MICRO FOCUS | SERENA

Solutions Business Manager
SBM JavaScript Library Guide

Part number: Product version: 11.7

Publication date: 2019-11-01

Table of Contents

Overview

The SBM JavaScript library is automatically included with every custom form. You do not need to do anything to access this library.

JavaScript can be added to a custom form in two ways:

- Importing or editing a JavaScript file, adding it to the process app, and then adding it to a custom form. The advantage of using this method is that you can reuse the file in several forms.
- Adding an HTML/JavaScript widget with the content of `<script>[your content]</script>` to a custom form. The advantage of using this method is that you can quickly edit and test the widget on a custom form, and later move its content to a JavaScript file to reuse on other custom forms.

After you create a JavaScript file, you can import it by adding it to your custom form from the **JavaScripts** tab in the Property Editor for a form in SBM Composer.

Tip: You can define form actions in SBM Composer in an intuitive, condition-driven interface that allows you to include dynamic behavior in custom forms without JavaScript programming. For more information, see the dynamic forms information in the *SBM Composer Guide*.

CAUTION:

The SBM JavaScript library and form actions are incompatible with the **Use Accessible Interface** User Profile display option in SBM Work Center.

Tips for Working with Fields

- Field names can be either the **Name** or **Database field name** specified in the Property Editor for the field. They can also be the name of a button or widget or other non-field control. The names must match the names in the Property Editor (for example, a **Database field name** must must be uppercase and cannot have spaces).
- When working in an environment where the field display names change often, it is best to use the **Database field name**.
- Forms must have fields visible and editable before you can use methods such as `ShowField`, `HideField`, `EnableField`, and `DisableField`. The JavaScript does not override what is set at the field or form level; it filters the form as it is.

Reference

The methods described in this section are divided into the following categories:

- Event Methods [page 6]
- Field Methods [page 14]

Event Methods

AddLoadCallback

Adds a callback that is invoked after the page loads.

Parameters

Name	Type	Description
`callback`	Function	The function to be invoked when the page loads.

Return Value

Result	Value
(none)	

Example

This example registers a named function to be called when the page loads:

```
function customLoadCallback()
{
    MakeFieldDisabled( "Priority" );
}

AddLoadCallback( customLoadCallback);
```

Comments

Callbacks are invoked in the order in which they are added. This delays scripting until all of the page objects load. If this method is called after the page loads, it has no effect. This method is often used to set the initial state of fields and controls on a form.

AddClickCallback

Used to respond to click events from **Button**, **Image**, or **HyperLink** controls.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field to find.
`callback`	Function	The function to be invoked when the control is clicked.

Return Value

Result	Value
(none)	

Example

This example registers a named function to be called on click:

```
function customClickCallback()
{
    RefreshWidget( "DataGrid" );
}

AddClickCallback("Refresh", customClickCallback);
```

Comments

Callbacks are invoked in the order in which they are added.

AddChangeCallback

Used to respond to changes in fields and controls, with the restrictions noted under **Comments**, below.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field to find.
`callback`	Function	The function to be invoked when the field or control changes.

Return Value

Result	Value

(none)	

Example

This example registers a named function to be called on change:

```
function customChangeCallback()
{
    var itemType = GetFieldValue( "Item Type" );
    if ( itemType == "Defect" )
    {
         EnableField( "Steps To Reproduce " );
         DisableField( "Requested Changes" );
    }
    else
    {
         DisableField( "Steps To Reproduce" );
         EnableField( "Requested Changes" );
    }
}
AddChangeCallback("Item Type", customChangeCallback );
```

Comments

Callbacks are invoked in the order in which they are added. Note the following restrictions:

Field/ Control Type	Field Style	Notes
Binary/ Trinary	Radio buttons	Use AddRadioCallback [page 9] instead.
Single Selection, Single Relational, User	Allow searching	Not supported.
Multi-Selection, Multi-Relational, Multi-User, Multi-Group	Allow searching	Before this method can be invoked to populate a searchable *Multi-Group*, *Multi-Relational*, *Multi-Selection*, or *Multi-User* field, values must be present in the left-hand box on the **ListBox** control.

Field/ Control Type	Field Style	Notes
Multi-Selection, Multi-Relational, Multi-User, Multi-Group		This method is not invoked for these field types when a user simply double-clicks a value or uses the right-hand arrow to move a value from the left-hand box on a **ListBox** control to the right-hand box. For the method to be invoked, the user must also click the values in the right-hand box to select them.
Text field, EditBox control		This method is invoked only after the user changes the value and moves focus out of the field or control.

AddRadioCallback

Used to respond to changes in *Binary* fields with the "Radio buttons" style. All other fields and controls should use AddChangeCallback [page 7] instead.

Note: AddChangeCallback [page 7] also works for *Binary* fields with the "Radio buttons" style, because these calls are automatically redirected to AddRadioCallback.

Parameters

Name	Type	Description
`objname`	String	The name of the field to find.
`callback`	Function	The function to be invoked when the radio button is clicked.

Return Value

Result	Value
(none)	

Example

This example registers a named function to be called on click:

```
function customRadioCallback()
{
    RefreshWidget( "DataGrid" );
}

AddRadioCallback("Refresh", customRadioCallback);
```

Comments

Callbacks are invoked in the order in which they are added.

AddSubmitCallback

Adds a callback that is invoked before the page is submitted.

Parameters

Name	Type	Description
`callback`	Function	The function to be invoked before the page is submitted.

Return Value

Result	Value
true	Continue processing other submit callbacks. If a function does not return a value, it is assumed to be "true".
false	Stop processing other submit callbacks and cancel the submit operation.

