Sage 300 Web Screens SDK

Grid Client Interface

May 2022

The MIT License (MIT)

Copyright © 2019 The Sage Group plc or its licensors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contents

[1. Introduction 5](#_Toc71584246)

[2. View List 6](#_Toc71584247)

[2.1 Assumptions and Implementation 6](#_Toc71584248)

[2.2 Configuration 6](#_Toc71584249)

[2.2.1 Init 6](#_Toc71584250)

[2.3 Methods 7](#_Toc71584251)

[2.3.1 refresh 7](#_Toc71584252)

[2.3.2 commit 8](#_Toc71584253)

[2.3.3 dirty (getter) 8](#_Toc71584254)

[2.3.4 dirty (setter) 8](#_Toc71584255)

[2.3.5 allowsInsert (getter) 8](#_Toc71584256)

[2.3.6 allowsInsert (setter) 9](#_Toc71584257)

[2.3.7 allowsDelete (getter) 9](#_Toc71584258)

[2.3.8 allowsDelete (setter) 9](#_Toc71584259)

[2.3.9 showColumn (getter) 9](#_Toc71584260)

[2.3.10 showColumn (setter) 10](#_Toc71584261)

[2.3.11 columnTemplate (getter) 10](#_Toc71584262)

[2.3.12 columnTemplate (setter) 10](#_Toc71584263)

[2.3.13 currentRecord 11](#_Toc71584264)

[2.3.14 filter (getter) 11](#_Toc71584265)

[2.3.15 filter (setter) 11](#_Toc71584266)

[2.3.16 readOnly (getter) 11](#_Toc71584267)

[2.3.17 readOnly (setter) 12](#_Toc71584268)

[2.3.18 columnCount 12](#_Toc71584269)

[2.3.19 fieldName 12](#_Toc71584270)

[2.3.20 columnTitle (getter) 13](#_Toc71584271)

[2.3.21 columnTitle (setter) 13](#_Toc71584272)

[2.3.22 columnEditable (getter) 13](#_Toc71584273)

[2.3.23 columnEditable (setter) 13](#_Toc71584274)

[2.3.24 RefreshCurrentRow(gridName, fieldName) 14](#_Toc71584275)

[2.3.25 UpdateCurrentRow(gridName) 14](#_Toc71584276)

[2.3.26 isEmpty(gridName) 14](#_Toc71584277)

[2.3.27 cancel(gridName) 14](#_Toc71584278)

[2.3.28 clear(gridName) 15](#_Toc71584279)

[2.3.29 refreshByKey(gridName, record) 15](#_Toc71584280)

[2.4 Events 16](#_Toc71584281)

[2.4.1 gridChanged 16](#_Toc71584282)

[2.4.2 gridAfterSetActiveRecord 17](#_Toc71584283)

[2.4.3 gridBeforeDelete 17](#_Toc71584284)

[2.4.4 gridAfterDelete 17](#_Toc71584285)

[2.4.5 gridBeforeCreate 18](#_Toc71584286)

[2.4.6 gridAfterCreate 18](#_Toc71584287)

[2.4.7 gridAfterInsert 18](#_Toc71584288)

[2.4.8 gridAfterLoadData 18](#_Toc71584289)

[2.4.9 columnChanged 19](#_Toc71584290)

[2.4.10 columnBeforeDisplay 19](#_Toc71584291)

[2.4.11 columnDoubleClick 20](#_Toc71584292)

[2.4.12 columnBeforeFinder 20](#_Toc71584293)

[2.4.13 columnFinderFocus 20](#_Toc71584294)

[2.4.14 columnBeforeEdit 21](#_Toc71584295)

[2.4.15 columnStartEdit 21](#_Toc71584296)

[2.4.16 columnEndEdit 21](#_Toc71584297)

[3. Optional Field Control 23](#_Toc71584298)

[3.1 Assumptions and Implementation 23](#_Toc71584299)

[3.2 Configuration 23](#_Toc71584300)

[3.2.1 Init 23](#_Toc71584301)

