Build Process

Description

The following chapters provide some information about build process aspects.

Using CMake for Building Courier resp. Courier Applications

General Setup

Courier including it's sample applications provides a complete cmake environment for targeting multiple platform setups (compiler, OS, etc.). To start the CMake process either the command-line tool (cmake) or the CMake GUI (cmake-gui) can be used. As source code entry point the location of the application's source root folder has to be set. In case building Courier standalone this entry point is the root folder of Courier (more on this in the next section Building Courier Standalone).

When using the CMake GUI you will encounter notifications (aka. CMake error) to

1. Define the *COURIER_PLATFORM* of choice , based on the platforms defined in the CourierPlatformDefs folder (see Platform Definition).
2. Define the *COURIER_PLATFORM_CONFIG*, based on the platform configurations supported by the selected platform (e.g. Host, Target, etc.)

When you encounter these notifications, interactively select the platform and the platform configuration of your choice and generate the build tool chain.

To avoid these notifications you may use the command line call for CMake instead, e.g.

Use own Platform Definition

If you want to access your own platform definitions, which are not located in the pre-defined folder *CourierPlatformDefs*, add the CMake parameter *COURIER_EXTENDED_SEARCH_PATHS*. Afterwards you can define your custom platform definition, e.g.

```
cmake -G "Visual Studio 15 2017 Win64" -
DCOURIER_EXTENDED_SEARCH_PATHS="<MyOwnPlatformDefFolder>" -
DCOURIER_PLATFORM=<MyCustomPlatform> -DCOURIER_PLATFORM_CONFIG=<AnyOfCustomPlatformConfigs> -
DCMAKE_BUILD_TYPE=Debug "<MyWorkspaceFolder>\cgi_studio_courier\src\Courier"
```

> You can set any of the CMake parameter defined in Courier in the command-line call.

Cross Compilation

CMake also supports cross compilation. The easiest way to handle this via CMake is to define a cross compilation toolchain file. CGI-Studio normally provides a basic set of toolchain files for the supported target cross compilations, which are located in *./cgi_studio_candera/cmake/toolchain-files*. In case there is none matching your setup, you can easily create your own toolchain file.

Here is an example:

```
include(CMakeForceCompiler)

# The name of the target operating system.
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR ARM)

# Which C and C++ compiler to use.
set(CMAKE_C_COMPILER <location to the C cross compiler of choice>)
set(CMAKE_CXX_COMPILER <location to the C++ cross compiler of choice>)
```

> Courier only needs the basic set of cross compilation information in the toolchain file, which you can see in the example above. If you like to see more settings (Candera resp. CGI Application specific settings) in a CGI-Studio toolchain file, you have to know, that these are often prepared to configure target builds of a CGI Application without Courier. Courier itself does this configuration already in the Courier platform definition files (see Platform Definition).

Once the toolchain file is available, cmake can be started for cross compilation (if CMake variable CMAKE_TOOLCHAIN_FILE is set properly) in the output directory of choice.

See here as example a linux build script:

```
#!/bin/bash

ROOT_DIR="~/work/cgi_studio"
GENERATOR="Unix Makefiles"
TOOLCHAIN_FILE="${ROOT_DIR}/cgi_studio_candera/cmake/toolchain-files/<name of the toolchain file
PLATFORM=<Courier platform name to use>
PLATFORM_CONFIG=<Courier platform configuration to use>
APPLICATION_PATH="${ROOT_DIR}/cgi_studio_courier_apps/src/CourierSampleApp"
```

```
BUILD_TYPE=Debug

cmake -G "$GENERATOR" -DCMAKE_TOOLCHAIN_FILE=$TOOLCHAIN_FILE -DCOURIER_PLATFORM=$PLATFORM -DCOUF

make
```

If the build finishes successful, the target executable can be found in the output directory. In the above example the executable will have the name CourierSampleApp_<PLATFORM>_<PLATFORM_CONFIG>. This executable then has to be loaded together with the target Asset to the device (e.g. via rootfs), for which the application was built.

