

Using External Properties in IBM Case Manager v5.2.1

By: Johnson Liu, Brent Taylor

Special thanks to:

Manpreet Kahlon

Yajie Yang

Dave Perman

Frankie Mosher

1. Introduction

External properties are ad-hoc properties that are not defined within the properties of a case or a work item but that can be rendered within their associated views in IBM Case Manager v5.2.1 or later. External properties are defined, retrieved and persisted through an external source such as a servlet, JSON file, or data service. The binding of external properties is handled by the Properties area in the Model layer.

This developerWorks article aims to discuss how to define external properties using:

- The properties view designer within IBM Case Builder
- JavaScript in a Script Adapter widget for Properties widget in IBM Case Client

2. What is an external property?

An external property is similar to a case property or task property with the exception that an external property can be defined outside of IBM Case Manager and then bound to the Model API layer so that the data can be rendered in the Properties widget for example.

An external property is:

- defined in its own collection and is not included in the collection of case properties or task properties
- defined, retrieved and persisted through an external data source like a servlet, JSON file, or data service
- added to the Properties widget using script adapter widget

External property binding is handled by the Properties area in the Model layer. The general use cases for external property are the following:

- Most organizations maintain several “systems of record” data sources
- These data sources often contain the most current “single source of truth”
- Easy access to this information helps case workers complete their work
- It is often necessary, and efficient, to access this data live and not store a copy

The specific use cases for external property are the following:

- While working a claim for a customer, an agent opens the case details
- Along side the claim specific data, the agent sees the customer's contact information, pulled live from their client database
- A list of past transactions is also conveniently displayed, saving the agent from having to access another system

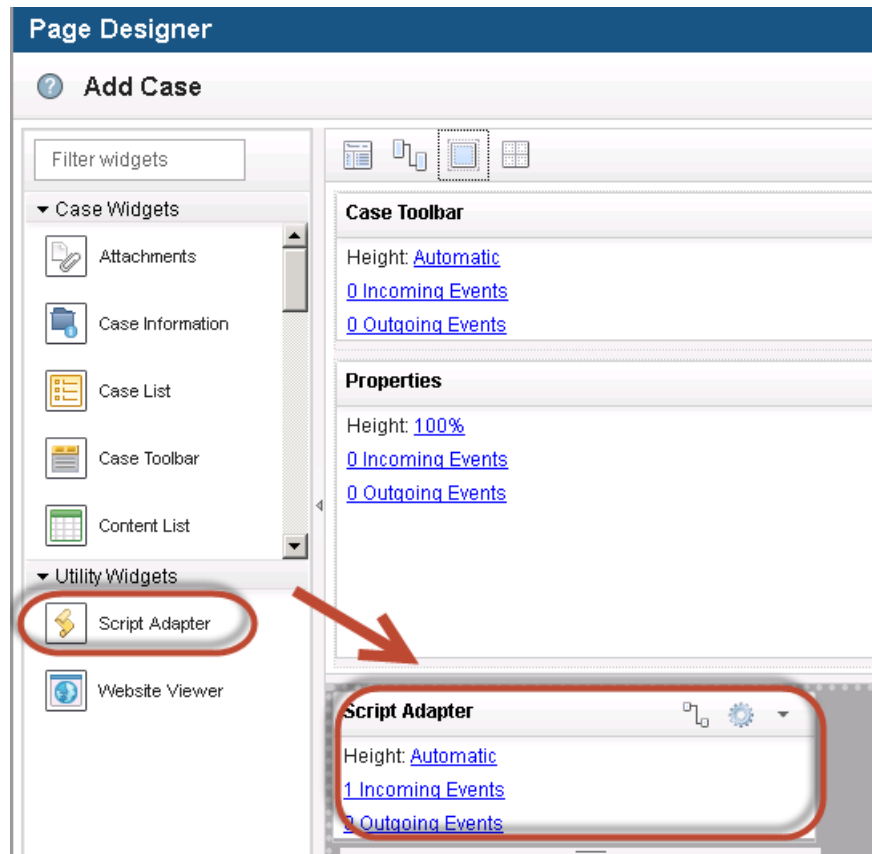
Some common approaches to define an external property outside of IBM Case Manager include:

- Java Servlet
- JSON file on mid-tier
- In-line in JavaScript
- and others

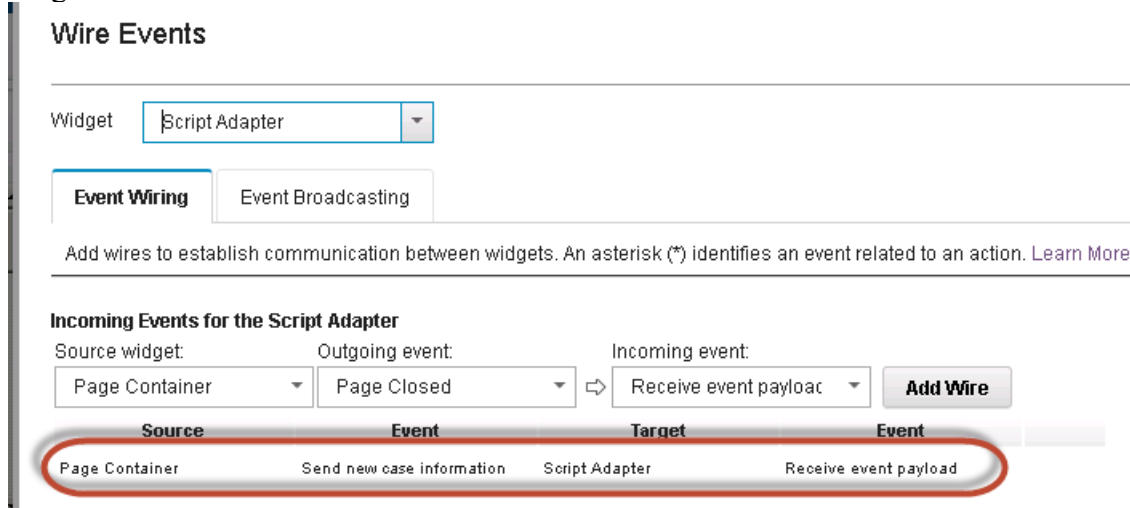
3. Hello World example

The following is a simple and quick way to define and bind an external property during run-time to demonstrate the extensibility, flexibility, and portability of external properties.

- Open Case Builder and on the Add Case or Case Details page, place a Script Adapter widget onto the page



- Click the Edit Wiring icon on the Script Adapter widget and wire send new case information widget event



- Click the Edit Settings icon on the Script Adapter widget and paste the following JavaScript into the window. The JavaScript here is defining an external property named PhoneNumber that we will mix in with our system-generated view that has other case properties on it.

```
require([
    "icm/model/properties/controller/ControllerManager",
    "icm/base/Constants"
], function(ControllerManager, Constants) {
    // Get the editable and coordination objects from the event payload.
    var coordination = payload.coordination;
    var editable = payload.caseEditable;
    var model;

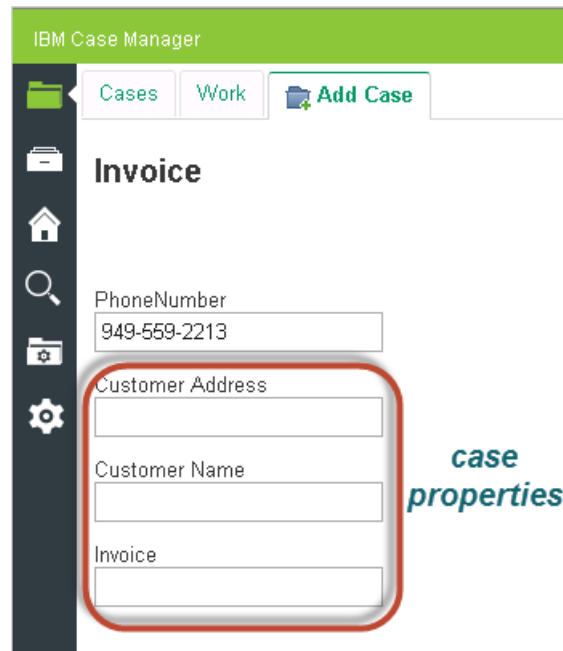
    // Participate in the BEFORELOADWIDGET topic to bind the external
    // properties into the controller.
    payload.coordination.participate(Constants.CoordTopic.BEFORELOADWIDGET,
        function(context, complete, abort) {
            model = {
                properties: {
                    "PhoneNumber": {
                        id: "PhoneNumber",
                        name: "Phone Number",
                        type: "string",
                        cardinality: "single",
                        value: "949-559-2213"
                    }
                }
            };
            var collectionController = ControllerManager.bind(editable);
            collectionController.bind("External", "External", model);
            complete();
        });

    // Participate in the AFTERLOADWIDGET topic to release the controller binding.
    payload.coordination.participate(Constants.CoordTopic.AFTERLOADWIDGET,
        function(context, complete, abort) {
            ControllerManager.unbind(editable);
            complete();
        });
});
```

It is important to note that all external property related binding typically should happen in the handler for the BEFORELOADWIDGET. This is also why we utilize specific events that occur before the Properties widget loads like “Send new case information” for the Add Cases page.