[3.3 Methods 25](#_Toc71584302)

[3.3.1 refresh 25](#_Toc71584303)

[3.3.2 commit 25](#_Toc71584304)

[3.3.3 dirty (getter) 25](#_Toc71584305)

[3.3.4 dirty (setter) 26](#_Toc71584306)

[3.3.5 allowsInsert (getter) 26](#_Toc71584307)

[3.3.6 allowsInsert (setter) 26](#_Toc71584308)

[3.3.7 allowsDelete (getter) 26](#_Toc71584309)

[3.3.8 allowsDelete (setter) 27](#_Toc71584310)

[3.3.9 filter (getter) 27](#_Toc71584311)

[3.3.10 filter (setter) 27](#_Toc71584312)

1. Introduction

This document is intended to provide information on the client-side grid interface.

There will be three types of grids for the web screens:

* View List
  + sg.viewList
  + Counterpart to the desktop’s **AccpacViewList**
  + Introduced in 2019.2

The only grid to support server-side customization

* Optional Field Control
  + sg.optionalFieldControl
  + Counterpart to the desktop’s **OFCtrl**
  + Introduced in 2019.2

Only supports EDIT mode currently (not SETUP mode)

* Grid Control
  + sg.gridControl
  + Counterpart to the desktop’s **AccpacGrid**
  + Supports a virtual table (i.e. Tax Authorities)

Not yet implemented

1. View List
   1. Assumptions and Implementation

* Client and server are always synchronized
  + For clarity and simplicity, the view list always synchronizes any client change to the server.
  + Though this might create some communication overhead in certain situations, it should work for most cases as a change in one field often results in cascading changes to other fields in the same entity for most ACCPAC business entities.
* All columns in a grid must be defined in the JSON configuration file
  + We don’t support dynamically adding a new column to a view list at runtime, but each column can be hidden or shown programmatically on the fly, and each column’s properties can be modified at runtime as well. Therefore, to mimic the dynamic adding/removing of a column, a programmer can define the column in the JSON file first and show or hide that column later at runtime.
* A view list must be defined in the razor view using the ViewList HTML helper function. The helper function will create several JavaScript objects for the grid that will be consumed by the view list JavaScript library.
  1. Configuration

|  |  |
| --- | --- |
| Name | Description |
| **Init** | Initializes a grid |

* + 1. Init

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.init*** (***gridName***, ***readOnly, updateColumnDefs***) | | |
| **Description** | Initializes a grid | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***readOnly*** | *Optional*. True to make the grid read only |
|  | ***updateColumnDefs*** | Function or function name. To update the column definitions before building the grid columns |
| **Returns** | None | |
| **Example** | sg.viewList.init("receiptGrid", false, receiptUI.updateGridModel); | |

* 1. Methods

|  |  |
| --- | --- |
| Name | Description |
| **refresh** | Read data from the server, refresh the view list, and reset the page number to 1 |
| **commit** | Commit unsaved changes to the server |
| **dirty** | Gets/Sets whether data has changed |
| **allowsInsert** | Gets/Sets the “Insertable” property |
| **allowsDelete** | Gets/Sets the “deletable” property |
| **showColumn** | Gets/Sets the visible property for a column |
| **columnTemplate** | Gets/Sets the template for a column |
| **currentRecord** | Get the current record |
| **filter** | Gets/Sets the filter |
| **readOnly** | Gets/Sets the read-only property |
| **columnCount** | Get the column count |
| **fieldName** | Get the field name of the specified column |
| **columnTitle** | Gets/Sets the title of the specified column |
| **columnEditable** | Gets/Sets the editable property of the specified column |
| **refreshCurrentRow** | Read grid current select row data from server, refresh the grid current row |
| **updateCurrentRow** | Send grid current select row data to server, update the grid current row data in server side |
| **isEmpty** | Return bool value to indicate whether the grid data is empty or not |
| **cancel** | Reset grid status, cancel the last row selection |
| **clear** | Clear the grid data |
| **refreshByKey** | When PageByKey setting is true, reads data from server from specified record |

