This article describes various "recipes" on how to accomplish certain tasks with the NiFi processor [ExecuteScript](https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi.processors.script.ExecuteScript/index.html" \t "_blank), with examples given in [Groovy](http://www.groovy-lang.org/), [Jython](http://www.jython.org/" \t "_blank), Javascript ([Nashorn](http://openjdk.java.net/projects/nashorn/" \t "_blank)), and [JRuby](http://jruby.org/" \t "_blank). Recipes in this article series include:

***Part 1 - Introduction to the NiFi API and FlowFiles***

* **Getting a flow file from an incoming queue**
* **Creating new flow files**
* **Working with flow file attributes**
* **Transferring a flow file**
* **Logging**

[Part 2](https://community.hortonworks.com/articles/75545/executescript-cookbook-part-2.html)*- FlowFile I/O and Error Handling*

* Reading from a flow file
* Writing to a flow file
* Reading and writing to/from a flow file
* Error Handling

[Part 3](https://community.hortonworks.com/articles/77739/executescript-cookbook-part-3.html)*- Advanced Features*

* Using Dynamic Properties
* Adding Modules
* State Management
* Accessing Controller Services

*Introduction*

ExecuteScript is a versatile processor that allows the user to code custom logic in a programming language that will be executed each time the ExecuteScript processor is triggered. The following variable bindings are provided to the script to enable access to NiFi components:

session: This is a reference to the ProcessSession assigned to the processor. The session allows you to perform operations on flow files such as create(), putAttribute(), and transfer(), as well as read() and write().

context: This is a reference to the ProcessContext for the processor. It can be used to retrieve processor properties, relationships, Controller Services, and the StateManager.

log: This is a reference to the ComponentLog for the processor. Use it to log messages to NiFi, such as log.info('Hello world!')

REL\_SUCCESS: This is a reference to the "success" relationship defined for the processor. It could also be inherited by referencing the static member of the parent class (ExecuteScript), but some engines such as Lua do not allow for referencing static members, so this is a convenience variable. It also saves having to use the fully-qualified name for the relationship.

REL\_FAILURE: This is a reference to the "failure" relationship defined for the processor. As with REL\_SUCCESS, it could also be inherited by referencing the static member of the parent class (ExecuteScript), but some engines such as Lua do not allow for referencing static members, so this is a convenience variable. It also saves having to use the fully-qualified name for the relationship.

Dynamic Properties: Any dynamic properties defined in ExecuteScript are passed to the script engine as variables set to the PropertyValue object corresponding to the dynamic property. This allows you to get the String value of the property, but also to evaluate the property with respect to NiFi Expression Language, cast the value as an appropriate data type (such as Boolean, e.g.), etc. Because the dynamic property name becomes the variable name for the script, you must be aware of the variable naming properties for the chosen script engine. For example, Groovy does not allow periods (.) in variable names, so an error will occur if "my.property" was a dynamic property name.

Interaction with these variables is done via the NiFi Java API, each recipe below will discuss the relevant API calls as they are introduced. The recipes in the following section perform various functions on flow files, such as reading/writing attributes, transferring to a relationship, logging, etc. Please note that the examples are snippets and do not run as-is. For example, if a flow file has been retrieved from the queue with session.get(), it must be transferred to a relationship or removed, or else an error will occur. The snippets are meant to be plain and clear to illustrate only the concept(s) presented, without the addition of boilerplate code to make them working examples. In a later article I will put them all together to show full working scripts that perform useful tasks.

*Recipes*

Recipe: Get an incoming flow file from the session

Use Case: You have incoming connection(s) to ExecuteScript and want to retrieve one flow file from the queue(s) for processing.