Example

This example registers a named function to be called before the page is submitted:

```
function customSubmitCallback()
{
    // Prevent submit if Description field is empty
    if ( IsFieldEmpty( "Description" ) )
    {
        ShowErrorDiv( true, "A description is required" );
        return false;
    }
    else
    {
        ShowErrorDiv( false );
        return true;
    }
}

AddSubmitCallback( customSubmitCallback );
```

Comments

Callbacks are invoked in the order in which they are added. This callback can be used to perform validation or other data verification. If you want to display a field validation error, call MakeFieldInvalid [page 15] and return "true" from this callback so that any required or invalid field errors are automatically processed.

AddFocusCallback

Adds a callback that is invoked when a field or control is focused or blurred.

Parameters

Name	Type	Description
`objname`	String	The name of the field or control to find.
`callback`	Function	The function to be invoked when the field or control is focused or blurred.
`focus`	Boolean	Whether to invoke the callback when the field or control is focused or blurred.

Return Value

Result	Value
(none)	

Example

This example registers a named function to be called when a field is focused or blurred:

```
function customFocusCallback()
{
    if ( IsFieldEmpty( "Description" ) )
    {
        ShowErrorDiv( true, "A description is required" );
    }
    else
    {
        ShowErrorDiv( false );
    }
}
AddFocusCallback( "Description", customFocusCallback, false );
```

Comments

Callbacks are invoked in the order in which they are added. This callback can be used to perform validation or other data verification. If you want to display a field validation error, call MakeFieldInvalid [page 15] from this callback so that any required or invalid field errors are automatically processed.

This method can be used with all field types and all input controls. For *Multi-Selection*, *Multi-Relational*, *Multi-User*, and *Multi-Group* fields, with the "Allow searching" field style, the focus or blur must occur on the right-hand side of the list box.

AddSectionCallback

Adds a callback that is invoked when a section is expanded or collapsed.

Parameters

Name	Type	Description
objname	String	The name of the section.
callback	Function	The function to be invoked when the section is expanded or collapsed.
expanded	Boolean	Whether to invoke the callback when the section is expanded or collapsed. If true, the callback is invoked when the user expands a section.

Return Value

Result	Value
(none)	

Example

This example registers a named function to be called when a section is expanded or collapsed:

```
function customSectionCallback()
{
    if ( IsFieldEmpty( "Problem" ) )
    {
        ShowErrorDiv( true, "A problem entry is required" );
    }
    else
    {
        ShowErrorDiv( false );
    }
}
AddSectionCallback( "ProblemsSection", customSectionCallback, false );
```

Comments

Callbacks are invoked in the order in which they are added. This callback can be used to perform validation or other data verification. If you want to display a field validation error, call MakeFieldInvalid [page 15] from this callback so that any required or invalid field errors are automatically processed.

AddTabCallback

Adds a callback that is invoked when a tab is activated or deactivated.

Note: This method applies to individual tabs, not the tab container control that holds the tabs.

Parameters

Name	Type	Description
`objname`	String	The name of the tab.
`callback`	Function	The function to be invoked when the tab is activated or deactivated.
`activated`	Boolean	Whether to invoke the callback when the tab is activated or deactivated.

Return Value

Result	Value
(none)	

Example

This example registers a named function to be called when a tab is activated or deactivated:

```
function customTabCallback()
{
    if ( IsFieldEmpty( "Problem" ) )
    {
         ShowErrorDiv( true, "A problem entry is required" );
    }
    else
    {
         ShowErrorDiv( false );
    }
}
AddTabCallback( "ProblemsSection", customTabCallback, false );
```

Comments

Callbacks are invoked in the order in which they are added. This callback can be used to perform validation or other data verification. If you want to display a field validation error, call MakeFieldInvalid [page 15] from this callback so that any required or invalid field errors are automatically processed.

GetFormActionsEventSource

Returns the source element for an event that is currently being handled and that was defined in a form action.

Parameters

None

Return Value

Result	Value
Source element	The element that was the source of the current event.

Example

This example turns the *Title* field red when it gains focus, and removes the color when it loses focus. (One of the events defined in the form action is "When Title field gains focus, then execute "Colorize();".)

```
function Colorize()
{
    var src = GetFormActionsEventSource();
    if (!src) return;

    src.style.backgroundColor = "#FF0000";
}

function Uncolorize()
{
    var src = GetFormActionsEventSource();
    if (!src) return;

    src.style.backgroundColor = "#FFFFFF";
}
```

Comments

The source element that is returned is only valid during the current event, so this method should be used within the context of a callback method.

Field Methods

MakeFieldInvalid

Marks the specified field as invalid, if it is not already marked as such. It does not update the form and the label. It displays an error on submit that prevents the form from being submitted.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.

`msg`	String	The message to display for this invalid field. The message is displayed when the user clicks **OK** in a transition form.

Return Value

Result	Value
(none)	

Examples

This example marks the *Unit Price* field as invalid:

```
MakeFieldInvalid("Unit Price", "The value in the 'Unit Price' field "
  + "must be greater than $5 and less than $500.")
```

Comments

(none)

MakeFieldValid

Marks the specified field as valid (that is, not invalid), if it is not already marked as such. It does not update the field or the label.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.

Return Value

Result	Value
(none)	

Examples

This example marks the Unit Price field as valid:

```
MakeFieldValid("Unit Price");
```

Comments

(none)

IsFieldEmpty

This method returns true or false, depending on whether the field contains any data.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.

Return Value

Result	Value
true	Returns `true` if the field does not contain any data.
false	Returns `false` if the field contains data.

Example

This example returns `true` if the *Description* field does not contain any text, or `false` if it does.

```
IsFieldEmpty("Description");
```

Comments

(none)

IsFieldChecked

This method should only be used on *Binary* fields with the "Check box" style. It returns true or false, depending on whether the check box for the given field is checked.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.

