GridBuddy Customization Guide

Table of Contents

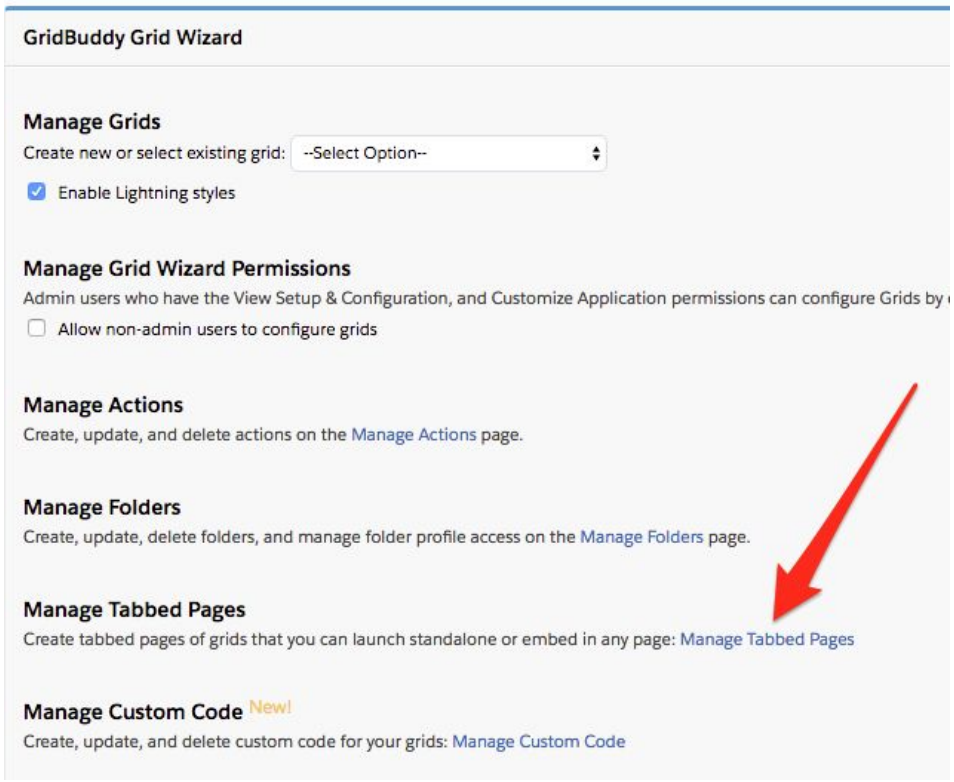
Introduction

GridBuddy is a Force.com web application developed by AppBuddy that sits on top of the Salesforce platform. There are two major components to the application - the Grid Wizard and the Grid. The Grid Wizard enables GridBuddy admins to define and configure Grid views. The Grid is what end-users use to see and manage their data.

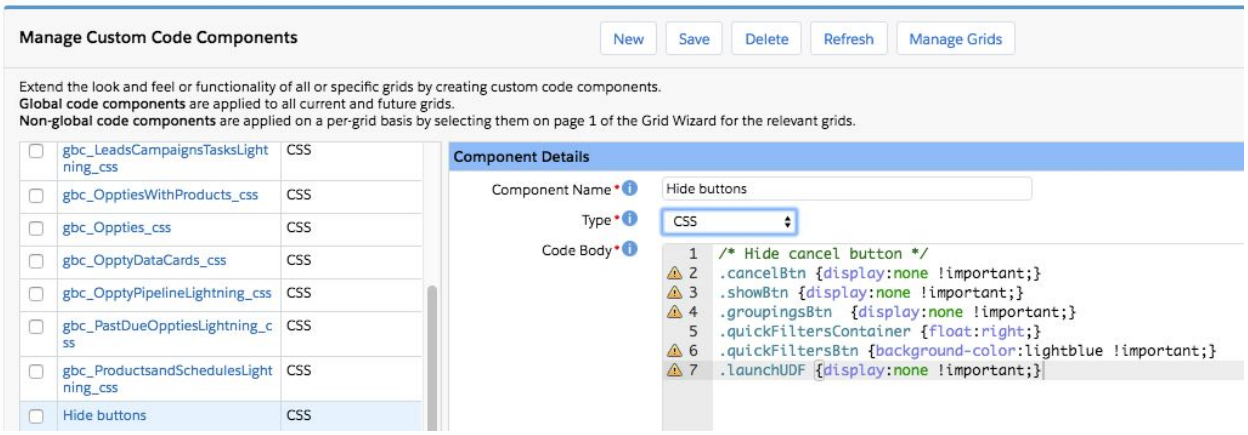
The Grid is highly customizable through GridBuddy's JavaScript and CSS integration API, which enables a range of enhancements possible, from small changes like color coding, to more complex changes, like querying the back-end for information and then constructing custom validation rules that are enforced on the front-end.

