

Type of Submission: Article

Title: IBM Case Manager

Subtitle: Using scripts to customize Case Manager Client

Keywords: scripts, customizing, widgets

Author: Guo, Ming Liang

Job Title: Senior Software Engineer, Case Manager and Enterprise Content Management

Email: guoml@cn.ibm.com

Bio: Architect of the IBM Case Manager Client.

Company: IBM

Contributor: Gao, Ying Ming

Job Title: Software developer, IBM Case Manager Client

Email: guoyingm@cn.ibm.com

Bio: Developer of the IBM Case Manager Client action framework.

Company: IBM

Abstract: This article describes how to use scripts to customize your IBM Case Manager client application.

Introduction

Case Manager Client is a Dojo based Web client that runs inside of IBM Content Navigator. (IBM Case Manager V5.2 uses Dojo V1.8.3). Because customers frequently want to customize the user interface, Case Manager Client provides different levels of customization capabilities, from simple to advanced:

- Configure edit-settings to change widget behavior
- Wire events between widgets to trigger widget operations and so on
- Write simple scripts to achieve customized behavior
- Create custom widgets to achieve customized behavior.

To edit widget settings and event wiring, the widgets must already have the customization capabilities in place for the user to configure. Alternatively, using scripts to customize widgets and actions is much more flexible and enables users to achieve sophisticated customization at the source code level.

In this article, we focus on using scripts to customize your widgets and actions.

IBM Case Manager provides the two utilities to support the use of scripts:

- Script Adapter widget
 - You can drag this widget onto a page and wire it to receive a payload that runs a preconfigured script against the payload. The Script Adapter widget then sends the resulting payload to other widgets.
- Script Action
 - You can use this action to configure a script that is run when a user invokes the action from a toolbar or menu.

This article uses some example scenarios to show how you can use scripts to achieve simple customization requirements. The article also provides tips for debugging your scripts. article.

Examples of using scripts for customization

This article uses the following examples:

- Script Adapter 1: This example uses the Script Adapter widget to manipulate an event payload.
- Script Adapter 2: This examples uses the Script Adapter widget to construct an in-basket filter and send the filter to the In-basket widget.
- Script Action 3: This example creates a Script action to open an entry template dialog.

1) Script Adapter 1: Use the Script Adapter widget to manipulate event payload

Scenario:

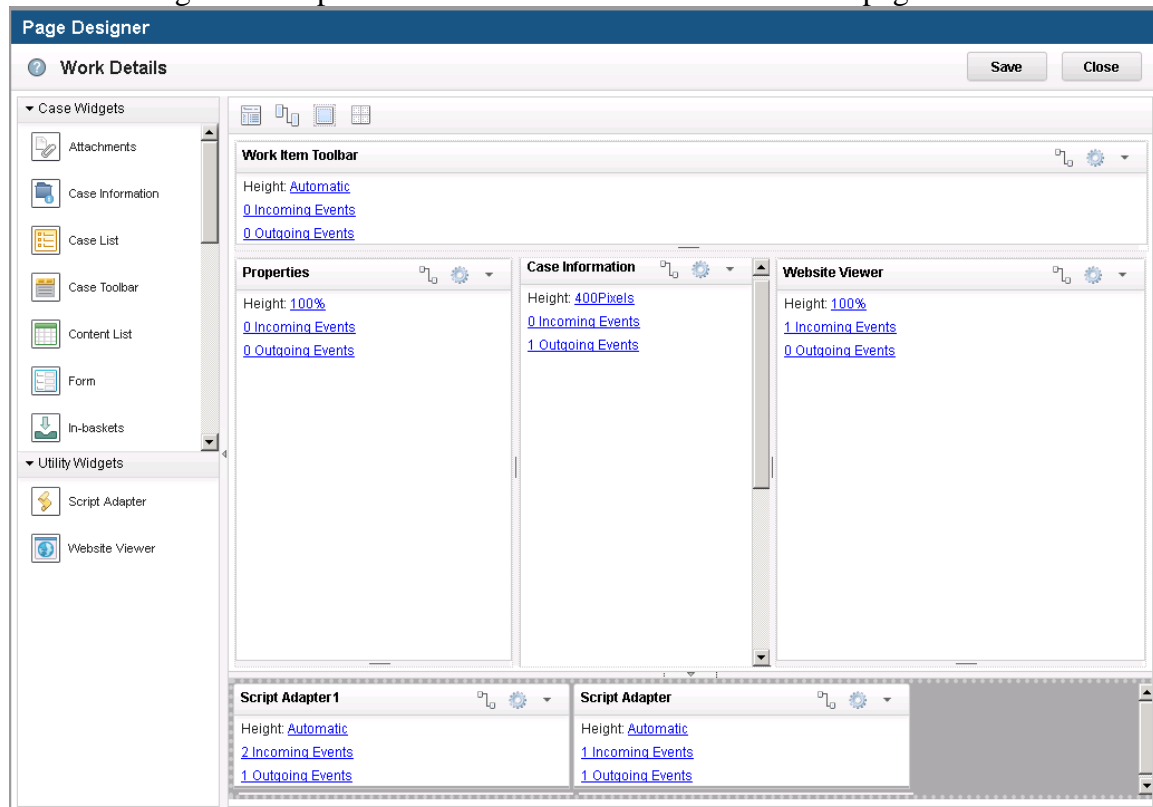
A work item has a stock code property. When a user processes the work item, he or she uses this code to check the stock information on a Yahoo website. Instead of copying the stock code and searching the website, we want to add a Website Viewer widget to the Work Details page in which the Yahoo stock page for the stock is displayed.

