Infor M3 H5 Development Guide

Version 10.2.2.0
Published

Publication Information

Release: Infor M3 H5 10.2.2.0

Publication date: October 23, 2014

Contents

About this guide

The M3 H5 Development Guide provides conceptual and how-to information for developing, deploying, and adding scripts.

Intended audience

This guide is for developers who want to develop, deploy, and create scripts for M3 H5.

Related documents

You can find the documents in the product documentation section of the Infor Xtreme Support portal, as described in "Contacting Infor" on page 5.

M3 UI Adapter Installation Guide for LifeCycle Manager 9.x

M3 UI Adapter Installation Guide for LifeCycle Manager 10.x

Infor M3 H5 User and Administration Guide

Contacting Infor

If you have questions about Infor products, go to the Infor Xtreme Support portal at www.infor.com/inforxtreme.

If we update this document after the product release, we will post the new version on this Web site. We recommend that you check this Web site periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1 M3 H5 scripts overview

1

This document contains information about using scripting tools.

Scripting overview

You can use scripts that are connected to the form to extend the functionality of M3 forms in H5. The scripts have access to the contents of the form and can modify existing content or add new content to the form.

Scripts are developed using the jQuery language. jQuery is a fast, small, and feature-rich JavaScript library that makes tasks, such as HTML document traversal and manipulation, event handling, animation, and Ajax, much simpler with an easy-to-use API that works across a multitude of browsers.

The scripts will have access to public classes exposed by the M3 H5 framework.

Resources

jQuery API Documentation can be accessed here:

`http://api.jquery.com/`

Creating a script in M3 H5

You can use an external text editor to create a script because the scripts to be created are only client-side scripts.

Important: If there is a mismatch between the script class and the script file name, the script will not work when it is deployed to the server.

Contents of a script file

A script file must follow certain rules to be used by the H5 framework. If the script does not follow these rules, it will not be executed:

- The script class name must match the script file name, excluding the file extension.
- The script must have a public function called Init with a specific signature.

Example script

```
var Test = new function(){
    this.Init = function(scriptArgs){
         var controller = scriptArgs.controller;
         var args = scriptArgs.args;
         var element = scriptArgs.elem;
         var debug = scriptArgs.debug;

         /* Write code here*/
    }
}
```

Init method

The script must have a public function called Init with this signature:

```
this.Init = function(scriptArgs){
    /* Write code here*/
}
```

The scriptArgs parameter of the Init method contains JSON object that can be used to access elements or objects from the framework, such as the element, controller, debug, and args.

Element parameter

The element parameter is the control to which the script is connected. This parameter can be null if the script is not connected to a specific element which is allowed. The element parameter can be a TextBox, ComboBox, CheckBox, and so on, depending on which element the script was connected to. For M3 forms scripts, this is always null as the script is always attached to the form. The element parameter can be accessed like this:

```
this.Init = function(scriptArgs){
     var element = scriptArgs.elem;
}
```

Args parameter

The args parameter contains any arguments passed to the script as a string. Most scripts do not require arguments, so this parameter is typically null or blank. The args parameter can be

accessed like this:

```
this.Init = function(scriptArgs){
      var args = scriptArgs.args;
}
```

Controller parameter

The controller parameter is an object that implements the InstanceController. The controller is used to access the content of an M3 form. The methods and properties that are available in the InstanceController are described later in this document. The controller parameter can be accessed like this:

```
this.Init = function(scriptArgs){
      var controller = scriptArgs.controller;
}
```

Debug parameter

The debug parameter is an instance of the class ScriptDebugConsole that can be used for writing trace messages. The methods that are available in the ScriptDebugConsole interface are described later in this document. The debug parameter can be accessed like this:

```
this.Init = function(scriptArgs){
      var debug = scriptArgs.debug;
}
```

Chapter 2 Developing scripts

2

Mforms panel layout

The form that shows the content of an M3 program is located between the command bar and the status bar. This is the form that can have its content modified by scripts. The layout of the contents of a form is displayed based on the computed positioning of each element passed from an M3 program.

