**src/main/resources/script/outputJSON.groovy**

import com.sap.gateway.ip.core.customdev.util.Message;
  
import java.util.HashMap;
  
import groovy.json.\*
  
  
  
class result {
  
 String name
  
 String iD
  
}
  
  
def Message processData(Message message) {
  
  
 //Get Response Payload
  
 def body = message.getBody(String.class);
  
  
 def jsonSlurper = new JsonSlurper()
  
 def json = jsonSlurper.parseText(body)
  
  
 def r = [];
  
 int i = 0;
  
  
 json.d.results.each{
  
 r[i]=new result(name:it.DisplayName, iD: it.TechnicalName);
  
 i++;
  
 }
  
 def output = JsonOutput.toJson(r);
  
  
 message.setBody("{\"IntegrationPackage\":" + output+ "}")
  
  
  
 return message;
  
  
}

**src/main/resources/script/getQueryParameter.groovy**

import com.sap.gateway.ip.core.customdev.util.Message;
  
import java.util.HashMap;
  
import java.net.URLDecoder;
  
  
def Message processData(Message message) {
  
 def qParm = message.getHeaders().get("CamelHttpQuery");
  
 def queryParams = qParm.split('&');
  
 def mapParams = queryParams.collectEntries { param -> param.split('=').collect { URLDecoder.decode(it) }};
  
 def id = mapParams['id'].toString();
  
 def version = mapParams['version'].toString();
  
 def mName = mapParams['mapName'].toString();
  
 if(version == "null" || version == "Draft")
  
 version = "1.0.0";
  
 message.setProperty("ID", id);
  
 message.setProperty("Version", version);
  
 message.setProperty("mName", mName);
  
 message.setHeader("CamelHttpQuery", "");
  
 return message;
  
}

**src/main/resources/script/script8.groovy**

import com.sap.gateway.ip.core.customdev.util.Message
  
import java.util.HashMap
  
import java.text.SimpleDateFormat
  
  
def Message processData(Message message) {
  
  
def staticContent = ""
  
  
 //Parse static content
  
 staticContent = new String(staticContent.decodeBase64(), "UTF-16LE")
  
 def baseUrl = "${message.getHeaders().get('CamelHttpUrl').replace('/iGenTemplate','')}".toString()
  
 //Replace endpoints with dynamic ones
  
 def externURL = message.getHeaders().get('externalHost')
  
  
 staticContent=staticContent.replace("{{SCPI\_Dev\_Tools}}", baseUrl)
  
 staticContent=staticContent.replace("{{SAP\_ADMIN\_MAPPING\_TITLE}}", "SAP CPI - SBX - Developer Tools")
  
 staticContent=staticContent.replace("{{externalHost}}", externURL)
  
  
 //Set output
  
 message.setBody(staticContent.getBytes("UTF-8"));
  
 message.setHeader("Content-Type", "text/html; charset=utf-8");
  
 return message
  
}

**src/main/resources/script/script10.groovy**

import com.sap.gateway.ip.core.customdev.util.Message
  
import com.sap.gateway.ip.core.customdev.processor.MessageImpl
  
import groovy.xml.MarkupBuilder
  
import java.util.zip.\*
  
import javax.xml.\*
  
import java.text.\*
  
import java.util.ArrayList;
  
import groovy.util.\*
  
  
  
import groovy.util.slurpersupport.GPathResult;
  
import java.util.HashMap;
  
import java.util.Date;
  
import static java.util.Calendar.\*
  
import groovy.transform.AutoClone
  
import groovy.time.TimeCategory
  
import groovy.json.\*
  
  
  
import org.w3c.dom.Document
  
import org.w3c.dom.Element
  
import org.w3c.dom.Node
  
import org.w3c.dom.NodeList
  
  
import com.google.gson.Gson;
  
import com.google.gson.GsonBuilder;
  
  
  
import javax.xml.parsers.DocumentBuilder
  
import javax.xml.parsers.DocumentBuilderFactory
  
  
  
def Message processData(Message message) {
  
 //Body
  
 def name=message.getProperty("fileName").toString()
  
 def body = message.getBody(java.lang.String);
  
 def xml = new XmlSlurper().parseText(body);
  
 def exist = false
  
 for(element in xml.entry){
  
 def currUrl = element.id
  
 for(properties in element.properties){
  
 def fileName= properties.DisplayName
  
 if(fileName == name){
  
 exist = true
  
 }
  
 }
  
  
 }
  
  
 if(exist == true){
  
 message.setProperty("exist","yes")// Url that will be call to download the file
  
 }else{
  
 message.setProperty("exist","no")// Url that will be call to download the file
  
 }
  
  
 return message
  
}

**src/main/resources/script/output\_IPackage.groovy**

import com.sap.gateway.ip.core.customdev.util.Message;
  
import java.util.HashMap;
  
import groovy.json.\*
  
  
  
class result {
  
 String name
  
 String iD
  
}
  
  
def Message processData(Message message) {
  
  
 //Get Response Payload
  
 def body = message.getBody(String.class);
  
  
 def jsonSlurper = new JsonSlurper()
  
 def json = jsonSlurper.parseText(body)
  
  
 def r = [];
  
 int i = 0;
  
  
 json.d.results.each{
  
 if(it.Mode.equals("EDIT\_ALLOWED"))
  
 {
  
 r[i]=new result(name:it.Name, iD: it.Id);
  
 i++;
  
 }
  
 }
  
 def output = JsonOutput.toJson(r);
  
  
 message.setBody("{\"IntegrationPackage\":" + output+ "}");
  
 def messageLog = messageLogFactory.getMessageLog(message);
  
 if(messageLog != null){
  
 messageLog.addAttachmentAsString("Log package results", "{\"IntegrationPackage\":" + output+ "}".toString(), "text/plain");
  
 }
  
 return message;
  
  
}

**src/main/resources/script/script3.groovy**

/\*  
 The integration developer needs to create the method processData   
 This method takes Message object of package com.sap.gateway.ip.core.customdev.util   
which includes helper methods useful for the content developer:  
The methods available are:  
 public java.lang.Object getBody()  
 public void setBody(java.lang.Object exchangeBody)  
 public java.util.Map<java.lang.String,java.lang.Object> getHeaders()  
 public void setHeaders(java.util.Map<java.lang.String,java.lang.Object> exchangeHeaders)  
 public void setHeader(java.lang.String name, java.lang.Object value)  
 public java.util.Map<java.lang.String,java.lang.Object> getProperties()  
 public void setProperties(java.util.Map<java.lang.String,java.lang.Object> exchangeProperties)   
 public void setProperty(java.lang.String name, java.lang.Object value)  
 public java.util.List<com.sap.gateway.ip.core.customdev.util.SoapHeader> getSoapHeaders()  
 public void setSoapHeaders(java.util.List<com.sap.gateway.ip.core.customdev.util.SoapHeader> soapHeaders)   
 public void clearSoapHeaders()  
 \*/  
import com.sap.gateway.ip.core.customdev.util.Message;  
import java.util.HashMap;  
def Message processData(Message message) {  
 //Body   
 def body = message.getBody(java.lang.String);  
 message.setProperty("configurations", body);  
 return message;  
}

**src/main/resources/script/script9.groovy**

import com.sap.gateway.ip.core.customdev.util.Message;
  
import java.util.HashMap;
  
import java.net.URLDecoder;
  
  
def Message processData(Message message) {
  
 def qParm = message.getHeaders().get("CamelHttpQuery");
  
 def queryParams = qParm.split('&');
  
 def fileParams = queryParams.collectEntries { param -> param.split('=').collect { URLDecoder.decode(it) }};
  
 def packageId = fileParams['packageId'].toString();
  
 def fName = fileParams['fileName'].toString();
  
 message.setProperty("packageId", packageId);
  
 message.setProperty("fileName", fName);
  
 message.setHeader("CamelHttpQuery", "");
  
 return message;
  
}

**src/main/resources/script/output\_IFlow.groovy**

import com.sap.gateway.ip.core.customdev.util.Message;
  
import java.util.HashMap;
  
import groovy.json.\*
  
  
  
class result {
  
 String name
  
 String iD
  
 //String type
  
 String Version
  
}
  
  
def Message processData(Message message) {
  
  
 //Get Response Payload
  
 def body = message.getBody(String.class);
  
  
 def messageLog = messageLogFactory.getMessageLog(message);
  
 if(messageLog != null){
  
 messageLog.addAttachmentAsString("Log iflows results", body.toString(), "text/plain");
  