To achieve this task, we add a Script Adapter widget and a Website Viewer widget in Work Details page. The Script Adapter widget extracts the stock code from the work item and constructs the URL for the stock. The Script Adapter widget then sends it to the Website Viewer Widget.

Procedures

Step1: Setup the page

The following screen capture shows our customized Work Details page:



Step2: Setup the wiring for Script Adapter Widget

Because the Script Adapter widget only performs code logic, we configure it as 'Hidden' so that it does not show in the page at run time.

The following screen capture shows the wiring for the Script Adapter widget:

Wire Events

Widget: Script Adapter

Event Wiring | Event Broadcasting

Add wires to establish communication between widgets. An asterisk (*) identifies an event related to an action. [Learn More](#)

Incoming Events for the Script Adapter

Source widget: Page Container Outgoing event: Page Closed Incoming event: Receive event payload Add Wire

Source	Event	Target	Event
Page Container	Send work item	Script Adapter	Receive event payload

Outgoing Events for the Script Adapter

Outgoing event: Send event payload Target widget: Page Container Incoming event: Add case Add Wire

Source	Event	Target	Event
Script Adapter	Send event payload	Website Viewer	Display URL

OK Cancel

We defined two wirings for the Script Adapter widget:

- From Send Work Item event of the Page Container to the Receive event of the Script Adapter. This wiring enables the Script Adapter widget to extract the stock code information from the work item payload and construct the URL for the stock.
- From Send event of the Script Adapter to the Display URL event of the Website Viewer. This wiring enables the Website Viewer widget receives the URL and display it.