Adding content to a form

Additional content can be added to the panel by calling the Add method of the RenderEngine instance.The RenderEngine instance can be accessed using the InstanceController object, thus giving access to the elements available in one form.

Example

```
var controller = scriptArgs.controller;
var content = controller.RenderEngine.Content;
var btn1_Elem = new ButtonElement();
btn1_Elem.Value = "Test Button";

var btn1 = ControlFactory.CreateButton(btn1_Elem);
btn1.attr("id", "testBtn_id");

var pos = new PositionElement();
pos.Width = "auto";
pos.Top = "20";
pos.Left = "300";
btn1.Position = pos;

content.Add(btn1);
```

Element layout

The standard content is positioned on the form using the Top, Left, Width, and Height properties of the PositionElement class. When you add custom elements to a form, the elements should always be positioned using these properties.

Example

```
var btn1_Elem = new ButtonElement();
btn1_Elem.Value = "Show User Details";

var btn1 = ControlFactory.CreateButton(btn1_Elem);
btn1.attr("id", "showDetailsBtn");

var pos = new PositionElement();
pos.Width = "auto";
pos.Top = "3";
pos.Left = "12";
btn1.Position = pos;
```

Events

Because M3 H5 is a web-based application, script developers can utilize the default events of the browser for the elements. For Button Elements, the commonly used event is the Click Event, which can be used to perform additional validation and to properly handle the data to be submitted or passed.

Events must also be properly handled. This means that some events may need to be unsubscribed or turned off after usage to prevent memory leaks or the calling of different instances of the same script at the same time. Data that is used inside the event must also be passed properly to avoid having anonymous functions, which also causes memory leak.

Example

```
var $host = scriptArgs.controller.ParentWindow;
var $el = scriptArgs.elem;
var $debug = scriptArgs.debug;

var txtBox = ScriptUtil.FindChild($host, $el.Name);
txtBox.on("change", function(){
     $debug.WriteLine($(this).val());
});
```

Chapter 3 Public APIs

3

API overview

There are many public classes, methods, and properties in M3 H5 but only a limited number of these are allowed for use by scripts. Many classes are public for technical reasons but should only be used internally by the M3 H5 framework.

In scripts, use only public classes that are documented. This ensures that scripts will not break when updated versions of M3 H5 are installed. The methods and properties below are documented using the jQuery signatures from the M5 H5 source code. There are examples on how to use them in the Scripts syntax. The examples are not complete and are mainly used to show the syntax.

Note: Only those parts of the classes and methods that are allowed to be used from scripts are documented.

InstanceController

Each running M3 program has one controller object. The controller provides access to the content of a panel through the RenderEngine property and makes it possible to trigger the function keys and list options from script code.

The PressKey and ListOption methods do not validate input. Use these methods with caution. The BE program could get invalid data if these methods are used incorrectly. They can, however, be used for safe keys, such as F5 or ENTER, or for basic list options that are always known to be available.

```
/// <summary>
/// Returns the current RenderEngine instance
/// </summary>
InstanceController.RenderEngine

/// <summary>
/// Returns the raw XML response of the M3 programs
/// </summary>
InstanceController.Response.RawContent

/// <summary>
/// Executes a key request.
/// </summary>
/// <param name="key">The key to press, F1-F24 or ENTER</param>
InstanceController.PressKey(key)

/// <summary>
/// Executes a list option request.
/// </summary>
/// <param name="option">The option to execute, 1-99.</param>
InstanceController.ListOption(option)
```

Example

```
Example
var controller = scriptArgs.controller;
controller.PressKey("F5");
controller.ListOption("5");
```

RenderEngine

The RenderEngine class generates the controls on a panel and can be used to update or add content to a panel from scripts.

```
/// <summary>
/// Returns all the elements in a M3 panel
/// </summary>
RenderEngine.Content.Children

/// <summary>
/// Adds an element in the content area of the M3 Panel
/// </summary>
/// <param name="element">The element to be added</param>
RenderEngine.Content.Add(element)

/// <summary>
/// Shows a message in the status bar or in a dialog depending on the
/// user settings
/// </summary>
/// <param name="msg">The message to be displayed</param>
RenderEngine.ShowMessage(msg)
```

