
November 14, 2018	Updated to include 4.1.0 new features and bug fixes.
November 16, 2018	Updated to include 4.1.1 bug fix.
January 31, 2019	Updated to include 4.2.0 new features and bug fixes.
July 05, 2019	Updated to include 4.2.1 enhancement and bug fixes. Added a note in the description of **IDFieldName** property based on TW-58681. Added a note in the **Column Renderers and Formats** section to support rendering with dynamic data shapes using dynamic configuration.

Software Change Log

Version	Release Date	Changes
1.0	-	Beta release.
1.1	March 22, 2017	General release.
2.0	May 17, 2017	General release.
2.1	July 31, 2017	General release.
3.0	September 29, 2017	General release.
3.0.5	January 5, 2018	General release.
3.0.6	March 2, 2018	General release.
4.0.0	June 06, 2018	General release

Version	Release Date	Changes
4.0.1	July 06, 2018	General release
4.0.2	August 31, 2018	General release
4.1.0	November 14, 2018	General release
4.1.1	November 16, 2018	General release
4.2.0	January 31, 2019	General release
4.2.1	July 05, 2019	General release

Prerequisites

Prerequisites
ThingWorx 8.2.x, 8.3.x or higher

Introduction

The Advanced Grid Extension includes two widgets: Advanced Grid and Advanced Tree Grid.

Both widgets provide flexible, interactive ways to display data in grid views. Each widget supports numerous ways to render column data and allows on-the-fly configuration of the data display.

How are the Advanced Grid and Advanced Tree Grid Widgets Different from the Standard Grid?

Both advanced grid widgets provide options to allow fully dynamic grid configuration. When a grid is configured dynamically, via a ThingWorx service, the grid can be built without dependence on a Data Shape. Both grid widgets also include an enhanced user experience that makes grid data easier to work with, both in Design Time and in Run Time environments.

In addition, the Advanced Tree Grid is designed to handle hierarchical data and can provide expandable nodes that display parent and child data relationships in a tree structure.

NOTE: The Advanced Grid and the Advanced Tree Grid widgets are not backwards compatible with the standard grid widget. These advanced grids are alternatives to the standard grid. They include many new advanced features but are not a one-to-one replacement of every feature available in the standard grid (see below for details). There is no upgrade path from the standard grid to one of the advanced grids.

The following subsections list the key features that are provided in both advanced grids, features that are unique to the advanced tree grid, and features that were available in the standard grid that are not currently included in the advanced grids.

Key Features in Both Advanced Grids

- Options for building grids using either a static or a dynamic configuration:
    - Static – Use the properties available in the Mashup Builder to configure the grid.
    - Dynamic – Bind the grid to a configuration service that returns a JSON object with the configuration parameters.
- Enhancements related to dynamic grid configuration:
    - Not limited by dependence on an underlying Data Shape because grid configuration parameters are passed in dynamically from a configuration service
    - More control over certain style properties, such as font settings
- Changes to grid configuration in both Mashup Builder and with a service:
    - Real-time data updates in Design View (design changes reflected on-the-fly in the data)
    - Subset of most useful column renderers available, including Boolean, Datetime, Html, Hyperlink, Imagelink, Integer, Location, Long, Number, and String

- Sorting on multiple columns
- Multiple-row selection options
- Grid Reset button
- Global grid Search field
- Auto-width column sizing and fixed-width column sizing (in pixels or percentages)
- Header and cell text alignment
- Toolbar and Tooltip styling options
- Overflow options and tooltip support for header and data cells

- Data Filter widget enhancements:
  - Live data filtering on all data types – data in the grid updates to reflect filtering
  - OR queries (in addition to the standard AND queries)
  - Data filtering can be combined with search and sort parameters
- Context menu in Run Time where columns can be hidden or unhidden from the column headers
- Server-side sorting and search functionality that will sort or search on all data rather than just the data currently loaded in the grid
- Per user/per grid cookie to persist display settings such as hidden columns, column order across the grid, column size, column sort order (ordering of rows), and row expansion in tree grids.
- Support for rendering images in a grid cell.
- Localization Support for column headers in both JSON and Mashup Builder properties (Depends on specific ThingWorx point releases. See Prerequisites.
- Addition of a footer section in a grid.

Features Unique to the Tree Grid

- Expandable nodes for viewing multiple levels of parent/child data
- Separate options to preload initial data and dynamically load child data
- Javascript tree-loading data service that provides search and filter functionality for parent and child data once the source of the data is defined
- Auto-expanding rows as defined from a service by specifying the ID of any row to be expanded

Features of the Standard Grid Not Currently Available in Advanced Grids

- Scroll to the top
- Cell editing

Cell editing is available for all column formats currently supported in the Advanced Grid. You can also edit Boolean checkboxes at runtime.

- Cell validation

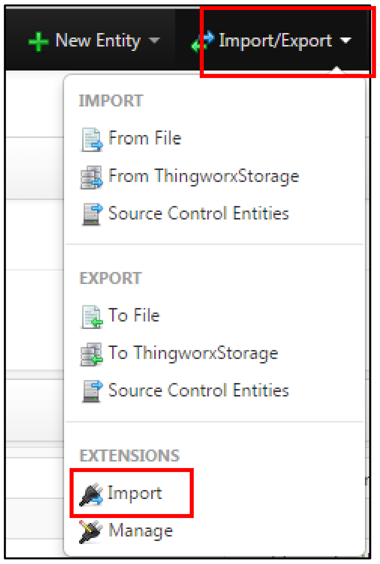
  Cell validation includes validation expressions and validation messages. Invalid values for a specific column type are not accepted by a grid; therefore, the existing valid value is kept.

- Support for all column renderers

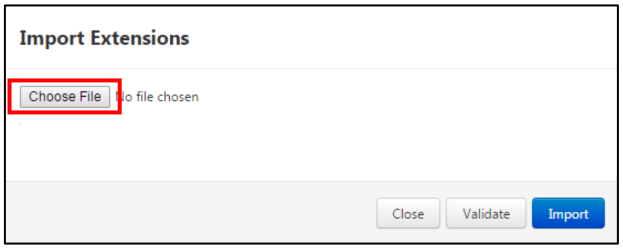
  The set of renderers currently supported in the advanced grids is limited to the following: STRING, NUMBER, LONG, LOCATION, BOOLEAN, HTML, HYPERLINK, IMAGELINK, and DEFAULT

Installing the Widgets

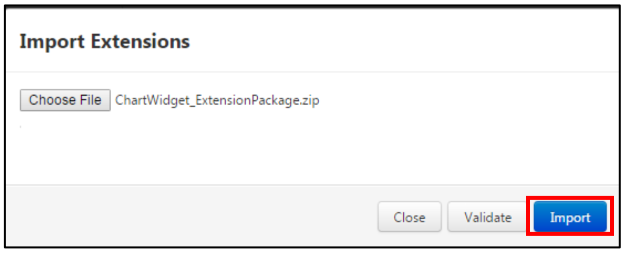
1. From a Web browser, launch ThingWorx.
2. Log into ThingWorx as an administrator.
3. Go to **Import/Export > Import**.



4. Click **Choose File** and select the **grid-advanced_ExtensionPackage.zip** from wherever you have saved it.



5. Click **Import**.
   NOTE: If an **Import Successful** message does not display, contact your ThingWorx System Administrator.



6. Click **Yes** to refresh Composer after importing the extension.

Building an Advanced or Tree Grids

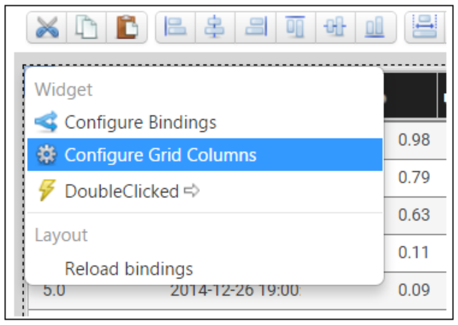
As with any data-rendering widget in ThingWorx Composer, a grid widget must be placed in a mashup and configured with incoming data bindings. To build an Advanced Grid or an Advanced Tree Grid, do the following:

1. Drag and drop one of the following widgets onto a mashup:
   - **Grid-Advanced**
   - **Tree-Grid-Advanced**
2. On the right, add a data source entity and, from the **Returned Data**, drag **All Data** to the grid and bind it to the **Data** property. This binding defines where the data is loaded from, when the grid is launched.

   Tree Grid: If you are building a Tree Grid, you can also bind a source for the child data. From **Returned Data**, either in the same data source entity or from a different source, drag **All Data** to the **Child Data** property on the grid. This binding defines where child data comes from when subsequent nodes are expanded and the child data is loaded dynamically.

   NOTE: Binding child data in a tree grid requires a specific kind of data service that provides the code necessary to properly sort, search, filter, and expand nodes. For more information, see Using a Tree-Loading Data Service.
3. Define the grid configuration parameters using one of the following methods:
   - Static Configuration – Use the list of properties available in the Mashup Builder to configure grid parameters. The available properties are listed in the left-side panel of the Design view. For information, see Properties below.
   - Dynamic Configuration – Write a configuration service that outputs a JSON object and bind it to the grid. For information and a sample script, see Working with a Configuration Service.

   NOTE: If the data source is tied to a Data Shape, you can also configure some grid parameters from the context menu available in the top left corner of the widget in Design view. For more information, see Column Configuration from the Context Menu.



4. Save and View the completed mashup.

Properties

The advanced and tree grid properties available in the Mashup Builder Design view can vary depending on whether you are configuring the grid via the Mashup Builder (static configuration) or via a service (dynamic configuration). The chart below lists all the properties available when configuring the grid from the Mashup Builder.

Properties that are only configurable from the Mashup Builder, and not via dynamic configuration, are marked with an asterisk * in the chart.

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
Id*	A unique identifier used internally by ThingWorx.	INTEGER	Gridadvanced-<id> or Treegridadvanced-<id>	N	Both
Type*	The widget type.	n/a	Grid-Advanced or Tree-Grid-Advanced	N	Both
DisplayName*	A user-defined name to identify the grid when displayed.	STRING	gridadvanced-n or treegridadvanced-n	N	Both
Description*	A user-defined description.	STRING	n/a	N	Both
Data*	Source of data that loads when the grid is launched. If the grid is bound to a data source, a filled arrow is displayed: 🡄 If there is no data source, the arrow is unfilled: ⇦	INFOTABLE	n/a	Y	Both
ChildData*	Source of child data that loads dynamically when nodes are expanded. If the grid is bound to a child data source, a filled arrow is displayed: 🡄 If there is no child data source, the arrow is unfilled: ⇦	INFOTABLE	n/a	Y	TreeGrid only

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
ParentIDFieldName	Identifies the parent ID Field Name. This property is required to create the hierarchical tree structure. IMPORTANT: For the top-level row that does not have a parent, the value should be a forward slash (/).	STRING	parentId	N	TreeGrid only
IDFieldName	The primary key column for the grid. The values in this column act as unique identifiers for each row of data. This property is optional for the advanced grid but required for the tree grid. When no field is specified, or if the specified field does not exist, the grid creates its own internal row ID. NOTE: Row id indexes are required to start with index value of 1. An error is displayed in the JavaScript console when a row with id of “0” (zero) is encountered and that row is not displayed in the grid.	STRING	id	N	Both
IDPathSeparator	Enables configuring the path separator character. The path separator character is used in Tree Grid for selections of rows that are dynamically loaded by the grid but have not yet been loaded on the client side. See section Row Selection of Dynamically Loaded Rows .	STRING	:;	N	TreeGrid only

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
HasChildrenFieldName	Specifies the name of the column that indicates whether a row has child data available. To indicate that the row does NOT have children, enter one of the following: ‘0’, 0, ‘false’, false, empty string, or undefined. Any other value means that the row does have children.	STRING	hasChildren	N	TreeGrid only
Configuration*	If the grid is bound to a configuration service, a filled arrow is displayed: ⬅ If there is no configuration service, the arrow is unfilled: ⇦ NOTE: An **Add** button is available for the **Configuration** property from 4.2.0 version of Grid, that enables you to enter a JSON in the pop-up page. The **Add** button becomes **Edit** when you enter the JSON, click **Done**, and go back to the property panel.	STRING	n/a	Y	Both
IsEditable	Determines whether or not the values in grid cells can be edited when the grid is displayed in run time. NOTE: To edit values in a specific column, the column must also be configured as editable. See the **Cell Editing Options** on the **Column Configuration** menu.	BOOLEAN	False	N	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
EnableEditButtons	Enables the **Edit**, **Save**, and **Cancel** buttons in the grid tool bar to allow for manual saving of edits. If you want changes to be saved automatically, enable the **IsEditable** property described above. If you want to allow for changes to be saved manually, enable this **EnableEditButtons** property. See the Cell Editing for more information. When you enable this property, **EnableAddDeleteButtons** property appears in the mashup properties panel.	BOOLEAN	False	N	Both
EnableAddDeleteButtons	If you set this property to True, the **Add** and **Delete** buttons appear in the grid tool bar that enables you to add or delete rows in the grid. Note: Click the **Edit** button at run time to view either the **Add** or **Delete** buttons in the tool bar.	BOOLEAN	False	N	Advanced Grid only
EditedTable	A bindable property that specifies an output location for updated values when cells are edited at run time. This property must be bound to an infotable update service to save the updated values. For example, bind to **AddOrUpdateDataTableEntries** service on a DataTable thing. NOTE: Before **EditedTable** property can be used, the **IsEditable** property must be	INFOTABLE	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	enabled. In addition, specific columns must be configured as editable. See the Cell Editing Options on the **Column Configuration** menu.				
DeletedTable	This property specifies an output location for rows that are deleted from a grid at run time. This property must be bound to an infotable update service to save the updated values. For example, bind the property to the **AddOrUpdateDataTableEntries** service on a DataTable thing.	INFOTABLE	n/a	Y	Advanced Grid only
DefaultSelectedRows	Defines which row numbers are highlighted by default when the grid is displayed. Values can include comma-separated numbers and ranges. **Example**: 1,2,4-5 The property can also be defined by a bound service. If service is bound, a filled arrow is displayed: 🡄 If there is no service, the arrow is unfilled: ⇦ NOTES: This property will have no effect if the **RowSelection** property is set to **none**. In order to select multiple rows, the **RowSelection** property must to be set to **multi**.	STRING	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	In a tree grid, default row selection depends on which rows are in view. When the **ExpandLoadedRows** property is enabled, all of the preloaded rows are expanded and the default selection starts at the top and counts down including both parent and child rows. If the preloaded rows are not expanded, the default selection starts at the top and incudes only parent rows.				
SelectedRows*	Defines, via an INFOTABLE source, which rows are highlighted by default when the grid is displayed. When used in a Tree grid, only the Row ID column is required to make row selections but other columns can be included. In an Advanced grid, row selections are handled by binding the output of the **SelectedRows** parameter in a service to the input **SelectedRows** property on the grid. This property is bindable in either an output or an input direction so that one entity can control the selection of rows in another. For example, one table can control the selection of rows in a second table, or a 3D image can be used to select rows in a table.	INFOTABLE	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	For the controlling entity, bind the service as an output INFOTABLE: ➡ For the entity being controlled, bind the service as an input INFOTABLE: ⬅ NOTE: This property will have no effect if the **RowSelection** property is set to **none**. In order to select multiple rows, the **RowSelection** property must to be set to **multi**.				
IncludeRowExpansionParents*	Determines whether or not parent rows that are not included in preloaded client-side data will be included when selecting or expanding child rows. If True, the parent rows will be fetched with the child rows so the hierarchy can be recreated. NOTE: Depending on the depth and size of your data, using this property can affect grid performance. See Tree Grid Performance Guidelines.	BOOLEAN	False	N	TreeGrid only
ExpandRows*	IDs of any top-level or child rows in the grid that should be expanded. Only the Row ID column is required in order to select rows for expansion.	INFOTABLE	n/a	Y	TreeGrid only
ExpandLoadedRows*	Enables auto-expanding of all preloaded data when the grid is launched. NOTES: Multiple levels of preloaded data must be available.	BOOLEAN	False	Y	TreeGrid only

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	When this property is enabled, it affects the way rows are highlighted when **DefaultSelectedRows** are defined. This property must be turned off in order for the **PreserveRowExpansion** property to be effective when enabled.				
ExpandRowOnDoubleClick	Allows a row with children to be expanded when double clicked. Rows can also be expanded by clicking on the node icon itself. In a JSON configuration service, the property name is: **treeSettings.expandRowOnDoubleClick**	BOOLEAN	False	N	TreeGrid only
PreserveRowExpansion	Enables row expansion selections to be preserved when the grid is refreshed. When using this property, make sure the **maxLevels** property in your tree-loading data service is set to a value greater than the level you want to expand to. For more information about the data service, see Using a Tree-Loading Data Service. NOTES: If the **ExpandLoadedRows** property is enabled, it will overwrite this property and expand all of the preloaded rows. If you want to preserve a specific expansion of rows, turn off the **ExpandLoadedRows** when you turn on **PreserveRowExpansion**.	BOOLEAN	False	N	TreeGrid only

Property Name	**Description**	**Base Type**	**Default Value**	**Bindable (Y/N)?**	**Applicable To Grid or TreeGrid?**
	The **CookiePersistence** property must be enabled in order to preserve row expansion values.				
RowSelection	Controls what row selection is possible to configure. Options: **none**, **single**, or **multi**. NOTE: If the **none** option is selected, other row selection properties will have no effect. NOTE: When you enable **IsEditable** or **EnableEditButtons** properties, **RowSelection** property will not take effect in the grid. The user can select rows when the grid is not in edit mode.	STRING	None	N	Grid only
AutoScroll	Controls whether or not the grid automatically scrolls to selected rows when the grid is resized or the service is refreshed.	BOOLEAN	False	N	Both
CookiePersistence*	Enables client-side persistence for certain column settings (order, size, visibility, and sort order).	BOOLEAN	True	N	Both
EnableContextMenu*	Enables or disables the display of a grid context menu, at run time, that allows an end user to show or hide specific columns. Works in conjunction with **CookiePersistence**:	BOOLEAN	True	N	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?

