Miva Merchant

MMBatchList Guide

Miva Merchant 9

MM9-MISC-307-1.0 7/10/15

Scope

This document describes how to create an **MMBatchList** in Miva Merchant by defining a JavaScript class. Experience with JavaScript is presumed.

Creating an MMBatchList

Creating a basic batch list requires performing the following steps.

1. Include the required JavaScript, CSS, and HTML code base
2. Create your custom derived **MMBatchList** class
3. Create your JSON loading function
4. Initialize your **MMBatchList** on document load

These steps are detailed in the following sections.

Step 1: Include the required JavaScript, CSS and HTML code base

The following elements are required for use with any custom batch list:

- **Element_MMBatchList_JavaScript**
- **Element_MMBatchList_CSS**
- **Element_MMBatchList_HTML**

In the head tag of the HTML DOM, include the following two function calls:

```
<head>
    <MvEVAL EXPR = "{ [ g.Module_Admin ].Element_MMBatchList_JavaScript() }">
    <MvEVAL EXPR = "{ [ g.Module_Admin ].Element_MMBatchList_CSS() }">


    <script language="JavaScript" src="{ g.clientside_url $ 'Module_Code=' $
     encodeattribute( l.module:code ) $ '&Filename=functions.js' }"></script>
    <script language="JavaScript" src="{ g.clientside_url $ 'Module_Code=' $
     encodeattribute( l.module:code ) $ '&Filename=mystatementsbatchlist.js' }"></
     script>
</head>
```

In the body tag, add the following:

```
<body>
    <MvEVAL EXPR = "{ [ g.Module_Admin ].Element_MMBatchList_HTML() }">
    <div id="mystatements_batchlist_id"></div> <!-- this id will be passed to the
     MMBatchList constructor in your derived MMBatchList class -->
</body>
```

Step 2: Creating a custom derived MMBatchList class

To create a basic display only (no extra features enabled) derived **MMBatchList** class, use the following code example as a guide.

mystatementsbatchlist.js

```
function MyStatementsBatchList()
{
    MMBatchList.call( this, 'mystatements_batchlist_id' );

    this.Feature_SearchBar_SetPlaceholderText( 'Search My Statements...' );
    this.SetDefaultSort( 'id', '' );
}

DeriveFrom( MMBatchList, MyStatementsBatchList );

MyStatementsBatchList.prototype.onLoad = MyStatementsList_Load_Query;
 <!-- Where MyStatementsList_Load_Query is a function with parameters
 "filter, sort, offset, count, callback, delegator" -->

MyStatementsBatchList.prototype.onCreateRootColumnList = function()
{
    var columnlist =
    [
        new MMBatchList_Column_Code( 'Statement Code', 'code',
         'MyStatement_Code' ),
        new MMBatchList_Column_Name( 'Client Name', 'client_name',
         'MyStatement_ClientName' ),
        new MMBatchList_Column_Currency( 'Statement Amount', 'amount',
         'MyStatement_Amount' )
    ];

    return columnlist;
}
```

The preceding code creates a generic batchlist that displays the **Statement Code**, **Client Name**, and **Statement Amount** for each record returned by the **MyStatementList_Load_Query** function.

The following code example shows how to implement the **MyStatementList_Load_Query** function. Setting up the AJAX load function, which returns the JSON list of records, is fairly straight forward. Advanced features, such as custom filtering and additional parameters, are discussed in *Advanced MMBatchList* on page 10.

function.js

```
function MyStatementsList_Load_Query( filter, sort, offset, count, callback,
 delegator )
{
    return AJAX_Call_Module( callback,
                             'admin',
                             'mystatement_module_code',
                               <!-- set up by my module -->
                             'MyStatementsList_Load_Query',
                             '&Filter=' + EncodeArray( filter )         +
                             '&Sort='   + encodeURIComponent( sort )    +
                             '&Offset=' + encodeURIComponent( offset )  +
                             '&Count='  + encodeURIComponent( count ),
                             delegator );
}
```

Step 3: Creating a JSON loading function

A number of functions have been implemented to make creating a JSON query easier. The following example shows how to create a basic load on a single table.

```
<MvFUNCTION NAME = "JSON_MyStatementsList_Load_Query" PARAMETERS = "module var"
 STANDARDOUTPUTLEVEL = "text, html, compresswhitespace">
    <MvCOMMENT> Always Sanitize Incoming Data </MvCOMMENT>
    <MvASSIGN NAME = "g.Filter"           VALUE = "{ trim( g.Filter ) }">
    <MvASSIGN NAME = "g.Sort"             VALUE = "{ trim( g.Sort ) }">
    <MvASSIGN NAME = "g.Offset"           VALUE = "{ trim( g.Offset ) }">
    <MvASSIGN NAME = "g.Count"            VALUE = "{ trim( g.Count ) }">

    <MvASSIGN NAME = "l.search_query"     VALUE = "">


    <MvCOMMENT>
        SQL_Query_SELECT( query var, select )
            query: query variable used by the SQL_Query_XXX functions
              to build the query
            select: the SQL selection parameters
    </MvCOMMENT>
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_SELECT( l.search_query,
     's.*' ) }">
    <MvCOMMENT>
        SQL_Query_FROM( query var, table_name, alias )
            query: query variable used by the SQL_Query_XXX functions
              to build the query
            table_name: the name of the database table
            alias: the alias to assign to this table
    </MvCOMMENT>
```

```
<MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_FROM( l.search_query,
 g.Store_Table_Prefix $ 'MyStatements', 's' ) }">


<MvCOMMENT>
    JSON_Filter( query var, filterlist, correlationlist )
        query: query variable used by the SQL_Query_XXX functions
          to build the query
        filterlist: the g.Filter variable passed in to your query by
          MMBatchList. Used for advanced filtering and custom filtering
          (explained later) and applied to the correlation list
        correlationlist: the list of code:table_column values. For example, if
          your mmbatchlist has a column code of client_name, but your table
          column is client.name, you would use the correlation
          client_name:client.name. This parameter is a
          white-list comma separated string of available search/filter
          columns. Only columns listed here can be searched/filtered
          using MMBatchList
</MvCOMMENT>
<MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Filter( l.search_query, g.Filter,
 'code:s.code,amount:s.amount' ) }">


<MvCOMMENT>
    SQL_Query_OrderBy_Fields( query var, sort, list, default_sort )
        query: query variable used by the SQL_Query_XXX functions
          to build the query
        sort: the g.Sort variable passed in to your query by MMBatchList. Used
          to apply the current sorting scheme
        list: the list of code:table_column values. For example, if your
          mmbatchlist has a column code of client_name, but your table column
          is client.name, you would use the correlation
          client_name:client.name. This parameter is a white-list comma
          separated string of available sorting columns. Only columns listed
          here can be sorted using the MMBatchList
        default_sort: the default sortby column for your query. It is a good
          idea to ALWAYS supply a default sort, or your data might be
          displayed in an inconsistent manner
</MvCOMMENT>
<MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_OrderBy_Fields(
 l.search_query, g.Sort, 'code:s.code,amount:s.amount', 's.id' ) }">


<MvASSIGN NAME = "l.search_sql"               VALUE = "{ [ g.Module_Library_DB
 ].SQL_Query_Build( l.search_query, l.search_fields ) }">


<MvIF EXPR = "{ NOT [ g.Module_Library_DB ].SQL_Query_Count( l.search_query,
 l.total_count ) }">
    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
     g.Error_Code, g.Error_Message ) }">
<MvELSEIF EXPR = "{ NOT [ g.Module_Library_Native_DBAPI ].DB_OPENVIEW_Range(
 'Merchant', 'MyStatements', l.search_sql, l.search_fields, g.Offset, g.Count
 ) }">
```

```
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error( 'MY-
         STATEMENTS-00001', g.MvOPENVIEW_Error ) }">
    </MvIF>

    <MvASSIGN NAME = "l.count" VALUE = 0>

    <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Response_Start() }">
    {
        "data":
        [
        <MvWHILE EXPR = "{ ( NOT MyStatements.d.EOF ) AND ( ( g.Count EQ 0 ) OR (
         l.count LT g.Count ) ) }">
            <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_ArrayElement_Start( l.count )
              }">
                    "id":       <MvEVAL EXPR = "{ int( MyStatements.d.id ) }">,
                    "code":     "<MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Encode(
                      MyStatements.d.code ) }">",
                    "amount":   <MvEVAL EXPR = "{ MyStatements.d.amount ROUND 2
                      }">
            <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_ArrayElement_End() }">

            <MvSKIP NAME = "Merchant" VIEW = "MyStatements" ROWS = 1>
        </MvWHILE>
        ],
        "total_count": <MvEVAL EXPR = "{ int( l.total_count ) }">,
        "start_offset": <MvEVAL EXPR = "{ int( g.Offset ) }">
    }
    <MvCLOSEVIEW NAME = "Merchant" VIEW = "MyStatements">
    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_End() }">
</MvFUNCTION>
```

The following example shows how to add a JOIN to the query to add additional data that might be stored in another table.

```
<MvFUNCTION NAME = "JSON_MyStatementsList_Load_Query" PARAMETERS = "module var"
  STANDARDOUTPUTLEVEL = "text, html, compresswhitespace">
    <MvCOMMENT> Always Sanitize Incoming Data </MvCOMMENT>
    <MvASSIGN NAME = "g.Filter"          VALUE = "{ trim( g.Filter ) }">
    <MvASSIGN NAME = "g.Sort"            VALUE = "{ trim( g.Sort ) }">
    <MvASSIGN NAME = "g.Offset"          VALUE = "{ trim( g.Offset ) }">
    <MvASSIGN NAME = "g.Count"           VALUE = "{ trim( g.Count ) }">

    <MvASSIGN NAME = "l.search_query"    VALUE = "">

    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_SELECT( l.search_query,
     's.id, s.code, s.amount, client.name AS
     client_name' ) }">
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_FROM( l.search_query,
     g.Store_Table_Prefix $ 'MyStatements', 's' ) }">
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_FROM( l.search_query,
     g.Store_Table_Prefix $ 'MyStatements_Clients', 'client' ) }">

    <MvCOMMENT>
        SQL_Query_WHERE( query var, where, fields )
            query: query variable used by the SQL_Query_XXX functions to build the
              query
            where: where clause (ie, alias.column = ?, or alias1.column =
              alias2.column)
            fields: list of fields that correlate to the parameterized query (ie,
              if 'alias.column = ?' is supplied, then you would put
              'g.Column_Value'). Note, if using the DB_OPENVIEW_Range function,
              you MUST use a global variable for the fields values
    </MvCOMMENT>
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_WHERE( l.search_query,
     's.client_id = client.id', '' ) }">

    <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Filter( l.search_query, g.Filter,
     'code:s.code,amount:s.amount,client_name:client.name' ) }">
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_OrderBy_Fields(
     l.search_query, g.Sort, 'code:s.code,
     amount:s.amount,client_name:client.name', 's.id' ) }">

    <MvASSIGN NAME = "l.search_sql"              VALUE = "{ [ g.Module_Library_DB
     ].SQL_Query_Build( l.search_query, l.search_fields ) }">

    <MvIF EXPR = "{ NOT [ g.Module_Library_DB ].SQL_Query_Count( l.search_query,
     l.total_count ) }">
    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
     g.Error_Code, g.Error_Message ) }">
```

```
    <MvELSEIF EXPR = "{ NOT [ g.Module_Library_Native_DBAPI ].DB_OPENVIEW_Range(
     'Merchant', 'MyStatements', l.search_sql, l.search_fields, g.Offset, g.Count
     ) }">
    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
     'MY-STATEMENTS-00001', g.MvOPENVIEW_Error ) }">
    </MvIF>

    <MvASSIGN NAME = "l.count" VALUE = 0>

    <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Response_Start() }">
    {
        "data":
        [
        <MvWHILE EXPR = "{ ( NOT MyStatements.d.EOF ) AND ( ( g.Count EQ 0 )
         OR ( l.count LT g.Count ) ) }">
            <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_ArrayElement_Start( l.count )
              }">
                            "id":          <MvEVAL EXPR = "{ int(
                              MyStatements.d.id ) }">,
                            "code":         "<MvEVAL EXPR = "{ [ g.Module_JSON
                              ].JSON_Encode( MyStatements.d.code ) }">",
                            "client_name": "<MvEVAL EXPR = "{ [ g.Module_JSON
                              ].JSON_Encode( MyStatements.d.client_name ) }">",
                            "amount":      <MvEVAL EXPR = "{
                              MyStatements.d.amount ROUND 2 }">
            <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_ArrayElement_End() }">

            <MvSKIP NAME = "Merchant" VIEW = "MyStatements" ROWS = 1>
        </MvWHILE>
        ],
        "total_count": <MvEVAL EXPR = "{ int( l.total_count ) }">,
        "start_offset": <MvEVAL EXPR = "{ int( g.Offset ) }">
    }
    <MvCLOSEVIEW NAME = "Merchant" VIEW = "MyStatements">
    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_End() }">
</MvFUNCTION>
```

Note: For a complete list of available **SQL_Query_XXX** functions, refer to the MivaScript documentation for Miva Merchant 9.