Return Value

Result	Value
true	Returns `true` if the check box is selected.
false	Returns `false` if the check box is cleared or if the field is not found.

Example

This example returns `true` if the check box for the *Regression* field is selected, or `false` if the check box is cleared.

```
IsFieldChecked("Regression");
```

Comments

(none)

GetFieldValue

Gets the value of the specified field.

Note: `GetFieldValue` is also a scripting method in SBM.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.
`defaultValue` (optional)	String	The default value to return if the field does not exist on the form, if it is not found, or if its value is empty. If this parameter is not provided, "null" is returned.
`path` (optional)	String	The name of the sub-field to return. On some field types, a path of "ID" will return a unique internal identification number.

Return Value

Field Type	Field Style	Return Value
Binary/Trinary	Drop down list, Radio buttons	String value of the field.
Binary/Trinary	Check box	Numeric value 1 if checked; empty value if cleared. (It is recommended that you use IsFieldChecked [page 17] instead for this field type and style.)

Date/Time	Date and time, Date only, Time of day	A Date object containing the value of the field.
Date/Time	Elapsed time	Numeric value for the number of milliseconds of elapsed time.
Folder	N/A	String value of the field
Multi-Group, Multi-User, Multi-Selection, Multi-Relational	N/A	String value of the field. If multiple values are selected, a comma-separated list is returned. If the values have embedded commas, use GetFieldValues [page 23] instead to retrieve the array.
Grid	Embedded Report, Relational Grid	String value of the field. If multiple values are selected, a comma-separated list is returned. A path parameter is required to specify the column to retrieve values from.
Numeric	N/A	Numeric value of the field.
User, Single Selection, Single Relational	N/A	String value of the field.
Sub-Relational	N/A	Depends on the referenced field.
Summation	N/A	Numeric value of the field.
Text	N/A	String value of the field. For journal fields, the return value depends on what you specify in the path (see below). If the path is not included, the current editable contents (or an empty string if the field is read-only) are returned.

For journal fields, you can include the following tokens in the path (in any order):

- `""` (a blank or non-existent path) – Returns an object that contains all properties of the most recent entry on a state form, or the current editable value on a transition.
- `date` – Returns the date of the most recent entry (returned as a JavaScript Date object).
- `user` – Returns the author of the most recent entry.

- `userid` – Returns the user ID for the author of the most recent entry.
- `contents` – Returns the plain or rich text value of the most recent entry. Contents are either un-escaped plaintext or HTML with tags (as determined by the **Enable Rich Text** option on the journal field).
- `[]` – Returns either:
  - An array of complete entry objects when by itself (for example, `[]`)
  - An array of whatever type is specified by the property token when used in combination with `[]`, starting with the oldest entry (for example, `contents[]`)

  If the path does not contain `[]`, only the most recent entry is returned. Also, the current editable entry is not returned (if any exists) in any case.
- `DESC` – Token that can be used in combination with `[]` to sort from most recent to oldest. For example, `[]DESC` returns an array of all entries, sorted from most recent to oldest.

Including any one of the `date`, `user`, `userid`, or `contents` properties in the path returns only that specific part of the output value. If none of these are in the path, the output will be an object (or an array of objects) that contains all four properties. The path does not accept more than one property token; therefore, if you need more than one property returned, do not specify any of them.

Examples

This example returns the string value of the *Unit Price* field:

```
GetFieldValue("Unit Price");
```

This example returns the string value of a multi-relational field, or "none" if there is nothing to return:

```
GetFieldValue("MyMultiRelational", "none");
```

This example returns a comma separated list of IDs corresponding to selected rows on an embedded report, or an empty string if there is nothing to return:

```
GetFieldValue("MyEmbeddedReport", "", "IDs");
```

This example returns an array of string values corresponding to the Title sub-field of an embedded report's selected rows, or "No items selected" if there is nothing to return:

```
GetFieldValue("MyEmbeddedReport", "No items selected", "Title[]");
```

This example returns an array of all the text entries from a journal field.

```
GetFieldValue("MyJournalField", [], "contents[]");
```

This example function could be placed in a form's included JavaScript files and would return the number of existing entries in a journal field.

```
function numJournalEntries(objname) {
  if (LookupFieldType(objname) != "text" || LookupFieldSubType(objname) != "journal") return 0;
  var contents = GetFieldValue(objname, [], "[]");
  return contents.length;
}
```

Comments

(none)

SetFieldValue

Sets the value of the specified field.

Note: *Sub-Relational* fields are not supported. These fields are driven by associated *Single Relational* or *Multi-Relational* fields, which can be set using the `SetFieldValue` method.

Note: `SetFieldValue` is also a scripting method in SBM.

Note: If you are using translations for *Selection* fields, you must include all translations for `SetFieldValue` to work. If the number of translations is too large to maintain, it is recommended to use form actions instead of `SetFieldValue`.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.

`value`	Various	The value to set on the field. The value is a string, with the following exceptions: • *Binary/Trinary* (Check box style): The value is a Boolean (true or false). • *Date/Time* (Date and time, Date only, Time of day style): The value is a Date object. • *Date/Time* (Elapsed time style): The value is a numeric value for the number of milliseconds of elapsed time. If the field is a Combo Box or List Box, the value must match the value of an item already in the list, and is case-sensitive. If the field is a *Multi-Group*, *Multi-Relational*, *Multi-Selection*, or *Multi-User* field, the value is a string of comma-separated values. If the values have embedded commas, use the SetFieldValues [page 24] method instead to return the array. In Modern Forms, the following input types are also accepted for *Single-* and *Multi- Group*, *Relational*, *Selection*, and *User* fields: • Single string. If the field is searchable and the value is not present, a search will be performed using this string. The search will attempt to set the desired value from the results. If no exact match is found, the default value from that search will be set. • Numeric database ID. If the field is searchable and the value is not present, a search will be performed using no keywords. The search will attempt to set the desired value from the results. • Name/value pair object in the form { name: "*value*", id: *id*}. If the value is not present, it will be inserted and selected without running a search. Arbitrary values may be inserted this way but the server will still perform validation of the values and dependencies. • An array of any of the above. If the field is searchable and any values are not present, a search will be performed using no keywords.  **Note:** Automatic searches performed by `SetFieldValue` may experience limitations with max search results or complex VDF. When possible, try to use name/value pairs when setting a searchable field, as any desired values that are already present or that can be constructed will be selected without searching.