 }
  
  
 def jsonSlurper = new JsonSlurper()
  
 def json = jsonSlurper.parseText(body)
  
  
 def output = " [{\"iD\": \"STOP\",\"name\": \"No IFlow in Selected Package\"}]"
  
  
 def r = [];
  
 int i = 0;
  
  
 json.d.results.each{
  
 r[i]=new result(name:it.Name, iD: it.Id/\*, type: it.Type\*/, Version: it.Version);
  
 i++;
  
 }
  
  
 if(i>0)
  
 output = JsonOutput.toJson(r);
  
  
 message.setBody("{\"IntegrationFlow\":" + output+ "}");
  
  
 return message;
  
  
}

**src/main/resources/script/script7.groovy**

import com.sap.gateway.ip.core.customdev.util.Message;
  
import java.util.HashMap;
  
import java.net.URLDecoder;
  
  
def Message processData(Message message) {
  
 def qParm = message.getHeaders().get("CamelHttpQuery");
  
 def queryParams = qParm.split('&');
  
 def fileParams = queryParams.collectEntries { param -> param.split('=').collect { URLDecoder.decode(it) }};
  
 def packageId = fileParams['packageId'].toString();
  
 def fName = fileParams['fileName'].toString();
  
 message.setProperty("packageId", packageId);
  
 message.setProperty("fileName", fName);
  
 message.setHeader("CamelHttpQuery", "");
  
 return message;
  
}

**src/main/resources/script/script6.groovy**

import com.sap.gateway.ip.core.customdev.util.Message
  
import com.sap.gateway.ip.core.customdev.processor.MessageImpl
  
import groovy.xml.MarkupBuilder
  
import java.util.zip.\*
  
import javax.xml.\*
  
import java.text.\*
  
import java.util.ArrayList;
  
import groovy.util.\*
  
  
  
import groovy.util.slurpersupport.GPathResult;
  
import java.util.HashMap;
  
import java.util.Date;
  
import static java.util.Calendar.\*
  
import groovy.transform.AutoClone
  
import groovy.time.TimeCategory
  
import groovy.json.\*
  
  
  
import org.w3c.dom.Document
  
import org.w3c.dom.Element
  
import org.w3c.dom.Node
  
import org.w3c.dom.NodeList
  
  
import com.google.gson.Gson;
  
import com.google.gson.GsonBuilder;
  
  
  
import javax.xml.parsers.DocumentBuilder
  
import javax.xml.parsers.DocumentBuilderFactory
  
  
  
def Message processData(Message message) {
  
 //Body
  
 def name=message.getProperty("fileName").toString()
  
 def body = message.getBody(java.lang.String);
  
 def xml = new XmlSlurper().parseText(body);
  
 def url =""
  
 for(element in xml.entry){
  
 def currUrl = element.id
  
 for(properties in element.properties){
  
 def fileName= properties.DisplayName
  
 if(fileName == name){
  
 url = currUrl
  
 }
  
 }
  
  
 }
  
 //message.setBody("Url",url.toString() + '/$value')
  
 //message.setBody(name)
  
 message.setProperty("Url",url.toString() + '/$value')// Url that will be call to download the file
  
 return message
  
}

**src/main/resources/script/StaticContent.groovy**

import com.sap.gateway.ip.core.customdev.util.Message
  
import java.util.HashMap
  
import java.text.SimpleDateFormat
  
  
def Message processData(Message message) {
  
  
def staticContent = ""
  
  
 //Parse static content
  
 staticContent = new String(staticContent.decodeBase64(), "UTF-16LE")
  
 def baseUrl = "${message.getHeaders().get('CamelHttpUrl').replace('/main','')}".toString()
  
 if (baseUrl.startsWith("http")) {
  
 baseUrl = "https" + baseUrl.substring(4)
  
 }
  
 def externURL = message.getHeaders().get('externalHost')
  
 def templateName = message.getHeaders().get('templateName')
  
 //Replace endpoints with dynamic ones
  
 staticContent=staticContent.replace("{{SCPI\_Dev\_Tools}}", baseUrl)
  
 staticContent=staticContent.replace("{{SAP\_ADMIN\_MAPPING\_TITLE}}", "SAP CPI - SBX - Developer Tools")
  
 staticContent=staticContent.replace("{{externalHost}}", externURL)
  
 staticContent=staticContent.replace("{{TemplateName}}", templateName)
  
  
 //Set output
  
 message.setBody(staticContent.getBytes("UTF-8"));
  
 message.setHeader("Content-Type", "text/html; charset=utf-8");
  
 return message
  
}

**src/main/resources/script/script4.groovy**

import com.sap.gateway.ip.core.customdev.util.Message;  
import java.util.HashMap;  
import javax.xml.\*  
import groovy.json.\*  
import groovy.util.slurpersupport.GPathResult;  
  
def Message processData(Message message) {  
 //Body   
 def runtime = message.getBody(java.lang.String);  
 //def header = message.getProperty("header")  
 //def configurations = message.getProperty("configurations")  
   
   
 //def builder = new JsonBuilder(runtime)  
 //def elements = new JsonSlurper().parseText( header )  
 /\*  
 builder{  
 //header data  
 for(element in elements){  
 "${element.key}"("${element.value}")  
 }  
   
 //configuration data  
 elements = new JsonSlurper().parseText(configurations)  
 for(element in elements){  
 "${element.key}"{"${element.value}"}  
 }  
   
 //runtime data  
 elements = new JsonSlurper().parseText(runtime)  
 for(element in elements){  
 "${element.key}"("${element.value}")  
 }  
   
 }\*/  
   
   
 message.setProperty("runtime", runtime);  
   
 return message;  
}

**src/main/resources/script/output\_IMapContent.groovy**

import javax.xml.parsers.DocumentBuilder
  
import javax.xml.parsers.DocumentBuilderFactory
  
  
import org.w3c.dom.Document
  
import org.w3c.dom.Element
  
import org.w3c.dom.Node
  
import org.w3c.dom.NodeList
  
  
import com.google.gson.Gson;
  
import com.google.gson.GsonBuilder;
  
import com.sap.gateway.ip.core.customdev.util.Message;
  
  
import org.apache.commons.compress.archivers.ArchiveException;
  
import org.apache.commons.compress.archivers.ArchiveInputStream;
  
import org.apache.commons.compress.archivers.ArchiveStreamFactory;
  
import org.apache.commons.codec.binary.Base64;
  
  
  
  
public class docHeader {
  
  
 private String sourceFields ="";
  
 private String constantValues ="";
  
 private String functionS ="";
  
 private String targetField ="";
  
  
 public void setsourceFields(String name) {
  
 this.sourceFields = this.sourceFields.concat(name);
  
 }
  
  
 public void setconstantValues(String name) {
  
 this.constantValues = this.constantValues.concat(name);
  
 }
  
  
 public void setfunctionS(String name) {
  
 this.functionS = this.functionS.concat(name);
  
 }
  
  
 public void settargetField(String name) {
  
 this.targetField = this.targetField.concat(name);
  
 }
  
}
  
  
def Message processData(Message message) {
  
  
 //Get Response Payload
  
 def body = message.getBody(String.class);
  
  
 InputStream inpStream = new ByteArrayInputStream(body.getBytes());
  
  
 Document document = processxml(inpStream);
  
  
 // Normalize the XML Structure; It's just too important !!
  
 document.getDocumentElement().normalize();
  
  
 // Here comes the root node
  
 Element root = document.getDocumentElement();
  
  
  
 Element content = (Element) root.getElementsByTagNameNS("urn:sap-com:xi", "content").item(0);
  
 Element metaData = (Element) content.getElementsByTagNameNS("urn:sap-com:xi:mapping:xitrafo","MetaData").item(0);
  
  
 String mapData;
  
  
 if(metaData.getElementsByTagName("mappingtool").getLength() > 0 ) {
  
 Element transformation = getTransformationTag(metaData);
  
 mapData = getMapData(transformation);
  
 }
  
 else if(metaData.getElementsByTagNameNS("urn:sap-com:xi:mapping:xitrafo", "blob").getLength()>0) {
  
  
 byte[] buffer = new byte[1024];
  
 String b64data = metaData.getElementsByTagNameNS("urn:sap-com:xi:mapping:xitrafo", "blob").item(0).getTextContent();
  
 b64data = b64data.replace("!zip!", "");
  
 byte[] decCon = Base64.decodeBase64(b64data);
  
  
 ByteArrayOutputStream oStream = getunzipData(decCon);
  
