CommonAPI D-Bus 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 D-Bus supports also 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 D-Bus binding requires the D-Bus (*libdbus* at compile time for linking and the running D-Bus daemon at runtime). The D-Bus version should be at least 1.6.x.

2.2 Dependencies

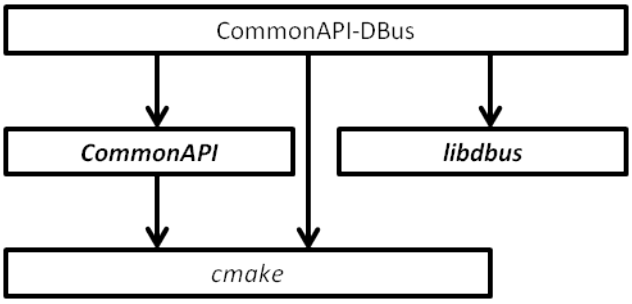

Figure 1: CommonAPI-D-Bus Dependencies

2.3 Restrictions

- The signature of D-Bus data must not be longer than 255 characters.

2.4 Compile Runtime

2.4.1 Patching libdbus

CommonAPI-D-Bus needs some API functions of libdbus which are not available in actual libdbus versions. For these additional API functions it is necessary to patch the required libdbus version with all the patches provided in the directory src/dbus-patches.

Note
Use autotools to build libdbus.

```
$ wget http://dbus.freedesktop.org/releases/dbus/dbus-<VERSION>.tar.gz
$ tar -xzf dbus-<VERSION>.tar.gz
$ cd dbus-<VERSION>
$ patch -p1 < </path/to/CommonAPI-D-Bus/src/dbus-patches/patch-names>.patch
$ ./configure --prefix=</path to your preferred installation folder for patched libdbus>
$ make -C dbus
$ make -C dbus install
$ make install-pkgconfigDATA
```

You can change the installation directory by the prefix option or you can let it uninstalled (skip the `make install` commands). WARNING: Installing the patched libdbus to /usr/local can prevent your system from booting correctly at the next reboot.

2.4.2 Command-line

In order to build the CommonAPI-D-Bus Runtime library the pkgconfig files of the patched libdbus library must be added to the `PKG_CONFIG_PATH`.

For example, if the patched libdbus library is available in /usr/local, set the `PKG_CONFIG_PATH` variable as follows:

```
$ export PKG_CONFIG_PATH="/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH"
```

Now use CMake to build the CommonAPI D-Bus runtime library. We assume that your source directory is `common-api-dbus-runtime`:

```
$ cd common-api-dbus-runtime
$ mkdir build
$ cd build
$ cmake -DUSE_INSTALLED_COMMONAPI=ON -DCMAKE_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 D-Bus runtime library libCommonAPI-DBus.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. The unit tests of CommonAPI D-Bus are implemented by using the Google C++ Testing Framework. If you want to build and run the unit tests the environment variable `GTEST_ROOT` must point to the correct directory (see the contributor's guide below).

Note
If you prefer to install CommonAPI D-Bus from a tar file you can get the actual tar file from: http://docs.projects.genivi.org/-yamaica-update-site/CommonAPI/runtime/

There are several options for calling CMake and make targets.

If you do not want to use nor the installed dbus library and neither the installed CommonAPI library:

```
$ cmake -DUSE_INSTALLED_DBUS=OFF -DUSE_INSTALLED_COMMONAPI=OFF ..
```

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

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

Generate makefile for building the release version of CommonAPI D-Bus (default is debug).

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

Without any further settings `make install` will copy CommonAPI D-Bus 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.

2.4.3 Eclipse

Follow the instructions in the CommonAPI User Guide.

2.5 Compile tools

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

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

COREPATH is the directory, that contains the target definition folder: org.genivi.commonapi.core.target.

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

2.6 Build DBus 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 D-Bus it is *org.genivi.commonapi.dbus.verification* in CommonAPI-D-Bus-Tools):

1. The cmake call below generates CommonAPI code for all requested fidl files using the CommonAPI- core and DBus code generator. The fidl files for the verification tests can be found in *org.genivi.commonapi.core.verification/fidl*.
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_TOOL_GENERATOR`	Core code generator executable with path
`COMMONAPI_DBUS_TOOL_GENERATOR`	DBus Code generator executable with path

Example to build DBus glue code for the verification tests:

