Building Custom VDAB Function Sets

Typically complete applications can be created using VDAB's core nodes or extended node packages.

When the available nodes do not provide the features or efficiency needed custom code can be written and incorporated in your flows.

There are two approaches to adding these capabilities including writing a complete *Custom VDAB Node* or writing a *VDAB Function Set*:

	Function Set	Custom Node
Level of Effort	Small	Moderate
Scope	Specific	General or Specific
Editable Attributes		X
Asynchronous response		X
Service Integration		X
Data Aggregation		X

In general the *VDAB Function Set* approach should be used when the processing is straightforward and does not require custom attributes.

This document describes the approach for writing a *VDAB Function Set* using Java that can be invoked using the built in *JavaFunction* or *JavaConditional* node.

For completely custom nodes that can be assembled and edited by VDAB see the *Creating Custom Nodes* Guide.

Before This	• You have reviewed the VDAB introductory tutorial and documentation to understand how to create basic flows from nodes. • You have some understanding of the Java programming language. • You have successfully setup a development environment. (See the *Custom Java Development* guide.)

Contents

Related Documentation

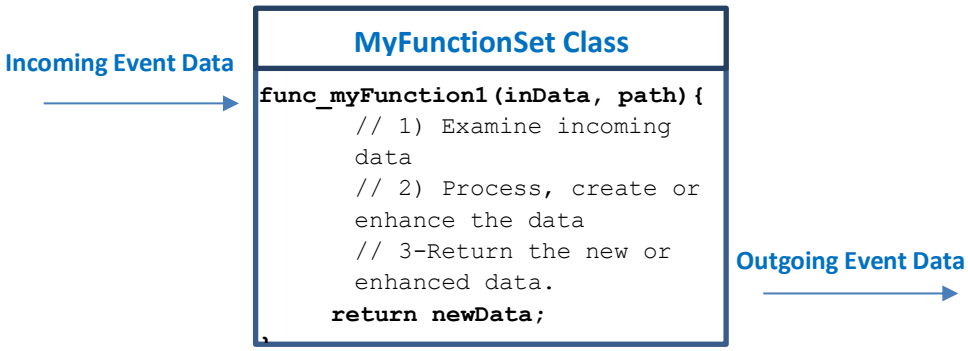
The following document and tutorials either are a) available or b) being developed to further support this subject. Those available are highlighted in green while those under development are currently grey.

Related Guides	Details
VDAB Standard Directory reference	This reference document defines the standard directory structure used by VDAB.
Custom Java Development	This guide details how to set up an environment for developing your own custom Java Functions and VDAB Nodes.
Building Custom Nodes	This guide documents the approach for writing complete custom nodes in Java.

Related Tutorial	Details

Java Function Overview

A simple java function set simply needs to implement one or more public methods which start with the prefix *func_*.



Once this class is compiled and deployed it can be invoked by simply adding the JavaFunction node to a flow and selecting the function set name followed by selecting the desired method.

Setting up the Development Environment

Follow the instructions in the *Custom Java Development* guide to set up your development environment.

Creating the Custom Function Set Class

The custom function class can be created by making a class that extends the class *JavaFunctionSet_A.*

Generally a function set should implement a group of related functions.

In the example below a function set AddressHandling is created that will include a number of functions that support handling account address data.

```
package vdab.samples.func;

import vdab.api.event.VDABData;
public class AddressHandling extends JavaFunctionSet_A {
```

Within the class one or more methods should be implemented that process the incoming event data. They must all begin with the prefix func_ and must include two parameters

Method name	Must start with func_
Parameters	The method signature must take two parameters • InData (VDABData) - the incoming event data. • selectedPath (String) – The data path of the selected data.
Returns	outData (VDABData)

The following demonstrates a method in the *AddressHandling* function set that includes two functions that help when handling addresses:

The first function takes the state name and adds the two digit state code to the address field

```
public VDABData func_stateNameToCode(VDABData vData, String selectedPath){

    if (c_StateNameToCode_map.size() < 50)
        initStateMap();

    String stateName = vData.getDataAsString(selectedPath);
    if (stateName == null){
        setError("State field must be selected and must contain data PATH="+sele
        return null;
    }

    String stateCode = c_StateNameToCode_map.get(stateName);
    if (stateCode == null){
        setError("State field did not match any state StateName="+stateName);
        return null;
    }

    vData.addDataObject("StateCode", stateCode);
    return vData;
}
```

The second function takes the components fields in the address and adds a field called FullAddress which has all the address components in a single string.

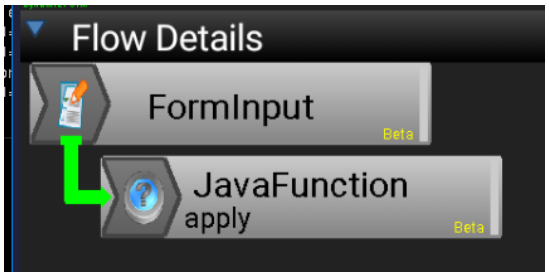
```
public class AddressHandling extends JavaFunctionSet_A {
 public VDABData func_getFullAddress(VDABData vData, String selectedPath) {

    // Build up the full address
    StringBuilder sb = new StringBuilder();
    String address = vData.getDataAsString(selectedPath+".Address");
    if (address == null){
        setError("Incoming data must include a 'Address' field ");
        return null;
    }
    if (address != null)
        sb.append(address).append("\n");
    String city = vData.getDataAsString(selectedPath+".City");
    if (city != null)
        sb.append(city).append(",");
    String state = vData.getDataAsString(selectedPath+".State");
    if (state != null)
        sb.append(state).append("  ");
    Integer postalcode = vData.getDataAsInteger(selectedPath+".ZipCode");
    if (postalcode != null)
        sb.append(postalcode);

    // Adds the full address to the data and returns it.
    vData.addDataObject(selectedPath+".FullAddress", sb.toString());
    return vData;
 }
```

