IBM® Tivoli® Software

Maximo for Building Information Models

Viewer Integration Framework

Document version 7.6.1.2

Contact: Thomas Knowles *tknowles@us.ibm.com*

The Maximo Autodesk Forge Plugin is released as trial software and distributed on the IBM Developer Community and Github. An existing Maximo Asset Management 7.6 license and installation is required. Trial software is not supported by the IBM Maximo support program; however, support is available directly from the IBM Maximo development team for as long as the software is available for download on the IBM Developer Community website. Check the community for the most recent version as well as continued availability. Only an English version is distributed during the trial. Your feedback is important because the software is being evaluated for possible integration with other Maximo products. Send all support questions and feedback to tknowles@us.ibm.com

CONTENTS

Contents

Changes History

Version 1.2.2

- Extracted from Maximo extensions for Building Information Models- User's Guide

Audience

This guide is targeted a Software developer who wish to write a viewer plug-in for the Maximo BIM module.

1 Introduction

Maximo provides a framework for BIM viewer vendors to integrate their solution into Maximo. Integrated viewers have the following capabilities:

- Automatic selection and display of the correct model
- Viewer context is synchronized to Maximo
- Maximo context is synchronized to the viewer
- The viewer may be used for asset selection
- Create work orders and service request from the viewer
- Display assets with open work order or service requests.
- Display Create and Edit Maximo systems

The BIM ISM solution contains two applications that have not been added to core Maximo. These are

- Product
- Design Specifications

These applications are enabled by the Forge viewer plug-in and may be enabled by 3rd party viewer plug-ins

1.1 Technology requirements

To be suitable for integration a viewer must meet the following requirements:

- Ba bale to be displayed in an HTML iFrame
- Have some unique identifier for each object in the viewer that can be stored in Maximo
- Support JavaScript scripting to:
  - Create instance of the viewer
  - Load a file in the viewer
  - Query the current selection returning a list of unique identifiers for the selected objects
  - Select one or more items based on the unique identifier
  - Report when the selection has changed

2 Framework Overview

The BIM viewer is implemented as a Maximo control. A typical Maximo control pairs a Java class that handles the Maximo processing with a .jsp that generates the HTML required to render the control. The architecture provides for pushing scripts to the browser without re-rendering the HTML. This is done via dojo. The mechanics are transparent to the viewer.

A Maximo control can be bound to a field in a Maximo Business Object (Mbo) through the datasource tag in the presentation XML where the control is referenced. The viewer control instance is generic and supports binding to any field in any Mbo. The control receives notifications whenever the value of the bound field changes.

All current usage binds the viewer to the ModelId field of the location or Asset Mbo, except for the Mange Building Models application where it is not bound.

The .jsp associated with the viewer is bimviewer.jsp. However, it loads an iFrame that source is viewerframe.jsp. most of the viewer and all of the integration frame work is implemented in the iFrame. The iFrame exposes a JavaScript control interface that allows Maximo to drive the viewer behavior through scripts run in bimviewer.

The Maximo control .jsp files are found in the directory

\applications\maximo\maximouiweb\webmodule\webclient\components

viewerframe,jsp has 4 includes that are found in bimvendor subdirectory. These are integration points.

The Autodesk Forge integration is packaged with this distribution and serves as a template for other viewer integration.

2.1 Selection Management

The viewer framework treats the viewer implementation as the authoritative source of the current selection and always asks when selection information is needed. This works well when the viewer is local such as the NavisWorks ActiveX control but may cause performance problems if the viewer implementation is remote. Viewer that require a network call to get the current selection should consider local caching.

The framework expects the viewer to communicate about the selection in terms of the values of the field in the Mbo the viewer is bound to. In theory, this is generic. In practice, the viewer is always bound to the ModelId field of the Location or Asset Mbo, and the value of that field is expected to be populated via the COBie import. This means the effective mechanism for commination about the selection is the IFC or Revit GUID.

The framework expects the viewer to have a zero based array with zero or more values (GUIDs) which is the current selection set. It also maintains an index into the array which is the active item, and if it exists in the Maximo database, the current Maximo item. There is a mechanism for stepping the active item forward or backward though the selection array.

2.2 The viewer and tabbed applications

Many of the Maximo applications have a list tab and several record specific tabs. The viewer is added as a new tab in these applications. When a tab is displayed, the HTML for the tab is rendered, but the rest of the application is not re-rendered. This could create a lot of inefficiency for a viewer on one of the tabs since it would have to redraw and reload the model each time the view tab is selected. To address this, the viewer is not rendered on a tab. It is rendered at the bottom of the application page, then its screen position is managed based on the selected tab so it is either off screen or positioned to appear as if it is on the displayed tab.