Example

```
var controller = scriptArgs.controller;
controller.RenderEngine.ShowMessage("The order number must be entered.");
```

ListControl

The ListControl class is a wrapper object for the list, and provides access to the actual ListView control and other data related to the list.

```
/// <summary>
/// Gets the ListView control for the list.
/// </summary>
ListControl.ListView()

/// <summary>
/// Gets the list of the column headers.
/// </summary>
ListControl.Headers()

/// <summary>
/// Gets a list of the column names.
/// </summary>
ListControl.Columns()
```

Example

```
var debug = scriptArgs.debug;
var list = ListControl.ListView();
debug.WriteLine(list.getColumns());

var headers = ListControl.Headers();
for(var i=0; i<headers.length;i++){
     debug.WriteLine(headers[i]);
}

var columns = ListControl.Columns();
for(var i=0; i<columns.length;i++){
     debug.WriteLine(columns[i]);
}
```

ScriptDebugConsole

The ScriptDebugConsole class can be used to write debug trace messages during the development of a script.

To launch the built-in debug console of a browser, such as Chrome and Internet Explorer, open the browser and press F12. A debug window is displayed. Click the Console tab to see the debug messages.

```
/// <summary>
/// Writes a string and a newline character to the browser console.
/// </summary>
/// <param name="text">The string to display.</param>
ScriptDebugConsole.WriteLine(text)

/// <summary>
/// Clears all text written to the console.
/// </summary>
ScriptDebugConsole.Clear()
```

Example

```
var controller = scriptArgs.controller;
var debug = scriptArgs.debug;

debug.Clear();
var content = controller.RenderEngine.Content;
for(var i=0; i<content.Children.length; i++) {
      debug.WrtieLine(content.Children[i].Name);
}
```

ScriptUtil

The ScriptUtil class contains utility methods for use by scripts.

```
/// <summary>
/// Finds an element in a panel.
/// </summary>
/// <param name="parent">The parent panel of the element to find.</param>
/// <param name="name">The name of the element to find.</param>
/// <returns>A child element or null if the element cannot be
/// found.</returns>
ScriptUtil.FindChild(parent, "WWPLEA");

/// <summary>
/// Launches a program, file or url. The default handler of the operating
/// is used to find the program for files or url's.
/// </summary>
/// <param name="fileName">A program name, file name or url.</param>
ScriptUtil.Launch (task);
```

Example

```
var element = ScriptUtil.FindChild(content, "WWPLEA");
if(element != null) {
       debug.WriteLine(element.Name);
}

ScriptUtil.Launch("OIS100");
ScriptUtil.Launch("mforms://MMS100");
ScriptUtil.Launch("http://www.infor.com");
ScriptUtil.Launch("net extension manager");
```

SessionCache

The SessionCache class is a helper class for scripts that can be used to cache items in the Mforms session. Make sure to use unique key names to avoid conflicts with other scripts when you add content to the cache.

```
/// <summary>
/// Adds value to the cache. If a value with the same key exists
/// it will be overwritten.
/// </summary>
/// <param name="key">The value key.</param>
/// <param name="value">The value to add.</param>
SessionCache.Add(key, val)

/// <summary>
/// Gets a value from the cache.
/// </summary>
/// <param name="key">The value key.</param>
/// <returns>The cached object or null if not found.</returns>
SessionCache.Get(key)

/// <summary>
/// Checks if a key exists in the cache
/// </summary>
/// <param name="key">The key to check.</param>
/// <returns>True if the key exists.</returns>
SessionCache.ContainsKey(key)

/// <summary>
/// Removes a value from the cache.
/// </summary>
/// <param name="key">The value key.</param>
/// <returns>True if the key existed.</returns>
SessionCache.Remove(key)
```

