TGH
Making Integrations Simpler

boomi Partner

Updating Data in Stream with Groovy and JavaScript in Boomi

Author
Anshul Singh

Contents

Introduction

It is possible to generate documents directly from data processing shapes in Boomi by storing the data in a streaming store and producing the document. In this discussion, we will explore how to achieve this using both Groovy and JavaScript.

Groovy code breakdown

Let's explore the code in detail:

```
import java.util.Properties;
import java.io.InputStream;
import com.boomi.execution.ExecutionUtil;
import java.io.BufferedReader;

def payload1 = "Hello, I am Groovy";

for (int i = 0; i < dataContext.getDataCount(); i++) {
   InputStream is = dataContext.getStream(i)
   Properties props = dataContext.getProperties(i)
   BufferedReader reader = new BufferedReader(new InputStreamReader(is));
   StringBuffer outData = new StringBuffer();
   outData.append(payload1);

   is = new ByteArrayInputStream(outData.toString().getBytes("UTF-8"));
   dataContext.storeStream(is, props)
}
```

Code Explanation

1. Imports:

- `import java.util.Properties;`: Imports the `Properties` class to work with properties associated with data items.

- `import java.io.InputStream;`: Imports `InputStream` for handling input streams.

- `import com.boomi.execution.ExecutionUtil;`: Imports `ExecutionUtil` from the Boomi platform, allowing access to data context and properties.

- `import java.io.BufferedReader;`: Imports `BufferedReader` for reading data from input streams.

2. Defining `payload1`:

- `def payload1 = "Hello, I am Groovy";`: This line defines a string variable named `payload1` with the text message we want to apply to the data items.

(As I have taken Static payload but it is not needed this payload may come from previous document or from any property)

3. For Loop:

- `for (int i = 0; i < dataContext.getDataCount(); i++) {`: Initiates a `for` loop that iterates through the data items in the `dataContext` collection.

4. Inside the Loop:

- **`InputStream is** = dataContext.getStream(i)`: Retrieves the input stream associated with the current data item using the loop counter `i`.

- **`Properties props** = dataContext.getProperties(i)`: Obtains the properties associated with the current data item.

- **`BufferedReader reader** = new BufferedReader(new InputStreamReader(is));`: Creates a `BufferedReader` to read the content of the input stream.

- **`StringBuffer outData** = new StringBuffer();`: Initializes a `StringBuffer` named `outData`. This variable will store the transformed data.

- **`outData.append(payload1);`:** Appends the `payload1` (the fixed text) to `outData`.

- **`is** = new ByteArrayInputStream(outData.toString().getBytes("UTF-8"));`: Converts the modified data in `outData` back into an input stream with UTF-8 encoding.

- **`dataContext.storeStream(is, props)`:** Stores the modified input stream back into the `dataContext`, replacing the original input stream for the current data item.

Conclusion

This Groovy code snippet demonstrates a simple yet practical example of data transformation within an integration context. It iterates through a collection of data items, appends a fixed text message to each item, and updates the data context with the transformed data. Understanding these concepts can be valuable when working on data manipulation tasks in your own integration projects.

Boomi implementation

Step1: create one process in process canvas and take start shape of no data

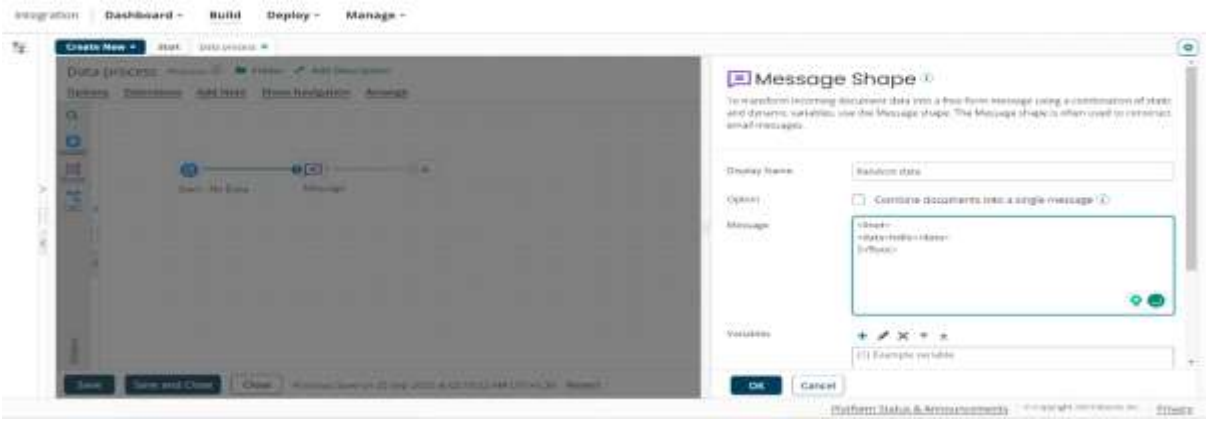
Step2: Take a message shape and store random xml data