Integrating Custom Code

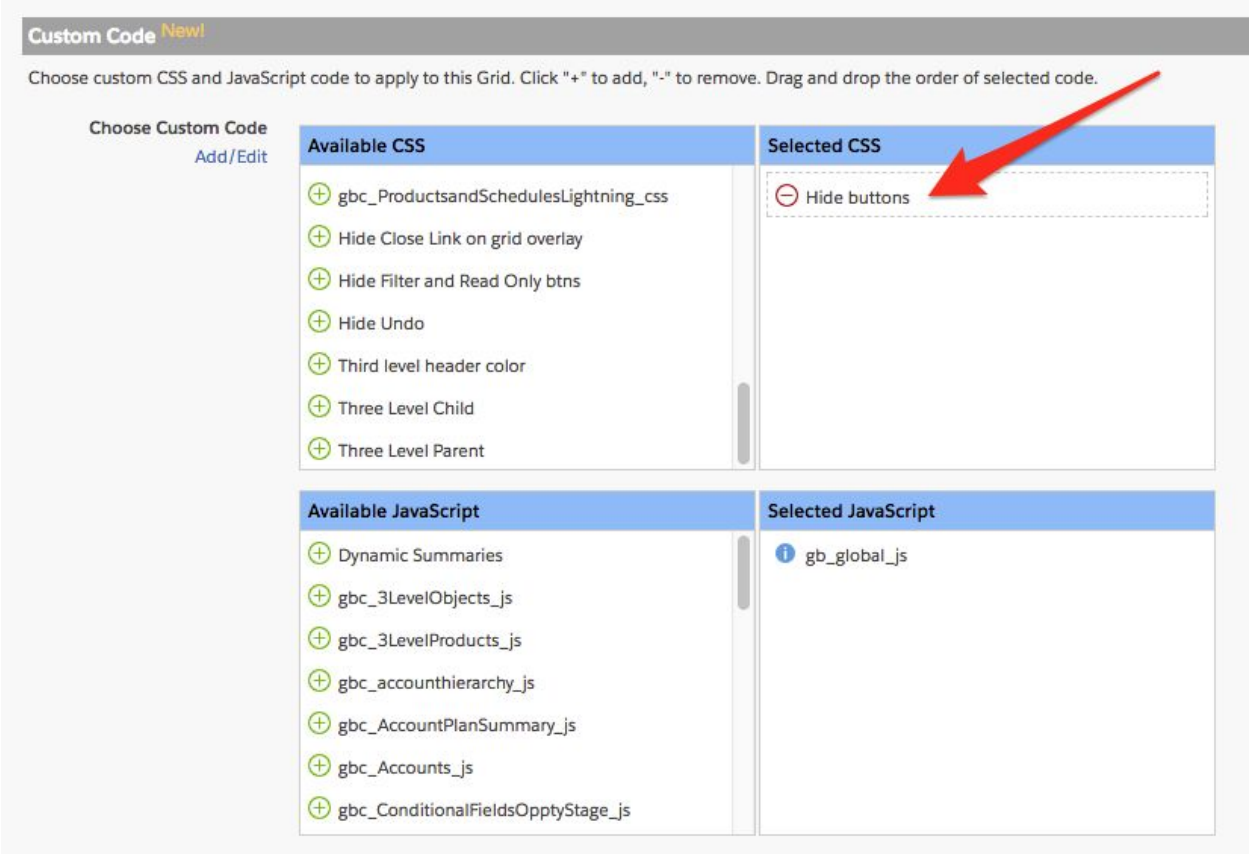
Custom code can be integrated in Grid Wizard > Manage Custom Code from the Grid Wizard landing page:



Here, CSS and JavaScript components can be defined using the text editor on the screen. The code modules can either be standalone CSS or JavaScript snippets, or be defined as Global CSS and Global JavaScript (which inherit to all grids automatically).