	• If both properties are enabled – a user can show/hide columns and those selections persist.
	• If **EnableContextMenu** is disabled and **CookiePersistence** is enabled – a user cannot show/hide columns but previous selections will persist.
	• If **EnableContextMenu** is enabled and **CookiePersistence** is disabled – a user can show/hide columns, but only for the current request.
	In a JSON configuration service, the property can be set as a top-level parameter as follows:
	`var config = {`
	`“enableContextMenu”: false,`
	`...`
	`}`				

EnableSorting	Must be enabled for any type of column sorting to take place, including ascending/descending toggling from headers, the **MultiColumnSortOrder** property, or binding a sorting service. When this option is enabled, the following properties become available in the properties panel: • **QueryFilter** – a bindable filter query for use with a data service	BOOLEAN	False	N	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	• **Filter –** a bindable event property to trigger a query data service				
MultiColumnSortOrder	Sets a default column sort order. Syntax: column name:order,column name:order **Example**: office:asc,title:des Note: **EnableSorting** must be turned on in order for **MultiColumnSortOrder** to have any effect.	STRING	n/a	N	Both
EnableGridSearch	Allows placement of a toolbar with a global Search box on the grid. When this option is enabled, the following properties become available in the properties panel: • **QueryFilter** – a bindable filter query for use with a data service • **Filter –** a bindable event property to trigger a query data service	BOOLEAN	False	N	Both
GridSearchLocation	Defines where to place the Search box. This option only becomes available when the **EnableGridSearch** property is turned on.	STRING	n/a	N	Both
QueryFilter*	A bindable query property used to bind a query service as the input query parameter to control sorting, searching, and filtering of the data. This property becomes available when either the **EnableSorting** or **EnableGridSearch** properties are turned on. It can be set	QUERY	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	from the properties panel or from the context menu on the grid itself. If you are using a data filter widget in your mashup, the output **QueryFilter** property can be bound to the input query property from either an Advanced or Tree grid widget. The grid combines all the query parameters to create a single output filter that is bound to the specified service. When the query filter is bound in both directions like this, filled arrows are displayed: ⇆ If there is no data filter widget and the binding is only in the output direction, one arrow is filled and the other is unfilled: ⇆				
EnableGridReset	Allows placement of a toolbar with a grid Reset button. Click **Reset** to clear all grid user settings stored in cookies and return the grid to its default configuration.	BOOLEAN	False	N	Both
EnableFilterEventOnConfigChange*	Enables and disables event firing when a configuration is updated from a service. When this property is enabled and a bound configuration is changed, a filter event is fired to update the data as well. If this property is disabled, the filter event does not fire when the bound configuration is updated.	BOOLEAN	True	N	Both
EnableFooter	Enables a footer section in the grid. When you set this property to True, two	BOOLEAN	False	N	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	additional properties named **FooterData** and **TableFooterStyle** are enabled. The data in the footer section is bound using the **FooterData** property. For more information, see Footer Section in Both Advanced Grids.				
FooterData	Contains the data to be displayed in the footer of the grid.	INFOTABLE	n/a	Y	Both
GridResetButtonLocation	Defines where to place the grid reset button.	STRING	n/a	N	Both
RowFormat	Opens a dialog box where optional row-based rules can be defined to apply dynamic **State Formatting**. These row-based rules can be over ridden by cell-based state formatting, which is available from the **Configure Grid Columns** option on the grid context menu.	STATE FORMATTING	State Formatting	N	Both
TableWrapperStyle	Defines the grid background styles. This adds outline color around the entire table and sets the background color. Attributes that are supported are line color, weight and type, background color, and alternate background color.	STYLE DEFINITION	DefaultTableWrapperStyle	N	Both
TableHeaderStyle	Defines the grid header styles.	STYLE DEFINITION	DefaultTableHeaderStyle	N	Both
FocusStyle	Defines the style of a row that has focus in the grid.	STYLE DEFINITION	DefaultFocusStyle	N	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
RowBackgroundStyle	Defines a row background style. This adds background color, font color, and weight style. But it does not add line style to each row.	STYLE DEFINITION	DefaultRowBackground Style	N	Both
RowAlternateBackgrou ndStyle	Defines a second row background style for alternate rows. This adds font color and weight style, but does not add line style.	STYLE DEFINITION	DefaultRowAlternateBa ckgroundStyle	N	Both
RowHoverStyle	Defines the style of a row when hovered over. This adds background color and font to every cell. Line color, weight and type is applied around the entire row which is hovered over only.	STYLE DEFINITION	DefaultRowHoverStyle	N	Both
RowSelectedStyle	Defines the style of a row when selected. This adds background color and font to every cell. Line color, weight and type is applied around the entire row which is selected only.	STYLE DEFINITION	DefaultRowSelectedSty le	N	Both
RowBorderStyle	Defines the row border styles. This adds line color, weight and type to horizontal sides of the cell.	STYLE DEFINITION	DefaultRowBorderStyle	N	Both
CellBorderStyle	Defines cell border styles. This adds line color, weight and type to vertical sides of the cell.	STYLE DEFINITION	DefaultCellBorderStyle	N	Both
ToolbarStyle	Defines styles for toolbars when displayed.	STYLE DEFINITION	DefaultToolbarStyle	N	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
TableFooterStyle	Defines the style for footer section in the grid. This includes background color, border style, and font style.	STYLE DEFINITION	DefaultGridAdvancedFooterStyle	N	Both
TooltipStyle	Defines styles for tooltips.	STYLE DEFINITION	DefaultTooltipStyle	N	Both
SortAscendingStyle	Defines the style of the sort ascending icon.	STYLE DEFINITION	DefaultSortAscendingStyle	N	Both
SortDescendingStyle	Defines the style of the sort descending icon.	STYLE DEFINITION	DefaultSortDescendingStyle	N	Both
CellValidationErrorStyle	Defines the style of the cell when a validation error occurs.	STYLE DEFINITION	DefaultCellValidationErrorStyle	N	Both
CellValidationErrorTooltipStyle	Defines the style of the cell tooltip when a validation error occurs.	STYLE DEFINITION	DefaultCellValidationErrorTooltipStyle	N	Both
RowIconStyle	Defines the style of the folder icon for tree nodes.	STYLE DEFINITION	DefaultRowIconStyle	N	TreeGrid only
RowExpansionIconStyle	Defines the style of the expansion icon for tree nodes.	STYLE DEFINITION	DefaultRowExpansionIconStyle	N	TreeGrid only
RowCollapseIconStyle	Defines the style of the collapse icon for tree nodes.	STYLE DEFINITION	DefaultRowCollapseIconStyle	N	TreeGrid only
HeaderOverflow	Provides options for handling header cell text that overflows. Options: • **fitted** – Text is fitted to the column width and subsequently wraps, even in mid-word. • **wrapped** – Text wraps to additional lines on white space or a dash.	STRING	tooltip	N	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	• **clipped** – Text is cut off at the end of the header cell. • **ellipsis** – Text is cut off but with an ellipsis (…) to show there is more text. • **tooltip** – Text is cut off with an ellipsis (…) and full text is displayed in a tooltip.				
DataOverflow	Provides options for data cell text that overflows. The same options are available as in the **HeaderOverflow** property.	STRING	clipped	N	Both
MaxHeaderHeight	The maximum height (in pixels) the header row can expand to before vertical scroll bars appear.	NUMBER	100	N	Both
MinRowHeight	The minimum height setting (in pixels) for a row in the grid. When not using an image renderer for a column that is showing images that are larger than the default minimum row height of 30 pixels, like a state definition that applies styles containing images, ensure you enlarge the row height setting to accommodate the height of the image.	NUMBER	0	N	Both
MaxRowCacheSize	The maximum number of rows that can be expanded, client-side, in the grid. When the limit is reached, a warning is generated and nodes will need to be collapsed before additional expansion.	NUMBER	50000	N	TreeGrid only
ShowDataLoading*	Displays data as it loads.	BOOLEAN	True	N	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
DoubleClicked*	A bindable event property fired when the grid is double-clicked.	EVENT	n/a	Y	Both
Filter*	A bindable query property used to bind a query service as the input query parameter to control sorting, searching, and filtering of the data. This property becomes available when either the **EnableSorting** or **EnableGridSearch** properties are turned on. It can be set from the properties panel or from the context menu on the grid itself.	EVENT	n/a	Y	Both
EditCellStarted	A bindable event property that can be triggered when a user begins to edit a cell value. Only active when the **IsEditable** parameter is enabled. This event can be used to change the state of other widgets in the mashup on editing.	EVENT	n/a	Y	Both
EditCellCompleted	A bindable event property that can be triggered when a user edits a cell and then either clicks Enter, Tab, or anywhere outside the edited cell. Clicking Esc will leave the value unedited. When the grid is refreshed, the edited values will display. Two uses for this event include the following: • It can be bound to an infotable update service so that edited values	EVENT	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	from the **EditedTable** infotable are persisted. • It can be bound to a service that enables a Save button widget in the Mashup. The Save button can, in turn, be bound to an infotable update service so that updated values from the **EditedTable** infotable are persisted.				
EditStarted	This event is triggered when you click the **Edit** button in the grid tool bar.	EVENT	n/a	Y	Both
EditCompleted	This event is triggered when you click the **Save** button in the grid tool bar.	EVENT	n/a	Y	Both
EditCancelled	This event is triggered when you click the **Cancel** button in the grid tool bar.	EVENT	n/a	Y	Both
Z-index*	The ordering for layered widgets. A lower Z-index will move the grid widget behind another widget with a higher Z-index.	NUMBER	10	N	Both
Visible*	If enabled, the grid displays in Run Time. This property can be defined by a service bound to the grid. If a visible service is bound to the grid, a filled arrow is displayed: ➡ If there is no visible service, the arrow is unfilled: ⇨	BOOLEAN	True	Y	Both
SelectedRowsChanged*	A bindable event property that can be used to trigger another widget or service	EVENT	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	when the user selects or deselects one or more rows in the grid. To use this event property: • Set the **RowSelection** property to **single** or **multi**. • Bind the **SelectedRows** property to another entity (a widget or service). • Bind the **SeletedRowsChanged** event to the other entity so that it will be triggered when **SelectedRows** changes.				

* When a configuration service is bound to the grid, only the starred properties are displayed in the Properties panel of the Mashup Builder. All of the other properties are hidden from view because those parameters are passed in from the JSON service.

NOTE: Grid Styles will take precedence over one another in the order listed below, with TableWrapperStyle having the lowest priority, and RowHoverStyle overriding all others.

1. TableWrapperStyle
2. RowBackground/RowAlternateBackgroundStyle
3. CellBorder/RowBorderStyle
4. FixedRowFormatter/State Definition
5. RowSelectedStyle
6. RowHoverStyle

Footer Section in Both Advanced Grids

You can add a footer section in the Advanced Grid and the Advanced Tree Grid using Mashup Builder properties or JSON configuration. The footer is configurable and it displays the totals for the data in the columns.

To add a footer, do the following:

1. Select the **EnableFooter** property to set it to true.

   This enables the **FooterData** and **TableFooterStyle** properties in the Mashup Builder. You can provide the footer data through a service that is bound to the FooterData infotable property.

   NOTE: If you do not bind the data, a message appears, and the grid will not render.

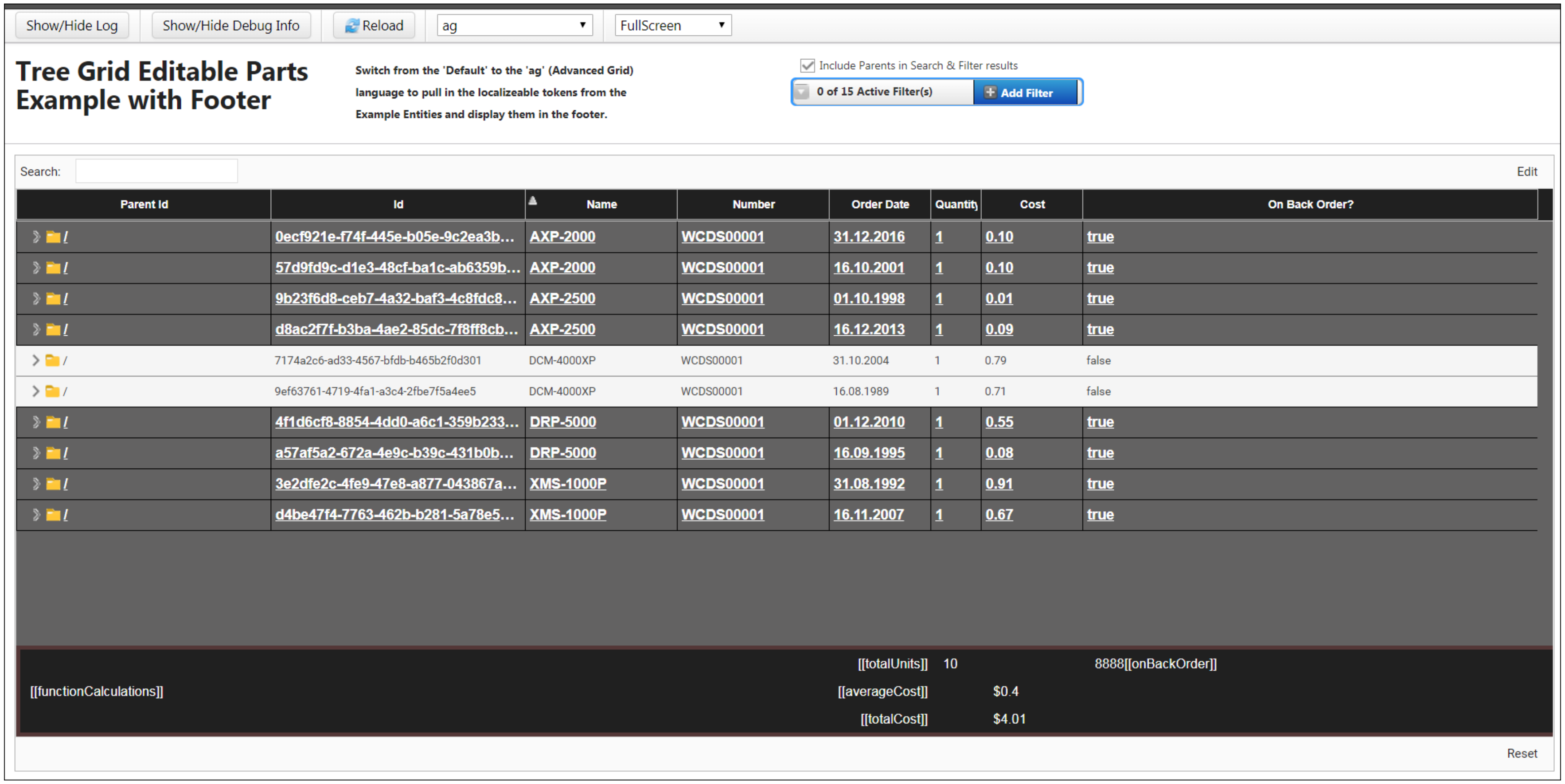
In a JSON configuration service, the footer property can be set as a top-level parameter as follows:

```
var config = {
“enableFooter”: false,
...
}
```

In a JSON configuration service, the footer style can be set in the style section as follows:

```
var config = {
“tableFooterStyle”:
...
}
```

The footer data is described in two example services, named **GetPartsFooterData** and **GetWeatherFooterData** on the **GridAdvancedExampleServices** Thing. The first service is used in the **HierarchicalEditablePartsWithFooterExample** and the second is used in the **WeatherFooterEXample**.



Tree Grid Editable Parts Example with Footer

Switch from the 'Default' to the 'ag' (Advanced Grid) language to pull in the localizeable tokens from the Example Entities and display them in the footer.

