appbuddy

gridbuddy

GridBuddy Extensions & Customization Guide

appbuddy

Contents

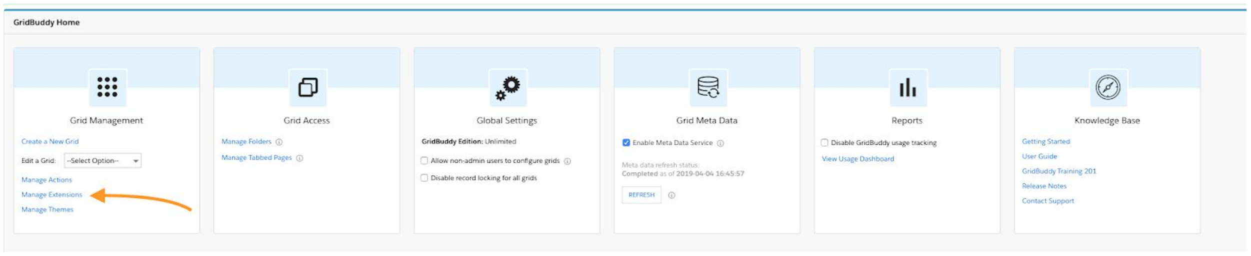
Introduction

GridBuddy is a highly configurable application for managing and editing data in Salesforce.com. Requirements that fall outside of out-of-box behavior can be easily met with GridBuddy's robust extension capabilities via GridBuddy's JavaScript and CSS integration API. This can be used to create highly tailored experiences, from small changes like hiding buttons, to more complex changes like querying the back-end for information and then constructing custom validation rules that are enforced on the front-end. This document outlines steps for creating and integrating extensions in your GridBuddy instance.

Creating Extensions

Extensions can be integrated in Grid Wizard > Manage Extensions (called 'Manage Custom Code' in versions prior to 7.0) from the Grid Wizard landing page:

[Grid Wizard in version 7.0 and after]


GridBuddy Home

Grid Management
Create a New Grid
Edit a Grid: --Select Option--
Manage Actions
Manage Extensions
Manage Themes

Grid Access
Manage Folders
Manage Tabbed Pages

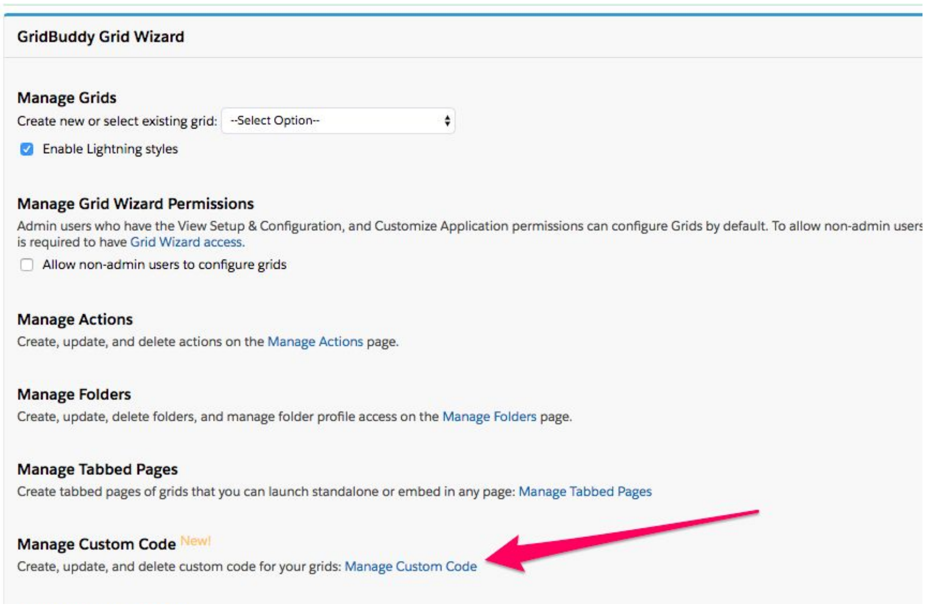
Global Settings
GridBuddy Edition: Unlimited
☐ Allow non-admin users to configure grids
☐ Disable record locking for all grids

Grid Meta Data
☑ Enable Meta Data Service
Meta data refresh status:
Completed as of 2019-04-04 16:45:57
REFRESH

Reports
☐ Disable GridBuddy usage tracking
View Usage Dashboard

Knowledge Base
Getting Started
User Guide
GridBuddy Training 201
Release Notes
Contact Support

[Grid Wizard in versions prior to 7.0]


GridBuddy Grid Wizard

Manage Grids
Create new or select existing grid: --Select Option--
☑ Enable Lightning styles

Manage Grid Wizard Permissions
Admin users who have the View Setup & Configuration, and Customize Application permissions can configure Grids by default. To allow non-admin users is required to have Grid Wizard access.
☐ Allow non-admin users to configure grids

Manage Actions
Create, update, and delete actions on the Manage Actions page.

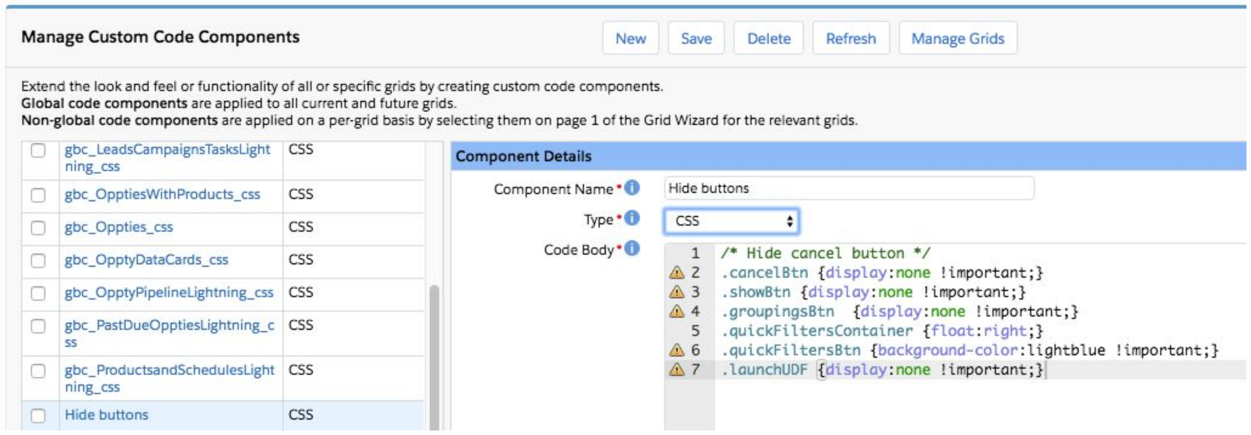
Manage Folders
Create, update, delete folders, and manage folder profile access on the Manage Folders page.

Manage Tabbed Pages
Create tabbed pages of grids that you can launch standalone or embed in any page: Manage Tabbed Pages

Manage Custom Code New!
Create, update, and delete custom code for your grids: Manage Custom Code

appbuddy

Here, CSS and JavaScript components can be defined using the text editor on the screen. The extension modules can either be standalone CSS or JavaScript snippets which you would apply on a grid-by-Grid basis, or they can be defined as Global CSS and Global JavaScript (which are automatically inherited by all grids).



Extend the look and feel or functionality of all or specific grids by creating custom code components.
Global code components are applied to all current and future grids.
Non-global code components are applied on a per-grid basis by selecting them on page 1 of the Grid Wizard for the relevant grids.

