TECHNICAL DOCUMENTATION REPORT ON

JSON TO EXCEL CONVERSION

SMART SERVICE PLUGIN

By

T. Saipriya (M0230)

K. Teja Prathap Reddy (M0223)

**INDEX**

[**1. SMART SERVICE PLUGIN :**](#_50gxlmkn2yun) **3**

[1.1 CREATING THE PLUG-IN :](#_j6zgci2xqz6n) 3

[**2. JSON TO EXCEL CONVERSION ;**](#_b7tmibvjwyxk) **6**

[2.1 INPUTS :](#_wq4dmdevvovy) 6

[2.2 PaletteInfo Annotation](#_twjetlok3vjh) 7

[2.3 Actual Class implementation](#_8rulb8dwp909) 7

[2.4 create en\_us.properties file](#_f16jwxz0v8f1) 9

[2.5 create appian-plugin.xml file](#_mceqnt5ueaqg) 9

[2.6 Add the necessary external libraries to META-INF - lib like given below](#_4v4cobvvb97c) 10

[**3. HOW TO USE THE DEPLOYED PLUGIN IN APPIAN ENVIRONMENT ;**](#_o73iiid5b2tx) **11**

[3.1 Open Appian Process Modeler](#_rq4bl813eeyn) 11

JSON TO EXCEL SMART SERVICE PLUGIN

# SMART SERVICE PLUGIN :

Smart service plug-ins are reusable code objects built using annotated Java classes, XML, and resource bundles. They can be used within your Process Models in the same manner as any other smart service, appearing in the Process Modeler for use in any process application.

## 1.1 CREATING THE PLUG-IN :

1. Create a new Java project in Eclipse.

* Click **File** > **New** > **Other…**.
* Select **Java Project** from the **Select a wizard** options. Click **Next**.
* Type a name for your project. Click **Finish** (accepting the default settings).

1. Configure the Java Build Path.

* Right-click the project. Click **Properties**.
* In the left navigation, select **Java Build Path**.
* Select the **Libraries** tab. Click the **Add External JARs…** button.
* Add the following Appian JAR as an external dependency and click **OK**.

<APPIAN\_HOME>/\_admin/sdk/appian-plug-in-sdk.jar

Your plug-in must be designed to access only the classes and methods documented in the [Public API javadocs](https://docs.appian.com/suite/help/22.4/APIs.html).

1. Configure Project Folders.

* In the Package Explorer (left navigation) right-click the src folder.
* Select **New** > **Folder**.
* Type META-INF in the **folder name** field and click **Finish**.
* With the META-INF folder selected, right-click and select **New** > **Folder**.
* Type lib in the **folder name** field and click **Finish**.
* With the src folder selected, right-click and select **New** > **Package**.
* Type your desired package structure in the **name** field.

For example: com.example.plugins.<YOUR\_PLUGIN>

Click **Finish**.

Your file structure should appear similar to the following:

|\_src

|\_com.example.plugins.<YOUR\_PLUGIN>

|

|\_ META-INF

|\_ lib

1. Add the JAR files required by your smart service to the src/META-INF/lib/ folder.

* These JAR files are deployed by your plug-in.
* If your plug-in needs a resource included in an Appian JAR file, do not add that file to this folder. Instead, reference the Appian JAR from your build path.
* Any third party libraries that are specifically used by your plug-in must be included in the src/META-INF/lib/ folder.

1. Create your class and add it to the src/com.example.plugins.<YOUR\_PLUGIN> package.

* Right-click the project and select **New** > **Class**.
* Enter the package for the class (such as com.example.pluginname).
* Enter the name of the class (the smart service name).
* In superclass, click **Browse**.
* Type AppianSmartService — OR — Type com.appiancorp.suiteapi.process.framework.AppianSmartService.
* Ensure that the **Inherited Abstract Methods** checkbox is selected and click **Finish**.

**Note:** Do not use Java 8 constructs (streams, lambdas, etc.) in the code for your custom plug-in. Doing so will cause the plug-in to fail to deploy.

1. Only use Appian's public Java API to invoke Appian functionality. Generally public interfaces are found in com.appiancorp.suiteapi.
2. Update your Java Build Path to include any new JAR files; otherwise, Eclipse won't compile.

* Right-click the project and select **Properties**.
* In the Package Explorer (left navigation) click **Java Build Path**.
* On the **Libraries** tab, click **Add JARs…**.