`fireEvent`	Boolean	If this parameter is set to true, then setting the values causes the change event to fire. If this parameter is set to false, then the change event does not fire.

Return Value

Result	Value
(none)	

Examples

This example sets the string value of a *Text* field:

```
SetFieldValue("UNITPRICE", "1.45", true);
```

This example sets the string value of a *Binary/Trinary* field with three radio button choices:

```
SetFieldValue("RELEASETYPE", "Beta", true);
```

This Modern Form example sets the value of a searchable *Single Relational* field (e.g. copied from another field referencing the same table). The value is inserted if necessary:

```
SetFieldValue("RELEASE", { name: "SBM 11.3", value: 12345 }, true);
```

Comments

(none)

GetFieldValues

Gets an array of values from the specified field.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.
`defaultValue` (optional)	String	The default value or array of values to return if the field does not exist on the form, if it is not found, or if its value is empty. If this parameter is not provided, "null" is returned.  **Note:** If the `defaultValue` is a single string value, it is converted to a single-element array.

Return Value

Result	Value
Success	An array of strings, dates, or numbers matching the field type.
Failure	Null or the default value

Examples

This example returns the array value of the field:

```
GetFieldValues("Reviewers");
```

This example returns a default array of values:

```
GetFieldValues("Reviewers", ["Tom Smith", "Susan Jones"]);
```

Comments

`GetFieldValues` is typically used with *Multi-Group*, *Multi-Relational*, *Multi-Selection*, and *Multi-User* fields. If this method is used with other field types, it returns an array with a single value.

SetFieldValues

Sets an array of values for the specified field.

Note: *Sub-Relational* fields are not supported. These fields are driven by associated *Single Relational* or *Multi-Relational* fields, which can be set using this method.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.
`value`	Array	The array of values to set on the field. If the field is a **Combo Box** or **List Box**, the value must match the value of an item already in the list, and is case-sensitive.

Return Value

Result	Value
(none)	

Examples

This example sets the array of string values for the field:

```
SetFieldValues("Numbers", ["one", "two", "three", "four", "five"]);
```

Comments

`SetFieldValues` is typically used with *Multi-Group*, *Multi-Relational*, *Multi-Selection*, and *Multi-User* fields. If this method is used with other field types, it sets the field to the first value in the array.

Before this method can be invoked to populate a searchable *Multi-Group*, *Multi-Relational*, *Multi-Selection*, or *Multi-User* field, values must be present in the left-hand box on the **ListBox** control.

GetMultiListValues

Gets an array of values from the specified **List Box** control or from a **Relational Grid** widget.

Note: This method supports only *Multi-Group*, *Multi-Relational*, *Multi-Selection*, and *Multi-User* fields.

Parameters

Name	Type	Description
`objName`	String	The name of the **List Box** control or widget.
`defaultValue` (optional)	String	The default value or array of values to return if the user did not select at least one list box value or row, or if the control does not exist on the form. If this parameter is not provided, "null "is returned.  **Note:** If the `defaultValue` is a single string value, it is converted to a single-element array.
`asValue` (optional)	String	If this parameter is set to "true," then the actual array of values as set in the HTML element are returned instead of display values. For widgets, "true" returns an array of objects, each of which is a row. If this parameter is set to "false" or is not used, then an array of display values is returned. In the `<option...value="NEW_YORK">New York</option>` example, New York is the display value, and NEW_YORK is the actual value.

Return Value

Result	Value
Success	An array of strings representing the display value or actual value of each selected item in the **List Box** control. For widgets, an array of objects representing selected rows is returned.
Failure	Null or the default value.

Examples

This example returns an array of actual values based on items the user selected in the **Sites** list box.

```
GetMultiListValues("Sites", "New York", true);
```

Comments

(none)

SetMultiListValues

Sets an array of values in the specified **List Box** control.

Note: This method supports only *Multi-Group*, *Multi-Relational*, *Multi-Selection*, and *Multi-User* fields.

Parameters

Name	Type	Description
`objName`	String	The name of the **List Box** control.
`values`	Array	An array of values to set in the **List Box** control. You can specify display values or internal values. (TS_ID represents the internal value in the database.)
`fireEvent`	String	If this parameter is set to "true," then setting the values causes the change event to fire. If this parameter is set to "false," then the change event does not fire.
`asValue` (optional)	String	If this parameter is set to "true," then the actual array of values as set in the HTML element are returned instead of display values. If this parameter is set to "false" or is not used, then an array of display values is returned. In the `<option...value="NEW_YORK">New York</option>` example, New York is the display value, and NEW_YORK is the actual value.

Return Value

Result	Value
(none)	

Examples

This example sets an array of display values based on the **Sites** list box.

```
SetMultiListValues("Sites", ["New York", "San Francisco", "Chicago",
→"Los Angeles", "Seattle"], true, false);
```

This example sets an array of internal values based on the **Developer** list box.

```
SetMultiListValues("Developers", [213, 214, 215], true, true);
```

Comments

Before this method can be invoked to populate a searchable *Multi-Group*, *Multi-Relational*, *Multi-Selection*, or *Multi-User* field, values must be present in the left-hand box on the **ListBox** control.

