OBJECTIFLUNE

Custom Plugin Architecture

Technical Article

For years now, Objectif Lune has made available two distinct SDKs for users to design their own Workflow plugins and thereby enhance the overall functionality of Workflow. The first one - which is pretty much deprecated by now - required knowledge of programming languages like Delphi or C++ to create native plugins. The second SDK, known as the Custom Plugin Architecture, only required knowledge of HTML/CSS and a scripting language, all technologies that Connect users are very familiar with.

While this second SDK has proven useful in allowing some of our partners to create tens of custom plugins, it is limited in its abilities to support more elaborate GUIs for end-users. In particular, it is very difficult to provide validation routines to ensure end users provide the proper information when they design a Workflow process that includes such custom plugins. In addition, the scripting languages used by the custom plugins at runtime (VBScript, Jscript) have been severely lagging behind the more recent improvements implemented in ECMASCRIPT-compliant versions of JavaScript.
Consequently, we decided to overhaul the custom plugin architecture to make it more current, practical and powerful.

What's new

The GUI

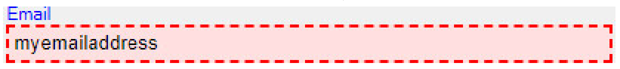
The most evident change is in the HTML rendering engine, which is used to display the GUI for the plugins at configuration time. The new rendering engine is fully HTML5/CSS3-compliant and supports ECMASCRIPT 2017 JavaScript. This allows plugin designers to use a combination of CSS and JS code to provide dynamic interactive validation of the parameters provided by the end users when they configure the plugin.
Furthermore, methods are available for the GUI to obtain a list of all Connect Resources available to Workflow, which allows plugin designers to build GUIs that mimic those available in native plugins.
Because the new engine supports HTML5, the GUI can be built around the latest HTML <FORM> elements, making native validation of standard fields much easier since they require very little - if any - code. For instance, if the GUI requires an email address and a phone number to be entered by the end user, the GUI can be built using the following elements:

```
<input id="fldEmail" type="email" name="Email" required />
<input id="test" type="phone" pattern="[0-9]\.[0-9]{3}\.[0-9]{3}\.[0-9]{4}"
name="phone" required />
```

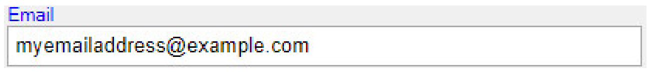
Additionally, the CSS could include something like:

```
input:invalid {
   border: 2px dashed red;
}
```

By specifying the proper input `type` for each field (i.e. email, phone) and setting the `required` attribute, the new HTML engine is able to automatically style each field to highlight any invalid value entered by the user. If the email address entered by the user is invalid, the engine would display it as:



But as soon as the email format is valid, the engine displays the field as:

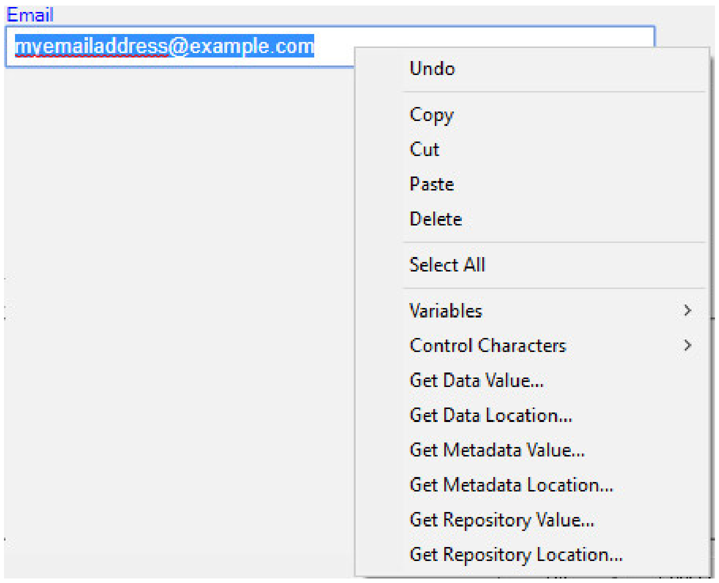


Of course, you can add JS code to further validate the input values, but the above is already a huge improvement over what could be achieved previously.