* + 1. refresh

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.refresh*** (***gridName***) | | |
| **Description** | Read data from the server, refresh the view list, and reset the page number to 1 | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | None | |
| **Example** | sg.viewList.refresh("receiptGrid") | |

* + 1. commit

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.commit*** (***gridName***) | | |
| **Description** | Commit unsaved changes to the server | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | None | |
| **Example** | if (sg.viewList.commit() && $("#frmReceipt").valid()) {… | |

* + 1. dirty (getter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.dirty*** (***gridName***) | | |
| **Description** | Gets whether data has changed | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | true if dirty otherwise false | |
| **Example** | var gridDirty = sg.viewList.dirty("receiptGrid") | |

* + 1. dirty (setter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.dirty*** (***gridName, dirty***) | | |
| **Description** | Sets whether data has changed | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***dirty*** | true/false |
| **Returns** | None | |
| **Example** | sg.viewList.dirty("receiptGrid", false) | |

* + 1. allowsInsert (getter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.allowsInsert*** (***gridName***) | | |
| **Description** | Gets the “Insertable” property | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | true if allows insert otherwise false | |
| **Example** | var allowInsert = sg.viewList.allowInsert("receiptGrid") | |

* + 1. allowsInsert (setter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.allowsInsert*** (***gridName, insertable***) | | |
| **Description** | Sets the “insertable” property. If does not allow insert, the “Add Line” button should be disabled | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***insertable*** | true/false |
| **Returns** | None | |
| **Example** | sg.viewList.allowInsert("receiptGrid", value === type.RECEIPT) | |

* + 1. allowsDelete (getter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.allowsDelete*** (***gridName***) | | |
| **Description** | Gets the “deletable” property | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | true if allows delete otherwise false | |
| **Example** | var allowDelete = sg.viewList.allowDelete("receiptGrid") | |

* + 1. allowsDelete (setter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.allowsDelete*** (***gridName, deletable***) | | |
| **Description** | Sets the “deletable” property. If does not allow delete, the “Delete Line” button should be disabled | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***deletable*** | true/false |
| **Returns** | None | |
| **Example** | sg.viewList.allowDelete("receiptGrid", value === type.RECEIPT) | |

* + 1. showColumn (getter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.showColumn*** (***gridName, columnName***) | | |
| **Description** | Gets the visible property for a column | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***columnName*** | The name of the column (internal column name) |
| **Returns** | true if visible otherwise false | |
| **Example** | var showColumn = sg.viewList.showColumn("receiptGrid", colName) | |

* + 1. showColumn (setter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.showColumn*** (***gridName, columnName, visible***) | | |
| **Description** | Sets the visible property for a column | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***columnName*** | The name of the column (internal column name) |
|  | ***visible*** | true/false |
| **Returns** | None | |
| **Example** | sg.viewList.showColumn("receiptGrid", colName, false) | |

* + 1. columnTemplate (getter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.columnTemplate*** (***gridName, columnName***) | | |
| **Description** | Gets the template for a column | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***columnName*** | The name of the column (internal column name) |
| **Returns** | The template definition | |
| **Example** | var template = sg.viewList.columnTemplate("receiptGrid", cols[i].field) | |

* + 1. columnTemplate (setter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.columnTemplate*** (***gridName, columnName, template***) | | |
| **Description** | Sets the template for a column | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***columnName*** | The name of the column (internal column name) |
|  | ***template*** | Column template (kendogrid format) |
| **Returns** | None | |
| **Example** | sg.viewList.columnTemplate("receiptGrid", cols[i].field, template) | |

* + 1. currentRecord

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.currentRecord*** (***gridName***) | | |
| **Description** | Gets the current record | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | An object containing field name/value pairs for all the fields in the business entity (not only the fields visible in the grid). If no active record, returns Null. | |
| **Example** | var currentRecord = sg.viewList.currentRecord("receiptGrid") | |