For ex

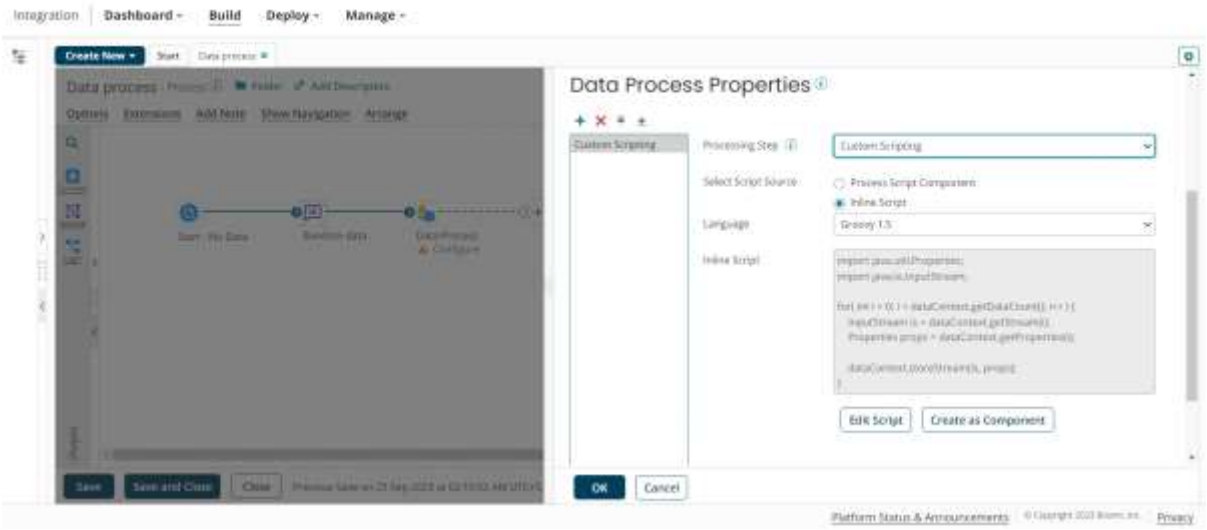
<Root>

<data>hello</data>

</Root>

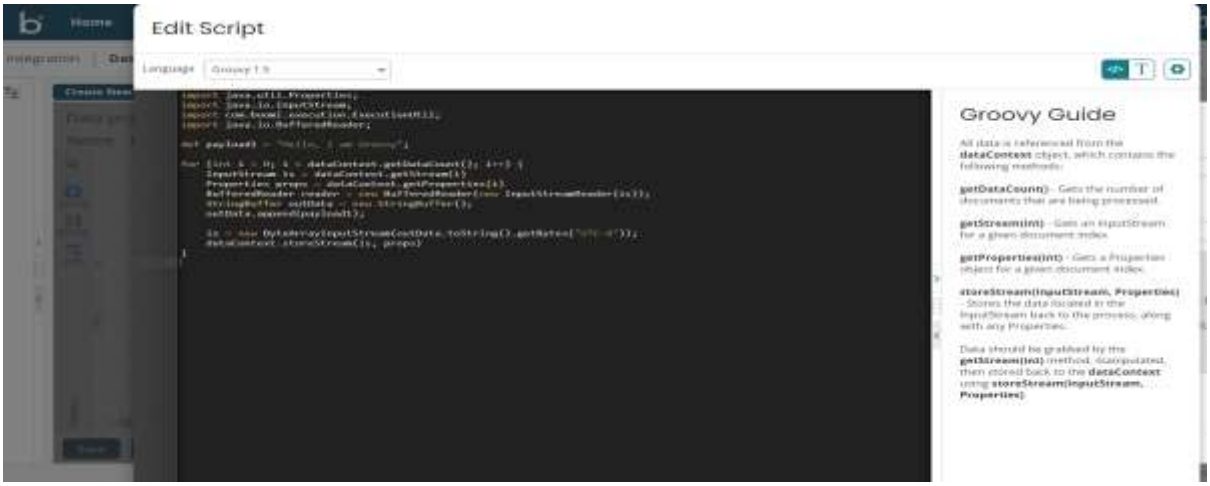


Step3: now take data process shape and choose custom scripting and choose Groovy



click Edit Script

Step4: copy paste the above code here

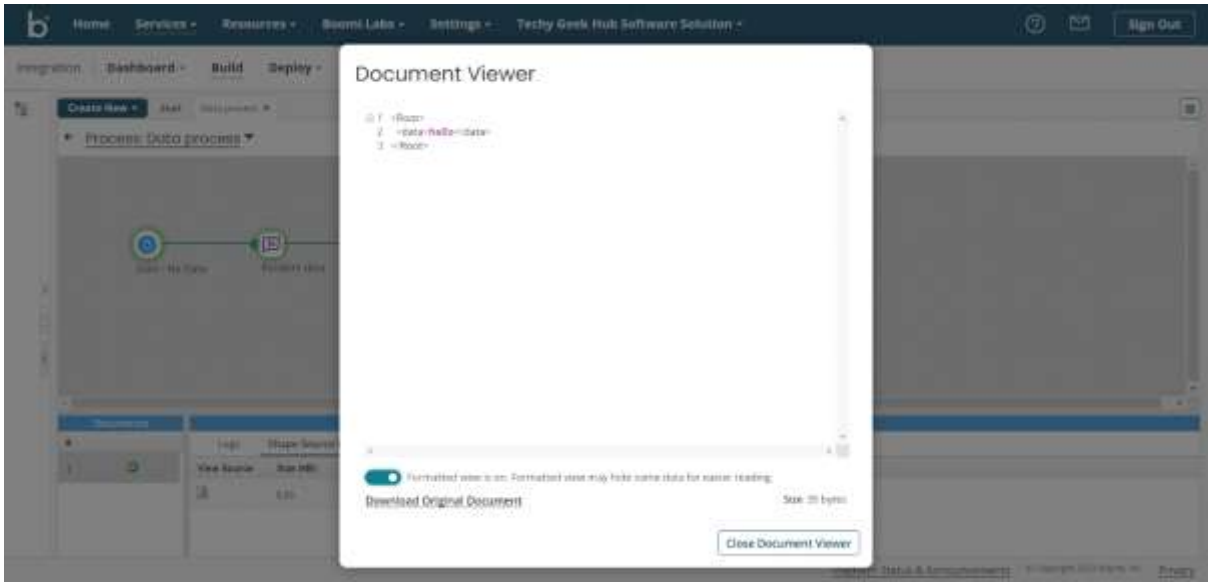


Save it

Step5: add stop shape at end and test the process

so, the data which is being passed to data process shape is xml while after that we are getting what ever we stored in payload1

Shape source data in Data process shape



Shape source data in stop shape

As the process is working for data process shape let's check for **JavaScript**

JavaScript code breakdown

Code:

```
// Load compatibility script
load("nashorn:mozilla_compat.js");
// Load Java classes
importClass(com.boomi.execution.ExecutionUtil);
importClass(java.io.ByteArrayInputStream);
// Iterate through incoming data
for (var i = 0; i < dataContext.getDataCount(); i++) {
    var is = dataContext.getStream(i);
    var props = dataContext.getProperties(i);
   // Create outputString
    var outputString = "Hi I am JavaScript";
   // Create a new ByteArrayInputStream, passing in the string
    var newStream = new java.io.ByteArrayInputStream(outputString.getBytes("UTF-8"));
  // Pass the new input stream back
    dataContext.storeStream(newStream, props);
}
```

Code Explanation

Let us break down the key components of this JavaScript script:

1.Loading Compatibility Script: The script starts by loading the compatibility script `nashorn:mozilla_compat.js`. This script is used to make JavaScript compatible with Java, as Boomi is built on Java technology.

2. Importing Java Classes: Next, it imports two Java classes, `com.boomi.execution.ExecutionUtil` and `java.io.ByteArrayInputStream`. These classes provide essential functions and operations required for handling data within Boomi.

3.Iteration through Data: The `for` loop iterates through the incoming data using `dataContext.getDataCount()`. For each data item, it retrieves the input stream and properties associated with it.

4. Creating Output String: It then creates a simple output string, "**Hi I am JavaScript**." This is the message that will replace the original data.