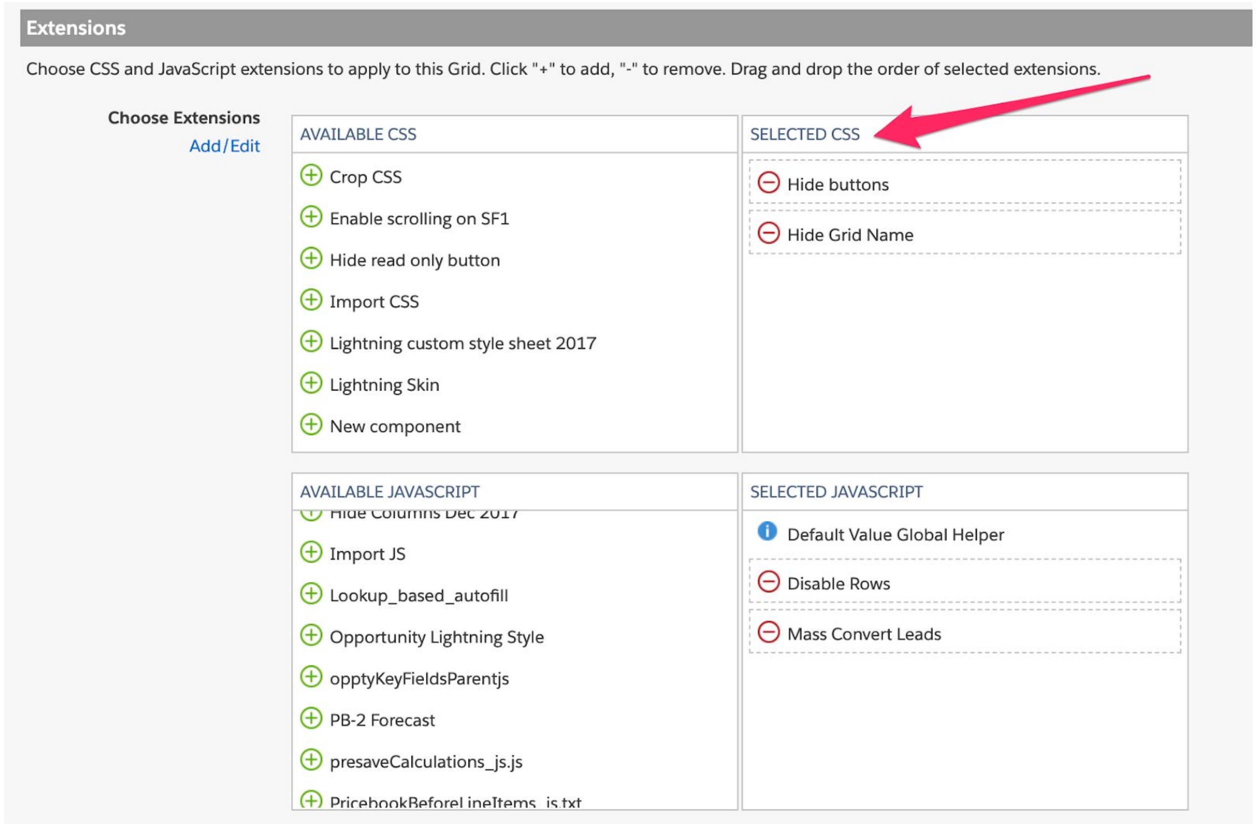
☐	gbc_LeadsCampaignsTasksLightning_css	CSS
☐	gbc_OpptiesWithProducts_css	CSS
☐	gbc_Oppties_css	CSS
☐	gbc_OpptyDataCards_css	CSS
☐	gbc_OpptyPipelineLightning_css	CSS
☐	gbc_PastDueOpptiesLightning_css	CSS
☐	gbc_ProductsandSchedulesLightning_css	CSS
☐	Hide buttons	CSS

Component Details

Component Name: Hide buttons
Type: CSS
Code Body:

```
/* Hide cancel button */
.cancelBtn {display:none !important;}
.showBtn {display:none !important;}
.groupingsBtn  {display:none !important;}
.quickFiltersContainer {float:right;}
.quickFiltersBtn {background-color:lightblue !important;}
.launchUDF {display:none !important;}
```

Once a standalone extension component has been defined, it can be applied to any grid by going to Step 1 of the Grid Wizard and adding the extension component to the “Selected CSS” or “Selected JavaScript” section.
Note: When multiple extensions components are applied to a single grid, they will trigger in the order in which they are applied to the grid, from top to bottom.



Choose CSS and JavaScript extensions to apply to this Grid. Click "+" to add, "-" to remove. Drag and drop the order of selected extensions.

AVAILABLE CSS
- Crop CSS
- Enable scrolling on SF1
- Hide read only button
- Import CSS
- Lightning custom style sheet 2017
- Lightning Skin
- New component

SELECTED CSS
- Hide buttons
- Hide Grid Name

AVAILABLE JAVASCRIPT
- Hide Columns Dec 2017
- Import JS
- Lookup_based_autofill
- Opportunity Lightning Style
- opptyKeyFieldsParentjs
- PB-2 Forecast
- presaveCalculations_js.js
- PricebookBeforeLineItems_js.txt

SELECTED JAVASCRIPT
- Default Value Global Helper
- Disable Rows
- Mass Convert Leads

Developing & Deploying Extensions

A recommended best practice is to develop extensions in a Sandbox environment. This is especially true for more complex extensions, and if the extension involves additional components such as Visualforce pages or Apex classes. Once the extension is ready to be deployed, create a Change Set that includes all the custom resources, and then deploy the Change Set to the production org. The GridBuddy Migration Guide (available from www.appbuddy.com/support/) provides detailed step-by-step instructions for how to deploy GridBuddy to different environments.

Developing Your Own Extensions: Grid Front-End Architecture

For development-savvy organizations, this section outlines the grid's front end architecture that can be manipulated through the extensions framework.

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 that 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 dynamic name attribute corresponds 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.

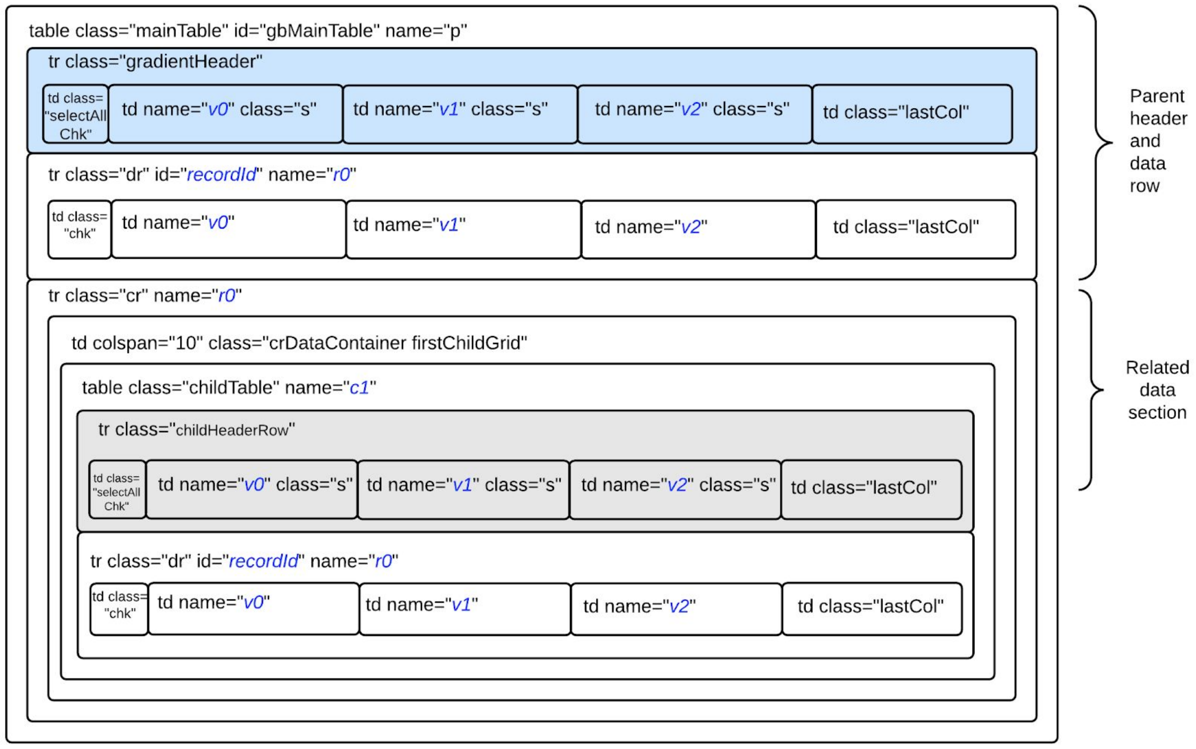


table class="mainTable" id="gbMainTable" name="p"
tr class="gradientHeader"
td class="selectAllChk"
td name="v0" class="s"
td name="v1" class="s"
td name="v2" class="s"
td class="lastCol"
tr class="dr" id="recordId" name="r0"
td class="chk"
td name="v0"
td name="v1"
td name="v2"
td class="lastCol"
tr class="cr" name="r0"
td colspan="10" class="crDataContainer firstChildGrid"
table class="childTable" name="c1"
tr class="childHeaderRow"
td class="selectAllChk"
td name="v0" class="s"
td name="v1" class="s"
td name="v2" class="s"
td class="lastCol"
tr class="dr" id="recordId" name="r0"
td class="chk"
td name="v0"
td name="v1"
td name="v2"
td class="lastCol"
Parent header and data row
Related data section