Example

```
SessionCache.Add("001", "123");
SessionCache.Add("002", "abcdef");
SessionCache.Add("003", "sample script");
var oBtn = new ButtonElement;
SessionCache.Add("004", oBtn);
$debug.WriteLine(SessionCache.SessionMap);

var isRemoved = SessionCache.Remove("003");
var isKeyExisting = SessionCache.ContainsKey("005");
$debug.WriteLine("Is Key '003' removed from the map? " + isRemoved
          + "\n" + "Is Key '005' existing in the map? " + isKeyExisting);
```

InstanceCache

The InstanceCache class is a helper class for scripts that can be used to cache items for a specific MForms instance. Make sure to use unique key names to avoid conflicts with other scripts when you add content to the cache.

```
/// <summary>
/// Adds value to the cache. If a value with the same key exists it will be
/// overwritten.
/// </summary>
/// <param name="controller">The controller for the instance.</param>
/// <param name="key">The value key.</param>
/// <param name="value">The value to add.</param>
InstanceCache.Add(controller, key, value)

/// <summary>
/// Gets a value from the cache.
/// </summary>
/// <param name="controller">The controller for the instance.</param>
/// <param name="key">The value key.</param>
/// <returns>The cached object or null if not found.</returns>
InstanceCache.Get(controller, key);

/// <summary>
/// Checks if a key exists in the cache
/// </summary>
/// <param name="controller">The controller for the instance.</param>
/// <param name="key">The key to check.</param>
/// <returns>True if the key exists.</returns>
InstanceCache.ContainsKey(controller, key)

/// <summary>
/// Removes a value from the cache.
/// </summary>
/// <param name="controller">The controller for the instance.</param>
/// <param name="key">The value key.</param>
/// <returns>True if the key existed.</returns>
InstanceCache.Remove(controller, key)
```

Example

```
var counter = 0;
var controller = scriptArgs.controller;
var key = "TestCounter";
if(InstanceCache.ContainsKey(controller, key)){
   counter = InstanceCache.Get(controller, key);
}
counter++;
debug.WriteLine("Script execution counter = " + counter);
InstanceCache.Add(controller, key, counter);
```

ConfirmDialog

The ConfirmDialog class contains a method for showing the kinds of dialogs that conform to the design system of M3 H5.

The ShowMessageDialog method has a parameter, a JSON object, which consists of options that can be used to define what type of Message Dialog to be displayed.

```
/// <summary>
/// Displays a message dialog.
/// </summary>
/// <param name="opts">A JSON object that contains the options that is available
/// for configuring the Message Dialog to be shown</param>
/// opts.dialogType = determines what type of dialog to be displayed. The value of
///   opts.dialogType can be "Question", "Information", "Warning", "Error",
///   "Success"
/// opts.header = the dialog header
/// opts.message = more detailed message to be displayed in the dialog
/// opts.id = only required for Warning and Question Dialog
/// opts.withCancelButton = optional property that flags if a cancel button should
///   be displayed or not
/// opts.isCancelDefault = optional property that flags if the Cancel Button is the
///   default button or not

ConfirmDialog.ShowMessageDialog(opts)
```

Example

```
var headerMsg = "Test Header", msg = "This is a Sample Message Dialog";
var opts = {    dialogType: "Information",
     header: headerMsg,
     message: msg,
     id: "testScript" }
ConfirmDialog.ShowMessageDialog(opts);
```

Chapter 4 Deploying scripts

4

Deploying a customized script in M3 H5

When a script has been developed and tested, it must be deployed to the M3 H5 Grid server in order to be available to other users. The scripts folder for H5 should contain all the customized scripts.

The scripts folder should be located similar to this path:

\grid\<Grid_Service>\applications\M3_UI_Adapter\webapps\mne\scripts

Note: After deployment, scripts must be added to forms through Personalization before they can be activated or executed for any user. See Adding scripts to a form.