Once a code component has been defined, it can be applied to any grid by going to Step 1 of the Grid Wizard and adding the code component to the “Selected CSS” or “Selected JavaScript” section. Note that multiple custom code components can be applied to a single grid.



Deploying Customizations

Depending on how extensive the customization is, and if it involves additional components, such as Visualforce pages or Apex classes, it is typically recommended to develop customizations in Sandbox. Once it’s ready to be deployed, create a Change Set that includes all the custom resources, and then deploy the Change Set to the production org.

If the customizations are minor, low impact, and the client is comfortable with it, the changes can be made directly in Production by creating Static Resources there instead of Sandbox. Once the static resources are present in production, the customizations will be visible immediately on the grids they’re intended for.

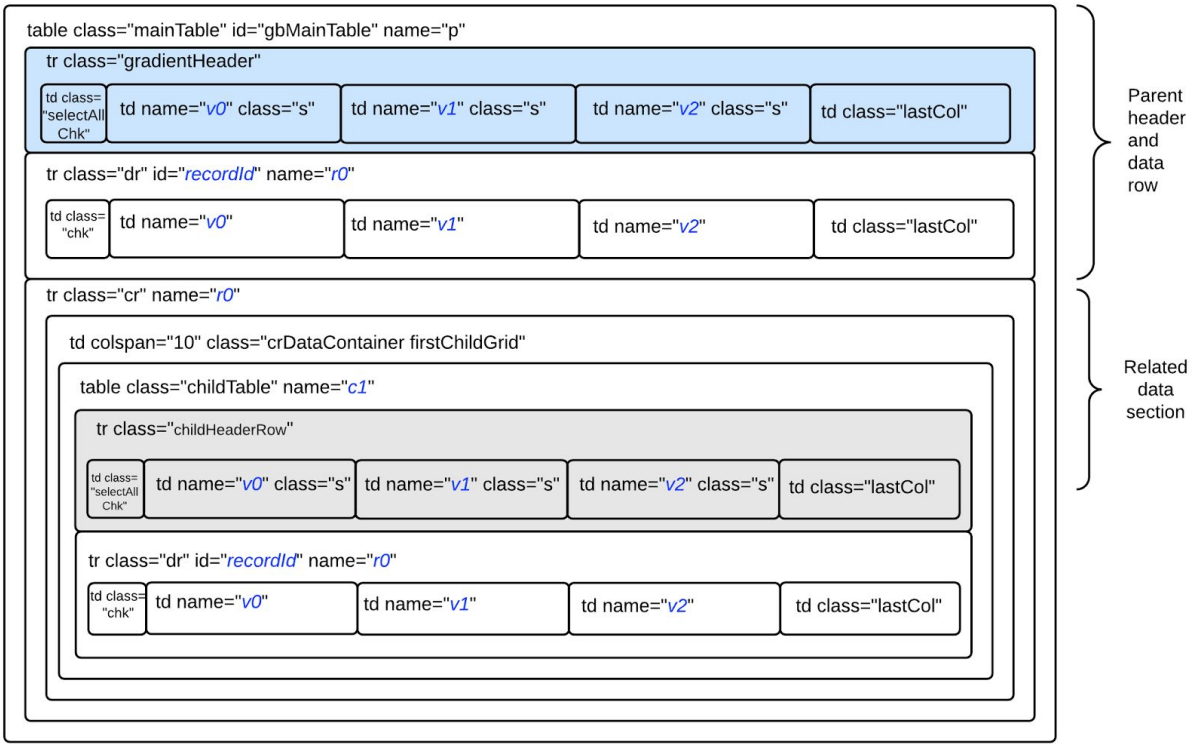
The Grid's Front-End Architecture

The Grid is rendered primarily using JavaScript with the help of jQuery (since GridBuddy version 3.39, GridBuddy uses jQuery version 1.11.2). jQuery can be accessed using the `jQuery` or `jq` namespace.

There are certain GridBuddy global JavaScript variables and objects that are available to use for customizations and provide access to the grid meta data and record data. These global objects can be manipulated before the Grid is done rendering to affect the display of existing records. There are also strategies for manipulating new records that appear after the Grid has loaded, for example, when creating new data.