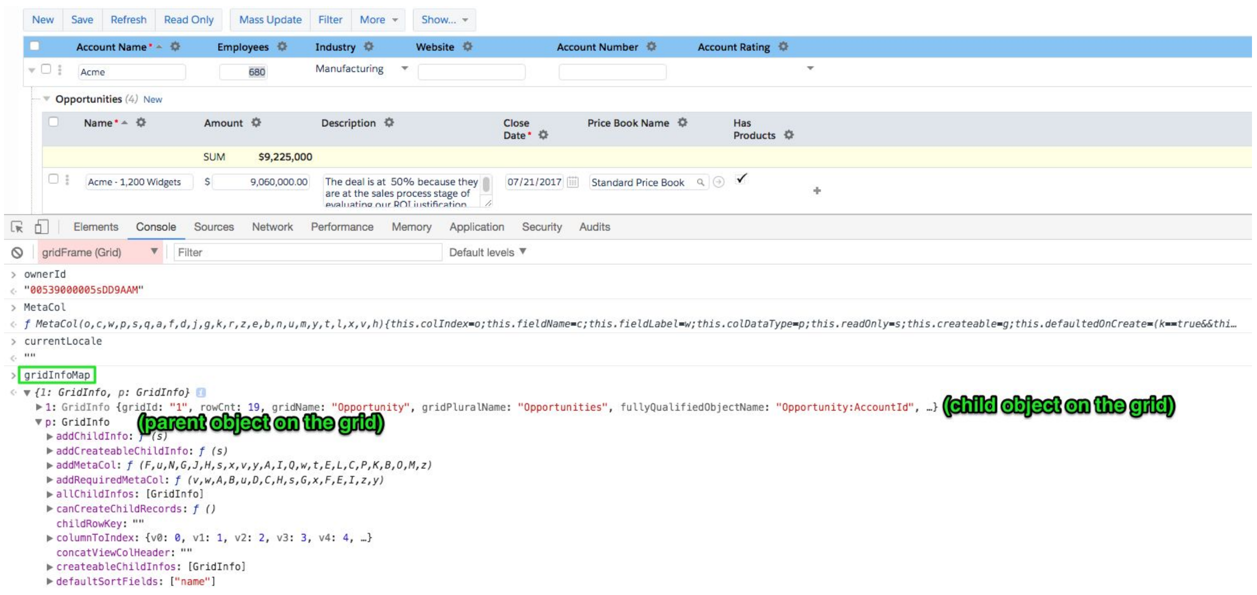
Main GridBuddy JavaScript Objects

Grid Info

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



	Account Name *	Employees	Industry	Website	Account Number	Account Rating
	Acme	680	Manufacturing			

Opportunities (4) New

	Name *	Amount	Description	Close Date *	Price Book Name	Has Products
		SUM $9,225,000				
	Acme - 1,200 Widgets	$ 9,060,000.00	The deal is at 50% because they are at the sales process stage of evaluating our ROI justification	07/21/2017	Standard Price Book	✓

```
> ownerId
< "00539000005sDD9AAM"
> MetaCol
< f MetaCol(o,c,w,p,s,q,a,f,d,j,g,k,r,z,e,b,n,u,m,y,t,l,x,v,h){this.colIndex=o;this.fieldName=c;this.fieldLabel=w;this.colDataType=p;this.readOnly=s;this.createable=g;this.defaultedOnCreate=(k==true&&thi…
> currentLocale
< ""
> gridInfoMap
< ▼{1: GridInfo, p: GridInfo}
   ▶1: GridInfo {gridId: "1", rowCnt: 19, gridName: "Opportunity", gridPluralName: "Opportunities", fullyQualifiedObjectName: "Opportunity:AccountId", …}
   ▼p: GridInfo
     ▶addChildInfo: f (s)
     ▶addCreateableChildInfo: f (s)
     ▶addMetaCol: f (F,u,N,G,J,H,s,x,v,y,A,I,Q,w,t,E,L,C,P,K,B,O,M,z)
     ▶addRequiredMetaCol: f (v,w,A,B,u,D,C,H,s,G,x,F,E,I,z,y)
     ▶allChildInfos: [GridInfo]
     ▶canCreateChildRecords: f ()
      childRowKey: ""
     ▶columnToIndex: {v0: 0, v1: 1, v2: 2, v3: 3, v4: 4, …}
      concatViewColHeader: ""
     ▶createableChildInfos: [GridInfo]
     ▶defaultSortFields: ["name"]
```

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.

Extensions and Actions

Below are sample GridBuddy actions and extensions that are readily available for you to use. Above each item is a note that clarifies whether it requires modification or if it is ready to use as-is.

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

Type: CSS

```
   /* hide view and open record detail actions */
#gbActionsMenu li.a-open-same-win,
#gbActionsMenu li.a-open {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.

Note: For every custom code component you create, a new GB Global Meta record gets created. If needed, you can migrate the respective GB Global Meta record filtered by the name and type.

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'));
    }
}
```

Display conditional fields in the data cards based on Stage

Modify the object name, control field, and displayed fields map for this extension to function on your grid.

Type: Javascript

```
var objName = 'Opportunity';
var controlField = 'StageName';

var displayedFieldsMap = {
  'Prospecting' : ['ForecastCategoryName', 'Description'],
  'Perception Analysis': ['Owner', 'LeadSource', 'NextStep']
}

//DO NOT MODIFY ANYTHING BELOW THIS LINE

var gridInfo = getGridInfoByApiName(objName);
var controlFieldCol = getMetaColByFieldName(gridInfo, controlField);
var colName = controlFieldCol.fieldId;


jq(document).ready(function(){
  var isChildGrid = gridInfo.gridId != 'p';
  var childGridId = isChildGrid ? 'c' + (parseInt(gridInfo.gridId) - 1) : '';
  for(var controlFieldValue in displayedFieldsMap){
    for(var parentRow in gridData){
      var rowData = gridData[parentRow];
      if(isChildGrid){
        var childRowData = rowData[childGridId];
        for(var childRow in childRowData){
          var fieldValue = childRowData[childRow][colName];
          if(fieldValue == controlFieldValue || (modData && modData[childRowData.id] &&
modData[childRowData.id][colName] == controlFieldValue)){
            showHideFieldsInDataCard(childRow, parentRow,
displayedFieldsMap[controlFieldValue]);
          }
        }
      }
      else{
        var fieldValue = rowData[colName];
        if(fieldValue == controlFieldValue || (modData && modData[rowData.id] &&
modData[rowData.id][colName] == controlFieldValue)){
          showHideFieldsInDataCard(parentRow, null, displayedFieldsMap[controlFieldValue]);
        }
      }
    }
  }
  jq('.gbPage').on('change', 'td[name='+colName+'] input', function(e){
    var field = jq(e.target);
    var val = field.val();
    var rowName = field.closest('tr.dr').attr('name');
    var isMassUpdate = field.closest('.massUpdatesBox').length == 1;
    var parentRowName = field.closest('tr.cr').length > 0 ? field.closest('tr.cr').attr('name') : '';
    if(displayedFieldsMap[val]){
      showHideFieldsInDataCard(rowName, parentRowName, displayedFieldsMap[val],
isMassUpdate);
    }
  })
})

function showHideFieldsInDataCard(row, parentRow, fieldsToShow, isMassUpdate){
  var dataCard;
  if(parentRow){
    dataCard = jq('tr.cr[name="'+parentRow+'"] tr.dr[name="'+row+'"]').next('.dataCard');
  }
  else{
    dataCard = isMassUpdate ? jq('.massUpdatesBox .dataCard') : jq('#gbMainTable > tbody >
tr.dr[name="'+row+'"]').next('.dataCard');
  }

  var sections = jq('.cardItem', dataCard);
  for(var i = 0, len = sections.length; i < len; i++){
    var section = jq(sections[i]);
    var sectionFieldName = getMetaColByCellName(section.attr('name'), gridInfo).fieldName;
    if(fieldsToShow.indexOf(sectionFieldName) == -1){
      section.hide();
    }
    else{
      section.show();
    }
  }
}
```

Create a new record within Editable Related Columns

No changes required for this extension to work on your grid.

