IFS REST Call

IFS REST Call

The IFS REST Call is a custom delegate that can be used to make requests to a specific endpoint and then parse and read the returned data into the execution variables to be used elsewhere in a Workflow.

The IFS REST Call Task requires the following input:

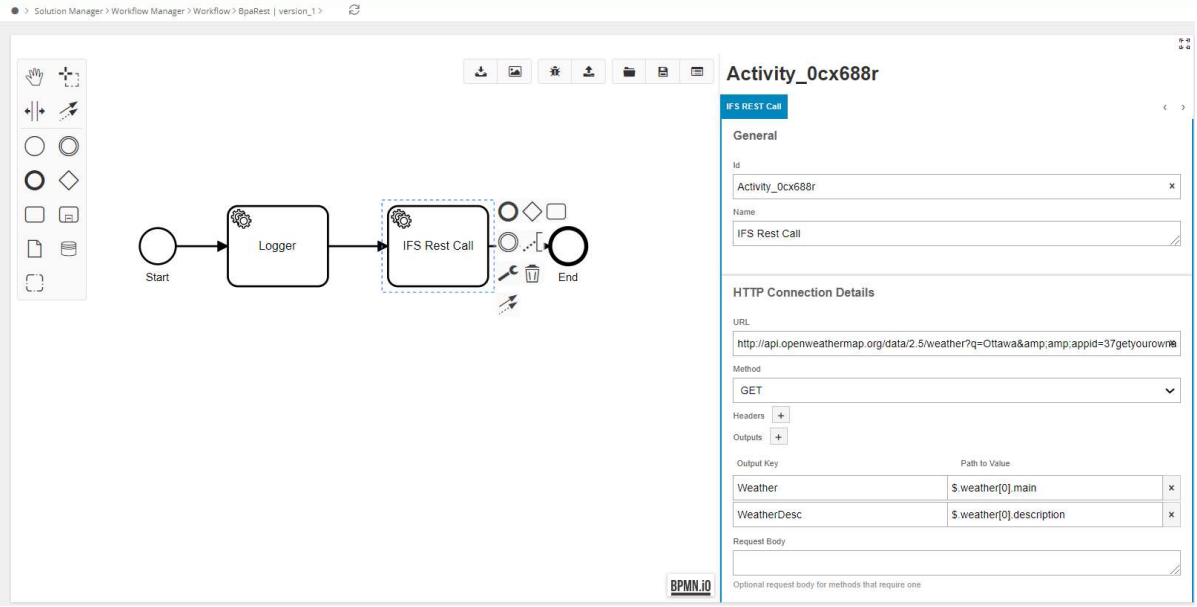
- **URL** (mandatory) - A HTTP or HTTPS endpoint that will return the desired JSON or XML data to parse
- **Method** (mandatory) - The Method to use on the endpoint
- **Headers** (optional) - Any headers that need to be included in the request
- **Outputs** (optional) - The variable key to put the retrieve data in and the path to the data
- **Request Body** (optional) - A request body if one is needed

A full URL including beginning with http or https is needed. The request methods Get, Post, Put, Patch and Delete are all supported. In the case that a request body is needed then one can be provided in the *Request Body* field. If one is not needed then the *Request Body* field can be left empty and no body will be added to the request. Expression Language can be used to insert any value into the body that will not be known ahead of time. Any headers put in the *Header* map field will be added to the request. If none are provided then the request will be sent with no headers.

The *Outputs* map field holds two key pieces of information of what to do after getting a successful response. Once a successful response comes back the returned data gets parsed and ready for reading. The *Path to Value* field tells the parser where to look for the data to read. The *Output Key* is the key the reader associates the read data with when it puts it into the execution variables. After the IFS REST Call completes other elements in a Workflow can access execution variables and find the retrieved data under the specified keys.

One more thing to note is that the response data returned must be below 1mb in size or it will not be parsed.