JavaScript and CSS changes should take advantage of the various selectors available throughout the Grid, for maintenance purposes. For example (this is not an exhaustive list):

1. Grid data rows (tr tag) have the class "dr", short for **d**ata **r**ow
2. Grid data rows (tr tag) for new records have the class "nr", short for **n**ew **r**ow
3. Grid data rows (tr tag) have an "id" attribute populated with the Salesforce record id
4. Grid header and data columns (td) have a dynamic name attribute corresponding to the field's order within the row. This name value corresponds to a property available in one of the GridBuddy global JavaScript objects (MetaCol) that exposes the Grid's meta data.
5. Related sections are wrapped in a row (tr tag) with the class "cr", short for **c**hild **r**ow
6. Related data rows (tr tag) are wrapped in a table with the class "childTable", and a name attribute that's dynamic corresponding to the order of the child object amongst all child objects on the grid
7. There are classes specific to data type widgets and more.



Main GridBuddy JavaScript Objects

GridInfo

Every object on the grid, whether parent or child, has an equivalent GridInfo object created for it. It contains the object's metadata needed for various grid functions, including CRUD access, list of fields represented as MetaCol objects, etc.

Some commonly used properties on a GridInfo instance:

- gridName - object label
- fullyQualifiedObjectName - gridbuddy formatted object API name including the foreign key to the parent, in case the object is a child, unrelated, or junction
- gridApiName - object API name
- metaColumns - array list of MetaCol objects for this object
- isParentInfo - returns true if the gridId property is 'p'
- objId - identifier for the object's table names
- isDeletable
- isCreatable
- isUpdateable

MetaCol

Every field on the grid has an equivalent MetaCol object created for it. It contains the field's metadata, including CRUD access, identifier information, data type functions, etc.

Some commonly used properties and functions on a MetaCol instance:

- colIndex
- fieldName - API name
- fieldLabel - label
- fieldId - used on the name attribute of cells (TDs)
- colDataType
- readOnly
- createable
- updateable
- summaryType
- aggregable
- isTypeID()
- isTypeBoolean()
- isTypeText()
- isTypeTextArea()
- isTypeURL()
- isTypeMultiPicklist()
- isTypeSinglePicklist()
- isTypePicklist()
- isTypeCurrency()
- isTypeDouble()
- isTypeInteger()
- isTypePercent()
- isTypeEmail()
- isTypePhone()
- isTypeDate()
- isTypeDateTime()
- isTypeReference()
- isRecordTypeField()
- isReadOnly() - returns true if grid and field are read-only

Common Global JavaScript Variables

Variable	Type	Description
currentLocale	String	Locale code, empty if US
filteringByParent	Boolean	Indicates if the grid is being filtered by the parent ID. If true there will be an ID URL parameter, and the grid will hide the parent

		record and only show the related records.
filterParentField	String	If the grid is being filtered by fpf / fpv parameters, this indicates the field API name that's part of the filter
ownerId	String	ID of the current User.
multiCurrency	Boolean	Indicates if the org has multi-currency enabled.
sId	String	Session ID
localeDigitSep	String	The thousands separator character based on the locale.
localeDecimalSep	String	The decimal separator character based on the locale.
gridInfoMap	Object (Map)	Holds a map of GridInfo objects, where the key is the object key and value is the GridInfo object instance. The object key is: p = parent object 1-4 = related (child) object in sequential order
profileId	String	18-character user's profile ID
profileId15Char	String	15-character user's profile ID
userId	String	User's ID (since v4.10)

Common JavaScript Functions

Function	Return Type	Description
getDataTable(pTableName)	jQuery object	Returns the table with the specified name as a jQuery object
getFirstDataRow(pDataTable)	jQuery object	Returns the first data row of the table, versus the first row which might be the header.
getParentGridInfo()	GridInfo	Returns the GridInfo object for the parent object
getChildGridInfo(objectName)	GridInfo	Gets the child GridInfo object based on the specified object API name.
getRowId(pCellElement)	String	Returns the Id attribute of the row the specified cell belongs to
isFieldCell(pCellName)	Boolean	Returns true if the spcified cell is a field cell, as opposed to the "select" cell or last column cell