* + 1. filter (getter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.filter*** (***gridName***) | | |
| **Description** | Gets the filter | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | The filter, if any | |
| **Example** | var filter = sg.viewList.filter("receiptGrid") | |

* + 1. filter (setter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.filter*** (***gridName, filter***) | | |
| **Description** | Sets the filter | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***filter*** | The filter |
| **Returns** | None | |
| **Example** | sg.viewList.filter("receiptGrid", filter) | |

* + 1. readOnly (getter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.readOnly*** (***gridName***) | | |
| **Description** | Gets the read-only property | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | true if read only otherwise false | |
| **Example** | var readOnly = sg.viewList.readOnly("receiptGrid") | |

* + 1. readOnly (setter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.readOnly*** (***gridName, readOnly***) | | |
| **Description** | Sets the read-only property | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***readOnly*** | true/false |
| **Returns** | None | |
| **Example** | sg.viewList.filter("receiptGrid", true) | |

* + 1. columnCount

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.columnCount*** (***gridName***) | | |
| **Description** | Gets the column count | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | Number of columns (visible or hidden) | |
| **Example** | var colCount = sg.viewList.columnCount("receiptGrid") | |

* + 1. fieldName

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.fieldName*** (***gridName, columnIndex***) | | |
| **Description** | Get the field name of the specified column | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***columnIndex*** | The column index (zero based) |
| **Returns** | The name of the column (internal column name) | |
| **Example** | var fieldName = sg.viewList.fieldName("receiptGrid", i) | |

* + 1. columnTitle (getter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.columnTitle*** (***gridName, column***) | | |
| **Description** | Get the title of the specified column | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***column*** | The name of the column or a zero-based integer |
| **Returns** | Column title | |
| **Example** | var title = sg.viewList.columnTitle("receiptGrid", i) | |

* + 1. columnTitle (setter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.columnTitle*** (***gridName, column, title***) | | |
| **Description** | Sets the template for a column | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***column*** | The name of the column or a zero-based integer |
|  | ***title*** | Column title |
| **Returns** | None | |
| **Example** | sg.viewList.columnTitle("receiptGrid", i, title) | |

* + 1. columnEditable (getter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.columnEditable*** (***gridName, column***) | | |
| **Description** | Get the editable property of the specified column | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***column*** | The name of the column or a zero-based integer |
| **Returns** | true if editable otherwise false | |
| **Example** | var isEditable = sg.viewList.columnEditable("receiptGrid", i) | |

* + 1. columnEditable (setter)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.columnEditable*** (***gridName, column, editable***) | | |
| **Description** | Sets the editable property for a column | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***column*** | The name of the column or a zero-based integer |
|  | ***editable*** | true/false |
| **Returns** | None | |
| **Example** | sg.viewList.columnEditable("receiptGrid", i, isEditable) | |

* + 1. RefreshCurrentRow(gridName, fieldName)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.refreshCurrentRow*** (***gridName, fieldName***) | | |
| **Description** | Refresh grid current select row data from server | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***fieldName*** | The field name |
| **Returns** | None | |
| **Example** | sg.viewList.refreshCurrentRow("receiptGrid", "VALUES") | |

* + 1. UpdateCurrentRow(gridName)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.updateCurrentRow*** (***gridName***) | | |
| **Description** | Update grid current select row data to send to server | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | None | |
| **Example** | sg.viewList.updateCurrentRow("receiptGrid") | |

* + 1. isEmpty(gridName)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.isEmpty*** (***gridName***) | | |
| **Description** | Check whether the grid data is empty | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | true if grid data is empty otherwise false | |
| **Example** | sg.viewList.isEmpty("receiptGrid") | |

* + 1. cancel(gridName)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.cancel*** (***gridName***) | | |
| **Description** | Reset grid status, cancel last select row edit | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | None | |
| **Example** | sg.viewList.cancel("receiptGrid") | |

* + 1. clear(gridName)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.clear*** (***gridName***) | | |
| **Description** | Clear grid data | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | None | |
| **Example** | sg.viewList.clear("receiptGrid") | |