> In case of a Courier demo application *./cgi_studio_courier_apps/src* the asset is copied to the binary root folder of the build (where the executable resides), named *AssetLibCff.bin*.

Selecting Courier Features

> CMake feature switches have been renamed for V3.x releases, see Change Log

CMAKE Features

Since Courier2 (V2.0.0) the interaction framework was enhanced with new substantial functionalities. Three new optional Courier software modules are now available, which are reflected by their pendants - the Courier CMake features:

ENHANCED

While the Courier CMake feature `COURIER_ENHANCED_ENABLED` was inherent part of Courier1 (V1.x.x), it is now possible to disable it for using Courier without Candera graphics engine. The only additional library, which is needed in such a setup is libFeatStd (actually every CGI Studio software product relies on libFeatStd).

The enhanced mode enabled activates the link to CGI-Studio's graphics engine Candera and enables in Courier the modules Visualization and DataBinding. This enhanced mode reflects the feature-set provided with previous Courier (1.x) versions.

Turn on the Courier CMake feature `COURIER_ENHANCED_ENABLED` (either via CMake-gui or the CMakeLists.txt file of the respective application) to have this mode available.

IPC

Courier also supports IPC with Courier messages, which is supported for Windows, Linux and Integrity environments. Courier IPC abstracts OS specific mechanisms to transport the serialized message over process borders. This IPC is (currently) limited to be used within one processor. To enable this functionality the CMake feature `FEATSTD_IPC_ENABLED` has to be set.

For this IPC mechanism the Courier generator was enhanced to generate the IPC infrastructure for the applications, which want to communicate via IPC.

MONITOR

Courier2 introduces an additional CMake feature, which is about monitoring messaging and rendering performance. Both monitoring features can be enabled separately having their distinct Courier CMake feature, called:

- `COURIER_MESSAGING_MONITOR_ENABLED`
- `COURIER_RENDERING_MONITOR_ENABLED`

These features are prepared to be connected to the CGI Studio Analyzer tool, if this optional CGI Studio module is available and activated via the respective Courier platform definition file using the CGI CMake flag `FEATSTD_MONITOR_ENABLED` (V2.x: `CGIFEATURE_ENABLE_MONITOR`).

Additionally if CGI Studio Analyzer option is not available, the Courier messaging monitor feature provides also a native support, while the rendering monitor is not available natively.

> Using CMake-GUI both monitoring features are available only if `FEATSTD_MONITOR_ENABLED` is set to **ON** in the respective platform definition file. If `FEATSTD_MONITOR_ENABLED` is set to **OFF**, then only the `COURIER_MESSAGING_MONITOR_ENABLED` is visible.

Building Courier Standalone

For building Courier in standalone mode simply start CMake pointing to the folder

```
./cgi_studio_courier/src/Courier/
```

After generating the buildsystem via CMake and building the code using the according build tool chain, the Courier platform dependent library is available. Additionally this solution includes the complete Candera, so it is easy to provide also the Candera libraries within one build step.

Remarks:

If the Courier library is build standalone and linked as external library to an customer application code, Courier has generated header files (e.g. Config.h and Version.h), which are created to the Courier build output directory

```
<CourierOutputDirectory>/gensrc/Courier
```

and have to be in the include path of the customer Courier application. To solve this issue the folder

```
<CourierOutputDirectory>/gensrc
```

has to be added to the customer application's include paths. (To copy the files to cgi_studio_courier/src/Courier may also be possible, but be aware that these generated files can be highly build configuration dependent!)

Building Courier together with an Application Solution

When using this option, there will be a dedicated CMakeLists.txt file for the Courier application available. Courier has defined a set of CMake macros which shall ease the CMake configuration in the application.

