



thingworx®

## Advanced Grid Extension

Version 3.0.5

**Copyright © 2018 PTC Inc. and/or Its Subsidiary Companies. All Rights Reserved.**

User and training guides and related documentation from PTC Inc. and its subsidiary companies (collectively "PTC") are subject to the copyright laws of the United States and other countries and are provided under a license agreement that restricts copying, disclosure, and use of such documentation. PTC hereby grants to the licensed software user the right to make copies in printed form of this documentation if provided on software media, but only for internal/personal use and in accordance with the license agreement under which the applicable software is licensed. Any copy made shall include the PTC copyright notice and any other proprietary notice provided by PTC. Training materials may not be copied without the express written consent of PTC. This documentation may not be disclosed, transferred, modified, or reduced to any form, including electronic media, or transmitted or made publicly available by any means without the prior written consent of PTC and no authorization is granted to make copies for such purposes. Information described herein is furnished for general information only, is subject to change without notice, and should not be construed as a warranty or commitment by PTC. PTC assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. It may not be copied or distributed in any form or medium, disclosed to third parties, or used in any manner not provided for in the software licenses agreement except with written prior approval from PTC.

**UNAUTHORIZED USE OF SOFTWARE OR ITS DOCUMENTATION CAN RESULT IN CIVIL DAMAGES AND CRIMINAL PROSECUTION.**

PTC regards software piracy as the crime it is, and we view offenders accordingly. We do not tolerate the piracy of PTC software products, and we pursue (both civilly and criminally) those who do so using all legal means available, including public and private surveillance resources. As part of these efforts, PTC uses data monitoring and scouring technologies to obtain and transmit data on users of illegal copies of our software. This data collection is not performed on users of legally licensed software from PTC and its authorized distributors. If you are using an illegal copy of our software and do not consent to the collection and transmission of such data (including to the United States), cease using the illegal version, and contact PTC to obtain a legally licensed copy.

**Important Copyright, Trademark, Patent, and Licensing Information:** See the About Box, or copyright notice, of your PTC software.

**UNITED STATES GOVERNMENT RIGHTS**

PTC software products and software documentation are "commercial items" as that term is defined at 48 C.F.R. 2.101. Pursuant to Federal Acquisition Regulation (FAR) 12.212 (a)-(b) (Computer Software) (MAY 2014) for civilian agencies or the Defense Federal Acquisition Regulation Supplement (DFARS) at 227.7202-1(a) (Policy) and 227.7202-3 (a) (Rights in commercial computer software or commercial computer software documentation) (FEB 2014) for the Department of Defense, PTC software products and software documentation are provided to the U.S. Government under the PTC commercial license agreement. Use, duplication or disclosure by the U.S. Government is subject solely to the terms and conditions set forth in the applicable PTC software license agreement.

**PTC Inc., 140 Kendrick Street, Needham, MA 02494 USA**

## Contents

Document Revision History.....	2
Software Change Log .....	2
Prerequisites .....	2
Introduction .....	3
How are the Advanced Grid and Advanced Tree Grid different from the standard grid? .....	3
Key features in both advanced grids.....	3
Features unique to the Tree Grid.....	4
Features of the standard grid not currently available in advanced grids .....	4
Installing the Widgets .....	5
Building Advanced and Tree Grids.....	6
Properties.....	7
Column Configuration from the Context Menu.....	22
Accessing Parameters on the Context Menu.....	22
Column Renderers and Formats .....	25
Working with a Configuration Service .....	27
Writing a Configuration Service Script.....	27
Configuration Service Parameters .....	28
Working with Tree Grid Data .....	29
Using a Tree-Loading Data Service.....	29
Tree Grid Performance Guidelines .....	30
Sorting, Searching, and Filtering in Advanced Grid and Tree Grid .....	32
Implement Sorting .....	32
Implement Searching.....	33
Implement Filtering.....	33
Samples File .....	35
Using the Advanced and Tree Grids in Run Time.....	35
Release Notes for Advanced Grids.....	37
New Functionality .....	37
Fixed Issues .....	37
Known Issues.....	38

## 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. <ol style="list-style-type: none"> <li>1. Revised the Prerequisites table.</li> <li>2. Removed the example entities from the document.</li> <li>3. Revised the Sample File section.</li> <li>4. Revised the bug fixes section.</li> <li>5. Revised the MinRowHeight Property Description.</li> <li>6. Revised the Cell Editing Options section.</li> </ol>