1. Register the smart service in an appian-plugin.xml file.
2. Add your appian-plugin.xml file at the root level.
3. Add your internationalization bundles to the src/com.example.plugins.<YOUR\_PLUGIN> package.
4. Export your project as a JAR file.

* Right-click your project and click **Export…**.
* Select the **JAR file** option as the **Export destination**.
* On the **Resources to export** dialog, clear the **.classpath** and **.project** selections as these files are used exclusively by Eclipse.
* Select the **\_admin/plugins** folder of your installation directory for your export destination. This directory is created during application server startup.

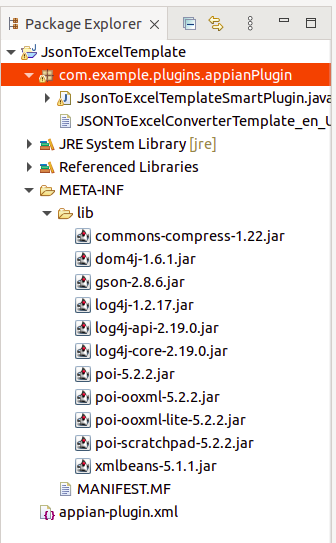
1. Click Finish.

Your plug-in is deployed. The plug-ins framework locates the new JAR file and deploys your custom smart service when the application server starts.

SMART SERVICE PLUGIN EXAMPLE :

# 2. JSON TO EXCEL CONVERSION ;

Json to excel conversion plugin file structure should appear similar to the following:



## 2.1 INPUTS :

**1.The @Order annotation accepts a list of input and output names in the order they should appear in the data tab. If not present, inputs appear in alphabetical order.**

**In this plugin we are using the following inputs.,**

@Order({ "Json", "InputExcelTemplate", "SheetNumber", "RowNumber", "HeaderFlag", "SaveInFolder", "OutputExcelDocumentName","SheetName" })

1. **JSON** : In this we are accepting JSON data as string which will come from a stored procedure from APPIAN.
2. **InputExcelTemplate** : If will pass null value to input excel template , plugin will generate new excel sheet else if will pass id to input excel template plugin will load the json data to given excel template .
3. **SheetNumber** : SheetNumber is only applicable for which has input excel template. If sheet number is given then data will be inserted in the given sheet number, if not by default sheet number is 1 then data will be inserted into the first sheet .
4. **RowNumber** : RowNumber is only applicable for which has input excel template , by default row number is 2 else If any row number is given then data will be inserted from given row number.
5. **HeaderFlag** : By default header flag is true , header flag should be true if headers are required else header flag should be false in that case data should be inserted without headers.
6. **SaveInFolder** : Have to give the path in which folder you want to save the file
7. **OutputExcelDocumentName** : Need to give the output excel file name , by default the output excel document name should be “output”.
8. **SheetName** : Have to give whatever output sheet name you want, else By default sheet number will be given “sheet1”
9. **Finish**

## 2.2 PaletteInfo Annotation

The following attributes of the @PaletteInfo annotation define the palette placement for the smart service:

* **paletteCategory**: The Process Modeler palette category (folder) where the smart service appears; if it's not already present, it is created. If the category palette doesn't exist, it is mapped to **Automation Smart Services**. (This information is not internationalized.)
* **palette**: the palette (subcategory) where the smart service appears; if it's not already present, it is created. (This information is not internationalized.)
* A set of convenience annotations that extend PaletteInfo are defined for the standard smart service palette categories. The table below describes the standard palette annotations that you can use.
* Use the following paletteCategory and palette values when you need to deprecate a custom smart service: @PaletteInfo(paletteCategory="#Deprecated#", palette="#Deprecated#"). Deprecated services will not appear in the palette, only via search.

