AllJoyn™ Software Framework: enabling the Internet of Everything

AllJoyn Code Generator

Joe Huffman
Qualcomm Connected Experiences, Inc.

March 13, 2014

Applications for automated code generation

Where automated code generation can yield benefits

- **Test –** Tests for multiple platforms can be rapidly built and run from a single XML source file.
- **Samples –** Create your own samples rather than reading documentation.
- **Prototypes –** Prototypes of systems can be rapidly created and tested on multiple platforms.
- **Problem isolation –** The real application isn’t working. Is it in the app or inside AllJoyn code?

Automated code generation from XML

XML definition of the service

The AllJoyn introspection specification is an XML definition of the API

We have a standardized way to describe an AllJoyn API which is easily parsed for translation into code for various platforms. A simple example:

```
<node name="/org/alljoyn/bus/samples">
    <interface name="org.alljoyn.bus.samples.SimpleInterface">
        <method name="Ping">
            <arg name="inStr" type="s" direction="in"/>
            <arg name="outStr" type="s" direction="out"/>
        </method>
    </interface>
</node>
```

It is based on the D-Bus specification.

See: http://dbus.freedesktop.org/doc/dbus-specification.html#introspection-format

Installing CodeGen

Prerequisites, building, and installation

Prerequisites

CodeGen was developed using Python and Cheetah. No knowledge of either is required to use the code generator but runtime modules from the installation of these products is required before installing and using the code generator.

Install Python and Cheetah in the following order and from the following locations:

- Python 2.7.x from http://www.python.org/download/
- Cheetah 2.4.4 from https://pypi.python.org/pypi/Cheetah/2.4.4

Building and Installing CodeGen

Obtain the CodeGen source from
https://git.allseenalliance.org/cgit/devtools/codegen.git

Follow the directions in the Readme.txt file found in the installation root directory for building the CodeGen installer. There are instructions for building both Windows and Linux installers.

Then run the appropriate installer for your development environment:

- AllJoynCodeGenSetup-X.xx.win32.exe
- AllJoynCodeGenSetup-X.xx.tar.gz

Running CodeGen

Command line options and Thin Client (Library) compilation

Command line user interface

The user interface is an ordinary command line with numerous options. After installation you will then be able to invoke the code generator from a command prompt like this:

C:\>ajcodegen.py --help

The above command will output the command line parameters and options to stdout.

There are three require command line parameters:

- -ttc
- -w*{well known name}*
- *{The xml source file}*

Sample XML source and invocation

Simple XML source:

```
<node name="/org/alljoyn/sample/voidmethod">
    <interface name="Sample.VoidMethod">
        <method name="MyVoid" />
    </interface>
</node>
```

If the above XML is put into the file Void.xml then the code generator can be successfully invoked with the following command line: “ajcodegen.py -ttc -wCodeGen.Test Void.xml”

This will produce Thin Client (Library) source code and a Sconscript file for building Thin Client (Library) executables in a Thin Client build environment subdirectory. When built there will be ServiceMain.exe and ClientMain.exe in the target directory. Use the -R (runnable) flag to produce executables that communicate back and forth.

For real world XML see the D-Bus introspection specification:

http://dbus.freedesktop.org/doc/dbus-specification.html#introspection-format

Thin Client (Library) Compilation

The easiest way to compile code produced for Thin Client (Library) targets is to create a directory under the root of a Thin Client (Library) installation. Then modify the root SConstruct file to include this directory.

For example make the directory “codegen” then add the following line to the SConstruct file:

```
env.SConscript('codegen/SConscript')
```

Put the code generated in this directory (use of the -p flag is useful here) then run scons as follows from Thin Client (Library) root directory:

```
scons WS=off
```

This will produce ServiceMain and ClientMain executables.

Code generator roadmap

Architecture and plans of the code generator

Architecture

Python core which drives language bindings built with Cheetah templates.

The command line user interface, XML parser, and a small part of the language binding is written in Python. The individual language bindings are written in the template language Cheetah.

The use of a template language allows developers to embed code in the language binding files rather than embed language bindings in code files. The use of a template language results greater readability and maintainability of the language bindings.

Future Language bindings

The use of a template language for the targeted environment means the code generator is easily extensible to other AllJoyn language bindings.

As part of planning for 14.06 we will determine what is next.


Questions?
質問は？
Questions?
问题？
Ερωτήσεις?
Questions?
Questions?
Fragen?
Вопросы?
Spørsmål?
Questions?
Domande?
Vragen?
Questions?
질문이 있습니까?
Cwestiynau?
Spørgsmål?
問題？
FRÅGOR?
¿Preguntas?
Questions?
Kysymyksiä?

Thank You!