## 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.

## Prerequisites

Prerequisites
ThingWorx 7.4.8 + or 8.0.7 + or 8.1.3+ or 8.2.0+

## 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 different from the standard grid?

Both advanced grids 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 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 do not represent 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

## Advanced Grid Widget Extension

- 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.](#))

## 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

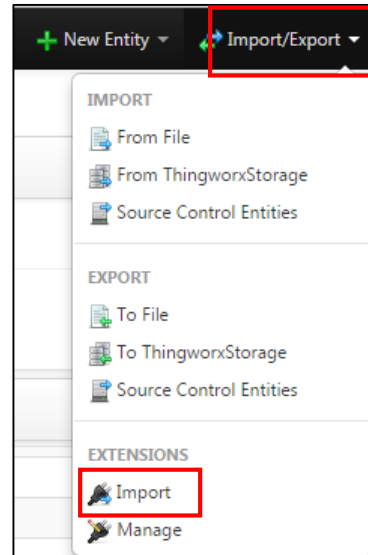
## Features of the standard grid not currently available in advanced grids

- Scroll to the top
- Cell editing – with the exception of Boolean checkboxes which **are** available for run time editing
- Support for all column renderers

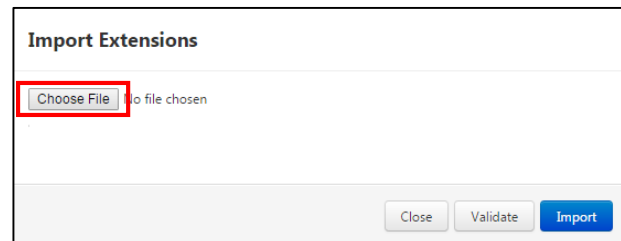
The set of renderers currently supported in the advanced grids is limited to the following: STRING, NUMBER, LONG, LOCATION, BOOLEAN, HTML, HYPERLINK, IMAGELINK, and DEFAULT

## Installing the Widgets

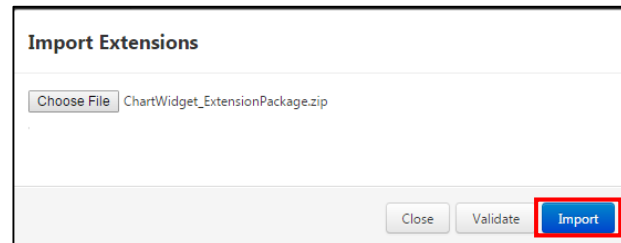
1. From a web browser, launch ThingWorx.
2. Log into ThingWorx as an administrator.
3. Go to **Import/Export > Import**.



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 Advanced and 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 Advance Grid or an Advanced Tree Grid:

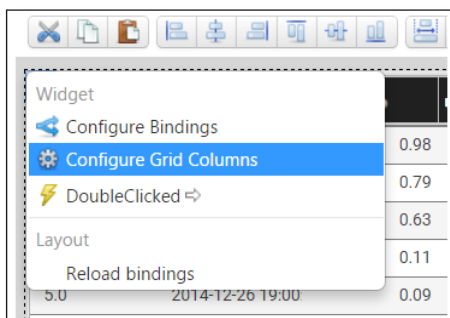
1. Drag and drop one of the following widgets onto a mashup:
  - **Grid-Advanced**
  - **Tree-Grid-Advanced**
2. On the right, add a data source entity and, from the **Returned Data**, drag **All Data** to the grid and bind it to the **Data** property. This binding defines where the data is loaded from,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