## 2.3 Actual Class implementation

public class JsonToExcelConverter extends AppianSmartService {

private String sheetName;

public JsonToExcelConverter(ContentService cs) {

super();

this.cs = cs;

}

@Override

public void validate(MessageContainer msg) {

// validations

}

@Override

public void onSave(MessageContainer msg) {

}

public void run() throws SmartServiceException {

/\*

\* This example uses scs.getSystemSecuredValue to get the site-wide credentials

\* for the external system. To get the map of per-user credentials, use

\* scs.getUserSecuredValues

\*/

}

@Input(required = Required.OPTIONAL, defaultValue = "Sheet1")

@Name("SheetName")

public void setSheetName(String sheetName) {

this.sheetName = sheetName;

}

}

## 2.4 create en\_us.properties file

Example :

#Properties of XmlToJson Smart Service#

name = JSON To Excel Converter

input.SheetName.comment: Result excel sheet name(non excel template scenario)

## 2.5 create appian-plugin.xml file

Example :

<appian-plugin name="Json To Excel Smart Plugin Converter" key="com.example.plugins.appianPlugin">

<plugin-info>

<description>Json To Excel conversion Smart Plugin</description>

<vendor name="Machint" url="https://www.machint.com/" />

<version>1.1.5</version>

<application-version min="10.0" />

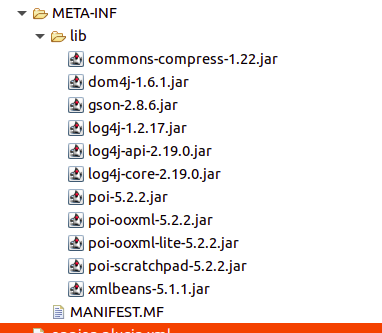
</plugin-info>

<!-- Smart Services -->

<smart-service name="JSON To Excel Converter Template" key="JSONToExcelConverterTemplate" class="com.example.plugins.appianPlugin.JsonToExcelTemplateSmartPlugin" />

</appian-plugin>

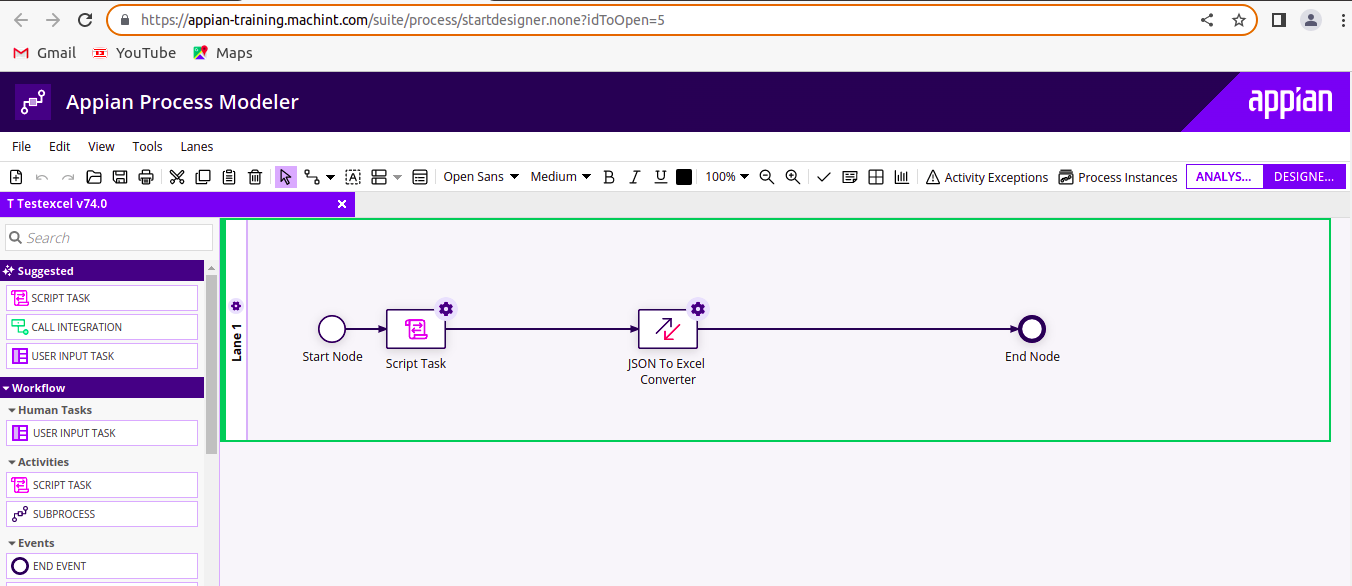
## 2.6 Add the necessary external libraries to META-INF - lib like given below