Approach: Use the get() method from the session object. This method returns the FlowFile that is next highest priority FlowFile to process. If there is no FlowFile to process, the method will return null. Note that it is possible to have null returned even if there is a steady flow of FlowFiles into the processor. This can happen if there are multiple concurrent tasks for the processor, and the other task(s) have already retrieved the FlowFiles. If the script requires a FlowFile to continue processing, then it should immediately return if null is returned from session.get()

Examples:

*Groovy*

flowFile = session.get()

if(!flowFile) return

*Jython*

flowFile = session.get()

if (flowFile != None):

# All processing code starts at this indent

# implicit return at the end

*Javascript*

var flowFile = session.get();

if (flowFile != null) {

// All processing code goes here

}

*JRuby*

flowFile = session.get()

if flowFile != nil

# All processing code goes here

end

Recipe: Get multiple incoming flow files from the session

Use Case: You have incoming connection(s) to ExecuteScript and want to retrieve multiple flow files from the queue(s) for processing.

Approach: Use the get(*maxResults*) method from the session object. This method returns up to *maxResults* FlowFiles from the work queue. If no FlowFiles are available, an empty list is returned (the method does not return null). NOTE: If multiple incoming queues are present, the behavior is unspecified in terms of whether all queues or only a single queue will be polled in a single call. Having said that, the observed behavior (for both NiFi 1.1.0+ and before) is described [here](https://issues.apache.org/jira/browse/NIFI-2751).

Examples:

*Groovy*

flowFileList = session.get(100)

if(!flowFileList.isEmpty()) {

flowFileList.each { flowFile ->

// Process each FlowFile here

}

}

*Jython*

flowFileList = session.get(100)

if not flowFileList.isEmpty():

for flowFile in flowFileList:

# Process each FlowFile here

*Javascript*

flowFileList = session.get(100)

if(!flowFileList.isEmpty()) {

for each (var flowFile in flowFileList) {

// Process each FlowFile here

}

}

*JRuby*

flowFileList = session.get(100)

if !(flowFileList.isEmpty())

flowFileList.each { |flowFile|

# Process each FlowFile here

}

end

Recipe: Create a new FlowFile

Use Case: You want to generate a new FlowFile to send to the next processor

Approach: Use the create() method from the session object. This method returns a new FlowFile object, which you can perform further processing on

Examples:

*Groovy*

flowFile = session.create()

// Additional processing here

*Jython*

flowFile = session.create()

# Additional processing here

*Javascript*

var flowFile = session.create();

// Additional processing here

*JRuby*

flowFile = session.create()

# Additional processing here

Recipe: Create a new FlowFile from a parent FlowFile

Use Case: You want to generate new FlowFile(s) based on an incoming FlowFile

Approach: Use the create(*parentFlowFile*) method from the session object. This method takes a parent FlowFile reference and returns a new child FlowFile object. The newly created FlowFile will inherit all of the parent's attributes except for the UUID. This method will automatically generate a Provenance FORK event or a Provenance JOIN event, depending on whether or not other FlowFiles are generated from the same parent before the ProcessSession is committed.

Examples:

*Groovy*

flowFile = session.get()

if(!flowFile) return

newFlowFile = session.create(flowFile)

// Additional processing here

*Jython*

flowFile = session.get()

if (flowFile != None):

newFlowFile = session.create(flowFile)

# Additional processing here

*Javascript*

var flowFile = session.get();

if (flowFile != null) {

var newFlowFile = session.create(flowFile);

// Additional processing here

}

*JRuby*

flowFile = session.get()

if flowFile != nil

newFlowFile = session.create(flowFile)

# Additional processing here

end

Recipe: Add an attribute to a flow file

Use Case: You have a flow file to which you'd like to add a custom attribute.

Approach: Use the putAttribute(*flowFile*, *attributeKey*, *attributeValue*) method from the session object. This method updates the given FlowFile's attributes with the given key/value pair. NOTE: The "uuid" attribute is fixed for a FlowFile and cannot be modified; if the key is named "uuid", it will be ignored.

Also this is a good point to mention that FlowFile objects are immutable; this means that if you update a FlowFile's attributes (or otherwise alter it) via the API, you will get a new reference to the new version of the FlowFile. This is very important when it comes to transferring FlowFiles to relationships. You must keep a reference to the latest version of a FlowFile, and you must transfer or remove the latest version of all FlowFiles retrieved from or created by the session, otherwise you will get an error when executing. Most often, the variable used to store a FlowFile reference will be overwritten with the latest version returned from a method that alters the FlowFile (intermediate FlowFile references will be automatically discarded). In these examples you will see this technique of reusing a flowFile reference when adding attributes. Note that the current reference to the FlowFile is passed into the putAttribute() method. The resulting FlowFile has an attribute named 'myAttr' with a value of 'myValue'. Also note that the method takes a String for the value; if you have an Object you will have to serialize it to a String. Finally, please note that if you are adding multiple attributes, it is better to create a Map and use putAllAttributes() instead (see next recipe for details).