Take a look at the following template to see what this may look like. The comments in the section explain the various steps in detail.

```
cmake_minimum_required(VERSION 3.8)
cmake_policy(VERSION 3.8)

# Requested Courier version
set(APP_COURIER_VERSION 3.12.0.0)

# ----------------------------------------------------------------------------
# Enable optional Courier features
set(COURIER_ENHANCED_ENABLED ON)

# Required to resolve root paths of CGI Studio products
set(CGI_ROOT_PATH_HINT "../../.." "../.." "../") # Adapt to specific needs

# ----------------------------------------------------------------------------
# Checks for Couriesr package with given version in given paths
function(FindPackage pkg version path0)
```

```
    string(TOLOWER ${pkg} searchString)
    foreach(p ${path0} ${ARGN})
        file(GLOB tmp "${p}/*${searchString}*")
        list(APPEND paths ${tmp})
    endforeach()
    message(STATUS "Searching for ${pkg} ${version} in '${paths}' ...")
    find_package("${pkg}" ${version} REQUIRED PATHS ${paths})
endfunction()

# Search for Courier package
FindPackage(Courier ${APP_COURIER_VERSION} ${CGI_ROOT_PATH_HINT})

# ----------------------------------------------------------------------------
# Set customer specific search path for Courier platform definitions
set(CGI_ROOT_PATH "${CMAKE_CURRENT_LIST_DIR}/../../..")
set(COURIER_PLATFORMDEFS_DIR "${CGI_ROOT_PATH}/cgi_studio_devices/src")

# If COURIER_PLATFORMDEF_FILE is not set, pre-select one of the available platforms
if (NOT COURIER_PLATFORM)
#    Preselection is disabled, as application can be used for different platforms
#    set(COURIER_PLATFORM "<anyPlatform>")
endif()

# Pre-select platform configuration
if (NOT COURIER_PLATFORM_CONFIG)
#    Preselection is disabled, as SampleApp can be used for different platforms
#    set(COURIER_PLATFORM_CONFIG "<anyPlatformConfig>")
    if (CMAKE_SYSTEM_NAME STREQUAL Windows)
        set(COURIER_PLATFORM_CONFIG "Simulation_Win32_x64")
    endif()
endif()

# ----------------------------------------------------------------------------
# Include Courier build system
include("${Courier_DIR}/cmake/Courier.cmake")

# ----------------------------------------------------------------------------
# Enable solution folder representation in Visual Studio
#CgiEnableSolutionFolderSupport(ON)
```

```
# ------------------------------------------------------------------------
# Set asset locations
if (CGIDEVICE_TARGET_BUILD)
    set(ASSET_POSTFIX Target)
else()
    set(ASSET_POSTFIX Simulation)
endif()
set(ASSET_SOURCE_LOCATION
"${CMAKE_CURRENT_LIST_DIR}/Assets/AssetLibCff_${CGIDEVICE_NAME}_${ASSET_POSTFIX}.bin")
set(ASSET_TARGET_LOCATION ${CMAKE_BINARY_DIR}/AssetLibCff.bin)

# ------------------------------------------------------------------------
# Courier application solution (incl. build of SCHost.dll)
CourierSolution("CourierSampleApp")

    # ------------------------------------------------------------------------
    # Generate application's widget library
    CourierAddProjectFiles(
        AppCDL.h
        AppCDL.cpp
        AppCDL.xcdl
        Constants.cpp
        Constants.h
        ComponentMessageReceiverThread.h
        CustomComponentId.h
        ExtCommOutputHandler.h
        ExtCommOutputHandler.cpp
        ListItemDataPresenter.h
        ListItemDataPresenter.cpp
        RenderStatisticsOverlayWrapper.h
        RenderStatisticsOverlayWrapper.cpp
        SampleApp.cpp
        SampleApp.h
        SampleAppControllerComponent.cpp
        SampleAppControllerComponent.h
        SampleExtCommOutputHandler.cpp
        SampleExtCommOutputHandler.h
        SampleAppLogRealm.cpp
        SampleAppLogRealm.h
        SampleAppMsgs.cpp
```

```
    SampleAppMsgs.h
    SampleAppMsgs.xcmdl
    SampleAppViewController.cpp
    SampleAppViewController.h
    SampleAppViewControllerFactory.cpp
    SampleAppViewControllerFactory.h
    SampleAppViewFactory.cpp
    SampleAppViewFactory.h
    SampleTypeConverter.cpp
)

# Add screen broker related files
if(CGIDEVICE_ENABLE_SCREENBROKER)