Step 4: Initializing MMBatchList on document load

Using the **MMScreen_LoadFinished** function, initialize your derived **MMBatchList** class as shown in the code sample below. Once you initialize it, you do not need to do anything else.

Note: The **MMScreen_LoadFinished** function is called by the **MMScreen** class when the page has finished loading. All your javascript files, resources, and elements will be available.

```
<body>
    <script language="JavaScript">
        MMScreen_LoadFinished( function() { new MyStatementsBatchList(); } );
    </script>
</body>
```

Advanced MMBatchList

The remainder of this document discusses **MMBatchList** features in detail and how to implement them.

Enabling MMBatchList Features

MMBatchList is broken up into features for ease of use and efficiency. Features must be explicitly enabled. Following is a list of all available features. By default, these features are enabled. To disable any of the default-enabled features, you must derive from the **MMBatchList_NoFeatures** class and then manually enable all desired features.

- Feature: **ActionButton**
- Feature: **Add**
- Feature: **Buttons**
- Feature: **ColumnMove**
- Feature: **ColumnResize**
- Feature: **Delete**
- Feature: **DialogEdit**
- Feature: **Display Order**
- Feature: **Edit**
- Feature: **Export**
- Feature: **Find in List**
- Feature: **GoTo**
- Feature: **Header**
- Feature: **MultipleSelect**

- Feature: **On Demand Columns**
- Feature: **Pagination**
- Feature: **Persistent Filters**
- Feature: **RecordCount**
- Feature: **Row Double Click**
- Feature: **SearchBar**

Feature: ActionButton

The **ActionButton** feature creates a button on the left edge of each record and is used as the **GoTo** button, the **Error** button, and the **Cancel** button as well as providing other functionality for that row. The **ActionButton** feature should almost always be enabled in your batchlist. The only time it should not be is if you derive from the **MMBatchList_NoFeatures** class and do not enable any other features (for example, if you only want to display a basic scrollable list with no edit, add or delete functionality).

If enabled, the action button will be created for each record in the **NewRow** function and will be updated in the **BindRow** function.

To enable:

```
this.Feature_Header_Enable();
```

Feature: Add

The **Add** feature allows users to insert a record and save that record all from the batchlist without ever having to leave the screen. For an example of a sub-record **Add**, see *Enabling Sub-Record (nested) Support* on page 42.

To enable:

```
this.Feature_Add_Enable();
```

The following code samples shows a complete example of using the **Add** feature.

mystatementsbatchlist.js

```
function MyStatementsBatchList()
{
    MMBatchList.call( this, 'mystatements_batchlist_id' );

    <!-- START Add Specific Code -->
    this.Feature_Add_Enable();
    <!-- END Add Specific Code -->

    this.Feature_SearchBar_SetPlaceholderText( 'Search My Statements...' );
    this.SetDefaultSort( 'id', '' );
}
```

```
DeriveFrom( MMBatchList, MyStatementsBatchList );

MyStatementsBatchList.prototype.onLoad = MyStatementsList_Load_Query;

MyStatementsBatchList.prototype.onCreateRootColumnList = function()
{
    var columnlist =
    [
        new MMBatchList_Column_Code( 'Statement Code', 'code', 'MyStatement_Code'
         ),
        new MMBatchList_Column_Name( 'Client Name', 'client_name',
         'MyStatement_ClientName' ),
        new MMBatchList_Column_Currency( 'Statement Amount', 'amount',
         'MyStatement_Amount' )
    ];

    return columnlist;
}

<!-- START Add Specific Code -->
MyStatementsBatchList.prototype.onCreate = function()
{
    var record;

    /* ALL record elements should be created here */

    record              = new Object();
    record.id           = 0;
    record.code         = '';
    record.client_name  = '';
    record.amount       = 0.00;

    return record;
}

MyStatementsBatchList.prototype.onInsert = function( item, callback, delegator )
{
    MyStatement_Insert( item.record.mmbatchlist_fieldlist, callback, delegator );
}
<!-- END Add Specific Code -->
```

function.js

```
...
function MyStatement_Insert( fieldlist, callback, delegator )
{
    return AJAX_Call_Module_FieldList( callback,
                                       'admin',
                                       'mystatement_module_code',
                                          <!-- set up by my module -->
                                       'MyStatement_Insert',
                                       '',
                                       fieldlist,
                                       delegator );
}
...
```

Note: Place **your_module.mv** somewhere below the **Element_MMBatchList_HTML()** call.

your_module.mv

```
<MvFUNCTION NAME = "JSON_MyStatement_Insert" PARAMETERS = "module var"
 STANDARDOUTPUTLEVEL = "text, html, compresswhitespace">
    <MvIF EXPR = "{ NOT [ g.Module_JSON ].JSON_Store_Open() }">
     <MvFUNCTIONRETURN>  </MvIF>


    <MvCOMMENT>
        The g.MyStatement_Code/g.MyStatement_ClientName/g.MyStatement_Amount
         variable names were set up in the onCreateRootColumnList function
    </MvCOMMENT>


    <MvASSIGN NAME = "g.MyStatement_Code"           VALUE = "{ trim(
     g.MyStatement_Code ) }">
    <MvASSIGN NAME = "g.MyStatement_Amount"         VALUE = "{ trim(
     g.MyStatement_Amount ) }">
    <MvASSIGN NAME = "g.MyStatement_ClientName"     VALUE = "{ trim(
     g.MyStatement_ClientName ) }">


    <MvIF EXPR = "{ ISNULL g.MyStatement_Code }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_FieldError(
         'MyStatement_Code', 'Code cannot be empty' ) }">
    </MvIF>


    <MvIF EXPR = "{ ISNULL g.MyStatement_ClientName }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_FieldError(
         'MyStatement_ClientName', 'Client Name cannot be empty' ) }">
    </MvIF>
    <MvIF EXPR = "{ NOT [ g.Module_Admin ].Validate_Currency_NonNegative_Required(
     g.MyStatement_Amount ) }">
```

```
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_FieldError(
         'MyStatement_Amount', g.Validation_Message ) }">
    </MvIF>
    <MvASSIGN NAME = "l.mystatement:code"       VALUE = "{ g.MyStatement_Code }">
    <MvASSIGN NAME = "l.mystatement:amount"     VALUE = "{ g.MyStatement_Amount }">
    <MvASSIGN NAME = "l.mystatement:clientname" VALUE = "{
     g.MyStatement_ClientName }">

    <MvIF EXPR = "{ NOT MyStatement_Insert( l.mystatement ) }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
         g.Error_Code, g.Error_Message ) }">
    </MvIF>

    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Success() }">
</MvFUNCTION>
```

Feature: Buttons

The **Buttons** feature is enabled by default on all derived **MMBatchList** classes. The **Buttons** feature allows dynamic and persistent buttons to be displayed for the batchlist. Dynamic buttons are only displayed when certain conditions are met (usually a record or group of records is selected) and hidden when those conditions are not met (no records selected). Persistent buttons are and should always be visible. They can, however, be visible and disabled (see the **Product Attributes** screen for an example of this).

There are numerous button functions available for use, including normal buttons, toggle buttons, dropdown buttons, and upload buttons. There are also dynamic buttons specific to each type of action (add mode, edit mode, delete mode, etc.) that will be displayed only when buttons for that action type are displayed. For example, if you create an edit mode button, that button will only be displayed when there is one or more records in edit mode.

To enable:

```
this.Feature_Buttons_Enable();
```

Creating a button is simple. Choose a button function that you wish to create (refer to the list of available functions below) and initialize it. If you need to access the button later, save it to a class member.

Example:

```
...
this.button_create_bitmap = this.Feature_Buttons_AddButton_Persistent(
 'Create Bitmap', 'Create a bitmap image of this record\'s svg title', 'add',
 this.Create );
...
```

In the preceding example, `this.Create` is a prototype function on the derived class. That function will be called in the context of the batchlist, meaning you will have access to all of your class members from within the called function (in other words, no need for an anonymous function `function() { self.Create(); }` to get the class context)

Following is a list of available button creation functions. This list may change and grow over time.

- **Feature_Buttons_AddButton_Persistent(text, hover_text, type, onclick)**
- **Feature_Buttons_AddButton_Persistent_Toggle(text, hover_text, type, default_pressed, onstatepressed, onstateup)**
- **Feature_Buttons_AddButton_Persistent_Dropdown(text, hover_text, type)**
- **Feature_Buttons_AddButton_Persistent_Upload(text, hover_text, type, onchange)**
- **Feature_Buttons_AddButton_Dynamic(text, hover_text, type, onclick)**
- **Feature_Buttons_AddButton_Dynamic_Upload(text, hover_text, type, support_multiple, onchange)**
- **Feature_Buttons_AddButton_Dynamic_Toggle(text, hover_text, type, default_pressed, onstatepressed, onstateup)**
- **Feature_Buttons_AddButton_Dynamic_AddMode(text, hover_text, type, onclick)**
- **Feature_Buttons_AddButton_Dynamic_EditMode(text, hover_text, type, onclick)**
- **Feature_Buttons_AddButton_Dynamic_DisplayOrder(text, hover_text, type, onclick)**
- **Feature_Buttons_AddButton_Dynamic_DeleteMode(text, hover_text, type, onclick)**
- **Feature_Buttons_AddButton_Dynamic_Error(text, hover_text, type, onclick)**

Feature: ColumnMove

The **ColumnMove** feature allows users of the batch list to reorder all columns (drag them to change position). Enabling this feature will also enable the **Header** feature (if not already enabled).

To enable:

```
this.Feature_ColumnMove_Enable();
```

Feature: ColumnResize

The **ColumnResize** feature allows users of the batch list to resize all columns. Enabling this feature will also enable the **Header** feature (if not already enabled).

To enable:

```
this.Feature_ColumnResize_Enable();
```

Feature: Delete

The **Delete** feature allows users to select one or more records (if multiple select is enabled) and delete them from the database table(s). For an example of a sub-record **Delete**, see *Enabling Sub-Record (nested) Support* on page 42.

To enable:

```
this.Feature_Delete_Enable();
```

The following code samples show a complete example of using the **Delete** feature.

mystatementsbatchlist.js

```
function MyStatementsBatchList()
{
    MMBatchList.call( this, 'mystatements_batchlist_id' );

    <!-- START Delete Specific Code -->
    this.Feature_Delete_Enable();
    <!-- END Delete Specific Code -->

    this.Feature_SearchBar_SetPlaceholderText( 'Search My Statements...' );
    this.SetDefaultSort( 'id', '' );
}

DeriveFrom( MMBatchList, MyStatementsBatchList );

MyStatementsBatchList.prototype.onLoad = MyStatementsList_Load_Query;

MyStatementsBatchList.prototype.onCreateRootColumnList = function()
{
    var columnlist =
    [
        new MMBatchList_Column_Code( 'Statement Code', 'code', 'MyStatement_Code'
         ),
        new MMBatchList_Column_Name( 'Client Name', 'client_name',
         'MyStatement_ClientName' ),
        new MMBatchList_Column_Currency( 'Statement Amount', 'amount',
         'MyStatement_Amount' )
    ];

    return columnlist;
}

<!-- START Delete Specific Code -->
MyStatementsBatchList.prototype.onDelete = function( item, callback,
 delegator )
{
    MyStatement_Delete( item.record.id, callback, delegator );
}
<!-- END Delete Specific Code -->
```

function.js

```
...
function MyStatement_Delete( id, callback, delegator )
{
    return AJAX_Call_Module( callback,
                             'admin',
                             'mystatement_module_code',
                               <!-- set up by my module -->
                             'MyStatement_Delete',
                             'MyStatement_ID=' + encodeURIComponent( id ),
                             delegator );
}
...
```

Note: Place **your_module.mv** somewhere below the **Element_MMBatchList_HTML()** call.

your_module.mv

```
<MvFUNCTION NAME = "JSON_MyStatement_Delete" PARAMETERS = "module var"
 STANDARDOUTPUTLEVEL = "text, html, compresswhitespace">
    <MvIF EXPR = "{ NOT [ g.Module_JSON ].JSON_Store_Open() }">
     <MvFUNCTIONRETURN>  </MvIF>

    <MvASSIGN NAME = "g.MyStatement_ID" VALUE = "{ int( g.MyStatement_ID ) }">

    <MvIF EXPR = "{ MyStatement_Load_ID( g.MyStatement_ID, l.mystatement ) }">
        <MvIF EXPR = "{ NOT MyStatement_Delete_ID( l.mystatement:id ) }">
            <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
              g.Error_Code, g.Error_Message ) }">
        </MvIF>
    </MvIF>

    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Success() }">
</MvFUNCTION>
```

Feature: DialogEdit

The **DialogEdit** feature allows records to be edited in a dialog. It adds a button to the dynamic action bar that when clicked (or the row is double clicked) triggers the overridden “onEdit” function, where you would put the dialog open commands.

To enable:

```
this.Feature_EditDialog_Enable( hover_text, text );
 /* hover_text is the hover text of the button */
 /* and text is the normal display text of the button */
```

Feature: Display Order

The **Display Order** feature allows users to change the display order of items (note: your database must support this). These edits will then be saved in the database table(s) through an AJAX request. For an example of a subrecord edit, see *Enabling Sub-Record (nested) Support* on page 42.

