



---

# IBM Case Manager 5.2 Enablement

## Using the Model API and REST API call in custom widgets for IBM Case Manager V5.2

By: Mathew Sprehn and Johnson Liu

With special thanks to

Amisha Parikh  
Tim Morgan  
Anu Shenoy  
Lauren Mayes

## Table of Contents

|  |    |
|--|----|
| 1. Overview.....   | 3  |
| 2. Making REST Calls in IBM Case Manager V5.2 Custom Widgets.....      | 3  |
| 5. Using the JavaScript API Reference Documentation.....               | 6  |
| 6. Making Model API calls in IBM Case Manager 5.2.....                 | 9  |
| 7. Passing the Model Object to other IBM Case Manager Widgets.....     | 10 |
| 8. Advantages of Model API over Rest API in IBM Case Manager V5.2..... | 11 |

# 1. Overview

This article shows how to make REST API calls in a custom widget on IBM Case Manager 5.2 and to convert a REST API call to Model API call.

The IBM Case Manager JavaScript APIs, which include the Model API, extend the IBM Content Navigator APIs. You can use the IBM Case Manager JavaScript APIs to extend IBM Content Navigator API functionality and to call the Case Manager API. For detailed information about the Model API calls, see [IBM Case Manager JavaScript API Reference](#) in the IBM Case Manager Version 5.2 Information Center. For information about the IBM Content Navigator API, see [IBM Content Navigator JavaScript API Reference](#) in the IBM FileNet P8 Version 5.2 Information Center. This article is a continuation of Tim Morgan's article Mapping Case REST API calls to Model API calls when converting custom widgets to ICM 5.2, which can be found [here](#).

We will briefly discuss using the REST APIs versus using the Model API in this article. In IBM Case Manager V5.1.1, you used the REST APIs in custom widget development. In IBM Case Manager V5.2, you are encouraged to use the new Model API in custom widgets. However, it is still possible to develop using the REST API in V5.2. This article discusses the advantages of using Model API calls over the REST API.

**Important:** You can continue to use the CMIS REST API in your existing custom widget code. However, if your custom widget interacts with the IBM Case Manager V5.2 widgets, you must convert the returned responses to JavaScript model objects.

The examples in this article are based on a custom search widget that was developed for IBM Case Manager V5.2. For information about this widget, see the developerWorks article [Converting a Custom Search Widget from ICM 5.1.1 to ICM 5.2](#).

You can download the code for the custom search widget from the following link:  
[https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=e8206aad-10e2-4c49-b00c-fee572815374#fullpageWidgetId=Wf2c4e43b120c\\_4ac7\\_80ae\\_2695b8e6d46d&file=e3ff1d40-31e0-469a-9ecf-aa7a03e73a46](https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=e8206aad-10e2-4c49-b00c-fee572815374#fullpageWidgetId=Wf2c4e43b120c_4ac7_80ae_2695b8e6d46d&file=e3ff1d40-31e0-469a-9ecf-aa7a03e73a46)

## 2. Making REST Calls in IBM Case Manager V5.2 Custom Widgets

To demonstrate using REST calls in an IBM Case Manager 5.2 widget, we add a REST call to the existing custom search widget. This call generalizes the widget by replacing the hard-coded list of caseType properties with a REST call that enumerates the properties.

We make this change in the CustomSearchWidget.js file, which is the main widget JavaScript file. We will change this part of the CustomSearchWidget.js file later on in section 6 to use Model API. The following code shows the REST call:

```
domReady(function() {  
    var htmlBuilder = "";  
    var ajaxResponse;  
    var xhrArgs = {  
        url: ".../.../CaseManager/CASEREST/v1/casetype/CCDMT_ManageDisputeItemTargetObjectStore=CMTOS",
```

```

handleAs: "json",no
load: function(serverResponse)
{
    console.log("success!");
    ajaxResponse = serverResponse;
    for (i = 0; i < ajaxResponse.Properties.length; i++)
    {
        if(ajaxResponse.Properties[i].SymbolicName.indexOf("_") != -1)
        {
            if (ajaxResponse.Properties[i].PropertyType == "boolean")
            {
                htmlBuilder += "<h3>" + ajaxResponse.Properties[i].DisplayName +
                "(" + ajaxResponse.Properties[i].PropertyType + ")</h3>";
                htmlBuilder += "<form>";
                htmlBuilder += 'true<input type="radio" size="25" id="' +
                ajaxResponse.Properties[i].DisplayName + '-true' value =
                "true"><br/>';
                htmlBuilder += 'false<input type="radio" size="25" id="' +
                ajaxResponse.Properties[i].DisplayName + '-false' value =
                "false"><br/>';
                htmlBuilder += "</form>";
            }
            else
            {
                htmlBuilder += "<h3>" + ajaxResponse.Properties[i].DisplayName + "(" +
                ajaxResponse.Properties[i].PropertyType + ")</h3>"
                htmlBuilder += '<input type="text" size="25" id="' +
                ajaxResponse.Properties[i].DisplayName + '">';
            }
        }
    }
    dojo.byId("Wrapper").innerHTML = htmlBuilder;
}
error:function(error){console.log("error!" + error);}
}