Advanced Grid Widget Extension

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
ParentIDFieldName	<p>Identifies the parent ID Field Name. This property is required to create the hierarchical tree structure.</p> <p>IMPORTANT: For the top-level row that does not have a parent, the value should be a forward slash (/).</p>	STRING	parentId	N	TreeGrid only
IDFieldName	<p>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.</p> <p>When no field is specified, or if the specified field does not exist, the grid creates its own internal row ID.</p>	STRING	id	N	Both
HasChildrenFieldName	<p>Specifies the name of the column that indicates whether a row has child data available.</p> <p>To indicate that the row does NOT have children, enter one of the following: '0', 0, 'false', false, empty string, or undefined.</p> <p>Any other value means that the row does have children.</p>	STRING	hasChildren	N	TreeGrid only
Configuration*	<p>If the grid is bound to a configuration service, a filled arrow is displayed: </p> <p>If there is no configuration service, the arrow is unfilled: </p>	STRING	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
IsEditable	<p>Determines whether or not the values in grid cells can be edited when the grid is displayed in run time.</p> <p>NOTE: To edit values in a specific column, the column must also be configured as editable. See the <a href="#">Cell Editing Options</a> on the <b>Column Configuration</b> menu.</p>	BOOLEAN	False	N	Both
EditedTable	<p>A bindable property that indicates an output location for updated values when cells are edited in run time. In order for the updated values to be saved, this property must be bound to an infotable update service.</p> <p>NOTE: Before this property can be used, the IsEditable property must be enabled. In addition, specific columns must be configured as editable. See the <a href="#">Cell Editing Options</a> on the <b>Column Configuration</b> menu.</p>	INFOTABLE	n/a	Y	Both
DefaultSelectedRows	<p>Defines which row numbers are highlighted by default when the grid is displayed. Values can include comma-separated numbers and ranges.</p> <p><b>Example:</b> 1,2,4-5</p> <p>The property can also be defined by a bound service. If service is bound, a filled arrow is displayed: </p> <p>If there is no service, the arrow is unfilled: </p>	STRING	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	<p>NOTES: This property will have no effect if the <b>RowSelection</b> property is set to <b>none</b>. In order to select multiple rows, the <b>RowSelection</b> property must to be set to <b>multi</b>.</p> <p>In a tree grid, default row selection depends on which rows are in view. When the <b>ExpandLoadedRows</b> 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.</p>				
SelectedRows*	<p>Defines, via an INFOTABLE source, which rows are highlighted by default when the grid is displayed.</p> <p>When used in a Tree grid, only the Row ID column is required in order to make row selections but other columns can be included.</p> <p>In an Advanced grid, row selections are handled by binding the output of the <b>SelectedRows</b> parameter in a service to the input <b>SelectedRows</b> property on the grid.</p> <p>This property is bindable in either an output or an input direction so that one entity can control the selection of rows in</p>	INFOTABLE	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	<p>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.</p> <p>For the controlling entity, bind the service as an output INFOTABLE: </p> <p>For the entity being controlled, bind the service as an input INFOTABLE: </p> <p>NOTE: This property will have no effect if the <b>RowSelection</b> property is set to <b>none</b>. In order to select multiple rows, the <b>RowSelection</b> property must to be set to <b>multi</b>.</p>				
IncludeRowExpansionParents*	<p>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.</p> <p>NOTE: Depending on the depth and size of your data, using this property can affect grid performance. See <a href="#">Tree Grid Performance Guidelines</a>.</p>	BOOLEAN	False	N	TreeGrid only
ExpandRows*	<p>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.</p>	INFOTABLE	n/a	Y	TreeGrid only

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
ExpandLoadedRows*	<p>Enables auto-expanding of all preloaded data when the grid is launched.</p> <p>NOTES: Multiple levels of preloaded data must be available.</p> <p>When this property is enabled, it affects the way rows are highlighted when <b>DefaultSelectedRows</b> are defined.</p> <p>This property must be turned off in order for the <b>PreserveRowExpansion</b> property to be effective when enabled.</p>	BOOLEAN	False	Y	TreeGrid only
ExpandRowOnDoubleClick	<p>Allows a row with children to be expanded when double clicked. Rows can also be expanded by clicking on the node icon itself.</p> <p>In a JSON configuration service, the property name is: <b>treeSettings.expandRowOnDoubleClick</b></p>	BOOLEAN	False	N	TreeGrid only
PreserveRowExpansion	<p>Enables row expansion selections to be preserved when the grid is refreshed.</p> <p>When using this property, make sure the <b>maxLevels</b> 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 <a href="#">Using a Tree-Loading Data Service</a>.</p> <p>NOTES: If the <b>ExpandLoadedRows</b> property is enabled, it will overwrite this property and expand all of the preloaded rows. If you want to preserve a specific</p>	BOOLEAN	False	N	TreeGrid only