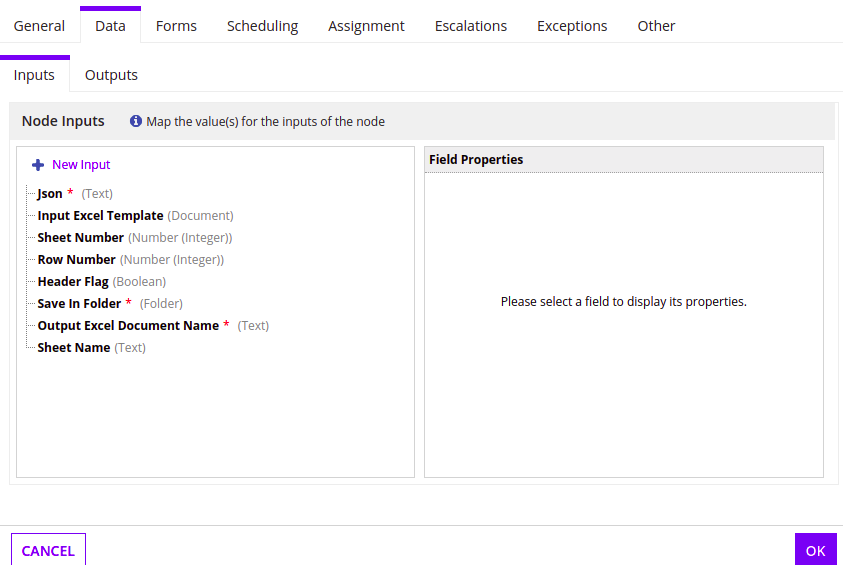
* Exporting the json to excel conversion project as a JAR file.
* Json to excel conversion .jar is deployed. The plug-ins framework locates the new JAR file and deploys your custom smart service when the application server starts.

# 3. HOW TO USE THE DEPLOYED PLUGIN IN APPIAN ENVIRONMENT

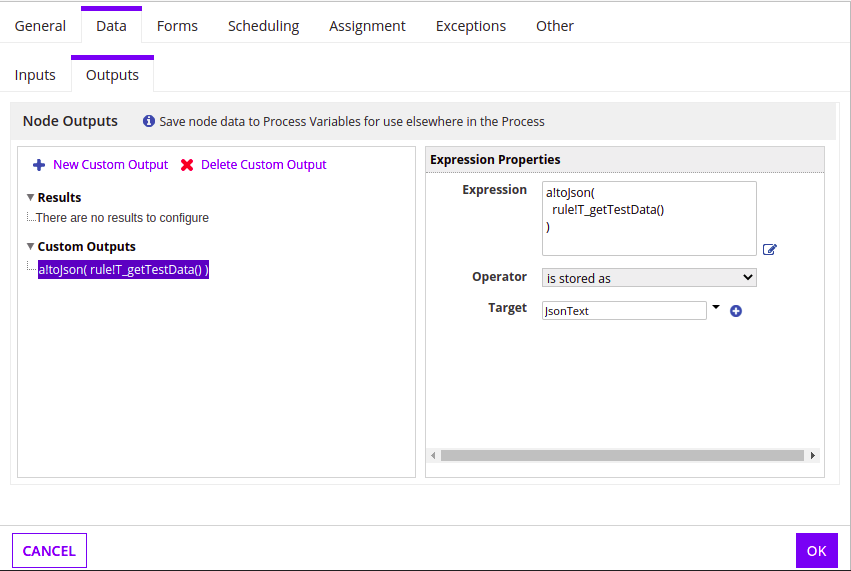
## 3.1 Open Appian Process Modeler (<https://appian-training.machint.com/suite/process/startdesigner.none?idToOpen=5>)