```

In this code:

1. The green text indicates the URL that is used to call the REST API. In this example, the URL is making a call to obtain information about the case type CCDMT\_ManageDisputeItem located at the object store CMTOS.

Note that the browser requires you to make AJAX calls only from the domain of origin. Therefore, if this script resided on ibm.com, it could not make a call to anywhere outside ibm.com.

2. The domReady() function is a Dojo function that ensures that the code inside the function does not run until the DOM object has been completely loaded.
3. The code writes "success" to the web browser console for debugging purposes. This message is displayed when the REST call executes successfully.
4. The remaining code enumerates through the REST response to gather the information that is needed for the custom search widget to work correctly and to build an HTML form based on the case properties.
5. The first if statement ( ajaxResponse.Properties[i].SymbolicName.indexOf("\_") != -1) filters out system properties from the array of case properties. System properties (such as the current solution id) are properties that all cases have, and are not very useful to search on. System properties use the format "SystemProperty" while case properties use the format "SOLUTIONPREFIX\_PropertyName". By filtering out properties that do not contain an underscore, we also filter out all system properties. Note that more specificity might be required or desired, but for the purposes of this example simply filtering on the presence of the underscore character is sufficient.

Note that to get the attribute names from the case solution, you must access the objects differently than from Model API. For example, you use the following call to get the attribute name from a REST API response:

```
ajaxResponse.Properties[i].DisplayName
```

There are other ways to retrieve the case attributes similar to the preceding REST API call. To use REST API in a custom widget in IBM Case Manager 5.2, you cannot use the REST API to create an object and then pass the response to a function that expects a model object without doing some post processing to rename the attributes to match the Model object. One way to create a model object is by using static methods such as the `fromJSON()` method that is defined on the `odel` class. Alternatively you can create model objects by using the model class constructor and setting properties from the JSON object on the model object. If available, you can use the `fromJSON()` method to convert an object that is returned from a REST call to a Model object.

First, we will look at how we finish up our REST widget. While we have successfully made a REST call, we still must pass the returned information to the rest of the widget. Most importantly, we need to build our search query, which involves creating a query to the Content Platform Engine. More specifics on implementing case search can be found in `icm.util.SearchPayload` in the IBM Case Manager JavaScript API Reference. We will use the following code to build the query:

```
for (i = 0; i < ajaxResponse.Properties.length; i++)
{
    var temp;
    var temp2;
    if (ajaxResponse.Properties[i].SymbolicName.indexOf("_") != -1)
    {
        if (ajaxResponse.Properties[i].PropertyType == "boolean")
        {
            temp = dojo.byId(ajaxResponse.Properties[i].DisplayName + '-true');
            temp2 = dojo.byId(ajaxResponse.Properties[i].DisplayName + '-false');
            if (temp.checked)
            {
                params.ceQuery += "t.[" + ajaxResponse.Properties[i].SymbolicName + "] = " +
                domAttr.get(temp, "value") + " AND ";
                empty = false;
            }
            else if (temp2.checked)
            {
                params.ceQuery += "t.[" + ajaxResponse.Properties[i].SymbolicName + "] = " +
                domAttr.get(temp2, "value") + " AND ";
                empty = false;
            }
        }
        else
        {
            temp = dojo.byId(ajaxResponse.Properties[i].DisplayName);
            if (domAttr.get(temp, "value") != '')
            {
                if (ajaxResponse.Properties[i].PropertyType == "integer" ||
                ajaxResponse.Properties[i].PropertyType == "float" ||
                ajaxResponse.Properties[i].PropertyType == "id")
                {
                    params.ceQuery += "t.[" +
                    ajaxResponse.Properties[i].SymbolicName + "]
                    = " + domAttr.get(temp, "value") + " AND ";
                    empty = false;
                }
                else if (ajaxResponse.Properties[i].PropertyType ==
                "string")
                {
                    params.ceQuery += "t.[" +
                    ajaxResponse.Properties[i].SymbolicName + "]
                    LIKE '%" + domAttr.get(temp, "value") + "%'
```



### 3. Using the JavaScript API Reference Documentation

The following table below shows the organization of the IBM Case Manager JavaScript classes and packages. You can use this information to help when you look up the functionalities to add.