* + 1. refreshByKey(gridName, record)

|  |  |  |
| --- | --- | --- |
| ***sg.viewList.refreshByKey*** (***gridName, record***) | | |
| **Description** | When PageByKey setting is true, reads data from server from specified record | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***record*** | Object containing key fields of the record |
| **Returns** | None | |
| **Example** | sg.viewList.refreshByKey("receiptGrid", { DOCID: “000001”}) | |

* 1. Events

Events are JavaScript call-back functions defined in the JSON file. There are two types of events:

* + Column level events are defined in the column section and are prefixed with “grid”
  + Grid level events are defined outside the column section and are prefixed with “column”

|  |  |
| --- | --- |
| Name | Description |
| **gridChanged** | Raised after field sent to the server and returns an updated record |
| **gridAfterSetActiveRecord** | Raised after a new record is selected as the active |
| **gridBeforeDelete** | Raised before a record is deleted |
| **gridAfterDelete** | Raised after a record has been deleted |
| **gridBeforeCreate** | Raised before the server generates a new record |
| **gridAfterCreate** | Raised after the server generates a new record |
| **gridAfterInsert** | Raised a record is inserted and synced to the server |
| **gridAfterLoadData** | Raised grid data source after grid load data |

|  |  |
| --- | --- |
| Name | Description |
| **columnChanged** | Raised on a changed field before sending to the server |
| **columnBeforeDisplay** | Raised before display of the current field value |
| **columnDoubleClick** | Raised after the left mouse button is double-clicked |
| **columnBeforeFinder** | Raised before the finder is launched |
| **columnFinderFocus** | Raised when the grid column finder get focus |
| **columnBeforeEdit** | Raised before the cell enters edit mode |
| **columnStartEdit** | Raised when the column enters edit mode. The callback function can set various properties for the field editor such as mask, data type, precision, finder, initial value, etc. |
| **columnEndEdit** | Raised when the editor loses focus |

* + 1. gridChanged

|  |  |  |
| --- | --- | --- |
| ***gridChanged*** (***value***) | | |
| **Description** | Raised after field sent to the server and returns an updated record | |
| **Parameters** | ***value*** | The record returned from the server |
| **Returns** | None | |
| **Example** | *{ "gridChanged": "receiptUI.customGridChanged" }* | |

* + 1. gridAfterSetActiveRecord

|  |  |  |
| --- | --- | --- |
| ***gridAfterSetActiveRecord*** (***value***) | | |
| **Description** | Raised after a new record is selected as the active | |
| **Parameters** | ***value*** | The new active record |
| **Returns** | None | |
| **Example** | *{ "gridAfterSetActiveRecord": "receiptUI.customGridAfterSetActiveRecord" }* | |

* + 1. gridBeforeDelete

|  |  |  |
| --- | --- | --- |
| ***gridBeforeDelete*** (***value, event***) | | |
| **Description** | Raised before a record is deleted | |
| **Parameters** | ***value*** | The record to be deleted |
|  | ***event*** | The event object. **Event.PreventDefault()** will cancel the delete, **Event.Proceed()** will let the grid proceed with the normal course (popup a delete confirmation message and proceed with delete if Yes is chosen). |
| **Returns** | None | |
| **Example** | *{ "gridBeforeDelete": "receiptUI.customGridBeforeDelete" }* | |

* + 1. gridAfterDelete

|  |  |  |
| --- | --- | --- |
| ***gridAfterDelete*** (***value***) | | |
| **Description** | Raised after a record has been deleted | |
| **Parameters** | ***value*** | The deleted record |
| **Returns** | None | |
| **Example** | *{ "gridAfterDelete": "receiptUI.customGridAfterDelete" }* | |

* + 1. gridBeforeCreate

|  |  |  |
| --- | --- | --- |
| ***gridBeforeCreate*** (***event***) | | |
| **Description** | Raised before the server generates a new record | |
| **Parameters** | ***event*** | The event object. **Event.PreventDefault()** will cancel the action and **Event.Proceed()** will let the grid to proceed with the normal course |
| **Returns** | None | |
| **Example** | *{ "gridBeforeCreate": "receiptUI.customGridBeforeCreate" }* | |