Example of a Get request:



Below is the generated XML for the IFS REST Call,

```
<bpmn:serviceTask id="Activity_0cx688r" name="IFS Rest Call"
camunda:class="com.ifsworld.fnd.bpa.delegate.IfsHttpConnectionDelegate">
      <bpmn:extensionElements>
        <camunda:inputOutput>
          <camunda:inputParameter
name="ifsBpaHttpConnectionMethod">GET</camunda:inputParameter>
          <camunda:inputParameter
name="ifsBpaHttpConnectionUrl">http://api.openweathermap.org/data/2.5/weather?
q=Ottawa&amp;amp;amp;appid=37getyourownappidifyouwanttotestthisexample</camunda:inpu
          <camunda:inputParameter name="ifsBpaHttpConnectionOutputs">
            <camunda:map>
              <camunda:entry key="Weather">$.weather[0].main</camunda:entry>
              <camunda:entry
key="WeatherDesc">$.weather[0].description</camunda:entry>
            </camunda:map>
          </camunda:inputParameter>
        </camunda:inputOutput>
      </bpmn:extensionElements>
      <bpmn:incoming>Flow_03h2irf</bpmn:incoming>
      <bpmn:outgoing>Flow_0eteipz</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:serviceTask id="Activity_1t9nz4n" name="Logger">
      <bpmn:incoming>Flow_1s6e95p</bpmn:incoming>
      <bpmn:outgoing>Flow_03h2irf</bpmn:outgoing>
    </bpmn:serviceTask>
```

Configuring IFS REST Call with a JSONPath

In order to find the specific data of interest inside the retrieved JSON/XML data, a JSONPath will be needed to specify where to look. Typically, JSONPath is JSON specific but in the case of our IFS REST Call JSONPath's functionality has been extended to also work for XML documents. So both JSON/XML data are supported without the user having to learn more than one query language.

Example of JSONPath for JSON Output

```
{
    "employees" : {
        "person" : [
            {
                "name" : "Bob",
                "age" : 45
            },
            {
                "name" : "Emma",
                "age" : 22
            },
            {
                "name" : "Sally",
                "age" : 31
            },
            {
                "name" : "George",
                "age" : 39
            },
            {
                "name" : "Billy",
                "age" : 19
            }
        ]
    },
    "person" : {
        "name" : "Joe",
        "age" : 35
    }
}
```

JSON Path	Result	Description
$.person.name	"Joe"	Get the owner's name
$.employees.person[0]	{"name":"Bob", "age":45}	Get the employee in the list at index 0

JSON Path	Result	Description
$.employees.person[?(@.name=='Emma')]	[{"name":"Emma","age":22}]	Get all employees be the name of emma
$.employees.person[0,3,4].name	["Bob","George","Billy"]	Get the employees in the list at indexes 0, 3 and 4
$.employees.person[1:3].name	["Emma","Sally"]	Get the employees in the list from index 1 up to but not including index 3
$.employees.person[*].age	[45,22,31,39,19]	Get the age of all employees
$..age	[45,22,31,39,19,35]	Get every age in the document
$.employees.person[?(@.age > 30)].name	["Bob","Sally","George"]	Get the name of every employee over the age of 30
$.employees.person.length()	5	Get the number of employees

To learn more about what JSONPath can be used to do and to view some examples check out the JsonPath Github.

Please note that the Expression Language can be used in a JSONPath. However, since some of the operators between the languages overlap there may be some unexpected behaviors.

Examples of JSONPath for XML Output

The XML is parsed in a fairly straightforward way. The element tags act as the names that get referenced in the JSONPath and the structure of the path is walked through the same way as JSON.

The element tags will be the name of the objects.

```
<root>
    <person>Bob</person>
</root>
```

JSONPath	Returns
$.root.person	Bob

Any attributes will be added as a property of an object and can be access the same way the element tags are.

```
<root id="1001">
    <person>Bob</person>
</root>
```

JSONPath	Returns
$.root.person	Bob
$.root.id	1001

If any attributes exist on an element that only has text as a child then the text will be put under the tag .

```
<root>
    <person id="1001">Bob</person>
</root>
```

JSONPath	Returns
$.root.person.	Bob
$.root.person.id	1001

The text is also placed under a tag if the text is not the only child of an element.

```
<root>
    <person>Bob</person>extra_text
</root>
```