Examples:

*Groovy*

flowFile = session.get()

if(!flowFile) return

flowFile = session.putAttribute(flowFile, 'myAttr', 'myValue')

*Jython*

flowFile = session.get()

if (flowFile != None):

flowFile = session.putAttribute(flowFile, 'myAttr', 'myValue')

# implicit return at the end

*Javascript*

var flowFile = session.get();

if (flowFile != null) {

flowFile = session.putAttribute(flowFile, 'myAttr', 'myValue')

}

*JRuby*

flowFile = session.get()

if flowFile != nil

flowFile = session.putAttribute(flowFile, 'myAttr', 'myValue')

end

Recipe: Add multiple attributes to a flow file

Use Case: You have a flow file to which you'd like to add custom attributes.

Approach: Use the putAllAttributes(*flowFile*, *attributeMap*) method from the session object. This method updates the given FlowFile's attributes with the key/value pairs from the given Map. NOTE: The "uuid" attribute is fixed for a FlowFile and cannot be modified; if the key is named "uuid", it will be ignored.

The technique here is to create a Map (aka dictionary in Jython, hash in JRuby) of the attribute key/value pairs you'd like to update, then call putAllAttributes() on it. This is much more efficient than calling putAttribute() for each key/value pair, as the latter case will cause the framework to create a temporary version of the FlowFile for each attribute added (see above recipe for discussion on FlowFile immutability). The examples show a map of two entries myAttr1 and myAttr2, set to '1' and the language-specific coercion of the number 2 as a String (to adhere to the method signature of requiring String values for both key and value). Note that a session.transfer() is not specified here (so the code snippets below do not work as-is), see the following recipe for that.

Examples:

*Groovy*

attrMap = ['myAttr1': '1', 'myAttr2': Integer.toString(2)]

flowFile = session.get()

if(!flowFile) return

flowFile = session.putAllAttributes(flowFile, attrMap)

*Jython*

attrMap = {'myAttr1':'1', 'myAttr2':str(2)}

flowFile = session.get()

if (flowFile != None):

flowFile = session.putAllAttributes(flowFile, attrMap)

# implicit return at the end

*Javascript*

var number2 = 2;

var attrMap = {'myAttr1':'1', 'myAttr2': number2.toString()}

var flowFile = session.get()

if (flowFile != null) {

flowFile = session.putAllAttributes(flowFile, attrMap)

}

*JRuby*

attrMap = {'myAttr1' => '1', 'myAttr2' => 2.to\_s}

flowFile = session.get()

if flowFile != nil

flowFile = session.putAllAttributes(flowFile, attrMap)

end

Recipe: Get an attribute from a flow file

Use Case: You have a flow file from which you'd like to inspect an attribute.

Approach: Use the getAttribute(*attributeKey*) method from the FlowFile object. This method returns the String value for the given attributeKey, or null if the attributeKey is not found. The examples show the retrieval of the value for the "filename" attribute.

Examples:

*Groovy*

flowFile = session.get()

if(!flowFile) return

myAttr = flowFile.getAttribute('filename')

*Jython*

flowFile = session.get()

if (flowFile != None):

myAttr = flowFile.getAttribute('filename')

# implicit return at the end

*Javascript*

var flowFile = session.get()

if (flowFile != null) {

var myAttr = flowFile.getAttribute('filename')

}

*JRuby*

flowFile = session.get()

if flowFile != nil

myAttr = flowFile.getAttribute('filename')

end

Recipe: Get all attributes from a flow file

Use Case: You have a flow file from which you'd like to retrieve its attributes.