* + 1. gridAfterCreate

|  |  |  |
| --- | --- | --- |
| ***gridAfterCreate*** (***value***) | | |
| **Description** | Raised after the server generates a new record | |
| **Parameters** | ***value*** | The new record |
| **Returns** | None | |
| **Example** | *{ "gridAfterCreate": "receiptUI.customGridAfterCreate" }* | |

* + 1. gridAfterInsert

|  |  |  |
| --- | --- | --- |
| ***gridAfterInsert*** (***value***) | | |
| **Description** | Raised a record is inserted and synced to the server | |
| **Parameters** | ***value*** | The new record being inserted |
| **Returns** | None | |
| **Example** | *{ "gridAfterInsert": "receiptUI.customGridAfterInsert" }* | |

* + 1. gridAfterLoadData

|  |  |  |
| --- | --- | --- |
| ***gridAfterLoadData*** (***value***) | | |
| **Description** | Raised when grid load data | |
| **Parameters** | ***value*** | The grid data source data |
| **Returns** | None | |
| **Example** | *{ "gridAfterLoadData": "receiptUI.customGridLoadData" }* | |

* + 1. columnChanged

|  |  |  |
| --- | --- | --- |
| ***columnChanged*** (***currentValue, value, event***) | | |
| **Description** | Raised on a changed field before sending to the server | |
| **Parameters** | ***currentValue*** | Thecurrent record stored in an object as field name/value pairs |
|  | ***value*** | The new value for the field |
|  | ***event*** | The event object. **Event.PreventDefault()** will cancel the delete, **Event.Proceed()** will let the grid proceed with the normal course (popup a delete confirmation message and proceed with delete if Yes is chosen). |
| **Returns** | None | |
| **Example** | *{ "columnChanged": "receiptUI.customColumnChanged" }* | |

* + 1. columnBeforeDisplay

|  |  |  |
| --- | --- | --- |
| ***columnBeforeDisplay*** (***value, properties***) | | |
| **Description** | Raised before display of the current field value | |
| **Parameters** | ***value*** | Thecurrent record stored in an object as field name/value pairs |
|  | ***properties*** | Properties for the column:   * Data type * Mask * Precision * Value   If the column already exists in the business entity, the grid will set the ***properties*** object with the field’s data type, mask, precision, and value. A developer can choose to override the properties of this object or leave it as it is. For a custom field, i.e. a field that doesn’t exist in the business entity, the developer must set all the properties for this object |
| **Returns** | None | |
| **Example** | *{ "columnBeforeDisplay": "receiptUI.customColumnBeforeDisplay" }* | |

* + 1. columnDoubleClick

|  |  |  |
| --- | --- | --- |
| ***columnDoubleClick*** (***value, event***) | | |
| **Description** | Raised after the left mouse button is double-clicked | |
| **Parameters** | ***value*** | Thecurrent record stored in an object as field name/value pairs |
|  | ***event*** | The event object. **Event.PreventDefault()** will cancel the delete, **Event.Proceed()** will let the grid proceed with the normal course (enter edit mode if column allows editing). |
| **Returns** | None | |
| **Example** | *{ "columnDoubleClick": "receiptUI.customDoubleClick" }* | |

* + 1. columnBeforeFinder

|  |  |  |
| --- | --- | --- |
| ***columnBeforeFinder*** (***value, finder***) | | |
| **Description** | Raised before the finder is launched | |
| **Parameters** | ***value*** | Thecurrent record stored in an object as field name/value pairs |
|  | ***finder*** | the finder object |
| **Returns** | None | |
| **Example** | *{ "columnBeforeFinder": "receiptUI.customColumnBeforeFinder" }* | |

