Runtime parameters for data mapping

Runtime parameters were introduced for Job Creation in version 2019.2. They proved useful in allowing users to know which values were required for any given Job Preset without having to open it up and inspect it. The next step was therefore to implement the same functionality for the Data Mapping operation. This article discusses the changes to the DataMapper, Workflow and the REST API

What are runtime parameters?

Runtime parameters are values that must be provided dynamically by any application that wants to perform an OL Connect operation. For instance, a data mapping configuration might require a Date to be provided at run time so that it can extract that value and compute the number of days between that date and other dates in the data. Another example could be a data mapping configuration that requires a base URL to be provided so that it can build PURL hyperlinks for each data record in the job.

Runtime parameters were already available in previous versions of Connect, but they were specific to Workflow: they were known as *Automation Variables*. In order to use them, you had to make sure your data mapping configuration included references to Workflow process variables that you *knew* were available in the Workflow config. When building a data mapping config, that usually meant a lot of back and forth between the DataMapper and Workflow in order to make sure you were creating the proper variables and setting them up correctly. In addition, because they were designed with Workflow variables in mind, their notation did not make much sense for other applications (scripts, NodeRED, web pages, etc.).

Changes to the DataMapper

In the DataMapper's Preprocessor step, the *scopes* available in the list of **Properties** have changed : what used to be named *Automation Variable* is now called *Runtime Parameter*. That's pretty much it as far as the GUI is concerned! From now on, scripts can use the notation `automation.parameters.myParameter` to access the runtime parameter named `myParameter`.

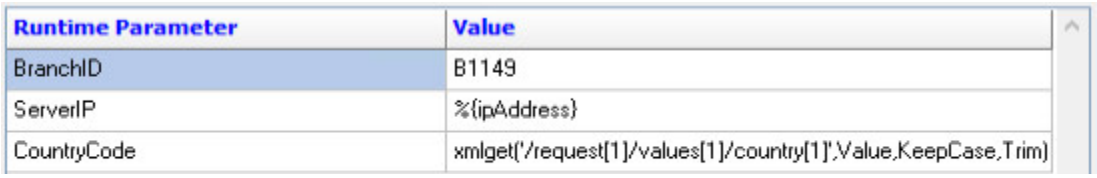
Note that your current data mapping configurations are automatically updated and will keep running as before. The`automation.variables.myVariable`syntax is still valid (in fact, for existing configs, *parameters* is just an alias for *variables*), and the `JobInfos` are also still available, but you should start using the `parameters` notation because the `variables/JobInfos` notation is now considered deprecated and will eventually be phased out.

So why the fuss, you ask? Isn't this just syntactic sugar?

Nope. Read on.

Changes to Workflow

In Workflow, both the Data Mapping and the All In One tasks have been modified in order to display the list of runtime parameters that are expected by the configuration you select.



Runtime Parameter	Value
BranchID	B1149
ServerIP	%{ipAddress}
CountryCode	xmlget('/request[1]/values[1]/country[1]',Value,KeepCase,Trim)

By default, each value displays the default value that was specified in the DM Config. But you can change those values to use any of the dynamic values available in Workflow, including data/metadata selections. This makes it a lot easier to pass the proper information to the DM Config without having to open it up first to see what it expects to receive.

Note that by default, existing configurations display the list of *Automation Variables* defined in the DM Config, so you don't have to do anything to start using the functionality.

Changes to the REST API

This is where the changes are most evident, even though the vast majority of users will only ever see the changes to the UI in Workflow and the DataMapper.

Before 2020.1, the JSON structure that was passed to the DataMapper engine looked like this:

```
{ "identifier": "25653",
   "ProcessInformations": {
      "ProcessName": "Process1",
      "OriginalFilename": "7_SampleData.txt",
      "TaskIndex": 2,
      "JobInfos": {
         "JobInfo1": "",
         "JobInfo2": "",
         "JobInfo3": "",
         "JobInfo4": "",
         "JobInfo5": "",
         "JobInfo6": "",
         "JobInfo7": "",
         "JobInfo8": "",
         "JobInfo9": ""
      },
      "LocalVariables": { },
      "GlobalVariables": { }
   }
}
```

This structure was obviously Workflow-oriented and it allowed the data mapping task in Workflow to automatically pass a lot of relevant information to the DM Config. Conversely, that also meant that *too much* irrelevant information was often passed to the config: imagine that JobInfo9, for some reason, contains the BASE64 code for an image that needs to be passed later on to the Content Creation process. Well that entire image would also be sent to the DataMapper automatically, which means a lot of bandwidth was being used for data that wasn't relevant to the DM operation.

Starting with Version 2020.1, the new JSON structure looks like this (this example is for the DM REST call; the All In One structure is slightly different but the changes to it are similar in scope):

```
{ "identifier": "25653",
  "ProcessInformations": {
      ...
   },
   "parameters": {
      "BranchID": "B1149",
      "ServerIP": "127.0.0.1",
      "CountryCode": "UK"
   }
}
```

The new `parameters` element has been added as a sibling to the existing `ProcessInformations` element. It contains only the parameters that are required by the DM Config. Note that when using the REST API, the `ProcessInformations` element can still be specified (for backward compatibility) but it is no longer *required*. The existing Workflow tasks (DM and AIO) will still keep using this element to maintain backward compatibility. However, you should start using explicit runtime parameters instead of relying on JobInfos or process variable names, because those are considered deprecated as they would be somewhat awkward to implement in automation applications like NodeRED, for instance.

Conclusion

The runtime parameters for data mapping help streamline operations being triggered by external applications. They eliminate the dependency on Workflow, thereby opening the door to other 3rd party automation applications. This relatively simple change is therefore in line with the long term plans for Connect, which will see it transform into a cloud-able server that can be called from any number of third-party applications.