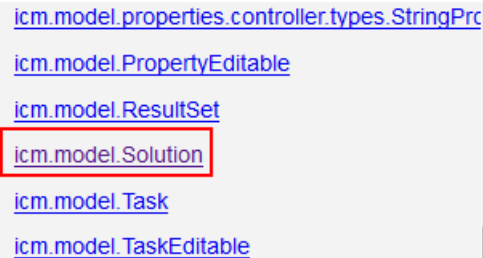
| Package  | Example classes   | When would I use?   | Example usage   |
|--|---|---|---|
| <b>icm.model</b><br>Objects in the Case Manager system and scratchpad data.<br>No UI components here | Case, CaseComment, CaseRelationship, HistoryEvent, PropertyController, Solution, Task | When creating custom widgets and actions, to access ICM data, used in conjunction with ICN model. Access data when scripting events.              | Get the case identifier for the case displayed on case detail page<br><br>Get editable model object<br><br>Update a case property |
| <b>icm.action</b><br>Out-of-the-box actions provided by ICM  | AddCustomTask, SendLink, ShowLink, AddDocumentfromLocal                               | When building a custom action, if I wanted to wrap an existing ICM action or create a new custom action   | Create a custom action for case document that publishes an event  |
| <b>icm.base</b><br>Base classes used to creation custom page widgets and actions                     | BasePageWidget, BaseActionContext, Constants, WidgetAttributes, _EventStub            | When creating a custom widget or action, use these base classes to provide the infrastructure and fill in the implementation with custom behavior | Create a custom search widget, create an action to display data from an external system.  |

| Package  | Example classes   | When would I use?  | Example usage   |
|--|---|--|---|
| <b>icm.dialog</b><br>ICM provided dialogs                      | AddCommentDialog, AddTaskDialog, DynamicTaskEditorDialog                          | When I want my custom widget or action to display a dialog   | Display dialog to add a comment when user clicks on an icon in a content list   |
| <b>icm.pgwidget</b><br>Classes that represent the page widgets | Attachment, CaseForm, CaseInfo, CaseList, CaseSearch, CaseToolbar, CaseVisualizer | When I want to create a custom widget that includes a page widget, or when I want to extend a page widget with additional behavior                           | Extend the in-basket widget to change font to red for overdue items<br><br>Add new tab in Case Info                           |
| <b>icm.util</b><br>Utility classes                             | Coordination, SearchPayload, InbasketFilterUtil                                   | When you want to participate in processing a page, such as dispatching a work item or saving a case.<br>When building nested CE queries for flexible search. | Custom widget that saves data to an external system can hook into the dispatch of a work item                                 |
| <b>icm.widget.menu</b><br>Toolbar and pop-up menu classes      | ContextualMenu, Menu, MenuManager, Toolbar  | When I want to include a context menu or toolbar in my custom widget, and use the Page Layout Designer to enable configuring the menus                       | Implement retrieval of data from an external system using a dialog that is configured as a toolbar action in my custom widget |

The JavaScript API Reference document shows you the actions and functions that you can use within your custom widgets.

In our scenario, we walk through how to use this information.

To get the properties for the case solution, we call the `icm.model.Solution.retrieveAttributeDefinitions` method. To get information about this method, we scroll the navigation pane in the JavaScript API Reference to find the `icm.model.Solution` class:



```
icm.model.properties.controller.types.StringProc  
icm.model.PropertyEditable  
icm.model.ResultSet  
icm.model.Solution  
icm.model.Task  
icm.model.TaskEditable
```

Next, we find the `retrieveAttributeDefinitions` function that allows us to retrieve the attributes. This function, which will return the information needed for the custom search widget, requires a callback function:

---

#### `retrieveAttributeDefinitions (callback)`

Retrieves attribute definitions that correspond to all attributes of any case types of the solution. This information is cached so that the callback function can be called right away.

##### **Parameters:**

###### **callback**

a function called with an array of `icm.model.AttributeDefinition` objects



Next, we want to find a Model API function to do the case search in the custom search widget. We look for a class called `icm.util.SearchPayload`. This class enables us to create the search payload that is used between the IBM Case Manager Search widget and the Case List widget. By following the details in the functions `getSearchPayload()` and `setModel(model)`, we are able to set the `CEQuery` variable and set the `SearchPayload` to broadcast to the `CaseList` widget.