* + 1. columnFinderFocus

|  |  |  |
| --- | --- | --- |
| ***columnFinderFocus*** (***value, finder***) | | |
| **Description** | Raised when the grid column finder get focus | |
| **Parameters** | ***value*** | Thecurrent record stored in an object as field name/value pairs |
|  | ***options*** | the finder object |
| **Returns** | None | |
| **Example** |  | |

* + 1. columnBeforeEdit

|  |  |  |
| --- | --- | --- |
| ***columnBeforeEdit*** (***value, event***) | | |
| **Description** | Raised before the cell enters edit mode | |
| **Parameters** | ***value*** | Thecurrent record stored in an object as field name/value pairs |
|  | ***event*** | The event object. **Event.PreventDefault()** will cancel the delete, **Event.Proceed()** will let the grid proceed with the normal course (enter edit mode). |
| **Returns** | None | |
| **Example** | *{ "columnBeforeEdit": "receiptUI.customColumnBeforeEdit" }* | |

* + 1. columnStartEdit

|  |  |  |
| --- | --- | --- |
| ***columnStartEdit*** (***value, editor***) | | |
| **Description** | Raised when the column enters edit mode. The callback function can set various properties for the field editor such as mask, data type, precision, finder, initial value, etc. | |
| **Parameters** | ***value*** | Thecurrent record stored in an object as field name/value pairs |
|  | ***editor*** | Editor for the column:   * Data type * Mask * Precision * Value * Finder |
| **Returns** | None | |
| **Example** | *{ "columnStartEdit": "receiptUI.customColumnStartEdit" }* | |

* + 1. columnEndEdit

|  |  |  |
| --- | --- | --- |
| ***columnEndEdit*** (***value, editor***) | | |
| **Description** | Raised when the editor loses focus | |
| **Parameters** | ***value*** | Thecurrent record stored in an object as field name/value pairs |
|  | ***editor*** | Editor for the column:   * Data type * Mask * Precision * Value * Finder |
| **Returns** | None | |
| **Example** | *{ "columnEndEdit": "receiptUI.customColumnEndEdit" }* | |

1. Optional Field Control
   1. Assumptions and Implementation

* Client and server are always synchronized
  + For clarity and simplicity, the optional field control always synchronizes any client change to the server.
  + All columns in an optional field grid a pre-defined like the desktop.
* An optional field control must be defined in the razor view using the ViewList HTML helper function. The helper function will create several JavaScript objects for the grid that will be consumed by the view list JavaScript library.
  1. Configuration

|  |  |
| --- | --- |
| Name | Description |
| **Init** | Initializes a grid |

* + 1. Init

|  |  |  |
| --- | --- | --- |
| ***sg.optionalFieldControl.init*** (***gridName, properties, readOnly***) | | |
| **Description** | Initializes an optional field grid | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***properties*** | ***viewId:*** Roto ID of the underlying optional field ACCPAC business entity (i.e. "IC0377" )  ***filter:*** Type of optional field records (i.e. "LOCATION=2")  ***allowsInsert:*** Controls whether the optional field grid allows insert.  ***allowsDelete***: Controls whether the optional field grid allows delete.  ***type***: The type of the optional field grid. 0 = ENTER, 1 = SETUP. Only “ENTER” is supported as of this release. |
|  | ***readOnly*** | *Optional*. True to mallow editing |
| **Returns** | None | |
| **Example** | sg.optionalFieldControl.init("rptOptionalFieldGrid", { "viewId": "IC0377", "filter": "LOCATION=2", "allowDelete": true, "allowInsert": true, "type": 0 }, false); | |

* 1. Methods

|  |  |
| --- | --- |
| Name | Description |
| **refresh** | Read data from the server, refresh the view list, and reset the page number to 1 |
| **commit** | Commit unsaved changes to the server |
| **dirty** | Gets/Sets whether data has changed |
| **allowsInsert** | Gets/Sets the “Insertable” property |
| **allowsDelete** | Gets/Sets the “deletable” property |
| **filter** | Gets/Sets the filter |