  
 ByteArrayOutputStream oStream1 = getunzipData(oStream.toByteArray());
  
  
 InputStream newinpStream = new ByteArrayInputStream(oStream1.toByteArray());
  
  
 Document doc1 = processxml(newinpStream);
  
 doc1.getDocumentElement().normalize();
  
 Element root1 = doc1.getDocumentElement();
  
 Element transformation1 = getTransformationTag(root1);
  
 mapData = getMapData(transformation1);
  
 }
  
  
 message.setHeader("Content-Type", "application/json");
  
 message.setBody("{\"mappingResult\":" + mapData + "}");
  
  
 return message;
  
  
}
  
  
def Document processxml(InputStream inpStream) throws Exception {
  
  
 DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
  
 factory.setNamespaceAware(true);
  
 DocumentBuilder builder = factory.newDocumentBuilder();
  
  
 Document document = builder.parse(inpStream);
  
  
 return document;
  
}
  
  
def Element getTransformationTag(Element root) {
  
  
 NodeList transf = root.getElementsByTagName("transformation");
  
 Element transformation = (Element) transf.item(0);
  
 return transformation;
  
}
  
  
def String getMapData(Element transformation) {
  
  
 NodeList nList1 = transformation.getChildNodes();
  
  
  
 def dh = [];
  
  
 int resar = 0;
  
  
 for (int temp = 0; temp < nList1.getLength(); temp++) {
  
  
 int sf = 0;
  
 int fr = 0;
  
 int cn = 0;
  
  
  
 Node node = nList1.item(temp);
  
  
 if(node.getNodeType() == Node.ELEMENT\_NODE) {
  
  
 dh[resar] = new docHeader();
  
 Element eElement = (Element) node;
  
  
 NodeList nList2 = eElement.getElementsByTagName("brick");
  
  
 for (int temp1 = 0; temp1 < nList2.getLength(); temp1++) {
  
  
 Node node1 = nList2.item(temp1);
  
 Element eElement1 = (Element) node1;
  
  
 if(eElement1.getAttribute("type").equals("Src")) {
  
 if(sf!=0)
  
 dh[resar].setsourceFields("\n");
  
 dh[resar].setsourceFields(eElement1.getAttribute("path"));
  
 sf++;
  
 }
  
 else if(eElement1.getAttribute("type").equals("Func") && eElement1.getAttribute("fname").equals("const")) {
  
 if(cn!=0)
  
 dh[resar].setconstantValues("\n");
  
 dh[resar].setconstantValues("Constant -" + eElement.getElementsByTagName("value").item(0).getTextContent());
  
 cn++;
  
 }
  
 else {
  
 String value=eElement1.getAttribute("fname");
  
 if(fr!=0 && !value.isEmpty())
  
 dh[resar].setfunctionS("\n");
  
 dh[resar].setfunctionS(value);
  
 fr++;
  
 }
  
  
 }
  
 dh[resar].settargetField(eElement.getAttribute("path"));
  
 resar++;
  
  
 }
  
  
 }
  
  
 Gson gson = new GsonBuilder().setPrettyPrinting().create();
  
 return gson.toJson(dh);
  
  
}
  
  
def ByteArrayOutputStream getunzipData(byte[] inp) throws Exception {
  
  
 byte[] buffer = new byte[1024];
  
  
 InputStream newinpStream = new ByteArrayInputStream(inp);
  
  
 ByteArrayOutputStream oStream = new ByteArrayOutputStream();
  
  
 ArchiveInputStream ais = new ArchiveStreamFactory().createArchiveInputStream("zip", newinpStream);
  
 ais.getNextEntry();
  
 int len;
  
 while ((len = ais.read(buffer)) > 0) {
  
 oStream.write(buffer, 0, len);
  
 }
  
  
 return oStream;
  
}

**src/main/resources/script/script2.groovy**

/\*  
 The integration developer needs to create the method processData   
 This method takes Message object of package com.sap.gateway.ip.core.customdev.util   
which includes helper methods useful for the content developer:  
The methods available are:  
 public java.lang.Object getBody()  
 public void setBody(java.lang.Object exchangeBody)  
 public java.util.Map<java.lang.String,java.lang.Object> getHeaders()  
 public void setHeaders(java.util.Map<java.lang.String,java.lang.Object> exchangeHeaders)  
 public void setHeader(java.lang.String name, java.lang.Object value)  
 public java.util.Map<java.lang.String,java.lang.Object> getProperties()  
 public void setProperties(java.util.Map<java.lang.String,java.lang.Object> exchangeProperties)   
 public void setProperty(java.lang.String name, java.lang.Object value)  
 public java.util.List<com.sap.gateway.ip.core.customdev.util.SoapHeader> getSoapHeaders()  
 public void setSoapHeaders(java.util.List<com.sap.gateway.ip.core.customdev.util.SoapHeader> soapHeaders)   
 public void clearSoapHeaders()  
 \*/  
import com.sap.gateway.ip.core.customdev.util.Message;  
import java.util.HashMap;  
def Message processData(Message message) {  
 //Body   
 def body = message.getBody(java.lang.String);  
 message.setProperty("header", body);  
 return message;  
}

**src/main/resources/script/script5.groovy**

import com.sap.gateway.ip.core.customdev.util.Message;
  
import java.util.HashMap;
  
import java.net.URLDecoder;
  
  
def Message processData(Message message) {
  
 def qParm = message.getHeaders().get("CamelHttpQuery");
  
 def queryParams = qParm.split('&');
  
 def mapParams = queryParams.collectEntries { param -> param.split('=').collect { URLDecoder.decode(it) }};
  
 def id = mapParams['id'].toString();
  
 def version = mapParams['version'].toString();
  
 def resource = mapParams['resource'].toString();
  
 def tName = mapParams['templateName'].toString();
  
 def tID = mapParams['templateID'].toString();
  
 def mName = mapParams['mapName'].toString();
  
 if(version == "null" || version == "Draft")
  
 version = "1.0.0";
  
 message.setProperty("ID", id);
  
 message.setProperty("resource", resource);
  
 message.setProperty("Version", version);
  
 message.setProperty("mName", mName);
  
 message.setProperty("tName", tName);
  
 message.setProperty("tID", tID);
  
 message.setHeader("CamelHttpQuery", "");
  
 return message;
  
}

**src/main/resources/script/script1.groovy**

import com.sap.gateway.ip.core.customdev.util.Message
  
import com.sap.gateway.ip.core.customdev.processor.MessageImpl
  
import groovy.xml.MarkupBuilder
  
import java.util.zip.\*
  
import javax.xml.\*
  
import java.text.\*
  
import java.util.ArrayList;
  
import groovy.util.\*
  
  
  
import com.sap.it.api.ITApiFactory;
  
import com.sap.it.api.securestore.SecureStoreService;
  
import com.sap.it.api.securestore.UserCredential;
  
  
import groovy.util.slurpersupport.GPathResult;
  
import java.util.HashMap;
  
import java.util.Date;
  
import static java.util.Calendar.\*
  
import groovy.transform.AutoClone
  
import groovy.time.TimeCategory
  
import groovy.json.\*
  
  
  
import org.w3c.dom.Document
  
import org.w3c.dom.Element
  
import org.w3c.dom.Node
  
import org.w3c.dom.NodeList
  
  
import com.google.gson.Gson;
  
import com.google.gson.GsonBuilder;
  
  
import javax.xml.parsers.\*
  
import java.nio.file.\*;
  
import java.util.Base64;
  
import java.io.BufferedInputStream;
  
  
import java.nio.charset.StandardCharsets
  
import org.codehaus.groovy.runtime.EncodingGroovyMethods
  
  
class GlobalObj{
  
  
 HashMap<String,String> HM\_ObjectPromt = [ 'Enricher':'Content Modifier',
  
 'ProcessCallElement':'ProcessCallElement',
  
 'activityType':'activityType',
  
 'Aggregator':'Aggregator',
  
 'Splitter':'Splitter',
  
 'Gather':'Gather',
  
 'XmlToCsvConverter':'XmlToCsvConverter',
  
 'DBstorage':'Data Storage'
  
  
 ];
  
  
 HashMap<String,String> HM\_Runtime = [
  
 'name': 'Name',
  
 'type': 'Type',
  
 'status': 'Status',
  
 'deployedBy':'Deployed By',
  
 'deployedOn':'Deployed On'
  
  
  
 ];
  
  
 HashMap<String,String> HM\_Header = [
  
 'id':'Iflow Id',
  
 'name': 'Iflow Name',
  
 'version': 'Iflow Version',
  
 'description':'Description',
  
 'packageId': 'Package Id'
  