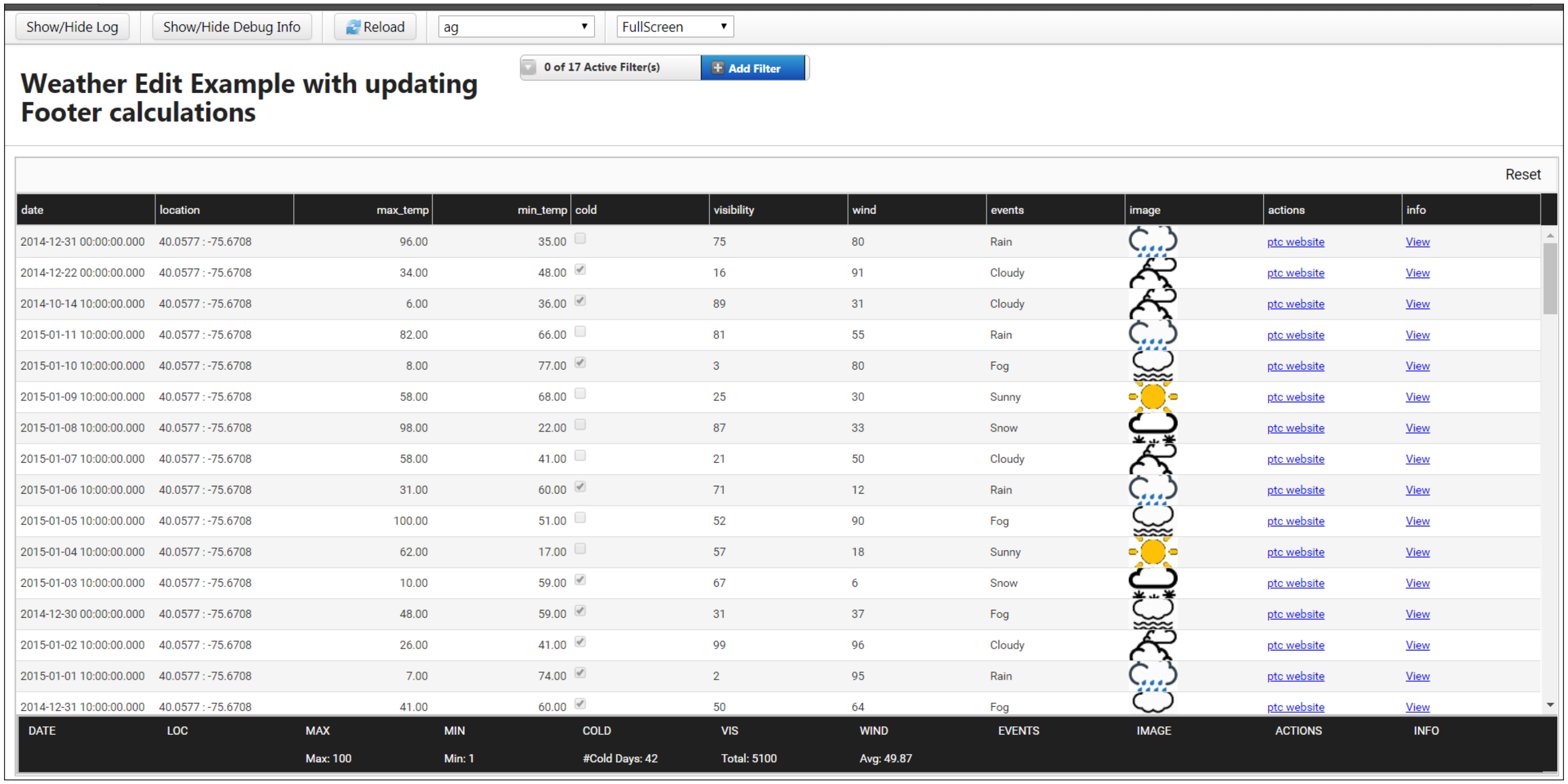
Search:

Parent Id	Id	Name	Number	Order Date	Quantity	Cost	On Back Order?
/	0ecf921e-f74f-445e-b05e-9c2ea3b…	AXP-2000	WCDS00001	31.12.2016	1	0.10	true
/	57d9fd9c-d1e3-48cf-ba1c-ab6359b…	AXP-2000	WCDS00001	16.10.2001	1	0.10	true
/	9b23f6d8-ceb7-4a32-baf3-4c8fdc8…	AXP-2500	WCDS00001	01.10.1998	1	0.01	true
/	d8ac2f7f-b3ba-4ae2-85dc-7f8ff8cb…	AXP-2500	WCDS00001	16.12.2013	1	0.09	true
/	7174a2c6-ad33-4567-bfdb-b465b2f0d301	DCM-4000XP	WCDS00001	31.10.2004	1	0.79	false
/	9ef63761-4719-4fa1-a3c4-2fbe7f5a4ee5	DCM-4000XP	WCDS00001	16.08.1989	1	0.71	false
/	4f1d6cf8-8854-4dd0-a6c1-359b233…	DRP-5000	WCDS00001	01.12.2010	1	0.55	true
/	a57af5a2-672a-4e9c-b39c-431b0b…	DRP-5000	WCDS00001	16.09.1995	1	0.08	true
/	3e2dfe2c-4fe9-47e8-a877-043867a…	XMS-1000P	WCDS00001	31.08.1992	1	0.91	true
/	d4be47f4-7763-462b-b281-5a78e5…	XMS-1000P	WCDS00001	16.11.2007	1	0.67	true

```
                                   [[totalUnits]]  10          8888[[onBackOrder]]
[[functionCalculations]]          [[averageCost]]       $0.4
                                     [[totalCost]]       $4.01
```

1-Tree Grid example with footer

NOTE: Put the token names in double square brackets in the infotable data to localize them. For example: **[[totalUnits]]** in the image above.



Weather Edit Example with updating Footer calculations

date	location	max_temp	min_temp	cold	visibility	wind	events	image	actions	info
2014-12-31 00:00:00.000	40.0577 : -75.6708	96.00	35.00	☐	75	80	Rain		ptc website	View
2014-12-22 00:00:00.000	40.0577 : -75.6708	34.00	48.00	✓	16	91	Cloudy		ptc website	View
2014-10-14 10:00:00.000	40.0577 : -75.6708	6.00	36.00	✓	89	31	Cloudy		ptc website	View
2015-01-11 10:00:00.000	40.0577 : -75.6708	82.00	66.00	☐	81	55	Rain		ptc website	View
2015-01-10 10:00:00.000	40.0577 : -75.6708	8.00	77.00	✓	3	80	Fog		ptc website	View
2015-01-09 10:00:00.000	40.0577 : -75.6708	58.00	68.00	☐	25	30	Sunny		ptc website	View
2015-01-08 10:00:00.000	40.0577 : -75.6708	98.00	22.00	☐	87	33	Snow		ptc website	View
2015-01-07 10:00:00.000	40.0577 : -75.6708	58.00	41.00	☐	21	50	Cloudy		ptc website	View
2015-01-06 10:00:00.000	40.0577 : -75.6708	31.00	60.00	✓	71	12	Rain		ptc website	View
2015-01-05 10:00:00.000	40.0577 : -75.6708	100.00	51.00	☐	52	90	Fog		ptc website	View
2015-01-04 10:00:00.000	40.0577 : -75.6708	62.00	17.00	☐	57	18	Sunny		ptc website	View
2015-01-03 10:00:00.000	40.0577 : -75.6708	10.00	59.00	✓	67	6	Snow		ptc website	View
2014-12-30 00:00:00.000	40.0577 : -75.6708	48.00	59.00	✓	31	37	Fog		ptc website	View
2015-01-02 10:00:00.000	40.0577 : -75.6708	26.00	41.00	✓	99	96	Cloudy		ptc website	View
2015-01-01 10:00:00.000	40.0577 : -75.6708	7.00	74.00	✓	2	95	Rain		ptc website	View
2014-12-31 10:00:00.000	40.0577 : -75.6708	41.00	60.00	✓	50	64	Fog		ptc website	View
DATE	LOC	MAX	MIN	COLD	VIS	WIND	EVENTS	IMAGE	ACTIONS	INFO
		Max: 100	Min: 1	#Cold Days: 42	Total: 5100	Avg: 49.87				

2- Weather data example with footer

You can find these examples in the sample file **GridAdvancedExampleEntities-V4.0.xml**. For more information, see Samples File.

Functions that can be Included in the Footer Infotable:

You can also perform client-side calculations in the grid using the following functions in the JSON configuration:

- `{#stat_count}` – Counts the number of rows.
- `{#stat_max}` - Calculates the maximum client-side value for the values in the column.

- `{#stat_min}` - Calculates the minimum client-side value for the values in the column.
- `{#stat_average}` - Calculates the average client-side value for the values in the column.
- `{#stat_total}` - Calculates the total client-side value for the values in the column.
- `{#cspan}` – Span columns.

NOTE: You can perform your own calculations and add them to the footer. For example, the calculation of Cold days in the **COLD** column in the image above: *Weather data example with footer*.

NOTE: You can align data in the footer using `#cspan` and text alignment settings `text-align:left` or `text-align:right`. Use HTML escape characters for comma in text, and the text following the comma is the alignment setting in the configuration, which is by default `text-align:left`.

NOTE: If you are using `#cspan` elements in columns to span several columns and you move columns, issues may occur at runtime. To ensure that text and data stay together in the footer when columns are moved, put them together in the same column.

Column Configuration from the Context Menu

If the data source for your grid is tied to a Data Shape, certain column parameters are configurable from the context menu.

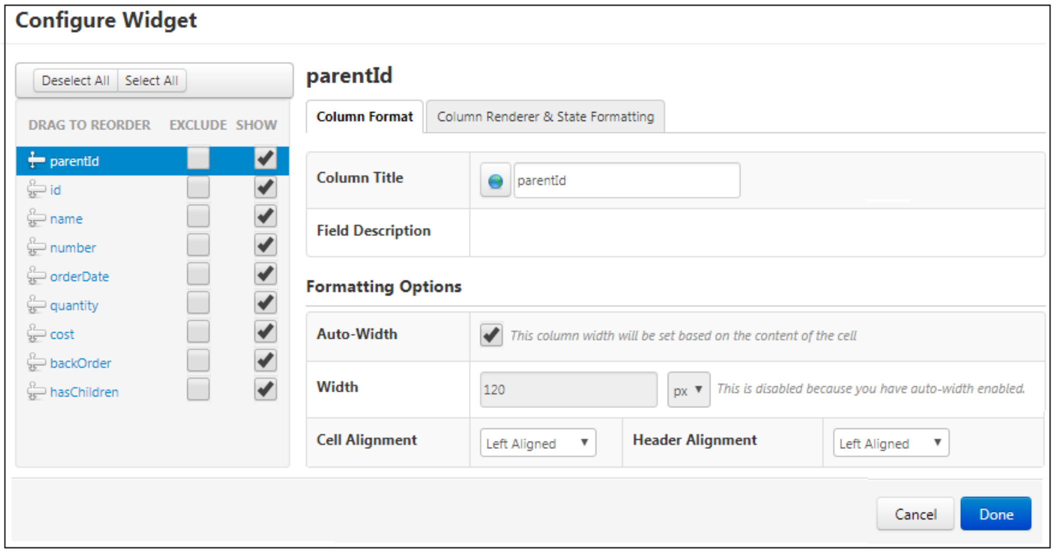
Accessing Parameters on the Context Menu

1. In the Mashup Builder:

   Hover over the menu drop down arrow in the top left corner of the grid, or

   Click the setting icon at the top of the properties panel on the left.

2. Select the **Configure Grid Columns** option. A Configure Widget dialog box opens.



DRAG TO REORDER	EXCLUDE	SHOW
parentId	☐	✓
id	☐	✓
name	☐	✓
number	☐	✓
orderDate	☐	✓
quantity	☐	✓
cost	☐	✓
backOrder	☐	✓
hasChildren	☐	✓

parentId

Column Title: parentId

Field Description:

Formatting Options

Auto-Width: ✓ *This column width will be set based on the content of the cell*

Width: 120 px *This is disabled because you have auto-width enabled.*

Cell Alignment: Left Aligned
Header Alignment: Left Aligned

3. Configure column properties in the following areas of the dialog box:

   - **Left pane**: Reorder the columns by dragging them to different locations in the list. When a column is selected, the tabs on the right display property settings specific to the selected column. You can also define which columns can be seen by the end user of the grid:

     - **Show –** Defines whether a specific grid column is initially displayed or hidden from view. End users can hide and unhide the column display in runtime by right-clicking the column header and using the context menu. To toggle this property on and off for all of the listed columns at once, use the **Hide All** and **Show All** buttons at the top of the panel. (This property corresponds to the **Hidden** property when writing a configuration service.)

     - **Exclude** – Defines whether a specific grid column can ever be seen by the end user. When checked, the end user will not see the column and will have no control over its display. However, data in the excluded column can still be used for state formatting. (This property corresponds to the **inLayout** property when writing a configuration service.)

- **Column Format tab:** Use this tab to edit the name and description of a specific column and to control the following column formatting options:
  - **Auto-Width** – Column width auto-adjusts to fit the content of the selected column.
  - **Width** – Column width can be set to a fixed pixel size or to a percentage. At runtime, after any fixed-width columns are sized, percentage-width columns divide up the remaining space according to the assigned percentages.

    If the total of all the percentage column widths exceeds 100%, each percentage column width is recalculated based on the specified percentage relative to the total. For example, if three columns are each assigned a percentage width of 50%, the calculation for each column width becomes: 50/150 = 33%.

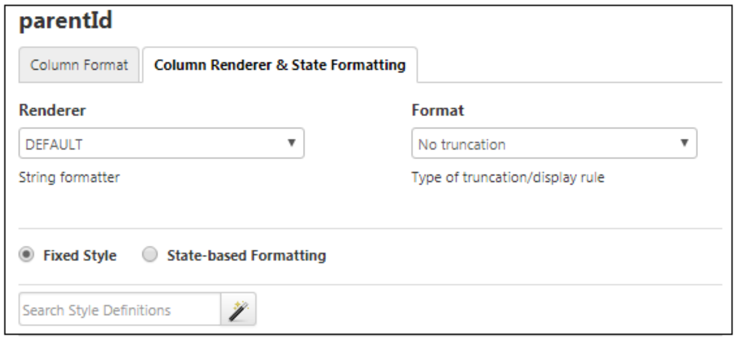
    Using percentage widths allows the grid to auto-resize responsively when the browser size changes.

    NOTE: To set individual column widths, the **Auto-Width** option must be unchecked.
  - **Cell Alignment** / **Header Alignment** – Alignment of text in column cells and the column header can be individually set.
  - **Cell Editing Options** – These options allow cell values in a column to be edited in run time. Click the **Editable** option for a specific column to enable run time editing. This option requires that the **IsEditable** grid parameter is also enabled. (Currently, cell editing is only available for Boolean fields, so the other options below the **Editable** option can be ignored.)

    NOTE: When configuring the order of columns in a tree grid, an editable column cannot be configured as the first column.

    See section Cell Editing for more information.

- **Column Renderer and State Formatting tab:** Use this tab to control how data is rendered in a column and to configure fixed or state formatting. The tab is divided into two sections:
  - The top portion contains the dropdown field to select a type of column **Renderer** and a corresponding **Format**. For more information about column renderers and their available formats, see Column Renderers and Formats.
  - The bottom portion contains options to select **Fixed Style** or **State-based Formatting**. The State formatting set on this tab is cell-based and overrides any row-based formatting set in the Mashup Builder or via a configuration service.



Renderer: DEFAULT
String formatter

Format: No truncation
Type of truncation/display rule

● Fixed Style ○ State-based Formatting

Cell Editing

Cell editing can be done using the **IsEditable** and **EnableEditButtons** properties and the methods explained below.

NOTE: You can set only one property at a time for cell editing.

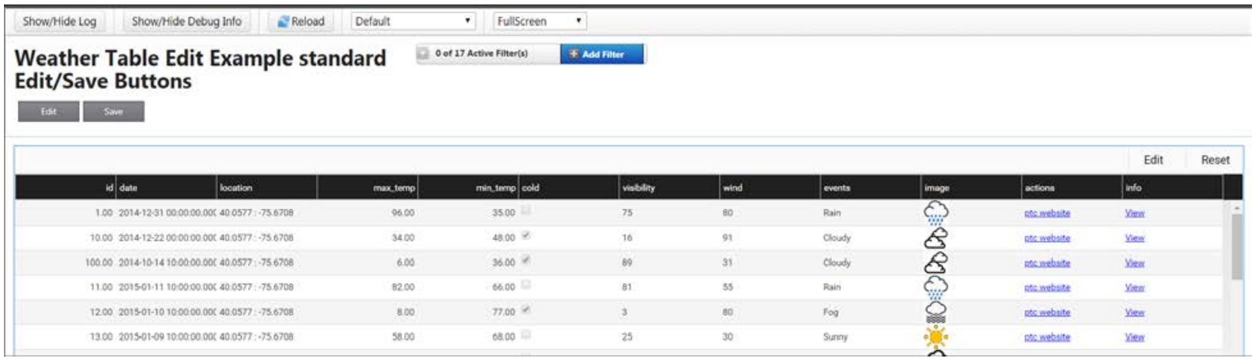
- **IsEditable** property – Cell editing is enabled for all supporting renderers by setting the **IsEditable** property to true in the Mashup Builder configuration or through JSON configuration. When you enable editing, an **EditedTable** property becomes available to bind the changed rows to a service. If you set these two properties, the grid will be in edit mode by default and no edit buttons will be available. You can use your own mashup edit buttons to put the grid in edit mode by binding the click events to a service, which would enable or disable the edit mode.