Type: Javascript

```
/**
 * Copyright © 2018 Primal Cause, Inc. All rights reserved.
 *
 * Add a New link to the Editable Related Popup. If user clicks the link the new row appears in
the popup as well as in the related object table on the grid
 */
jq(document).ready(function() {
    addNewButton();
    jq(mainTableJqueryId).on('click', '#relatedColumnWidget .createNew', function(e) {
        jq('.gbPage .gbBtnGroup .saveBtn').addClass('savePending');
        var parentTableName = jq('#relatedColumnWidget').attr('data-parent-row-name');
        var childTableName = jq('#relatedColumnWidget .childTable').attr('name');
        var gridInfo = gridInfoMap[childTableName.charAt(1)];
        var childRows = jq('.cr[name="'+parentTableName+'"]');
        var relatedColumnsWidget = jq('#relatedColumnWidget');
        var relatedColumnsWidgetFrozenHeader =
relatedColumnsWidget.find('.frozenTableHeader');
        var relatedColumnsWidgetBody = relatedColumnsWidget.find('.body');
        //find the child object in the grid
        for(var i = 0, len = childRows.length; i < len; i++){
            var childRow = childRows[i];
            var childTable = jq(childRow).find('.childTable[name="'+childTableName+'"]');
            var newRow;
            if(childTable.length > 0){
                if (GBFreezeHeaders.isFreezeHeadersEnabled() && childTable.is(':visible')) {
                    // scroll back to the child object container before adding the new record
                    GBHelpers.scrollTo(childTable.closest('.crDataContainer'), function() {
                        newRow = insertNewRow(childTable);
                        jq(window).trigger('gbScroll', {refreshHeader: true});
                    });
                } else {
                    newRow = insertNewRow(childTable);
                }
                var childTableCloneForFrozenHeader =
childTable.clone().addClass('copy').removeAttr('width');
                var childTableClone = childTable.clone().addClass('copy').removeAttr('width');
                childTable.find('td select').each(function() {
                    var selectElem = jq(this),
                        selectMetaCol = getMetaCol(selectElem, gridInfo),
                        cellContainer = selectMetaCol.getCellContainer(),
                        parentCell = selectElem.parents(cellContainer + ':first'),
                        parentCellName = parentCell.attr('name'),
                        parentRowName = parentCell.closest('tr.dr').attr('name'),
                    selectElemClone;

                if (parentCell.hasClass('dvc')) {
                    selectElemClone = childTableClone.find('tr.dr[name="' + parentRowName + '"]
' + cellContainer + '[name="' + parentCellName + '"].dvc select');

                } else {
                    selectElemClone = childTableClone.find('tr.dr[name="' + parentRowName + '"]
' + cellContainer + '[name="' + parentCellName + '"]:not(.dvc) select');
                }

                selectElemClone.val(selectElem.val());

                if (selectElem.is(':disabled')) {
                    selectElemClone.prop('disabled', true)
                }
            });
            relatedColumnsWidgetFrozenHeader.empty();
            relatedColumnsWidgetBody.empty();

relatedColumnsWidgetFrozenHeader.append(childTableCloneForFrozenHeader.find('.dr').re
move().end()); // set frozen header
            relatedColumnsWidgetFrozenHeader.find('.summaryRow').remove(); // would be
duplicate
            relatedColumnsWidgetBody.append(childTableClone); // set content
            // If freeze cols is enabled, make sure that all the columns are shown
            if (GBFreezeCols.isFreezeColsEnabled(true)) {
                relatedColumnsWidgetBody.find('td.none').removeClass('none');
            }
            //set up event listeners for new rows
            relatedColumnsWidgetBody.find('tr.nr').each(function(){
                newRow = jq(this);
                setEventsForNewRow(newRow);
            });

            // set child table with a negative margin so the first row is flush with the frozen
header.
            childTableClone.css('margin-top',
-(childTableClone.find('.childHeaderRow').height()));

            GBRowHelper.setWidthsForRelatedWidgetFrozenHeader(childTableClone,
childTableCloneForFrozenHeader);

            _setupRelatedWidgetEvents(relatedColumnsWidgetBody, childTableClone,
childTableCloneForFrozenHeader);

            // this is needed to avoid the child section from collapsing after clicking New
            e.stopPropagation();
            break;
          }
        }

    });
});

function addNewButton() {
    jq('#relatedColumnWidget .header .count').after('<span class="createNew"
style="padding-left: 10px; color: white !important;">New</span>');
}

function _setupRelatedWidgetEvents(relatedColumnsWidgetBody, childTableClone,
childTableCloneForFrozenHeader) {
    relatedColumnsWidgetBody.scroll(function() {
        var scrollLeft = jq(this).scrollLeft();
        childTableCloneForFrozenHeader.css('margin-left', -(scrollLeft));
    });

    _setupRelatedWidgetTextAreaResizeEvent(childTableClone,
childTableCloneForFrozenHeader);
}

function _setupRelatedWidgetTextAreaResizeEvent(childTableClone,
childTableCloneForFrozenHeader) {
    // reisze frozen header when text area width changes the cell widths
    var textAreas = childTableClone.find('textarea');

    // store init (default) state
    textAreas.data('x', textAreas.outerWidth());
    textAreas.data('y', textAreas.outerHeight());

    textAreas.mouseup(function(){
        var jqThis = jq(this);

        if (jqThis.outerWidth() != jqThis.data('x') || jqThis.outerHeight() != jqThis.data('y')) {
            setWidthsForRelatedWidgetFrozenHeader(childTableClone,
childTableCloneForFrozenHeader);
        }

        // store new height/width
        jqThis.data('x', jqThis.outerWidth());
        jqThis.data('y', jqThis.outerHeight());
    });
}
```

Rename the default filter

Type whatever name you'd to replace 'default' in the filter menu.

Type: Javascript

```
//Change Default Filter Name to Custom

var defaultFilterName = 'My Accounts';

// DO NOT EDIT ANYTHING BELOW THIS LINE

jq(function() {
  jq('.filterOptions option[value="Default"]').text(defaultFilterName);
});
```

Hide the default filter

No changes needed for this to function.

Type: Javascript

```
    jq(document).ready(function() {
      jq('.filterOptions option[value="Default"]').remove()
    });
```

Hide grid artifacts like Show, Read Only/Edit, Grid Name, etc

Hide the grid name:

Type: CSS

```
.gridHeaderCell { display: none;}
```

Hide multiple buttons: read-only, mass update, show, grouping, quick filter

Type: CSS

```
    .cancelBtn {display:none !important;}
    .massUpdates {display:none !important;}
    .showBtn {display:none !important;}
    .groupingsBtn  {display:none !important;}
    .quickFiltersContainer {float:right;}
    .quickFiltersBtn {background-color:lightblue !important;}
    .launchUDF {display:none !important;}
```

Import from Excel/CSV into a single object grid with copy/paste

This requires a CSS and JS file. Both are shown here. Be sure to note the limitations called out in the comment section at the top of each file.

Type: CSS

```
/* IMPORT - 2 files. This is a CSS file. There is JS file too.

* CSS feature works under some conditions which need to be followed -
* Date formatting in Excel must show all 4 year numbers: 2012 (not /12)
* Add both CSS and JS files on the Grid.
* The grid should be a Single Object Grid
* Code does not support 'Owner' field
* Create an Excel Sheet or CSV with API names of the grid fields
* In date fields, be sure to include all 4 digits for the year (such as 2020)
* You can skip any non-required/ optional fields
* Custom code will add a 'Import' button on top
* Click on the Import button and it should launch an import window

* Copy the excel file data along with the field API names and paste it (ctrl+c, ctrl+v)

*Save and it should save the imported record on the grid

*/

#importBox .header {

  padding: 5px;

  font-size: 12px;

  font-weight: bold;

}


#importBox .title {

  float: left;

  color: #EFEFEF;

  font-size: 1.1em
}

#importBox .closeX {
  float: right;
  padding-top: 0;
  padding-right: 0;
  color: #EFEFEF;
  font-weight: normal;
}

#importBox {
  background-color: #0088BA;
}

#importBox #importBtns {
  background-color: #F5F5F5;
}

#importBox {
  width: 660px;
  display: none;
  position: absolute;
  padding: 5px;
  font-size: 13px;
}
```

```
#importBox #importBody {

  min-height: 80px;

  padding: 20px 18px;

  background-color: white;

  overflow: auto;

}

#importBox #importBtns {

  text-align: center;

  padding: 12px 20px;

}

#importBtns .importInfo{

  text-align: left;

}

#importBox #importBody, #importBox #importBtns {

  cursor: default;

}

#importBox.importBox {

  cursor: move;

}

#importBox ul.udc {

  list-style-type: none;
  padding-left: 0;

  margin-top: 0;

}


#importBox ul.udc .udcField {

  cursor: move;

}


#importBox .objectSection {

  margin: 0

}


#importBox .objectSection .objectLabel {

  display: inline-block;

  margin-bottom: 5px;

}


#importBox {

  z-index: 6001 !important;

}
```