### Field Summary

| Field Attributes | Field Name and Description  |
|------------------|---|
|                  | <a href="#"><u>ceQuery</u></a><br>Content Engine SQL for searching cases.                     |
|                  | <a href="#"><u>searchTemplate</u></a><br>An instance of <code>ecm.model.SearchTemplate</code> |

### Method Summary

| Method Attributes | Method Name and Description   |
|-------------------|---|
|                   | <a href="#"><u>constructor</u></a> ()   |
|                   | <a href="#"><u>getSearchPayload</u></a> ()<br>Get the search payload that will be sent out.                       |
|                   | <a href="#"><u>setModel</u></a> (model)<br>Set the schema and data structures associated with the current search. |

## 4. Making Model API calls in IBM Case Manager 5.2

Now, we can place the custom search widget in any solution and use the widget to find attributes and make them searchable. We now replace the code that we inserted in sections 2 and 3 with the following code. The green colored text indicates the Model API call that we are using to retrieve the solution property attributes.

```

var modelResponse
ContentPaneHtml = "";
widget.solution.retrieveAttributeDefinitions(function(theResponse){
  modelResponse = theResponse
  for (i=0; i< theResponse.length; i++)
  {
    if (modelResponse[i].id.indexOf("_") != -1)
    //checks if item is not a system attribute. System attributes do not have the
    "SOLUTIONPREFIX_NAME" naming convention
    {
      if (modelResponse[i].dataType == "xs:boolean")
      {
        ContentPaneHtml += "<h3>" + modelResponse[i].name + "(" + modelResponse[i]
        ].dataType + ")</h3>";
        ContentPaneHtml += "<form>";
        ContentPaneHtml += 'true<input type="radio" size="25" id="' +
        modelResponse[i].name + '-true" value = "true"><br/>';
        ContentPaneHtml += 'false<input type="radio" size="25" id="' +
        modelResponse[i].name + '-false" value = "false"><br/>';
        ContentPaneHtml += "</form>";
      }
    }
  }
}

```

```

        else
        {
            ContentPaneHtml += "<h3>" + modelResponse[i].name + "(" + modelResponse[i]
            ].dataType + "</h3>"
            ContentPaneHtml += '<input type="text" size="25" id="' + modelResponse[i]
            ].name + '">';
        }
    }
    dojo.byId("Wrapper").innerHTML = ContentPaneHtml;
});

```

Note that we no longer have to execute an AJAX call. Instead, we access the solution in which the custom search widget is used and enumerate the case type properties. In addition, we do not need to convert the object to a Model Object to pass it to an IBM Case Manager widget that expects a Model object.

When we add the custom search widget to a copied version of Credit Card Dispute Management, we can see these properties are dynamically populated in the widget.

The screenshot shows the IBM Case Manager interface. The top navigation bar includes the IBM logo and user information (msprehn). The main content area is titled "Credit\_Card\_Dispute\_Management2 | Customer Service Representative". On the left, there is a search widget with a red border. This widget contains several input fields, each with a label and a data type in parentheses: "Account ID(xs:string)", "AssignedDate(xs:timestamp)", "Fraud Amount(xs:double)", "Fraud Type(xs:string)", "CustomerName(xs:string)", and "Date Case Closed(xs:timestamp)". To the right of the search widget is a large empty area with the text "No items to display".

## 5. Passing the Model Object to other IBM Case Manager Widgets

In IBM Case Manager, widgets often communicate with other widgets by sending or broadcasting events. While this mechanism is essentially the same in IBM Case Manager V5.2 as in earlier releases, the content of the payloads is dramatically different. Therefore, any existing code that sends or receives events in interactions with IBM Case Manager widgets requires modification. In addition, the package name of the events also changes from "com.ibm.ecmwidgets.acm" to "icm". For example, the custom search code uses a broadcast to send the desired search to the Case List widget for processing. The API call to effect the broadcast changes, the event name changes its prefix, and the type of the payload is now an object that references a Model object:

```
widget.onBroadcastEvent("icm.SearchCases", payload);
```

We must ensure that we pass the correct Model object that the IBM Case Manager V5.2 Case List widget expects with this broadcast event to avoid a runtime error.

## **6. Advantages of Model API over Rest API in IBM Case Manager V5.2**

Using the Model API instead of the REST API provides the following advantages:

- You can easily extend IBM Content Navigator and IBM Case Manager functionality when you develop custom widgets.
- The Model API clearly separates business logic from the user interface layer.
- Model objects that containing business or server side objects can be shared across widgets, including custom widgets.
- The CMIS REST API is still available in the IBM Content Navigator API.
- The JavaScript toolkit provides a more standard, object-oriented programming model.