Advanced Grid Widget Extension

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	<p>expansion of rows, turn off the <b>ExpandLoadedRows</b> when you turn on <b>PreserveRowExpansion</b>.</p> <p>The <b>CookiePersistence</b> property must be enabled in order to preserve row expansion values.</p>				
RowSelection	<p>Controls what row selection is possible to configure. Options: <b>none</b>, <b>single</b>, or <b>multi</b>.</p> <p>NOTE: If the <b>none</b> option is selected, other row selection properties will have no effect.</p>	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*	<p>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 <b>CookiePersistence</b>:</p> <ul style="list-style-type: none"> <li>If both properties are enabled – a user can show/hide columns and those selections persist.</li> </ul>	BOOLEAN	True	N	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	<ul style="list-style-type: none"> <li>• If <b>EnableContextMenu</b> is disabled and <b>CookiePersistence</b> is enabled – a user cannot show/hide columns but previous selections will persist.</li> <li>• If <b>EnableContextMenu</b> is enabled and <b>CookiePersistence</b> is disabled – a user can show/hide columns, but only for the current request.</li> </ul> <p>In a JSON configuration service, the property can be set as a top-level parameter as follows:</p> <pre>var config = {   "enableContextMenu": false,   ... }</pre>				
EnableSorting	<p>Must be enabled for any type of column sorting to take place, including ascending/descending toggling from headers, the <b>MultiColumnSortOrder</b> property, or binding a sorting service. When this option is enabled, the following properties become available in the properties panel:</p> <ul style="list-style-type: none"> <li>• <b>QueryFilter</b> – a bindable filter query for use with a data service</li> <li>• <b>Filter</b> – a bindable event property to trigger a query data service</li> </ul>	BOOLEAN	False	N	Both
MultiColumnSortOrder	Sets a default column sort order. Syntax: column name:order,column name:order	STRING	n/a	N	Both



Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	<p><b>Example:</b> office:asc,title:des</p> <p>Note: <b>EnableSorting</b> must be turned on in order for <b>MultiColumnSortOrder</b> to have any effect.</p>				
EnableGridSearch	<p>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:</p> <ul style="list-style-type: none"> <li>• <b>QueryFilter</b> – a bindable filter query for use with a data service</li> <li>• <b>Filter</b> – a bindable event property to trigger a query data service</li> </ul>	BOOLEAN	False	N	Both
GridSearchLocation	<p>Defines where to place the Search box. This option only becomes available when the <b>EnableGridSearch</b> property is turned on.</p>	STRING	n/a	N	Both
QueryFilter*	<p>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 <b>EnableSorting</b> or <b>EnableGridSearch</b> properties are turned on. It can be set from the properties panel or from the context menu on the grid itself.</p> <p>If you are using a data filter widget in your mashup, the output <b>QueryFilter</b> property can be bound to the input query</p>	QUERY	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	<p>property from either an Advanced or Tree grid widget. The grid combines all of 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: </p> <p>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: </p>				
EnableGridReset	Allows placement of a toolbar with a grid Reset button. Click <b>Reset</b> to clear all grid user settings stored in cookies and return the grid to its default configuration.	BOOLEAN	False	N	Both
EnableFilterEventOnConfigurationChange*	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
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 <b>State Formatting</b> . These row-based rules can be overridden by cell-based state formatting, which is available	STATE FORMATTING	State Formatting	N	Both

Advanced Grid Widget Extension

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	from the <b>Configure Grid Columns</b> option on the grid context menu.				
TableWrapperStyle	Defines the grid background styles.	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
RowBackgroundStyle	Defines a row background style.	STYLE DEFINITION	DefaultRowBackgroundStyle	N	Both
RowAlternateBackgroundStyle	Defines a second row background style for alternate rows.	STYLE DEFINITION	DefaultRowAlternateBackgroundStyle	N	Both
RowHoverStyle	Defines the style of a row when hovered over.	STYLE DEFINITION	DefaultRowHoverStyle	N	Both
RowSelectedStyle	Defines the style of a row when selected.	STYLE DEFINITION	DefaultRowSelectedStyle	N	Both
CellBorderStyle	Defines cell border styles.	STYLE DEFINITION	DefaultCellBorderStyle	N	Both
ToolbarStyle	Defines styles for toolbars when displayed.	STYLE DEFINITION	DefaultToolbarStyle	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

Advanced Grid Widget Extension

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
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: <ul style="list-style-type: none"> <li>• <b>fitted</b> – Text is fitted to the column width and subsequently wraps, even in mid-word.</li> <li>• <b>wrapped</b> – Text wraps to additional lines on white space or a dash.</li> <li>• <b>clipped</b> – Text is cut off at the end of the header cell.</li> <li>• <b>ellipsis</b> – Text is cut off but with an ellipsis (...) to show there is more text.</li> <li>• <b>tooltip</b> – Text is cut off with an ellipsis (...) and full text is displayed in a tooltip.</li> </ul>	STRING	tooltip	N	Both
DataOverflow	Provides options for data cell text that overflows. The same options are available as in the <b>HeaderOverflow</b> 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