Chapter 5 Adding scripts to a form

5

Adding a personalized script to a form

Scripts can be added to an M3 H5 form as a personalization. A script personalization can be created by one user and then deployed to many users using global or role personalizations.

The Personalized Scripts dialog can be used to connect scripts to elements on a form.

This topic describes how to create a script and a script shortcut for easy access. The Personalized Scripts dialog can be used to connect scripts to elements on a form.

1. Select Tools > Personalization > Scripts.
2. Select an element where the script should be connected.
3. Specify the name of the script and optional arguments.
4. Click Save.

If no other steps are taken, the script becomes active only for the user who created the personalization. Scripts can be associated with roles or assigned to run globally.

Adding a script shortcut

Scripts can also be connected to shortcuts. The Shortcuts dialog can be opened using the icon in the Shortcuts Panel, at the right part of a form panel.

1. Open the Shortcuts dialog.
2. Expand the Advanced area.
3. Specify the text to be displayed in the tool box, including the name of the script and optional arguments.
4. Click Add then Save.

Chapter 6 Script examples

6

This chapter contains examples of scripts. The scripts are complete and can be copied from this document to a new JavaScript file and deployed to M3 H5.

ShowXml.js

This example displays the raw XML content of an Mform from the View Definitions file. The ShowXML.js retrieves the XML file, and converts text symbols to HTML codes to properly display the text symbols in a Message Dialog.

```
var ShowXml = new function(){
      var CHAR_MAP = {
            '<': '&lt;',
            '>': '&gt;',
            '&': '&amp;',
            '"': '&quot;',
            "'": '&#39;',
            '!': '&#33;',
            '[': '&#91;',
            ']': '&#93;'
      };

      this.Init = function(scriptArgs){
            var controller =  scriptArgs.controller;
            var xml = controller.Response.RawContent;
            var me = this;

            var strXml = me.EscapeChar(xml);
            ConfirmDialog.ShowMessageDialog({
                  dialogType: "Information",
                  header : "XML Response",
                  message: strXml
            });
      }

        this.EscapeChar = function(str){
            return str.replace(/[<>&"'!]/g, function (ch) {
                  return CHAR_MAP[ch];
            });
      }
}
```

ScriptTester.js

This example shows how to access the elements and arguments passed through the parameter when attaching the script using the Personalized Scripts dialog.

```
var ScriptTester = new function(){
     var message;
     this.Init = function(scriptArgs){
          var debug = scriptArgs.debug;
          var element = scriptArgs.elem;
          var args = scriptArgs.args;

          debug.WriteLine("Script Initializing...");

          if(element != null){
               debug.WriteLine("Connected element" + element.Name);
          }

          message = "";

          if (element) {
               message = message + "Connected element: " + element.Name + "\n";
          } else {
               message = message + "No element connected.\n";
          }

          if (args != null) {
               message = message + "Arguments: " + args;
          }

          this.ShowMessage(message);
     }

     this.ShowMessage = function(message) {
          ConfirmDialog.ShowMessageDialog({
               dialogType: "Information",
               header : "Script Tester",
               message: message
          });
     }
}
```

FieldHelpBtn.js

This example shows how to display the Field Help document of a chosen element. This script must only be attached to a Button Element to trigger the opening of the Field Help on its click event. Specify the Field Help ID of the element to be shown in the Arguments text box when adding the script using the Personalized Script Dialog.

```
var FieldHelpBtn = new function(){
     this.Init= function(scriptArgs){
          var element = scriptArgs.elem;
          var controller = scriptArgs.controller;
          var fieldhelp = scriptArgs.args;
          var response = controller.Response;
          var $host = $(".lawsonHost:visible");

          var $elem = ScriptUtil.FindChild($host, element.Name);
          $elem.click({_response:response, _controller:controller, _$host:$host,
_fieldhelp:fieldhelp}, function(e){
               var _cont = e.data._controller, _resp = e.data._response,
               _$host = e.data._$host, _hlp = e.data._fieldhelp;

               _cont.RenderEngine.OpenFieldHelp(_resp, _$host, _cont, _hlp);
          });
     }
}
```