  
 ];
  
  
 HashMap<String,String> HM\_Configurations = [
  
 'ParameterKey':'ParameterKey',
  
 'ParameterValue': 'ParameterValue',
  
 'DataType': 'DataType'
  
  
 ];
  
  
 HashMap<String,HashMap<String,String>>HM\_Process =[
  
 'Enricher':['propertyTable':'Properties','headerTable':'Headers','wrapContent':'Body'],
  
 'ProcessCallElement':['subActivityType':'Type of Process'],
  
 'activityType':['subActivityType':'Type'],
  
 'Aggregator':['incomingFormat':'incomingFormat','aggregationAlgorithm':'Aggregation Algorithm','messageSequenceExpression':'messageSequenceExpression','lastMessageCondition':'Message Condition','correlationExpression':'correlationExpression','datastoreName':'datastoreName'],
  
 'Splitter':['exprType':'Expression Type ','splitExprValue':'Split Value','splitType':'Split Type'],
  
 'Gather':['messageType':'Message Type','aggregationAlgorithm':'Aggregation Algorithm'],
  
 'XmlToCsvConverter':['Field\_Separator\_in\_CSV':'Field\_Separator\_in\_CSV','XPath\_Field\_Location':'XPath\_Field\_Location'],
  
 'DBstorage':['visibility':'Visibility','encrypt':'Encrypt','messageId':'Message ID','override':'Override','operation':'Operation','storageName':'Storage Name']
  
 ];
  
  
 HashMap<String,HashMap<String,String>>HM\_collaboration = [
  
 'HTTP':['httpMethod':'Method','ComponentType':'Component Type','httpAddressQuery':'Query','direction':'direction','TransportProtocol':'Transport Protocol','httpAddressWithoutQuery':'Address','credentialName':'User Credential Name'],
  
 'SOAP':['MessageProtocol':'Message Protocol','address':'Address','userRole':'User Role','senderAuthType':'Sender Auth Type','direction':'direction','soapServiceName':'Service Name','soapWsdlPortName':'Wsdl Port Name','operationName':'Operation Name','privateKeyAlias':'privateKeyAlias','soapWsdlURL':'Wsdl URL'],
  
 'Mail':['server':'Address','body':'Body','from':'From','direction':'direction','ComponentType':'Component Type','to':'To','user':'User Credential Name'],
  
 'HCIOData':['resourcePath':'Resource Path','customQueryOptions':'Query','MessageProtocol':'Message Protocol','direction':'direction','ComponentType':'Component Type','address':'Address','TransportProtocol':'Transport Protocol','operation':'Operation'],
  
 'HTTPS':['TransportProtocol':'Transport Protocol','urlPath':'URL Path','userRole':'User Credential Name','direction':'direction'],
  
 'SFTP':['path':'Path','host':'Host','connectTimeout':'Connect Timeout','authentication':'Authentication','credential\_name':'User Credential Name','fileName':'File Name','direction':'direction','doneFileName':'doneFileName','noop':'noop','file.move':'file.move','scheduleKey':'scheduleKey'],
  
 'IDOC':['ComponentType':'Component Type','address':'Address','system':'System','TransportProtocol':'Transport Protocol','MessageProtocol':'Message Protocol','direction':'direction'],
  
 'ProcessDirect':['ComponentType':'Component Type','address':'Address','system':'System','direction':'direction'],
  
 'XI':['ComponentType':'Component Type','Address':'Address','CommunicationPartyReceiver':'Communication Party Receiver','CommunicationComponentSender':'Communication Component Sender','location\_id':'Location ID','CommunicationComponentReceiver':'Communication Component Receiver','MessageProtocol':'Message Protocol','direction':'direction','AuthenticationType':'Authentication Type','ServiceInterfaceReceiver':'Service Interface Receiver','CommunicationPartySender':'Communication Party Sender','system':'System'],
  
 'RFC':['ComponentType':'Component Type','destination':'Destination','direction':'direction','MessageProtocol':'Message Protocol','system':'System'],
  
 'SF\_SOAP':['ComponentType':'Component Type','MessageProtocol':'Message Protocol','alias':'User Role','urlSuffixSOAP':'URL Suffis','direction':'direction','address':'Address','query':'Query','entity':'Entity']
  
 ];
  
  
}
  
  
HM\_Adapter=new HashMap<String,String>();
  
  
  
class MyZIS extends ZipInputStream {
  
  
 public MyZIS(InputStream input) {
  
 super(input);
  
 }
  
  
 @Override
  
 public void close() throws IOException {
  
 }
  
  
 public void myClose() throws IOException {
  
 super.close();
  
 }
  
}
  
  
  
public class docHeader {
  
  
 private String sourceFields ="";
  
 private String constantValues ="";
  
 private String functionS ="";
  
 private String targetField ="";
  
  
 public void setsourceFields(String name) {
  
 this.sourceFields = this.sourceFields.concat(name);
  
 }
  
  
 public void setconstantValues(String name) {
  
 this.constantValues = this.constantValues.concat(name);
  
 }
  
  
 public void setfunctionS(String name) {
  
 this.functionS = this.functionS.concat(name);
  
 }
  
  
 public void settargetField(String name) {
  
 this.targetField = this.targetField.concat(name);
  
 }
  
}
  
  
  
////////////////////////////////////////
  
class GraphNode{
  
  
 String name;
  
 boolean visited;
  
  
 GraphNode(){}
  
 GraphNode(String name, boolean visited){
  
 this.name = name;
  
 this.visited = visited;
  
 }
  
 GraphNode(String name){
  
 this.name=name;
  
 this.visited = false;
  
 }
  
  
 void visit(){
  
 this.visited = true;
  
 }
  
}
  
  
  
class Graph
  
{
  
 //private int V; // No. of vertices
  
 HashMap<GraphNode, ArrayList<GraphNode>> adj;
  
 ArrayList<String> DFSres;
  
 // Array of lists for Adjacency List Representation
  
  
 // Constructor
  
 Graph() {
  
 adj = new HashMap<>();
  
 DFSres = new ArrayList<>();
  
 }
  
  
 //Function to add an edge into the graph
  
 void addEdge(String key, String value)
  
 {
  
 def node =new GraphNode(key)
  
 def x = adj.find{ it.key.name == key }?.key
  
  
 if(x!= null){
  
 adj[x].add(new GraphNode(value))
  
 }
  
 else{
  
 adj.put(new GraphNode(key),[new GraphNode(value)])
  
 }
  
 }
  
  
 void afis(){
  
  
  
  
 this.adj.each{
  
 key, value -> print("["+key.name+" "+key.visited+"]: ");
  
 value.each{
  
 element-> print(element.name+", ")
  
 }
  
 println()
  
 }
  
  
 }
  
  
 // A function used by DFS
  
 void DFSUtil(String v)
  
 {
  
 // Mark the current node as visited and print it
  
 //define
  
 def x = this.adj.find{ it.key.name == v }?.key
  
 if(x){
  
 //adj.get(x).visited = true;
  
 //this.adj.find{ it.key.name == v }?.key.visited = true;
  
 x.visit();
  
 DFSres.add(x.name);
  
  
 //this.afis();
  
  
 //print(x.name+" ")
  
 // Recur for all the vertices adjacent to this vertex
  
 adj[x].each{element ->
  
 def node = adj.find{ it.key.name == element.name }?.key// we need to check visited from key not from child
  
  
 if(node == null){// this is a leaf
  
 DFSres.add(element.name);
  
 //print(element.name+" ")
  
 }
  
 if(node!= null && node.visited == false){
  
 DFSUtil(element.name)
  
 }
  
 }
  
 }
  
 }
  
  
 // The function to do DFS traversal. It uses recursive DFSUtil()
  
 void DFS(String name)
  
 {
  
  
 // Call the recursive helper function to print DFS traversal
  
 DFSUtil(name);
  
  
 }
  
}
  
  
class Header{
  
  
 String Id;
  
 String Version;
  
 String PackageId;
  
 String Name;
  
 String Description;
  
 ArrayList<JsonBuilder> urls = new ArrayList<>();
  
  
 Header(){}
  
 Header(String Id,String Version,String PackageId,String Name,String Description){
  
  
 this.Id= Id;
  
 this.Version= Version;
  
 this.PackageId= PackageId;
  
 this.Name= Name;
  
 this.Description= Description;
  
