CommonAPI SOME/IP C++ User Guide

Contents

1 Introduction

1.1 Aim of this document

This document complements the CommonAPI tutorial with D-Bus specific information. Please read the base tutorial first.

2 Integration Guide for CommonAPI users

The following descriptions assume that host and target platform are Linux platforms. However CommonAPI SOME/IP also supports Windows as host and target platform. All you need to know for Windows concerning CommonAPI you find in the separate Windows paragraph below at the end of this Integration Guide.

2.1 Requirements

CommonAPI was developed for GENIVI and will run on most Linux platforms. Additionally it is possible to run it under Windows for test and development purposes. Please note:

- CommonAPI uses a lot of C++11 features, as variadic templates, std::bind, std::function and so on. Make sure that the compiler of your target platform is able to compile it (e.g. gcc 4.8).
- The build system of CommonAPI is CMake; please make sure that it is installed on your host.
- Do not use earlier versions of Eclipse as Luna; it could work but there is no warranty.
- The build tool chain for the code generators is Maven; make sure that at least Maven 3 is available. If you use eclipse make sure that the maven plug-in is installed.

The CommonAPI SOME/IP binding requires the SOME/IP implementation vsomeip.

2.2 Dependencies

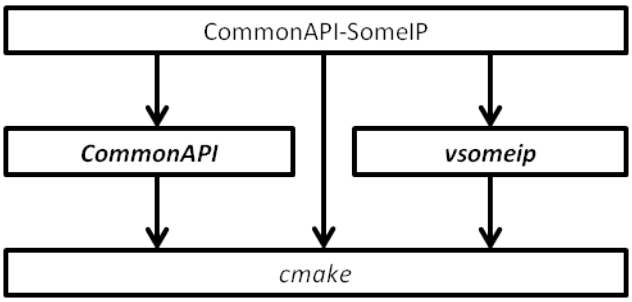

Figure 1: CommonAPI-SomeIP-Dependencies

2.2.1 Command-line

In order to build the CommonAPI SOME/IP Runtime library the vsomeip library must be available on your system. Please consult the vsomeip documentation for informations on building vsomeip.

Now use CMake to build the CommonAPI SOME/IP runtime library. We assume that your source directory is `common-api-someip-`

```
$ cd common-api-someip-runtime
$ mkdir build
$ cmake -D USE_INSTALLED_COMMONAPI=ON -D CMAKE_INSTALL_PREFIX=/usr/local ..
$ make
$ make install
```

You can change the installation directory by the CMake variable `CMAKE_INSTALL_PREFIX` or you can let it uninstalled (skip the `make install` command). If you want to use the uninstalled version of CommonAPI set the CMake variable USE_INSTALLED_COMMONAPI to OFF.

This is the standard procedure and will hopefully create the shared CommonAPI SomeIP runtime library libCommonAPI-SomeIP.so in the build directory. Note that CMake checks if doxygen and asciidoc are installed. These tools are only necessary if you want to generate the documentation of your own.

There are several options for calling CMake and make targets.

Generate makefile for building a static CommonAPI library (default is a shared library). The library will be in */build/src/CommonAPI/someip*.

```
$ cmake -DBUILD_SHARED_LIBS=OFF ..
```

Generate makefile for building the release version of CommonAPI SOME/IP (default is debug).

```
$ cmake -DCMAKE_BUILD_TYPE=Release ..
```

Without any further settings `make install` will copy CommonAPI SOME/IP libraries and header files to */usr/local*. You can change this destination directory by changing the installation prefix (e.g. to test).

```
$ cmake -DCMAKE_INSTALL_PREFIX=/test  ..
```

Make targets:

`make all`	Same as make. Will compile and link CommonAPI.
`make clean`	Deletes binaries, but not the files which has been generated by CMake.
`make maintainer-clean`	Deletes everything in the build directory.
`make install`	Copies libraries to */user/local/lib/commonapiX.X.X* and header files to */user/local/include/commonapiX.X.X/CommonAPI*.
`make DESTDIR=< install_dir > install`	The destination directory for the installation can be influenced by `DESTDIR`.