MakeFieldRequired

Marks the specified field as required, if it is not already marked as such. It updates the field and the label appropriately.

Note: This method can be used only on a field that is set in SBM Composer or SBM Application Administrator to be not required, not read-only, and in a field section that is visible on a form. It is used with the `MakeFieldOptional` method to dynamically change the required or optional state of a non-required field on a form.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.
`bShowIfHidden` (optional)	Boolean	Show the field if it was hidden by the `HideField` method. The default value is true.

Return Value

Result	Value
(none)	

Examples

This example marks the *Unit Price* field as required:

```
MakeFieldRequired("Unit Price");
```

This example marks the *Unit Price* field as required but does not show it if it is hidden:

```
MakeFieldRequired("Unit Price", false);
```

Comments

(none)

MakeFieldOptional

Marks the specified field as optional (that is, not required), if it is not already marked as such. It updates the field and the label appropriately.

Note: This method can be used only on a field that is set in SBM Composer or SBM Application Administrator to be not required, not read-only, and in a field section that is visible on the form. It is used with the `MakeFieldRequired` method to dynamically change the optional or required state of a non-required field on a form. This method does not apply to fields originally set as required on a workflow or transition level. It applies only to fields set as required with the `MakeFieldRequired` method.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.
`bShowIfHidden` (optional)	Boolean	Show the field if it is currently hidden. The default value is true.

Return Value

Result	Value
(none)	

Examples

This example marks the *Unit Price* field as optional:

```
MakeFieldOptional("Unit Price");
```

This example marks the *Unit Price* field as optional but does not show it if it is hidden:

```
MakeFieldOptional("Unit Price", false);
```

Comments

(none)

DisableField

Marks the specified field as disabled/read-only, if it is not already marked as such. It updates the field only, not the label.

Note: This method is used with the `EnableField` method to dynamically change the disabled or enabled state of a non-required field on a form. This method does not apply to fields originally set as enabled on a workflow or transition level. It applies only to fields set as enabled with the `EnableField` method.

Important: Values for disabled fields that use check boxes are not stored in the database.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.
`bShowIfHidden` (optional)	Boolean	Show the field and the label if they are currently hidden. The default value is true.

Return Value

Result	Value
(none)	

Examples

This example marks the *Unit Price* field as disabled/read-only:

```
DisableField("Unit Price");
```

Comments

(none)

EnableField

Marks the specified field as enabled (that is, not disabled), if it is not already marked as such. It updates the field only, not the label.

Note: This method is used with the `DisableField` method to dynamically change the disabled or enabled state of a non-required field on a form. This method does not apply to fields originally set as disabled on a workflow or transition level. It applies only to fields set as disabled with the `DisableField` method.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.
`bShowIfHidden` (optional)	Boolean	Show the field and the label if they are currently hidden. The default value is true.

Return Value

Result	Value
(none)	

Examples

This example marks the *Unit Price* field as enabled:

```
EnableField("Unit Price");
```

Comments

(none)

HideField

Marks the specified field as hidden, if it is not already marked as such. It updates the field and the label appropriately.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.
`bHideFieldOnly` (optional)	Boolean	Hide the field only. The label remains visible. The default value is false.

Return Value

Result	Value
(none)	

Examples

This example marks the *Unit Price* field and its label as hidden:

```
HideField("Unit Price");
```

This example marks the *Unit Price* field as hidden, but the label remains visible:

```
HideField("Unit Price", true);
```

Comments

(none)

ShowField

Marks the specified field as visible, if it is not already marked as such. It updates the field and the label appropriately.

Note: This method does not apply to hidden fields (fields that belong in the Hidden section of a form).

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.
`bShowFieldOnly` (optional)	Boolean	Show the field only. The label remains hidden. The default value is false.

Return Value

Result	Value
(none)	

Examples

This example marks the *Unit Price* field and its label as visible:

```
ShowField("Unit Price");
```

This example marks the *Unit Price* field as visible; however, the field's label remains hidden.

```
ShowField("Unit Price", true);
```

Comments

(none)

HideSection

Marks the specified section or tab as hidden, if it is not already marked as such.

Parameters

Name	Type	Description
`sectionName`	String	The name of the section or tab.

Return Value

Result	Value
(none)	

Examples

This example marks the **Notes** section as hidden:

```
HideSection("Notes");
```

Comments

This currently works only with sections and tabs.

ShowSection

Marks the specified section or tab as visible, if it is not already marked as such.

Parameters

Name	Type	Description
`sectionName`	String	The name of the section or tab.

Return Value

Result	Value
(none)	

Examples

This example marks the **Notes** section as visible:

```
ShowSection("Notes");
```

Comments

This currently works only with sections and tabs.

IsSectionShown

Returns whether the given section is shown or hidden.

Parameters

Name	Type	Description
`sectionName`	String	The name of the section.

Return Value

Result	Value
true	The section is shown.
false	The section is hidden.

Examples

This example returns `true` if the **Notes** section is shown, or `false` if it is hidden:

```
IsSectionShown("Notes");
```

Comments

(none)

ExpandSection

Expands the specified expander section, if it is not already expanded.

Parameters

Name	Type	Description
`sectionName`	String	The name of the expander section.

Return Value

Result	Value
(none)	

Examples

This example expands the **Notes** section:

```
ExpandSection("Notes");
```

Comments

This currently works only with expander sections.

CollapseSection

Collapses the specified expander section, if it is not already collapsed.

Parameters

Name	Type	Description
sectionName	String	The name of the expander section.

Return Value

Result	Value
(none)	

Examples

This example collapses the **Notes** section:

```
CollapseSection("Notes");
```

Comments

This currently works only with expander sections.

IsSectionExpanded

Returns whether the expnder section is expanded or collapsed.