2.3 Status notification

There is a status line at the bottom of the viewer that should be used for informational and error messages to the user. In the default implementation, these include error encountered loading a model, and notification if an item request for selection by Maximo is not in the model (or current view).

The HTML id of the status bar is <%=statusId%>

2.4 Internationalization

The viewer externalizes strings that require translation into the Maximo component-registry.xml file and the Maximo message catalog. The Maximo component architecture is not fine grained enough to allow for easy vendor extension of this file within the context of a single component. Therefore, the viewer framework does not address internationalization and translation. Vendors supplying viewer plugins must provide for translated versions themselves.

The Forge viewer loads some strings from the Maximo message catalog. This is implemented in bimlmv/strings.jsp. This mechanism may be cloned and extended by new viewer implementations

2.5 Naming and directory structure

The viewer framework allows multiple viewers to be installed in Maximo, but only a single viewer may be active. Each viewer must have an internal name that is used to isolate and select its files and UI elements. A list of installed viewer is maintained in the synonym domain

BIMVIEWERTYPE

The active viewer is controlled by the system property

bim.viewer.active

The active viewer name determines the directory from which the viewerframe.jsp and its includes are loaded, and which section is displayed in the Manage BIM Viewer application

2.6 Integration steps

The basic steps to integrate your BIM viewer into Maximo are:

1. Add and entry to the synonym domain BIMVIEWERTYPE of the form:

```
<synonymvalueinfo value="MyViewer"
                  maxvalue=" MyViewer "
                  defaults="true"
                  description="My 3rd Party viewer" />
```

2. Copy or rename maximouiweb/webmodule/webclient/components/bimvendor to bimMyViewer where MyViewer is the case sensitive name of your viewer
3. Implement bimMyViewer\header.jsp to load any libraries your viewer requires
4. Implement bimMyViewer\script.jsp to include any instance specific scripting you need into the viewer. This includes the interface for the ViewerWrapper class, and the stub for LoadControl – Both described below.
5. Optionally, Implement bimMyViewer \toolbar.jsp to define the bottom toolbar of the viewer. This should include any navigation or control interface required by your viewer. You will also need to copy some of the NavisWorks implementation: the navigation buttons that appear on the right side of the toolbar. The Forge viewer uses the native toolbar but calls several methods in the support code to implement the toolbar functions.
6. If necessary, implement bimMyViewer /footer.jsp. The NavisWorks implementation uses this to define iFrames for many of the popups
7. Implement the ViewerWrapper JavaScript interface described below. This defines all but one of the methods Maximo uses to drive your viewer's context and behavior

8. Implement the loadControl JavaScript function described below to generate the dynamic HTML required to create your viewer.
9. Implement notification to the viewer framework of selection change events in your viewer (Described below)
10. Add a section in the bimmodel application presentation with an id of myviewer_main and define the UI needed to associate your viewer with a location. Do the same thing for the BIM_ADDMOD dialog in the LOCATION application.
11. Add any images you require to the directories:
    maximo\applications\maximo\maximouiweb\webmodule\webclient\skins\tivoli09\Images\bim
    maximo\applications\maximo\maximouiweb\webmodule\webclient\skins\tivoli13\Images\bim

    The JSP files in the bimmyviewer directory may be altered anyway the viewer implementation sees fit. For example, the Forge viewer implementation does not include ..\bim\maximo-toolbar.jsp replacing it with a toolbar native to the viewer, but using the same scripts as call from that JSP.

    However, the supported interface is what is described below. Changes that circumvent the below API may not be compatible with new versions of the framework.

3 Writing the viewer JSP files

3.1.1 Viewer Initialization

Note: The JavaScript classes referenced below are found in bimviewr.js

When the viewer is initialized – a full re-render of the HTML, the following is the basic flow:

1. bimserver.jsp is loaded which includes an iFrame that loads bimMyVierwer/viewerframe.jsp
2. While rendering the viewer table, `loadControl` is called to create the viewer object
3. On load bimviewer.jsp runs scripts that:
   a. Sets the position of the entire control (more on positioning below)
   b. Call the `setModelVisibility` method to select to display either a message indicating no model is available, or the viewer
   c. Calls `initModelManager` which returns the `ModelManager` instance
      i. Creates an instance of the ViewerWrapper class
      ii. Creates a `ModelManager` and s `SelectionManager` instance
      iii. Calls `ViewerWrapper.initialize( modelMgr, selMgr );`
   d. `Calls ModelManager.resetModelList();`
   e. For every model in the model list: Calls `ModelManager.addModel(…);` The model list is pre-sorted by priority
   f. Calls `ModelManager.populateModelList( value )` where value is the ModelId of the current Maximo location
      i. Populates the model selection combobox

ii. Calls `ViewerWrapper.setCurrentModel()` with the `ModelDef` record for the first model in the list

iii. Calls `ViewerWrapper.loadFile( url );` with the url for the current model (From the URL field of the BuildingModel Mbo. (This is opaque so your viewer can use something other than a URL)

iv. Calls `ViewerWrapper.selectValue( value, isAutoZoom );`

3.1.2 Viewer Updates

There are four situations that can result in viewer scripts being called without the viewer being re-rendered:

1. The current Maximo locations has changed and the new location has a different list of models

   This is similar to 3.c to the end above except that the existing ModleManager, ViewerWrapper, and SelectionManager instances are used, and if the current model is in the new model list it remains the current model and no calls to `ViewerWrapper.serCurrentModel()` or `ViewerWrapper.loadFile( url );` are issued.

2. The current Maximo location has changed, but the list of models is the same as for the previous location

3. A Maximo dialog requests the viewer to display a selection set

   The `multiSelect( selectionArray[] );` method is called with a list of ModelIds.

4. There is a message to display on the status line of the viewer.

   The `setStatus( msg );` method is called

3.2 The Building Model table

As described in Maximo BIM - Forge Viewer User's Guide, model files, or really model URLs or definitions are associated with Maximo locations. This is done through the BuildingModel table described below.

Name	Title	Type	Length	Default	Notes:
ASSETVIEW	Asset	ALN	35		Default saved view to display in asset app
ATTRIBUTECLASS	Find Attribute Class	ALN	50	LcRevitData	Class of find attribute to use with the model file
ATTRIBUTENAME	Find Attribute Name	ALN	50	Element	Name of find attribute to use with model file
DESCRIPTION	Description	ALN	100		Describes the Building Model. To enter or view additional information, click the Long Description button
LOCATION	Location	UPPER	12		Identifies the Building Model's location.
SYSTEMID	System name	UPPER	12		Identifies the Building Model's system.
LOCATIONVIEW	Location	ALN	35		Default saved view to display in location app
LOOKUPVIEW	Lookup	ALN	35		Default saved view to display in the lookup dialog
ORGID	Organization	UPPER	8		Organization Identifier
PARAMCLASS	Find Parameter Class	ALN	50	LcRevitData	Class of find parameter to use with model file
PARAMNAME	Find Parameter Name	ALN	50	Guid	Name of find parameter to use with model file
PRIORITY	Priority	INTEGER	4	0	Sort order for model list in viewer
SITEID	Site Identifier	UPPER	8		
TITLE	Title	ALN	30		Title
URL	URL	ALN	1024		URL for the Building model file for this location
VIEWERTYPE	Viewer	ALN	30	lmv[1]	The viewer to use with the model file
WORKORDERVIEW	Work order	ALN	35		Default saved view to display in the work order app

Information from the BuildingModel table is copied into the ModelDef JavaScript class and passed to the viewer as follows:

```
function ModelDef(
      modelId, location, binding, title, url, attribClass, attribName,
paramClass, paramName, defaultView
) {
      this.modelId     = modelId;
      this.location    = location;
      this.binding     = binding;
      this.title       = title;
      this.url         = url;
      this.attribClass = attribClass;
      this.attribName  = attribName;
      this.paramClass  = paramClass;
      this.paramName   = paramName;
      this.defaultView = defaultView;
}
```

[1] Default value is vendor if no viewer is installed, and lmv if the Forge viewer is installed.

modelId: A sequence #
location: The location ID of the Maximo location associated with the model
binding: The ModelId field for location with which the BuildingModel is associated. If the location was created through a COBie import, this should track back to the IFC or Revit GUID.
title: The Title field from the BuildingModel table. If the Title field doesn't have a value, then the location ID of the Maximo location associated with them model is used. This is the value displayed in the Select Model combobox.
url: The URL field from the BuildingModel table
attribClass: The AttributeClass field from the BuildingModel table
attribName: The AttributeName field from the BuildingModel table
paramClass: The ParamClass field from the BuildingModel table
paramName: The ParamName field from the BuildingModel table
defaultView One of AssetView, LocationView, LookupVoew, or WorkorderView fields from the BuildingModel table. The field is selected based on the context in which the viewer is displayed.

Name	Value	Type
modelId	1	Number
location	"CC-000001"	String
binding	"e8588825-8be1-496d-b15d...	String
title	"Clinic"	String
url	"http://192.168.196.2/maxi...	String
attribClass	"LcRevitData"	String
attribName	"Element"	String
paramClass	"LcRevitData"	String
paramName	"Guid"	String
defaultView	"All Rooms"	String

The fields url to the end of the class are not used by the viewer framework and may be used by the viewer plugin however required.