One other feature than can be controlled through a basic CSS class is the ability to assign the standard Workflow context menu to certain fields when those fields allow for dynamic values like Jobinfos, data selections or system variables. Simply add the "**is-parsable**" class to an input field as in this example:

```
<input id="fldEmail" class="is-parsable" type="email" name="Email" required />
```

When right clicking an input field with that class, the Workflow context menu is displayed:



Many other goodies become available with the new rendering engine, and going through all of them would be tedious, but they can be summarized with this statement: if you can achieve something in a standard Web Form, it is highly likely that you do the same in a custom plugin's GUI. By the way, it has come to our attention that some of our own people have already started using Connect Designer to create GUIs for custom plugins!

The run time scripting engine

The second critical part of custom plugins is the Scripting Engine used at run time to actually perform

operations. Our intention was to provide a fully ECMASCRIPT 2017-compliant JavaScript engine, to take advantage of the massive improvements implemented in JS, particularly over the last few years. Unfortunately, because of time constraints, we have not been able to implement this new JS engine for the 2019.2 release

(note that the JS engine used at run time is distinct from the one used at design time), so the current JScript engine must still be used. However, we managed to significantly upgrade the version of JScript. Make sure to read the Updated JScript article for more information on that.

We are now planning to implement the completely new JS engine in the 2020.1 release, but given the vast improvements to the GUI engine we decided we wouldn't hold off until then to start deploying plugins built with this upgraded custom architecture. All new plugins created by the R&D teams will, from this moment forward, be using this new architecture.

Plugins built with the new architecture

Several new plugins shipped with Connect 2019.1 have been built using the new custom plugin architecture. As far as end-users are concerned, those plugins are the same as any other in Workflow. But from an Objectif Lune perspective, using the new architecture allows all R&D departments to work on them, instead of just the Montreal team, which should translate into a significantly shorter time to market for new features.

All plugins shipped with the standard distribution of Connect are always encrypted, in order to avert the risk of having users modify the code (which could quickly become a nightmare for our Support teams). However, we will make the original code for each plugin available on demand to PSO's if there is ever a need to tweak the functionality in some specific instances, or to serve as learning tools for those who wish to create their own custom plugins.

SDK availability

As we have done many times in the past with other features, we will not yet be releasing this updated SDK to customers and partners because it is still in a state of flux and will likely keep changing in the coming months. And because its documentation is also in need of a major overhaul, we are not yet ready to release it internally to our professional services teams, but we will be putting significant efforts into making it available to them as soon as possible, hopefully before year's end.

Note that the new plugin architecture can not yet be used to create Input/Output plugins. Those features are on our road map, but we have not determined yet if they will be included with the initial release of the architecture.

Internally, we will keep using it to develop all new Workflow plugins, which will allow us to tweak it and make sure it contains everything it should before allowing 3rd parties to use it. We are expecting to make it a public release along with the Connect 2020.1 version.

A community of plugins

Some of you may already be familiar with Node-RED, an open-source automation tool that was initially created to simplify the wiring of IoT devices and APIs. The application has gained momentum over the last couple of years and is now catering to a much broader range of applications. Node-RED uses a system of plugins and allows you to string them into processes that can respond to events, similar to what Workflow does. It should therefore come as no surprise that Node-RED uses the same kind of plugin architecture that Workflow does: HTML/CSS/JS. The only notable difference is found in the plugin's configuration file, but we are currently working on bridging that gap.

What this means is that plugins built for Node-RED can be imported relatively easily into Workflow, and vice-versa. Since Node-RED already has a public library of thousands of open source plugins, extending Workflow to support the same kind of functionality could now become a much easier task. Conversely, providing Connect-specific plugins to the Node-RED community could also foster general awareness about Connect in a community that would otherwise have never heard of it.

One of the driving factors behind Node-RED's increasing popularity is its vibrant community of open-source developers, who release tons of new plugins each week. Over the next few months, we will be evaluating whether Objectif Lune should form its own community of open-source developers and plugins, or perhaps merge with Node-RED's existing community, assuming that we can reach an acceptable level of compatibility between the two applications. But in the meantime, we will be publishing new plugins through our existing help.objectiflune.com page so that end users don't have to wait for a new release of the software before being able to use new functionality.