Parameters

Name	Type	Description
sectionName	String	The name of the expander section.

Return Value

Result	Value

true	The section is expanded.
false	The section is collapsed or not found.

Examples

This example returns `true` if the **Notes** section is expanded or `false` if it is collapsed:

```
IsSectionExpanded("Notes");
```

Comments

This currently works only with expander sections.

ActivateTab

Activates the given tab.

Note: This method applies to individual tabs, not the tab container control that holds the tabs.

Parameters

Name	Type	Description
`objName`	String	The name of the tab.

Return Value

Result	Value
(none)	

Examples

This example activates the **Development** tab:

```
ActivateTab("Development");
```

Comments

This currently works only with tabs (not tab controls).

IsTabActivated

Returns whether the given tab is activated or deactivated.

Parameters

Name	Type	Description
`objName`	String	The name of the tab.

Return Value

Result	Value
true	The tab is activated.
false	The tab is deactivated or not found.

Examples

This example returns `true` if the **Development** tab is activated, or `false` if it is deactivated:

```
IsTabActivated("Development");
```

Comments

This currently works only with tabs (not tab controls).

RefreshWidget

Refreshes the specified widget. This causes the widget to be refreshed or reloaded. New values are obtained and passed to the widget to display the results.

Parameters

Name	Type	Description
`widgetName`	String	The name of the widget.

Return Value

Result	Value
(none)	

Examples

This example refreshes the widget named RSS:

```
RefreshWidget("RSS");
```

Comments

(none)

SetLabelText

Sets the current text of the specified label.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.
`Text`	String	The text to set on the label. This method automatically adds a colon (`:`) and an asterisk `*` (for "required") as appropriate.

Return Value

Result	Value
(none)	

Examples

This example sets the text of the label:

```
SetLabelText("Unit Price", "Price/Unit");
```

Comments

(none)

GetLabelText

Gets the current text of the specified label.

Parameters

Name	Type	Description
`fieldName`	String	The name of the field.

Return Value

Result	Value
Success	The original value for the label on the form, or the value last set with the `SetLabelText` method.

Failure	Null

Examples

This example gets the text of the label:

```
GetLabelText("Unit Price");
```

Comments

(none)

Advanced Functions

Querying REST Service Results

The `SecureRESTServiceWrapper` and `RESTServiceWrapper` objects can be used to query results from a REST service without the need to display the REST Grid widget on a custom form.

Both JSON data and XML (POX) data are accepted. For JSON data, the results of the REST service are evaluated as a JavaScript object.

Note: See the *SBM Composer Guide* for widget information.

SecureRESTServiceWrapper

`SecureRESTServiceWrapper` is written to always use proxy ID to ensure that the REST service call is secure. This object defines the REST service to query through parameters as described in the following table.

Name	Type	Description
`url`	String	*Required.* Base URL of the service to query
`ID`	String	Custom endpoint ID
`applicationId`	String	Application ID

jsonResponse	Boolean	This controls whether JSON responses are evaluated in the JavaScript using the JavaScript Eval() method. If this parameter is not supplied, it is considered to be true.

For more information and an example using the SecureRESTServiceWrapper object, refer to the Community website.

For information on the functions, refer to the following sections:

get [page 39]

post [page 40]

RESTServiceWrapper

RESTServiceWrapper can be written to use proxy ID or to use basic authentication. This object defines the REST service to query through parameters as described in the following table.

Name	Type	Description
url	String	*Required.* Base URL of the service to query
username	String	User name to access the REST service, if authentication is required
password	String	Password to access the REST service, if authentication is required
useProxy	Boolean	This controls whether to access the REST service through the SBM proxy. The default value is false.
jsonResponse	Boolean	This controls whether JSON responses are evaluated in the JavaScript using the JavaScript Eval() method. If this parameter is not supplied, it is considered to be true.

For an examples of using the RESTServiceWrapper object, refer to the Community website and solution S138336 in the Support Knowledgebase.

For information on the functions, refer to the following sections.

get

This function returns data from a REST service. Its parameters are described in the following table.

Name	Type	Description
path	String	The service path on which to do a GET
queryString	String	The query string with which to do a GET

`callback`	Function	This is the JavaScript function to invoke when the REST data is returned. The first parameter will be the result data if `success` is true; otherwise, it will be the failure message. The second parameter is the user-defined `tag` as passed to the `get` call.
`tag`	*Depends on parameter*	A user-defined parameter to be passed to the callback along with the REST data
`customHeaders`	Array	An array of two-element arrays containing the name and value of each custom header to pass to the callback

post

This function posts data returned from a REST service to the custom form. Its parameters are described in the following table.

Name	Type	Description
`path`	String	The service path on which to do a POST
`queryString`	String	The query string on which to do a POST
`callback`	Function	The JavaScript function to invoke when the REST data is posted. The first parameter will be the result data if `success` is true; otherwise, it will be the failure message. The second parameter is the user-defined `tag` as passed to the `post` call.
`tag`	*Depends on parameter*	A user-defined parameter to be passed to the callback along with the REST data
`customHeaders`	Array	An array of two-element arrays containing the name and value of each custom header to pass to the callback

Pre-Selecting Rows in a REST Grid Widget

The `SelectRowsByData` function in the REST Grid widget can be used to select rows in the grid by referring to data in the row to select. This topic describes the function and provides example JavaScript code.

Note: See the *SBM Composer Guide* for widget information.

SelectRowsByData(columnName, values, fireEvent)

Name	Type	Description
`columnName`	String	The name of the column to search for the values supplied in the `values` parameter.  **Note:** This should be a fully-qualified column name, from the root element down.
`values`	String	The values of the given column that correspond to the rows to select.
`fireEvent`	Boolean	Whether to fire a selection event when the rows are selected. The default value is false.