When you enable the **IsEditable** property, the following events are available:

  - **EditCellStarted** Event is triggered when you click on a cell to start editing.
  - **EditCellCompleted** Event is triggered when you edit a cell and press **Enter**, press the Tab key, or click outside of the cell.

- **EnableEditButtons** property – You can enable the **EnableEditButtons** property instead of the **IsEditable** property. If you set this property to true, a set of edit toolbar buttons appear in the grid, which can be positioned with the **EditButtonsLocation** drop-down settings to top left or right or bottom left or right. Now the end user can save the edits by clicking **Edit** followed by **Save** or **Cancel**.

See the images below to find the **Edit** button at the top right of the infotable.



Weather Table Edit Example standard Edit/Save Buttons

id	date	location	max_temp	min_temp	cold	visibility	wind	events	image	actions	info
1.00	2014-12-31 00:00:00.00(	40.0577 : -75.6708	96.00	35.00	☐	75	80	Rain	🌧	ptc website	View
10.00	2014-12-22 00:00:00.00(	40.0577 : -75.6708	34.00	48.00	✓	16	91	Cloudy	☁	ptc website	View
100.00	2014-10-14 10:00:00.00(	40.0577 : -75.6708	6.00	36.00	✓	89	31	Cloudy	☁	ptc website	View
11.00	2015-01-11 10:00:00.00(	40.0577 : -75.6708	82.00	66.00	☐	81	55	Rain	🌧	ptc website	View
12.00	2015-01-10 10:00:00.00(	40.0577 : -75.6708	8.00	77.00	✓	3	80	Fog	🌫	ptc website	View
13.00	2015-01-09 10:00:00.00(	40.0577 : -75.6708	58.00	68.00	☐	25	30	Sunny	☀	ptc website	View

The following events are available when you enable the edit buttons:

- **EditStarted** - when you click on the **Edit** button to start editing.
- **EditCompleted** - when you click **Save** button.
- **EditCancelled** - when you click the **Cancel** button.

Edits are done per cell by clicking on the cell which will then show the raw unformatted value. A cell editor appears in which you can make changes. The cell renderer is applied to format the value in its display format.

NOTE: Press the **Esc** key to leave the cell unchanged with its original value.

Selection of type of cell editor is done in the Mashup Builder in the **Cell Editing Options** table in the column configuration when the column is set to **Editable**.

Saving Edits

Edits to the grid are saved as follows:

- Auto-save: When you use the **IsEditable** property and bind **EditedRows** directly to the **AddOrUpdateProperties** service when the **EditCellCompleted** event is triggered.
- Manual save: When you use the edit buttons in the grid toolbar where the **EditedRows** are bound to the **AddOrUpdateProperties** service. This service is triggered by the **EditCompleted** event.

NOTE: The events are triggered only when the actual values are changed.

NOTE: If you provide an invalid value for a data type in the cell, the cell value remains unchanged. For example, if you provide a string value instead of a number value in the cell, the original value will not be changed.

Adding and Deleting Rows

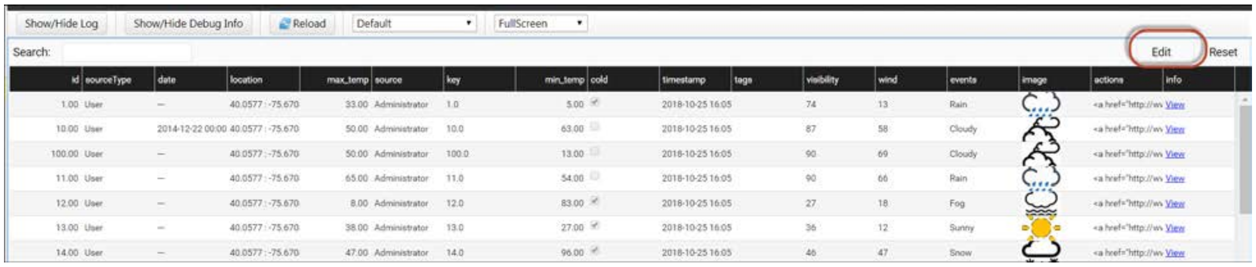
You can add or delete rows in Advanced Grid using the **EnableEditButtons** and **EnableAddDeleteButtons** properties. If you set the **EnableEditButtons** property to true, the **EnableAddDeleteButtons** property appears in the Mashup Builder configuration. When you set this property to true, **Add** and **Delete** buttons appear in the Grid tool bar enabling you to add or delete rows.

When you enable row addition and deletion, the **DeletedTable** property also becomes available. Use this property to bind deleted rows to a service.

NOTE: The **Add** and **Delete** buttons are not available at the same time as their operations are separated to avoid any data corruption or overwriting.

Click **Edit** button in the tool bar to see **Add**, **Save**, and **Cancel** buttons. Also, a new column with a check box in each cell appears on the left side of the grid.

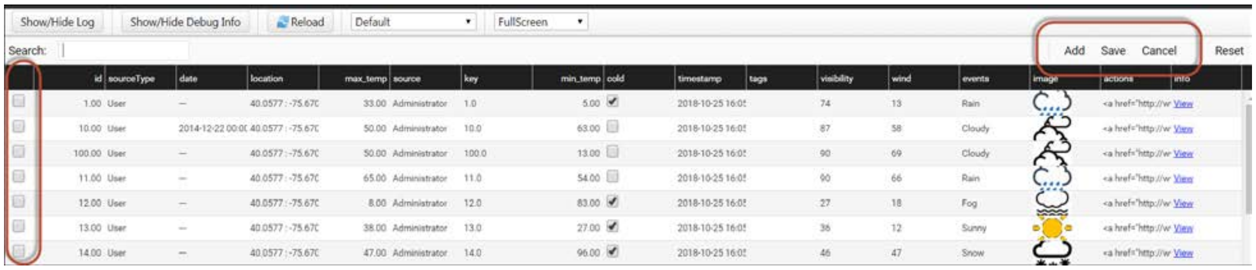
For example:



Show/Hide Log | Show/Hide Debug Info | Reload | Default | FullScreen

Search: Edit Reset

id	sourceType	date	location	max_temp	source	key	min_temp	cold	timestamp	tags	visibility	wind	events	image	actions	info
1.00	User	—	40.0577 : -75.670	33.00	Administrator	1.0	5.00	✓	2018-10-25 16:05		74	13	Rain		<a href="http://wv	View
10.00	User	2014-12-22 00:00	40.0577 : -75.670	50.00	Administrator	10.0	63.00	☐	2018-10-25 16:05		87	58	Cloudy		<a href="http://wv	View
100.00	User	—	40.0577 : -75.670	50.00	Administrator	100.0	13.00	☐	2018-10-25 16:05		90	69	Cloudy		<a href="http://wv	View
11.00	User	—	40.0577 : -75.670	65.00	Administrator	11.0	54.00	☐	2018-10-25 16:05		90	66	Rain		<a href="http://wv	View
12.00	User	—	40.0577 : -75.670	8.00	Administrator	12.0	83.00	✓	2018-10-25 16:05		27	18	Fog		<a href="http://wv	View
13.00	User	—	40.0577 : -75.670	38.00	Administrator	13.0	27.00	✓	2018-10-25 16:05		36	12	Sunny		<a href="http://wv	View
14.00	User	—	40.0577 : -75.670	47.00	Administrator	14.0	96.00	✓	2018-10-25 16:05		46	47	Snow		<a href="http://wv	View



Show/Hide Log | Show/Hide Debug Info | Reload | Default | FullScreen

Search: Add Save Cancel Reset

	id	sourceType	date	location	max_temp	source	key	min_temp	cold	timestamp	tags	visibility	wind	events	image	actions	info
☐	1.00	User	—	40.0577 : -75.67C	33.00	Administrator	1.0	5.00	✓	2018-10-25 16:0		74	13	Rain		<a href="http://w	View
☐	10.00	User	2014-12-22 00:0	40.0577 : -75.67C	50.00	Administrator	10.0	63.00	☐	2018-10-25 16:0		87	58	Cloudy		<a href="http://w	View
☐	100.00	User	—	40.0577 : -75.67C	50.00	Administrator	100.0	13.00	☐	2018-10-25 16:0		90	69	Cloudy		<a href="http://w	View
☐	11.00	User	—	40.0577 : -75.67C	65.00	Administrator	11.0	54.00	☐	2018-10-25 16:0		90	66	Rain		<a href="http://w	View
☐	12.00	User	—	40.0577 : -75.67C	8.00	Administrator	12.0	83.00	✓	2018-10-25 16:0		27	18	Fog		<a href="http://w	View
☐	13.00	User	—	40.0577 : -75.67C	38.00	Administrator	13.0	27.00	✓	2018-10-25 16:0		36	12	Sunny		<a href="http://w	View
☐	14.00	User	—	40.0577 : -75.67C	47.00	Administrator	14.0	96.00	✓	2018-10-25 16:0		46	47	Snow		<a href="http://w	View

- **To add a row:**

  1. Click **Add**.
  2. A new row appears at the bottom of the grid.
  3. Enter the desired values and click **Save**.

     New rows appear only at the bottom of the grid.

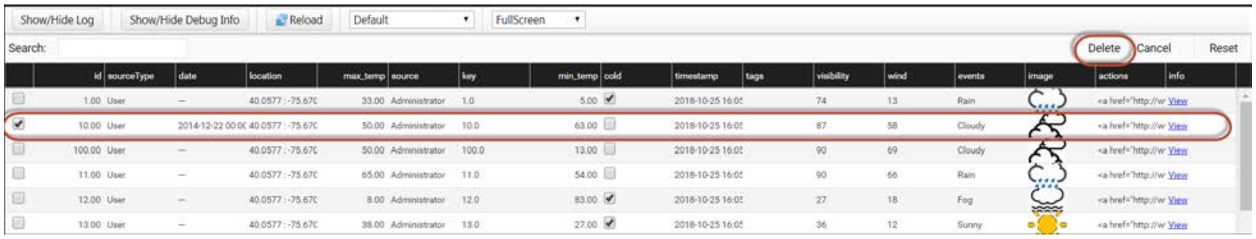
     Make sure that you enter correct values in the fields, as they are validated.

- **To delete a row:**

  1. Select the check box on the row you want to delete.
  2. Click **Delete**.

     After you delete a row, the **Edit** button reappears in the grid.

For example:



Show/Hide Log | Show/Hide Debug Info | Reload | Default | FullScreen

Search: Delete Cancel Reset

	id	sourceType	date	location	max_temp	source	key	min_temp	cold	timestamp	tags	visibility	wind	events	image	actions	info
☐	1.00	User	—	40.0577 : -75.67C	33.00	Administrator	1.0	5.00	✓	2018-10-25 16:0		74	13	Rain		<a href="http://w	View
✓	10.00	User	2014-12-22 00:0	40.0577 : -75.67C	50.00	Administrator	10.0	63.00	☐	2018-10-25 16:0		87	58	Cloudy		<a href="http://w	View
☐	100.00	User	—	40.0577 : -75.67C	50.00	Administrator	100.0	13.00	☐	2018-10-25 16:0		90	69	Cloudy		<a href="http://w	View
☐	11.00	User	—	40.0577 : -75.67C	65.00	Administrator	11.0	54.00	☐	2018-10-25 16:0		90	66	Rain		<a href="http://w	View
☐	12.00	User	—	40.0577 : -75.67C	8.00	Administrator	12.0	83.00	✓	2018-10-25 16:0		27	18	Fog		<a href="http://w	View
☐	13.00	User	—	40.0577 : -75.67C	38.00	Administrator	13.0	27.00	✓	2018-10-25 16:0		36	12	Sunny		<a href="http://w	View

The **Cancel** button enables you to cancel the addition or deletion and go back to the last operation.

Cell Editing in the JSON configuration – Add the following top-level global property to enable cell editing in the JSON configuration.

```
var config = {
   “cellEditingEnabled”: true,
   “columns”:
```

```
   ...
   }
};
```

Add the following content to any column that needs to be edited.

```
...
“ColumnFormatter”: {
   “type”: “boolean”,
   “format”: “notext”,
   “cellEditor”: {
“enabled”: true,  // *{boolean} to indicate whether cell editing is enable
for this column
},
}
...
```

Cell Validation

Cell validation is enabled in Advanced Grid and Advanced Tree Grid. Cell validation is essential to avoid any invalid entry in the grid during editing. Through cell validation, you can display a validation error and provide an error correction message in a tooltip. You can do this in the Mashup Builder configuration using either the **ValidationErrorCellStyle** and **ValidationErrorTooltipStyle** properties or through JSON.

Validation is available for each renderer type. Depending on the column renderer type, you can specify the validator in the JSON configuration. The following validators are available for each renderer type:

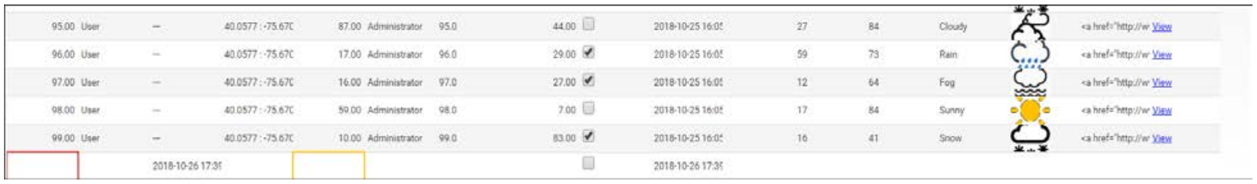
Renderer	Validators
string	<None>, NotEmpty, ValidURL
integer	<None>, NotEmpty, (ValidInteger is default)
long	<None>, NotEmpty, (ValidNumeric is default)
number	<None>, NotEmpty, (ValidNumeric is default)
boolean	<None>, NotEmpty, (ValidBoolean is default)
location	<None>, NotEmpty, (ValidLocation is default)
html	<None>, NotEmpty
imagelink	<None>, NotEmpty
hyperlink	<None>, NotEmpty, ValidURL
datetime	<None>, NotEmpty, (ValidDatetime is default)

When a validation fails, the configured error message is displayed in the tooltip for the cell, and the cell border is outlined in either red or yellow depending upon the type of error.

The validation errors are of two types:

- Blocking error: These are warning errors. For example, the errors in an id column or the errors in an empty cell that has been configured with the “NotEmpty” validator. You cannot save the edits until these errors are resolved. These errors are highlighted with a red border by default.
- Non-blocking error: These are not critical. For example, a string value entry in a cell that was configured for integer. You can save the edits before resolving these errors. These errors are highlighted with a yellow border by default.

See the following example to see the difference between these errors.



95.00	User	—	40.0577 : -75.67C	87.00	Administrator	95.0	44.00	☐	2018-10-25 16:0!	27	84	Cloudy		<a href="http://w View
96.00	User	—	40.0577 : -75.67C	17.00	Administrator	96.0	29.00	☑	2018-10-25 16:0!	59	73	Rain		<a href="http://w View
97.00	User	—	40.0577 : -75.67C	16.00	Administrator	97.0	27.00	☑	2018-10-25 16:0!	12	64	Fog		<a href="http://w View
98.00	User	—	40.0577 : -75.67C	59.00	Administrator	98.0	7.00	☐	2018-10-25 16:0!	17	84	Sunny		<a href="http://w View
99.00	User	—	40.0577 : -75.67C	10.00	Administrator	99.0	83.00	☑	2018-10-25 16:0!	16	41	Snow		<a href="http://w View
		2018-10-26 17:3!						☐	2018-10-26 17:3!					

NOTE: Cell validation is important on the primary key column to avoid any server error or exception. The primary key column is assigned using the **IDFieldName** property. Usually, it is the id column.

The **ValidationErrorCellStyle** property enables you to style the cell when a validation error occurs. This style supersedes any other style that has been configured for a cell.

The **ValidationErrorTooltipStyle** property enables you to style the tooltip when a validation error occurs.

Cell validation using JSON configuration – A validation error message can be configured in the JSON only. Also, a validation message can be localized by providing a l8n token in JSON configuration only.