Advanced Grid Widget Extension

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
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
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 <b>EnableSorting</b> or <b>EnableGridSearch</b> 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	EVENT	n/a	Y	Both

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	<p>value. Only active when the <b>IsEditable</b> parameter is enabled.</p> <p>This event can be used to change the state of other widgets in the mashup on editing.</p>				
EditCellCompleted	<p>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.</p> <p>Two uses for this event include the following:</p> <ul style="list-style-type: none"> <li>• It can be bound to an infotable update service so that edited values are saved to the <b>EditedGrid</b> infotable.</li> <li>• 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 are saved to the <b>EditedGrid</b> infotable.</li> </ul>	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	BOOLEAN	True	Y	Both

Advanced Grid Widget Extension

Property Name	Description	Base Type	Default Value	Bindable (Y/N)?	Applicable To Grid or TreeGrid?
	<p>bound to the grid, a filled arrow is displayed: ➔</p> <p>If there is no visible service, the arrow is unfilled: ⇨</p>				
SelectedRowsChanged*	<p>A bindable event property that can be used to trigger another widget or service when the user selects or deselects one or more rows in the grid.</p> <p>To use this event property:</p> <ul style="list-style-type: none"> <li>• Set the <b>RowSelection</b> property to <b>single</b> or <b>multi</b>.</li> <li>• Bind the <b>SelectedRows</b> property to another entity (a widget or service).</li> <li>• Bind the <b>SelectedRowsChanged</b> event to the other entity so that it will be triggered when <b>SelectedRows</b> changes.</li> </ul>	EVENT	n/a	Y	Both

\* 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.

## 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.

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.

**Cell Editing in the json configuration** – Add the top level global property mentioned below to enable cell editing in the json configuration.

```
var config = {
  "cellEditingEnabled": true,
  "columns":
    ...
}
};
```

Also add the following on the columns that needs to be edited.

```
...
"ColumnFormatter": {
  "type": "boolean",
  "format": "notext",
  "cellEditor": {
    "enabled": true, // *{boolean} to indicate whether cell
    editing is enable for this column
    "type": 'checkbox' // *{string} the editor type to use
    for the column.
  },
}
...

```

- **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.

The screenshot shows a configuration interface for a column with the ID 'parentId'. It features two tabs: 'Column Format' and 'Column Renderer & State Formatting', with the latter being the active tab. Under the 'Renderer' section, there is a dropdown menu set to 'DEFAULT' with the label 'String formatter' below it. To the right, under the 'Format' section, there is a dropdown menu set to 'No truncation' with the label 'Type of truncation/display rule' below it. Below these sections, there are two radio buttons: 'Fixed Style' (which is selected) and 'State-based Formatting'. At the bottom, there is a search box labeled 'Search Style Definitions' with a magnifying glass icon to its right.

## 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	<ul style="list-style-type: none"> <li>• <b>raw</b> – The actual HTML is displayed in the grid cell.</li> <li>• <b>format</b> – The HTML is encoded, XSS sanitized, and interpreted by the browser for display.</li> <li>• <b>unsanitized</b> – The HTML is encoded but is NOT XSS sanitized before it is interpreted by the browser for display.</li> </ul>	<p>The format options listed in the column to the left are for JSON use. In the Mashup Builder, these options are labeled:</p> <ul style="list-style-type: none"> <li>• Raw (no formatting)</li> <li>• With Formatting</li> <li>• With Formatting, Unsanitized (not-secure)</li> </ul> <p><b>Important:</b> 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.</p>
Hyperlink	<ul style="list-style-type: none"> <li>• <b>_blank</b> – The navigation target is a new window or browser tab (depending on the browser).</li> <li>• <b>_self</b> – The navigation target is the current window or tab.</li> <li>• <b>_parent</b> – The navigation target is the parent of the iframe.</li> <li>• <b>_top</b> – The navigation target is the top frame.</li> </ul>	<p>When the hyperlink renderer is selected, a <b>Link Text</b> column is also available. Enter the text to be displayed by the link.</p> <p>The following is a JSON example for configuring a hyperlink column:</p> <pre> "columnFormatter": {   "type": "hyperlink",   "format": "_blank",   "params": {     "textFormat": "Click here!"   } } </pre>
Imagelink	<ul style="list-style-type: none"> <li>• <b>image</b> – Displays the image at its actual size.</li> <li>• <b>scaledtowidth</b> – Scales the image to fit the column width.</li> <li>• <b>scaledtoheight</b> – Scaled the image to fit the row height.</li> <li>• <b>hyperlink</b> – Displays a link that can be clicked to view the image.</li> </ul>	