To enable:

```
this.Feature_DisplayOrder_Enable( 'display_order_sort',
 'root_display_order_prefix' );
 <!-- display_order_sort is the column code where the display order data is
 stored (ie, disp_order), root_display_order_prefix is the prefix assigned for
 saving (similar to the field_name in MMBatchList_Column) -->
```

The following code samples shows a complete example of using the **Display Order** feature.

mystatementsbatchlist.js

```
function MyStatementsBatchList()
{
    MMBatchList.call( this, 'mystatements_batchlist_id' );

    <!-- START Display Order Specific Code -->
    this.Feature_DisplayOrder_Enable( 'disp_order', 'MyStatements_Order' );
    <!-- END Display Order Specific Code -->

    this.Feature_SearchBar_SetPlaceholderText( 'Search My Statements...' );
    this.SetDefaultSort( 'id', '' );
}

DeriveFrom( MMBatchList, MyStatementsBatchList );

MyStatementsBatchList.prototype.onLoad = MyStatementsList_Load_Query;

MyStatementsBatchList.prototype.onCreateRootColumnList = function()
{
    var columnlist =
    [
        new MMBatchList_Column_Code( 'Statement Code', 'code',
          'MyStatement_Code' ),
        new MMBatchList_Column_Name( 'Client Name', 'client_name',
          'MyStatement_ClientName' ),
        new MMBatchList_Column_Currency( 'Statement Amount', 'amount',
          'MyStatement_Amount' )
    ];

    return columnlist;
}
```

```
<!-- START Display Order Specific Code -->
MyStatementsBatchList.prototype.onDisplayOrderSave = function( fieldlist, callback
 )
{
    /* Your display order update function will receive a list of changed display
     * orders. The list will be in the following format:
     *
     * root_item[ index ]:id
     * root_item[ index ]:offset
     * root_item[ index ]:original_offset
     * child_item[ root_item_index ][ index ]:id
     * child_item[ root_item_index ][ index ]:offset
     * child_item[ root_item_index ][ index ]:original_offset
     * childs_child_item[ root_item_index ][ child_item_index ][ index ]:id
     * childs_child_item[ root_item_index ][ child_item_index ][ index ]:offset
     * childs_child_item[ root_item_index ][ child_item_index ][ index
     *    ]:original_offset
     * ... etc.
     */
    MyStatementsList_DisplayOrder_Update( fieldlist, callback );
}
<!-- END Display Order Specific Code -->
```

function.js

```
...
function MyStatementsList_DisplayOrder_Update( fieldlist, callback )
{
    return AJAX_Call_Module_FieldList( callback,
                                       'admin',
                                       'mystatement_module_code',
                                         <!-- set up by my module -->
                                       'MyStatementsList_DisplayOrder_Update',
                                       '',
                                       fieldlist );
}
...
```

Note: Place **your_module.mv** somewhere below the **Element_MMBatchList_HTML()** call.

your_module.mv

```
<MvFUNCTION NAME = "JSON_MyStatementsList_DisplayOrder_Update" PARAMETERS = ""
 STANDARDOUTPUTLEVEL = "text, html, compresswhitespace">
    <MvIF EXPR = "{ NOT JSON_Store_Open() }">
     <MvFUNCTIONRETURN>  </MvIF>


    <MvFOREACH ITERATOR = "l.mystatement_order" ARRAY = "g.MyStatements_Order"
     INDEX = "l.pos">
```

```
        <MvASSIGN NAME = "l.mystatement_order:id"               VALUE = "{ trim(
         l.mystatement_order:id ) }">
        <MvASSIGN NAME = "l.mystatement_order:offset"           VALUE = "{ trim(
         l.mystatement_order:offset ) }">
        <MvASSIGN NAME = "l.mystatement_order:original_offset"  VALUE = "{ trim(
         l.mystatement_order:original_offset ) }">


        <MvIF EXPR = "{ NOT [ g.Filename_Admin
         ].Validate_WholeNumber_Positive_Required( l.mystatement_order:offset )
         }">
            <MvFUNCTIONRETURN VALUE = "{ JSON_Response_FieldError(
              'g.MyStatements_Order[' $ l.pos $ ']', g.Validation_Message ) }">
        </MvIF>
    </MvFOREACH>

    <MvASSIGN NAME = "l.change_count"                             VALUE = 0>

    <MvFOREACH ITERATOR = "l.mystatement_order" ARRAY = "g.MyStatements_Order">
        <MvIF EXPR = "{ l.mystatement_order:offset EQ
         l.mystatement_order:original_offset }">
            <MvFOREACHCONTINUE>
        </MvIF>

        <MvASSIGN NAME = "l.change_count"                         VALUE = "{
         l.change_count + 1 }">
        <MvASSIGN NAME = "l.changes" INDEX = "{ l.change_count }" VALUE = "{
         l.mystatement_order }">
    </MvFOREACH>

    <MvIF EXPR = "{ l.change_count EQ 0 }">
        <MvFUNCTIONRETURN VALUE = "{ JSON_Response_Success() }">
    </MvIF>

    <MvCOMMENT>
        HERE IS WHERE YOU WOULD PUT YOUR OFFSET UPDATING CODE
    </MvCOMMENT>

    <MvFUNCTIONRETURN VALUE = "{ JSON_Response_Success() }">
</MvFUNCTION>
```

Feature: Edit

The Edit feature allows users to select one or more records (if multiple select is enabled) and edit them inline. These edits will then be saved in the database table(s) through an AJAX request. For an example of a subrecord **Edit**, see *Enabling Sub-Record (nested) Support* on page 42.

To enable:

```
this.Feature_Edit_Enable();
```

The following code samples show a complete example of using the **Edit** feature.

mystatementsbatchlist.js

```
function MyStatementsBatchList()
{
    MMBatchList.call( this, 'mystatements_batchlist_id' );

    <!-- START Edit Specific Code -->
    this.Feature_Add_Enable();
    <!-- END Edit Specific Code -->

    this.Feature_SearchBar_SetPlaceholderText( 'Search My Statements...' );
    this.SetDefaultSort( 'id', '' );
}

DeriveFrom( MMBatchList, MyStatementsBatchList );

MyStatementsBatchList.prototype.onLoad = MyStatementsList_Load_Query;

MyStatementsBatchList.prototype.onCreateRootColumnList = function()
{
    var columnlist =
    [
        new MMBatchList_Column_Code( 'Statement Code', 'code',
         'MyStatement_Code' ),
        new MMBatchList_Column_Name( 'Client Name', 'client_name',
         'MyStatement_ClientName' ),
        new MMBatchList_Column_Currency( 'Statement Amount', 'amount',
         'MyStatement_Amount' )
    ];

    return columnlist;
}
<!-- START Edit Specific Code -->
MyStatementsBatchList.prototype.onSave = function( item, callback, delegator )
{
    MyStatement_Update( item.record.id, item.record.mmbatchlist_fieldlist,
     callback, delegator );
}
<!-- END Edit Specific Code -->
```

functions.js

```
...
function MyStatement_Update( id, fieldlist, callback, delegator )
{
    return AJAX_Call_Module_FieldList( callback,
                                       'admin',
                                       'mystatement_module_code',
                                          <!-- set up by my module -->
                                       'MyStatement_Update',
                                       'MyStatement_ID=' +
                                          encodeURIComponent( id ),
                                        fieldlist,
                                        delegator );
}
...
```

Note: Place **your_module.mv** somewhere below the **Element_MMBatchList_HTML()** call.

your_module.mv

```
<MvFUNCTION NAME = "JSON_MyStatement_Update" PARAMETERS = "module var"
 STANDARDOUTPUTLEVEL = "text, html, compresswhitespace">
    <MvIF EXPR = "{ NOT [ g.Module_JSON ].JSON_Store_Open() }">
     <MvFUNCTIONRETURN>  </MvIF>


    <MvASSIGN NAME = "g.MyStatement_ID"           VALUE = "{ int( g.MyStatement_ID
     ) }">
    <MvCOMMENT>
        The g.MyStatement_Code/g.MyStatement_ClientName/g.MyStatement_Amount
         variable names were set up in the onCreateRootColumnList function
    </MvCOMMENT>
    <MvASSIGN NAME = "g.MyStatement_Code"           VALUE = "{ trim(
     g.MyStatement_Code ) }">
    <MvASSIGN NAME = "g.MyStatement_Amount"         VALUE = "{ trim(
     g.MyStatement_Amount ) }">
    <MvASSIGN NAME = "g.MyStatement_ClientName"     VALUE = "{ trim(
     g.MyStatement_ClientName ) }">


    <MvIF EXPR = "{ NOT JSON_MyStatement_Load( g.MyStatement_ID,
     l.mystatement ) }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
         g.Error_Code, g.Error_Message ) }">
    </MvIF>


    <MvIF EXPR = "{ ISNULL g.MyStatement_Code }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_FieldError(
         'MyStatement_Code', 'Code cannot be empty' ) }">
    </MvIF>
```

```
    <MvIF EXPR = "{ ISNULL g.MyStatement_ClientName }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_FieldError(
         'MyStatement_ClientName', 'Client Name cannot be empty' ) }">
    </MvIF>

    <MvIF EXPR = "{ NOT [ g.Module_Admin ].Validate_Currency_NonNegative_Required(
     g.MyStatement_Amount ) }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_FieldError(
         'MyStatement_Amount', g.Validation_Message ) }">
    </MvIF>

    <MvASSIGN NAME = "l.mystatement:code"       VALUE = "{ g.MyStatement_Code }">
    <MvASSIGN NAME = "l.mystatement:amount"     VALUE = "{ g.MyStatement_Amount }">
    <MvASSIGN NAME = "l.mystatement:clientname" VALUE = "{
     g.MyStatement_ClientName }">

    <MvIF EXPR = "{ NOT MyStatement_Update_ID( l.mystatement:id, l.mystatement )
     }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
         g.Error_Code, g.Error_Message ) }">
    </MvIF>

    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Success() }">
</MvFUNCTION>
```

Feature: Export

The **Export** feature is enabled by default in all derived **MMBatchList** classes. It allows selected records to be downloaded into a CSV file. Only visible columns will be downloaded. **MMBatchList_Column** classes can override the **Export Data** function using the **SetOnExportData(onexport)** function.

To enable:

```
this.Feature_Export_Enable();
```

Feature: Find In List

The **Find in List** feature is a sub-feature of **SearchBar**. This feature is not enabled by default in the derived **MMBatchList** class. **Find In List** allows users to enter search queries into the search bar and jump to matching results instead of winnowing down the data like normal search does. The **Find In List** lists must implement the **onLoadRecordIndex** function in order to function correctly. An example of this is the **Products** batch list. Choose the **Find In List** option from the search dropdown while on the **Products** list screen, enter a search query, and use the **Previous**/**Next** buttons to jump to matching results. If no results are found, a notification will temporarily be displayed showing the lack of matches.

To enable:

```
this.Feature_SearchBar_Enable_FindInList();
```

Enabling **Find In List** is relatively straight forward and can be broken down into the following steps:

1. Enable the feature in your **MMBatchList** class;
2. Override the necessary **MMBatchList** class function (**onLoadRecordIndex**);
3. Implement JSON code to handle the index searching.

Step 1: Enable the feature in your MMBatchList class

```
function MyStatementsBatchList()
{
    MMBatchList.call( this, 'mystatements_batchlist_id' );
    ...
    <!-- START Find In List Specific Code -->
    this.Feature_SearchBar_Enable_FindInList();
    <!-- END Find In List Specific Code -->
    ...
```

Step 2: Override the necessary MMBatchList class function (onLoadRecordIndex)

mystatementsbatchlist.js

```
function MyStatementsBatchList()
{
    MMBatchList.call( this, 'mystatements_batchlist_id' );

    <!-- START Find In List Specific Code -->
    this.Feature_SearchBar_Enable_FindInList();
    <!-- END Find In List Specific Code -->

    this.Feature_SearchBar_SetPlaceholderText( 'Search My Statements...' );
    this.SetDefaultSort( 'id', '' );
}

DeriveFrom( MMBatchList, MyStatementsBatchList );

MyStatementsBatchList.prototype.onLoad = MyStatementsList_Load_Query;

MyStatementsBatchList.prototype.onCreateRootColumnList = function()
{
    var columnlist =
    [
        new MMBatchList_Column_Code( 'Statement Code', 'code', 'MyStatement_Code'
         ),
        new MMBatchList_Column_Name( 'Client Name', 'client_name',
         'MyStatement_ClientName' ),
```

```
        new MMBatchList_Column_Currency( 'Statement Amount', 'amount',
         'MyStatement_Amount' )
    ];

    return columnlist;
}

<!-- START Find In List Specific Code -->
MyStatementsBatchList.prototype.onLoadRecordIndex = function( record )
{
    MyStatementIndex_Load_ID( record.id, this.filter, this.sort_direction +
     this.sort_field, callback );
}
<!-- END Find In List Specific Code -->

...

function MyStatementIndex_Load_ID( id, filter, sort, callback, delegator )
{
    return AJAX_Call_Module( callback,
                             'admin',
                             '<MvEVAL EXPR = "{ g.Encoded_Module_Code }">',
                             'MyStatementIndex_Load_ID',
                             'Statement_ID=' + encodeURIComponent( id ) +
                               '&Filter=' + EncodeArray( filter ) + '&Sort=' +
                               encodeURIComponent( sort ),
                             delegator );
}
```