Add the following content to enable cell validation and to define the validation error style and tooltip style.

```
"cellEditor": {                    // optional: defines cell-editing options for this column

 "enabled": true,              // *{boolean} to indicate whether cell editing is enable for this column

  "validator":  {                 // optional: when defined will validate input values against the chosen
validator types and error and success messages will be displayed.

  "types": ['NotEmpty'],    // {Array} optional: defines list of validator types, e.g. 'NotEmpty', 'ValidURL' or
'Custom'.

 "errorMessage":       "[[invalidNumberNotEmpty]]",   // *{string} localization token name of the error
message.

    "errorStyle":        "cellValidationErrorStyle"      // *{string} style definition name of the error message.
}

 }
```

```
"cellValidationErrorStyle":           {

      "backgroundColor":         "",

      "secondaryBackgroundColor": "",

      "foregroundColor":         "black",

      "textDecoration":          "none",

      "image":                   "",
      "lineColor":              "red",
      "borderStyle":            "solid",
      "borderWidth":            "1px",
      "fontFamily":             ["helvetica", "arial"],
      "fontStyle":              "normal",
      "fontSize":               "11px",
      "fontWeight":             "normal"
     },
     "cellValidationErrorTooltipStyle":          {
      "backgroundColor":        "red",
      "secondaryBackgroundColor": "",
      "foregroundColor":        "white",
      "textDecoration":         "none",
      "image":                  "",
      "lineColor":              "white",
      "borderStyle":            "solid",
      "borderWidth":            "1px",
      "fontFamily":             ["helvetica", "arial"],
      "fontStyle":              "normal",
      "fontSize":               "11px",
      "fontWeight":             "normal"
     },
     "tooltipStyle": {
      "backgroundColor":        "#0000ff",
      "secondaryBackgroundColor": "",
      "foregroundColor":        "#ffffff",
      "textDecoration":         "",
      "image":                  "",
      "lineColor":              "",
      "borderStyle":            "none",
        "borderWidth":          "",
        "fontFamily":          ["helvetica", "arial"],
        "fontStyle":           "normal",
        "fontSize":            "12px",
        "fontWeight":            ""
      },
```

Apply Style Theme

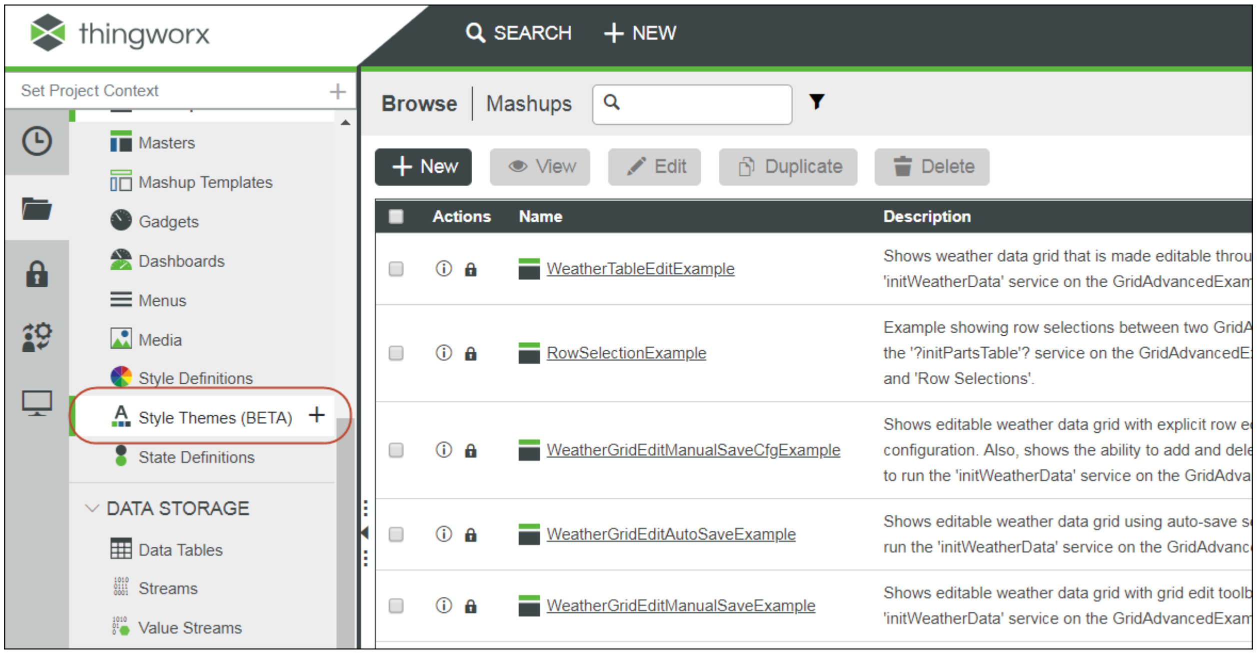
The Advanced Grid and Advanced Tree Grid widgets support custom style theme in a mashup for ThingWorx 8.4.x and higher versions. You can custom style the grid list in a mashup using this new feature.

NOTE: For information about the style themes feature, see the topic **Mashup Builder**> **Style Themes (Beta)** in the ThingWorx Platform Help Centre.

The steps to apply a sample style theme to a grid mashup are explained in brief here.

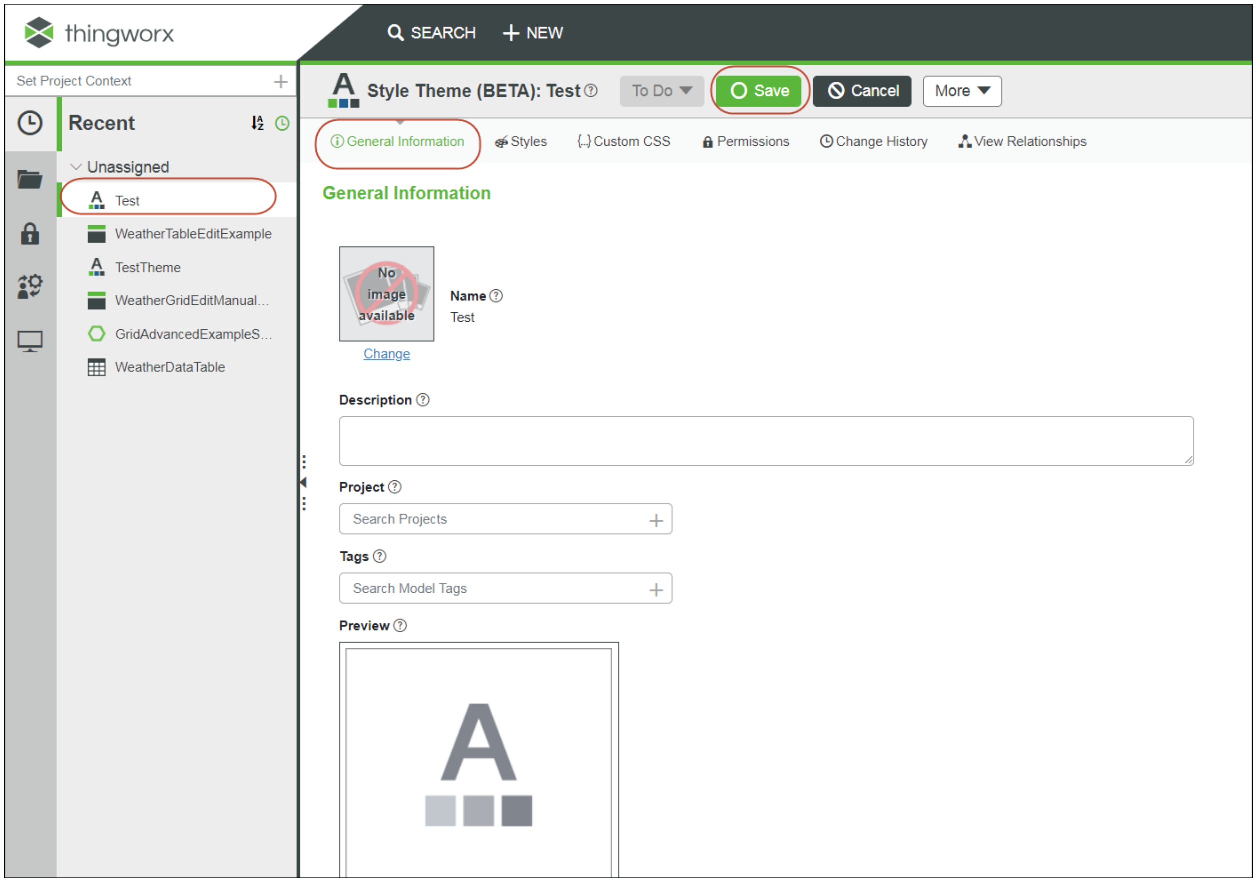
1. Create a custom style theme.

   a) Click **Browse**> **VISUALIZATION**> **Style Themes (BETA)**> **+**.

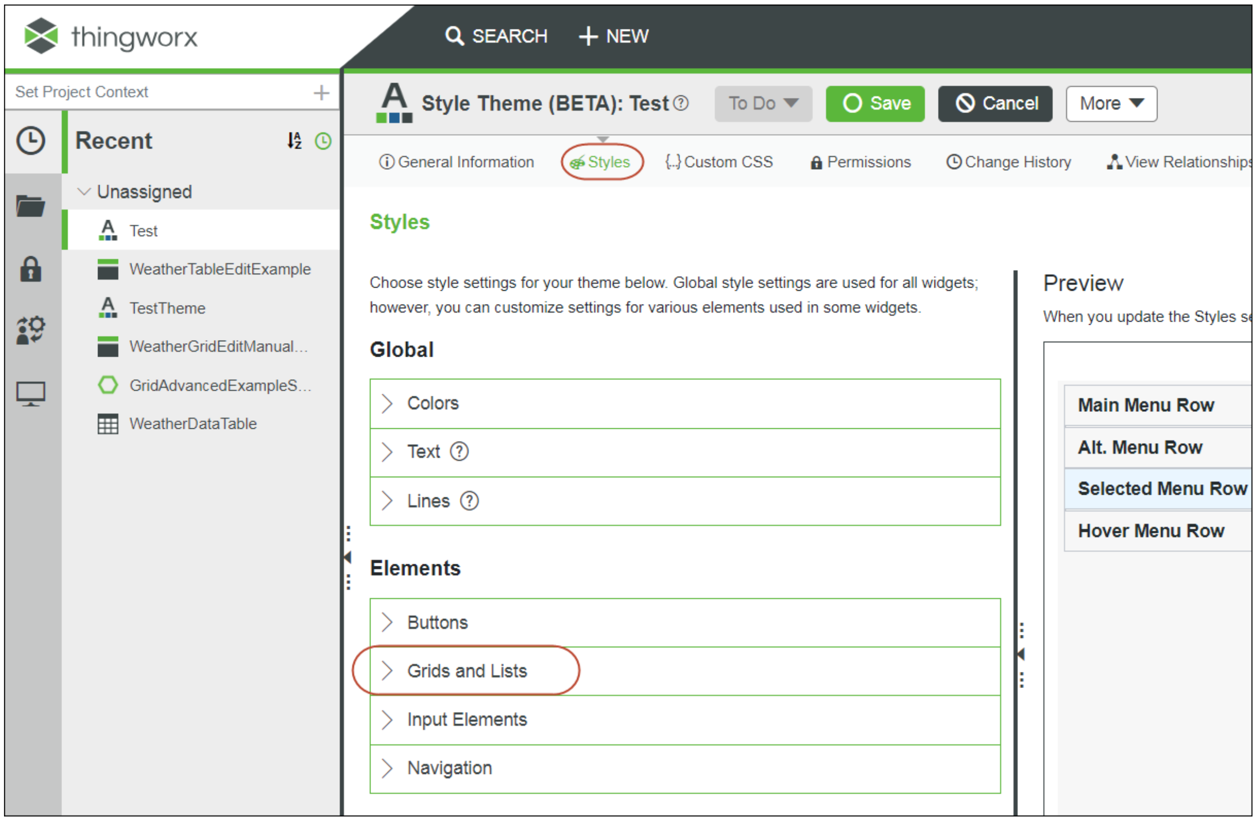


Actions	Name	Description
ⓘ 🔒	WeatherTableEditExample	Shows weather data grid that is made editable throu 'initWeatherData' service on the GridAdvancedExam
ⓘ 🔒	RowSelectionExample	Example showing row selections between two GridA the '?initPartsTable'? service on the GridAdvancedE and 'Row Selections'.
ⓘ 🔒	WeatherGridEditManualSaveCfgExample	Shows editable weather data grid with explicit row e configuration. Also, shows the ability to add and dele to run the 'initWeatherData' service on the GridAdva
ⓘ 🔒	WeatherGridEditAutoSaveExample	Shows editable weather data grid using auto-save s run the 'initWeatherData' service on the GridAdvanc
ⓘ 🔒	WeatherGridEditManualSaveExample	Shows editable weather data grid with grid edit toolb 'initWeatherData' service on the GridAdvancedExam

   b) A new page is opened. Fill the required details in the **General Information** tab.

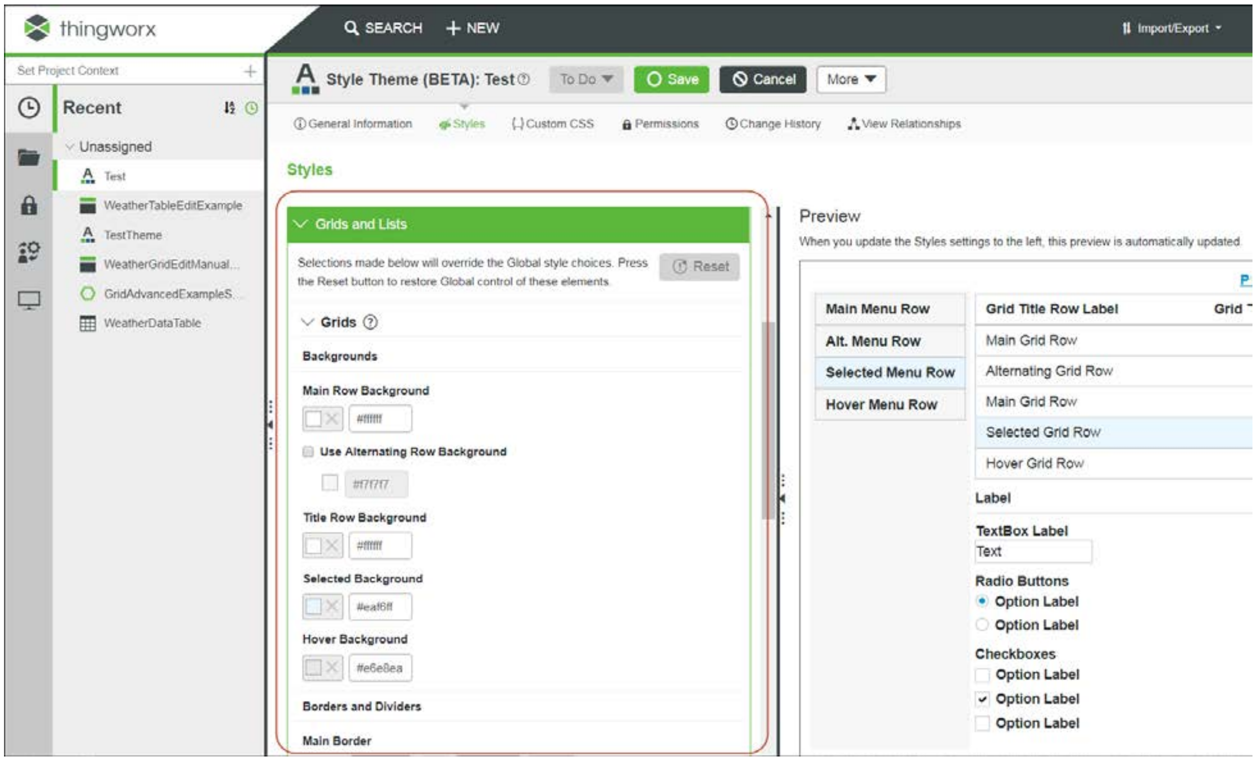


c) In the **Styles** page, go to **Elements**> **Grids and Lists**. You can set the style from the listed options.



Choose style settings for your theme below. Global style settings are used for all widgets; however, you can customize settings for various elements used in some widgets.

When you update the Styles se

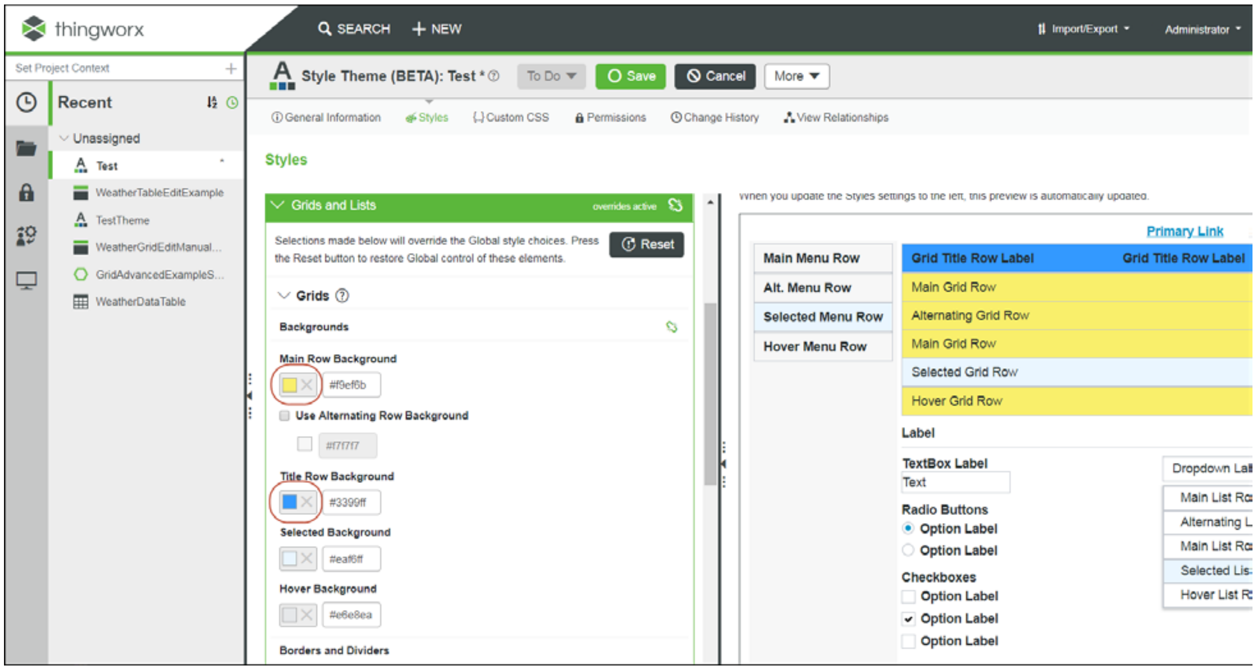


Selections made below will override the Global style choices. Press the Reset button to restore Global control of these elements.