Further make targets will be described in the contributor's guide below.

2.2.2 Eclipse

Follow the instructions in the CommonAPI User Guide.

2.3 Compile tools

Like the CommonAPI core code generators you can build the SOME/IP generator by calling maven from the command-line. Open a console and change in the directory org.genivi.commonapi.someip.releng of your CommonAPI-Tools directory. Then call:

```
mvn clean verify -DCOREPATH=< path to your CommonAPI-Tools dir> -Dtarget.id=org.genivi. ↩
    commonapi.someip.target
```

After the successful build you will find the commond-line generators archived in org.genivi.commonapi.someip.cli.product/target/produc
and the update-sites in org.genivi.commonapi.someip.updatesite/target.

2.4 Build SomeIP Glue Code

The glue code library contains the binding specific, generated code. It depends on your specific project how exactly this library is built (with or without skeleton code, divided up into several libraries, e.g. for services and clients, and so on). The glue code for the verification tests can be built by means of the binding specific verification project (for SOME/IP it is *org.genivi.commonapi.someip.verification* in CommonAPI-SomeIP-Tools):

1. Generate CommonAPI code for all requested fidl files using the CommonAPI code generator and generate CommonAPI-SomeIP Code from fdepl files using the CommonAPI-SomeIP generator. The fidl files for the verification tests can be found in *org.genivi.commonapi.core.verification/fidl*. The fdepl files are created and used for generating code when using the example cmake call shown below.
2. Create a build directory for an out of source build.
3. Call cmake as described below with additional parameters (in eclipse create a make target).

CMake parameters:

`USE_INSTALLED_COMMONAPI`	ON or OFF
`COMMONAPI_CMAKE_INSTALL_PATH`	Path to the build directory of CommonAPI (e.g. CommonAPI/build)
`COMMONAPI_SOMEIP_TOOL_GENERATOR`	SomeIP Code generator executable with path
`COMMONAPI_TOOL_GENERATOR`	Core Code generator executable with path

Example to build SomeIP glue code for the verification tests:

```
cd CommonAPI-SomeIP-Tools/org.genivi.commonapi.someip.verification/
mkdir build
cd build
cmake \
-DCOMMONAPI_SOMEIP_TOOL_GENERATOR=myworkpath/CommonAPI-SomeIP-Tools/org.genivi.commonapi. ↩
    someip.cli.product/target/products/org.genivi.commonapi.someip.cli.product/target/ ↩
    products/org.genivi.commonapi.someip.cli.product/linux/gtk/MYARCH/commonapi--someip- ↩
    generator-linux-MYARCH \
-DCOMMONAPI_TOOL_GENERATOR=myworkpath/CommonAPI-Tools/org.genivi.commonapi.core.cli.product ↩
    /target/products/org.genivi.commonapi.core.cli.product/linux/gtk/MYARCH/commonapi- ↩
    generator-linux-MYARCH \
-DCommonAPI_DIR=myworkpath/CommonAPI/build \
-DCommonAPI-SomeIP_DIR=myworkpath/CommonAPI-SomeIP/build \
-Dvsomeip_DIR=myworkpath/vSomeIP/build \
-DCOMMONAPI_TEST_FIDL_PATH=myworkpath/CommonAPI-Tools/org.genivi.commonapi.core. ↩
    verification/fidl ..

make -j4
```

2.5 Project Setup

2.5.1 Configuration

CommonAPI-SomeIP can be configured as CommonAPI itself by an ini-file. The default name of this configuration file is `commonapi-someip.ini`. There are three places where CommonAPI-SomeIP Runtime tries to find this file (in the following order):

1. in the directory of the current executable. If there is a `commonapi-someip.ini` file, it has the highest priority.
2. in the directory which is specified by the environment variable `COMMONAPI_SOMEIP_CONFIG`.
3. in the global default directory `/etc`.

The configuration file has 2 possible kinds of sections; all sections are optional.