Renderer Type	Formats	Notes
String	<ul style="list-style-type: none"> <li>• <b>full</b> – Displays the entire text string.</li> <li>• <b>notext</b> – Displays no text.</li> <li>• <b>limitN</b> – Limits the display of text to the first N characters. Limits are usually unnecessary when using data overflow options.</li> </ul>	
Boolean	<ul style="list-style-type: none"> <li>• <b>checkbox</b> – Displays a view-only checkbox in the grid cell.</li> <li>• <b>text</b> – Displays text options such as <b>true</b> or <b>false</b>.</li> <li>• <b>notext</b> – Displays no data at all. This option is used for state formatting only.</li> </ul>	
Datetime	Follow the links on the right for more information about using the <b>momentjs</b> and <b>jdate</b> formats.	For more information, see the following: <ul style="list-style-type: none"> <li>• <a href="http://momentjs.com/docs/">http://momentjs.com/docs/</a></li> <li>• <a href="https://github.com/MadMG/moment-jdateformatparser">https://github.com/MadMG/moment-jdateformatparser</a></li> </ul>
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 <b>not</b> 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: <pre> "columnFormatter": {   "type": "location",   "format": "0.000000", }           </pre>

## 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 from view. If the configuration service is unbound, the other properties will be redisplayed.

4. Save and View the completed mashup.

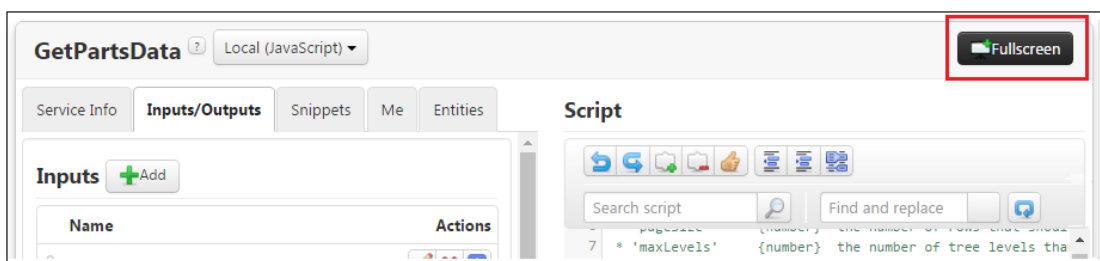
### Writing a Configuration Service Script

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 attached to this document. 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.



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 attached 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.

```
29 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
30 // YOUR DATA STORE, e.g. DataTable, Stream etc ACCESS HERE:  
31 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
32  
33 // Specify your data-table name:  
34 var YOUR_DATATABLE_THING = 'PartsTable';  
35  
36 /**  
37 * Update this function to query your data-store with the provided Query parameters and return an InfoTable with the  
38 * found rows. The other code in this file provides the necessary tree operations to include the appropriate number of  
39 * tree levels and optional parent rows when requested.  
40 * @param {object} params Params object containing filters and sorts  
41 * @return {InfoTable} representing the infotable returned  
42 */  
43 function getEntriesFromDataStore(params) {  
44     return Things[YOUR_DATATABLE_THING].QueryDataTableEntries(params);  
45 }  
46 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

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

## 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.

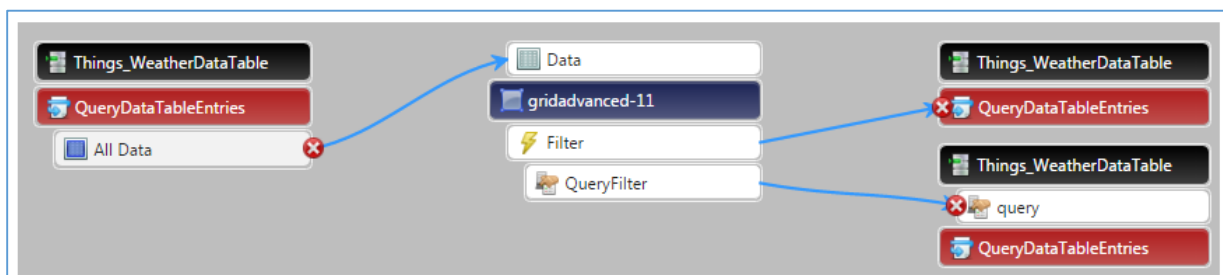
### Implement Sorting

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":"LIKE",
    "value":"%Rain%"},
    {"fieldName":"date","type":"LIKE",
    "value":"%Rain%"},
    {"fieldName":"max_temp","type":"LIKE",
    "value":"%Rain%"},
    {"fieldName":"min_temp","type":"LIKE",
    "value":"%Rain%"},
    {"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",
    "value":"%Rain%"},
    {"fieldName":"image","type":"LIKE",
    "value":"%Rain%"},
    {"fieldName":"key","type":"LIKE",
    "value":"%Rain%"},
    {"fieldName":"location","type":"LIKE",
    "value":"%Rain%"},
    {"fieldName":"source","type":"LIKE",
    "value":"%Rain%"},
    {"fieldName":"sourceType","type":"LIKE",
    "value":"%Rain%"},
    {"fieldName":"tags","type":"LIKE",
    "value":"%Rain%"},
    {"fieldName":"timestamp","type":"LIKE",
    "value":"%Rain%"}]}}}
```