    CourierAddProjectFiles(
        ScreenBrokerClient.cpp
        ScreenBrokerClient.h
    )
endif()

# Bind widgets
CgiAddProject(Widgets)

# Add subfolder sources
CourierAddListFile("DataModel/DataModel.cmake")
CourierAddListFile("AppPlatform/AppPlatform.cmake")

# Add Memory Pool specific files
if(FEATSTD_MEMORYPOOL_ENABLED)
    CourierAddProjectFiles(
        MemoryPoolConfig.cpp
    )
    CourierAddListFile("MemoryPoolUtil/MemoryPoolUtil.cmake")
endif()

# Add IPC resources
if(FEATSTD_IPC_ENABLED)
    CourierAddProjectFiles(
        IpcCDL.cpp
        IpcCDL.h
        IpcCDL.xcdl
```

```
            IpcMsgs.cpp
            IpcMsgs.h
        )
        CourierAddListFile("Ipc/Shared/Shared.cmake")
    endif()

set(CGIAPP_TYPE_NATIVE_DLL "NativeDll")
set(CGIAPP_TYPE_NATIVE_EXE "NativeExe")
set(CGIAPP_TYPES ${CGIAPP_TYPE_NATIVE_EXE} ${CGIAPP_TYPE_NATIVE_DLL})
set(CGIAPP_BUILD_TYPE ${CGIAPP_TYPE_NATIVE_EXE} CACHE STRING "Binary type of the built
application, one of: ${CGIAPP_TYPES}")
set_property(CACHE CGIAPP_BUILD_TYPE PROPERTY STRINGS ${CGIAPP_TYPES})

    if (${CGIAPP_BUILD_TYPE} STREQUAL ${CGIAPP_TYPE_NATIVE_DLL})
        CgiSetSolutionFolderName("CourierApplication")
        CourierAddDynamicLibrary(CourierSampleAppLib)
        CgiCreateOutputName(PRV_APP_TARGET_NAME CourierSampleAppLib)
    else()
        CgiSetSolutionFolderName("CourierApplication")
        CourierAddStaticLibrary(CourierSampleAppLib)
        # add application top level source files
        CourierAddProjectFiles(
            Main.cpp
        )
        CourierAddExecutable(CourierSampleApp TARGET_NAME COURIER_APPLICATION_TARGET)
        CgiCreateOutputName(PRV_APP_TARGET_NAME CourierSampleApp)
    endif()

    # ------------------------------------------------------------------------
    # Include target specific build targets
    # ------------------------------------------------------------------------
    if (CMAKE_SYSTEM_NAME STREQUAL Integrity)
        set(COURIER_KERNEL_TARGET_DIR
"${COURIER_PLATFORMDEFS_DIR}/${CGIDEVICE_NAME}/CourierPlatformDefs/${COURIER_PLATFORM_CONFIG}/
KernelTargets")
        include("${COURIER_KERNEL_TARGET_DIR}/KernelTargets.cmake")
    endif()

    # ------------------------------------------------------------------------
    # Generate application's SCHost dynamic link library project for scene composer
```

```
    CourierAddSCHostLibrary()

    CgiClearSolutionFolderName()
CourierSolutionEnd()

#! [COURIER_XCDLGeneration]
# ------------------------------------------------------------------------------
# Generate h/cpp sources from XCDL files
#   Attention: As soon as the generation output dir (3rd parameter) is changed,
#              the courier messages cannot be generated any more!
#              SampleAppMsgs.xcmdl has to be adapted to that effect.
if (FEATSTD_IPC_ENABLED)
    CourierGenerateFromXcdl(CourierSampleAppLib ${CMAKE_SOURCE_DIR}/IpcCDL.xcdl
${CMAKE_SOURCE_DIR} -i ${COURIER_SRC_DIR} -gm -gi -gfa -v)
endif()

CourierGenerateFromXcdl(CourierSampleAppLib ${CMAKE_SOURCE_DIR}/AppCDL.xcdl
${CMAKE_SOURCE_DIR} -i ${COURIER_SRC_DIR} -gm -gd -gfa -v)
#! [COURIER_XCDLGeneration]

# ------------------------------------------------------------------------------
# Copy device specific asset to working dir (with target name AssetLibCff.bin)
if (EXISTS "${ASSET_SOURCE_LOCATION}")
    if (${CGIAPP_BUILD_TYPE} STREQUAL ${CGIAPP_TYPE_NATIVE_DLL})
        CourierLog0("Asset file found at ${ASSET_SOURCE_LOCATION}. Copy to binary dir
manually!")
    else()
        CourierLog0("Asset file found at ${ASSET_SOURCE_LOCATION}. Copying to binary dir")
        add_custom_command(TARGET ${COURIER_APPLICATION_TARGET}
                           PRE_LINK
                           COMMENT "Copy AssetLibCff.bin"
                           COMMAND ${CMAKE_COMMAND} -E copy_if_different
${ASSET_SOURCE_LOCATION} ${ASSET_TARGET_LOCATION}
                           VERBATIM)
    endif()
else()
    CourierLog0("No asset file found at ${ASSET_SOURCE_LOCATION}. Cannot copy to
${ASSET_TARGET_LOCATION}")
endif()
```