The BuildingModel Mbo may have additional fields added as needed, but there is no mechanism to pass these fields to the viewer JSP.

3.3 JSP integration points

The Java control class, viewerframe.jsp and the maximo header JSP define many variables. Some of the more useful are:

`ctrlId;` The HTML id that is assigned to the viewer Object

`appType` The context in which this viewer instance is displayed. The framework uses this to customize the options on the top toolbar, and to determine it is displayed on a dialog or main application and tailor scripts accordingly. Valid values are:

```
BIMViewer.TYPE_UNKNOWN
BIMViewer.TYPE_ASSET
BIMViewer.TYPE_LOCATION
BIMViewer.TYPE_LOOKUP
BIMViewer.TYPE_WORKORDER
```

BIMViewer.TYPE_MODEL

bldgMdl.getRecordType() The Maximo record to which the viewer is bound. Valid values are:
BIMViewer.RECORD_UNKNOWN
BIMViewer.RECORD_ASSET
BIMViewer.RECORD_LOCATION
BIMViewer.RECORD_MODEL

height The height of the control specified in the presentation XML

width The width of the control specified in the presentation XML

toolabarHeight The height of the top and bottom toolbar as specified in the control definition in maximo\properties\registry-extensions/bim-component-registry.xml

BIM_IMAGE_PATH The path to the viewer image directory adjusted for the correct Maximo skin

4 JavaScript API

The viewer plugin must implement the following JavaScript interface:

4.1 Viewer Wrapper

This is the primary interface between the viewer framework and the viewer plugin. It should be in the scripts.jsp file in the bimmyviewer directory so it is included in viewerframe.jsp. The viewer framework will instantiate an instance of it.

```
/*****************************************************************/
// This class contains all the functions needed to interact with the
// native viewer
/*****************************************************************/
function ViewerWrapper(
      ctrl  // The control handle
) {

    // Return the version of this interface.  Implementations written to
    // this spec should always return "1.0"
    this.getInterfaceVersion = function()
    {
      return "1.1";
    };

    // Called once after all objects are created, but before any methods
    // are called with the exception of setCurrentModel.
    // modelMgr       The ModelManager instance
    // selectionMgr   The SelectionManager instance
    // Return:        Nothing
      this.initialize = function(
            modelMgr,
            selectionMgr
      ) {
      };
```

```
// Request that the viewer load the specified file.  Errors should
// be reported on the viewer status line by calling
// setStatus( status )
// file:          The URL attribute from the Maximo
//                BuildingModel table
// Return:        Nothing
  this.loadFile = function(
        file
  ) {
  };

// Requests the viewer plugin to select a single item clearing
// any previous selection
// Value          Id of item to select
// zoomToContext: Flag indicating if the viewer should zoom in on
//                the resulting selection set.
// Return:        Nothing
  this.selectValue = function(
        value,
        zoomToContext
  ) {
  };

// Request the view plugin to select a list of items clearing
// any previous selection
// valueList:     Array of ids that is the desired selection set.
// zoomToContext: Flag indicating if the viewer should zoom in on
//                the resulting selection set.
// Return:        Number of items found and selected
  this.selectValueList = function(
        valueList,        // Array of itds to select
        zoomToContext
  ) {
  };

// Called when the current model is changed. This method does not
// need to load the new model.  loadFile is called for that
// Return:        Nothing
  this.setCurrentModel = function(
        currentModel
  ) {
  };

// Requests the value (id) of the selected item with a specified
// index in the selection list.
// index   The index of the active selection item in the current
//         selection list
// return  The id of the item in the selection list referenced by
//         index. If the index is out of bounds then an empty string
//         is returned
  this.getSelection = function(
        index
  ) {
  };
```

```
// An array of the currently selected items ids
this.getSelectionList = function()
{
};

// Return:  THe number of items currently selected
this.getSelectionCount = function()
{
};

// Search the selection list for selected item and return the imdex
// Return the index of the selected item in the selection list.
this.getItemIndex = function(
  selectedItem      // Any item in the list of selected items
) {
};

// Clear all selected item and set the selection list to zero length
this.clearSelection = function()
{
};

// Zoom the current view to focus on the item in the selection set
// indicated by index
this.focusOnSelectedItem = function(
  itemIndex   // The index of the item in the selection list
) {
};

this.enableMultiSelect = function (
  enable
) {
};

  // Called to notify the viewer of changes to the auto soom state
  // The viewer need implement this only if it supports auto zoom
  // beyond what is controled by the flags in setValye and
  // setValueList
  this.setAutoZoom = function(
        enable
  ) {
  };

  // Called when the controling applcation has resized the viewer
  // container The viewer must adjust to the new container size
  this.reziseViewer = function(
        height,
        width
  ) {
  };
```

```
    // Called when the selection changes after the common processing
    // ctrl           HTML id of the viewer control
    // selectionList  Array of Ids that are currently selected. this
    //                is the list returned from calling
    //                this.getSelectionList
    // selection,     The active item in the selection set.
    // count,         The number of items in the selection set.  This is
    //                the value returned from calling
    //                this.getSelectionCount
    // index          The index in the selection set of the active item
    //                This is the result of calling this.getItemIndex
    // Return:        Nothing
    this.onSelectionChange = function(
            ctrl,
            selectionList,
            selection,
            count,
            index
      ) {
      };
}
```