- Save the page and save and deploy your changes to the solution.

- Open the solution in Case Client and in our example, add a case where you will see the external property Phone Number with other case properties on the page.



4. Defining external properties using the Script Adapter widget

An external properties collection is defined in the model layer as a single object such as the following that provides a collection of property objects.

```
var model = {  
  properties: {  
    "name": {  
      id: "name",  
      type: "string",  
      value: "Rip Van Winkle"  
    },  
    "age": {  
      id: "age",  
      type: "integer",  
      value: 200  
    },  
  },  
}
```

The implementation for the definition, retrieval and persistence of external properties is application specific and must be provided via custom code. Typically, this custom code is implemented in the Script Adapter widget.

Your custom code can call the `PropertyCollectionController`'s `bind` method as shown below to include your external properties collection in the set of properties associated with the view. You may bind as many collections as you wish. A unique collection identifier must be specified for each.

```
controller.bind(collectionId, collectionName, model);
```

The `PropertyCollectionController` supports standard model signatures with which it can bind implicitly. In some cases, your application may need to support a non-standard model signature associated with your application. If so, you can provide an `Integration` object as part of the binding to instruct the controller how to interact with the associated model.

```
controller.bind(collectionId, collectionName, model, integration);
```

The `PropertyCollectionController` automatically updates the state of the model as changes occur within the view.

Setting up wiring for Script Adapter widget

It is important to specify an incoming event for the Script Adapter widget that is loaded before the actual widget loads on the page. For example, for the Add Cases page, we should wire the Script Adapter widget to the Page Container's send new case information event as shown in the screenshot below.

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: Send new case inform | Incoming event: Receive event payload | Add Wire

Source	Event	Target	Event
Page Container	Send new case information	Script Adapter	Receive event payload

Likewise, you can use the page container's send case information event for the case details page and the page container's send work item event for the work details page. These events will allow the Script Adapter widget to execute the JavaScript code so that the external properties are bound to the Properties widget before it is loaded.

Using external properties defined via Model object using Script Adapter widget

The example below illustrates the definition of a typical collection of external properties of various data types.

```
{
  properties: {
    "description": {
      id: "description",
      name: "Description",
      label: "Description",
      type: "string",
      cardinality: "single",
      value: "description here"
    },
    "price": {
      id: "price",
      name: "Price",
      label: "Price",
      type: "float",
      cardinality: "single",
      value: 22.2
    },
    "booleanTest": {
      id: "booleanTest",
      name: "booleanTest",
      label: "booleanTest",
      type: "boolean",
      cardinality: "single"
    },
    "datetimeTEST": {
      id: "datetimeTEST",
      name: "datetimeTEST",
      label: "datetimeTEST",
      type: "datetime",
      cardinality: "single"
    },
    "quantityINT": {
      id: "quantityINT",
      name: "quantityINT",
      label: "quantityINT",
      type: "integer",
      cardinality: "single"
    },
    "total": {
      id: "total",
      name: "Total",
      label: "Total",
      type: "float",
      cardinality: "single",
    },
    "MyMultiInteger": {
      id: "MyMultiInteger",
      type: "integer",
      cardinality: "multi",
      value: [1, 2, 3]
    },
    "multiCategory": {
      id: "multiCategory",
      name: "MultiCategory",
      label: "MultiCategory",
      type: "integer",
    }
  }
}
```

```
cardinality: "multi",
choices: [{
  label: "Small",
  value: 0
},
{
  label: "Large",
  value: 1
}
]
}
}
```

Here is a screenshot to show the properties that we defined in the JSON on the Add Case page:

The screenshot displays the 'Add Case' page in IBM Case Manager for a case type named 'testcasetype1'. The page features a sidebar with navigation icons and a main content area with several input fields. The fields are grouped into two categories:

- external properties:** This group includes fields for 'Description' (text input), 'MultiCategory' (dropdown menu showing 'No items to display'), 'MyMultiInteger' (dropdown menu showing 'No items to display'), and 'Price' (text input with value '22.2').
- case properties:** This group includes fields for 'str111' (text input), 'teststring' (text input), 'Total' (text input), 'booleanTest' (checkbox), 'datetimeTEST' (date and time picker showing '8/14/2014' and '12:00 AM'), and 'quantityINT' (text input).