Type: Javascript

```
/* Import feature works under some conditions which need to be followed -

 * Date formatting in Excel must show all 4 year numbers: 2012 (not /12)

 * Add both CSS and JS files on the Grid.

 * The grid should be a Single Object Grid
 * Code does not support 'Owner' field

 * Create an Excel Sheet with API names of the grid fields

 * You can skip any non-required/ optional fields

 * Custom code will add a 'Import' button on top

 * Click on the Import button and it should launch an import window

 * Copy the excel file data along with the field API names and paste it (ctrl+c, ctrl+v)

 * If Excel file import doesn't work, copy the data from Google Spreadsheets

 * To update existing records, add "Id" as the *first* column

 * Save and it should save the imported record on the grid

 */

var importdata;


jq(document).ready(function () {

  init();

});


function init(){


  // append button

  jq('.gbBtnGroup:eq(1)').after('<span class="gbBtnGroup"><input class="importPopup gbBtn"
type="button" value="Import" title="Open import from excel popup"></span>')


  // append popup

  jq('body').append('<div id="importBox" class="msgBox boxShadow importBox"> <div
id="importDiv"> <div class="header"> <span class="title">Import from spreadsheet</span>
<span class="closeX">Close&nbsp;<b>X</b></span> <br clear="all"/> </div><div
id="importBtns" class="importInfo"> <div>Please paste the spreadsheet data while this popup
is open by clicking Cmd + V or Ctrl + V</div><div>If the data contains record ids, the records
will be updated, if they don\'t, new records will be created.</div></div><div id="importBody">
```

```
<div id="importStats"></div></div><div id="importBtns"> <input type="button" class="gbBtn
applyImport" value="Apply"/> <input type="button" class="gbBtn importCancel"
value="Cancel"/> </div></div></div>');


  jq('#importBox').find('span.closeX, .importCancel').click(function () {

    // close the widget

    toggleFieldSelectorPopup(false);

  });


  jq('#importBox .applyImport').click(function () {

    applyImport(false);

  });


  jq('.importPopup').on('click', function(){

    toggleFieldSelectorPopup(true);

  });


  document.addEventListener('paste', handlePaste);

}


function toggleFieldSelectorPopup(pShowBox) {

  var importBox = jq('#importBox');



  if (pShowBox) {

    jq('#importStats').empty();

    importBox.find('.applyImport').hide();


    // center and show the widget, and show the background overlay
    jq('#gbOverlay').show();

    importBox.center();

    importBox.show();

  } else {

    // hide the widget and overlay

    importBox.hide();

    jq('#gbOverlay').hide();

  }

}

function handlePaste (e) {

  var clipboardData, pastedData;

  // Stop data actually being pasted into div

  e.stopPropagation();

  e.preventDefault();

  // Get pasted data via clipboard API

  clipboardData = e.clipboardData || window.clipboardData;

  pastedData = clipboardData.getData('Text');

  // parse pasted data to a table

  var rows = pastedData.split(/\r\n|\n|\r/);

  for (var i = 0; i < rows.length; i++) {

    rows[i] = rows[i].split('\t');
  }


  // remove last empty row on Windows paste

  if (rows[rows.length -1] !== undefined && rows[rows.length -1].length === 1 &&
rows[rows.length -1][0] === '') {

    rows.splice(-1,1);

  }


  importdata = {

    columns: rows.shift(),

    data: rows

  };


  renderImportStats();

}


function renderImportStats(){

  var hasIds = importdata.columns.some(function(item){

    if (item.toLowerCase() == 'id') return true;

  });

  var statsText = 'The pasted data will ' + (hasIds?'update':'create') + ' ' +
importdata.data.length + ' records. Would you like to apply these changes?'


  jq('#importBox .applyImport').show();

  jq('#importStats').empty().append('<span>'+statsText+'</span>')

}


function applyImport(){
  var hasIds = importdata.columns.some(function(item){
    if (item.toLowerCase() == 'id') return true;
  });

  // reverse the order of data so that it looks like the excel sheet
  importdata.data.reverse();

  if(hasIds){
    editRecords();
  }else{
    createNewRecords();
  }
  toggleFieldSelectorPopup(false);
}

function createNewRecords() {
  var importRow;
  var newRow;

  for (var i = 0; i < importdata.data.length; i++) {
    importRow = importdata.data[i];
    newRow = insertNewRow(getDataTable('p'));
    applyRowData(newRow, importRow);
  }
}
```

```
function editRecords(){
  for (var i = 0; i < importdata.data.length; i++) {
    importRow = importdata.data[i];
    applyRowData(null, importRow);
  }
}

function applyRowData(rowElem, rowData){
  var colVal;
  var newCol;
  var metaCol;

  for (var j = 0; j < importdata.columns.length; j++) {

    // if column is id, find row
    if(importdata.columns[j].toLowerCase() == 'id'){
      rowElem = jq('#'+rowData[j]);
      continue;
    }

    // fill in column with data
    metaCol = getMetaColByFieldName(gridInfoMap.p, importdata.columns[j].trim());
    colVal = rowData[j];
    newCol = rowElem.find('td[name="' + metaCol.fieldId + '"]');

    // data type specific handling
    if(metaCol.colDataType == 'PICKLIST'){

      newCol.find('.plTxt').click();

    }


    newCol.find('input, select, textarea').val(colVal).change();

  }

}
```

Set a default picklist value on new records

Customize the object, the field, and the value for this to function correctly in your grid.
Type: Javascript

```
/**
* 1. Defaults the value of a picklist
*/

jq(document).ready(function(){
if (readOnlyGrid==false) {
defaultTheNameField();
}

function defaultTheNameField() {
// object api name to map of field api name to default value, Sample Account Object here
var fieldsToDefault = {
'Account':{ 'Type':'Press'}
//Object_API Name : 'Picklist_API_Name' : 'Picklist Default Value'
};

///// DON'T CHANGE ANYTHING BELOW THIS LINE /////

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

            if (fieldsToDefault[gridInfoObj.gridApiName]) {
                var fieldsWithDefaultValues = fieldsToDefault[gridInfoObj.gridApiName],
                    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 (fieldsWithDefaultValues[thisCol.fieldName]) {
                        var defaultValue = fieldsWithDefaultValues[thisCol.fieldName];
                        setDefaultValueForField(thisCol, gridInfoObj, defaultValue);
                    }
                }
            }
        }
    }

    function setDefaultValueForField(pMetaCol, pGridInfo, defaultValue) {
        // get the name of the table we're in
        var defaultValueTable = jq('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()) {
                // for text inputs
                var defaultValueInput = defaultCol.find('input[type="text"]');
                defaultValueInput.val(defaultValue);
            }else if (pMetaCol.isTypePicklist()) {
                // for picklists
                defaultCol.find('select option:selected').val(defaultValue);

            }
        }
    }

});
```

Format your data card into sections

Update the sections map at the top for this to function correctly on your grid.

