Custom Validation of Task Step Data

Dave Perman – Product Manager | IBM Case Manager
ICM Version - 5.1.1

Use Case

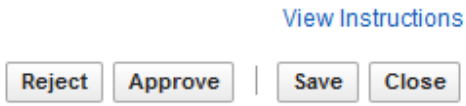
The standard out-of-the-box case data widget used on both case and step details pages supports basic validation of data types it knows about. This includes dates, numbers and choice lists.

In many solutions, you might require additional checking of the data the user enters, and often, the validation is based on other factors such as previously entered data or which response was selected when completing a work item.

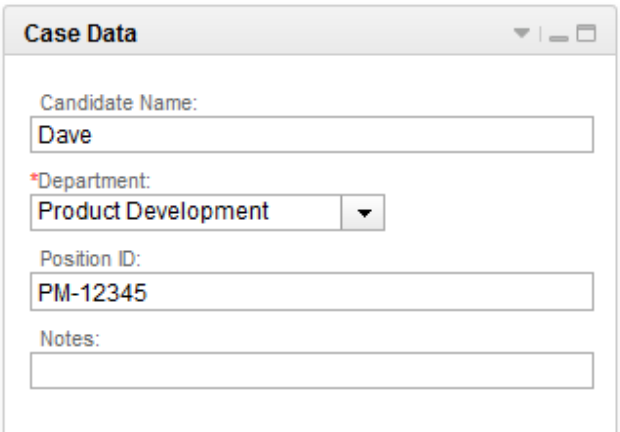
Use of an eForm as a replacement for the case/step data UI helps greatly with the checks and calculations side, but still doesn't address the response issue.

For this use case, let's imagine the following:

- The task step has two possible responses, Approve and Reject, and a Save option.
- When saving, the missing data validations do not need to be performed since the work is not being completed yet
- The data validation if different depending upon the response selected



The data for our example looks like this:



The logic we want will be:

- If the user clicks Save, do not perform any extra data validation and just do the save
- If the user clicks Approve, the Position ID field must not be blank and the user must not be allowed to complete the work
- If the user clicks Reject, the Notes field must have an entry and the user must not be allowed to complete the work

Standard ICM Widget Events

ICM uses a hidden Command Widget to orchestrate communication between page actions and the widgets on the page.

When the user performs an action such as Complete on a task step page, a series of events are initiated to ensure:

- The data in each widget is valid
- The step data is updated from each participating widget (eg data and attachments)
- Each widget cleans up in preparation for the page closing
- The step is updated or dispatched according to the action selected

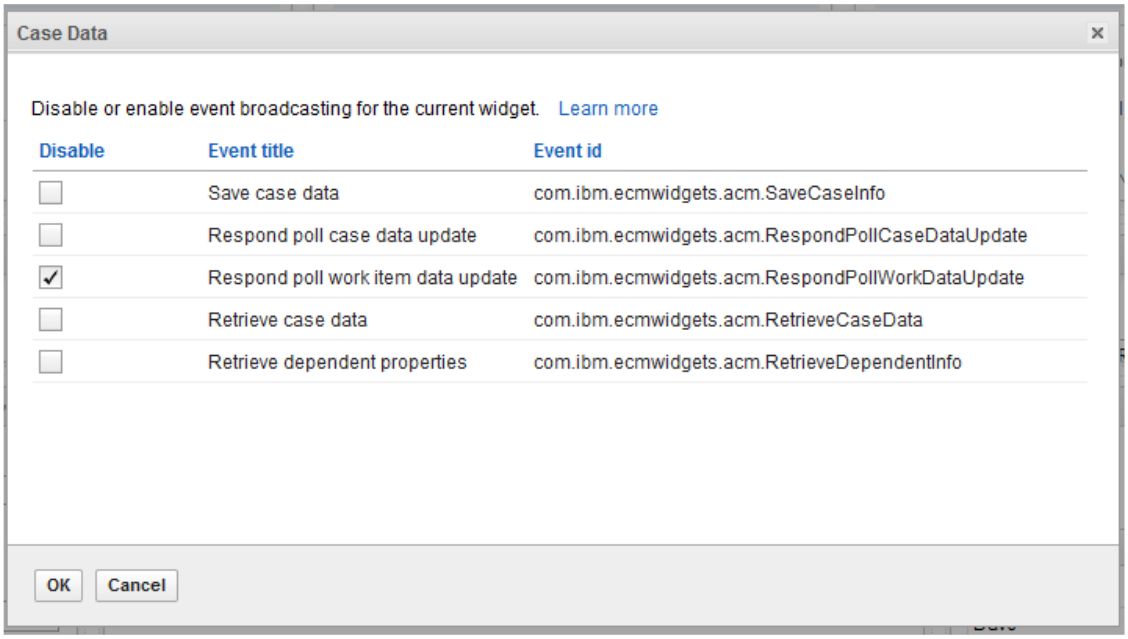
These events are sent between the widgets using a *Broadcast* method. This means that the widgets themselves are not actually manually wired together but instead, each