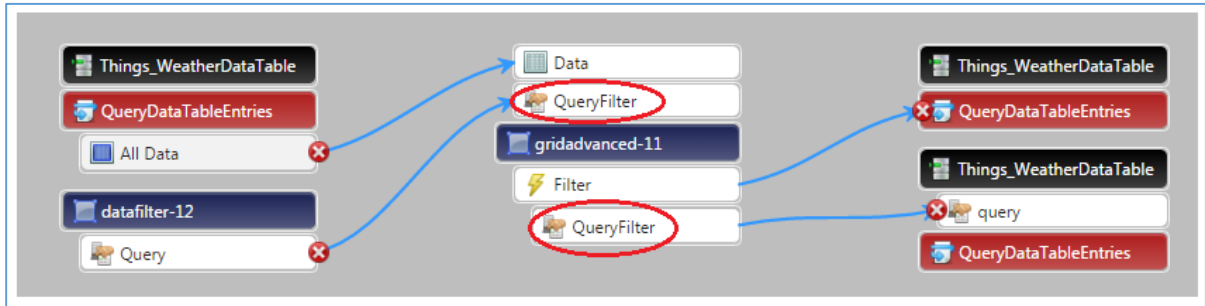
## Implement Filtering

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.

## Advanced Grid Widget Extension

- 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.