Type: Javascript

```
/* This creates rows with headers within the data card. */

GBDataCard = (function(window, document, jq) {

  var sectionsMap = {
    Account: {
      licenses: {
        title: 'Licenses',
        fields: ['of_GB_Licenses__c', 'of_SFDC_Licenses__c']
      },
      billing: {
        title: 'Billing',
        fields: ['BillingCity', 'BillingCountry']
      },
      created: {
        title: 'Created',
        fields: ['CreatedById', 'CreatedDate']
      },
      hierarchy: {
        title: 'Hierarchy',
        fields: ['Account_Hierarchy_Level__c', 'Hierarchy_Link__c']
      }
    }
  };



  ///// DON'T CHANGE ANYTHING BELOW THIS LINE /////

  var sidebarEnabled,
    sidebarCellWidth;

  function init() {
    sidebarEnabled = (window.location.href.indexOf('&ssb=1') > 0),
      sidebarCell = jq('#sidebarCell'),
      sidebarCellWidth = sidebarCell.outerWidth();

    _initEventHandlers();
  }

  function getDataCardIcon(gridInfo) {
    if (gridInfo.hasDataCardFields) {
      return '<span class="icon-wrap"></span>';
    }
    return '';
  }

  function getDataCardCellHtml(gridInfo, metaCol, cellHtml, cellClasses) {
    var dataCardClasses = ['cardItem', cellClasses],
      dataCardHtmlArr = [],
      gridInfoToUse = gridInfo;

    if (metaCol.gridInfoGridId != gridInfo.gridId) {
      // this metaCol is for a denormalized view child object
      gridInfoToUse = gridInfoMap[metaCol.gridInfoGridId];
    }

    dataCardHtmlArr.push('<div class="'+ dataCardClasses.join(' ')
      +'" name="'+ metaCol.fieldId +'">');
    dataCardHtmlArr.push('<label class="cardLabel">'+
metaCol.fieldLabelForDisplay(gridInfoToUse) +'</label>');
    dataCardHtmlArr.push('<div class="cardCell">'+ cellHtml +'</div>');
    dataCardHtmlArr.push('</div>');

    return dataCardHtmlArr;
  }

  function getDataCardRowHtml(gridInfo, rowId, rowName, classes,
                              dataCardHtmlArr, showByDefault, childRowId, childRowName) {

    var childIdAttr = (childRowId ? ' data-childId="'+ childRowId +'"' : ''),
      childRowNameAttr = (childRowName ? ' data-childRowName="'+ childRowName +'"' : ''),
      colspan = GBRowHelper.getColCount(gridInfo, true),
      htmlArr = [];

    if (!showByDefault) {
      classes.push('none');
    }

    htmlArr.push('<tr id="'+ rowId +'" class="' + classes.join(' ')
      + ' dataCard" name="'+ rowName +'"'
      + childIdAttr + childRowNameAttr + '>');
    htmlArr.push('<td class="firstCol"></td>');
    htmlArr.push('<td class="dataCardCell" colspan="'+ colspan +'">');
    htmlArr.push('<div class="cardTable">'+ dataCardHtmlArr.join('') +'</div>');
    htmlArr.push('</td></tr>');

    // THIS BELOW CODE IS THE ONLY PART THAT HAS CHANGED
    setTimeout(function(){
      var sections = sectionsMap[gridInfo.gridApiName];
    var section;
    var sectionIndex = 0;
    var metaCol;
    var cardSelector;
    var isLastSection;
    var sectionStyle;

    for(var k in sections) {
      section = sections[k];
      cardSelector = [];
      sectionIndex ++;

      if(section.fields == undefined) continue;
      for (var i = 0; i < section.fields.length; i++) {
        metaCol = getMetaColByFieldName(gridInfo, section.fields[i]);
        isLastSection = sectionIndex == Object.keys(sections).length;
        if(metaCol){
          cardSelector.push('[name=' + metaCol.fieldId + ']');
        }
      }
      sectionStyle = isLastSection ? '' : 'margin-bottom: 8px; border-bottom: 1px solid
#d4dadc;';
      jq('#'+ rowId + '.dataCard').find(cardSelector.join(',')).wrapAll('<div
class="dataCardCustomSection ' + k + '" style="'+sectionStyle+'"/>');
      if(section.info && section.info !== ''){
        jq('#'+ rowId + '.dataCard').find('.dataCardCustomSection.'+k).prepend('<h3
style="display: block;">' + section.info + '</h3>');
      }
      jq('#'+ rowId + '.dataCard').find('.dataCardCustomSection.'+k).prepend('<h1
style="display: block;">' + section.title + '</h1>');
    }
  }, 0);

  return htmlArr.join('');
 }

 function openDataCard(dataCardCell) {
  if (dataCardCell.closest('.dataCard').hasClass('none')) {
    dataCardCell.closest('.dataCard').prev('tr.dr').find('.icon-wrap').click();
  }
 }

 function addEventHandlerForMassUpdate() {
  jq('#massUpdatesTable').on('click', '.icon-wrap', _onIconClicked);
 }

 function removeCollapsedDataCards(rows) {
  rows = rows.filter(function() {
    if (jq(this).hasClass('dataCard')
      && !jq(this).prev('.dr').find('.icon-wrap').hasClass('on')) {
      // this is a data card and it was collapsed
      return false;
    }
    return true;
  });

  return rows;
 }

 function _initEventHandlers() {
  jqFrozenAndUnfrozenTables.on('click', '.icon-wrap', _onIconClicked);

  jq(window).resize(function() {
    jq('.dataCard:visible').each(function() {
      var dataCardRow = jq(this),
        primaryRow = dataCardRow.prev('#'+dataCardRow.attr('id')),
        datCardCell = dataCardRow.find('.dataCardCell'),
        cardTable = dataCardRow.find('.cardTable'),
        firstColumnWidth = primaryRow.find('td:eq(0)').outerWidth(),
        lastColumnWidth = primaryRow.find('td.lastCol').outerWidth(),
        primaryRowWidth = _getPrimaryRowWidth(primaryRow, dataCardRow,
firstColumnWidth, lastColumnWidth),
        dataCardRowWidth = _getDataCardRowWidth(primaryRowWidth, firstColumnWidth);

      datCardCell.css('width', primaryRowWidth);
      cardTable.css('width', dataCardRowWidth);
    });
  });
 }

 function _onIconClicked() {
  var jqThis = jq(this),
    elemInFrozenTable = GBFreezeCols.isElementInFrozenTable(jqThis),
    isInEditableColumnWidget = (jqThis.closest('#relatedColumnWidget').length > 0),
    primaryRow,
    iconWrap,
    firstPrimaryCol;

  if (!isInEditableColumnWidget && elemInFrozenTable) {
    if (GBFreezeCols.isFreezeColsEnabled()) {
      GBFreezeCols.unfreezeColumns(false, GBFreezeCols.dataCardsNotSupportedMsg);
    }

    var parentRow = jqThis.closest('tr'),
      parentRowId = parentRow.attr('id'),
      parentTd = jqThis.closest('td'),
      rowToUpdate,
      tableToUpdate;

    primaryRow = GBFreezeCols.getRowToUpdate(parentRow, parentRowId,
jq(mainTableJqueryId));
    iconWrap = primaryRow.find('.icon-wrap');
    firstPrimaryCol = primaryRow.find('td[name="' + parentTd.attr('name') + '"]');

  } else {
    // This case covers if there is an empty required field in the data card and freeze columns
is enabled.
    // The validateRemainingRequiredRowFields fn would've clicked the .icon-wrap icon in the
unfrozen table in this case.
    if (GBFreezeCols.isFreezeColsEnabled() && !jqThis.is(':visible')) {
      GBFreezeCols.unfreezeColumns(false, GBFreezeCols.dataCardsNotSupportedMsg);
    }

    primaryRow = jqThis.closest('tr.dr');
    iconWrap = jqThis;
    firstPrimaryCol = jqThis.closest('td');
  }

  var dataCardRow = primaryRow.next('#'+ primaryRow.attr('id') +'.dataCard');

  // hide the data card row
  if (!dataCardRow.hasClass('none')) {
    GBMixpanel.track('Change Grid Display', {
      'Display Control': 'Data Card',
      'Option': '',
      'State': 'Collapse'
    });

    dataCardRow.addClass('none');
    iconWrap.removeClass('on');

    // remove a border between the primary and data card row on the first column
    firstPrimaryCol.removeClass('borderBottomNone');

    if (!isInEditableColumnWidget) {
      GBFreezeCols.freezeColumnsIfEnabled();
    }

  } else {
    // show the data card row
    var cardTable = dataCardRow.find('.cardTable'),
      dataCardCell = dataCardRow.find('.dataCardCell'),
      firstColumnWidth = primaryRow.find('td:eq(0)').outerWidth(),
      lastColumnWidth = primaryRow.find('td.lastCol').outerWidth(),
      primaryRowWidth = _getPrimaryRowWidth(primaryRow, dataCardRow,
firstColumnWidth, lastColumnWidth),
      dataCardRowWidth = _getDataCardRowWidth(primaryRowWidth, firstColumnWidth);

    GBMixpanel.track('Change Grid Display', {
      'Display Control': 'Data Card',
      'Option': '',
      'State': 'Expand'
    });

    addAutocompleteToLookupFields(dataCardRow, false);
    // initialize picklists when the data card is opened
    dataCardRow.find('.plTxt').click();
    dataCardRow.removeClass('none');
    iconWrap.addClass('on');

    // add a border between the primary and data card row on the first column
    firstPrimaryCol.addClass('borderBottomNone');

    dataCardCell.css('width', primaryRowWidth);
    cardTable.css('width', dataCardRowWidth);
  }

  if (!isInEditableColumnWidget && !GBFreezeCols.isFreezeColsEnabled()) {
    GBFreezeCols.updateHeightOfTablesWithNoFrozenCols();
  }
}

function _getPrimaryRowWidth(primaryRow, dataCardRow, firstColumnWidth,
lastColumnWidth) {
  var primaryRowWidth;

  primaryRowWidth = primaryRow.width()-(firstColumnWidth+lastColumnWidth);

  // data card row is in child table
  if (dataCardRow.closest('.childTable').length > 0) {
    var parentPrimaryRow = jq(mainTableJqueryId).find('tbody > tr').eq(0),
      parentfirstColumnWidth = parentPrimaryRow.find('td:eq(0)').outerWidth(),
      parentlastColumnWidth = parentPrimaryRow.find('td.lastCol').outerWidth(),
      parentPrimaryRowWidth =
parentPrimaryRow.width()-(parentfirstColumnWidth+parentlastColumnWidth);

    // if parent primary row is shorter than the child data car row, use the parent primary row
for calculation
```

```
      primaryRowWidth = (parentPrimaryRowWidth < primaryRowWidth) ? primaryRowWidth :
parentPrimaryRowWidth;
    }

    return primaryRowWidth;
  }

  function _getDataCardRowWidth(primaryRowWidth, firstColumnWidth) {
    var gridViewportWidth;

    if (sidebarEnabled && sidebarCell.is(':visible')) {
      gridViewportWidth = window.innerWidth-(sidebarCellWidth+firstColumnWidth);
    } else {
      gridViewportWidth = window.innerWidth-firstColumnWidth;
    }

    gridViewportWidth -= firstColumnWidth;

    return (primaryRowWidth > gridViewportWidth) ? gridViewportWidth+'px' : '100%';
  }

  return {
    init: init,
    getDataCardIcon: getDataCardIcon,
    getDataCardCellHtml: getDataCardCellHtml,
    getDataCardRowHtml: getDataCardRowHtml,
    openDataCard: openDataCard,
    addEventHandlerForMassUpdate: addEventHandlerForMassUpdate,
    removeCollapsedDataCards: removeCollapsedDataCards
  };

})(window, document, jq); // end GBDataCard
```