Main Row Background: #ffffff
Use Alternating Row Background: #f7f7f7
Title Row Background: #ffffff
Selected Background: #eaf6ff
Hover Background: #e6e8ea

When you update the Styles settings to the left, this preview is automatically updated.

d) For example, set the **Main Row Background** style as Yellow and **Title Row Background** style as Blue.



Selections made below will override the Global style choices. Press the Reset button to restore Global control of these elements.

When you update the Styles settings to the left, this preview is automatically updated.

Main Row Background: #f9ef6b

Use Alternating Row Background: #f7f7f7

Title Row Background: #3399ff

Selected Background: #eaf6ff

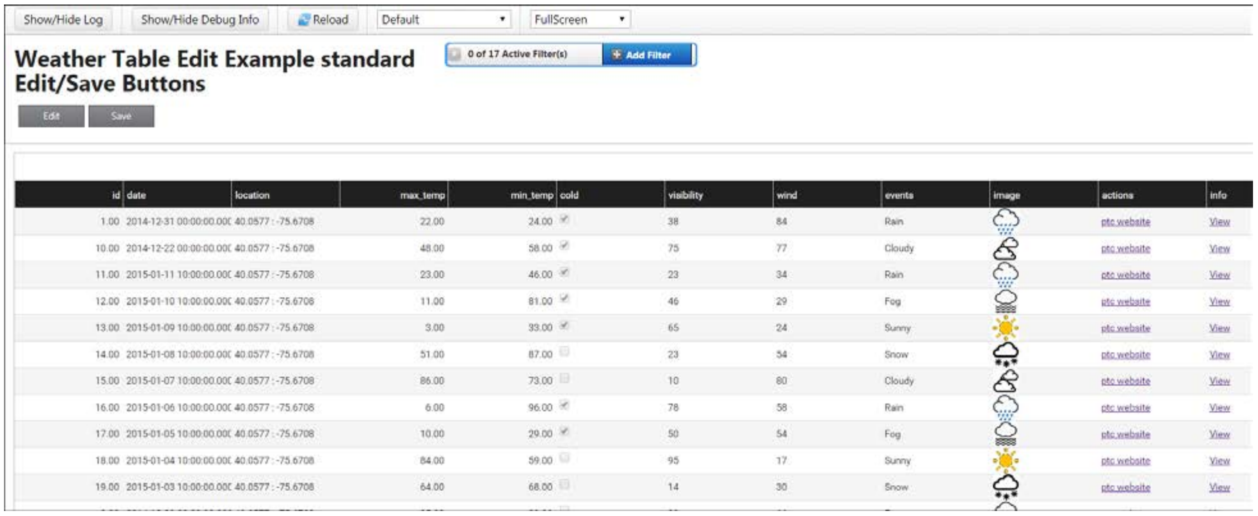
Hover Background: #e6e8ea

e) Click **Save** and save your theme.

2. Select your mashup that has grid.

   For example, select **WeatherTableEditExample** mashup in the example entities.

   View the mashup.

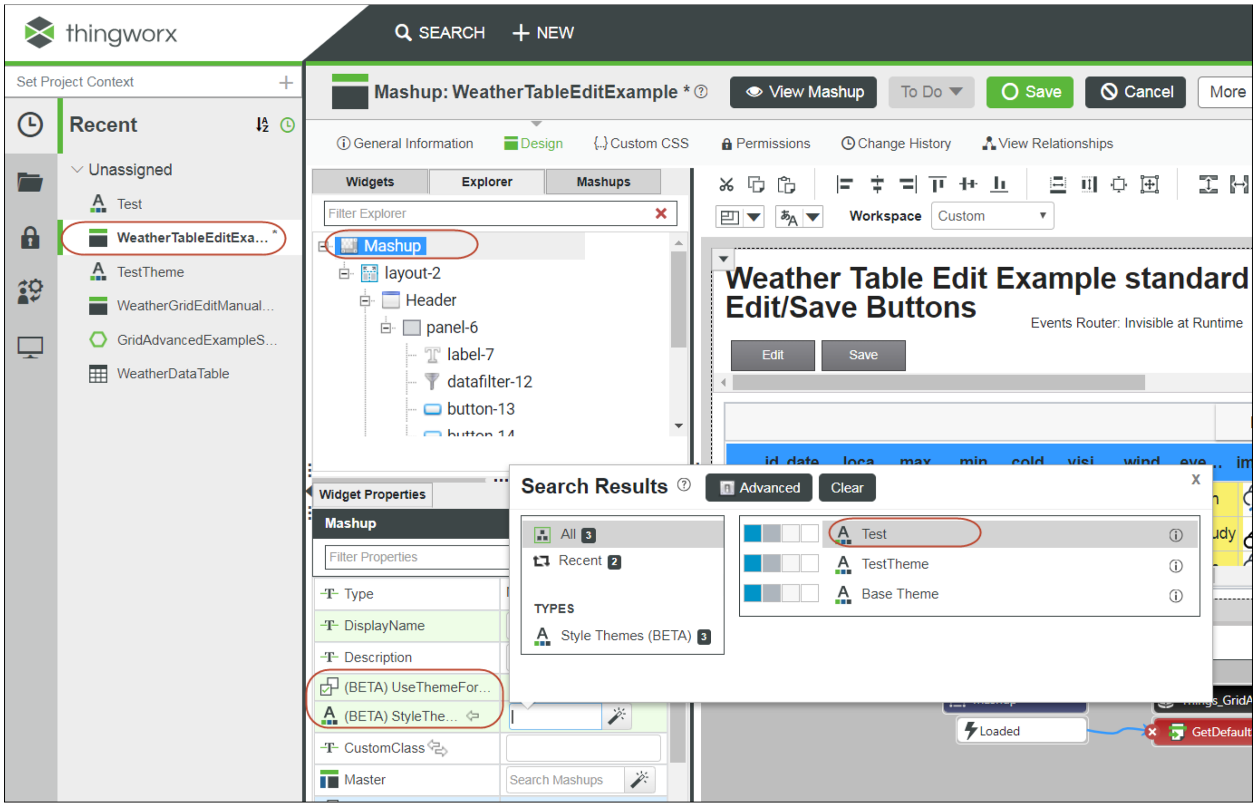


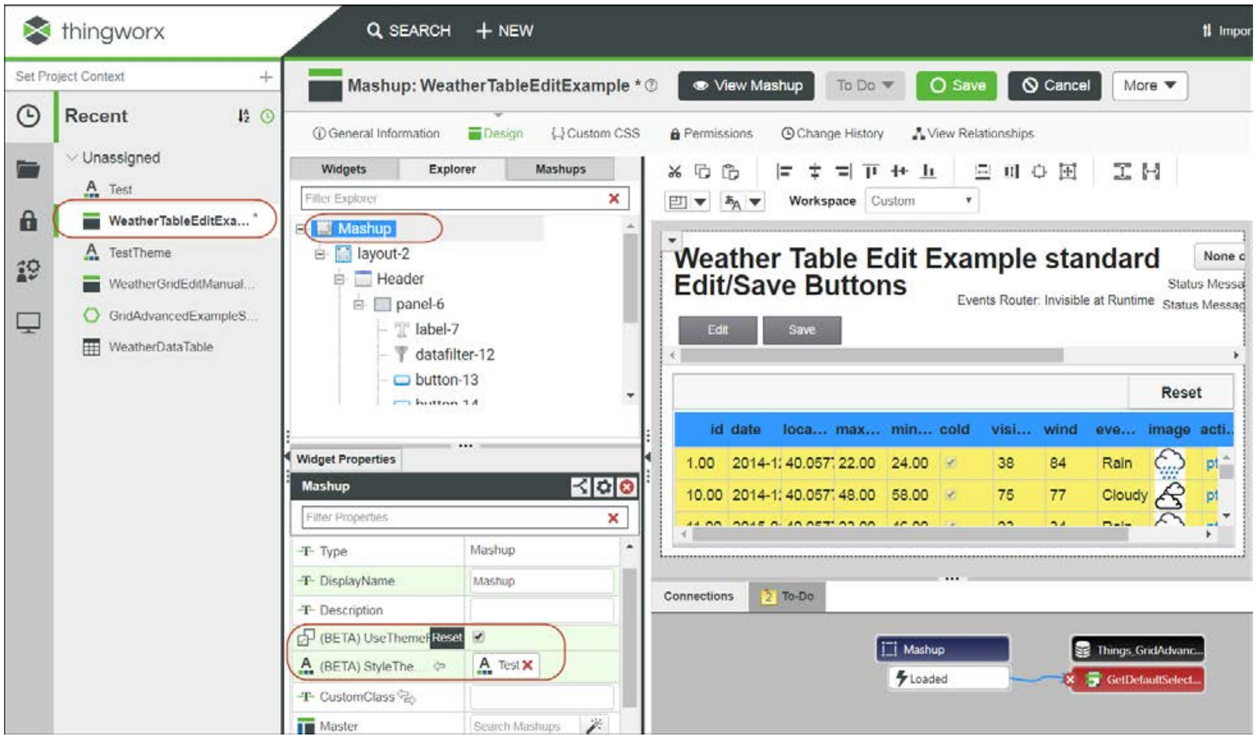
Weather Table Edit Example standard Edit/Save Buttons

id	date	location	max_temp	min_temp	cold	visibility	wind	events	image	actions	info
1.00	2014-12-31 00:00:00.00(	40.0577 : -75.6708	22.00	24.00	✓	38	84	Rain		ptc website	View
10.00	2014-12-22 00:00:00.00(	40.0577 : -75.6708	48.00	58.00	✓	75	77	Cloudy		ptc website	View
11.00	2015-01-11 10:00:00.00(	40.0577 : -75.6708	23.00	46.00	✓	23	34	Rain		ptc website	View
12.00	2015-01-10 10:00:00.00(	40.0577 : -75.6708	11.00	81.00	✓	46	29	Fog		ptc website	View
13.00	2015-01-09 10:00:00.00(	40.0577 : -75.6708	3.00	33.00	✓	65	24	Sunny		ptc website	View
14.00	2015-01-08 10:00:00.00(	40.0577 : -75.6708	51.00	87.00	☐	23	54	Snow		ptc website	View
15.00	2015-01-07 10:00:00.00(	40.0577 : -75.6708	86.00	73.00	☐	10	80	Cloudy		ptc website	View
16.00	2015-01-06 10:00:00.00(	40.0577 : -75.6708	6.00	96.00	✓	78	58	Rain		ptc website	View
17.00	2015-01-05 10:00:00.00(	40.0577 : -75.6708	10.00	29.00	✓	50	54	Fog		ptc website	View
18.00	2015-01-04 10:00:00.00(	40.0577 : -75.6708	84.00	59.00	☐	95	17	Sunny		ptc website	View
19.00	2015-01-03 10:00:00.00(	40.0577 : -75.6708	64.00	68.00	☐	14	30	Snow		ptc website	View

   Go back to the canvas and click **Explorer**> **Mashup**. In the **Widget Properties** panel, find the **UseThemeForHybrids** and **StyleTheme** properties.

3. Click and select the style theme that you created in Step 1.

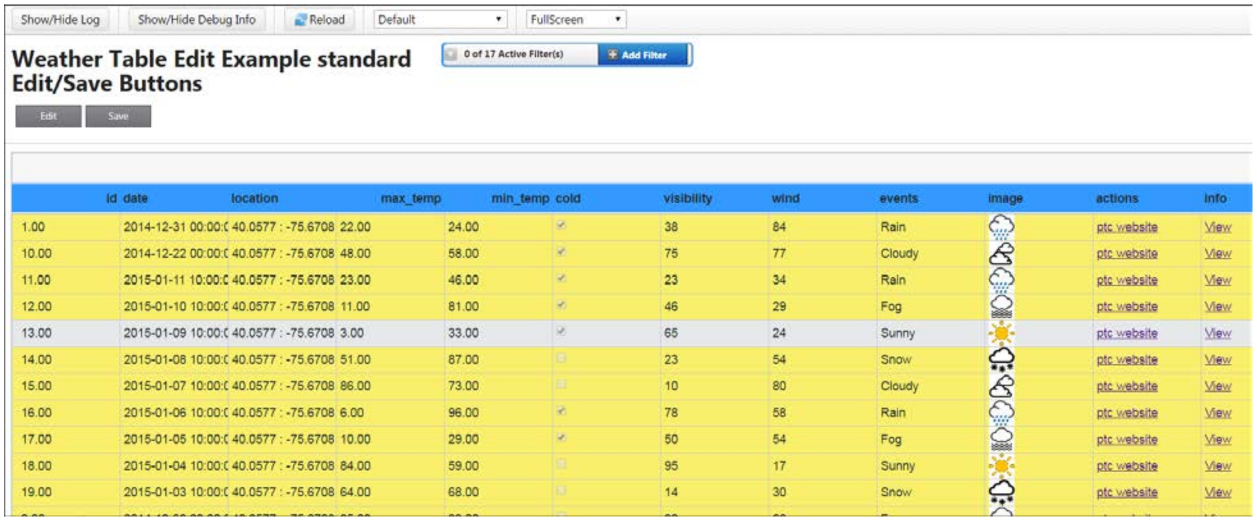




4. Select the **UseThemeForHybrids** property.

5. Click **Save** to save the mashup.

6. Reload the mashup and see that the new style themes are applied to it.



Weather Table Edit Example standard Edit/Save Buttons

id	date	location	max_temp	min_temp	cold	visibility	wind	events	image	actions	info
1.00	2014-12-31 00:00:(	40.0577 : -75.6708	22.00	24.00	✓	38	84	Rain	🌧	ptc website	View
10.00	2014-12-22 00:00:(	40.0577 : -75.6708	48.00	58.00	✓	75	77	Cloudy	☁	ptc website	View
11.00	2015-01-11 10:00:(	40.0577 : -75.6708	23.00	46.00	✓	23	34	Rain	🌧	ptc website	View
12.00	2015-01-10 10:00:(	40.0577 : -75.6708	11.00	81.00	✓	46	29	Fog	🌫	ptc website	View
13.00	2015-01-09 10:00:(	40.0577 : -75.6708	3.00	33.00	✓	65	24	Sunny	☀	ptc website	View
14.00	2015-01-08 10:00:(	40.0577 : -75.6708	51.00	87.00	☐	23	54	Snow	🌨	ptc website	View
15.00	2015-01-07 10:00:(	40.0577 : -75.6708	86.00	73.00	☐	10	80	Cloudy	☁	ptc website	View
16.00	2015-01-06 10:00:(	40.0577 : -75.6708	6.00	96.00	✓	78	58	Rain	🌧	ptc website	View
17.00	2015-01-05 10:00:(	40.0577 : -75.6708	10.00	29.00	✓	50	54	Fog	🌫	ptc website	View
18.00	2015-01-04 10:00:(	40.0577 : -75.6708	84.00	59.00	☐	95	17	Sunny	☀	ptc website	View
19.00	2015-01-03 10:00:(	40.0577 : -75.6708	64.00	68.00	☐	14	30	Snow	🌨	ptc website	View

Column Renderers and Formats

The set of column renderers available for use in the grid widgets is listed below, along with their corresponding formats and some other notes.

Renderer Type	Formats	Notes
Default	N/A	Renders the data as a string when possible.
Number	Select or enter a string containing the format to be used.	JSON configuration service – formats support both % and $ Mashup builder – formats only support $. Both support decimals.
Long	Select or enter a string containing the format to be used.	JSON configuration service – formats support both % and $ Mashup builder – formats only support $. Both support decimals.
HTML	• **raw** – The actual HTML is displayed in the grid cell. • **format** – The HTML is encoded, XSS sanitized, and interpreted by the browser for display. • **unsanitized** – The HTML is encoded but is NOT XSS sanitized before it is interpreted by the browser for display.	The format options listed in the column to the left are for JSON use. In the Mashup Builder, these options are labeled: • Raw (no formatting) • With Formatting • With Formatting, Unsanitized (not-secure) **Important**: When using the unsanitized format, make sure that no user data is exposed in the grid column. Make sure that only application data, created by a developer and free from security vulnerabilities, is shown.