4.2 Load Control

This method is called synchronously while the viewer is being rendered. It must create the actual viewer as a descendant of the parentCtrl. It may create any additional controls that it requires as dedendents of the parentCtrl. The NavisWoks implementation creates the property side panel in this method. The viewer control must have its HTML id specified as `<%=ctrlId%>`.

The `versionLableCtrl` is a small label next to the logo in the default top toolbar that can be used to convey version or other information about what was loaded to the user.

```
/*********************************************************************/
// Loads the NavisWorks control tryng different versions then sets the
// lable next to the AD logo to the version loaded
/*********************************************************************/
   function loadControl(
            parentCtrl,
            versionLableCtrl
      ) {
      }
```

4.3 Selection Events

Whenever the current selection in the viewer changes though user action, including being cleared, the viewer must notify the framework. By calling

```
selMgr.updateSelectionSet( <%=ctrlId%> )
```

The framework should not be notified if the selection changes as a result of a call to `ViewerWapper.selectValue()` or `ViewerWapper.selectValueList()`

4.4 IFrames and Objects in Maximo

The dialogs in Maximo are not implemented as iFrames. Therefore, they don't draw on top of HTML object such as ActiveX controls and may not draw on top of iFrame depending on

what is loaded into the iFrame. To address this, when Maximo displays a menu or a dialog it searches the DOM tree in the browser for all entitles of type OBJECT and sets their size to 0 height and width. When all the dialogs that should be rendered on top of the OBJECT are closed, the objects are restored to their original size.

By default, this behavior is not applicable to iFrames, however it can be requested. If an iFrame has and attribute named “autoHide” and it has a value of true, it is hidden when menus and dialogs are displayed. If your viewer requires the auto hide behavior, you must set the attribute when the viewer object is created or when the ViewerWrapper is initialized.

```
/* Cause the whole frame not just the viewer to auto hide */
var frames = window.top.document.getElementsByTagName("IFRAME");
var l = frames.length;
for(var i = 0; i < l; i ++ )
{
      var frame = frames[i];
      if( frame.id == "<%=id%>_frame" )
      {
            frame.setAttribute( "autoHide", true );
      }
}
```

5 Database and UI updates

The Maximo updated command runs scripts to extend the Maximo database, UI, and message catalog. It also adds listing of extensions to the Maximo about dialog. In may be utilized by 3rd party viewer plug-ins.

Script execution and the help listing is controlled by an XML file in the directory

applications\maximo\properties\product

By convention, the file should be name

bimmyviewer

Here is a sample:

```
<product>
      <name>My Viewer viewer plug-in</name>
      <version>
            <major>7</major>
            <minor>6</minor>
            <modlevel>0</modlevel>
            <patch>6</patch>
            <build>$build$</build>
      </version>
      <dbmaxvarname>BIMMYVIEWER</dbmaxvarname>
      <dbscripts>bimmyviewer</dbscripts>
      <dbversion>V7606-09</dbversion>
      <lastdbversion>V7606-00</lastdbversion>
</product>
```

`dbscripts` names the directory where the database, UI, and message scripts are located. For an English language install it is:

tools\maximo\en\bimmyviewer

`lastdbversion` names the last script in the previous version.

`dbversion`

Script number must be sequential as illustrated below

The forge viewer scripts are used as an example. Several of these scripts can be re -used by 3rd party viewers.