Example

This example refers to the Yahoo geocode REST service referenced in Querying REST Service Results [page 38]. The following code uses the GetWidget Method [page 42] method. It will select the rows in the grid that correspond to the given values for the "ResultSet.Results.line2" column.

 **Note:** For a process app that demonstrates this functionality, go to solution S138336 in the Knowledgebase and download the `REST Grid ENH.msd` file.

```
var grid = GetWidget( "RESTGridWidget" );
grid.SelectRowsByData( "ResultSet.Results.line2", ["Springfield, CO",
→"Springfield, ID"], true );
```

Intercepting and Modifying REST Grid Widget Data

The `AddResultCallback` object can be used to intercept and modify REST Grid data. This topic describes the object and the methods that perform this function, and provides example JavaScript code.

`AddResultCallback` sets a user-defined JavaScript callback function, the purpose of which is to modify data before it's displayed.

Parameters

Name	Type	Description
`fieldName`	String	The name of the widget using the callback function. Currently only REST widgets are supported.
`callback`	Function	A callback function for modifying the widget's data. It must accept a data object as its only parameter and return the modified data object in a usable format.

Return Value

Result	Value
(none)	

Example

This example passes a function that replaces the first row of data in a REST Grid widget named *MyRESTGridWidget* with the given text strings, where the attribute *responseData* is specific to the data returned by the REST service.

```
AddResultCallback("MyRESTGridWidget",
  function(data) {
    data.responseData.results[0].title = "Micro Focus";
    data.responseData.results[0].content = "Solutions Business Manager";
    data.responseData.results[0].visibleUrl = "www.microfocus.com";
    return data;
  });
```

For a more detailed example, see Community website.

Comments

None.

GetWidget Method

Retrieves a reference to the REST Grid widget object for the given name.

Parameters

Name	Type	Description
`objName`	String	The name of the widget control.

Return Value

Result	Value
Success	The JavaScript widget object.
Failure	Null

Examples

See Pre-Selecting Rows in a REST Grid Widget [page 40] for an example of this method.

Comments

(none)

ShowErrorDiv Method

Displays an error message at the top of a page.

Parameters

Name	Type	Description
`show`	Boolean	Whether to display the message.
`message`	String	The message to display.

Return Value

Result	Value
(none)	

Example

This example displays an error message if the *Description* field is not filled in before a page is submitted:

```
function customSubmitCallback()
{
    // Prevent submit if Description field is empty
    if ( IsFieldEmpty( "Description" ) )
    {
         ShowErrorDiv( true, "A description is required" );
         return false;
    }
    else
    {
         ShowErrorDiv( false );
         return true;
    }
}

AddSubmitCallback( customSubmitCallback );
```

Comments

(none)

Internal Functions

Internal functions are typically used by other functions in the SBM JavaScript library.

- GetFieldByName [page 44]
- GetLabelByName [page 44]
- GetFieldWidgetByName [page 45]

GetFieldByName

Finds an HTML DOM object that matches the specified name. Use this command to find the field to use in other JavaScript calls.

Note: Use the GetFieldWidgetByName [page 45] method for complex field types (such as *Single Selection*, *Multi-Selection*, *Single Relational*, and *Multi-Relational*).

Parameters

Name	Type	Description
`fieldName`	String	The name of the field to find. The **Name** or **Database field name** specified on the field Property Editor can be used.

Return Value

Result	Value
Success	Field object corresponding to a field. This can be an `<input>`, `<select>`, `<textarea>`, or `<span>` element.
Failure	Null

Examples

This example returns the field object:

```
GetFieldByName("State");
```

Comments

(none)

GetLabelByName

Finds an HTML DOM object that matches the label for the specified name. Use this command to find the label to use in other JavaScript calls.

Parameters

Name	Type	Description
objName	String	The name of the field label to find.

Return Value

Result	Value
Success	Label object corresponding to a field. This is a <label> element.
Failure	Null

Examples

This example returns the label object:

```
GetLabelByName("Unit Price");
```

Comments

(none)

GetFieldWidgetByName

Gets the wrapper element for the specified complex field type (such as *Single Selection*, *Multi-Selection*, *Single Relational*, and *Multi-Relational* fields) or form widget (such as the PDF widget and REST Grid widget).

Parameters

Name	Type	Description
objName	String	The name of the complex field or form widget control.

Return Value

Result	Value
Success	The wrapper element for the complex field type or widget.
Failure	Null

Examples

This example returns the wrapper element for the "Testers" *Multi-Selection* field:

```
GetFieldWidgetByName("Testers");
```

Comments

(none)

JavaScript Examples

Setting Field Properties Based on Field Values

To recreate these examples, your application should contain the following fields and they should be added to your custom form:

Field type	Database name	Display name	Values
`Single Selection`	REQUEST_TYPE	Request Type	Billable Nonbillable
`User`	ACCOUNT_MANAGER	Account Manager	Users assigned to roles specified for this field or added as values for the field in SBM Application Administrator.
`Text`	SOW_NUMBER	SOW Number	Specified by users.
`Binary`	SALES_APPROVAL	Sales Approval	Yes No
`User`	SALES_MANAGER	Sales Manager	Users assigned to roles specified for this field or added as values for the field in SBM Application Administrator .