Hyperlink	• **_blank** – The navigation target is a new window or browser tab (depending on the browser).
	• **_self** – The navigation target is the current window or tab.
	• **_parent** – The navigation target is the parent of the iframe.
	• **_top** – The navigation target is the top frame.
		When the hyperlink renderer is selected, a **Link Text** column is also available. Enter the text to be displayed by the link.
		The following is a JSON example for configuring a hyperlink column:
		`“columnFormatter”: {`
		`   “type”: “hyperlink”,`
		`   “format”: “_blank”,`
		`   “params”: {`
		`       “textFormat”: “Click here!”`
		`   }`
		`}`

Imagelink	• **image** – Displays the image at its actual size. • **scaledtowidth** – Scales the image to fit the column width. • **scaledtoheight** – Scaled the image to fit the row height. • **hyperlink** – Displays a link that can be clicked to view the image.	

Renderer Type	Formats	Notes
String	• **full** – Displays the entire text string. • **notext** – Displays no text. • **limitN** – Limits the display of text to the first N characters. Limits are usually unnecessary when using data overflow options.	
Boolean	• **checkbox** – Displays a view-only checkbox in the grid cell. • **text** – Displays text options such as **true** or **false**. • **notext** – Displays no data at all. This option is used for state formatting only.	
Datetime	Follow the links on the right for more information about using the **momentjs** and **jdate** formats.	For more information, see the following: • http://momentjs.com/docs/ • https://github.com/MadMG/moment-jdateformatparser
Integer	Select or enter a string containing the format to be used.	JSON configuration service – formats support both % and $ Mashup builder – formats only support $. Integer does **not** support decimals.

Location	Select or enter a string containing the latitude/longitude/elevation format to be used to identify a location. The format string can be used to truncate the precision of the latitude/longitude/elevation values. When truncated, the values will be rounded up. If no elevation value is included, it will be omitted from the output string.	An icon can be displayed with the location by using a state definition. Configure the state to define when to show the icon, depending on the value of the location string.
		The following is a JSON example for configuring a location column:
		`“columnFormatter”: {`
		`  “type”: “location”,`
		`  “format”: “0.000000”,`
		`  }`
		`}`

NOTE: From 4.2.0 version, the Advanced Grid and Advanced Tree Grid list can render and edit **Image**, **ThingCode**, **Vec2**, **Vec3**, **Vec4**, and **Infotable** basetypes at runtime.

NOTE: Rendering with dynamic data shapes is supported when you provide dynamic configuration. When data in Advanced Grid is populated using dynamic data shapes, provide configuration at runtime to render the values as expected. It can be provided using a service that is bound to **Configuration** of Advanced Grid. This service is like the one which provides data based on data shape name.

Working with a Configuration Service

To configure grids dynamically, either advanced or tree grids, follow the steps below:

1. In ThingWorx Composer, write a JavaScript configuration service that will output results as a JSON object. For more information, see Writing a Configuration Service Script.
2. In the Mashup Builder, where you are creating the grid, add the configuration service as another entity in the right side panel.
3. From the configuration entity in the right panel, under **Returned Data/All Data**, drag **result** to the grid and bind it to the **Configuration** property.

   NOTE: When you bind the configuration service to the grid, most of the properties in the Mashup Builder panel disappear. If the configuration service is unbound, the other properties will be redisplayed.
4. Save and View the completed mashup.

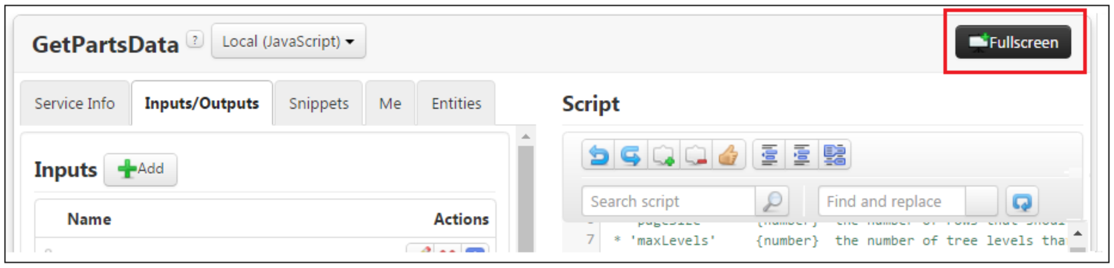
Writing a Configuration Service Script

The configuration script can be written in any of the following ways:

- Create a new service on a Thing in Composer and write original Javascript. Several tabs are available with code snippets and other helpful shortcuts.
- Write a Javascript service in any text editor you prefer and copy and paste it into the script window of a service on a Thing in Composer.
- Modify one of the sample configuration services. To work with sample services, save and import the sample files from Thingworx marketplace. For more information, see Samples File.

To work with one of the imported sample configuration services in ThingWorx Composer:

1. Navigate to **MODELING/Things** and open the Thing called **GridAdvancedExampleServices**.
2. Click **Services** in the left panel and the available sample services are displayed on the right.
3. Select one of the configuration services and click the **Edit** icon to view the script window.
4. Click **Fullscreen** for easier viewing.



```
* 'maxLevels'      {number}  the number of tree levels tha
```

5. Modify the script and save it. For more information, see Configuration Service Parameters.

Configuration Service Parameters

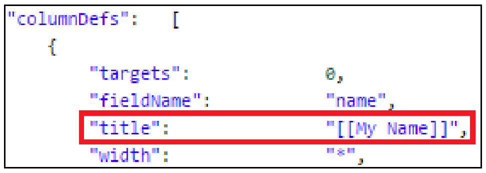
The script for a configuration service contains the following sections of parameters:

- **Columns –** Contains column definitions and some additional properties that define column behavior in the grid, such as column header and multi-column sort order.

  Most of the column definition properties are easy to match with the corresponding properties available in the Mashup Builder. However, the following column definition properties are only available in the Mashup Builder when the data source for the grid is tied to a Data Shape. Then the following properties correspond to similar options in the Configure Grid Columns dialog box:

  - **hidden** – Defines whether a specific grid column is initially visible or hidden from view. End users can hide and unhide the column display in Run time by right-clicking the column header and using the context menu. (Corresponds to the **Show** property in the Mashup Builder/Configure Grid Columns.)
  - **inLayout** – Defines whether a specific grid column can ever be seen by the end user. When set to false, the end user will not see the column and will have no control over its display. However, data in the column can still be used to for state formatting. (Corresponds to the **Exclude** property in the Mashup Builder/Configure Grid Columns.)

  NOTE: Column header titles in the JSON script can be localized by placing a localization token in double square brackets, as shown below. At runtime, the tokenized value is translated.

```
"columnDefs":   [
    {
        "targets":              0,
        "fieldName":            "name",
        "title":                "[[My Name]]",
        "width":                "*",
```

  If you use a tokenized header, but the token does not yet exist in ThingWorx, the column header will display as “???” at runtime. To create or modify tokens in ThingWorx, navigate to **SYSTEM** -> **Localization Tables** and work with the **Localization Tokens** list in the **Default** table. To add a new token to the Default table, you can use the AddLocalizationToken service provided as part of the GridAdvancedExamplesServices Thing.

- **Rows –** Contains row properties, such as default row selection, row height, and row-based state formatting behavior.
- **Styles** – Contains optional style definitions that control the display of the grid, such as background colors, border styles, fonts, and state-specific styles.

  NOTE: Control of font properties is only available when configuring with a service. Font selection is not a property available from the Mashup Builder.

- **Search** – Defines whether global searching is enabled and locates the Search box on the grid.
- **resetButton** – Defines whether or not the grid reset option is enabled and the location of the Reset button on the grid.

Working with Tree Grid Data

Using a Tree-Loading Data Service

In a Tree Grid, the relationships between parent and child nodes of data add complexity to querying and filtering tasks. To simplify the process, most of the functionality is encoded in one JavaScript data service, a sample of which is provided as an attachment to this document. You can add the provided JavaScript code to a service, either entirely or in pieces, to support tree grid features in your own mashups.

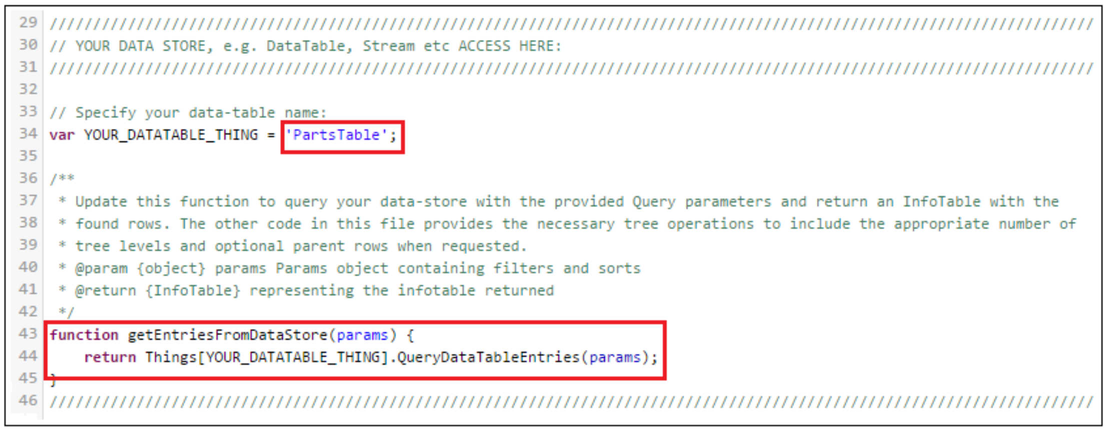
When necessary, you can also convert the API implementations described in the sample data service into a java-based service. Ensure that the input parameter names remain the same and the returned InfoTable contains the correct listing of rows for each API required in the service.

Binding this data service to your grid is required in order to take full advantage of tree grid functionality, such as:

- Loading initial child data, with optional query and data filter parameters.
- Auto-expanding rows according to a specified expansion path (leafID).
- Searching for child data that matches specified query parameters.
- Using a data filter widget to filter for child data that matches specified filter query parameters.

To use the provided **GetPartsData** sample tree-loading data service, it must be slightly customized (to point to the location of your data), added to a Thing in Composer, and bound to the grid. Follow these steps:

1. Use the information in the Samples File section to save the samples and import them into ThingWorx Composer.
2. The sample data service for tree grid functionality is called **GetPartsData**. To find it:
    - Navigate to **MODELING/Things** and open the **GridAdvancedExampleServices** Thing.
    - Click **Services** in the left panel and the available sample services are displayed.
    - Select the **GetPartsData** service and click the **Edit** icon to view the script window.
    - Click **Fullscreen** for easier viewing.
3. In the section of the script called **Your Data Store**, customize the **getEntriesFromDataStore** function so that it points to the location of your child data source (see the figure below).
    - If the source is a data table, only update the name of the table in the YOUR_DATATABLE_THING variable.
    - If the source is a data stream, a data shape, or a third party platform, update the getEntriesFromDataStore function accordingly.



```
/////////////////////////////////////////////////////////////////////////////////////////////////////////
// YOUR DATA STORE, e.g. DataTable, Stream etc ACCESS HERE:
/////////////////////////////////////////////////////////////////////////////////////////////////////////

// Specify your data-table name:
var YOUR_DATATABLE_THING = 'PartsTable';

/**
 * Update this function to query your data-store with the provided Query parameters and return an InfoTable with the
 * found rows. The other code in this file provides the necessary tree operations to include the appropriate number of
 * tree levels and optional parent rows when requested.
 * @param {object} params Params object containing filters and sorts
 * @return {InfoTable} representing the infotable returned
 */
function getEntriesFromDataStore(params) {
    return Things[YOUR_DATATABLE_THING].QueryDataTableEntries(params);
}
/////////////////////////////////////////////////////////////////////////////////////////////////////////
```

4. In the **rows** section of the script, make sure the **parentId** value is ‘**/**’ for any top-level row that does not have a parent row. This value indicates root level and is necessary to ensure that the **GetPartsData** service can properly sort and search your data.

   If you prefer to use a different value to indicate root level, modify the **ROOT_ID_VALUE** parameter at the top of the script. You can use any non-empty string, such as: ‘**/Root**’, ‘**//**’, or a single space ‘ ‘.

5. Save your changes. You can now use the service as is or copy and paste the script to a service on your own Thing in Composer.

6. In the Mashup Builder, where you are creating the tree grid, add the data service as another entity in the right-side panel.

7. From the data service entity in the right panel, under **Returned Data**, drag **All Data** to the grid and bind it to the **Data** or **ChildData** property. Data and Child Data can be connected to the same or different sources.

8. Bind the **Filter** and the **Filter Query** properties to the data service so that all of the sort, search, and filter parameters can be combined and appropriate results can be output.

9. Save and View the completed mashup.

Tree Grid Performance Guidelines

The tree grid widget is designed to support two separate use cases. Before building your own tree grid, consider which of the following scenarios your situation falls into:

- **Use Case 1**: a grid with a fixed amount of data, including 5 or less tree levels and fewer than 1000 total rows of data

- **Use Case 2**: a grid with a growing amount of data, including 5 to 25 tree levels and anywhere from 1000 to 100K total rows of data

Based on these aspects of the depth and size of your data, the use of specific grid features can affect the performance of your grid. In other words, the set of tree grid features that are practical to use will differ depending on the depth and size of your data.

The chart below shows how specific grid features should be used in each use case scenario. As a Mashup developer, determine in advance whether the number of rows in your grid will remain fixed or will grow over time. If the number of rows will remain fixed, you can use any or all of the features listed as supported in the **Use Case 1** column. Otherwise, always opt for **Use Case 2** and limit your use of grid features accordingly.

Grid Feature	Use Case 1 – fixed # of rows	Use Case 2 – growing # of rows
Total Rows	< 1000	> 1000 and < 100K
Tree Levels	< = 5	> 5 and < 25
Preload Levels (maxLevels)	Supported	Supported for 1 or 2 levels
Dynamically Load Nodes	Supported	Must use a dynamic child data-loading service.
Server-side Sorting	Supported	Supported
Server-side Searching	Supports matched rows and parents	Supported for matched rows only, no parents
Server-side Data Filtering	Supported for matched rows and parents	Supported for matched rows only, no parents
Expand All Rows	Supported	Only for client-side preloaded levels
Preserver Row Expansion	Supported	Only for client-side preloaded levels
Default Selected Rows	Supported	Only for client-side preloaded levels
Expand Nodes	Supported (any level)	Only for client-side preloaded levels
Selected Rows	Supported (any level)	Only for client-side preloaded levels
Include Row Expansion Parents	Supported	Not Supported

Row Selection of Dynamically-Loaded Rows

To expand and select rows in a Tree Grid that have not been loaded on the client yet, provide an infotable with at least an **id** column indicated by the **IDFieldName** property that contains the fully-qualified path of row IDs to the selected row. By default, the **:;** character combination is used as the path separator, but you can change it by setting the **IDPathSeparator** property in Mashup Builder or in the JSON config file.

For example to select a row with ID ddd you will have to create an infotable with a column with ID value: `//aaa:;bbb:;ccc:;ddd`.

If you want the Tree Grid to generate an output selected rows infotable with fully-qualified ID paths, set the **IncludeRowExpansionParents** property to true, otherwise it will only use the single leaf ID. When the selected rows infotable is sent to the grid and a listed row has not been loaded yet by the grid, the Tree Grid automatically generates a request to the bound data service. If the **IncludeRowExpansionParents** property is enabled, the following parameters are included:

```
{id: 'aaa', leafId: 'ddd', maxLevels: 25}
```

The grid is requesting all rows from ID aaa to its leaf node ddd to fully create the expanded path to the row with ID ddd.

In JSON configuration, add the following:

```
"treeSettings":
{ // Required for Tree Grid.
“IncludeRowExpansionParents”:true, //{boolean} Fetch parent rows of expanded or
selected rows that are not pre-loaded.
 "IDPathSeparator": ':;' // {string} The ID path separator that is used in ID
paths for the selection of non-loaded rows.
}
```

For more information about the settings, see the **RowSelection** example in the **GridAdvancedExampleEntities-V4.0.xml**. In this example, we are binding an infotable with full path row selections to the right tree grid from the left grid. Therefore, the left grid needs to generate a selected row infotable with fully-qualified paths and not just the leaf ID. To do this, you must set the **IncludeRowExpansionParents** property. Or, you can create a service that generates an infotable with fully-qualified paths to the rows you want to select. The path separator character is configurable by the **IDPathSeparator** property. By default, we use **:;** but you can change it to anything else. In the row selection example, you have to change the setting in both tree grids to match.

Sorting, Searching, and Filtering in Advanced Grid and Tree Grid

Sorting, searching, and filtering your grid data can all be handled through a standard platform query service with a single **Filter** event and a **QueryFilter** parameter. When the **Filter** event is triggered, whether to sort, search, or filter the grid data, the **QueryFilter** parameter ensures that the returned data meets all of the specified conditions.

Ways to query your data for sorting, searching, and filtering:

- Set up a Data Table that contains your data and access it using the standard Platform QueryDataTableEntries API.
- If generating data dynamically, through a data service, use the Query InfoTable function to sort and search data in an InfoTable.

For more information about query parameters, see the Query Parameter for Query Services section of the ThingWorx Help Center.