Flite Name	Description
V7606_01.mxs	UI for Product and Design Specifications applications
V7606_02.mxs	Add viewer option to asset lookup menus
V7606_03.mxs	UI Extensions for BIM Projects application to enable import of Product data
V7606_04.mxs	Forge viewer UI extensions to Mange BIM Viewer application
V7606_05.msg	Message catalog for Forge viewer translatable messages
V7606_06.dbc	Grants right for MAXADMIN group to BIM Projects and Manage BIM Viewer applications so they are visible after a viewer install
V7606_07.dbc	Creates the application definition for the Product application
V7606_08.dbc	Creates the database table definitions as application definitions for the Design Specification application
V7606_09.dbc	Creates the tables and application extensions needed for the Forge

The non-Forge specific file may be copies, renamed, and redistributed by a 3rd party viewer to allow it to enable the Product and Design Specifications applications

The tool:

`tools\maximo\screen-upgrade\mxdiff`

Can be used to create mxs files to install UI changes. It calculates the difference between an old file and a new file and creates a script to upgrade the old file to the new file. This can be used to install the changes required for the bimmodels application

Appendix

A portion of the Java control class that generated the JaavScript calls to bimviewer.jsp is presented her to aid in understanding and debugging. This interface is unsupported and may change without notice

```
/*
 * (C) COPYRIGHT IBM CORP. 2010 - 2016
 */

//*************************************************************************
// The section has methods that are called from the .jsp to move data from
// Maximo to the .jsp.  Many of the methods clear the data once it is transfered
//
// These methods should not be used except by the .jsp
```

```
   //*****************************************************************************

   public Set<String> jspGetMultiSelection()
   {
        Set<String> selection = _multiSelection;
        _multiSelection = null;
        return selection;
   }

   /**
    * Generate JavaScript to configure the current state of the control
    * <p>
    * This method supports the following control states and changes between those
states:
    * - The entire control is either in its proper position on the screen or is
"Stored"
    *   off the top of the screen
    * - There is a model file for the current bound value and the active X control is
    *   displayed, or there is not and it is hidden and a message is displayed
    * - If it is a partial refresh and the HTML does not need to be rerendered, then
this
    *   code is executed inside a script tag in the hidden frame.  IF it is run as
part
    *   of rendering the control then it is run wint the control HTML in the MAINDOC
    * - If it is a partial refresh, it may update the selected value, the both the
model file
    *   and the selected value, or post a message to the status line
    */
   public String jspScript(
        String id
   )
        throws RemoteException,
               MXException
   {
        StringBuffer script = new StringBuffer();
        script.append( "" );

       boolean designmode = _wcs.getDesignmode();

        String containerTable   = id + "container";

        boolean needsRendered = needsRender();

        //If the code is run in the hidden frame, then calls need to be prefixed with
        // MAINDOC this only applies to pre 7.5 versions
        String doc = "";
        if(    !needsRendered
           && getMxVersion() < VERSION_75_OR_GREATER )
        {
           doc = "MAINDOC.";
        }

        String value   = null;

        // Set the top value
        script.append( "try {" );
        script.append( "var containerTbl = " + doc + "document.getElementById( \"" );
```

```
        script.append( containerTable );
        script.append( "\" ); " );
        script.append( "if( containerTbl != undefined ) { " );
           script.append( "containerTbl.style.top = \"" + jspGetViewerTop() + "px\";"
);
           script.append( "containerTbl.style.left = \"" + _leftOffset + "px\";" );
        script.append( "}" );

        script.append( "var isLoaded = true; " );
        script.append( "AF = window.frames." + id + "_frame; " );
        script.append( "var frame = window.top.frames." + id + "_frame; " );
        script.append( "if( frame != undefined && frame.contentWindow != undefined ) {
" );
           script.append( "frame = frame.contentWindow;" );  // Chrome
        script.append( "} " );
        script.append( "if( frame == undefined || frame.setModelVisibility ==
undefined ) { " );
           script.append( "isLoaded = false; " );
        script.append( "} " );

        // The controls has two forms:
        // - A message indicating there is no model file
        // - The actual control
        // The correct form is selected based on wether the current location has a
model file
        //
        // If the control is alredy displayed, then it may be necessary to select
between
        // The message window and the control window. Its simpler to always set these
values
        // The to track the current state
        boolean showModel = false;

        script.append( scriptResize() );

        if( isControlVisible() )
        {
           showModel = itemHasModel();
           value = getValue();
           script.append( "if( isLoaded ) { " );
              script.append( "frame.setModelVisibility( " );
              script.append( showModel );
              script.append( " ); " );
           script.append( "} " );
        }

        //View tab is selected so the control should be visible on the page
        if( isControlVisible() && showModel && !designmode )
        {
           boolean hasScript = false;
           if( jspHasStatusUpdate() )
           {
              script.append( "frame.setStatus( \"" );
              script.append( jspGetStatusUpdate().trim() );
              script.append( "\" ); " );
              hasScript = true;
           }
```

```
            if( isHasMultiSelect() )
            {
               Set<String> selection = jspGetMultiSelection();
               Iterator<String> itr = selection.iterator();
               script.append( "var selection = new Array(); " );
               while( itr.hasNext() )
               {
                    script.append( "selection[ selection.length ] = \"" );
                    script.append( itr.next() );
                    script.append( "\"; " );
               }
               script.append( "frame.multiSelect( selection, \"" );
               script.append( value );
               script.append( "\", " );
               if( _multeSelectZoomToContext )
               {
                    script.append( "true" );
               }
               else
               {
                    script.append( "false" );