* + 1. refresh

|  |  |  |
| --- | --- | --- |
| ***sg.optionalFieldControl.refresh*** (***gridName***) | | |
| **Description** | Read data from the server, refresh the view list, and reset the page number to 1 | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | None | |
| **Example** | sg.optionalFieldControl.refresh("rptOptionalFieldGrid ") | |

* + 1. commit

|  |  |  |
| --- | --- | --- |
| ***sg. optionalFieldControl.commit*** (***gridName***) | | |
| **Description** | Commit unsaved changes to the server | |
| **Parameters** | ***gridName*** | Optional. The name of the grid. If omitted, commit changes in all optional fields grids |
| **Returns** | true if successful otherwise false | |
| **Example** | sg.optionalFieldControl.commit("rptOptionalFieldGrid ") | |

* + 1. dirty (getter)

|  |  |  |
| --- | --- | --- |
| ***sg. optionalFieldControl.dirty*** (***gridName***) | | |
| **Description** | Gets whether data has changed | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | true if dirty otherwise false | |
| **Example** | var isDirty = sg.optionalFieldControl.dirty("rptOptionalFieldGrid ") | |

* + 1. dirty (setter)

|  |  |  |
| --- | --- | --- |
| ***sg. optionalFieldControl.dirty*** (***gridName, dirtyFlag***) | | |
| **Description** | Sets whether data has changed | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***dirtyFlag*** | true/false |
| **Returns** | None | |
| **Example** | sg.optionalFieldControl.dirty("rptOptionalFieldGrid ", true) | |

* + 1. allowsInsert (getter)

|  |  |  |
| --- | --- | --- |
| ***sg. optionalFieldControl.allowsInsert*** (***gridName***) | | |
| **Description** | Gets the “Insertable” property | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | true if allows insert otherwise false | |
| **Example** | var allowsInsert = sg.optionalFieldControl.allowsInsert("rptOptionalFieldGrid ") | |

* + 1. allowsInsert (setter)

|  |  |  |
| --- | --- | --- |
| ***sg. optionalFieldControl.allowsInsert*** (***gridName, insertable***) | | |
| **Description** | Sets the “insertable” property. If does not allow insert, the “Add Line” button should be disabled | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***insertable*** | true/false |
| **Returns** | None | |
| **Example** | sg.optionalFieldControl.allowsInsert("rptOptionalFieldGrid ", true) | |

* + 1. allowsDelete (getter)

|  |  |  |
| --- | --- | --- |
| ***sg. optionalFieldControl.allowsDelete*** (***gridName***) | | |
| **Description** | Gets the “deletable” property | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | true if allows delete otherwise false | |
| **Example** | var allowsDelete = sg.optionalFieldControl.allowsDelete("rptOptionalFieldGrid ") | |

* + 1. allowsDelete (setter)

|  |  |  |
| --- | --- | --- |
| ***sg. optionalFieldControl.allowsDelete*** (***gridName, deletable***) | | |
| **Description** | Sets the “deletable” property. If does not allow delete, the “Delete Line” button should be disabled | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***deletable*** | true/false |
| **Returns** | None | |
| **Example** | sg.optionalFieldControl.allowsDelete("rptOptionalFieldGrid ", true) | |

* + 1. filter (getter)

|  |  |  |
| --- | --- | --- |
| ***sg. optionalFieldControl.filter*** (***gridName***) | | |
| **Description** | Gets the filter | |
| **Parameters** | ***gridName*** | The name of the grid |
| **Returns** | The filter, if any | |
| **Example** | var filter = sg.optionalFieldControl.filter("rptOptionalFieldGrid ") | |

* + 1. filter (setter)

|  |  |  |
| --- | --- | --- |
| ***sg. optionalFieldControl.filter*** (***gridName, filter***) | | |
| **Description** | Sets the filter | |
| **Parameters** | ***gridName*** | The name of the grid |
|  | ***filter*** | The filter |
| **Returns** | None | |
| **Example** | sg.optionalFieldControl.filter("rptOptionalFieldGrid ", filter) | |