Solution Folder

An interesting option (for Visual Studio only) is the solution folder support, which groups projects to logical groups (aka. solution folder). Though Courier implements the solution folders internally, the option itself is turned off by default (solution folders are not supported by Visual Studio Express edition installations and may cause problems using Express Editions). It can easily be enabled by calling the Courier CMake macro *CgiEnableSolutionFolderSupport()*.

Alternatively setting the environment variable *CGI_STUDIO_ENABLE_SOLUTION_FOLDERS* to value 1 enables the solution folder support without changing cmake files. With CGI_STUDIO_ENABLE_SOLUTION_FOLDERS variable solution folder support can be enabled per user / workstation.

> After CGI_STUDIO_ENABLE_SOLUTION_FOLDERS has been defined, CMake needs to be restarted and the solution be re-generated.

If various application libraries shall be grouped, then the scoping macros *CgiSetSolutionFolderName()* and *CgiClearSolutionFolderName()* can be used. The example above already makes use of the options mentioned here.

Building SCHost.dll

How to build an SCHost.dll for the CGI SceneComposer is described in this chapter.

It is possible to build an additional SCHost.dll for SceneComposer (containing the widgets) by adding the following simple cmake macro to the of the application CMakeLists.txt:

```
    CourierAddSCHostLibrary()
```

right after the entry

```
    CourierAddExecutable(<solutionName>)
```

> It is important to separate the application project in a widget library and an application executable. A simple guideline for the separation is to put only the main entry point (e.g. main.cpp) into the executable and the rest of the application code into the application (widget) library. You can also use a different separation if you so desire, but please make sure that all widget dependent code resides in the application widget library.

As mentioned above the application widget library must be self-contained as it shall be able it to the SCHost.lib without any link error. Just take the CMakeList.txt of the CourierSampleApp as a reference (see Platform Configuration & Build Process).

Having the macro *CourierAddSCHostLibrary()* applied in the applications CMakeLists.txt file you will find a new CMAKE option in the Courier section named **COURIER_ADD_SCHOST_PROJECT**. Enable this option, configure until there is no configuration line marked red and generate the solution.

> SCHost generation is only applicable (at least tested) with the CMake generators for Visual Studio versions that are mentioned in the release notes.

When opening your Visual Studio project you see an additional **CourierSCHost** solution folder, which contains the project **CourierSCHostDll**. This DLL project links the applications widget project (e.g. CourierSampleAppLib inside CourierSampleApp) and the pre-built static Courier SCHost library (provided at **cgi_studio_courier/lib** folder).