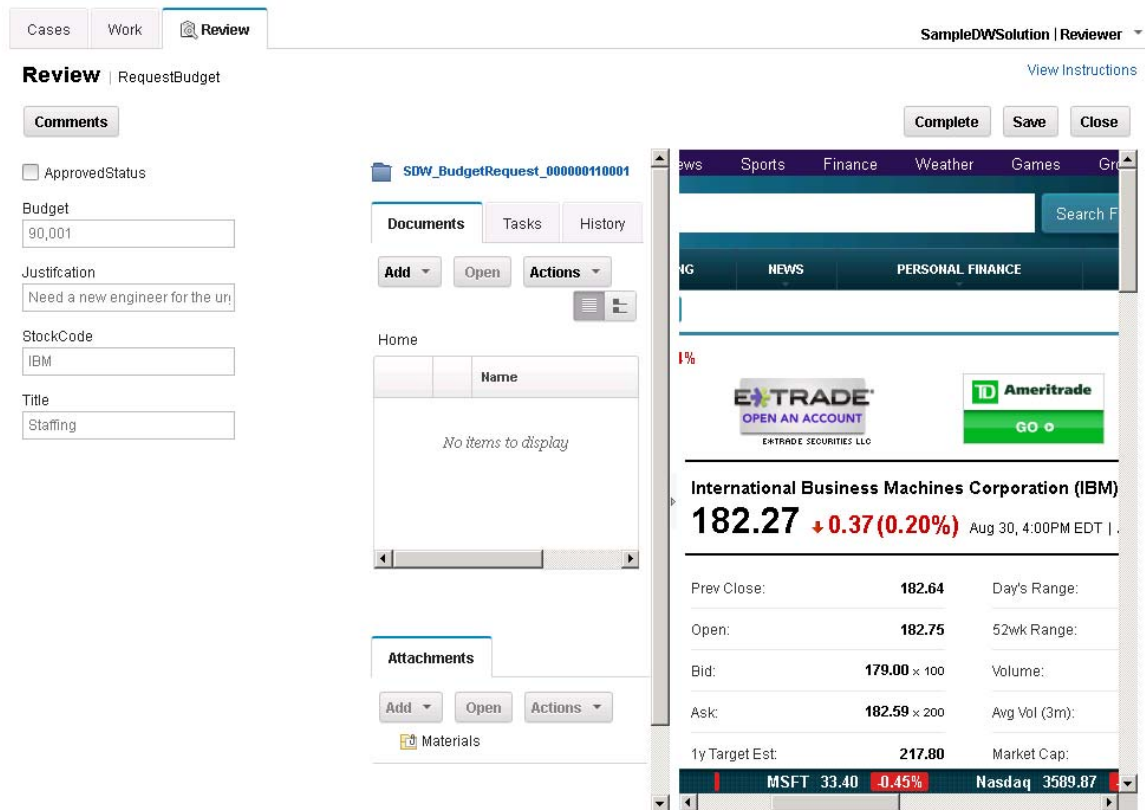
Step3: Configure the Script Adapter Code

To construct the URL payload, we need the following script in the Script Adapter widget:

```
var workItem = payload.workItemEditable.icmWorkItem.ecmWorkItem;  
var stockCode = workItem.attributes["SDW_StockCode"];  
var stockURL = "http://finance.yahoo.com/q?s=" + stockCode;  
return stockURL;
```

Result:

With this wiring and script, we see the page when we open the work item:



Summary:

With a simple script, we have accomplished some complex tasks. We called an external REST service to retrieve information from an external system. We then used that information to assist the review procedure, put a record in an external system, and so on.

2) Script Adapter 2: Use Script Adapter to construct an in-basket filter and send the filter to the In-basket widget

Scenario

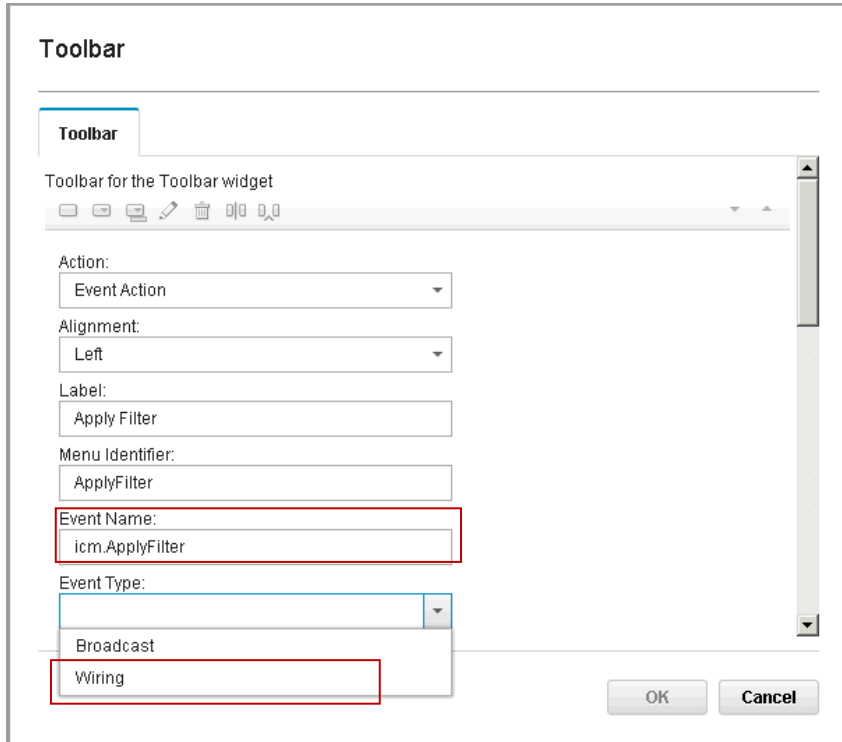
We need to specify a special filter to display the work items in the In-basket widget.