		which is a filler
getMetaCol(pJqueryElem, pGridInfo)	MetaCol	Finds the MetaCol object corresponding to a field on the grid using the specified jquery element (pJqueryElem), which should be an element within a cell (TD), and the GridInfo object to which the field belongs.
getMetaColByCellName(pCellName, pGridInfo)	MetaCol	Finds the MetaCol using the cell's (TD) name and GridInfo
getMetaColByColIdx(pGridInfo, metaColIdx)	MetaCol	Finds the MetaCol based on the GridInfo and specified column index
getMetaColByFieldName(pGridInfo, fieldName)	MetaCol	Finds the MetaCol based on the GridInfo and field's API name
getGridInfo(pJqueryElem)	GridInfo	Finds the GridInfo object using the specified jquery element, which should be within the object's table
getGridInfoByName(gridTableName)	GridInfo	Finds the GridInfo object using the object's table's name attribute, which is mapped to GridInfo's objId attribute.
getGridInfoByApiName(objectApiName)	GridInfo	Finds the GridInfo object using the object's API name, which is mapped to the GridInfo's gridApiName attribute.
GBRowHelper.getParentCol(jQueryElem)	jQuery object	Finds the enclosing parent cell (TD).
GBRowHelper.getParentRow(jQueryElem)	jQuery object	Finds the enclosing parent row (TR).
GBRowHelper.getParentRowForChildRow(childRow)	jQuery object	Finds the parent object row (TR) for the given related child row (TR).
GBRowHelper.getParentTable(jQueryElem)	jQuery object	Finds the enclosing parent table.
GBHelpers.getParamValue(paramName)	String	Finds the value of the URL parameter name, or blank if the parameter isn't present.

Customization Examples

Hide “View Record Detail” / “Open Record Detail” links

Type: CSS

```
/* hide view and open record detail actions */
a.a-open, a.a-open-same-win {display:none !important}
```

`a.a-open` and `a.a-open-same-win` are selectors that identify the record detail actions in the record-level action menu.

Color Coding Fields

(Use the Conditional Formatting feature on Grid Wizard 2 where possible instead of using custom code)

Type: CSS

```
/*Copyright © 2010 Primal Cause, Inc. All rights reserved.*/

/* define the CSS rule to color code the marked cells */
.gb-marked-col {
	background-color: red !important;
}
```

Type: JavaScript

```
/*Copyright © 2010 Primal Cause, Inc. All rights reserved.*/
/*
	Color codes a set of fields that may or may not exist in the grid. The
actual color value should be defined in the '.gb-marked-col' CSS rule.
	This example uses the Account, Contact, Task, and Lead objects.
*/

(function() {

	// define the parent object and the related fields
	var objNameToFields = {
		'Account': ['Effective_Date__c'],
		'Contact': ['FirstName', 'LastName'],
		'Task': ['Priority'],
		'Lead': ['Status']
	};
```

```
	colorCodeFields();

	// adds the 'gb-marked-col' class to the listed fields
	function colorCodeFields() {
		// loop through each object
		jq.each(objNameToFields, function(objName, fields) {
			// select the object we'll be working with
			var gridInfo = getGridInfoByApiName(objName);
			if (!gridInfo) {
				return;
			}
			var gridRows = jq('table[name="' + gridInfo.objId + '"]
tbody > tr.dr');

			// loop through each field to mark the grid column
			jq.each(fields, function(i, fieldName) {
				// obtain the column meta data
				var colInfo = getMetaColByFieldName(gridInfo,
fieldName);
				if (!colInfo) {
					return;
				}
				// add the marker class to the cells
				var cells = gridRows.find('td[name="' +
colInfo.fieldId + '"]');
				cells.addClass('gb-marked-col');
			});
		});
	}
})();
```

`GridInfo` is one of the global GridBuddy JavaScript objects and contains meta data for the objects configured on the grid. `getGridInfoByApiName` and `getMetaColByFieldName` are functions within the Grid's JavaScript logic.