Step 3: Implement JSON code to handle the custom filtering

```
<MvCOMMENT> IN MODULE_JSON CODE </MvCOMMENT>
<MvFUNCTION NAME = "JSON_MyStatementsList_Load_Query" PARAMETERS = "module var"
 STANDARDOUTPUTLEVEL = "text, html, compresswhitespace">
    <MvASSIGN NAME = "g.Filter"               VALUE = "{ trim( g.Filter ) }">
    <MvASSIGN NAME = "g.Sort"                 VALUE = "{ trim( g.Sort ) }">
    <MvASSIGN NAME = "g.Offset"               VALUE = "{ trim( g.Offset ) }">
    <MvASSIGN NAME = "g.Count"                VALUE = "{ trim( g.Count ) }">

    <MvASSIGN NAME = "l.search_query"         VALUE = "">

    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_SELECT( l.search_query,
     's.id, s.code, s.amount' ) }">
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_FROM( l.search_query,
     g.Store_Table_Prefix $ 'MyStatements', 's' ) }">

    <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Filter( l.search_query, g.Filter,
     'code:s.code,amount:s.amount' ) }">
```

```
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_OrderBy_Fields(
     l.search_query, g.Sort, 'code:s.code,amount:s.amount', 's.id' ) }">

    <MvASSIGN NAME = "l.search_sql"    VALUE = "{ [ g.Module_Library_DB
     ].SQL_Query_Build( l.search_query, l.search_fields ) }">

    <MvIF EXPR = "{ NOT [ g.Module_Library_DB ].SQL_Query_Count( l.search_query,
     l.total_count ) }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
         g.Error_Code, g.Error_Message ) }">
    <MvELSEIF EXPR = "{ NOT [ g.Module_Library_Native_DBAPI ].DB_OPENVIEW_Range(
     'Merchant', 'MyStatements', l.search_sql, l.search_fields, g.Offset, g.Count
     ) }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error( 'MY-
         STATEMENTS-00001', g.MvOPENVIEW_Error ) }">
    </MvIF>

    <MvASSIGN NAME = "l.count" VALUE = 0>

    <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Response_Start() }">
    {
        "data":
        [
        <MvWHILE EXPR = "{ ( NOT MyStatements.d.EOF ) AND ( ( g.Count EQ 0 ) OR (
         l.count LT g.Count ) ) }">
            <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_ArrayElement_Start( l.count )
              }">
               "id":          <MvEVAL EXPR = "{ int( MyStatements.d.id ) }">,
               "code":        "<MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Encode(
                MyStatements.d.code ) }">",
               "amount":      <MvEVAL EXPR = "{ MyStatements.d.amount ROUND 2 }">
            <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_ArrayElement_End() }">

            <MvSKIP NAME = "Merchant" VIEW = "MyStatements" ROWS = 1>
        </MvWHILE>
        ],
        "total_count": <MvEVAL EXPR = "{ int( l.total_count ) }">,
        "start_offset": <MvEVAL EXPR = "{ int( g.Offset ) }">
    }
    <MvCLOSEVIEW NAME = "Merchant" VIEW = "MyStatements">
    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_End() }">
</MvFUNCTION>

<MvFUNCTION NAME = "JSON_MyStatementIndex_Load_ID" PARAMETERS = "module var"
 STANDARDOUTPUTLEVEL = "text, html, compresswhitespace">
    <MvIF EXPR = "{ g.Session_Type NE 'admin' }">       <MvFUNCTIONRETURN>  </MvIF>
```

```
<MvIF EXPR = "{ NOT [ g.Module_JSON ].JSON_Store_Open() }">
 <MvFUNCTIONRETURN>  </MvIF>
<MvIF EXPR = "{ ( NOT g.Admin_User_Administrator ) AND
( g.Admin_User_ID NE g.Store:manager_id ) }">       <MvFUNCTIONRETURN>  </MvIF>

<MvASSIGN NAME = "l.query"              VALUE = "">
<MvASSIGN NAME = "l.record"             VALUE = "">
<MvASSIGN NAME = "g.Statement_ID"       VALUE = "{ int( g.Statement_ID ) }">
<MvASSIGN NAME = "g.Filter"             VALUE = "{ trim( g.Filter ) }">
<MvASSIGN NAME = "g.Sort"               VALUE = "{ trim( g.Sort ) }">

<MvREFERENCE NAME = "l.statement"       VARIABLE = "l.record:s">

<MvIF EXPR = "{ NOT Statement_Load_ID( g.Statement_ID, l.statement ) }">
    <MvIF EXPR = "{ [ g.Module_Library_DB ].Error_Is_EOF() }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
          '#Error#', 'Statement ' $ g.Statement_ID $ ' not found' ) }">
    </MvIF>

<MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
 g.Error_Code, g.Error_Message ) }">
</MvIF>

<MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_FROM( l.query,
 g.Store_Table_Prefix $ 'MyStatements', 's' ) }">

<MvCOMMENT>
|
| NOTE: the JSON_Filter and SQL_Query_OrderBy_Fields correlation columns MUST
| contain the table alias (ie, [table_alias].[column_name]) for the matching to
| work correctly.
|
</MvCOMMENT>

<MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Filter( l.search_query, g.Filter,
 'code:s.code,amount:s.amount' ) }">
<MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_OrderBy_Fields(
 l.search_query, g.Sort, 'code:s.code,amount:s.amount', 's.id' ) }">

<MvIF EXPR = "{ NOT [ g.Module_Library_DB ].SQL_Query_Index( l.query,
 l.record, l.index ) }">
    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
     g.Error_Code, g.Error_Message ) }">
</MvIF>


<MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Response_Start() }">
```

```
    {
        "index":    <MvEVAL EXPR = "{ int( l.index ) }">
    }
    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_End() }">
</MvFUNCTION>
```

Feature: GoTo

The **GoTo** feature allows users to hover over a row and select the **GoTo** button in the row, or select a single record and push the **GoTo** button in the Button bar. The **GoTo** button should follow convention and either take you to a new screen, or open a dialog.

Note: This feature is a "Go To" action, not a "do something else" action. Follow convention to prevent confusion.

When enabling this feature, you can pass an optional "path" variable that will override the default **GoTo** button image. If you want to change the icon at a later time, you can call the **Feature_GoTo_SetImage(path)** function. Note, however, that this will update all rows and the Button bar **GoTo** button to use this new icon (in other words, it is not a "per-row" change).

To enable:

```
this.Feature_GoTo_Enable( path );
 <!-- path is the default image path to be displayed in the button -->
```

Feature: Header

The **Header** feature shows the header row. It is enabled by default when deriving from the **MMBatchList** class. Displaying the header row allows users to see header text for data specific to a column as well as allowing users to reorder and resize columns (if those features are also enabled). If you wish for your batchlist to not display a header row, you must derive from the **MMBatchList_NoFeatures** class and ***not*** enable the following features: **ActionButton**, **ColumnResize**, **ColumnMove** and **Display Order.**

To enable:

```
this.Feature_Header_Enable();
```

Feature: MultipleSelect

The **MultipleSelect** feature is enabled by default in derived **MMBatchList** classes. When enabled, users can select multiple records at once.

To enable:

```
this.Feature_MultipleSelect_Enable();
```

Feature: On Demand Columns

Enabling this feature will send a list of visible columns in the Filter (**g.Filter**) parameter to be used in the JSON functions. When this feature is enabled, only data for columns passed in the **ondemandcolumns** filter (comma separated string) should be loaded and returned. The **l.displayable_fields** can then be matched later to determine whether or not a value should be loaded/displayed.

Currently this feature is only used in conjunction with Custom Fields, so that stores using hundreds of custom fields that will never be displayed in the list do not load/pass unnecessary amounts of data on every call.

To enable:

```
this.Feature_OnDemandColumns_Enable();
```

The following example shows how to parse on demand columns in MivaScript JSON code.

```
<MvCOMMENT>
|
|   Build Custom Field Displayable data from MMBatchList "ondemandcolumns"
|
</MvCOMMENT>


<MvFOREACH ITERATOR = "l.filter" ARRAY = "l.filters" COUNT = "{ JSON_Array_String(
 l.filterlist, l.filters ) }">
    <MvASSIGN NAME = "l.filter_name"    VALUE = "{ trim( decodeattribute( gettoken(
     l.filter, ':', 1 ) ) ) }">
    <MvASSIGN NAME = "l.filter_value"   VALUE = "{ trim( decodeattribute( gettoken(
     l.filter, ':', 2 ) ) ) }">


    <MvIF EXPR = "{ l.filter_name EQ 'ondemandcolumns' }">
        <MvFOREACH ITERATOR = "l.column_code" ARRAY = "l.column_codes" COUNT = "{
         [ g.Module_Library_Utilities ].SplitStringAndTrim( l.filter_value, ',',
         l.column_codes ) }">
            <MvIF EXPR = "{ ISNULL l.column_code }">
                <MvFOREACHCONTINUE>
            </MvIF>


            <MvREFERENCEARRAY NAME = "l.displayable" VARIABLE =
              "l.displayable_fields">
                <MvMEMBER NAME = "{ l.column_code }">
            </MvREFERENCEARRAY>


            <MvASSIGN NAME = "l.displayable" VALUE = 1>
        </MvFOREACH>
    </MvIF>
</MvFOREACH>
```

Feature: Pagination

The Pagination feature turns on Pagination mode — turning off infinite scroll and providing a list of page controls.

Note: There is a user preference to determine which mode to use (Pagination or Infinite Scroll). Therefore, manually triggering this option should be avoided.

To enable:

```
this.Feature_Pagination_Enable();
```

Feature: Persistent Filters

The **Persistent Filters** feature is used when the normal search and advanced search functionality is not enough. The **Persistent Filters** feature must be used in conjunction with the MivaScript **MMBatchListPersistentFilters_Begin(l.prefix)** and **MMBatchListPersistentFilters_End()** functions. You place your custom persistent filters code inside those two functions.

An example of this would be the **Order Processing** screen. On the order processing screen we have added a quick and convenient way to filter data by date range, batch, payment, status, etc. This is made convenient by using a simple drop-down select instead of going into Advanced Settings and setting multiple fields to get the same results. If your list has similar needs, the **Persistent Filters** feature is where this would be implemented.

To enable:

```
this.Feature_Persistent_Filters_Enable( 'prefix_value' );
 <!-- this prefix must match the prefix specified in
 MMBatchListPersistentFilters_Begin -->
```

Enabling custom filtering can be broken down into the following steps:

1. Output the necessary HTML elements (using the **MMBatchListPersistentFilters_Begin/End** functions)
2. Override the necessary **MMBatchList** class functions and initialize the data
3. Implement JSON code to handle the custom filtering

These steps are detailed in the following sections.

Step 1: Output the necessary HTML elements

There are two required MivaScript functions:

- **MMBatchListPersistentFilters_Begin(prefix)**
- **MMBatchListPersistentFilters_End()**

In between those two functions you must set up your custom elements (selects, inputs, etc). Keep in mind that space is limited and styling should match the rest of the batchlist. The following code sample shows how to use this in a real world setting.