To achieve this task, we need the Script Adapter widget to construct the filter payload and send the payload to the In-basket widget. In addition, we need an Event Action to trigger the Script Adapter widget to publish the event.

Procedures:

Step 1: Add an Event Action to Toolbar Widget

First, we add an Event Action to the Toolbar Widget in the Work page as shown in the following screen capture. This Event Action is used to trigger the Script Adapter to publish events.



The screenshot shows the 'Toolbar' configuration dialog box. The 'Event Name' field is set to 'icm.ApplyFilter' and the 'Event Type' is set to 'Wiring'. Both fields are highlighted with red boxes. The 'Action' dropdown is set to 'Event Action', 'Alignment' is 'Left', 'Label' is 'Apply Filter', and 'Menu Identifier' is 'ApplyFilter'. The 'Event Type' dropdown is open, showing 'Broadcast' and 'Wiring' options, with 'Wiring' selected. The 'OK' and 'Cancel' buttons are at the bottom right.

For the Event Action, we must define the "Event Name" and "Event Type." The "Event Name" is used in wiring. Here we set the event name to "icm.ApplyFilter" and the event type to "Wiring."

Step 2: Configure the Script Adapter widget

We add the Script Adapter widget to the Work page and set it as "Hidden."

We then enter the following script in the Script Adapter widget to construct the in-basket filter:

```
var json = {
  "queueName": "SDW_Reviewer",
  "inbasketName": "Reviewer",
  "hideFilterUI": "true",
  "queryFilter": "(SDW_StockCode = :A) OR (SDW_Budget > :A)",
  "queryFields": [
    {
      "name": "SDW_StockCode",
      "type": "xs:string",
    }
  ]
}
```

```

        "value": "IBM"
    },
    {
        "name": "SDW_Budget",
        "type": "xs:integer",
        "value": 99
    }
],
"hideLockedByOther": "true"
/* true will hide the work items locked by other. */
};
var dynamicFilter = icm.model.InbasketDynamicFilter.fromJSON(json);
var dynamicFilters = [];
dynamicFilters.push(dynamicFilter);
var payload = {"dynamicFilters": dynamicFilters};
return payload;

```

This script uses the DynamicFilter class to construct a filter that filters stock code and budget. For the more information about constructing an in-basket filter, see the handleApplyFilter() method for the [icm.pgwidget.inbasket.Inbasket](#) class.

Step 3: Set up wirings between the Event Action, Script Adapter widget, and In-basket widget

For the wiring, the Script Adapter widget sits in between the Event Action and the In-basket widget as shown in the following screen capture:

Wire Events

Widget
Script Adapter

Event Wiring
Event Broadcasting

Add wires to establish communication between widgets. An asterisk (*) identifies an event related to an action. [Learn More](#)

Incoming Events for the Script Adapter

Source widget:
Toolbar

Outgoing event:
*icm.ApplyFilter

Incoming event:
Receive event payload

Add Wire

Source	Event	Target	Event
Toolbar	*icm.ApplyFilter	Script Adapter	Receive event payload