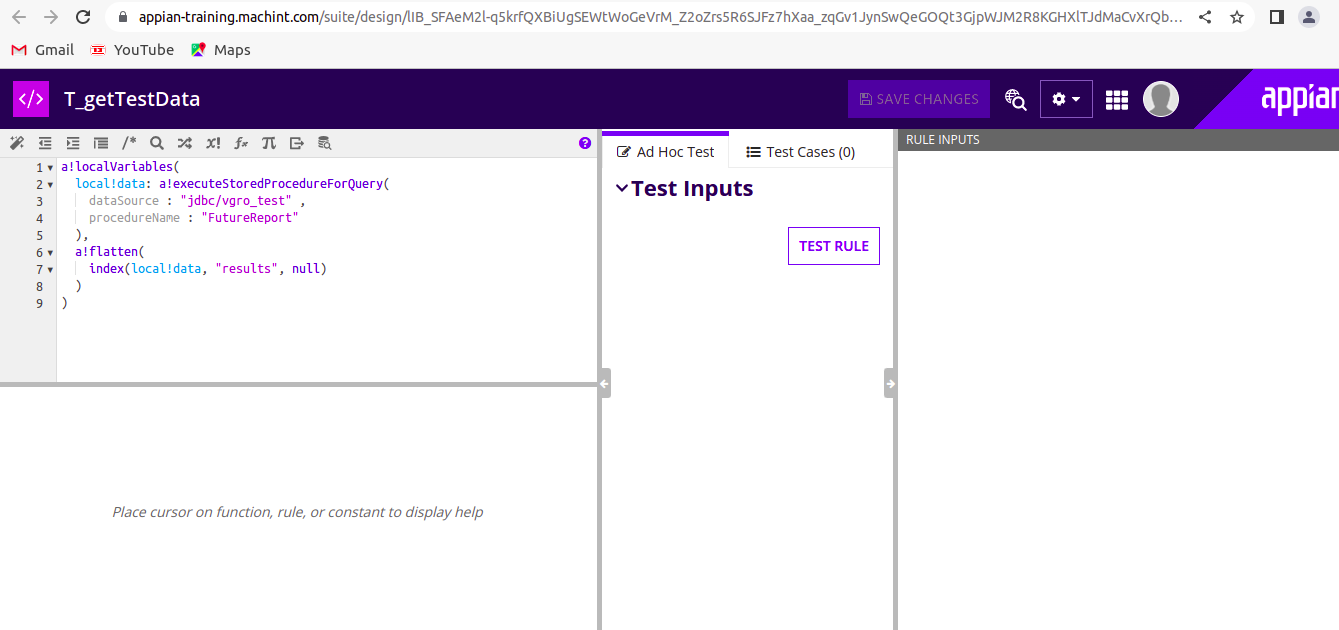
## 3.2 Double click on json To Excel plugin to give inputs



## 3.3 Json input will be given as output of the script task



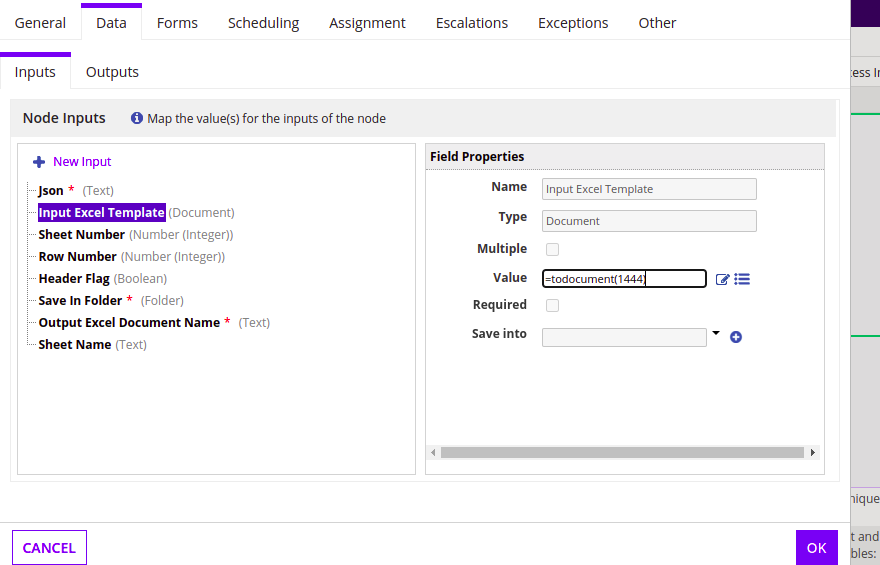
## 3.4 Expression rule is derived from the t\_getTestData() expression given below



## 3.5 Input Excel Template as Input

Second input field to plugin is Input excel template , If will pass null value to input excel template , plugin will generate new excel sheet else if will pass id to input excel template plugin will load the json data to given excel template .

(ex : with template (=todocument(1444)))



## 3.6 Remaining Inputs

Same as input excel template need to give following inputs

* Sheet number
* Row number
* Headerflag
* Save in folder
* Output excel document name
* Sheet name

## 3.7 Click on **OK**

After giving inputs v

## 3.8 Click on file then select save and publish , after saving the process model again click on file then select start process for debugging , plugin will start

## 3.9 Process model started

## 3.10 And can check the duration , process id , process name , node and status in process details

In that process details select the process nodes as given below