Automatically open data cards when the grid loads

No changes needed for this to function

Type: Javascript

```
jq(document).ready(function() {
jq('span.icon-wrap').click();
});
```

Add a button that expands and collapses all data cards in your grid

No changes needed for this extension to function

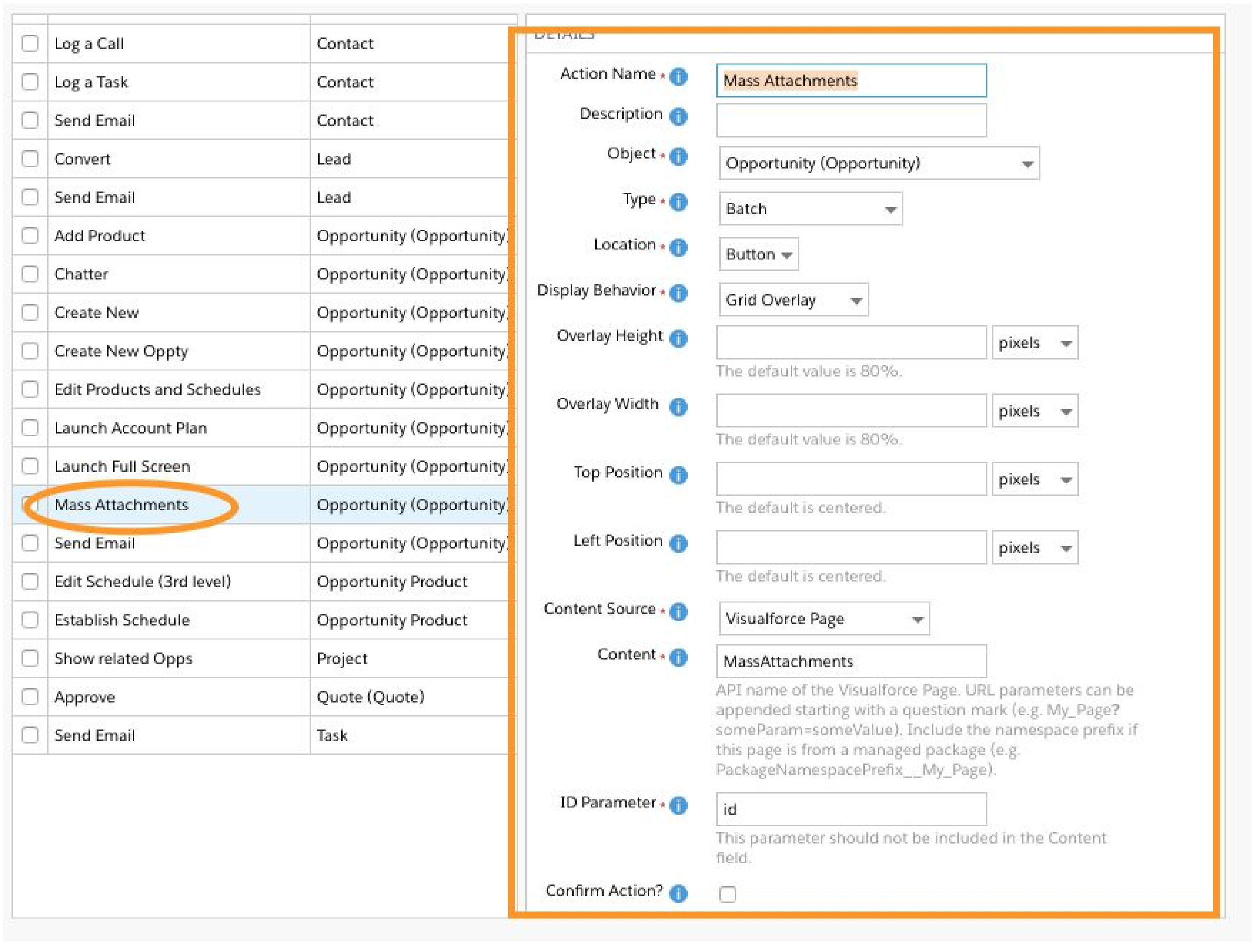
Type: Javascript

```
jq( document ).ready(function() {

    var openDataCardsButton = "<button class='gbBtn openDataCards' type='button'
style='vertical-align: top;'><span class='icon-wrap on' style='vertical-align:
sub;'></span></button>";
    var areDataCardsOpen = false;

    jq('.gridBtnsContainer').append(openDataCardsButton);

    jq('.gridBtnsContainer').on('click', '.openDataCards', function() {
        if (!areDataCardsOpen) {
            jq('tr.dr.dataCard').attr("style", "display: table-row!important");
            areDataCardsOpen = true;

        } else {
           jq('tr.dr.dataCard').attr("style", "display: none!important");
           areDataCardsOpen = false;
        }

    });
});
```

Attach a file to single or multiple records

No changes needed for this extension to function. Note that this is a Visualforce page so the attachment limitations align with what Salesforce supports.

GridBuddy Managed Action Setup



☐	Log a Call	Contact
☐	Log a Task	Contact
☐	Send Email	Contact
☐	Convert	Lead
☐	Send Email	Lead
☐	Add Product	Opportunity (Opportunity)
☐	Chatter	Opportunity (Opportunity)
☐	Create New	Opportunity (Opportunity)
☐	Create New Oppty	Opportunity (Opportunity)
☐	Edit Products and Schedules	Opportunity (Opportunity)
☐	Launch Account Plan	Opportunity (Opportunity)
☐	Launch Full Screen	Opportunity (Opportunity)
☐	Mass Attachments	Opportunity (Opportunity)
☐	Send Email	Opportunity (Opportunity)
☐	Edit Schedule (3rd level)	Opportunity Product
☐	Establish Schedule	Opportunity Product
☐	Show related Opps	Project
☐	Approve	Quote (Quote)
☐	Send Email	Task

DETAILS

Action Name *	Mass Attachments
Description	
Object *	Opportunity (Opportunity)
Type *	Batch
Location *	Button
Display Behavior *	Grid Overlay
Overlay Height	pixels The default value is 80%.
Overlay Width	pixels The default value is 80%.
Top Position	pixels The default is centered.
Left Position	pixels The default is centered.
Content Source *	Visualforce Page
Content *	MassAttachments API name of the Visualforce Page. URL parameters can be appended starting with a question mark (e.g. My_Page?someParam=someValue). Include the namespace prefix if this page is from a managed package (e.g. PackageNamespacePrefix__My_Page).
ID Parameter *	id This parameter should not be included in the Content field.
Confirm Action?	☐

Type: Visualforce

```
<apex:page controller="MassAttachmentsController" lightningStylesheets="true">
    <style type="text/css">
        .container, .buttonsArea {
            margin: 30px;
        }

        .buttonsArea.pbButton {
            text-align: left;
        }
    </style>

    <apex:form >
        <apex:pageMessages />
        <section id="container" class="container" >
            <div class="fileInputArea">
                <apex:inputFile value="{!file}" fileName="{!fileName}" />
            </div>
        </section>
        <section class="buttonsArea pbButton">
            <apex:commandButton action="{!upload}" value="Upload" />
        </section>
    </apex:form>
</apex:page>
```

Type: Apex Class

```
public class MassAttachmentsController {
    public blob file { get; set; }
    public string fileName { get; set; }

    public PageReference upload() {
        ApexPages.getMessages().clear();

        try {
            List<String> parentIds =
      ApexPages.currentPage().getParameters().get('id').split(',');
            List<ContentDocumentLink> filesToInsert = new List<ContentDocumentLink>();

            ContentVersion v = new ContentVersion();

            v.versionData = this.file;
            v.title = this.fileName;
            v.pathOnClient = '/' + this.fileName;

            insert v;

            Id contentVersionId = [SELECT Id, ContentDocumentId FROM ContentVersion
      WHERE Id =: v.Id].ContentDocumentId;

            for (String parentId : parentIds) {
                ContentDocumentLink cdl = new ContentDocumentLink();
                cdl.ContentDocumentId = contentVersionId;
                cdl.LinkedEntityId = parentId;
                cdl.ShareType = 'V';

                filesToInsert.add(cdl);
            }

            insert filesToInsert;
            this.file = null;

            Apexpages.addMessage(new
      ApexPages.Message(ApexPages.SEVERITY.CONFIRM, 'Upload successful.'));
```

```
        } catch(Exception ex) {
            Apexpages.addMessage(new
        ApexPages.Message(ApexPages.SEVERITY.ERROR, 'There was an error while
        uploading a file.'));
        }

        return null;
    }
}
```

Set a default record type in new records

There are two files required for this to function. Please review the comments at the top of each code file. Modifications are required for this to function correctly.