Address Translation Sections

This kind of section determines how CommonAPI addresses are translated into SOME/IP addresses (service identifier and instance identifier). The name of the section is the CommonAPI address and the parameters are:

- `service`
- `instance`
- `major`
- `minor`

Example:

```
[local:de.ABC:v1_1:de.app1]
service=0x1234
instance=0x5678
major=1
minor=2
```

2.5.2 Deployment

As there is no predefined translation of CommonAPI addresses to SOME/IP addresses, you need to specify translations for the service instances you want to use. The same is true for all methods identifier, event identifiers and attribute getter and setter method identifiers. Thus, for each interface you need to specifiy the SOME/IP service identifier:

```
import "platform:/plugin/org.genivi.commonapi.core/deployment/CommonAPI-4- ↩
    SOMEIP_deployment_spec.fdepl"

define org.genivi.commonapi.someip.deployment for interface <CommonAPI interface name> {
    SomeIpServiceID = <id>

    method <CommonAPI method name> {
        SomeIpMethodID = <id>
    }

    attribute <CommonAPI method name> {
        SomeIpGetterID = <id>
        SomeIpSetterID = <id>
        SomeIpNotifierID = <id>
        // All these ID settings are optional, but you need to specify at least one
        // if you specify a SomeIpNotifierID you also need to specify an eventgroup
        SomeIpEventGroups = { <id> }
    }
}

define org.genivi.commonapi.someip.deployment for provider as <Name> {
    instance <CommonAPI interface name> {
        InstanceId = <CommonAPI instance name>
        SomeIpInstanceID = <id>
    }
}
```

The easiest way to define a complete deployment is to use the "auto-complete" function of the Eclipse editor for .fidl/.fdepl files. It will insert all mandatory elements and you only need to provide the settings for them.

For a better understanding of the SOME/IP deployment parameters please refer to the SOME/IP specification at http://some-ip.com/. Here are the most important rules:

- Method identifiers must be unique within the interface but must also be unique considering the extensions. Extended interfaces must not have the same method identifiers as base interfaces. Method identifiers are: SomeIpMethodID, SomeIpGetterID and SomeIpSetterID.
- The range of method identifiers must be 1 to 32767.
- Event identifiers must be unique within the interface (also when extensions are considered). The range must be 32769 to 65534. Event identifiers are SomeIpEventID and SomeIpNotifierID.
- Every selective broadcast must have its own event group; there must not be two selective broadcasts in the same eventgroup within one interface (again considering extensions).
- The eventgroup identifier (SomeIpEventGroups) must at least 1; each event (broadcast, attributes with notifier) must be at least in one event group.
- If attributes have a notifier they must be in at least one event group.
- Getter and setter identifiers of attributes must be consistent concerning the attribute keywords readonly and nosubscriptions.

Note
All these rules which can be checked within one interface and its dependencies are validated by the CommonAPI-SomeIp code generator (warning). There are additional rules which are not checked at the moment because they concern all applications or services in one device; the user has to take care of it without support from the code generator: - The combination of SomeIpServiceID and SomeIpInstanceID must be unique in the SOME/IP network. - Do not deploy two instances of the same interface on the same port.

2.6 Windows

2.6.1 Build vsomeip

You need to start with building boost 1.54 or higher (at least the log, system and thread libraries). Then execute within the vsomeip source directory:

- mkdir build
- cd build
- cmake ..

to create a Visual Studio 2013 solution. Finally, open the solution, adapt the pathes to boost and build vsomeip.

2.6.2 Build CommonAPI

Execute within the CommonAPI source directory:

- mkdir build
- cd build
- cmake ..

to create a Visual Studio 2013 solution. Finally, open the solution and build CommonAPI.

2.6.3 Build CommonAPI-SomeIP

Execute within the CommonAPI-SomeIP source directory:

- mkdir build
- cd build
- cmake ..

to create a Visual Studio 2013 solution. Finally, open the solution and build CommonAPI-SomeIP.