participating widget *listens* for events it understands and *broadcasts* their events to anyone who is interested.

Let's look at the flow that occurs when you click a response button in the Work Item toolbar.

- The Toolbar widget broadcasts the **Complete Work Item** event. The payload for this event includes the name of the button that was clicked which includes any response values that may have been defined in the workflow
- The Command Widget receives the **Complete Work Item** event and broadcasts out a **Poll Work Item** which tells each widget to respond with a **Respond Poll Work Item Update** event.
- Each participating widget responds with the **Respond Poll Work Item Update** event which includes a payload containing the updated data from the widget and whether the data is valid
- The Command widget receives the **Respond Poll Work Item Update** event and if all responses come back valid, merges the data together and completes the step.

Each ICM widget has a settings panel that allows you to disable the broadcast events it sends.



This allows customizations where another widget, such as the Script Adaptor, can intercept an event and potentially change its default behavior.

NOTE: You can always just wire the Script Adapter or a custom widget to another widget's events using the Wiring settings and listen in, but that does not prevent the broadcast event from occurring as well and being handled by other widgets..

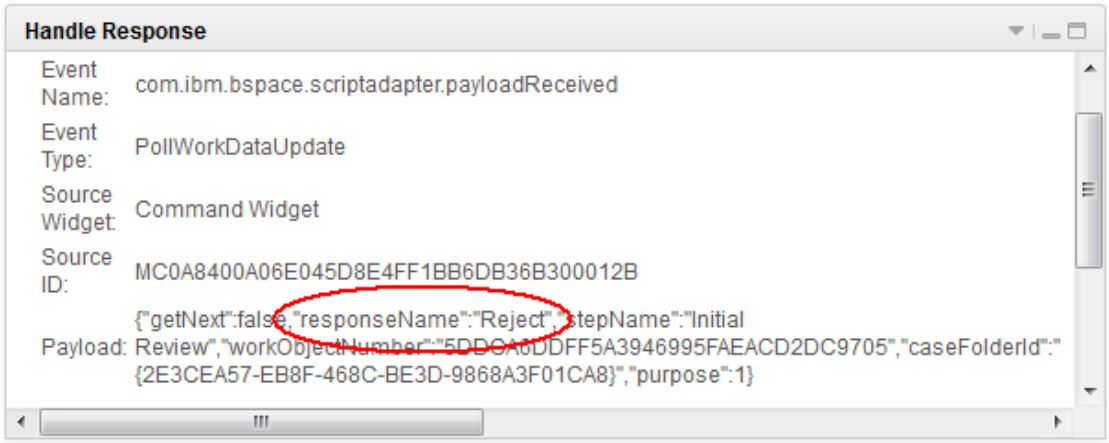
When you disable an event using the widget settings, you must manually wire that event up where it is expected if you want the page to behave in its normal manner.