Outgoing Events for the Script Adapter

Outgoing event:
Send event payload

Target widget:
In-baskets

Incoming event:
Apply filter

Add Wire

Source	Event	Target	Event
Script Adapter	Send event payload	In-baskets	Apply filter

OK
Cancel

As shown, we set up two wirings:

- From icm.ApplyFilter event of the Toolbar widget to the Receive event payload event of the Script Adapter widget. This wiring is used primarily to trigger the Script Adapter widget to publish the payload
- From Apply filter event of the Script Adapter widget to the Send Event payload event of the In-basket widget. This wiring is used to send the in-basket filter to the In-basket widget to filter work items.

3) Script Action: Use a Script Action to open an entry template dialog

Scenario:

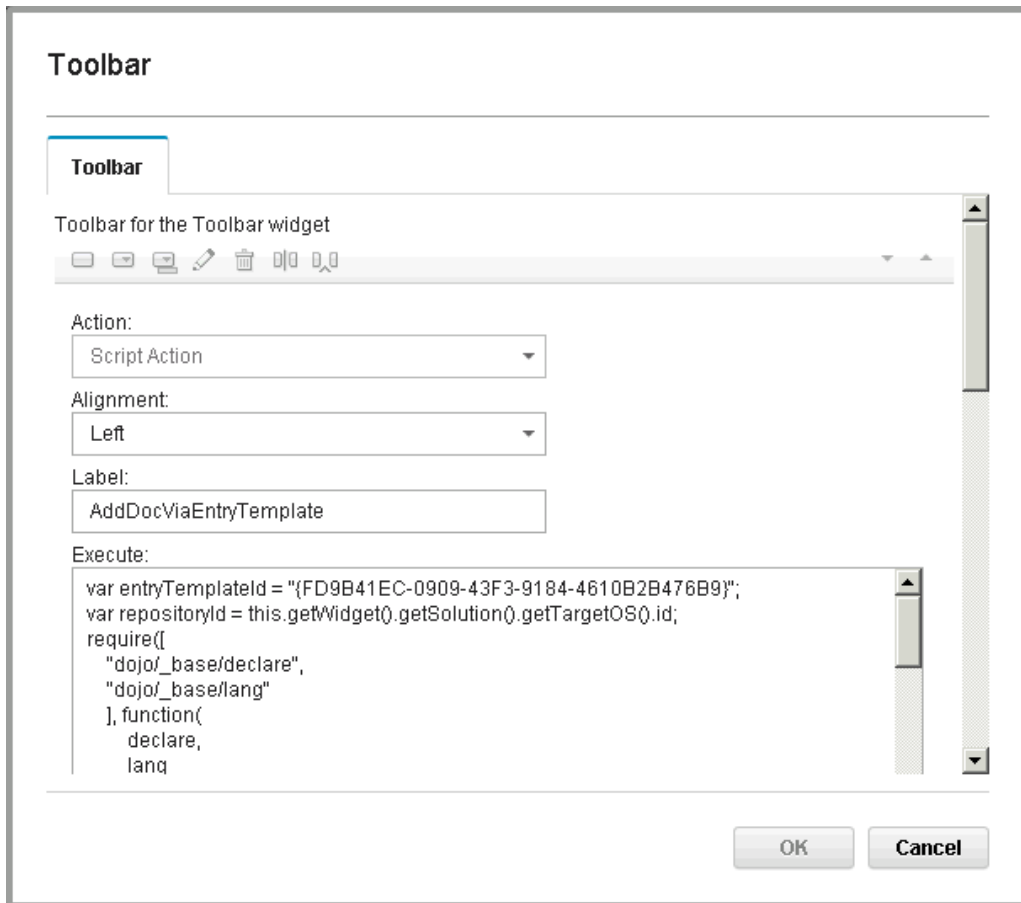
We want to use one entry template for creating a document when reviewing work items in In-basket widget. (Note that we must create and configure the entry template in Workplace XT. This procedure is not shown in this article.)