  
 }
  
 void runHeaderGenerator(header){
  
 def hed = new JsonSlurper().parseText(header);
  
 this.Id = hed.Id;
  
 this.Version = hed.Version;
  
 this.PackageId = hed.PackageId;
  
 this.Name = hed.Name;
  
 this.Description = hed.Description;
  
 this.urls = hed.Urls;
  
  
  
  
 }
  
  
}
  
  
class Configurations{
  
 ArrayList<JsonBuilder> results = new ArrayList<>();
  
 Configurations(){}
  
  
 void runConfigurationsGenerator(configurations){
  
 def conf = new JsonSlurper().parseText(configurations);
  
 this.results = conf.results;
  
  
 }
  
}
  
  
class Runtime{
  
  
 String Name;
  
 String Type;
  
 String DeployedBy;
  
 String DeployedOn;
  
 String Status;
  
  
 Runtime(){}
  
  
 Runtime(String Name,String Type,String DeployedBy,String DeployedOn,String Status){
  
  
 this.Name= Name;
  
 this.Type= Type;
  
 this.DeployedBy= DeployedBy;
  
 this.DeployedOn= DeployedOn;
  
 this.Status= Status;
  
  
 }
  
  
 void runRuntimeGenerator(runtime){
  
 def run = new JsonSlurper().parseText(runtime);
  
 this.Name = run.Name;
  
 this.Type = run.Type;
  
 this.DeployedBy = run.DeployedBy;
  
 this.DeployedOn = run.DeployedOn;
  
 this.Status = run.Status;
  
 }
  
}
  
  
  
  
class Adapter{
  
 String Name;
  
 String SourceName;
  
 String TargetName;
  
 String Type;
  
 String componentName;
  
 JsonBuilder d = new JsonBuilder();
  
  
 Adapter(){}
  
 Adapter(String Name,String SourceName,String TargetName,String Type,String componentName){
  
  
 this.Name= Name;
  
 this.SourceName= SourceName;
  
 this.TargetName= TargetName;
  
 this.Type= Type;
  
 this.componentName= componentName;
  
  
 }
  
  
 public addJsonNode(json,name,value){
  
 json."$name"(value)
  
 }
  
  
 public findType(source){
  
  
 def valueType = null
  
 def valueProtocol = null
  
 def goOut = false
  
 def node1 = false
  
 def node2 = false
  
  
 for (EXMFlowNode in source.extensionElements.property){
  
 def node = EXMFlowNode.key.text()
  
  
 if(node == 'ComponentType'){
  
 valueType = EXMFlowNode.value.text()
  
 node1 = true
  
 }
  
 if(node == 'MessageProtocol'){
  
 valueProtocol = EXMFlowNode.value.text()
  
 node2 = true
  
 }
  
  
 if( node1 && node2 ){
  
 if(valueProtocol.contains('OData')){
  
 valueType = 'HCIOData'
  
 }
  
 if(valueProtocol == "SOAP" && valueType == "SuccessFactors"){
  
 valueType = 'SF\_SOAP'
  
 }
  
 break
  
 }
  
  
  
 }
  
  
 return valueType
  
 }
  
  
 public createAdapterDetailsNode(source){
  
  
 def type = findType(source)
  
 def adapter = new GlobalObj().HM\_collaboration[type] // only values for specific adapter
  
 d{
  
  
 for (EXMFlowNode in source.extensionElements.property){
  
 if(adapter == null){
  
 continue;
  
 }
  
  
 def node = EXMFlowNode.key.text()
  
 def value = adapter[node] // name of xml node
  
 if(value != null){// one of important property
  
 this.addJsonNode(delegate, adapter[node], EXMFlowNode.value.text())
  
 }
  
 }
  
 }
  
  
 return d.content
  
 }
  
  
 void set\_componentName(sourceName, process,participant){
  
  
 for(node in process){//interate every process
  
 for(sendNode in node.serviceTask){//take serviceTasks for every process node
  
 if(sourceName.toString() == "${sendNode.@id}"){
  
 this.componentName = "${sendNode.@name}"
  
 }
  
 }
  
  
 if(this.componentName == null){// then we can have end or start
  
  
 for(sendNode in node.endEvent){//take serviceTasks for every process node
  
 if(sourceName.toString() == "${sendNode.@id}"){
  
 this.componentName = "${sendNode.@name}"
  
 }
  
 }
  
 }
  
 }
  
  
 if(this.componentName == null){// component is a participant(sender/ receiver)
  
  
 for(node in participant){
  
 if(sourceName.toString() == "${node.@id}"){
  
 this.componentName = "${node.@name}"
  
 }
  
 }
  
 }
  
 }
  
  
 void generat(source, process,participant){
  
 this.Name = "${source.@name}";
  
 this.SourceName = "${source.@sourceRef}";
  
 this.TargetName = "${source.@targetRef}";
  
 this.Type = findType(source)
  
 this.d = createAdapterDetailsNode(source);
  
 set\_componentName(this.SourceName,process,participant);
  
 }
  
  
}
  
  
class AdapterList{
  
  
 private ArrayList adapterSourceList;
  
 private ArrayList adapterProcList;
  
 private ArrayList participant;
  
 ArrayList<Adapter> adapterList;
  
  
 AdapterList(){
  
 adapterList = new ArrayList<>();
  
 adapterSourceList = new ArrayList();
  
 adapterProcList = new ArrayList();
  
 }
  
  
 AdapterList(adapterList){
  
 this.adapterList = adapterList;
  
  
 }
  
  
 void set\_adapterProcList(source){
  
 this.adapterProcList = source;
  
 }
  
  
 void set\_participant(source){
  
 this.participant = source;
  
 }
  
  
 void set\_adapterSourceList(source){
  
 this.adapterSourceList = source;
  
 }
  
  
 void add\_element(Adapter a){
  
 adapterList.add(a);
  
 }
  
  
 void runAdaptersGenerator(){
  
 for(adaptor in adapterSourceList){
  
 def adp = new Adapter();
  
 adp.generat(adaptor, this.adapterProcList, this.participant)
  
 this.adapterList.add(adp);
  
  
 }
  
 }
  
}
  
  
class Process{
  
  
 private ArrayList processSourceList = new ArrayList<>();
  
 private HashMap<String, ArrayList<String>> adj = new HashMap<>();
  
 private ArrayList<String> orider = new ArrayList<>();
  
 String name;
  
 ArrayList<JsonBuilder> callActivity = new ArrayList<>();
  
 ArrayList<JsonBuilder> exclusiveGateway = new ArrayList<>();
  
 ArrayList<JsonBuilder> subProcess = new ArrayList<>();
  
  
 Process(){}
  
  
 void set\_processSourceList(processSourceList){
  
 this.processSourceList=processSourceList;
  
 }
  
  
 private addJsonNode(json,name,value){
  
 json."$name"(value)
  
  
 }
  
 private filterFunc(source){
  
  
 def type = ''
  
 for(SubEXcallActivity in source.extensionElements.property){//all property for an object
  
 if(SubEXcallActivity.key == 'activityType'){
  
 type = SubEXcallActivity.value.text()
  
 break
  
 }
  
 }
  
 if(new GlobalObj().HM\_Process[type]){
  
 return type
  
 }
  
 return null
  
 }
  
 private objectType(source){
  
  
 def type = ''
  
 for(SubEXcallActivity in source.extensionElements.property){//all property for an object
  
 if(SubEXcallActivity.key == 'activityType'){
  
 type = SubEXcallActivity.value.text()
  
 break
  
 }
  
 }
  
 return new GlobalObj().HM\_ObjectPromt[type]
  
  
 }
  
 private findPropertyType(source){
  
  
  
 def type = ''
  
 for(SubEXcallActivity in source.extensionElements.property){//all property for an object
  
 if(SubEXcallActivity.key == 'activityType'){
  
 type = SubEXcallActivity.value.text()
  
 break
  
 }
  
 }
  
 return type
  
  
 }
  
  
 private get\_processName(processId){
  
  
 def res = "";
  
 for(proc in processSourceList){
  
 if("${proc.@id}" == processId.toString()){
  
 res ="${proc.@name}";
  
 }
  
 }
  
 return res;
  
 }
  
  
 private createActivityDetailsNode(source){
  
 def builder = new JsonBuilder()
  
 builder{
  
  
 this.addJsonNode(delegate, "Name", "${source.@name}");
  
 def type = this.findPropertyType(source)
  
 if(type == "ProcessCallElement"){
  
  
 def processId = ""
  
 source.extensionElements.property.each{
  
 element-> if(element.key=='processId'){
  
 processId = element.value
  
 }
  
 }
  
 def processName = get\_processName(processId);
  
 this.addJsonNode(delegate, "Source Name", processName)
  