JSONPath	Returns
$.root.person	Bob
$.root.	extra_text

All text that exists as the child of an element will be concatenated together with a space separating it.

```
<root>extra_text_1
    <person>Bob</person>extra_text_2
</root>
```

JSONPath	Returns
$.root.person	Bob
$.root.	extra_text_1 extra_text_2

Any CData is handled like text except for the fact that it always goes under the tag.

```
<root>
    <![CDATA[this_is_cdata_1]]>
<info><![CDATA[this_is_cdata_3]]></info>
<![CDATA[this_is_cdata_2]]>
</root>
```

JSONPath	Returns
$.root['']	this_is_cdata_1 this_is_cdata_2

JSONPath	Returns
$.root.info['']	this_is_cdata_3

If element has multiple children with the same tag then these elements will all be put into a list under the shared tag name. Be aware that the parser will only know if an element is meant to be in a list if there is more then one element with the same name. Lists of one are not supported.

```
<root>
    <id>1001</id>
    <person>Bob</person>
    <person>Emma</person>
</root>
```

JSONPath	Returns
$.root.id	1001
$.root.person	[Bob,Emma]

If an element has an attribute and child (or children) that share a tag/name then they also get thrown into a list together. Attributes will always be added to the end of a list.

```
<root id="1002">
    <id>1001</id>
    <person>Bob</person>
</root>
```

JSONPath	Returns
$.root.id	[1001, 1002]
$.root.person	Bob

NOTE: *Other then the elements, attributes, text and CData everything else is basically ignored. Comments and processing instructions are ignored and won't be available on the path for reading. Entities and entity references may behave as expected but are not officially supported.*

Internal Input Variables

Some input variables are automatically added and set behind the scenes to pass around important information. These input variables start with the prefix ifsBpa and therefore it is recommended to avoid naming variables with this prefix.

Internal Input Variable	Use
ifsBpaExecutionUserId	Added internally at runtime. Holds the current user ID.
ifsBpaCurrentProjectionName	Added internally at runtime. Name of the projection that was invoked.
ifsBpaCurrentInputEntityTypeName	Added internally at runtime. Name of current entity type being used.
ifsBpaProjectionAction	Added by IfsProjectionDelegate. CRUD action to invoke.
ifsBpaProjectionName	Added by IfsProjectionDelegate. Name of the projection to invoke a CRUD action on.
ifsBpaProjectionEntitySetName	Added by IfsProjectionDelegate. Name of the entity set to use.
ifsBpaProjectionCallSignature	Added by IfsProjectionDelegate. Method signature of the Call to use.
ifsBpaCallReturnValueName	Added by IfsProjectionDelegate. The key the returned Call value gets paired with in the execution variables.
ifsBpaProjectionETagVariableName	Added by IfsProjectionDelegate. The name of the variable that the invoked projection's ETag will get carried around in.

Internal Input Variable	Use
ifsBpaProjectionParameters	Added by IfsProjectionDelegate. Parameters to feed to projection being invoked.
ifsBpaHttpConnectionUrl	Added by IfsHttpConnectionDelegate. The URL to send the request to.
ifsBpaHttpConnectionMethod	Added by IfsHttpConnectionDelegate. The method of the request.
ifsBpaHttpConnectionHeaders	Added by IfsHttpConnectionDelegate. The headers of the request.
ifsBpaHttpConnectionOutputs	Added by IfsHttpConnectionDelegate. The variable key for the data and path to the data.
ifsBpaHttpConnectionBody	Added by IfsHttpConnectionDelegate. The body of the request.
ifsBpaValidationErrorMessages	Added by Terminal end events. Contains the error message translation map of the error that was triggered by a failed validation.
ifsBpaValidationErrorCode	Added by Terminal end events. Contains a BpaErrorCode of an error that was triggered by a failed validation.
ifsBpaImpl *+ Form Field ID*	Added by User Task Form Fields. Holds the label translation map for a form field.
ifsBpaProjectionOperation	Added internally at runtime. Holds the Projection Action Name. Possible values are *CREATE*, *UPDATE*, *READ*, *DELETE*.