DEVELOPMENT: Testing the Function Set

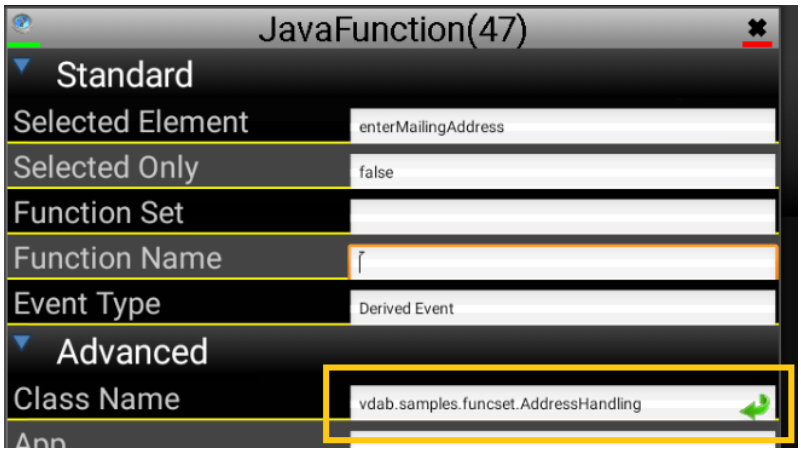
Once the Function Set has been created it can be run it can be tested and debugged by deploying a *JavaFunction* node and selecting the ClassName advanced node attribute.

1. Run VDAB from your IDE

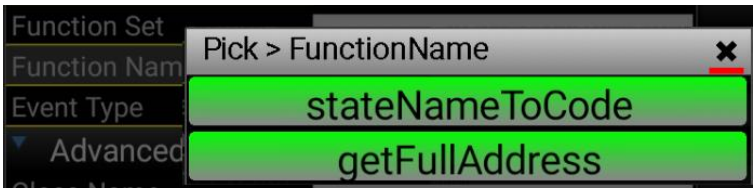
2. Add a *JavaFunction* node to another node



3. Open the JavaFunction node for editing Edit the JavaFunction ClassName attribute by expanding to show the Advanced attributes. Enter the full class name of your Function Set class. Hit the green return arrow.



4. After entering the *ClassName* field, click on the *FunctionName* field and select the function you wish to test. (If properly referenced and constructed, it should appear in the *FunctionName* pick box.

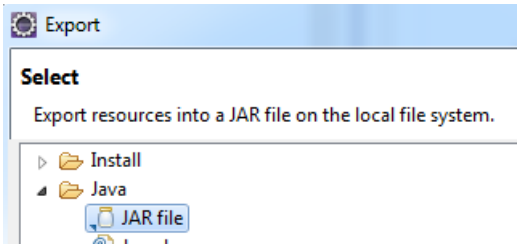


5. Click in the upper left to save the edits and start your test flow to test. You can debug your function using the standard approach for your IDE.

PRODUCTION: Deploying and Running the Function Set

After finishing development you should export the Function Set as a jar and place it in the *./ext/functions* directory. This will make it available to your program and anyone will be able to select it by selecting the Function Set name which will be the simple class name.

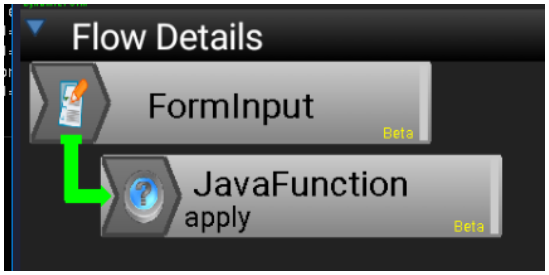
1. Export a simple jar file containing only the Function Set class or classes.



2. Deploy the Function Class JAR file by copying the jar file to the **./ext/functions** directory.

3. Restart VDAB.

4. In your flow, add a *JavaFunction* node to another node



5. Now that the Function Set has been deployed you can pick the Function Set attributed from the pick list provided. After picking one, hit the green return error.

6. Now that a Function Set has been selected, the Function Name popup will list available functions. Choose one and save the edits to begin using this function in your flow.

APPENDICES:

The VDABData Class

The VDABData class is used for reading, creating and modifying event data. Java Docs are available with a summary found below:

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	addDataObject(java.lang.String path, java.lang.Object data)
static VDABData	createDataObjectFromJSON(java.lang.String json)
static VDABData	createDataObjectFromXML(java.lang.String xml)
java.lang.Object	getData(java.lang.String path)
java.lang.Integer	getDataAsInteger()
java.lang.Integer	getDataAsInteger(java.lang.String path)
java.lang.Long	getDataAsLong()
java.lang.Long	getDataAsLong(java.lang.String path)
java.lang.String	getDataAsString()
java.lang.String	getDataAsString(java.lang.String path)
DataHolder	getDataHolder()
VDABData	getDataObject(java.lang.String path)
java.lang.String	getJSON()
java.lang.String	getXML()
boolean	isNumeric()
boolean	isSimple()
boolean	isSimple(java.lang.String path)
VDABData	mapData(VDABData from, java.lang.String[][] mapPairs)
VDABData	mapData(VDABData from, java.lang.String fromPath, java.lang.String toPath)
VDABData	removeData(java.lang.String path)