Intercepting the Standard ICM Widget Events

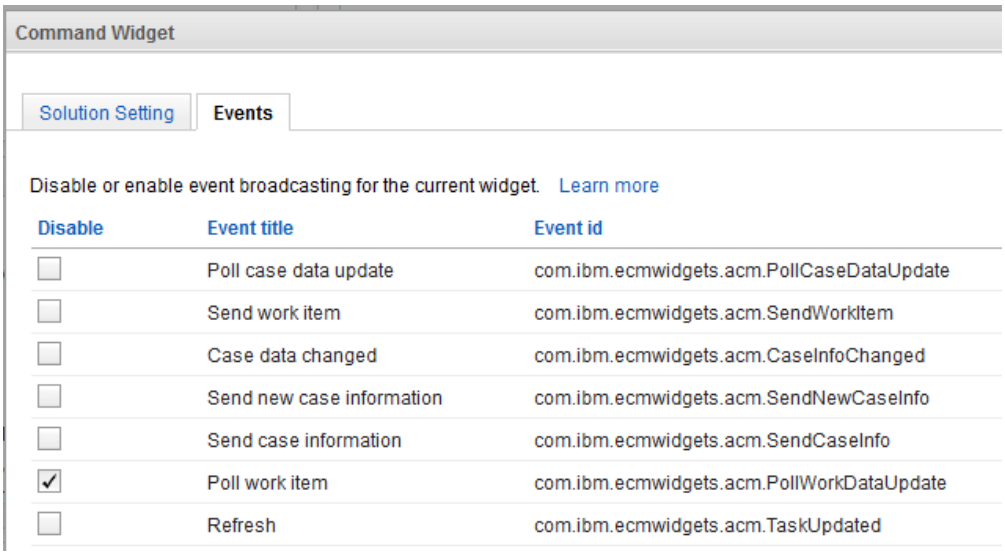
To handle this use case, we need to do two key things:

- Capture the response value that was clicked so that we can use is in our validation logic
- Validate the data we are interested in and notify the user if something is wrong and make them fix it

The first event we are going to intercept is the **Poll Work Item** event broadcast from the Command widget. The payload for this event happens to contain the response that was clicked.



We start by disabling the **Poll Work Item** in the **Command widget** settings.

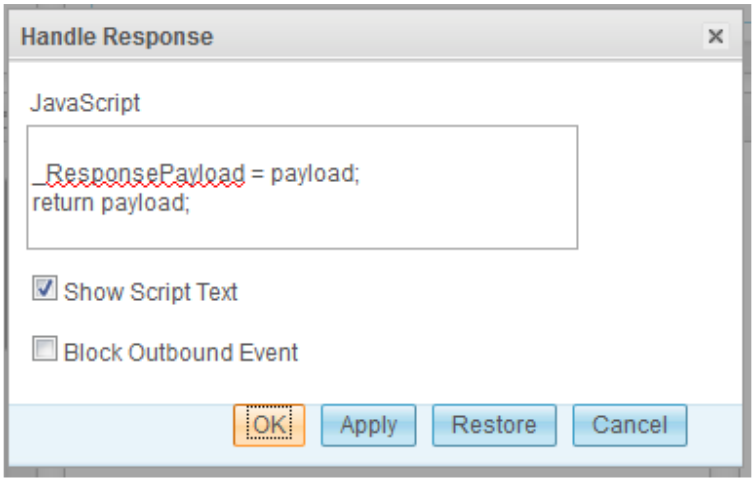


We then add a **Script Adapter** to the page, hide it, and rename it Handle Response.

We then wire the same **Poll Work Item** event to each widget we know is expecting it. This includes our new Handle Response script adaptor and the standard Attachment widget.

When the event is fired, we grab the response value in our Handle Event script adapter widget by copying it into a global javascript variable we define on the fly called `_ResponsePayload`. The script then simply passed on the payload it was given using the return statement.

NOTE: It is good practice to make sure any global variables you define have a name that is unique and won't accidently collide with an existing object.