To achieve this task, we add a Script Action to the Toolbar widget in the Work page. The Script Action represents the button that is used to add a document by using the entry template.

Procedures:

Configuring the Script Action is fairly simple because it does not require wiring with other widgets. Instead, we just configure the Script Action in the Toolbar widget. The action is then rendered as a button in Toolbar widget and the user clicks this button to run the script.

The following screen capture shows how we add the Script Action in Toolbar widget in Work page:



We enter the following script for the Script Action. Actually, the script reuses the entry template dialog from IBM Content Navigator. So in theory, we could use this Script Action to accomplish a number of tasks.

```
var entryTemplateId = "{39F5922E-9ADE-49B7-A3E6-A95ED5478796}"; /*
Version Series ID of the entry template. You need to update it
according to the id of the entry template you are using in your
environment*/
var repositoryId = this.getWidget().getSolution().getTargetOS().id;
require([
  "dojo/_base/declare",
  "dojo/_base/lang"
], function(
  declare,
  lang
) {
  ecm.model.desktop.getActionsHandler(lang.hitch(this,
function(actionsHandler) {
  var repository =
ecm.model.desktop.getRepository(repositoryId);
  repository.retrieveItem(entryTemplateId, function(item)
{
  if (item && item.mimetype) {
    switch (item.mimetype) {
      case "application/x-filenet-
```

```
documententrytemplate":
    case "application/x-filenet-entrytemplate":
        actionsHandler.actionView(repository, [
            item
        ]);
        break;
    }
}, "EntryTemplate", "current", entryTemplateId);
});
```

After we redeploy the solution, we can use the AddDocViaEntryTemplate button to add documents by using entry template:

The screenshot shows a web application interface with a 'Work' tab. A modal dialog titled 'Add Document by Using Entry Template' is open. The dialog has three sections: General, Properties, and Security. In the General section, the 'Entry template' is set to 'docEntry', 'Save in' is 'StoredSearch', 'What do you want to save?' is 'Local document', and 'File name' is empty with a 'Browse...' button. A 'Major version' checkbox is checked. In the Properties section, 'Class' is 'Document' and 'Document Title' is 'Doc via Entry Template'. In the Security section, 'Share with' is 'Specific users and groups' with a 'Select...' button. The background shows a table with 'Application Request' items.

Debugging tips for using scripts

Compared to writing custom widgets or actions, it is relatively more difficult to debug a script for the Script Adapter widget or a Script Action. The code in the script is evaluated and interpreted by the JavaScript engine at run time.

In general, you can make debugging easier by following these guidelines:

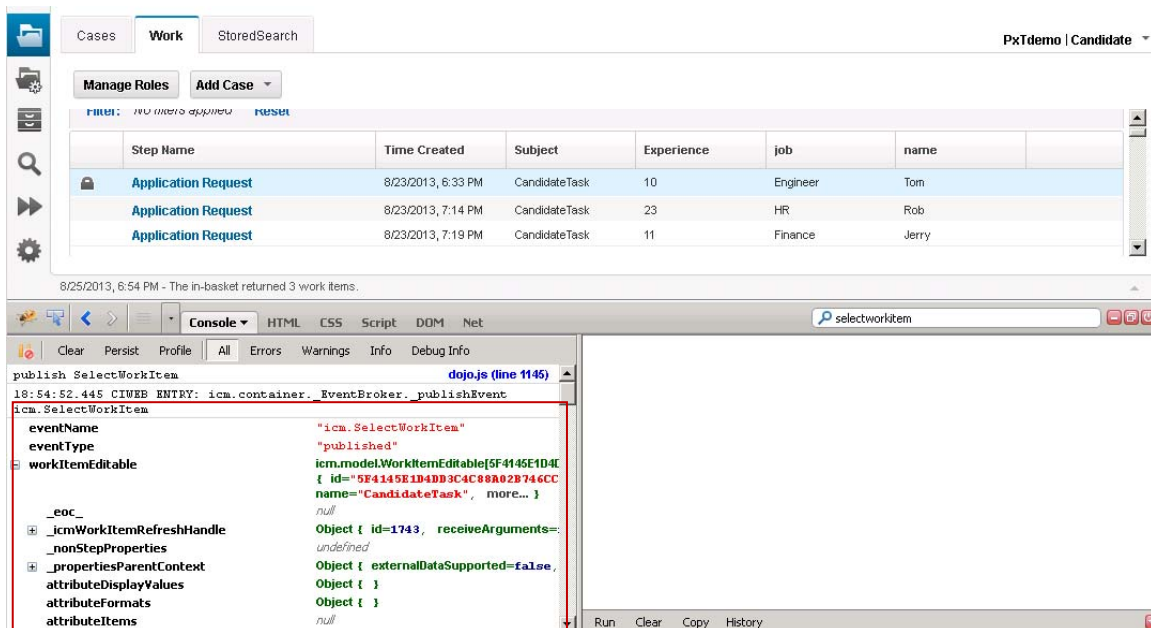
- Use Firefox and Firebug as your main development environment. After everything works well in Firefox, you can test your script in Internet Explorer.
- Write simple scripts that are easy to understand and debug. If your script is complex, consider instead creating a custom widget or action to perform the task.