Approach: Use the getAttributes() method from the FlowFile object. This method returns a Map with String keys and String values, representing the key/value pairs of attributes for the flow file. The examples show an iteration over the Map of all attributes for a FlowFile.

Examples:

*Groovy*

flowFile = session.get()

if(!flowFile) return

flowFile.getAttributes().each { key,value ->

// Do something with the key/value pair

}

*Jython*

flowFile = session.get()

if (flowFile != None):

for key,value in flowFile.getAttributes().iteritems():

# Do something with key and/or value

# implicit return at the end

*Javascript*

var flowFile = session.get()

if (flowFile != null) {

var attrs = flowFile.getAttributes();

for each (var attrKey in attrs.keySet()) {

// Do something with attrKey (the key) and/or attrs[attrKey] (the value)

}

}

*JRuby*

flowFile = session.get()

if flowFile != nil

flowFile.getAttributes().each { |key,value|

# Do something with key and/or value

}

end

Recipe: Transfer a flow file to a relationship

Use Case: After processing a flow file (new or incoming), you want to transfer the flow file to a relationship ("success" or "failure"). In this simple case let us assume there is a variable called "errorOccurred" that indicates which relationship to which the FlowFile should be transferred. Additional error handling techniques will be discussed in part 2 of this series.

Approach: Use the transfer(*flowFile*, *relationship*) method from the session object. From the documentation: this method transfers the given FlowFile to the appropriate destination processor work queue(s) based on the given relationship. If the relationship leads to more than one destination the state of the FlowFile is replicated such that each destination receives an exact copy of the FlowFile though each will have its own unique identity.

NOTE: ExecuteScript will perform a session.commit() at the end of each execution to ensure the operations have been committed. You do not need to (and should not) perform a session.commit() within the script.

Examples:

*Groovy*

flowFile = session.get()

if(!flowFile) return

// Processing occurs here

if(errorOccurred) {

session.transfer(flowFile, REL\_FAILURE)

}

else {

session.transfer(flowFile, REL\_SUCCESS)

}

*Jython*

flowFile = session.get()

if (flowFile != None):

# All processing code starts at this indent

if errorOccurred:

session.transfer(flowFile, REL\_FAILURE)

else:

session.transfer(flowFile, REL\_SUCCESS)

# implicit return at the end

*Javascript*

var flowFile = session.get();

if (flowFile != null) {

// All processing code goes here

if(errorOccurred) {

session.transfer(flowFile, REL\_FAILURE)

}

else {

session.transfer(flowFile, REL\_SUCCESS)

}

}

*JRuby*

flowFile = session.get()

if flowFile != nil

# All processing code goes here

if errorOccurred

session.transfer(flowFile, REL\_FAILURE)

else

session.transfer(flowFile, REL\_SUCCESS)

end

end

Recipe: Send a message to the log at a specified logging level

Use Case: You want to report some event that has occurred during processing to the logging framework.

Approach: Use the log variable with the warn(), trace(), debug(), info(), or error() methods. These methods can take a single String, or a String followed by an array of Objects, or a String followed by an array of Objects followed by a Throwable. The first one is used for simple messages. The second is used when you have some dynamic objects/values that you want to log. To refer to these in the message string use "{}" in the message. These are evaluated against the Object array in order of appearance, so if the message reads "Found these things: {} {} {}" and the Object array is ['Hello',1,true], then the logged message will be "Found these things: Hello 1 true". The third form of these logging methods also takes a Throwable parameter, and is useful when an exception is caught and you want to log it.

Examples:

*Groovy*

log.info('Found these things: {} {} {}', ['Hello',1,true] as Object[])

*Jython*

from java.lang import Object

from jarray import array

objArray = ['Hello',1,True]

javaArray = array(objArray, Object)

log.info('Found these things: {} {} {}', javaArray)

*Javascript*

var ObjectArrayType = Java.type("java.lang.Object[]");

var objArray = new ObjectArrayType(3);

objArray[0] = 'Hello';

objArray[1] = 1;

objArray[2] = true;

log.info('Found these things: {} {} {}', objArray)

*JRuby*

log.info('Found these things: {} {} {}', ['Hello',1,true].to\_java)