Creating an Integration object

Creating a custom integration object is useful when you are working with a third party model object with a different signature than the default signature supported by the controller. Some examples include an incoming JSON object from a web service or a object from within your existing model layer. You can create a custom Integration object that instructs the controller how to map its attributes to fields of the custom object.

The following script demonstrates how to create the Integration object used for such a binding. The yellow highlighting below shows the parts that are important to note for custom integration configuration.

```
// Create the external properties model with the custom model signature.
var model = {
  props: {
    "PhoneNumber": {
      symbolicName: "PhoneNumber",
      name: "Phone Number",
      type: "string",
      multiValue: false,
      value: "949-559-2213"
    }
  }
};

// Create a custom integration object for the custom model signature.
var integration = new Integration();
integration.mergeConfiguration(basicIntegrationConfiguration);
integration.mergeConfiguration(customIntegrationConfiguration);

// Add a binding for the external properties to the controller.
collectionController.bind("External", "External", model, integration);
```

The script below illustrates the custom integration configuration object required for the custom integration in the script above. Typically, this configuration object is implemented in a separate dojo module.

You should merge the basic integration configuration before merging your custom integration configuration as shown above.

```

var customIntegrationConfiguration = {
  bindings: {
    collection: {
      attributes: {
        properties: {
          //Get the properties from the "props" member of the model.
          get: "props"
        }
      }
    },
    property: {
      attributes: {
        common: {
          id: {
            //Get the id from the "symbolicName" member of
            //the model object.
            get: "symbolicName"
          },
          cardinality:
            // Compute the cardinality from the "multiValue"
            //member of the model object.
            get: function(model) {
              return model.multiValue ? "multi" : "single";
            }
        }
      }
    }
  }
}

```

5. Adding an external property to a view

You can add an external property to a view in the Properties View Designer within Case Builder. The procedure is similar to adding a case property to a view. You must drag and drop an “External Property” object from the Properties palette to the canvas and then must manually configure its binding settings in the Settings panel.

Adding an external property on to a custom view allows you to have the same control over its placement and style within the view as you have with case properties and task properties.

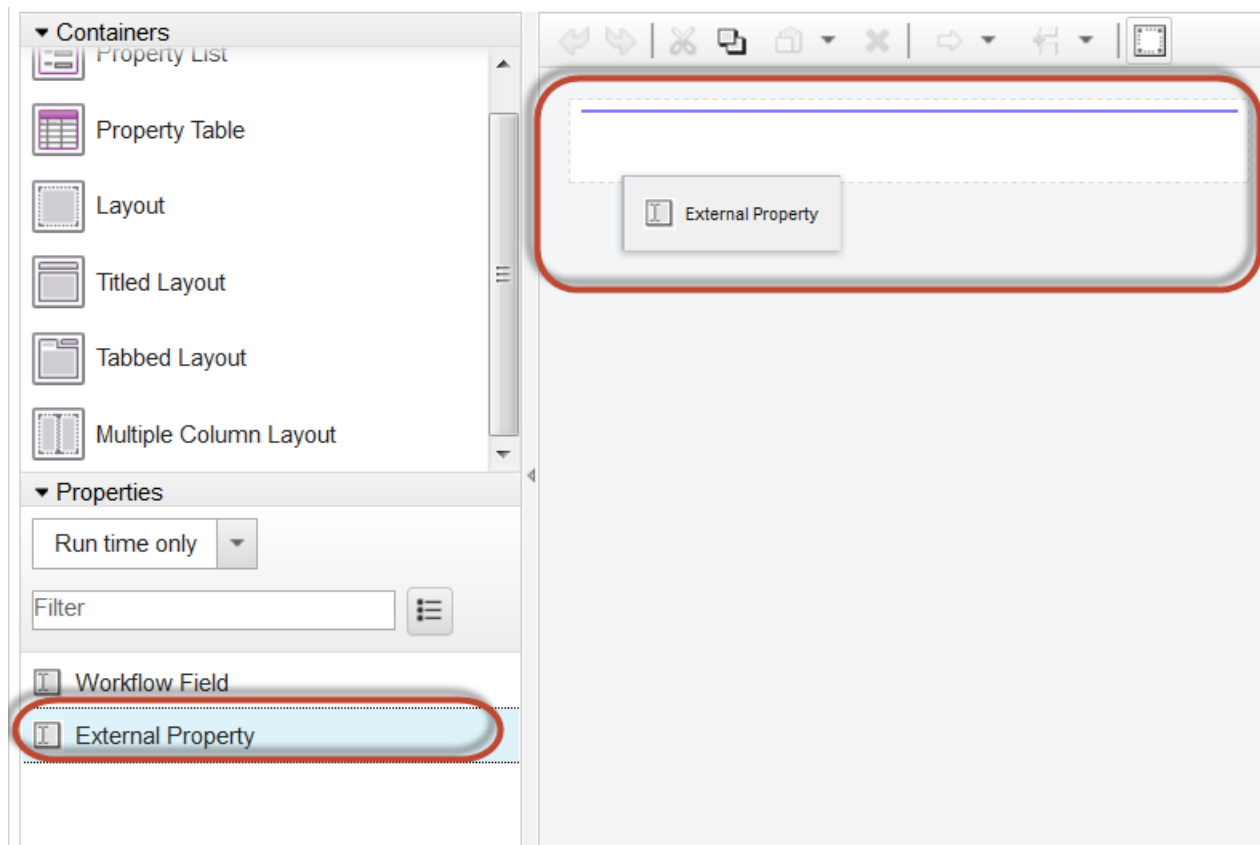
The binding settings consist of the collection identifier and property identifier associated with the external property at run time. The collection identifier must match the collection identifier passed to the PropertyCollectionController's bind method when the external properties collection is bound to the controller at run time. The property identifier must match the “id” attribute of the associated property within this collection. If a matching property is not found in the collection at run time, the Property object will be removed from the view.