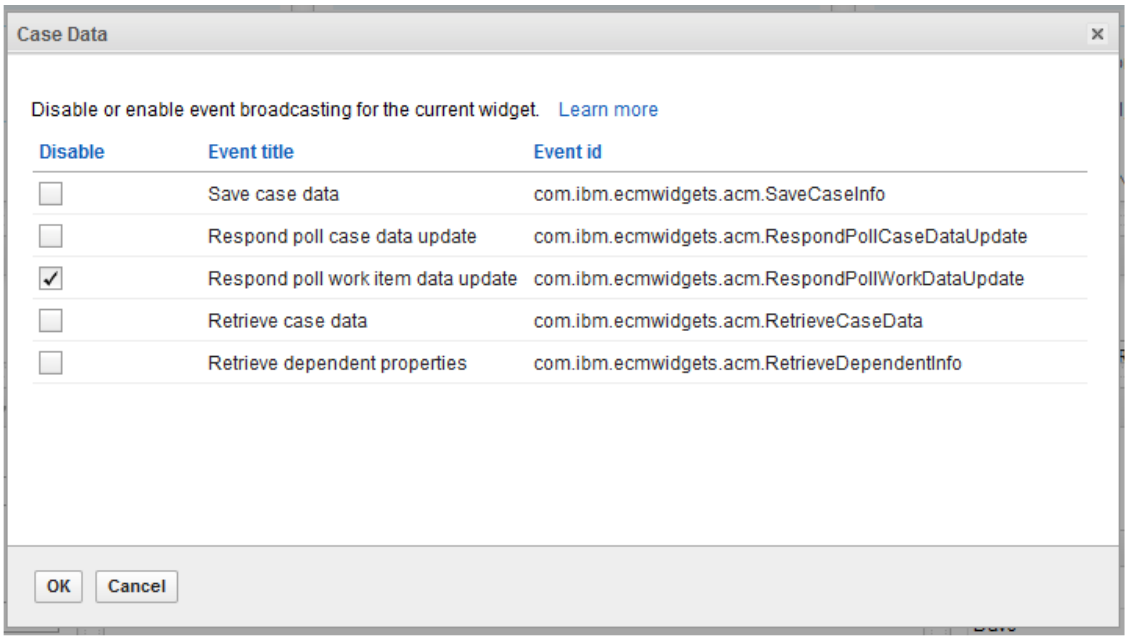


To make sure the event is passed on as usual, we need to wire it to the Case Data widget's Poll Work Item event.

With that completed, we need to add another script adapter to perform the actual validation work. Like before, add a Script Adapter widget to the page, hide it, and rename it Validate Data.

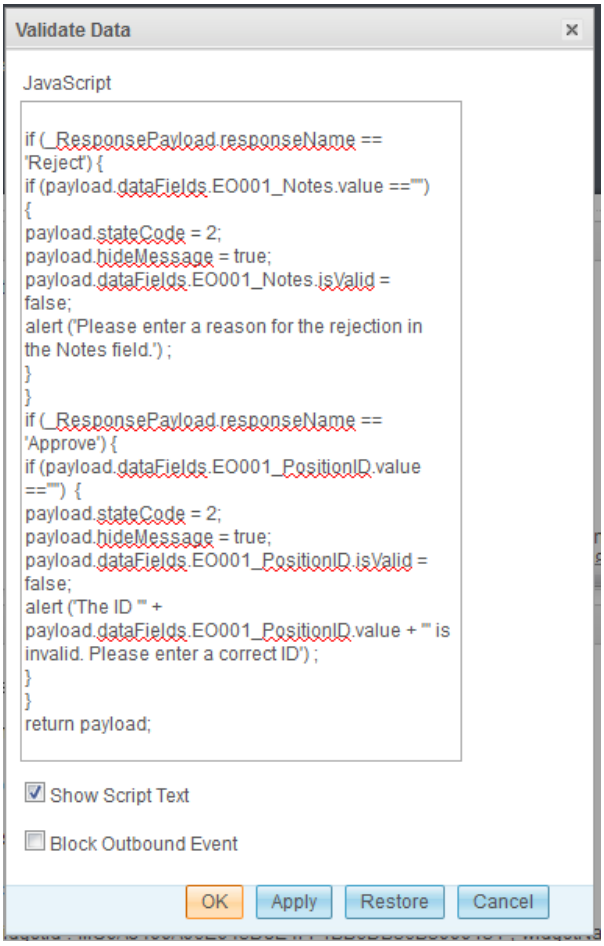
This time, we are intercepting the **Respond Poll Work Item** event that is being sent from the Case Data widget back to Command widget in response to the initial poll event.

Like before, we also need to disable the broadcast event from the Case Data widget so our Script Adapter can intercept, not just listen in.



We then wire our Validate Data script adapter to both the Case Data and the Command widget, making sure to wire the **Respond Poll Work Item** on each end (be sure not to choose the similar events that are used for case data on a case details page).

We now enter our validation code as javascript in our Validate Data script adapter. We use the payload we intercepted before and stored as a global to get the response value. The code below checks which button was clicked and performs the validation required for each. Any other action is simply passed through untouched.



The key to this step are the lines:

```
payload.stateCode = 2;
payload.hideMessage = true;
```

This stateCode tells the Command widget whether the data in the widget is valid or not with the value 2 indicating an invalid state and that a message has already been shown to the end user. The hideMessage parameter makes sure any default error messages that the Command widget might display are suppressed.

Conclusion

The ability to intercept and override ICM widget events allows for the creation of custom solutions that extend the standard behavior of the out-of-the-box widgets.