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

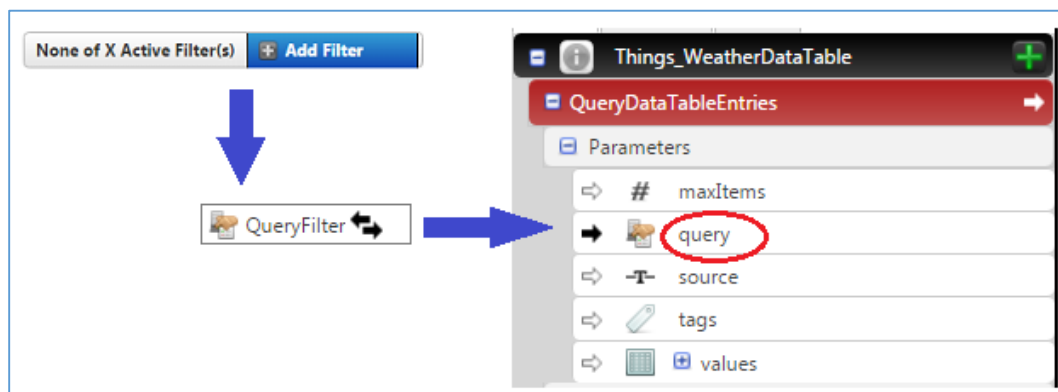
The following example shows a Data Filter query with a single filter parameter, an event value of **Rain**:

```
{ "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": "min_temp", "isAscending": true } ], "filters": { "type": "And", "filters": [ { "type": "And", "filters": [ { "fieldName": "events", "type": "LIKE", "value": "Rain*" }, { "fieldName": "cold", "type": "EQ", "value": false } ] }, { "type": "OR", "filters": [ { "fieldName": "id", "type": "LIKE", "value": "%21%" }, { "fieldName": "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": "LIKE", "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:

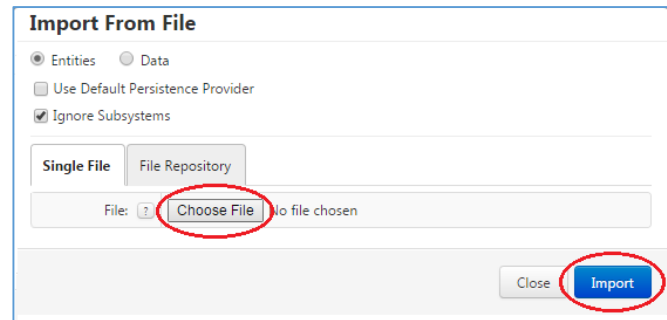


## Samples File

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-V3.0.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:

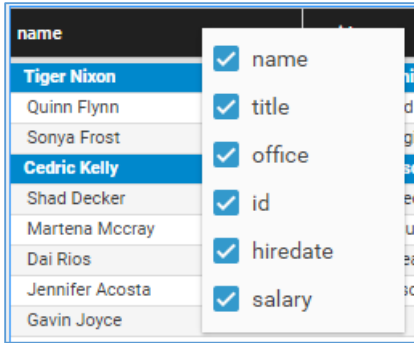
1. Navigate to **MODELING/Things** and open the **GridAdvancedExampleServices** Thing.
2. Click **Services** in the left panel and the available sample services are displayed on the right.
3. Select one of the configuration services and click the **Edit** icon to view the script window.

## 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.

## Advanced Grid Widget Extension



- **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.

A screenshot of a grid widget showing the 'office' and 'id' columns. A double-headed arrow cursor is positioned over the border between the two columns, indicating that the width is being adjusted. The data rows show 'Edinburgh' in the 'office' column and '6224', '2290', and '8822' in the 'id' column.

▲ office	id
Edinburgh	6224
Edinburgh	2290
Edinburgh	8822

## Release Notes for Advanced Grids

### New Functionality

- **EXT-672** – In the Advanced and Tree grids, when you right click on a column header in a run time grid, a grid context menu is displayed which allows specific columns to be shown or hidden. This behavior remains but is now configurable through a new property called **EnableContextMenu**. When this property is enabled (the default), the grid context menu is displayed on right-click. When the property is not enabled, the display of the grid context menu is disabled and right clicking on the grid displays the browser's context menu instead.

The new **EnableContextMenu** property works in conjunction with the **CookiePersistence** property as follows:

- **EnableContextMenu** enabled / **CookiePersistence** enabled – an end user can show/hide columns in a run time grid and those selections will persist in subsequent sessions.
  - **EnableContextMenu** disabled / **CookiePersistence** enabled – an end user cannot show/hide columns in a run time grid but any selections made previously will persist in subsequent sessions.
  - **EnableContextMenu** enabled / **CookiePersistence** disabled – an end user can show/hide columns in a run time grid but those selections will only be in effect for the current request and will not persist in subsequent sessions.
- **EXT-830** – In the Advanced and Tree grids, when a change to a bound configuration service is passed in, a filter event fires to update the data. This behavior remains but is now configurable through a new user-configurable property called **EnableFilterEventOnConfigChange**. When the new property is enabled and a configuration change is made via a service bound to a grid, the filter event will fire. When the property is disabled, the filter event will not fire when the configuration is updated from a service.

### Fixed Issues

- **EXT-814** – The Grid Advanced widget will not display in elements at runtime in the Collection view widget due to non-unique IDs. This has been resolved and ensured that the grid IDs are unique.
- **EXT-806** – When a row expand icon was clicked, after the rows above it were already expanded, the loader gif was displaying incorrectly on the parts below. This issue has been resolved by removing the individual row-loading icon in favor of the standard grid global loading spinner that now shows up when individual rows are expanded.
- **EXT-798** – In the Advanced Grid, the header can be cut off when the text is forced to wrap to two lines. This issue has been resolved and a fix has been provided to handle the combination of vertical and horizontal scroll bars that were causing it.
- **EXT-784** – In the Advanced Grid, the context menu appears behind the modal popup. This issue has been resolved.

- **EXT-693** – In the Advanced Grid, the column selection expands beyond the screen during run time. This issue has been resolved and scroll bars have been added to the context menu if it extends beyond the screen height.

### Known Issues

- **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.
- **TW-20640** – In the 3.0 and earlier versions of both the Advanced Grid and the Advanced Tree Grid, these widgets cannot be used simultaneously with the standard grid or list widgets. Certain style clashes can cause visual display problems for the standard grid or list. The issue will be resolved in a future ThingWorx Core platform release.