(As I have taken Static payload but it is not needed this payload may come from previous document or from any property)

5.Creating ByteArrayInputStream: A new `ByteArrayInputStream` is created, converting the output string into bytes using UTF-8 encoding.

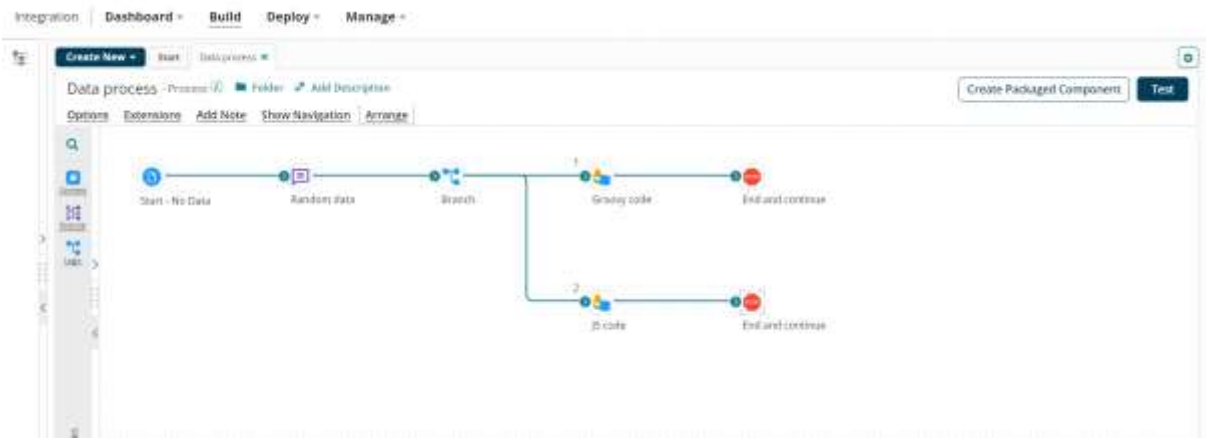
6.Storing the Modified Data: Finally, the modified data, in the form of a new input stream, is stored back using `dataContext.storeStream(newStream, props)`. This step effectively replaces the original data with the JavaScript-generated message.

Conclusion

JavaScript scripting in Dell Boomi provides a powerful way to customize your integration processes, enabling you to perform a wide range of data transformations and operations. In this blog post, we've demonstrated a simple JavaScript script that iterates through incoming data and replaces it with a predefined message. You can build upon this foundation to create more complex and tailored integration solutions to meet your specific business needs.

