Service Logic Interpreter Directed Graph Guide

Kevin Smokowski (ks6305@att.com) created a great wiki on Directed Graphs for use by SDNC developers at AT&T. We have published those pages to the ONAP Wiki so that it is available to the community. Many thanks to Kevin for a great set of documentation on the topic.

What is a Directed Graph?

DG is short for directed graph and this abbreviation is used heavily in every day discussion about SDNC. We have built a tool (dgbuilder) for creating directed graphs on top of IBM's Node-RED a graphical, open-source project that was also used in an Internet of Things application. Node-RED is hosted at https://nodered.org/.

Mainly the Node-RED graphical editor (with some customization) and JSON storage have been re-used. All of the plugins being used were written in house . The git repository for dgbuilder is hosted here .

The Node-RED documentation will not be beneficial because DG builder is highly customized.

The plugins on the public Node-RED site are written using javascript, AT&T's implementation is using Java. This means public Node-RED plugins are incompatible with DG builder.

DG builder refers to the graphical tool for creating a DG. A DG is the actual graph which is an instance of a service flow. DG builder uses json strings as its internal format but generates an XML format for use by the SDNC Service Logic Interpreter.

When a REST RPC executes, the Service Logic Interpreter runs the XML format of the DG. The XML is interpreted by the service logic interpreter. SLI is a piece of java code written in house. The code can be found at gerrit.

Why use DG builder?

The goal of DG is to provide an execution environment for quickly written and highly customized service flows.

Read this article found here.

The goal is that "SDN-C can be “taught” new things without any coding". DG builder is not traditional coding such as java, although it can require java calls for heavy lifting.

In the future service engineers would be capable of implementing requirements using DG builder alone.

This long term goal has not been fully realized.

Installing DG builder locally

Follow the page Install Directed Graph Builder on Ubuntu Desktop .

Accessing DG builder in an ONAP OOM Installation

A Docker container for DG builder is created when installing SDNC in an OOM installation, and the container exposes a NodePort at port 30203 (in the master branch on Dec. 5, 2018). When browsing to <host>:30203, you will be prompted for credentials, which are dguser:test123.

Introduction to the DG builder GUI

Hitting this button causes current DG to be backed up to the database connected to DG builder.

This does not actually cause any code to become active in any test environment.

This button updates the JSON not the XML. The XML is what is used at run-time.

To the left of start is a beige rectangle. This is actually a button, when clicked a menu will appear.

Validate XML - makes sure the XML is valid. Generally I find this isn't as helpful as other XML validating tools because it doesn't have line highlighting. Sometimes it is easier to get the XML version of the DG and use another tool to perform this function. This option does not validate the XML against any schema.

Email Flow - e-mails the current user a version of the flow in .html format. This can be used to share the DG with reviewers that might not have access to a dgbuilder.

Upload XML - uploads the actual XML to the database (usually a development instance). This updates the existing flow and is really like "deploying the code". When uploading the XML it is also checked whether it complies with the grammar. Not all valid XML complies with the grammar SLI uses. After uploading the XML you still need to activate the graph before the code is really running.

View Dg List - displays which versions of your DG exist on the remote database.

Download Xml - downloads an XML format of the current DG to your desktop computer. This is useful if you want to manually load and activate a DG or when you want to check your code into Git.

Download JSON - downloads a JSON format of the current DG to your desktop. This is useful because DGs are imported from their JSON form. This should also be used when you are ready to check your code into Git.

Close - closes the current pop-up menu

Service Logic Nodes

The page Service Logic Interpreter Nodes gives a brief description of each node, but does not cover the plugins.

When running DG builder selecting a node also provides a short description in the bottom right of the screen.

The easiest method is often looking at an existing DG and see how nodes are being used together to accomplish something.

If you want specific examples you can search the service-logic or platform-logic directory in Eclipse or simply using Grep. Many times what you are trying to accomplish has been done before.

Writing A DG

Many people think a DG is purely graphical but is not. DG's start with dragging various graphical nodes from a palette.

Behind each graphical node is specially structured XML. This XML is really what drives a DG.

It is not possible to write a DG without manually editing the specially structured XML behind each graphical node.

This XML is written out by hand (or copy-pasted) and is not generated by the tool (DG builder or otherwise).

Because of this it is fair to say DG is its own programming language. After all DG does use its own grammar to define proper syntax.

ANTLR (used to define a grammar to parse the XML) and SLI (in house java code) are what drive this "programming language".

Approach To Learning

The best way to start is by following this tutorial and going through each step.

After learning the fundamentals there are a few examples of existing DGs in gerrit (platform-logic).

File Formats (XML and JSON)

The XML file is always what is interpreted by the SLI (Service Logic Interpreter).

JSON is the only format that can be loaded into DG builder. This is considered the source format, because the XML is generated from it.

DG builder can load a JSON flow and save it as an XML flow. The XML is considered the target format, it is what actually runs.

Always save both the JSON and the XML files. The file names will be identical, only the file extension will vary.

Going from XML back to JSON is not possible. If you forget to save the JSON it will be lost.

Your First Graph

Continue the tutorial by creating Your First Graph