               }
               script.append( " ); " );

               _multeSelectZoomToContext = false;
               hasScript = true;
            }

            if( hasScript )
            {
               script.append( "} catch( e ) { console.log( \"BIMViewer error\\\n\" );
console.log( e ); }" );
               return script.toString();
            }

            // If the model file has changed due to a change in the value bound to
            // the control, or the control is being rerendered, then the models
            // must be reloaded into the control
            if( isModelListChanged() || needsRendered )
            {
               script.append( "if( isLoaded ) { " );
                    script.append( "var mm = frame.initModelManager(); " );
                    script.append( "if( mm != null ) { " );
                          script.append( "mm.resetModelList(); " );
                          script.append( jspGetScriptModelList() );
                          script.append( "mm.populateModelList( \"" );
                          script.append( value );
                          script.append( "\" ); " );
                    script.append( "} " );
               script.append( "} " );
            }
            else if(  (  isValueChanged() || needsRendered )
                     &&  getAppType() !=  BIMViewer.TYPE_LOOKUP )    // popup doesn't
need to be bound to a value
            {
```

```
            script.append( "if( isLoaded ) { " );
            script.append( "frame.select( \"" );
            script.append( value );
            script.append( "\" ); " );
            script.append( "} " );
            setValueChanged( false );
         }
      } // close if( isControlVisible() )

      if( needsRendered  && getAppType() != TYPE_LOOKUP )
      {
         // Stack<AppInstance> appStack = _wcs.getAppStack(); Might sometime fixe
return from app
         script.append( " rehideObjs();" );
      }
      script.append( "} catch( e ) { console.log( e ); }" );

      return script.toString();
   }

   public String scriptResize()
   {
      StringBuffer resize = new StringBuffer();
      resize.append( "" );
      if( !isControlVisible() )
      {
         resize.append( "if( isLoaded ) { frame.resize( \"-1\" ); }" );
         return resize.toString();
      }

      int opt;
      if( getAppType() ==  BIMViewer.TYPE_LOOKUP  )
      {
         resize.append( "if( isLoaded ) { frame.resizeDlg( \"" );
         opt = jspGetRezieDlgOption();
      }
      else
      {
         resize.append( "if( isLoaded ) { frame.resize( \"" );
         opt = jspGetRezieOption();
      }

      resize.append( opt );
      resize.append( "\" ); } " );
      return resize.toString();
   }

   /**
    * Generate JavaScript to add all the models in _currentModelList
    * to the ComboBox specified by listCtrlId
    * @param listCtrlId
    * @return JavaScript fragment
    */
   public String jspGetScriptModelList()
   {
      StringBuffer script = new StringBuffer();
```

```
        for( int i = 0; i < _currentModelList.size(); i++ )
        {
           BIMModelSpec modelSpec = _currentModelList.get( i );

           script.append( "mm.addModel(" );
           script.append( modelSpec.getModelId() );
           script.append( ", \"" );
           script.append( modelSpec.getLocation() );
           script.append( "\", \"" );
           script.append( modelSpec.getBinding() );
           script.append( "\", \"" );
           if( modelSpec.getTitle() != null && modelSpec.getTitle().length() > 0  )
           {
              script.append( modelSpec.getTitle() );
           }
           else if( modelSpec.getDescription() != null &&
modelSpec.getDescription().length() > 0  )
           {
              script.append( modelSpec.getDescription() );
           }
           else
           {
              script.append( modelSpec.getLocationName() );
           }
           script.append( "\", \"" );
           script.append( modelSpec.getModelURL() );
           script.append( "\", \"" );
           script.append( modelSpec.getAttribClass() );
           script.append( "\", \"" );
           script.append( modelSpec.getAttribName() );
           script.append( "\", \"" );
           script.append( modelSpec.getParamClass() );
           script.append( "\", \"" );
           script.append( modelSpec.getParamName() );
           script.append( "\", \"" );
           if( _type == TYPE_ASSET )
           {
              script.append( modelSpec.getAssetView() );
           }
           else if( _type == TYPE_LOCATION )
           {
              script.append( modelSpec.getLocationView() );
           }
           else if( _type == TYPE_LOOKUP )
           {
              script.append( modelSpec.getLookupView() );
           }
           else if( _type == TYPE_WORKORDER )
           {
              script.append( modelSpec.getWorkOrderView() );
           }
           script.append( "\", \"" );
           script.append( modelSpec.getSelectionMode() );
           script.append( "\", \"" );
           script.append( modelSpec.getSiteId() );
           String mboKey = getMboKey();
           if( mboKey != null && mboKey.length() > 0 )
```

```
            {
               script.append( "\", \"" );
               script.append( mboKey );
            }
            script.append( "\" ); " );
        }

        _hasModelListChanged = false;
        String tmp = script.toString();
        return tmp;
    }

    public String scriptFooter()
    {
        StringBuffer footer = new StringBuffer();
        footer.append( "" );
        return footer.toString();
    }

    /**
     * Retrieves a stored resize option from the HTTP session and makes it
     * available to the .jsp.  This version is used when the viewer is
     * displayed on an application main tab
     * @return
     */
    public int jspGetRezieOption()
    {
        HttpServletRequest thisRequest = _wcs.getRequest();
        HttpSession session = thisRequest.getSession();
        Object o = session.getAttribute( ATTRIB_RESIZE );
        if( o == null || !(o instanceof String) )
        {
        return 0;
        }
        String option = (String)o;
        if( option.length() == 0 )
        {
        return 0;
        }
        return Integer.parseInt( option );
    }

    /**
     * Retrieves a stored resize option from the HTTP session and makes it
     * available to the .jsp.  This version is used when the viewer is
     * displayed on an dialog
     * @return
     */
    public int jspGetRezieDlgOption()
    {
        HttpServletRequest thisRequest = _wcs.getRequest();
        HttpSession session = thisRequest.getSession();
        Object o = session.getAttribute( ATTRIB_RESIZE_DLG );
        if( o == null || !(o instanceof String) )
        {
        return 0;
        }
```