To add an external property to a container in Properties View Designer:

- Open the desired custom view
- In the Properties palette, select Run time only from the drop-down list.



- Then, drag External Property into the container.



- Select the external property and edit the settings:
- Enter the collection identifier and property identifier that Case Manager Client will use to locate the property at run time.

- Select the data type of the external property and define the property settings as needed.

▼ Settings

External Property Settings

Collection ID: ? External

Property ID: ? propertyId1

Type: ? String ▼

Multiple values: ? ☐

Property Settings

Label: ? propertyId1

Help: ?

Read-only: ? ☐

Required: ? ☐

Hidden: ? ☐

Default Value: ?

Editor Settings

Editor: ? Text box ▼

You can leave the data type as Unspecified. In this situation, Case Manager Client determines the data type at run time and uses the appropriate default control for editing the property.

6. Retrieving and persisting external properties

The retrieval and persistence of external properties depend entirely on your unique application requirements and must therefore be implemented in custom code. For one scenario, they might be located in a database and for another they might be from a web service. Another scenario might simply use the data to update the status of another widget on the page.

Typically, you will coordinate the retrieval of external properties with the rendering of the page and the persistence of external properties with the persistence of the page data. A Coordination object is provided to coordinate these activities among the various widgets on the page. Your custom code in the Script Adapter widget can participate in this coordination as follows (where `getExternalProperties` and

setExternalProperties are application-specific methods provided in your custom code).

```
require(["icm/base/Constants",
"icm/model/properties/controller/ControllerManager"],
function(Constants, ControllerManager) {
    /* Get the coordination and editable objects from the event payload.
    */
    var coordination = payload.coordination;
    var editable = payload.caseEditable;

    /* Use the BEFORELOADWIDGET coordination topic handler to obtain the
    controller binding */
    /* for the editable and to update the properties. */
    coordination.participate(Constants.CoordTopic.BEFORELOADWIDGET,
function(context, complete, abort) {
    /* Obtain the controller binding for the editable. */
    var controller = ControllerManager.bind(editable);
    /* Retrieve the external properties and bind them to the controller. */
    var externalProperties = getExternalProperties(); /* You must provide
    this function. */
    controller.bind("Ext1", "Ext1", externalProperties); /* You may
    optionally provide an integration object if a non-standard model is
    used. */

    /* Call the coordination completion method. */
    complete();
});
/* Use the SAVE coordination topic handler to release the controller
binding for the editable. */
coordination.participate(Constants.CoordTopic.SAVE,
function(context, complete, abort) {
    /* Release the controller binding for the editable. */
    ControllerManager.unbind(editable); /* Will automatically release
    the external properties binding. */

    /* Save the external properties. */
    saveExternalProperties(externalProperties); /* You must provide this
    function. */

    /* Call the coordination completion method. */
    complete();
});
});
```

7. Product Help and References

- IBM Case Manager Knowledge Center

http://www01.ibm.com/support/knowledgecenter/SSCTJ4_5.2.1/com.ibm.casemgmttoc.doc/casemanager_5.2.1.htm?lang=en

- Creating custom property editors devWorks article

https://www.ibm.com/developerworks/community/blogs/e8206aad-10e2-4c49-b00c-fee572815374/resource/ACM_LP/ExternalPropertiesICM.pdf?lang=en