 }
  
  
  
 def importantProp = new GlobalObj().HM\_Process[type]
  
 for(SubEXcallActivity in source.extensionElements.property){//all property for an object
  
  
 if(importantProp == null){
  
 continue
  
 }
  
  
 def node = SubEXcallActivity.key.text()
  
 def val = importantProp[node]//what I promt in json
  
  
 if(val != null){
  
 if(val == 'Properties' || val == 'Headers'){
  
 this.addJsonNode(delegate, val,convertRows(SubEXcallActivity.value.text()))
  
 }
  
 else{ this.addJsonNode(delegate, val, SubEXcallActivity.value.text()) }
  
 }
  
 }
  
  
 }
  
 return builder
  
 }
  
  
 void mappNodesForDFS(){
  
  
 for(nod in processSourceList){
  
 def name = "${nod.@name}"
  
 def list = new ArrayList<String>() //value for node with value name
  
 def calls = nod.'\*\*'.findAll{
  
 node->
  
 node.name() == 'callActivity' // Retrieve all person nodes
  
 }
  
  
  
 for(call in calls){ //search for posible SubProcess
  
 boolean ok = false;
  
 for(element in call.extensionElements.property){
  
 if(element.key == "activityType" && element.value== "ProcessCallElement"){// we find it
  
 ok= true;
  
 //list.add("${call.@name}")
  
 //adj.put(name,"${call.@name}")
  
 }
  
 }
  
  
 if(ok == true ){
  
 def processId = ""
  
 call.extensionElements.property.each{
  
 element-> if(element.key=='processId'){
  
 processId = element.value
  
 }
  
 }
  
 def processName = get\_processName(processId);
  
 list.add(processName);
  
 }
  
 }
  
 if(!list.isEmpty()){
  
 this.adj.put(name,list)
  
 }
  
 }
  
 }
  
  
 private set\_order(root){
  
  
 mappNodesForDFS();
  
 def graph = new Graph();
  
 adj.each{
  
 key, value-> value.each{
  
 subNode -> graph.addEdge(key,subNode);
  
 }
  
 }
  
 graph.DFS(root);
  
 this.orider = graph.DFSres;
  
 }
  
  
 private createGatwayDetailsNode(elements,name){
  
  
 def builder = new JsonBuilder()
  
 builder{
  
 for(GPathResult element in elements){
  
 if(element.'@sourceRef' == name){
  
  
 this.addJsonNode(delegate, "${element.@name}",element.conditionExpression.text())
  
  
 }
  
 }
  
 }
  
 return builder
  
 }
  
 private void set\_callActivity(source){
  
  
  
 def callActivities = source.'\*\*'.findAll{
  
 node->
  
 (node.name() == 'callActivity') || (node.name() == 'serviceTask') // Retrieve all callActivity
  
 }
  
 for(lvCallActivity in callActivities){
  
 def builder = new JsonBuilder()
  
 if(filterFunc(lvCallActivity) != null){
  
 builder{
  
 this.addJsonNode(delegate, "${objectType(lvCallActivity)}", createActivityDetailsNode(lvCallActivity).content)
  
 }
  
  
 }
  
  
 if(builder.content != null){
  
 this.callActivity.add(builder.content);
  
 }
  
 }
  
  
  
 }
  
  
 private void set\_exclusiveGateway(source){
  
 def sequenceFlow = source.'\*\*'.findAll{ node->node.name() == 'sequenceFlow'}
  
 def gadewayNode = source.'\*\*'.findAll{ node->node.name() == 'exclusiveGateway'}//.collect {item-> item.'@id'}
  
  
 for(lvGadewayNode in gadewayNode){
  
 def builder = new JsonBuilder()
  
  
 builder{
  
 this.addJsonNode(delegate, "Name", "${lvGadewayNode.@name}")
  
 this.addJsonNode(delegate, "Path", createGatwayDetailsNode(sequenceFlow,"${lvGadewayNode.@id}").content)
  
 }
  
  
 this.exclusiveGateway.add(builder.content);
  
 }
  
 }
  
 private convertRows(source){
  
  
 def text = source
  
 text = "<root>" + text.toString() + "</root>"
  
  
 def xml = new XmlSlurper().parseText(text)
  
 def rows = xml.'\*\*'.findAll{ node->node.name() == 'row'}
  
 def result ='{ "rows":[ '
  
  
 for(GPathResult row in rows){
  
 result=result+"{"
  
 for(cell in row.cell){
  
  
 result = result+'"' +cell.'@id'+'":"'+cell+'",'
  
  
  
 }
  
  
 if (result != null && result.length() > 0 ) {
  
 result = result.substring(0, result.length() - 1);
  
 }
  
 result=result+"},"
  
  
  
 }
  
 if (result != null && result.length() > 0 ) {
  
 result = result.substring(0, result.length() - 1);
  
 }
  
 result = result+ "]}"
  
  
 return result
  
  
 }
  
  
 private void set\_subProcess(source){
  
  
 def subProcessList = source.'\*\*'.findAll{
  
 node->
  
 node.name() == 'subProcess' // Retrieve all callActivity
  
 }
  
  
 for(lvSubProcesses in subProcessList.callActivity){// is an array of processes
  
 for(lvSubProcess in lvSubProcesses){
  
 def builder = new JsonBuilder()
  
 if(filterFunc(lvSubProcess) != null){
  
 builder{
  
 Name("${source.subProcess.@name}"+ lvSubProcess.size().toString())
  
 this.addJsonNode(delegate, "${objectType(lvSubProcess)}", createActivityDetailsNode(lvSubProcess).content)
  
 }
  
  
 }
  
  
 if(builder.content != null){
  
 this.subProcess.add(builder.content);
  
 }
  
 }
  
 }
  
 }
  
  
 void generat(source,root){
  
  
 this.name = "${source.@name}";
  
 set\_callActivity(source);
  
 set\_exclusiveGateway(source);
  
 set\_subProcess(source);
  
 set\_order(root);
  
 }
  
}
  
  
class ProcessList{
  
  
 private ArrayList processSourceList;
  
 ArrayList<Process> processList;
  
  
 ProcessList(){
  
  
 processSourceList = new ArrayList();
  
 processList = new ArrayList<>();
  
 }
  
  
 ProcessList(processList){
  
  
 this.processList = processList;
  
 }
  
  
 void set\_processSourceList(processSourceList){
  
 this.processSourceList = processSourceList;
  
  
 }
  
  
 private String get\_rootProc(){
  
  
 String root = "";
  
 for(process in processSourceList){
  
 for(item in process.extensionElements.property){
  
 if(item.key == "transactionalHandling" && item.value == "Required"){
  
 root = "${process.@name}";
  
 }
  
 }
  
  
 }
  
 return root;
  
 }
  
  
 void runProcessGenerator(){
  
 ArrayList<String> orider = new ArrayList<>();
  
 def root = get\_rootProc();
  
  
 for(process in processSourceList){
  
 def proc = new Process();
  
 proc.set\_processSourceList(processSourceList);
  
 proc.generat(process,root);
  
 orider = proc.orider;
  
 this.processList.add(proc);
  
  
 }
  
  
 processList.sort { proc -> orider.indexOf(proc.name) }
  
 }
  
  
}
  
  
class Resource{
  
 String name;
  
 String resource;
  
  
 Resource(){}
  
  
 Resource(name,resource){
  
 this.name = name;
  
 this.resource = resource;
  
 }
  
  
}
  
  
class ArrayResource{
  
  
 HashMap<String,List<Resource>> HM\_resource=new HashMap<String,List<Resource>>();
  
 ArrayResource(HM\_resource){
  
 this.HM\_resource = HM\_resource;
  
 }
  
  
 ArrayResource(){this.HM\_resource = new HashMap<>()}
  
  
 public add\_element(String type,String name, String resource){
  
 // convert string into Hexa
  
 byte[] data = resource.getBytes()
  
 resource = data.encodeHex().toString()
  
  
 def objFoo = new Resource(name,resource) // new Resource object
  
 if(this.HM\_resource[type] == null){
  
 // is first lenement
  
 HM\_resource.put(type,[objFoo])
  
 }
  
 else{
  
 this.HM\_resource[type].add(objFoo)
  
 }
  
 }
  
  
 public get\_elem(String type){
  
  
 return this.HM\_resource[type].getClass()
  