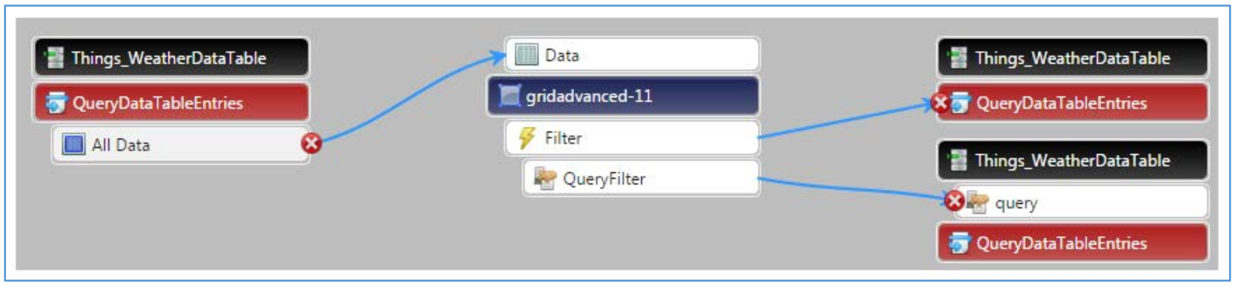
Implement Sorting

1. Set the **EnableSorting** property to true, either by clicking it in the properties panel of the Mashup Builder or defining it in the JSON script of a dynamic configuration service. The **QueryFilter** property and **Filter** event will become available in the properties panel.
2. Bind the **QueryFilter** property to the output query parameter where the data to be sorted is located:
    - If your data is in a table, bind the **QueryFilter** to the query parameter of the QueryDataTableEntries service.
    - If you are generating data via a data service, bind the **QueryFilter** to the queryFilter parameter of the data service.
3. Bind the **Filter** event to the service that will be triggered when sorting begins:
    - If you are using a data table, bind the **Filter** event to QueryDataTableEntries service.
    - If you are generating data via a data service, bind the **Filter** event to the data service.

The example below shows a query parameter with two sort columns applied (name and title):

```
{"maxItems":100000,"query":{"sorts":[{"fieldName":"name","isAscending":true},{"fieldName":
"title","isAscending":true}]}}
```

When these binding steps are complete, the Connections panel should look like the following:



Implement Searching

Searching provides the ability to find a string value in any column in a grid.

1. Set the **EnableGridSearch** property to true, either by clicking it in the properties panel of the Mashup Builder or defining it in the JSON script of a dynamic configuration service. The **GridSearchLocation** property, the **QueryFilter** property, and **Filter** event all become available in the properties panel.
2. Use the **GridSearchLocation** property, either in the Mashup Builder properties panel or in a JSON script, to configure a location for the Search field in the grid. Available options include: top right, top left, bottom right, and bottom left.
3. Bind the **QueryFilter** property to the output query parameter where the data to be searched is located:
    - If your data is in a table, bind the **QueryFilter** to the query parameter of the QueryDataTableEntries service.
    - If you are generating data via a data service, bind the **QueryFilter** to the queryFilter parameter of the data service.
4. Bind the **Filter** event to the service that will be triggered when searching begins:
    - If you are using a data table, bind the **Filter** event to QueryDataTableEntries service.
    - If you are generating data via a data service, bind the **Filter** event to the data service.

The following example shows a search query that searches for an event called **Rain** in all columns:

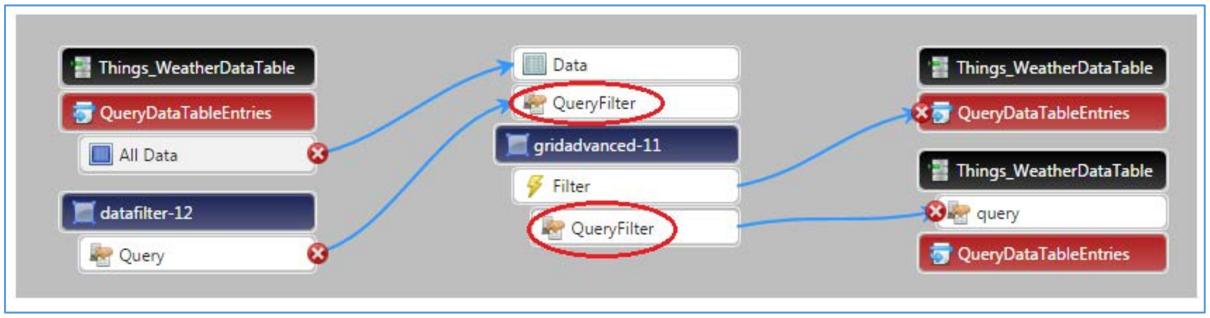
```
{"maxItems":100000,"query":{"filters":{"type":"OR","filters":[{"fieldName":"id","type":"LI
KE","value":"%Rain%"},{"fieldName":"date","type":"LIKE","value":"%Rain%"},{"fieldName":"ma
x_temp","type":"LIKE","value":"%Rain%"},{"fieldName":"min_temp","type":"LIKE","value":"%Ra
in%"},{"fieldName":"cold","type":"LIKE","value":"%Rain%"},{"fieldName":"visibility","type"
:"LIKE","value":"%Rain%"},{"fieldName":"wind","type":"LIKE","value":"%Rain%"},{"fieldName"
:"precipitation","type":"LIKE","value":"%Rain%"},{"fieldName":"events","type":"LIKE","valu
e":"%Rain%"},{"fieldName":"image","type":"LIKE","value":"%Rain%"},{"fieldName":"key","type
":"LIKE","value":"%Rain%"},{"fieldName":"location","type":"LIKE","value":"%Rain%"},{"field
Name":"source","type":"LIKE","value":"%Rain%"},{"fieldName":"sourceType","type":"LIKE","va
lue":"%Rain%"},{"fieldName":"tags","type":"LIKE","value":"%Rain%"},{"fieldName":"timestamp
","type":"LIKE","value":"%Rain%"}]}}}
```

Implement Filtering

To implement filtering in a grid, a Data Filter widget can be added to the Mashup where you are building the grid. A Data Filter widget can only be added to a grid that is bound to a data table based on an underlying Data Shape.

1. From the **Widgets** tab on the left side of the Mashup Builder (above the properties panel), select the Data Filter widget and drag it into your mashup.

2. Bind the output query parameter of the Data Filter widget to the **QueryFilter** property of the Advanced Grid. In this scenario, the **QueryFilter** property is serving both an input and an output function. It receives input from the Data Filter, which is automatically combined with any sorting and searching input that is enabled, and generates a single output for the query parameter.



3. Bind the **QueryFilter** property to the query parameter of the QueryDataTableEntries service of the data table being filtered, sorted, or searched.

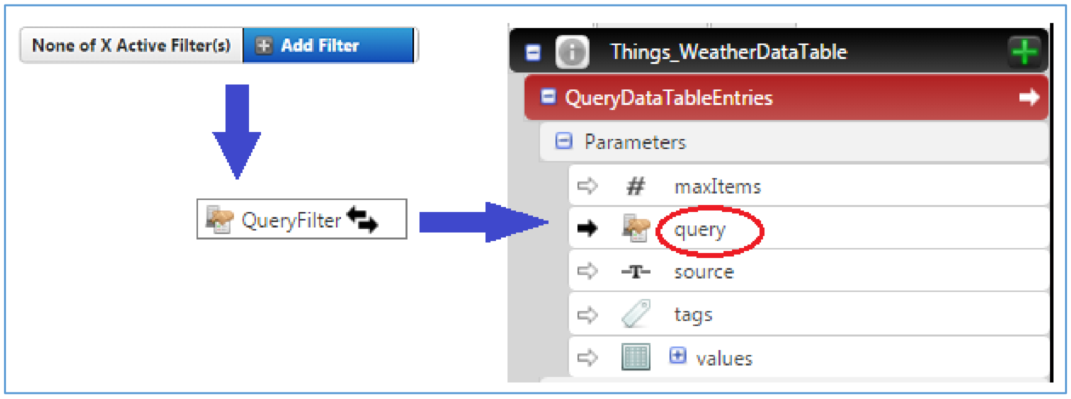
The following example shows a Data Filter query with a single filter parameter, an event value of **Rain**:

```
{"maxItems":100000,"query":{"filters":{"fieldName":"events","type":"LIKE","value":"Rain*"}
}}
```

A filter query can become much more complex when multiple filters are applied, or when filter input is combined with search and sort parameters. The following example shows a combination of sort, search, and filter parameters in a single output query:

```
{"maxItems":100000,"query":{"sorts":[{"fieldName":"id","isAscending":true},{"fieldName":"m
in_temp","isAscending":true}],"filters":{"type":"And","filters":[{"type":"And","filters":[
{"fieldName":"events","type":"LIKE","value":"Rain*"},{"fieldName":"cold","type":"EQ","valu
e":false}]},{"type":"OR","filters":[{"fieldName":"id","type":"LIKE","value":"%21%"},{"fiel
dName":"date","type":"LIKE","value":"%21%"},{"fieldName":"max_temp","type":"LIKE","value":
"%21%"},{"fieldName":"min_temp","type":"LIKE","value":"%21%"},{"fieldName":"cold","type":"
LIKE","value":"%21%"},{"fieldName":"visibility","type":"LIKE","value":"%21%"},{"fieldName"
:"wind","type":"LIKE","value":"%21%"},{"fieldName":"precipitation","type":"LIKE","value":"
%21%"},{"fieldName":"events","type":"LIKE","value":"%21%"},{"fieldName":"image","type":"LI
KE","value":"%21%"}]}]}}}
```

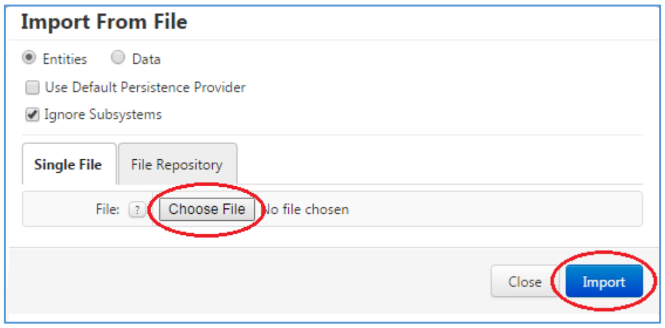
In the advanced grid, when filtering is in use along with sorting and/or searching, the bindings should look like the following diagram when complete:



Samples File

The sample data and configuration entities are provided as a link in the ThingWorx Marketplace and it can be downloaded by the user. To use the samples, follow the steps below to find the samples file and import it into the ThingWorx Composer.

1. Search and find the link for **GridAdvancedExampleEntities-V4.0.xml** from the ThingWorx Marketplace.
2. Click to download and save the example file to your local directory/to a location you can find when you need to import it into Composer.
3. Open ThingWorx Composer.
4. From the **Import** menu at the top of the Composer screen, select **From a File**. The Import From File dialog box opens.
5. Click **Choose File** and navigate to the saved example file and select it.
6. Click the **Import** button on the dialog box. The entities in the file are imported to Composer.



7. When the file import is complete, the entities listed below will be available for you to use in Composer and the Mashup Builder.

   - A Thing called **GridAdvancedExampleServices** that contains both data and configuration services for a set of sample employee data, a set of sample weather data, and a set of sample hierarchical parts data
   - Several Data Shapes (which are tied to services in the **GridAdvancedExampleServices** Thing)
   - Several sample Mashups that include advanced grids or tree grids, all built with the sample data services and some configured with the sample configuration services

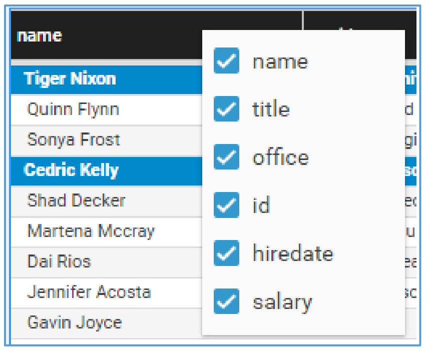
To view a sample configuration service in Composer:

1. Navigate to **MODELING**/**Things** and open the **GridAdvancedExampleServices** Thing.
2. Click **Services** in the left panel and the available sample services are displayed on the right.
3. Select one of the configuration services and click the **Edit** icon to view the script window.

Using the Advanced and Tree Grids in Run Time

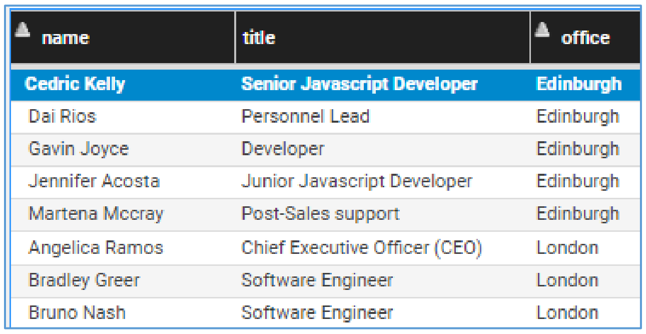
The Advanced Grid and Advanced Tree Grid both include several responsive behaviors that are available to end users in Run time:

- **Hide/Unhide Columns** – Right click in the column header row to display a context menu listing all the columns in the grid. Click to hide or unhide specific columns.



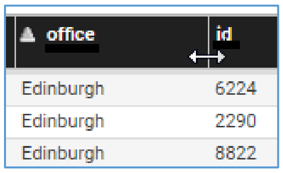
name
Tiger Nixon
Quinn Flynn
Sonya Frost
Cedric Kelly
Shad Decker
Martena Mccray
Dai Rios
Jennifer Acosta
Gavin Joyce

- **Multiple Column Sort** – To sort by multiple columns, click the first column you want to sort on, hold down the **Control** key on your keyboard and click additional columns to sort them in either descending or ascending order. To start over, release the **Control** key and click on a column. The multiple-column sort order will be released.



name	title	office
Cedric Kelly	Senior Javascript Developer	Edinburgh
Dai Rios	Personnel Lead	Edinburgh
Gavin Joyce	Developer	Edinburgh
Jennifer Acosta	Junior Javascript Developer	Edinburgh
Martena Mccray	Post-Sales support	Edinburgh
Angelica Ramos	Chief Executive Officer (CEO)	London
Bradley Greer	Software Engineer	London
Bruno Nash	Software Engineer	London

- **Resize Column Width** – To resize the width of a specific column, hover over the column border until your cursor changes to a double-sided arrow. Then click and drag the column border to whatever width you want.



office	id
Edinburgh	6224
Edinburgh	2290
Edinburgh	8822

NOTE: You can set the column width to less than 20px during design time. However, at runtime when the end-user resizes the column, it cannot be sized to less than 20px to avoid making the column too small accidentally which could prevent it to be able to be resized altogether.

Release Notes for Advanced Grids

Enhancements

- **EXT- 1395** – Supported rendering with dynamic data shapes through dynamic configuration.

Fixed Issues

- **TW-58674** – Fixed an issue with the Advanced Grid and Tree Grid that showed unexpected border when viewed on low resolution devices like an iPad.

  The user needs to add a meta tag to the index.html file in the runtime folder of ThingWorx war (<server-file-path>\ThingWorx\Runtime\index.html). The meta tag is as follows:

  <meta name="viewport" content="width=device-width,height=device-height, initial-scale=1.0">

  This renders the grid on low resolution devices considering the native width and height of the device.

- **EXT-1372** – Fixed an issue with the Advanced Grid and Tree Grid that did not display data properly when dynamic data shapes were used.

Known Issues

- **EXT- 1337** – The user cannot apply **Selected Background** and **Hover Background** theming styles with **RowFormat** style in both Advanced Grid and Advanced Tree Grid widgets. This issue exists only for ThingWorx 8.4.x version and will be fixed in a future release.

- **EXT-1282** – In an IE11 browser, when you resize a column in the grid, the resize control is not released, and it behaves in an undesired manner. The issue is caused by the webcomponet-lite library used by both Grid extensions. This issue exists only for ThingWorx 8.4.x version and will be fixed in a future release.

- **TWX-53316** – In an IE11 browser, the cell validation and row addition features in the Advanced Grid are not working because the cells are not editable. You can only delete rows. This issue exists only for ThingWorx 8.4.x version and will be fixed in a future release.

- **TW-20818** – In a Chrome browser, when a grid contains many rows, blank rows might appear during fast scrolling. This issue is caused by the DPI setting changes Chrome introduced in version 54. To prevent the issue, make sure the Microsoft Windows Display setting on your computer, and the Zoom Level setting in your Chrome browser are both set to 100%.

- **TW-19529** – In a Chrome browser, header cells become slightly offset from data cells.

  Root Cause and Work-around: The Chrome 54 update introduced some minor modifications to Google's browser. Google Chrome now automatically detects your DPI (Dots Per Inch) settings. This change has scaled up the Chrome user interface so that it can appear more zoomed in for users whose Windows DPI settings are above 100%. Slight miscalculations in the grid row height and column width can occur, introducing white space at the top of the grid or slight misalignment between header and grid cells. To resolve the issue, you can do either of the following:

- Change your Windows Display settings for text size from **Medium** (125%) or **Large** to **Smaller** (for example, 100%).
- Change your zoom level to less than 100% in your Chrome browser by using the **Ctrl-Shift minus** keys.