Refresh Parent Detail After Embedded Grid Save or Delete

Type: JavaScript

```
/**
 * Copyright © 2010 Primal Cause, Inc. All rights reserved.
 * Redirects to the id specified in the "fpv" parameter, or the "id" parameter after
grid save or delete
 * Note: this function needs to be called immediately, not on document ready.
 */
```

```
(function refreshParentDetail() {
	var refreshAfterSave = true,	// change to false if you don't want to
refresh the parent after save
		refreshAfterDelete = true, // change to false if you don't want to
refresh the parent after delete
		saveSuccessful = (GBHelpers.getParamValue('us') == '1'),
		deleteSuccessful = (GBHelpers.getParamValue('dsrc').length > 0),
		isFilteringByParentId = (filteringByParent == true),
		hasValidFilterByParentField = (GBHelpers.getParamValue('fpf') != ''),
		parentDetailId;

	if (isFilteringByParentId) {
		parentDetailId = GBHelpers.getParamValue('id');

	} else if (hasValidFilterByParentField) {
		parentDetailId = GBHelpers.getParamValue('fpv');
	}

	if (isFilteringByParentId || hasValidFilterByParentField) {
		if ((refreshAfterSave && saveSuccessful) || (refreshAfterDelete &&
deleteSuccessful)) {
			// reload the detail page
			top.location.href = '/' + parentDetailId;
		}
	}
})();
```

The `filteringByParent` variable is global and comes from the Grid page. `GBHelpers.getParamValue` is a function within the Grid's JavaScript logic.

Defaulting Fields with Hard Coded and Dynamic Values

This demonstrates defaulting a text field to a hardcoded value, as well as querying a lookup field value for defaulting another field when creating new records.

Type: JavaScript

```
/**
 * In Edit mode, set the default Sales Org to 'AT01', and the Agent to the Soldto
Account's Agent on the Quote object
 */
jq(document).ready(function(){
	if (readOnlyGrid==false) {

		var simpleFieldsToDefault = {'Sales_Org__c': 'AT01'},
			soldToAccountApiName = 'Soldto_Account__c',
			agentApiName = 'Agent__c',
			// key = field api name, value = column name
			fieldsForAgentDefault = {'Soldto_Account__c':-1, 'Agent__c':-1};

		for (var gridInfoKey in gridInfoMap) {
			var gridInfoObj = gridInfoMap[gridInfoKey];

			if (gridInfoObj.gridApiName == 'eQuote__c') {
				var thisCol;
```

```
                        // loop through the meta cols now and set the default for
the specified fields
                        for (var i=0; i < gridInfoObj.metaColumns.length; i++) {
                              thisCol = gridInfoObj.metaColumns[i];

                              if (simpleFieldsToDefault[thisCol.fieldName]) {
                                    // set the new default value
                                    setDefaultValueForField(thisCol,
gridInfoObj, simpleFieldsToDefault[thisCol.fieldName]);

                              } else if
(fieldsForAgentDefault[thisCol.fieldName]) {
                                    // set the field's column name
                                    fieldsForAgentDefault[thisCol.fieldName] =
thisCol.fieldId;
                              }
                        }
                  }
            }

            // setup the default behavior on new records for the Agent lookup
field, which is based on the Soldto Account's Agent field
            jq('#gbMainTable').on('change', 'tr.nr
td[name="'+fieldsForAgentDefault[soldToAccountApiName]+'"] input', function() {

                  var thisInput = jq(this),
                        accountId = thisInput.attr('name'),
                        thisRow = thisInput.parents('tr.nr:first'),
                        agentInput =
thisRow.find('td[name="'+fieldsForAgentDefault[agentApiName]+'"] input');

                  if (accountId !== undefined && agentInput.val().length == 0) {
                        // Soldto Account has been specified and Agent has not
been set
                        // get the default Agent based on the Account
                        // note, this must be a JSONP request due to cross-domain
security issues, since the GridBuddy package is on (gblite.visual.force...) and the
custom vf is on (c.visual.force...)
                        jq.ajax({
                       url: getAjaxResponderURL() +
'?reqtype=AccountAgent&accountId='+accountId+'&rowName='+thisRow.attr('name')+'&colN
ame='+fieldsForAgentDefault[agentApiName],
                        dataType: 'jsonp',
                        jsonp: 'callback',
                        jsonpCallback: 'handleJsonpCallback'
                    });
                  }
            });
      }

      function setDefaultValueForField(pMetaCol, pGridInfo, defaultValue) {
            // get the name of the table we're in
            var defaultValueTable =
jQuery('table[name="new_'+pGridInfo.gridId+'"]');
            if (defaultValueTable.length > 0) {
                  // get the column with the matching name
                  var defaultCol =
defaultValueTable.find('td[name="'+pMetaCol.fieldId+'"]');
```

```
                    if (pMetaCol.showTextInput()) {
                          var defaultValueInput =
defaultCol.find('input[type="text"]');
                          if (defaultValueInput.length > 0) {
                                defaultValueInput.val(defaultValue);
                          }
                    }
             }
       }
});

function getAjaxResponderURL() {
             var currentLocation = window.location.href,
                    // replace the namespace with 'c', the namespace of custom vf
pages
                    ajaxURL = currentLocation.replace('gblite.','c.');

             // changed from ajaxURL.indexOf('Grid?') so that tracking still works
on embedded grids that don't have the right case in the URL for "grid"
             ajaxURL = ajaxURL.substring(0, ajaxURL.toLowerCase().indexOf('grid?'))
                          + 'GridAjaxResponder';

             return ajaxURL;
       }


// public function called by the GridAjaxResponder VF
function handleJsonpCallback(data) {
       if (!data) return;

       var agentId = data.agentId || '',
             agentName = data.agentName || '',
             agentInput = jq('#gbMainTable').find('tr.nr[name="'+data.rowName+'"]
td[name="'+data.colName+'"] input');

       // set the lookup value on the Agent
       agentInput.val(agentName).attr('name', agentId).change();
}
```