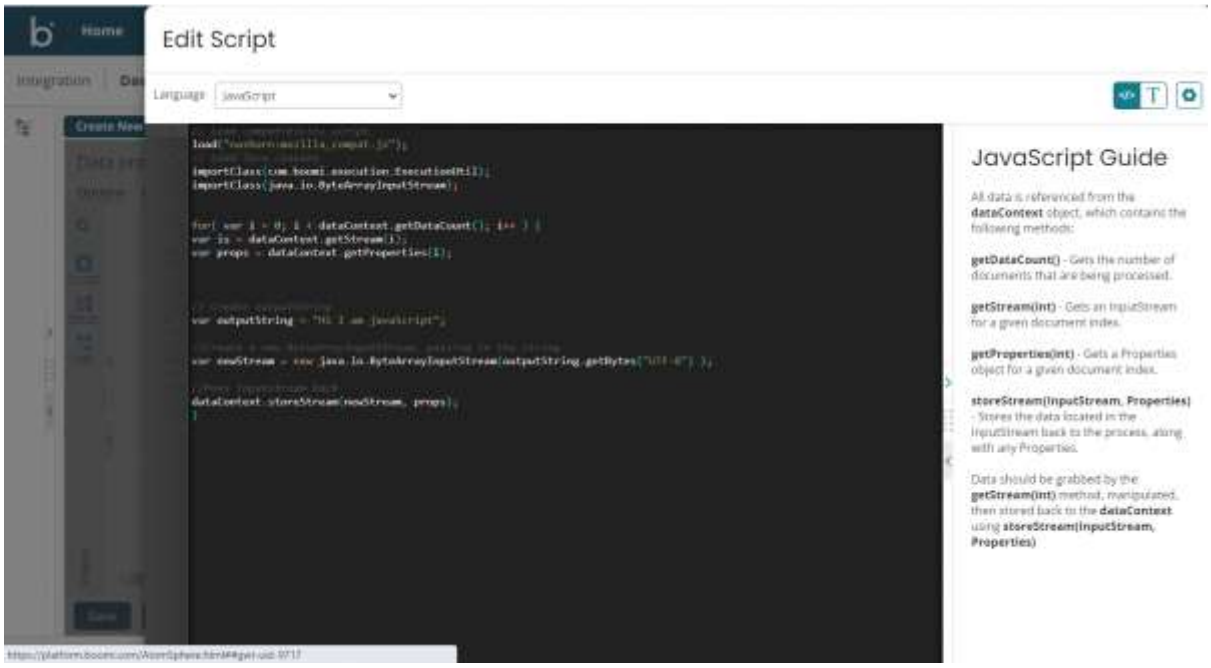
Boomi implementation

Step1: In the same process where, we attached message shape we will add branch shape



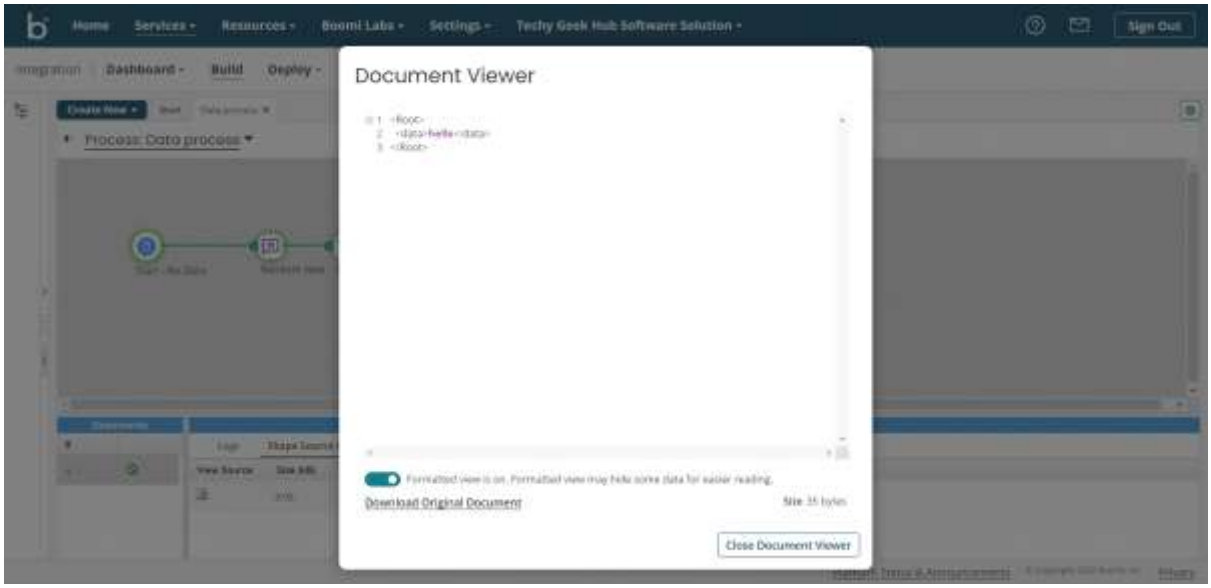
in the second branch we will add data process shape and stop shape

Step2: now open the data process shape and paste the above JavaScript code



we will save this and test the process

Shape source data in Data process shape



Shape source data in stop shape

Final process

(These scripts are not dependent can be implemented individually)

Reference

- https://resources.boomi.com/
- https://help.boomi.com/
- https://www.youtube.com/

TGH

Making Integrations Simpler

boomi Partner

TGH Software Solutions Pvt. Ltd.

www.techygeekhub.com

At TGH, we specialize in driving digital transformation through seamless Integration Technologies.

Operating as an INTEGRATION FACTORY, we serve as a one-stop shop for all your integration needs. Our expert team is well-versed in enterprise software and legacy system integration, along with leading iPaaS technologies like Boomi, MuleSoft, Workato, OIC, and more.

We're committed to enhancing business processes and solving problems through our integration expertise.

Email address

connect@techygeekhub.com

Phone number

+ 011-40071137

+ 91-8810610395

Our offices

Noida Office

iThum
Plot No -40, Tower A,
Office No: 712,
Sector-62, Noida,
Uttar Pradesh, 201301

Hyderabad Office

Plot no: 6/3, 5th Floor,
Techno Pearl Building,
HUDA Techno Enclave,
HITEC City, Hyderabad,
Telangana 500081