If you also apply the correct SceneComposer binary path in the CMAKE configuration (before generating the project), see CMAKE option **CGIAPP_SCENE_COMPOSER_PATH**, the built DLL is automatically renamed to SCHost.dll and copied into the SceneComposer binary folder. If this path is not set, the built DLL CourierSCHostDll_<COURIER_PLATFORM>_<COURIER_PLATFORM_CONFIG>.dll has to be renamed to SCHost.dll and copied to the SceneComposer binary folder manually.

For details on developing widgets please also refer to Candera documentation.

Building SCHost.dll when using Memory Pools

As a SCHost build using the Memory Pool feature is not binary compatible to a build not using memory pools, a new restriction is added for building SCHost.dll:

- In the solution for building SCHost.dll, the feature FEATSTD_MEMORYPOOL_ENABLED must not be set!

As an example, the Courier SampleApp platform configuration has been extended to provide two different host simulation configurations:

- ***Simulation_Win32_x64***: Disabled memory pool feature, option **COURIER_ADD_SCHOST_PROJECT** available. Use this one to integrate your widgets to SceneComposer.
- ***Simulation_Win32_x86_MP***: Enabled memory pool feature, but no option to add the Courier SCHost project available. Use this one to integrate and test memory pool

configurations on host simulation as preparation for the target.

Adding External Widgets

Adding External Widgets to SampleApp

In the last major update it is now possible to add additional external widgets (e.g. from a widget repository) to the application. For this reason the widgets of CourierSampleApp are collected in a separate sub-project enhancing the application with the possibility to add additional external defined widget libraries to be linked to the application.

The usage is basically the same like it is already available in CGIApplication (without Courier).

This means a CMake file variable (`CGIAPP_ADDITIONAL_WIDGET_PATH`) was added pointing to a file, which contains all the additional widget paths. If the paths in this file are valid and the second CMake flag `CGIAPP_ENABLE_ADDITIONAL_WIDGETS` is checked, than all widgets found in these folders resp. subfolders are added as separate projects to the solution. The format of defining these widgets follow the same structure as it is already used in CGIApplication resp. the sample widget repository cgi_studio_widgets.

The CourierSampleApp got a new folder named Widgets, which includes the basic CMake infrastructure for this new feature. Additionally the already used (and adapted) widgets (SampleWidget and SpeedWidget) and the base classes (CgiWidget2D & CgiWidget3D) are moved to this new Widgets folder.

The WidgetSet itself was removed from the CourierSampleApp source code, and will be replaced with a generated one (based on all the widgets linked to the project), which is located in ${CMAKE_BINARY_DIR}/gensrc/CanderaGen/WidgetSet.cpp.

Migration Info of "Binding of External Widget Libraries"

Migration info of the "Binding of external widget libraries" feature to any other (customer) Courier/CIT application (called in the following <myProject>):

1. Copy Folder CourierSampleApp/Widgets to <myProject>/Widgets (optional: remove CourierSampleApp's widgets: SampleWidget and SpeedWidget).
2. Move project (local) owned widgets (if already available) to <myProject>//Widgets.
3. Remove project (local) owned widget base classes (CgiWidget2D & CgiWidget3D) and WidgetSet.cpp.
4. Adapt the file AdditionalWidgetsPaths.txt to your needs or let the CMake variable `CGIAPP_ENABLE_ADDITIONAL_WIDGETS` point to a custom provided additional widgets file.
5. Optional: adapt <myProject>//Widgets/CMakeLists.txt, if the additional widgets project name shall be changed, resp. the local widgets search path shall be adapted.

6. To build the new SCHost.dll, don't forget to check the Courier CMake flag `COURIER_ADD_SCHOST_PROJECT`.
7. Compile and build the solution (or at least the project CourierSCHostDll_<myProject>_<myProject>).
8. If the CMake variable `CGISTUDIO_SCENE_COMPOSER_PATH` was set (or found correctly) the newly built CourierSCHostDll.dll is automatically copied to the previous mentioned SceneComposer path (with name SCHost.dll).
9. Start SceneComposer and enjoy the newly added widgets.

Revision #14
Created 2023-02-20 05:17:05 UTC by Tsuyoshi.Kato
Updated 2025-01-10 11:22:47 UTC by Elisabeth Zauner