crs020_PreviewListHeader.js

This example is a program-specific script. It is also a script that is loaded from the MNEAI element of an Mform panel. This script searches for two specific elements that contain the headers and the row content to be displayed in a table. When this script is loaded, in CRS020/F, a table with one row of data is displayed.

```
var crs020_PreviewListHeader = new function(){
     this.Init = function(scriptArgs){
          var controller = scriptArgs.controller;
          var content = controller.RenderEngine.Content;

          var WWSFLL = this.FindElement(content, "WWSFLL");
          var WWSFHL = this.FindElement(content, "WWSFHL");

          var tmpStr = WWSFHL.Value.split(' ');
          var colVal = "", ctr = 0;
          var ListElem = new ListElement();
          ListElem.Position = new PositionElement();
          ListElem.Position.Width = 100;
          ListElem.Position.Height = 50;

          for(var i=0; i<tmpStr.length;i++ ){
               if(tmpStr[i] != ""){
                    var cols = {};
                    if(/^[A-Z]/.test(tmpStr[i]) == true){
                         colVal = tmpStr[i] + " ";
                         ctr += 1;
                         cols = {
                                   id:  "C"+ctr,
                                   name: colVal,
                                   field: colVal

                         };
                         ListElem.Columns.push(cols);
                    }else{
                         colVal += tmpStr[i] + " ";
```

```
                                   cols = {
                                  id:  "C"+ctr,
                                  name: colVal,
                                  field: colVal
                             };
                             ListElem.Columns.splice(ListElem.Columns.length-1, 1,
cols);
                        }
                   }
              }

              rows = WWSFLL.Value.split(' ');
              var rowItem = [];
              for(var j=0; j<rows.length; j++){
                   if(rows[j].trim() != ""){
                        rowItem.push(rows[j]);
                   }
              }

              var data = [];
              var row = {};
              row["id"] = "R1";

              for(var j=0; j<ListElem.Columns.length; j++){
                   row[ListElem.Columns[j]["field"]] = rowItem[j];
              }
              data[0] = row;

              var id = "crs020FGrid";
              var $list = $("<div>",{
                   "class":"inforDataGrid",
                   "id": id,
                   "width": "auto",
                   "height": "500px"
              });
              $list.Position = new PositionElement();
              $list.Position.Width = "100%";
              $list.Position.Top = "250";
              $list.Position.Left = "0";

              content.Add($list);

              var jqueryListId = "#" + id;
              var options = Configuration.Current.ListConfig('id', ListElem.Columns,
data);
              options["forceFitColumns"] = true;
              options["autoHeight"] = true;

              var grid = $(jqueryListId).inforDataGrid(options);
              grid.render();
         }

         this.FindElement = function(content, element){
              for(var i=0; i<content.Children.length; i++) {
                   if(content.Children[i].Name == element) {
                        return content.Children[i];
                   }
              }
              return null;
         }
}
```

ShowOnMap.js

This example is also a program-specific script. This can be used in OIS002/F. This script adds a custom button element in the panel that opens a location in Google Maps depending on the values that were specified in three specific fields.

```
var ShowOnMap = new function(){
     this.Init = function(scriptArgs){
          var controller =  scriptArgs.controller;
          var content = controller.RenderEngine.Content;
          var me = this;

          var showMapBtn_elem = new ButtonElement();
          showMapBtn_elem.Value = "Show Map";

          var showMapBtn = ControlFactory.CreateButton(showMapBtn_elem);
          showMapBtn.attr("id", "showMapBtn");
          showMapBtn.attr("style", "margin-left:5px");

          var $host = $(".visible-tab-host");
          var geoX = ScriptUtil.FindChild($host, "WFGEOX");

          geoX.after(showMapBtn);
          showMapBtn.on("click", me.OnClickShowMap);
     }
      this.OnClickShowMap = function(e){
          var $host = $(".visible-tab-host");
          var lat = ScriptUtil.FindChild($host, "WFGEOX");
          var lng = ScriptUtil.FindChild($host, "WFGEOY");
          var zoom = ScriptUtil.FindChild($host, "WFGEOZ");

          var x=0, y=0, z=0, loc = "", url = "";
          if(lat && lng){
               x = lat.val();
               y = lng.val();
               z = zoom.val();

               if(x.indexOf("-") > -1){
                    x = x.replace("-", "");
                    x = "-" + x;
               }

               if(y.indexOf("-") > -1){
                    y = y.replace("-", "");
                    y = "-" + y;
               }

               loc = x + "+" + y;
               if(z != ""){
                    url = "https://maps.google.com/maps?z="+ z +"&t=m&q=loc:" + loc
+ "&output=embed";
               }else{
                    url = "https://maps.google.com/maps?z=15&t=m&q=loc:" + loc +
"&output=embed";
               }
          }

          ScriptUtil.Launch(url);
     }
}
```