Note: Place **your_module.mv** somewhere below the **Element_MMBatchList_HTML()** call.

your_module.mv

```
...
<MvEVAL EXPR = "{ MMBatchListPersistentFilters_Begin( 'mystatements_filters' ) }">
    <select id="mystatementslist_filter_client_show"></select>
    <select id="mystatementslist_filter_client_type"></select>
<MvEVAL EXPR = "{ MMBatchListPersistentFilters_End() }">
...
```

Step 2: Override the necessary MMBatchList class functions and initialize the data

Persistent filters require you to override the **onPersistentFiltersSetContent**, **Feature_SearchBar_onSearch_GetFilter**, and **Feature_SearchBar_AdvancedSearchSet** functions in your derived **MMBatchList** class. See the following code sample for an example.

mystatementsbatchlist.js

```
function MyStatementsBatchList()
{
    MMBatchList.call( this, 'mystatements_batchlist_id' );

    <!-- START Persistent Filter Specific Code -->
    this.Feature_Persistent_Filters_Enable( 'mystatements_filters' );
    <!-- END Persistent Filter Specific Code -->

    this.Feature_SearchBar_SetPlaceholderText( 'Search My Statements...' );
    this.SetDefaultSort( 'id', '' );
}

DeriveFrom( MMBatchList, MyStatementsBatchList );

MyStatementsBatchList.prototype.onLoad = MyStatementsList_Load_Query;

MyStatementsBatchList.prototype.onCreateRootColumnList = function()
{
    var columnlist =
    [
        new MMBatchList_Column_Code( 'Statement Code', 'code', 'MyStatement_Code'
         ),
        new MMBatchList_Column_Name( 'Client Name', 'client_name',
         'MyStatement_ClientName' ),
        new MMBatchList_Column_Currency( 'Statement Amount', 'amount',
         'MyStatement_Amount' )
    ];
    return columnlist;
}
```

```
<!-- START Persistent Filter Specific Code -->
MyStatementsBatchList.prototype.onPersistentFiltersSetContent = function()
{
    var self = this;

    this.mystatementsbatchlist_filter_client_show               =
     document.getElementById( 'mystatementslist_filter_client_show' );
    this.mystatementsbatchlist_filter_client_show.add( new Option( '<Select One>',
     '' ) );
    this.mystatementsbatchlist_filter_client_show.add( new Option( 'All', 'All' )
     );
    this.mystatementsbatchlist_filter_client_show.add( new Option( 'Active',
     'Active' ) );
    this.mystatementsbatchlist_filter_client_show.onchange      = function() {
     self.onSearch(); };

    this.mystatementsbatchlist_filter_client_type               =
     document.getElementById( 'mystatementslist_filter_client_type' );
    this.mystatementsbatchlist_filter_client_type.add( new Option( '<Select One>',
     '' ) );
    this.mystatementsbatchlist_filter_client_type.add( new Option( 'Registered',
     'R' ) );
    this.mystatementsbatchlist_filter_client_type.add( new Option( 'Unregistered',
     'U' ) );
    this.mystatementsbatchlist_filter_client_type.onchange      = function() {
     self.onSearch(); };

    this.mystatementsbatchlist_filter_client_expired            =
     document.getElementById( 'mystatementslist_filter_client_type' );
    this.mystatementsbatchlist_filter_client_expired.add( new Option( '<Select
     One>', '' ) );
    this.mystatementsbatchlist_filter_client_expired.add( new Option( 'Yes', 'Y' )
     );
    this.mystatementsbatchlist_filter_client_expired.add( new Option( 'No', 'N' )
     );
    this.mystatementsbatchlist_filter_client_expired.onchange   = function() {
     self.onSearch(); };
}

MyStatementsBatchList.prototype.Feature_SearchBar_onSearch_GetFilter = function()
{
    var filterlist, from;

    filterlist =
     MMBatchList_NoFeatures.prototype.Feature_SearchBar_onSearch_GetFilter.call(
     this );

    if ( filterlist.length )
    {
```

```
        return filterlist;
    }

    <!-- Shows three different ways of implementing the filter -->
    filterlist = new Array();
    filterlist.push( 'Client_Show:' + encodeURIComponent(
     this.mystatementsbatchlist_filter_client_show.value ) );

    if ( this.mystatementsbatchlist_filter_client_type.value != '' )
    {
        filterlist.push( 'advancedsearch:client_type' + encodeURIComponent( ':EQ:'
         ) + encodeURIComponent(
         this.mystatementsbatchlist_filter_client_type.value ) );
    }

    if ( this.mystatementsbatchlist_filter_client_expired.value != '' )
    {
        filterlist.push( 'advancedsearch:client_expired' + encodeURIComponent(
         ':EQ:' ) + encodeURIComponent(
         this.mystatementsbatchlist_filter_client_expired.value ) );
    }

    return filterlist;
}

MyStatementsBatchList.prototype.Feature_SearchBar_AdvancedSearchSet = function(
 set )
{
    MMBatchList_NoFeatures.prototype.Feature_SearchBar_AdvancedSearchSet.call(
     this, set );

    <!-- If using the advanced search dialog, we want to reset all the persistent
     filters or they will interfere with our advanced search -->
    if ( set )
    {
        this.mystatementsbatchlist_filter_client_show.selectedIndex      = 0;
        this.mystatementsbatchlist_filter_client_type.selectedIndex      = 0;
        this.mystatementsbatchlist_filter_client_expired.selectedIndex   = 0;

        this.mystatementsbatchlist_filter_client_show.disabled       = true;
        this.mystatementsbatchlist_filter_client_type.disabled       = true;
        this.mystatementsbatchlist_filter_client_expired.disabled    = true;
    }
    else
    {
        this.mystatementsbatchlist_filter_client_show.disabled       = false;
        this.mystatementsbatchlist_filter_client_type.disabled       = false;
```

```
        this.mystatementsbatchlist_filter_client_expired.disabled    = false;
    }
}
<!-- END Persistent Filter Specific Code -->
```

Step 3: Implement JSON code to handle the custom filtering

Note: Place **your_module.mv** somewhere below the **Element_MMBatchList_HTML()** call.

```
<MvCOMMENT> IN MODULE_JSON CODE </MvCOMMENT>
<MvFUNCTION NAME = "JSON_MyStatementsList_Load_Query" PARAMETERS = "module var"
 STANDARDOUTPUTLEVEL = "text, html, compresswhitespace">
    <MvASSIGN NAME = "g.Filter"               VALUE = "{ trim( g.Filter ) }">
    <MvASSIGN NAME = "g.Sort"                 VALUE = "{ trim( g.Sort ) }">
    <MvASSIGN NAME = "g.Offset"               VALUE = "{ trim( g.Offset ) }">
    <MvASSIGN NAME = "g.Count"                VALUE = "{ trim( g.Count ) }">

    <MvASSIGN NAME = "l.search_query"         VALUE = "">

    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_SELECT( l.search_query,
     's.id, s.code, s.amount, client.name AS client_name' ) }">
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_FROM( l.search_query,
     g.Store_Table_Prefix $ 'MyStatements', 's' ) }">
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_FROM( l.search_query,
     g.Store_Table_Prefix $ 'MyStatements_Clients', 'client' ) }">
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_WHERE( l.search_query,
     's.client_id = client.id', '' ) }">

    <MvCOMMENT>
        For Persistent Filters, since we are passing in a custom filter, we need to
         handle it in a custom way... User JSON_Filter_Callback
        For the client_type filter, notice we can simply use the correlation list,
         but for the client_show filter, we must use the
         JSON_MyStatementsList_Load_Query_Filter function
    </MvCOMMENT>

    <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Filter_Callback( l.search_query,
     g.Filter, 'code:s.code,amount:s.amount,client_name:client.name',
     g.Module_Root $ l.module:module,  'JSON_MyStatementsList_Load_Query_Filter',
     'JSON_MyStatementsList_Load_Query_AdvancedSearchFilter', l.null ) }">
    <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_OrderBy_Fields(
     l.search_query, g.Sort,
     'code:s.code,amount:s.amount,client_name:client.name,client_type:client.type
     ', 's.id' ) }">

    <MvASSIGN NAME = "l.search_sql"           VALUE = "{ [ g.Module_Library_DB
     ].SQL_Query_Build( l.search_query, l.search_fields ) }">
```

```
    <MvIF EXPR = "{ NOT [ g.Module_Library_DB ].SQL_Query_Count( l.search_query,
     l.total_count ) }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error(
         g.Error_Code, g.Error_Message ) }">
    <MvELSEIF EXPR = "{ NOT [ g.Module_Library_Native_DBAPI ].DB_OPENVIEW_Range(
     'Merchant', 'MyStatements', l.search_sql, l.search_fields, g.Offset, g.Count
     ) }">
        <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_Error( 'MY-
         STATEMENTS-00001', g.MvOPENVIEW_Error ) }">
    </MvIF>

    <MvASSIGN NAME = "l.count" VALUE = 0>

    <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Response_Start() }">
    {
        "data":
        [
        <MvWHILE EXPR = "{ ( NOT MyStatements.d.EOF ) AND ( ( g.Count EQ 0 ) OR (
         l.count LT g.Count ) ) }">
            <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_ArrayElement_Start( l.count )
              }">
               "id":           <MvEVAL EXPR = "{ int( MyStatements.d.id ) }">,
               "code":        "<MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Encode(
                MyStatements.d.code ) }">",
               "client_name": "<MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_Encode(
                MyStatements.d.client_name ) }">",
               "amount":       <MvEVAL EXPR = "{ MyStatements.d.amount ROUND 2
                }">
            <MvEVAL EXPR = "{ [ g.Module_JSON ].JSON_ArrayElement_End() }">

            <MvSKIP NAME = "Merchant" VIEW = "MyStatements" ROWS = 1>
        </MvWHILE>
        ],
        "total_count": <MvEVAL EXPR = "{ int( l.total_count ) }">,
        "start_offset": <MvEVAL EXPR = "{ int( g.Offset ) }">
    }
    <MvCLOSEVIEW NAME = "Merchant" VIEW = "MyStatements">
    <MvFUNCTIONRETURN VALUE = "{ [ g.Module_JSON ].JSON_Response_End() }">
</MvFUNCTION>

<MvFUNCTION NAME = "JSON_MyStatementsList_Load_Query_Filter" PARAMETERS = "query
 var, field_count var, filter_name, filter_value, data var" STANDARDOUTPUTLEVEL =
 "">
    <MvASSIGN NAME = "l.filter_name"    VALUE = "{ toupper( l.filter_name ) }">
    <MvASSIGN NAME = "l.filter_value"   VALUE = "{ toupper( l.filter_value ) }">

    <MvIF EXPR = "{ l.filter_name NE 'CLIENT_SHOW' }">
```

```
        <MvFUNCTIONRETURN VALUE = 0>
    </MvIF>

    <MvIF EXPR = "{ l.filter_value EQ 'ALL' }">
    <MvELSE>
        <MvEVAL EXPR = "{ [ g.Module_Library_DB ].SQL_Query_WHERE( l.query,
         '( client.active = 1 )', '' ) }">
    </MvIF>

    <MvCOMMENT>
        RETURNING 1 CAUSES THE JSON_Filter_Callback FUNCTION TO CONTINUE ON TO THE
         NEXT FILTER VALUE INSTEAD OF ATTEMPTING TO PROCESS THIS ONE.
    </MvCOMMENT>

    <MvFUNCTIONRETURN VALUE = 1>
</MvFUNCTION>

<MvFUNCTION NAME = "JSON_MyStatementsList_Load_Query_AdvancedSearchFilter"
 PARAMETERS = "query var, field_count var, filter_name, filter_operator,
 filter_value, data var" STANDARDOUTPUTLEVEL = "">
    <MvIF EXPR = "{ tolower( l.filter_name ) NE 'client_expired' }">
        <MvFUNCTIONRETURN VALUE = 0>
    </MvIF>

    <MvIF EXPR = "{ tolower( l.filter_value ) EQ 'n' }">       <MvEVAL EXPR = "{ [
     g.Module_Library_DB ].SQL_Query_WHERE( l.query, 'client.enabled = 0', '' )
     }">
    <MvELSEIF EPXR = "{ tolower( l.filter_value ) EQ 'y' }">    <MvEVAL EXPR = "{ [
     g.Module_Library_DB ].SQL_Query_WHERE( l.query, 'client.enabled = 1', '' )
     }">
    </MvIF>

    <MvCOMMENT>
        RETURNING 1 CAUSES THE JSON_Filter_Callback FUNCTION TO CONTINUE ON TO THE
         NEXT ADVANCED FILTER VALUE INSTEAD OF ATTEMPTING TO PROCESS THIS ONE.
    </MvCOMMENT>

    <MvFUNCTIONRETURN VALUE = 1>
</MvFUNCTION>
```

Feature: RecordCount

The **RecordCount** feature allows users to append a record count to the batch list. Enabling this feature places a bar at the bottom of every batch list with the total number of records in the list and the index range that is currently visible. For example, “1-12 of 500”.

To enable:

```
this.Feature_RecordCount_Enable();
```

Feature: Row Double Click

The **Row Double Click** feature allows an action to be performed when double clicking a row. If you are using the **Edit** feature in conjunction with the **Row Double Click** feature, by default the **onRowDoubleClick** function will enter Edit mode for the double clicked row. If your batch list requires different functionality or does not support the **Edit** feature, you must override the **onRowDoubleClick** function.

To enable:

```
this.Feature_RowDoubleClick_Enable();
```

The following code sample shows an example of using the **Row Double Click** feature.

mystatementsbatchlist.js

```
function MyStatementsBatchList()
{
    MMBatchList.call( this, 'mystatements_batchlist_id' );

    <!-- START Row Double Click Specific Code -->
    this.Feature_RowDoubleClick_Enable();
    <!-- END Row Double Click Specific Code -->

    this.Feature_SearchBar_SetPlaceholderText( 'Search My Statements...' );
    this.SetDefaultSort( 'id', '' );
}

DeriveFrom( MMBatchList, MyStatementsBatchList );

MyStatementsBatchList.prototype.onLoad = MyStatementsList_Load_Query;

MyStatementsBatchList.prototype.onCreateRootColumnList = function()
{
    var columnlist =
    [
        new MMBatchList_Column_Code( 'Statement Code', 'code', 'MyStatement_Code'
         ),
        new MMBatchList_Column_Name( 'Client Name', 'client_name',
         'MyStatement_ClientName' ),
        new MMBatchList_Column_Currency( 'Statement Amount', 'amount',
         'MyStatement_Amount' )
    ];

    return columnlist;
```

```
}

<!-- START Row Double Click Specific Code -->
MMBatchList_NoFeatures.prototype.onRowDoubleClick = function( item )
{
    // Custom code here
}
```

Feature: SearchBar

The **SearchBar** feature is enabled by default in the derived **MMBatchList** class. The **SearchBar** allows users to enter search queries to filter results in the batch list.