Type: Javascript

```
/* GLOBAL FILE - DEFAULT 'RECORD TYPE' FIELD VALUE ON A GRID (EMBEDDED/STANDALONE)
  * This is one of the 2 extension (custom code) files the grid needs to have applied for this feature to be available.
  * Please make this file type - Global Javascript. Apply this file as is and edit the other local file before applying on
the grid
  * Defaults lookup field searches to run on all Searchable Fields vs just the Name field
  * This needs to run immediately, not on document ready. This is a global JS file you need to have one more local
JS file to make it work
  */
(function defaultLookupSearchType() {

    if (gridInfoMap) {
        for (key in gridInfoMap) {
            var thisGridInfo = gridInfoMap[key];

            for (var i=0; i<thisGridInfo.metaColumns.length; i++) {
                var thisCol = thisGridInfo.metaColumns[i];

                                                if (thisCol.isTypeReference()) {
                                                        thisCol.searchFieldType = 'all';
                                                }
            }
        }
    }
})();

/**
 * Helper module for defaulting values, called from multiple grids
 */
var GBDefaultValueHelper = (function() {
	function setDefaultRecordTypeForObject(objectApiName, recordTypeApiName, recordTypeLabel) {
		for (recordTypeId in objectRecordTypeIdToNames) {
			if (objectRecordTypeIdToNames[recordTypeId] == recordTypeApiName) {
				// use this record type id as the default
				defaultRecordTypeIds[objectApiName] = recordTypeId;
				break;
			}
		}

		var objectToFieldMap = {};
		objectToFieldMap[objectApiName] = { 'RecordTypeId': recordTypeLabel, 'RecordTypeIdValue':
recordTypeId };
		setFieldDefaults(objectToFieldMap);
	}

	function setFieldDefaults(mapOfObjectToFields) {
		///// DON'T CHANGE ANYTHING BELOW THIS LINE /////

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

	    if (mapOfObjectToFields[gridInfoObj.gridApiName]) {
	      var fieldsWithDefaultValues = mapOfObjectToFields[gridInfoObj.gridApiName],
	        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 (fieldsWithDefaultValues[thisCol.fieldName]) {
	          var defaultValue = fieldsWithDefaultValues[thisCol.fieldName],
	              recordTypeIdValue;

	          if (thisCol.fieldName == 'RecordTypeId') {
	              recordTypeIdValue = fieldsWithDefaultValues['RecordTypeIdValue'];
	          }

	       _setDefaultValueForField(thisCol, gridInfoObj, defaultValue, recordTypeIdValue);
	        }
	      }
	    }
	  }
	}

	// private function
	function _setDefaultValueForField(pMetaCol, pGridInfo, defaultValue, defaultRecordTypeId) {

	  // get the name of the table we're in
	  var defaultValueTable = jq('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.isTypeReference() && pMetaCol.isRecordTypeField()) {
                                                    // record type
                                                    // in certain cases we need to set the record type id as the value, otherwise
the default may not work
                                                    defaultCol.find('select
option:selected').text(defaultValue).val(defaultRecordTypeId);

                    } else if (pMetaCol.isTypePicklist()) {
                        // for picklists

                        // special handling for the Currency (CurrencyIsoCode) field
                                                    if (pMetaCol.fieldName == 'CurrencyIsoCode' &&
defaultCol.find('select').length == 0) {
                            // salesforce doesn't expose the currency field on standard create new, so we need to create a
picklist for it for defaulting
                            // otherwise the below is not necessary for any other picklist fields
                            // the None option is required by the GB js logic which assumes there will be None option
generated by Salesforce and checks if the total options are greater than 1
                            defaultCol.append('<select><option>--None--</option><option
selected="true"></option></select>');
                                                    }
                        defaultCol.find('select option:selected').val(defaultValue);

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

            return {
                        setFieldDefaults: setFieldDefaults,
                        setDefaultRecordTypeForObject: setDefaultRecordTypeForObject
            }
})(); // end GBDefaultValueHelper


/**
 * GridBuddy Tracking code
 */
jq(document).ready(function() {

//contain these four and the functions and the call functions
```

```
        trackGridView();
        trackGridSave();
        trackGridExport();
        trackReorderHideColumns();

        // tracks the grid view for this session
        function trackGridView() {
                var trackingURL = getAjaxResponderURL() +
'?reqType=trackGridView&gname='+getParamValue('gname');
                makeTrackingRequest(trackingURL);
        }

        // submits a tracking request when the user clicks the Save button and the form successfully submits
        function trackGridSave() {
                if (readOnlyGrid==false) {
                        var gbForm = document.forms[jq('form.gbForm').attr('name')];

                        jq(gbForm).submit(function() {
                                if (btnClick == 'save') {

                                        if (jq.browser.mozilla) {
                                                // the delay fixes a FF issue where the ajax call doesn't get
submitted during the form submit
                                                setTimeout(makeGridSaveTrackingRequest, 25);
                                        } else {
                                                makeGridSaveTrackingRequest();
                                        }
                                }
                        });
                }
        }

        // submits a tracking request when the user clicks the Export action
        function trackGridExport() {
                jq('div.gbPage').on('click', 'li.exportItem', function() {
                        var trackingURL = getAjaxResponderURL() +
'?reqType=trackGridExport&gname='+getParamValue('gname');
                        makeTrackingRequest(trackingURL);
                });
        }

        // submits a tracking request when the user clicks the Reorder/Hide Columns action
        function trackReorderHideColumns() {
                jq('div.gbPage').on('click', 'li.reorderColsItem', function() {
                        var trackingURL = getAjaxResponderURL() +
'?reqType=trackReorderColumns&gname='+getParamValue('gname');
                        makeTrackingRequest(trackingURL);
                });
        }
        function makeTrackingRequest(trackingURL) {
                jq.ajax({
                        url: trackingURL,
                        dataType: 'jsonp',
                        jsonp: 'callback',
                        jsonpCallback: 'handleJsonpCallback'
                });
        }

        function makeGridSaveTrackingRequest() {
                var trackingURL = getAjaxResponderURL() +
'?reqType=trackGridSave&gname='+getParamValue('gname')+'&fieldNames='+JSON.stringify(getModifiedFields()
);
                makeTrackingRequest(trackingURL);
        }

        // returns an object mapping of Object API Name to an array of Field API Names, only for modified fields
        function getModifiedFields() {
                var modifiedFields = {},
                        tempObjectToFields = {};

                for (var rowId in modData) {
                        var rowData = modData[rowId],
                                fullyQualifiedObjectName = rowData['nm'],
                                gridInfo = (getParentObjectName() == fullyQualifiedObjectName ?
getParentGridInfo() : getChildGridInfo(fullyQualifiedObjectName));

                        for (var key in modData[rowId]) {
                                if (isFieldCell(key)) {

                                        var metaCol = gridInfo.getMetaColByFieldId(key),
                                                fieldName = metaCol.fieldName;

                                        if (!tempObjectToFields[fullyQualifiedObjectName]) {
                                                tempObjectToFields[fullyQualifiedObjectName] = {};
                                                modifiedFields[fullyQualifiedObjectName] = [];
                                        }

                                        if (!tempObjectToFields[fullyQualifiedObjectName][fieldName]) {
                                                tempObjectToFields[fullyQualifiedObjectName][fieldName]
= true;

                                                modifiedFields[fullyQualifiedObjectName].push(fieldName);

                                        } else {
                                                // this field has already been added, skip it to avoid dupes
                                        }
                                }
                        }
```

```
                    }
                    return modifiedFields;
          }

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

          function getParamValue(paramName) {
                    var urlParams = window.location.search,
                              paramValue = '';

                    if (urlParams) {
                              var paramArray = urlParams.split('&');

                              jq.each(paramArray, function(i, item) {
                                        if (item.match(paramName + '=')) {
                                                  paramValue = item.substring(item.indexOf('=') + 1);
                                                  return false;
                                        }
                              })
                    };
                    return paramValue;
          }
});

// public function called by the GridAjaxResponder VF
function handleJsonpCallback(data) {
          // no op
}
```

This is the second file required for the Default Record Type extension to function. Modifications are required for this to function correctly.
Type: Javascript

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

/**
 * (objectApiName, recordTypeApiName, recordTypeLabel)
 */
jq(document).ready(function() {
        GBDefaultValueHelper.setDefaultRecordTypeForObject('Opportunity', 'US_Cement', 'US Cement');
});
```

Office
1001 Broadway Avenue, Millbrae, CA 94030

For more information
Call: + 1-877-648-5437
E-mail: info@appbuddy.com
Web: www.appbuddy.com

About AppBuddy
AppBuddy is the industry leader in data interaction solutions that simplify complex work. With AppBuddy businesses create a single user experience from disparate business data, and enable their users to get work done fast. AppBuddy takes the enterprise systems you have and intelligently turns them into applications that are loved by end users.

Copyright © 2019, AppyBuddy. All rights reserved.