```
export PKG_CONFIG_PATH=path/to/patched/libdbus:$PKG_CONFIG_PATH

cd CommonAPI-D-Bus-Tools/org.genivi.commonapi.dbus.verification/
mkdir build
cd build

cmake \
-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_DBUS_TOOL_GENERATOR=myworkpath/CommonAPI-D-Bus-Tools/org.genivi.commonapi.dbus. ↩
    cli.product/target/products/org.genivi.commonapi.dbus.cli.product/linux/gtk/MYARCH/ ↩
    commonapi-dbus-generator-linux-MYARCH \
-DCommonAPI_DIR=myworkpath/CommonAPI/build \
-DCommonAPI-DBus_DIR=myworkpath/CommonAPI-D-Bus/build \
-DCOMMONAPI_TEST_FIDL_PATH=myworkpath/CommonAPI-Tools/org.genivi.commonapi.core. ↩
    verification/fidl ..

make -j4
```

2.7 Project Setup

2.7.1 Configuration

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

1. in the directory of the current executable. If there is a `commonapi-dbus.ini` file, it has the highest priority.
2. in the directory which is specified by the environment variable `COMMONAPI_DBUS_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 D-Bus addresses (object path, interface name and so on). The name of the section is the CommonAPI address and the parameters are:

- `service`
- `path`
- `interface`

Example:

```
[local:de.ABC:de.app1]
service=de.ABC_de.app1
path=/de/app1
interface=de.ABC
```

If there is no address configuration for a certain CommonAPI address available the CommonAPI address will be translated by default as shown in the following picture:

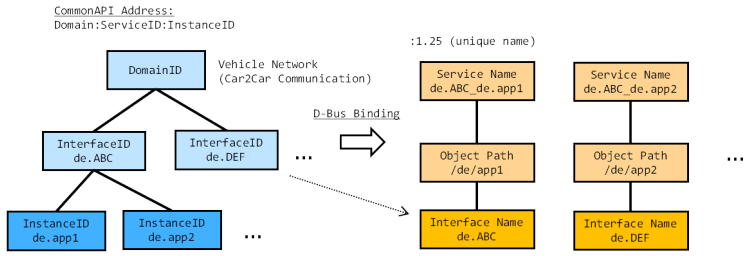

Figure 2: D-Bus Addresses

There might be some confusion concerning the terms and definitions:

- Franca IDL and CommonAPI define interfaces; these interfaces can be instantiated once or several times in one or several services. The CommonAPI term *interfaceID* and the D-Bus term *interface name* are equivalent.
- The D-Bus *object path* is a name used to refer to an object instance. The *object names* are organized into a filesystem-style hierarchy. The *object path* corresponds to the CommonAPI *instanceIDs*.
- Additionally D-Bus services are "connected" to the bus with an automatically created internal so-called unique bus name (e.g. :1.25). For this bus name one or several well-known names may be defined (here it is called service name). The parameter *service* in the configuration file defines this well-known name; the default name is *interfaceID___instanceID*.

Connections

Section for assigning a certain D-Bus bus segment (Session, System, Starter) to the connection name.

2.7.2 Deployment

The D-Bus code generator can generate code completely without any deployment settings. But for the combination of CommonAPI applications and D-Bus applications which are written without CommonAPI there are some deployment parameters necessary.