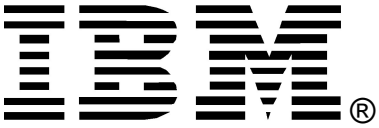
```
        String option = (String)o;
        if( option.length() == 0 )
        {
        return 0;
        }
        return Integer.parseInt( option );
    }



    /**
     * Provides an update to the control status line and resets
     * hasStatusUpdate to false
     * @return Text of status update
     */
    public String jspGetStatusUpdate()
    {
        if( _statusUpdate == null ) return "";         // Shouldm't be called in this
situation
        String status = _statusUpdate;
        _statusUpdate = null;
        return status;
    }


    /**
     * If the control is displayed on a tab of an application, it's really not on the
tab,
     * it at the bottom of the page below the tab set.  This prevents reloading of the
     * control and of control data as the use moved between tabs.  To cause it to
     * display correctly, the control class tracks which tab is visible and if the
     * control should be displayed.  If it should, top is set to a value on the tab
     * if it shouldn't, top is set to a large negative value
     * @return
     */
    public int jspGetViewerTop()
    {
        //Is the control on the screen or in "Storage"?
        if( isControlVisible() )
        {
           return _topOffset;
        }
        return -5000;
    }

    /**
     * Indicates if there is a status message to display on the control
     * Status line
     * @return True if there is a status message
     */
    public boolean jspHasStatusUpdate()
    {
        return _statusUpdate != null;
    }
```



IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PAPER “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes may be made periodically to the information herein; these changes may be incorporated in subsequent versions of the paper. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this paper at any time without notice.

Any references in this document to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
4205 South Miami Boulevard
Research Triangle Park, NC 27709 U.S.A.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Autodesk, the Autodesk logo, BIM360, NavisWorks, Revit, are registered trademarks or trademarks of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and/or other countries. All other brand names, product names, or trademarks belong to their respective holders. Autodesk reserves the right to alter product and services offerings, and specifications and pricing at any time without notice, and is not responsible for typographical or graphical errors that may appear in this document. © [2016] Autodesk, Inc. All rights reserved.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Notices

Apache POI
Copyright 2009 The Apache Software Foundation
This product includes software developed by The Apache Software Foundation (http://www.apache.org/).
This product contains the DOM4J library (http://www.dom4j.org). Copyright 2001-2005 (C) MetaStuff, Ltd. All Rights Reserved.
This product contains parts that were originally based on software from BEA. Copyright (c) 2000-2003, BEA Systems, <http://www.bea.com/>.
This product contains W3C XML Schema documents. Copyright 2001-2003 (c) World Wide Web Consortium (Massachusetts Institute of Technology, European
Research Consortium for Informatics and Mathematics, Keio University)
This product contains the Piccolo XML Parser for Java (http://piccolo.sourceforge.net/). Copyright 2002 Yuval Oren.