- Test your script outside of the Script Adapter widget or Script Action before you input the script in Case Manager Builder.
- Ensure that you understand the payload format before you start writing the scripts. Use the Firebug console to inspect the payload format.

Following are more detailed tips:

Tip1: Inspect the payload in Firebug.

By default, the Script Adapter widget dumps the payload it received by using the `console.dir()` method. If you want more detailed information about a payload, you can wire the event to a Script Adapter widget and inspect the payload in the Firebug console. The following screen capture illustrates this procedure for the "icm.SelectWorkItem" payload:



Tip 2: Understand how scripts are being run

The Script Adapter widget and the Script Action use a similar mechanism to run scripts. For example, the script that is entered in the Script Adapter widget is converted to a function and attached to the widget. This behavior enables the script to access all the properties and methods of the Script Adapter widget including `this.solution`, `this.role`, `this.publishEvent()`, `this.broadcastEvent()`, and so on. Similarly, the script for a Script Action is converted to a method that is invoked inside of the `execute` method. This behavior enables the script in a Script Action to call typical script methods.

The following screen capture shows the code that converts the scripts as *theRunner* function on the Script Adapter widget itself:

```
59 },
60
61 /**
62  * @private
63  */
64 setupRunner: function(scripts) {
65     if(scripts==null)
66         scripts='';
67     var jsonStr = dojo.trim(scripts);
68     if (jsonStr.length > 0) {
69         this.theRunner = dojo.hitch(this, dojo.fromJson('\"runScript: function(payload) { \" + jsonStr + \"}\" ).runScript);
70     }
71     else{
72         this.theRunner = dojo.hitch(this, dojo.fromJson('\"runScript: function(payload) { return payload; }\" ).runScript);
73     }
74 }
```

The following screen capture shows the code that invokes *theRunner* function with the payload that was passed in.

```
260 }
261
262 try{
263     var newPayload = this.theRunner(payload);
264     var event = new Array();
265     event["name"] = "icm.SendEventPayload";
266     event["type"] = "published";
267     event["payload"] = newPayload;
268     this.setSentEvent(event);
269 }
270 catch (ex) {
271     var newPayload = this.resourceBundle.CAN_NOT_INTERPRET_EVENT;
272     errorEncountered = true;
273 }
274
275 }
```

Tip 3: Step by step debugging

Sometimes, it is helpful to do a step-by-step debugging for the scripts in the Script Adapter widget or a Script Action. To do this, you must set breakpoints at *theRunner* function in Firebug. When the breakpoint is reached, you then must step in until you reach the scripts that you added. The following screen captures show this procedure:

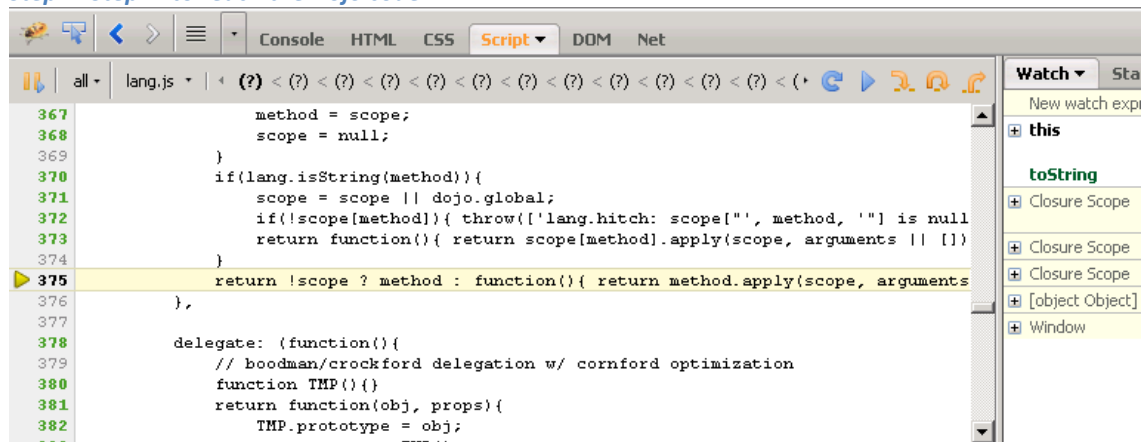
Step 1: Reach the breakpoint of theRunner function

```
255 var scriptText = this.widgetProperties.payload;
256 try {
257     this.setupRunner(scriptText.replace(/#_##/g, ""));
258 } catch (e) {
259     result = e.message;
260 }
261
262 try{
263     var newPayload = this.theRunner(payload);
264     var event = new Array();
265     event["name"] = "icm.SendEventPayload";
266     event["type"] = "published";
267     event["payload"] = newPayload;
268     this.setSentEvent(event);
269 }
270 catch (ex) {
271     var newPayload = this.resourceBundle.CAN_NOT_INTERPRET_EVENT;
272     errorEncountered = true;
273 }
274
275 }
```

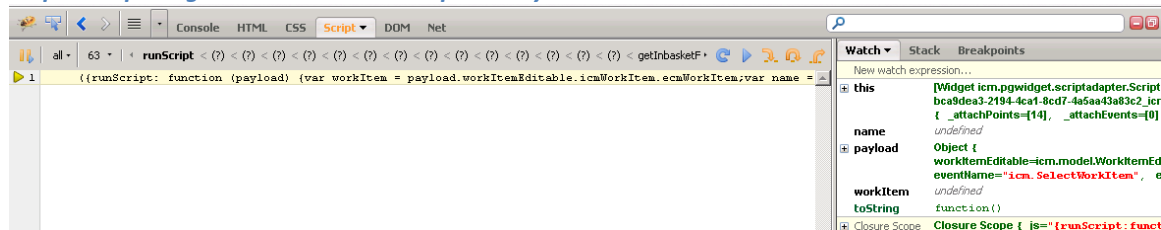
Watch panel:

- this
- event
- newPayload
- payload
- scriptText
- showScriptText
- toString
- Closure Scope
- Closure Scope
- [Object Object]
- Window

Step 2: Step in to reach the Dojo code



Step 3: Step in again to reach the scripts that you added:



Summary

In this article, we used several examples to illustrate how to use the scripting capabilities to customize Case Manager Client. And we also introduced some debugging tips when developing with scripts. These examples will help you build up the basic skills for customizing Case Manager Client and will also help you troubleshoot problems when debugging scripts.

Resources

- IBM Case Manager 5.2 Information Center - <http://pic.dhe.ibm.com/infocenter/casemgmt/v5r2m0/index.jsp>
- IBM Content Navigator API reference - <http://pic.dhe.ibm.com/infocenter/p8docs/v5r2m0/topic/com.ibm.developingeuc.doc/eucrf000.htm>
- Dojo Toolkit - <http://dojotoolkit.org/>



[sampleDWSolution exported file](#)

SampleDWSolution_solution.zip