Type: Visualforce

```
<!-- Used for custom Ajax requests made from GridBuddy grids.
       The contentType must be text/javascript since the Ajax calls use JSONP.
-->
<apex:page controller="GridAjaxDelegator" action="{!processRequest}"
contentType="text/javascript; charset=utf-8" showHeader="false"
standardStylesheets="false" sidebar="false">
handleJsonpCallback({!jsonResponse})
</apex:page>
```

Type: Apex

```
/**
 * This class is used for custom Ajax requests made from GridBuddy grids
```

```
 */
public with sharing class GridAjaxDelegator {

    public String jsonResponse {get; private set;}

    public void processRequest() {
        // determine what the ajax request is for
        String reqType =
ApexPages.CurrentPage().getParameters().get('reqtype');

        if (reqType == 'AccountAgent') {
            this.jsonResponse = getAgentFromAccount();
        }
    }

    public String getAgentFromAccount() {
        String accountId =
ApexPages.CurrentPage().getParameters().get('accountId');
        String rowName =
ApexPages.CurrentPage().getParameters().get('rowName');
        String colName =
ApexPages.CurrentPage().getParameters().get('colName');

        Map<String, String> result = new Map<String, String>{
            'agentId' => '',
            'agentName' => '',
            'rowName' => rowName,
            'colName' => colName
        };

        try {
            List<Account> accountWithAgent = [select Agent_Distributor__c,
Agent_Distributor__r.Name from Account where Id = :accountId limit 1];

            if (accountWithAgent != null && accountWithAgent.size() > 0) {
                String agentId = accountWithAgent.get(0).Agent_Distributor__c;

                if (agentId != null) {
                    result.put('agentId', agentId);
                    result.put('agentName',
accountWithAgent.get(0).Agent_Distributor__r.Name);
                }
            }

            if (Test.isRunningTest()) {
                // for coverage
                throw new TestException('testing');
            }
        } catch (Exception e) {
            // ignore
            System.debug('error retrieving Agent for Account:
'+e.getMessage());
        }
```

```
        return JSON.serialize(result);
    }

    private class TestException extends Exception {}

    @isTest
    public static void testAll() {
        ApexPages.CurrentPage().getParameters().put('reqtype',
'AccountAgent');
        ApexPages.CurrentPage().getParameters().put('accountId', '');
        ApexPages.CurrentPage().getParameters().put('rowName', 'r-1');
        ApexPages.CurrentPage().getParameters().put('colName', 'v1');

        GridAjaxDelegator delegator = new GridAjaxDelegator();
        delegator.processRequest();

        Map<String, String> resultMap = (Map<String,
String>)JSON.deserialize(delegator.jsonResponse, Map<String, String>.class);
        System.assert(resultMap.containsKey('agentName'));
        System.assert(resultMap.containsKey('agentId'));
        System.assert(resultMap.containsKey('rowName'));
        System.assert(resultMap.containsKey('colName'));
    }
}
```