 }
  
}
  
  
class Mapping{
  
  
 HashMap<String, JsonBuilder> map;
  
  
 Mapping(){
  
 this.map = new HashMap<String, JsonBuilder>();
  
  
 }
  
  
 void addMapElement(MyZIS zipstream){
  
  
  
 this.map.put(entry.getName(),new JsonSlurper().parseText(mapData));
  
 }
  
  
}
  
  
  
class Template{
  
  
 Template(){}
  
  
 String getTemplate(ext,tID,tName){//tID and tName are details about location where template is located
  
  
 def secret = 'your\_cred\_iflow'; //always is your\_cred
  
 def secureStorageService = ITApiFactory.getApi(SecureStoreService.class, null)//gone into security material
  
 def secretCred = secureStorageService.getUserCredential(secret);//take credential details
  
 def get = new URL(ext+"/iFile?packageId="+tID+"&fileName="+tName).openConnection();
  
  
 get.setRequestProperty('Authorization', 'Basic ' + (secretCred.getUsername()+':'+new String(secretCred.getPassword())).bytes.encodeBase64().toString());
  
 def getRC = get.getResponseCode();
  
  
 byte[] data = get.getInputStream().getBytes();
  
 Writable printableHex = data.encodeHex();
  
  
 return printableHex.toString();
  
 //return secretCred.getUsername()+':'+new String(secretCred.getPassword())
  
 }
  
}
  
  
  
class Root{
  
 Header header;
  
 Runtime runtime;
  
 Configurations configurations;
  
 AdapterList adapterList;
  
 ProcessList processList;
  
 ArrayResource resource;
  
 String tem;
  
 HashMap<String, JsonBuilder> map;
  
  
  
 Root(){
  
  
 header = new Header();
  
 runtime = new Runtime();
  
 configurations = new Configurations();
  
 adapterList = new AdapterList();
  
 processList = new ProcessList();
  
 resource = new ArrayResource();
  
 map = new HashMap<>();
  
 }
  
}
  
  
class ElementsRouter{
  
 //purpose of this class is to route every XML element from ZIP files. So is comunication between class and message method.
  
 private GPathResult source;
  
 Root root;
  
  
 ElementsRouter(GPathResult source){
  
 this.root = new Root();
  
 this.source = source;
  
 }
  
  
 ElementsRouter(){
  
 this.root = new Root();
  
 }
  
  
 void set\_source(GPathResult source){
  
 this.source= source;
  
 }
  
  
 void set\_Header(String header){
  
 root.header.runHeaderGenerator(header);
  
 }
  
  
 void set\_Configurations(String configurations){
  
 root.configurations.runConfigurationsGenerator(configurations);
  
 }
  
 void set\_Runtime(String runtime){
  
 root.runtime.runRuntimeGenerator(runtime);
  
 }
  
  
 void set\_Adapters(){
  
  
 def messageFlow = this.source.'\*\*'.findAll{
  
 node->
  
 node.name() == 'messageFlow' // Retrieve all person nodes
  
 }
  
  
 def participant = this.source.'\*\*'.findAll{
  
 node->
  
 node.name() == 'participant' // Retrieve all person nodes
  
 }
  
  
 def process = this.source.'\*\*'.findAll{
  
 node->
  
 node.name() == 'process' // Retrieve all person nodes
  
 }
  
  
  
 root.adapterList.set\_adapterSourceList(messageFlow);
  
 root.adapterList.set\_adapterProcList(process);
  
 root.adapterList.set\_participant(participant);
  
 root.adapterList.runAdaptersGenerator();
  
 }
  
  
 void set\_Process(){
  
  
 def process = this.source.'\*\*'.findAll{
  
 node->
  
 node.name() == 'process' // Retrieve all person nodes
  
 }
  
 root.processList.set\_processSourceList(process);
  
 root.processList.runProcessGenerator();
  
  
 }
  
  
 void set\_Template(String ext, String tID, String tName){
  
 this.root.tem = new Template().getTemplate(ext, tID,tName);
  
 }
  
  
  
 JsonBuilder changeItemKey(String s){
  
  
 def source = new JsonSlurper().parseText(s)
  
 def headerVal = new GlobalObj().HM\_Header
  
 def runtimeVal = new GlobalObj().HM\_Runtime
  
 def configVal = new GlobalObj().HM\_Configurations
  
  
 if (source.root.header ){//check if the node header exist
  
 headerVal.each{
  
 key, value ->
  
 if(source.root.header[key] && key != value){//key <> value(if not then will delete the node without to create )
  
 source.root.header[value] = source.root.header[key];
  
 source.root.header.remove(key);
  
 }
  
 }
  
 }
  
 if (source.root.runtime){//check if the runtime node exist
  
 runtimeVal.each{
  
 key, value ->
  
 if(source.root.runtime[key] && key != value){//key <> value(if not then will delete the node without to create
  
 source.root.runtime[value] = source.root.runtime[key];
  
 source.root.runtime.remove(key);
  
 }
  
 }
  
 }
  
  
 if (source.root.configurations.results){//check if the configurations node exist
  
 for(configuration in source.root.configurations.results){
  
  
 configVal.each{
  
 key, value ->
  
 if(configuration[key] && key != value){//key <> value(if not then will delete the node without to create
  
 configuration[value] = configuration[key];
  
 configuration.remove(key);
  
 }
  
 }
  
 }
  
 }
  
  
 return source;
  
  
 }
  
  
  
  
}
  
  
///////////////////////////////////////
  
  
  
  
Message processData(Message message) {
  
  
  
 def input = message.getBody(java.io.InputStream);
  
 def header = message.getProperty("header")
  
 def configurations = message.getProperty("configurations")
  
 def runtime = message.getProperty("runtime")
  
 def resource = message.getProperty("resource")
  
 def tID = message.getProperty("tID")
  
 def tName = message.getProperty("tName")
  
 def ext = "${message.getHeaders().get('CamelHttpUrl').replace('/iSourceContent','')}".toString()
  
  
 if (ext.startsWith("http")) {
  
 ext = "https" + ext.substring(4)
  
 }
  
  
 def xmlSource = null;
  
 def listResource = [];
  
  
 HashMap<String, JsonBuilder> listMmap = new HashMap<String, JsonBuilder>();
  
 MyZIS zipstream = new MyZIS(input);
  
  
 ZipEntry entry = null
  
 isFound = false
  
 //resource = "groovy,mmap,wsdl,xsd"// de sters!!!!
  
 def resourceList = resource.split(",")
  
 def lementsRouter = new ElementsRouter();
  
  
 while ((entry = zipstream.getNextEntry()) != null) {
  
  
 if(entry.getName()[-5..-1] == """.iflw"""){
  
 xmlSource = zipstream.text;
  
 continue;
  
 }
  
  
 for(element in resourceList){
  
 def ln = (element.length() + 1) \* -1 // and 1 element for "." and have negative value vecouse we take last ln element from entry name
  
 def val ="." + element
  
  
  
 if((entry.getName()[ln..-1] != """.mmap""") && (entry.getName()[ln..-1] == val) ){ // sure we know that value<> mmap.
  
 lementsRouter.root.resource.add\_element(element,entry.getName(),zipstream.text)//.bytes.encodeBase64().toString()
  
 }
  
  
 if((entry.getName()[ln..-1] == """.mmap""") && (entry.getName()[ln..-1] == val) ){
  
  
 Document document = processxml(zipstream);
  
  
 // Normalize the XML Structure; It's just too important !!
  
 document.getDocumentElement().normalize();
  
  
 // Here comes the root node
  
 Element root = document.getDocumentElement();
  
  
  
 Element content = (Element) root.getElementsByTagNameNS("urn:sap-com:xi", "content").item(0);
  
 Element metaData = (Element) content.getElementsByTagNameNS("urn:sap-com:xi:mapping:xitrafo","MetaData").item(0);
  
  
 String mapData;
  
  
 if(metaData.getElementsByTagName("mappingtool").getLength() > 0 ) {
  
 Element transformation = getTransformationTag(metaData);
  
 mapData = getMapData(transformation);
  
 }
  
 else if(metaData.getElementsByTagNameNS("urn:sap-com:xi:mapping:xitrafo", "blob").getLength()>0) {
  
  
 byte[] buffer = new byte[1024];
  
 String b64data = metaData.getElementsByTagNameNS("urn:sap-com:xi:mapping:xitrafo", "blob").item(0).getTextContent();
  
 b64data = b64data.replace("!zip!", "");
  
 byte[] decCon = Base64.decodeBase64(b64data);
  