To enable:

```
this.Feature_SearchBar_Enable();
```

Accessing Records

The following terms are used in this section:

- **record** – the JavaScript object that contains all the data loaded in the **Load Query** function.
- **item** – the JavaScript object that holds a reference to all the important data for each record. Think of this "item" as a parent container that hold references to the record, row, branch, etc. For example, to access the HTML row for a given record (if that record is visible), you would use **item.row** to return that element. **item.record** returns the record for that row.
- **branch** – the JavaScript object that holds a reference to all the branch data (mostly important if using sub records).

Accessing records from **MMBatchList** functions can be done in a couple of different ways (depending on what function you are in). Some functions, such as **Record Save**, **Record Insert**, etc., pass the item object and you can access the record directly through **item.record**. Other functions will not pass the data. If you want to access another record's data, you will need to use one of the following functions:

GetListItem(index)	returns item.
GetListItemRecord(index)	returns item record.
GetListItem_Root(index)	returns the root level item (use for finding the root parent of the child item that is at the "index"). If the index you pass in is the root element, that item will be returned.
GetListItemRecord_Root(index)	returns root level item record (use for finding the root parent of the child item that is at the "index").
GetListItem_Parent(index)	returns parent item of the current index. Returns null if there is no parent.
GetListItemRecord_Parent(index)	returns parent item record of the current index. Returns null if there is no parent.

GetListItem_PreviousSibling(index)	returns item of the next element at that level.
GetListItemRecord_PreviousSibling(index)	returns item.

The following example shows how to get an item if you know the index.

```
MyStatementsBatchList.prototype.ButtonClick_DoSomething = function()
{
    var item = this.GetListItem( 7 );
}
```

The following example shows how to get the item when a single record is selected in the batch list.

```
MyStatementsBatchList.prototype.ButtonClick_DoSomething = function()
{
    var item;

    if ( this.SingleRecordSelected() )
    {
        if ( ( item = this.ActiveItemList_ItemAtIndex( 0 ) ) !== null )
        {
            // Do something with the item
        }
    }
}
```

Note: We used the **ActiveItemList_ItemAtIndex(index)** function instead of one of the **GetListItemXXX** functions. The active item list is a list of all currently selected items. This is probably the most common use case for the **ActiveItemList_XXX** functions.

Enabling Assign Lists

The **Assign List** is a derived **MMBatchList** class with some extra features enabled to allow for easy and standards-conforming implementations of lists that will have data "assigned" to some parent item. For example, if you edit a single product and then go to its **Related Products** tab, you will be presented with a list of products. You can then assign/unassign each of those products, individually or in a batch, to the parent product (the product you are editing). This functionality has been extended for third party use. Use the following code samples as a guide.

mystatement_assigneditems.js

```
function MyStatementAssignedItemList( mystatement_id )
{
    /* Instead of deriving from the MMBatchList class, we will derive from the
     MMBatchList_AssignList class */
    MMBatchList_AssignList.call( this, 'my_statement_assigneditems', true );
```

```
    /* We save the mystatement_id, which is the "parent" of our assigned items.
     AKA, all items that we assign in this list will be assigned to this
     mystatement_id record */
    this.mystatement_id = mystatement_id;

    /* the "this.button_assignlist" class member is a reference to the "Show All,
     Show Assigned, Show Unassigned" dropdown list button. You can change the hover
     text to be specific to your case (default is "Show Records") */
    if ( this.button_assignlist )
    {
        this.button_assignlist.ContainedButton().SetHoverText( 'Show My Statement
         Items' );
    }

    this.Feature_SearchBar_SetPlaceholderText( 'Search My Statement Items...' );
    this.SetDefaultSort( 'id', '' );
}

DeriveFrom( MMBatchList_AssignList, MyStatementAssignedItemList );

MyStatementAssignedItemList.prototype.onLoad = function( filter, sort, offset,
 count, callback, delegator )
{
    /* Since we have a special case here, we need to provide a custom onLoad
     function that calls into our load query function with different parameters.
        this.mystatement_id is the parent id we will link to in our load function
        this.load_assigned tells us whether we want to load all assigned records
        this.load_unassigned tess us whether we want to load all unassigned
         records

        the load_assigned and load_unassigned are not mutually exclusive. If both
         are set to true, you should return all results from the database */

    MyStatementAssignedItemList_Load_Query(
        this.mystatement_id,
        this.load_assigned,
        this.load_unassigned,
        search,
        filter,
        sort,
        offset,
        count,
        callback,
        delegator
    );
}
```

```
/* The onSaveAssigned function is called any time we change the "assigned" status
 of a record. If in Edit mode, this "assigned" state will be saved before we save
 the remaining record. If in normal select mode, changing the "assigned" state will
 call just this onSaveAssigned function (and not update the rest of the record) */
MyStatementAssignedItemList.prototype.onSaveAssigned = function( item, callback,
 delegator )
{
    MyStatementAssignedItem_Update_Assigned( this.mystatement_id, item.record.id,
     item.record.assigned, callback, delegator );
}


MyStatementAssignedItemList.prototype.onCreateRootColumnList = function()
{
    var columnlist =
    [
        new MMBatchList_Column_AssignListCheckbox( 'Assigned', 'assigned',
         'MyStatementItem_Assigned', this ).SetSortByField( '' ).SetSearchable(
         false ) );
        new MMBatchList_Column_Code( 'Item Code', 'code', 'MyStatementItem_Code'
         ),
        new MMBatchList_Column_Name( 'Item Name', 'name', 'MyStatementItem_Name' )
    ];

    return columnlist;
}
```

function.js

```
function MyStatementAssignedItemList_Load_Query( mystatement_id, assigned,
 unassigned, filter, sort, offset, count, callback, delegator )
{
    return AJAX_Call( callback, 'admin', 'MyStatementAssignedItemList_Load_Query',
                'MyStatement_ID=' + encodeURIComponent( mystatement_id )  +
                '&Filter='        + EncodeArray( filter )                 +
                '&Sort='          + encodeURIComponent( sort )            +
                '&Offset='        + encodeURIComponent( offset )          +
                '&Count='         + encodeURIComponent( count )           +
                '&Assigned='      + ( assigned ? '1' : '0' )              +
                '&Unassigned='    + ( unassigned ? '1' : '0' ),
                delegator );
}


function MyStatementAssignedItem_Update_Assigned( mystatement_id, related_id,
 assigned, callback, delegator )
{
    return AJAX_Call( callback, 'admin',
     'MyStatementAssignedItem_Update_Assigned',
            'MyStatement_ID='             + encodeURIComponent( mystatement_id ) +
```

```
            '&MyStatementAssignedItem_ID=' + encodeURIComponent( related_id ) +
            '&Assigned='                   + ( assigned ? '1' : '0' ),
            delegator );
}
```

Enabling Sub-Record (nested) Support

Using sub-records with a derived **MMBatchList** can be complex. It helps to keep all like functions grouped together when they are declared. In some cases, you may require the use of hook functions.

Take the Product Attributes Batch List as an example. With the inclusion of attribute templates displayed within the batchlist, you need to insure that those are not editable inline, as normal attributes and options are. With a combination of hook functions, you can create a system that leaves no holes where a user must refresh the page to reset the list.

Adding sub-records ("children") is accomplished with the use of "branches". Every derived **MMBatchList** class will have at least one branch – the **branch_root**. Branches work just like a family tree, everything ties back to the root branch. A branch layout may look like any of the following:

```
branch_root
    branch
    branch
        branch
            branch
    branch
    branch
```

—OR—

```
branch_root
    branch
        branch
```

—OR—

```
branch_root
```

—OR—

```
...
```

MMBatchList supports an infinite number of branches, provided you handle the necessary overrides (described below), as long as they can all be linked back up to the root—that is, every child has a parent. Be sure to save the branch object (returned from function **SetColumnBranch**) for later use, as you will need this to set up any features.

The following code example shows how to use sub-records.