ShowUserDetails.js

This example adds a custom Button Element on the Panel. When the button is clicked, a message shows M3-related data from the M3 API about the logged on user.

```
var ShowUserDetails = new function(){
   this.Init = function(scriptArgs){
      var $host = scriptArgs.controller.ParentWindow;
      var $cont = scriptArgs.controller;
      var $args = scriptArgs.args;
      var $el = scriptArgs.elem;
      var $debug = scriptArgs.debug;
      var content = $cont.RenderEngine.Content;

      $debug.Clear();
      var btn1_Elem = new ButtonElement();
      btn1_Elem.Value = "Show User Details";

      var btn1 = ControlFactory.CreateButton(btn1_Elem);
      btn1.attr("id", "showDetailsBtn");

      var pos = new PositionElement();
      pos.Width = "auto";
      pos.Top = "3";
      pos.Left = "12";
      btn1.Position = pos;

      content.Add(btn1);
      btn1.click(function(){
         var headerMsg = "USER DETAILS";
         //userContext is a global variable that contains all user information from
the M3 API
         var msg = "User Id: " + userContext['USID'] + "<br />" +
            "Name: " + userContext['NAME'] + "<br />" +
            "Company: " + userContext['CONO'] + "<br />" +
            "Division: " + userContext['DIVI'] + "<br />" +
            "Language: " + userContext['LANC'] + "<br />" +
            "Date Format: " + userContext['DTFM'] + "<br />" +
            "Decimal Format: " + userContext['DCFM'] + "<br />" +
            "Time Zone: " + userContext['TIZO'] + "<br />" +
            "Facility: " + userContext['FACI'] + "<br />" +
            "Warehouse: " + userContext['WHLO'] + "<br />" +
            "Customer: " + userContext['CUNO'] + "<br />" +
            "Department: " + userContext['DEPT'] + "<br />" +
            "Company Name: " + userContext['TX40'] + "<br />" +
            "Division Name: " + userContext['CONM'] + "<br />" +
            "Menu Version: " + userContext['MNVR'] + "<br />" +
            "Default Menu: " + userContext['DFMN'] + "<br />" +

            "EMAL: " + userContext['EMAL'] + "<br />" +
            "Company: " + userContext['Company'] + "<br />" +
            "Current Company: " + userContext['CurrentCompany'] + "<br />" +
            "Current Company Name: " + userContext['CurrentCompanyName'] + "<br />"
+
            "Division: " + userContext['Division'] + "<br />" +
            "Current Division: " + userContext['CurrentDivision'] + "<br />" +
            "Current Division Name: " + userContext['CurrentDivisionName'] + "<br
/>" +
            "Language: " + userContext['Language'] + "<br />" +
            "Current Language: " + userContext['CurrentLanguage'];
```

```
      var opts = {
             dialogType: "Information",
             header: headerMsg,
             message: msg,
             id: "msgDetails",
             withCancelButton: true,
             isCancelDefault: false
          }
             ConfirmDialog.ShowMessageDialog(opts);
       });
   }
}
```