Example 1

In this example, you can set the *Account Manager* and *SOW Number* fields as required when users select **Billable** from the *Request Type* field. If another value is selected from the *Request Type* field, the *Account Manager* and *SOW Number* fields are removed from the form, and are made optional.

```
function RequestTypeChanged()
{
    var text = GetFieldValue("REQUEST_TYPE");
    if (text == "Billable") {
        MakeFieldRequired("ACCOUNT_MANAGER", true);
        MakeFieldRequired("SOW_NUMBER", true);
    }
    else {
        HideField("ACCOUNT_MANAGER");
        HideField("SOW_NUMBER");
        MakeFieldOptional("ACCOUNT_MANAGER", false);
        MakeFieldOptional("SOW_NUMBER", false);
    }
}

AddChangeCallback("REQUEST_TYPE", RequestTypeChanged);
```

Example 2

In this example, you can set the *Account Manager* and *SOW Number* fields as required and display the *Sales Approval* field when users select **Billable** from the *Request Type* field. If another value is selected from the *Request Type* field, the *Account Manager*, *SOW Number*, and *Sales Approval* fields are removed from the form, and *Account Manager* and *SOW Number* fields are made optional. In addition, when users select **Yes** in the *Sales Approval* field, the *Sales Manager* field appears on the form. If another value is selected, the *Sales Manager* field is removed from the form. **Note:** The *Sales Approval* field must be a drop-down list. It cannot be a radio button or a check box.

```
function RequestTypeChanged()
{
    var text = GetFieldValue("REQUEST_TYPE");
    if (text == "Billable") {
        MakeFieldRequired("ACCOUNT_MANAGER", true);
        MakeFieldRequired("SOW_NUMBER", true);
        ShowField("SALES_APPROVAL");
    }
    else {
        HideField("ACCOUNT_MANAGER");
        HideField("SOW_NUMBER");
        HideField("SALES_APPROVAL");
        MakeFieldOptional("ACCOUNT_MANAGER", false);
        MakeFieldOptional("SOW_NUMBER", false);
    }
}
```

```
AddChangeCallback("REQUEST_TYPE", RequestTypeChanged);

function SalesApprovalChanged()
{
    var text = GetFieldValue("SALES_APPROVAL");
    if (text == "Yes") {
        ShowField("SALES_MANAGER");
    }
    else {
        HideField("SALES_MANAGER");
    }
}

AddChangeCallback("SALES_APPROVAL", SalesApprovalChanged);
```

Changing Field Properties Based on Date Change

To recreate these examples, your application should contain the following fields and they should be added to your custom form:

Field type	Database name	Display name	Values
Date/ Time	DUE_DATE	Due Date	On the **Options** tab, select **Date Only**.
User	EMRG_APPROVER	Emergency Approver	Users assigned to roles specified for this field or added as values for the field in SBM Application Administrator.
Text	EMRG_REASON	SOW Number	Specified by users.

Example

In this example, when users specify a value in the *Due Date* field that is fewer than seven days from the current date, the *Emergency Approver* and *Emergency Reason* fields are set as required. If the *Due Date* field value is more than seven days past the current date, the *Emergency Approver* and *Emergency Reason* fields are removed from the form. If the due date is past January 1, 2015, users are informed that the due date is too far in the future.

```
function DueDateChanged()
{
    var text = GetFieldValue("DUE_DATE");
    var DueDate = new Date();
    DueDate.setTime(Date.parse(text));

    var NextWeek = new Date();
    NextWeek.setDate(NextWeek.getDate()+7);
```

```
    if (DueDate < NextWeek) {
        MakeFieldRequired("EMRG_APPROVER", true);
        MakeFieldRequired("EMRG_REASON", true);
    }
    else {
        HideField("EMRG_APPROVER");
        HideField("EMRG_REASON");
    }

    var FixedDate = new Date();
    FixedDate.setFullYear(2015,1,1);
    if (DueDate > FixedDate) {
        alert('Date too far in the future!');
    }
}

AddChangeCallback("DUE_DATE", DueDateChanged);
```

Changing Field Properties Based on Field Value Length

To use the following example, verify that the system *Description* field has been added to your application and is available on your custom form. This example also uses the system *Title* field, which is automatically added to all applications.

Example

In this example, the *Description* field is set as required if the *Title* field contains fewer than 20 characters.

```
function TitleChanged()
{
    var text = GetFieldValue("TITLE");
    if (text.length < 20) {
        MakeFieldRequired("DESCRIPTION");
    }
    else {
        MakeFieldOptional("DESCRIPTION");
    }
}

AddChangeCallback("TITLE", TitleChanged);
```

Marking a Field as Optional or Required

In this example, the *Description* field is optional if the *Request_Amount* field is less than $1000.00. This example assumes that the *Request_Amount* field is numeric. If it is a string, it must be converted into a number before the comparison is made.

Example

For this example, the *Description* field must not be set as required in SBM Composer or overridden in SBM Application Administrator.

```
function RequestAmountChanged()
{
    var amount = GetFieldValue("REQUEST_AMOUNT");
    if (amount < 1000.00) {
        MakeFieldOptional("DESCRIPTION");
    }
    else {
        MakeFieldRequired("DESCRIPTION");
    }
}

AddChangeCallback("REQUEST_AMOUNT", RequestAmountChanged);
```

Using the tzOffset Variable

To recreate this example, add the following fields to your application and custom form:

Field type	Database name	Display name	Values
`Date/Time`	LOCAL_DATE_TIME	Local Date Time	On the **Options** tab, select **Date and time**

Example

In this example, when the form is loaded the value of the **Local Date Time** field is updated to use the time zone offset that is selected by the current user in his or her user profile. This script uses the `tzOffset` variable, which is always accessible via JavaScript in SBM. The `tzOffset` variable represents the time zone offset for the given user in seconds in relation to GMT, and it can be used to make calculations like the one below.

```
function UpdateDateTimeField()
{
 var now = new Date();
 var adjustedUTCTime = now.getTime() +
 →(tzOffset * 1000) + (now.getTimezoneOffset() * 60 * 1000);
 var adjustedDate = new Date(adjustedUTCTime);
 SetFieldValue("LOCAL_DATE_TIME", adjustedDate);
}
AddLoadCallback(UpdateDateTimeField);
```