mystatement_attributes.js

```
function MyStatementAttributeList( mystatement_id )
{
    MMBatchList.call( this, 'mm9_batchlist_productattributelist' );

    /*
    this.branch_root: The root level branch, this contains our top level elements
    this.branch_child1: The first child of root
    this.branch_child1_child: the child of branch_child

    Picture it like this:
        branch_root
            branch_child1
                branch_child1_child

Having multiple children can be helpful if you have unique data structures such as:

    data:       // Root level data structure would be associated with "branch_root"
    [
        {
            id: 1,
            mystatement_id: 1,
            code: 'test1',
            name: 'Test 1',
            attributes:             // attributes associated with "branch_child1"
            [
                {
                                id: 1,
                                code: 'blah1',
                                options:
                                  // attributes associated with "branch_child1_child"
                                [
                                    {
                                        ...
                                    },
                                    ...
                                ]
                },
                ...
            ]
        },
        ...
    ]
    */
```

```
this.mystatement_id                = mystatement_id;
this.branch_child1                 = this.AddBranch(
 this.CreateColumnList_Child1(),        'attributes', this.branch_root );
this.branch_child1_child           = this.AddBranch(
 this.CreateColumnList_Child1_Child(),  'options', this.branch_child1 );

/* Display Order Specific functions */
this.Feature_DisplayOrder_Enable( 'disp_order', 'Attribute_Order' );

this.Branch_SetDisplayOrderPrefix( this.branch_child1, 'Attribute_Order' );
this.Branch_SetDisplayOrderPrefix( this.branch_child1_child, 'Option_Order' );

/* Add Specific functions */

this.button_add_child1_child       =
 this.Feature_Buttons_AddButton_Persistent_Toggle( 'Add Option', 'Add Option
 (Child of Attribute)', '', false, function() { self.Feature_Add_Cancel();
 self.button_add_child1_child.Toggle_StatePressed();
 self.Feature_Add_Insert(); }, function() { self.Feature_Add_Cancel(); } );
this.button_add_child1             =
 this.Feature_Buttons_AddButton_Persistent_Toggle( 'Add Attribute', 'Add
 Attribute (Child of root item)', '', false, function() {
 self.Feature_Add_Cancel(); self.button_add_child1.Toggle_StatePressed();
 self.Feature_Add_Insert(); }, function() { self.Feature_Add_Cancel(); } );

this.Feature_Add_Enable();

/* branch_root uses function onCreate, so we don't need to set a create
 function */
this.Branch_SetCreateFunction( this.branch_child1, this.Attribute_Create );
this.Branch_SetCreateFunction( this.branch_child1_child, this.Option_Create );

/* branch_root uses function onInsert, so we don't need to set an insert
 function */
this.Branch_SetInsertFunction( this.branch_child1, this.Attribute_Insert );
this.Branch_SetInsertFunction( this.branch_child1_child, this.Option_Insert );

/* branch_root uses function onFindIndex_Params, so we don't need to set a
 FindIndex_Params function */
this.Branch_SetFindIndex_ParamsFunction( this.branch_child1,
 this.Attribute_FindIndex_Params );
this.Branch_SetFindIndex_ParamsFunction( this.branch_child1_child,
 this.Option_FindIndex_Params );

/* branch_root uses function onFindIndex_Compare, so we don't need to set a
 FindIndex_Compare function */
this.Branch_SetFindIndex_CompareFunction( this.branch_child1,
 this.Attribute_FindIndex_Compare );
```

```
    this.Branch_SetFindIndex_CompareFunction( this.branch_child1_child,
     this.Option_FindIndex_Compare );

    /* Edit Specific functions */
    this.Feature_Edit_Enable();

    /* branch_root uses function onSave, so we don't need to set a save function */
    this.Branch_SetSaveFunction( this.branch_child1, this.Attribute_Save );
    this.Branch_SetSaveFunction( this.branch_child1_child, this.Option_Save );

    this.Feature_RowDoubleClick_Enable();

    /* Delete Specific functions */
    this.Feature_Delete_Enable();

    /* branch_root uses function onDelete, so we don't need to set a delete
     function */
    this.Branch_SetDeleteFunction( this.branch_child1, this.Attribute_Delete );
    this.Branch_SetDeleteFunction( this.branch_child1_child, this.Option_Delete );

    if ( this.button_inlineadd_add )
    {
        this.button_inlineadd_add.SetText( 'Add Attribute' );
        this.button_inlineadd_add.SetImage( '' );
        this.button_inlineadd_add.SetHoverText( 'Add Product Attribute' );
    }

    this.Feature_SearchBar_SetPlaceholderText( 'Search My Statement Attributes' );
    this.SetDefaultSort( 'disp_order', '' );
}

DeriveFrom( MMBatchList, MyStatementAttributeList );

MyStatementAttributeList.prototype.onLoad = function( filter, sort, offset, count,
 callback, delegator )
{
    return MyStatementAttributeAndOptionList_Load_Query( this.mystatement_id,
     filter, sort, offset, count, callback, delegator );
}

MyStatementAttributeList.prototype.onProcessLoadedData = function( recordlist,
 start_index )
{
    /*
    In a normal batch list with no child records, there is no need to override this
     function. However,since there can potentially be great complexity in the
     structure of sub-record lists, we passed the processing of loaded data (how it
```

```
 gets inserted into the list) on to you through the use of this
 onProcessLoadedData.

Remember, our data structure looks something like this, so we'll need to parse
 out the data.
recordlist:
[
    {
        id: 1,
        mystatement_id: 1,
        code: 'test1',
        name: 'Test 1',
        attributes:
        [
            {
                id: 1,
                code: 'blah1',
                name: 'Blah 1',
                options:
                [
                     {
                        id: 9,
                        code: 'option_1',
                        name: 'Option 1
                     },
                     {
                        id: 10,
                        code: 'option_2',
                        name: 'Option 2
                     }
                ]
            }
        ]
    }
]
*/

var i, j, k, index, root_index, attribute_index;

index = start_index;

for ( i = 0; i < recordlist.length; i++ )
{
    root_index = index;
```

```
            /* ItemList_CreateInsertAtIndex( record, index, parent_index, branch ) */
            this.ItemList_CreateInsertAtIndex( recordlist[ i ], index++, -1,
             this.branch_root );

            if ( recordlist[ i ].attributes )
            {
                for ( j = 0; j < recordlist[ i ].attributes.length; j++ )
                {
                    attribute_index = index;
                    this.ItemList_CreateInsertAtIndex( recordlist[ i ].attributes
                        [ j ], index++, root_index, this.branch_child1 );

                    if ( recordlist[ i ].attributes[ j ].options )
                    {
                          for ( k = 0; k < recordlist[ i ].attributes
                           [ j ].options.length; k++ )
                          {
                             this.ItemList_CreateInsertAtIndex( recordlist[
                              i ].attributes[ j ].options[ k ], index++,
                              attribute_index, this.branch_child1_child );
                          }
                    }
                }
            }
        }
    }

    MyStatementAttributeList.prototype.onSetDisplayOrder = function( recordlist,
     start_index )
    {
        /*
        In a normal batch list with no child records, there is no need to override this
         function. However,since there can potentially be great complexity in the
         structure of sub-record lists, we passed the display orderprocessing of
         loaded data (how the display order value is set) on to you through the use of
         this onSetDisplayOrder.
        Each level should start at 1 and continue through N.

        NOTE: This function only needs to be overridden if the Display Order feature is
         enabled
        */

        var i, j, k;

        for ( i = 0; i < recordlist.length; i++ )
        {
```

```
        this.Feature_DisplayOrder_SetRecordOrder( recordlist[ i ], start_index + i
         + 1 );

        if ( recordlist[ i ].attributes )
        {
            for ( j = 0; j < recordlist[ i ].attributes.length; j++ )
            {
                    this.Feature_DisplayOrder_SetRecordOrder( recordlist[ i ],
                     j + 1 );

                    if ( recordlist[ i ].attributes[ j ].options )
                    {
                       for ( k = 0; k < recordlist[ i ].attributes[ j
                        ].options.length; k++ )
                       {
                              this.Feature_DisplayOrder_SetRecordOrder(
                                recordlist[ i ], k + 1 );
                       }
                 }
            }
        }
    }
}

MyStatementAttributeList.prototype.onDisplayOrderSave = function( fieldlist,
 callback )
{
    /* fieldlist is a list of namevaluepairs as a JavaScript array. */
    MyStatementAttributeList_DisplayOrder_Update( this.mystatement_id, fieldlist,
     callback );
}

MyStatementAttributeList.prototype.onRetrieveChildBranch = function( item )
{
    if ( item.branch == this.branch_child1_child )  return null;
     /* has no children */
    else if ( item.branch == this.branch_child1 )   return
     this.branch_child1_child;    /* has one child branch */
    else if ( item.branch == this.branch_root )    return this.branch_child1;
     /* has one child branch */
}

MyStatementAttributeList.prototype.onCreate = function()
{
    /* onCreate is used for the branch_root only */
    var record;
```

```
    record                      = new Object();
    record.mystatement_id       = this.mystatement_id;
    record.id                   = 0;
    record.code                 = '';
    record.name                 = '';
    record.attributes           = new Array();

    return record;
}

MyStatementAttributeList.prototype.Attribute_Create = function()
{
    var record;

    record          = new Object();
    record.id       = 0;
    record.code     = '';
    record.name     = '';
    record.options  = new Array();

    return record;
}

MyStatementAttributeList.prototype.Option_Create = function()
{
    var record;

    record          = new Object();
    record.id       = 0;
    record.code     = '';
    record.name     = '';

    return record;
}

MyStatementAttributeList.prototype.onInsert = function( item, callback, delegator
 )
{
    /* onInsert is used for the branch_root only */
    Attribute_Insert( this.mystatement_id, item.record.mmbatchlist_fieldlist,
     callback, delegator );
}

MyStatementAttributeList.prototype.Attribute_Insert = function( item, callback,
 delegator )
{
```

```
    var mystatement_record, error;

    if ( ( mystatement_record = this.GetListItemRecord_Parent( item.index ) ) ==
     null )
    {
        error                           = new Object();
        error.validation_error          = true;
        error.error_field_message       = 'MyStatement record not found';
        error.error_field               = 'Code';

        return onerror( error );
    }

    /* MyStatementAttribute_Insert( mystatement_id, mystatement_record_id,
     fieldlist, callback, delegator ); */
    MyStatementAttribute_Insert( this.mystatement_id, mystatement_record.id,
     item.record.mmbatchlist_fieldlist, callback, delegator );
}

MyStatementAttributeList.prototype.Option_Insert = function( item, callback,
 delegator )
{
    var mystatement_record, mystatementattribute_record, error;

    if ( ( mystatementattribute_record = this.GetListItemRecord_Parent( item.index
     ) ) == null )
    {
        error                       = new Object();
        error.validation_error      = true;
        error.error_field_message   = 'MyStatementAttribute record not found';
        error.error_field           = 'Code';

        return onerror( error );
    }

    if ( ( mystatement_record = this.GetListItemRecord_Parent(
     mystatementattribute_record.index ) ) == null )
    {
        error                           = new Object();
        error.validation_error          = true;
        error.error_field_message       = 'MyStatement record not found';
        error.error_field               = 'Code';

        return onerror( error );
    }
```

```
    /* MyStatementOption_Insert( mystatement_id, mystatement_record_id,
     mystatementattribute_id, fieldlist, callback, delegator ); */
    MyStatementOption_Insert( this.mystatement_id, mystatement_record.id,
     mystatementattribute_record.id, item.record.mmbatchlist_fieldlist, callback,
     delegator );
}

/*
 FindIndex_Params and FindIndex_Compare functions are used in conjunction with one
 another. This functionality was added into the BatchList class to allow you to
 scroll to the last inserted item's position within the list. This is helpful if
 the list is sorted in a way that differs from the default "display order" sort
 (ie, you insert a record and it is no longer in the spot you inserted it at
 because the list is sorted by name alphabetically).

 FindIndex_Params allows you to pass back a search parameter based on your inserted
 item and compare that data against each of the loaded items. Remember, this
 compares ONLY against LOADED items. If your inserted item is NOT returned in the
 data, a match will not be found.

 FindIndex_Compare takes the current item in the list (as it iterates through) and
 attempts to match against your parameters. Returning true means a match was found,
 returning false means no match was found

 !!!!THE FindIndex_Params/FindIndex_Options FUNCTIONS ARE NOT REQUIRED!!!!

 Below is an example of us finding a match based on the statement code
*/
MyStatementAttributeList.prototype.onFindIndex_Params = function( item )
{
    /* onFindIndex_Params is used for the branch_root only */
    return { 'MyStatement_Code': item && item.record ? item.record.code : '' };
}

MyStatementAttributeList.prototype.Attribute_FindIndex_Params = function( item )
{
    var mystatement_record = this.GetListItemRecord_Parent( item.index );

    return { 'MyStatementRecord_ID': mystatement_record ? mystatement_record.id :
     '', 'MyStatementAttribute_Code': item && item.record ? item.record.code : ''
     };
}

MyStatementAttributeList.prototype.Option_FindIndex_Params = function( item )
{
    var mystatement_record = this.GetListItemRecord_Parent( item.index );
    var mystatementattribute_record = mystatement_record ?
     this.GetListItemRecord_Parent( mystatement_record.index ) : null;
```

```
    return {
        'MyStatementRecord_ID': mystatement_record ? mystatement_record.id : '',
        'MyStatementAttribute_Code': mystatementattribute_record ?
         mystatementattribute_record.code : '',
        'MyStatementOption_Code': item && item.record ? item.record.code : ''
    };
}

MyStatementAttributeList.prototype.onFindIndex_Compare = function( item, params )
{
    /* onFindIndex_Compare is used for the branch_root only */
    if ( item && item.record && ( item.record.code == params[ 'MyStatement_Code' ]
     ) )
    {
        return true;
    }

    return false;
}

MyStatementAttributeList.prototype.Attribute_FindIndex_Compare = function( item,
 params )
{
    var mystatement_record = this.GetListItemRecord_Parent( item.index );

    if ( mystatement_record &&
        ( mystatement_record.id == params[ 'MyStatementRecord_ID' ] ) &&
        ( item.record.code == params[ 'MyStatementAttribute_Code' ] ) )
    {
        return true;
    }

    return false;
}

MyStatementAttributeList.prototype.Option_FindIndex_Compare = function( item,
 params )
{
    var mystatement_record = this.GetListItemRecord_Parent( item.index );
    var mystatementattribute_record = mystatement_record ?
     this.GetListItemRecord_Parent( mystatement_record.index ) : null;

    if ( attribute_record &&
        ( mystatement_record.id == params[ 'MyStatementRecord_ID' ] ) &&
        ( mystatementattribute_record.code == params[ 'MyStatementAttribute_Code'
         ] ) &&
```

```
        ( item.record.code == params[ 'MyStatementOption_Code' ] ) )
    {
        return true;
    }

    return false;
}

MyStatementAttributeList.prototype.onSave = function( item, callback, delegator )
{
    /* onSave is used for the branch_root only */
    MyStatement_Update( item.record.id, item.record.mmbatchlist_fieldlist,
     callback, delegator );
}

MyStatementAttributeList.prototype.Attribute_Save = function( item, callback,
 delegator )
{
    MyStatementAttribute_Update( item.record.id,
     item.record.mmbatchlist_fieldlist, callback, delegator );
}

MyStatementAttributeList.prototype.Option_Save = function( item, callback,
 delegator )
{
    MyStatementOption_Update( item.record.id, item.record.mmbatchlist_fieldlist,
     callback, delegator );
}

MyStatementAttributeList.prototype.onDelete = function( item, callback, delegator
 )
{
    /* onDelete is used for the branch_root only */
    MyStatement_Delete( item.record.id, callback, delegator );
}

MyStatementAttributeList.prototype.Attribute_Delete = function( item, callback,
 delegator )
{
    MyStatementAttribute_Delete( item.record.id, callback, delegator );
}

MyStatementAttributeList.prototype.Option_Delete = function( item, callback,
 delegator )
{
    MyStatementOption_Delete( item.record.id, callback, delegator );
}
```

```
MyStatementAttributeList.prototype.Feature_Add_EnableDisableButtons_Hook =
 function()
{
    /* We override the Feature_Add_EnableDisableButtons_Hook function here so we
     can set the enabled/disabled state of our add buttons
     We only want the sub record add buttons enabled if we are selecting its parent
     */
    var item;

    MMBatchList_NoFeatures.prototype.Feature_Add_EnableDisableButtons_Hook.call(
     this );

    if ( this.SingleRecordSelected() )
    {
        if ( ( item = this.ActiveItemList_ItemAtIndex( 0 ) ) === null )
        {
            return;
        }

        if ( item.branch === this.branch_root )  this.button_add_child1.Enable();
        else                                     this.button_add_child1.Disable();

        if ( item.branch === this.branch_child1 )
         this.button_add_child1_child.Enable();
        else
         this.button_add_child1_child.Disable();
    }
}

MyStatementAttributeList.prototype.onCreateRootColumnList = function()
{
    this.rootcolumn_code = new MMBatchList_Column_Code( 'Code', 'code',
     'MyStatement_Code' ).SetContentAttributeList( { 'class':
     'mm9_batchlist_level_col' } );
    this.rootcolumn_name = new MMBatchList_Column_Name( 'Name', 'name',
     'MyStatement_Name' ).SetContentAttributeList( { 'class':
     'mm9_batchlist_level_col' } );

    var columnlist =
    [
        this.rootcolumn_code,
        this.rootcolumn_name
    ];

    return columnlist;
}
```

```
MyStatementAttributeList.prototype.CreateColumnList_Child1 = function()
{
    var columnlist =
    [
        new MMBatchList_Column_Code( 'Code', 'code', 'MyStatementAttribute_Code' )
          .SetContentAttributeList( { 'class': 'mm9_batchlist_level_col' } )
          .SetRootColumn( this.rootcolumn_code ),
        new MMBatchList_Column_Name( 'Name', 'name', 'MyStatementAttribute_Name' )
          .SetContentAttributeList( { 'class': 'mm9_batchlist_level_col' } )
          .SetRootColumn( this.rootcolumn_name )
    ];

    return columnlist;
}

MyStatementAttributeList.prototype.CreateColumnList_Child1_Child = function()
{
    var columnlist =
    [
        new MMBatchList_Column_Code( 'Code', 'code', 'MyStatementOption_Code' )
          .SetContentAttributeList( { 'class': 'mm9_batchlist_level_col' } )
          .SetRootColumn( this.rootcolumn_code ),
        new MMBatchList_Column_Name( 'Name', 'name', 'MyStatementOption_Name' )
          .SetContentAttributeList( { 'class': 'mm9_batchlist_level_col' } )
          .SetRootColumn( this.rootcolumn_name )
    ];

    return columnlist;
}
```

Hook Functions

Hook functions allow you to handle custom actions during certain key phases of **MMBatchList**'s code execution. For example, if the **Edit** feature is enabled but there are certain rows you do not want to allow to be edited (for example. you do not want child records to be editable), you can add a hook to disable editing on a per-row basis. The following sections describe the currently available hook functions.