  
 ByteArrayOutputStream oStream = getunzipData(decCon);
  
  
 ByteArrayOutputStream oStream1 = getunzipData(oStream.toByteArray());
  
  
 InputStream newinpStream = new ByteArrayInputStream(oStream1.toByteArray());
  
  
 Document doc1 = processxml(newinpStream);
  
 doc1.getDocumentElement().normalize();
  
 Element root1 = doc1.getDocumentElement();
  
 Element transformation1 = getTransformationTag(root1);
  
 mapData = getMapData(transformation1);
  
  
 }
  
  
 listMmap.put(entry.getName(),new JsonSlurper().parseText(mapData));
  
 }
  
  
 }
  
  
 }
  
  
 zipstream.myClose();
  
 def source = new XmlSlurper().parseText(xmlSource)
  
 lementsRouter.set\_source(source);
  
 lementsRouter.set\_Header(header);
  
 lementsRouter.set\_Configurations(configurations);
  
 lementsRouter.set\_Runtime(runtime);
  
 lementsRouter.set\_Adapters();
  
 lementsRouter.set\_Process();
  
 lementsRouter.set\_Template(ext,tID,tName)
  
 lementsRouter.root.map = listMmap;
  
  
  
  
  
 message.setBody(lementsRouter.changeItemKey(new JsonBuilder(lementsRouter).toString()))
  
  
 return message
  
  
}
  
  
def Document processxml(InputStream inpStream) throws Exception {
  
  
 DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
  
 factory.setNamespaceAware(true);
  
 DocumentBuilder builder = factory.newDocumentBuilder();
  
  
 Document document = builder.parse(inpStream);
  
  
 return document;
  
}
  
  
def Element getTransformationTag(Element root) {
  
  
 NodeList transf = root.getElementsByTagName("transformation");
  
 Element transformation = (Element) transf.item(0);
  
 return transformation;
  
}
  
  
  
def String getMapData(Element transformation) {
  
  
 NodeList nList1 = transformation.getChildNodes();
  
  
  
 def dh = [];
  
  
 int resar = 0;
  
  
 for (int temp = 0; temp < nList1.getLength(); temp++) {
  
  
 int sf = 0;
  
 int fr = 0;
  
 int cn = 0;
  
  
  
 Node node = nList1.item(temp);
  
  
 if(node.getNodeType() == Node.ELEMENT\_NODE) {
  
  
 dh[resar] = new docHeader();
  
 Element eElement = (Element) node;
  
  
 NodeList nList2 = eElement.getElementsByTagName("brick");
  
  
 for (int temp1 = 0; temp1 < nList2.getLength(); temp1++) {
  
  
 Node node1 = nList2.item(temp1);
  
 Element eElement1 = (Element) node1;
  
  
 if(eElement1.getAttribute("type").equals("Src")) {
  
 if(sf!=0)
  
 dh[resar].setsourceFields("\n");
  
 dh[resar].setsourceFields(eElement1.getAttribute("path"));
  
 sf++;
  
 }
  
 else if(eElement1.getAttribute("type").equals("Func") && eElement1.getAttribute("fname").equals("const")) {
  
 if(cn!=0)
  
 dh[resar].setconstantValues("\n");
  
 dh[resar].setconstantValues("Constant -" + eElement.getElementsByTagName("value").item(0).getTextContent());
  
 cn++;
  
 }
  
 else {
  
 String value=eElement1.getAttribute("fname");
  
 if(fr!=0 && !value.isEmpty())
  
 dh[resar].setfunctionS("\n");
  
 dh[resar].setfunctionS(value);
  
 fr++;
  
 }
  
  
 }
  
 dh[resar].settargetField(eElement.getAttribute("path"));
  
 resar++;
  
  
 }
  
  
 }
  
  
 Gson gson = new GsonBuilder().setPrettyPrinting().create();
  
 return gson.toJson(dh);
  
  
}

**src/main/resources/script/output\_IMaps.groovy**

import com.sap.gateway.ip.core.customdev.util.Message;
  
import java.util.HashMap;
  
import groovy.json.\*
  
  
  
class result {
  
 String name
  
 String iD
  
}
  
  
def Message processData(Message message) {
  
  
 //Get Response Payload
  
 def body = message.getBody(String.class);
  
  
 def jsonSlurper = new JsonSlurper()
  
 def json = jsonSlurper.parseText(body)
  
  
 def output = " [{\"iD\": \"STOP\",\"name\": \"No Message Mapping in IFlow\"}]"
  
  
 def r = [];
  
 int i = 0;
  
  
 json.d.results.each{
  
 r[i]=new result(name:it.Name.replace(".mmap",""), iD: it.Name);
  
 i++;
  
 }
  
  
 if(i>0)
  
 output = JsonOutput.toJson(r);
  
  
  
 message.setBody("{\"IntegrationMaps\":" + output+ "}");
  
  
 return message;
  
  
}

**src/main/resources/script/script11.groovy**

import com.sap.gateway.ip.core.customdev.util.Message  
import com.sap.gateway.ip.core.customdev.processor.MessageImpl  
import groovy.xml.MarkupBuilder  
import java.util.zip.\*  
import javax.xml.\*  
import java.text.\*  
import java.util.ArrayList;  
import groovy.util.\*  
import groovy.json.\*  
  
import javax.xml.bind.DatatypeConverter  
  
class MyZIS extends ZipInputStream {  
  
 public MyZIS(InputStream input) {  
 super(input);  
 }  
  
 @Override  
 public void close() throws IOException {  
 }  
  
 public void myClose() throws IOException {  
 super.close();  
 }  
}  
  
def Message processData(Message message) {  
 def input = message.getBody(java.io.InputStream);  
 def templateContent = null;  
 def resourcesInfo = null;  
 MyZIS zipstream = new MyZIS(input);  
 ZipEntry entry = null  
   
 while ((entry = zipstream.getNextEntry()) != null) {  
   
 if(entry.getName() == """resources.cnt"""){  
 resourcesInfo = zipstream.text;  
 }  
 }  
 zipstream.myClose();  
 def decodedResourcesInfo = DatatypeConverter.parseBase64Binary(resourcesInfo) // Decoded byte array  
 resourcesInfo = new String(decodedResourcesInfo, 'UTF-8') // Convert byte array to string  
   
 message.setProperty("TemplateID",getResourceID(resourcesInfo,"MyTemplate.docx"))  
 return message;  
}  
  
def getResourceID(jsonStr, templatetName){  
   
 def json = new JsonSlurper().parseText(jsonStr)  
 def targetResource = json.resources.find{ it.name == templatetName && it.resourceType == "File"}  
   
 if(targetResource){  
 targetResource=targetResource.id //get only the ID not all the info node  
 }  
   
 return targetResource  
   
}  
  
def getTemplateContent(java.io.InputStream input,String ID){  
  
 MyZIS zipstream = new MyZIS(input);  
 ZipEntry entry = null  
 def content = ""  
   
 while ((entry = zipstream.getNextEntry()) != null) {  
 content = content +" "+ entry.getName()  
 if(entry.getName() == ID.toString()+"""\_content"""){  
 //content = zipstream.text;  
 content = content +" "+ entry.getName()  
 }  
 }  
 zipstream.myClose();  
   
 return content  
}

**src/main/resources/script/script12.groovy**

import com.sap.gateway.ip.core.customdev.util.Message  
import com.sap.gateway.ip.core.customdev.processor.MessageImpl  
import groovy.xml.MarkupBuilder  
import java.util.zip.\*  
import javax.xml.\*  
import java.text.\*  
import java.util.ArrayList;  
import groovy.util.\*  
import groovy.json.\*  
  
import javax.xml.bind.DatatypeConverter  
  
class MyZIS extends ZipInputStream {  
  
 public MyZIS(InputStream input) {  
 super(input);  
 }  
  
 @Override  
 public void close() throws IOException {  
 }  
  
 public void myClose() throws IOException {  
 super.close();  
 }  
}  
  
def Message processData(Message message) {  
 def input = message.getBody(java.io.InputStream);  
 def templateID = message.getProperty("TemplateID");  
   
 MyZIS zipstream = new MyZIS(input);  
 ZipEntry entry = null  
 def content = ""  
   
 while ((entry = zipstream.getNextEntry()) != null) {  
 if(entry.getName() == templateID.toString()+"""\_content"""){  
 content = zipstream.text;  
 }  
 }  
 zipstream.myClose();  
   
 message.setBody(content)  
 return message;  
}