```
import "platform:/plugin/org.genivi.commonapi.core/deployment/CommonAPI_deployment_spec. ↩
    fdepl"

specification org.genivi.commonapi.dbus.deployment extends org.genivi.commonapi.core. ↩
    deployment {
    for interfaces {
        PropertiesType: {CommonAPI, freedesktop} (default: CommonAPI);
    }
}
```

Use the deployment parameter *PropertiesType* for the support of implementations of the *org.freedesktop.DBus.Properties* interface.

2.8 Windows

2.8.1 Build D-Bus

To build the Windows version of D-Bus the following steps have to be done:

- Download D-Bus from http://www.freedesktop.org/wiki/Software/dbus/ e.g. dbus-1.8.0.tar.gz and unpack the archive into a directory.
- For building D-Bus, CMake is needed. Download CMake from http://cmake.org/cmake/resources/software.html. The result of the download is e.g. cmake-2.8.12.2-win32-x86.zip. Unpack the archive into a directory.
- Furthermore `expat` is needed. `Expat` can be downloaded from http://expat.sourceforge.net. Example: download expat-win32bin-2.1.0.exe from http://sourceforge.net/projects/expat/files/expat_win32/2.1.0 and install it.
- Now the Windows D-Bus patch is needed. Download the patch from http://gnuwin32.sourceforge.net/packages/patch.htm e.g. patch-2.5.9-7-setup.exe and install the patch.
- Apply all the CommonAPI D-Bus patches (located in *CommonAPI-D-Bus/src/dbus-patches*) e.g. call

```
cd dbus-1.8.0
find ../CommonAPI-D-Bus/src/dbus-patches/*.patch | xargs -n1 patch -p1 -F3 -i
```

(assuming that *CommonAPI-D-Bus* is on same folder level as *dbus-1.8.0*)

- Build D-Bus via the CMake command line tool: — Open the developer command line for VS2013. — Change into root directory of the unpacked D-Bus archive e.g. dbus-1.8.0. — Create a new directory dbus-build-dir e.g. `mkdir dbus-build-dir`. — Change into this new created directory e.g. `cd dbus-build-dir`. — Create the Visual Studio files for building the D-Bus:

```
cmake -DEXPAT_INCLUDE_DIR="<expat-path>\Source\lib"
        -DEXPAT_LIBRARY="<expat-path>\Bin\libexpat.lib" -G "Visual Studio 12" < ↩
            DBusSourcenPath>\cmake
```

Example:

```
C:\Work\cmake-2.8.12.2-win32-x86\bin\cmake.exe
    -DEXPAT_INCLUDE_DIR="C:\Program Files (x86)\Expat 2.1.0\Source\lib"
    -DEXPAT_LIBRARY="C:\Program Files (x86)\Expat 2.1.0\Bin\libexpat.lib"
    -G "Visual Studio 12" C:\Work\dbus-1.8.0\cmake
```

The Visual Studio files are created in the directory dbus-build-dir. Open the solution dbus.sln and build all projects. Info for dbus-1.8.0: In file printf.c the #include "test-utils.h" must be changed to #include "test/test-utils.h".

2.8.2 Build CommonAPI

To build now the Windows version with Visual Studio 2013, you can use the solution file CommonAPI-DBus.sln. It is configured to build CommonAPI-D-Bus as a static library.

Before opening the solution file CommonAPI-DBus.sln the following environment variables must be set:

- DBUS_DIR: directory containing the patched D-Bus sources e.g. <path_to_folder>\dbus-1.8.0
- DBUS_BUILD_DIR: directory containing the build DBus e.g. <path_to_folder>\dbus-1.8.0\dbus-build-dir
- COMMONAPI_DIR: directory containing the CommonAPI e.g. <path_to_folder>\CommonAPI

To run the tests, you need to add the location of your built dbus-1d.dll or dbus-1.dll (depending on Debug or Release built) to the PATH environment variable.

2.8.3 Restrictions

- Calling disconnect and later connect on the same DBusConnection somehow damages the libdbus connection. On Linux implementation this all works fine.
- The DBusLoadTest revealed that a number of more than about 50 proxies slows down the dbus communication significantly. That's why we run the DBusLoadTest on windows with just 50 instead of 100 proxies.