Feature_Add_OnRowCreated_AddHook

This hook function is called any time a row is created with the **Add** feature. This function will be called every time the row gets the **BindRow** function applied, which means if you scroll away from the row and then scroll back, it will be called. This is desirable because of the way batchlist handles row creation and destruction. For speed and efficiency, data intensive tasks should not be performed here.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.Feature_Add_OnRowCreated_AddHook( this.MyStatements_Add_OnRowCreated );
    ...
}

MyStatementsBatchList.prototype.MyStatements_Add_OnRowCreated = function( row )
{
    /* row columns can be accessed by row.column_type, where "type" is the code of
     the column. So, for our "name" column, we would access the root div via
     row.column_name; */

    var selectlist, name_select;

    if ( !row || !row.column_name )
    {
        return;
    }

    selectlist = row.column_name.getElementsByTagName( 'select' );

    if ( selectlist.length == 0 )
    {
        return;
         /* Our name column will never have a select in it, this is just to give
         you an idea of what you could do with it */
    }

    name_select = selectlist[ 0 ];
    name_select.onchange = function() { alert( 'woo! you changed for me!' ); }
}
```

Feature_Add_RowSupportsChildren_AddHook

This hook function is called before **MMBatchList** allows a row to add a child. If the **RowSupportsChildren** hook function returns false, the currently selected/active row will not be able to add a child record. This function is only used if sub records are enabled in your list.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.Feature_Add_RowSupportsChildren_AddHook(
     this.MyStatements_RowSupportsChildren );
```

```
    ...
}

    MyStatementsBatchList.prototype.MyStatements_RowSupportsChildren = function(
     item )
    {
        if ( item.branch === this.branch_child1 && item.record.type = 'nochid' )
        {
            return false; // Disallows child creation
        }

    return true; // Allows child creation
}
```

Feature_Buttons_EnableDisableButtons_AddHook

This hook function is called any time the **Feature_Buttons_EnableDisableButtons** hook function is called. For example, use this hook if you have custom buttons attached to the button bar that require special handling (show, hide, enable, disable).

Note: Miva recommends that dynamic buttons be hidden or shown. Persistent buttons should always be visible but enabled or disabled.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.button_dosomething = this.Feature_Buttons_AddButton_Dynamic_EditMode(
     'Action', 'Some Action', '', this.DoSomething );
    this.Feature_Buttons_EnableDisableButtons_AddHook(
     this.MyStatements_EnableDisableButtons );
    ...
}

MyStatementsBatchList.prototype.DoSomething = function()
{
    // Dummy placeholder function to show that DoSomething is a function
}

MyStatementsBatchList.prototype.MyStatements_EnableDisableButtons = function()
{
    if ( this.SingleItemSelected() )
    {
        this.button_dosomething.Show();
    }
```

```
    else
    {
        this.button_dosomething.Hide();
    }
}
```

Feature_Buttons_onSetSortList_AddHook

This hook function is called any time **onSearch** is called. For example, the first time you load the list, whenever the search value is changed, etc.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.Feature_Buttons_onSetSortList_AddHook(
     this.MyStatements_Feature_Buttons_onSetSortList_Hook );
    ...
}

MyStatementsBatchList.prototype.MyStatements_Feature_Buttons_onSetSortList_Hook =
 function( sortlist )
{
    /* Perform some action based on sortlist. In reality, you should never need to
     do anything here. */
}
```

Feature_Edit_OnRowCreated_AddHook

This hook function is called any time a row is created with the **Edit** feature.

Note: This function is called every time the **BindRow** function is applied to the row. If you scroll away from the the row and scroll back, the function will be called. This behavior is desirable because of the way batchlist handles row creation and destruction. But for speed and efficiency, data intensive tasks should not be performed here.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.Feature_Edit_OnRowCreated_AddHook( this.MyStatements_Edit_OnRowCreated );
    ...
}

MyStatementsBatchList.prototype.MyStatements_Edit_OnRowCreated = function( row )
```

```
{
    /* row columns can be accessed by row.column_type, where "type" is the code of
     the column. So, for our "name" column, we would access the root div via
     row.column_name; */

    var selectlist, name_select;

    if ( !row || !row.column_name )
    {
        return;
    }

    selectlist = row.column_name.getElementsByTagName( 'select' );

    if ( selectlist.length == 0 )
    {
        return;
         /* Our name column will never have a select in it, this is just to give
         you an idea of what you could do with it */
    }

    name_select = selectlist[ 0 ];
    name_select.onchange = function() { alert( 'woo! you changed for me!' ); }
}
```

Feature_Edit_RowSupportsEditing_AddHook

This hook function is called when a row attempts to enter inline edit mode. If the hook returns false, editing will be disabled for that element. This is a CONDITIONAL function and should be used if only certain rows can enter **Edit** mode.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.Feature_Edit_RowSupportsEditing_AddHook(
     this.MyStatements_Edit_RowSupportsEditing );
    ...
}

MyStatementsBatchList.prototype.MyStatements_Edit_RowSupportsEditing = function(
 item )
{
    if ( item.branch === this.branch_child1 )
    {
        return false; // Disallows child record editing
```

```
    }

    return true; // Allows parent record editing
}
```

OnBeforeUnloadVisibleRow_AddHook

This hook function is called just before the visible row moves off screen (from scrolling or when rebinding a row). If you want an action to take place on the row before this happens, add an **OnBeforeUnloadVisibleRow** hook.

WARNING: This hook function can create a lot of overhead if implemented improperly.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.OnBeforeUnloadVisibleRow_AddHook(
     this.MyStatements_OnBeforeUnloadVisibleRow );
    ...
}

MyStatementsBatchList.prototype.MyStatements_OnBeforeUnloadVisibleRow = function(
 row )
{
    var item;

    if ( ( item = this.GetListItem( row.index ) ) === null )
    {
        return;
    }

    /*
        Custom handler code here
    */
}
```

OnLoadSingleUse_AddHook

This hook function is called once after the next load of record data is returned. This is a burner function — after it is run a single time, the function is destroyed and never called again.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.OnLoadSingleUse_AddHook( this.MyStatements_OnLoadSingleUse_Hook );
    ...
}

MyStatementsBatchList.prototype.MyStatements_OnLoadSingleUse_Hook = function(
 start_index )
{
    /* Handler code here for something you want to happen when the next batch of
     records is loaded (example could be scrolling to one of the loaded records) */

    // Be sure that anything done here does not interfere with the user experience

    // Will be run a grand total of ONCE
}
```

OnResetList_AddHook

This hook function is called any time the list is reset (records cleared, etc.)

Example:

```
function MyStatementsBatchList()
{
    ...
    this.OnResetList_AddHook( this.MyStatements_OnResetList_Hook );
    ...
}

MyStatementsBatchList.prototype.MyStatements_OnResetList_Hook = function()
{
    // The list was reset

    /* Generally here, you would reset any defaults your list might have that are
     based on list records (ie, if you add your own feature that keeps track of the
     count of records that have been loaded, you would reset that count to 0 here)
     */
}
```

OnRowDataBound_AddHook

This hook function is called every time the data for a row is updated or redrawn. This can happen very frequently.

WARNING: The performance consequences can be drastic when using this hook. Be extremely careful with your implementation.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.OnRowDataBound_AddHook( this.MyStatements_OnRowDataBound_Hook );
    ...
}

MyStatementsBatchList.prototype.MyStatements_OnRowDataBound_Hook = function( item,
 row )
{
    // Please don't do this... it is just an example

    if ( item.record.active )   row.style.backgroundColor = '#f00';
    else                        row.style.backgroundColor = '';
}
```

OnSearch_GetFilter_AddHook

This hook function is called whenever **onSearch** is called. For example, when you first load the list, any time the search value is changed, etc.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.OnSearch_GetFilter_AddHook( this.MyStatements_OnSearch_GetFilter_Hook );
    ...
}

MyStatementsBatchList.prototype.MyStatements_OnSearch_GetFilter_Hook = function()
{
    // MUST RETURN ARRAY
    var code, filterlist, column_codes;

    filterlist = new Array();
    filterlist.push( 'custom_filter:' + encodeURIComponent( 'my_value' ) );

    return filterlist;
}
```

RecordShouldUpdate_AddHook

This hook function is called just before attempting to update the record (used in the **Edit** feature and in the **AssignList** to when updating the assigned/unassigned value). A false from this hook causes the record to ***not*** be updated. Use this hook if only records meeting a certain condition should be updated.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.RecordShouldUpdate_AddHook( this.MyStatements_RecordShouldUpdate );
    ...
}

MyStatementsBatchList.prototype.MyStatements_RecordShouldUpdate = function( item )
{
    if ( item && item.branch == this.branch_child1 )
    {
    return false; // Returning false prevents update
    }

    return true;
}
```

Resize_AddHook

This hook function is called any time we trigger a **Resize** on the list — for example, when the window is resized.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.Resize_AddHook( this.MyStatements_OnResizeHook );
    ...
}

MyStatementsBatchList.prototype.MyStatements_OnResizeHook = function()
{
    /*
        Custom handler code here
    */
}
```

RowShouldBecomeActive_AddHook

This hook function is called just before a record item is added to the **ActiveItemList**. This allows you to disallow certain rows meeting specific conditions from being selectable.

Note: If you manually add an item to the **ActiveItemList**, you will bypass this **RowShouldBecomeActive** call.

Example:

```
function MyStatementsBatchList()
{
    ...
    this.RowShouldBecomeActive_AddHook( this.MyStatements_RowShouldBecomeActive );
    ...
}

MyStatementsBatchList.prototype.MyStatements_RowShouldBecomeActive = function(
 item )
{
    if ( item && item.branch == this.branch_child1 )
    {
        return false;
         // Returning false prevents row from being set to active state
    }

    return true;
}
```

Overridable Functions

The following list of **MMBatchList** functions are meant to be overridden. Most of these functions are only used when certain features are enabled. For a detailed discussion, read this entire document to see when these functions ***should*** be overridden.

- **onCreate(item, callback, delegator)**
- **onCreateRootColumnList()**
- **onDelete(item, callback, delegator)**
- **onDeleteConfirmationMessage()**
- **onDisplayOrderGetRecordID(record)**
- **onDisplayOrderReset()**
- **onDisplayOrderReset_Check()**
- **onDisplayOrderSave(parameters, callback)**
- **onEdit(item)**
- **onFindIndex_Compare(item, params)**

- **onFindIndex_Params(item)**
- **onGetRecordID(record)** – This function should be overridden by any list that does not use 'id' as the unique identifier for the record. For example, if your database table or dataset does not have an **id** column but instead has a **unique_id** column or **code** that is guaranteed to be unique, you must override this function and return that value instead.
- **onGoTo(item)**
- **onInsert()**
- **onLoad(filter, sort, offset, count, callback, delegator)**
- **onLoadRecordIndex(record, callback)**
- **onProcessLoadedData(recordlist, start_index)**
- **onRetrieveChildBranch(item)**
- **onRowDoubleClick(item)**
- **onSave(item, callback, delegator)**
- **onSaveAssigned(item, callback, delegator)**
- **onSetDisplayOrder(recordlist, start_index)**

Load Query Return Data Format

While the **MMBatchList** is very versatile to the needs of the list creator, the **MMBatchList** class does expect a structured JSON response.

In the examples on the following pages, **total_count** represents the total count of all returnable records.

Important: Included in the count is the count of ***all*** sub records. If you have a single parent with 3 children, the **total_count** would be 4 (**parent + child + child + child**). The **start_offset** is the start position of the returned data. By default, **MMBatchList** calculates the number of records to return based on the number of displayable records, meaning the second time it went to load data, the **start_offset** would be **count + 1**.

Example: Normal batch list (no sub records)

```
{
    "success": 1,
    "data": {
        "total_count": 1000,
        "start_offset": 0,
        "data": [
        {
            "name1": "value1",
            "name2": "value2"
        },
        {
            "name1": "value1",
```

```
            "name2": "value2"
        },
        {
            "name1": "value1",
            "name2": "value2"
        },
        ...
        ]
    }
}
```

Example: Sub record batch list

```
{
    "success": 1,
    "data": {
        "total_count": 1500000,
        "start_offset": 0,
        "data": [
            {
                "name1": "value1",
                "name2": "value2",
                "child":
                [
                    {
                            "name1": "value1",
                            "name2": "value2"
                    },
                    {
                            "name1": "value1",
                            "name2": "value2"
                    }
                ]
            },
            {
                "name1": "value1",
                "name2": "value2"
                "child":
                [
                    {
                            "name1": "value1",
                            "name2": "value2"
                    },
                    {
```

```
                                    "name1": "value1",
                                    "name2": "value2"
                                },
                                ...
                            ]
                        },
                        ...
                    ]
                }
            }
```