Building Player, Widgets, and SCHost

- Introduction
- Building CGI Panel
- Building Player and Widgets
- Activating Additional Behaviors for Player
- Building SCHost for SceneComposer
- Customizing Widgetset Generation
- BehaviorSet Library with Behavior Set Parts
- Candera Version Validation
- Using Player on 64 bit Systems
- Debugging Hints
- Used Behaviors Plugin

Introduction

CGI Studio provides an integrated development environment based on Visual Studio for widget and application layer development, located at:

- <cgi-studio-root>/cgi_studio_player

In the cgi_studio_player directory, 4 different projects are defined:

- **CGI Panel**
  - A graphical user interface which controls the flow of the Player
  - C# Application with predefined controlling interface towards the Player
- **Player**
  - The application layer (C++) links Candera, demo application and Widgets as an executable for host system
  - Player depends on CGI Panel, hence it refers to CGI Panel as "debug" application and stores its output near CgiPanel.exe in order to enable CgiPanel.exe to execute it.
- **Behaviors_Examples**
  - The concrete behavior set implementation (C++)
  - Based on Candera and compiled as a static library
- **SCHost**
  - Makes SceneComposer aware of Candera including the Candera Device used in the solution, and widgets
  - Compiled as an dll and loaded by SceneComposer
  - Output dll is placed directly into SceneComposer installation directory to avoid manual copying.

The following pages explain how to build these components.

Building CGI Panel

The CGI Panel is a C# project which provides a graphical interface which supports the control of scenes and animations by managing the flow of the Player.

Building CGI Panel

Usually there is no need to build the CGI Panel on your own, since the executable is already provided for the Player build environment - refer to either:

- *<cgi_studio_root>/bin/Player/CGIPanel.exe or*
- *<cgi_studio_root>/bin/LightPlayer/Win32/CGIPanel.exe.*

Building CGI Panel

For building a new CGI Panel, refer to the following steps:

1. Start Visual Studio 2017 or higher and load solution *<cgi-studio-root>/cgi_studio_player/src/CGIPanel/CGIPlayer.sln*.
2. Build the project and then press F5 to start the CGI Panel in Debug Mode.

This solution allows the customer to extend the windows UI control of the Player.

Actually, the C# Panel is responsible for providing the following Player control panel:

For additional information regarding CGI Panel User Interface please take a look at CGI Panel User Interface - Overview.

Upon opening a specific asset library, the CGI Panel will attach to a Player for asset loading and content display. The next section explains how to create the application.

Building Player and Widgets

Building Player and Widgets

The Player executable used by CGI Panel collects following components:

- Candera Engine including Candera Device Platform
- Player as a demo application for loading assets and displaying / manipulating content: *<cgi_studio_root>/cgi_studio_player/src/Player*.

Build output will be named according to the Candera device platform configuration.

For example if you build for your device on host simulation, the following executable will be created:

- *<cgi_studio_root>/bin/Player/device/[your device]/**Player.exe***.

Load Assets in Player

After starting CGIPanel.exe from this location, you will be able to load assets generated for host simulation with the Player. For building the Player, refer to following steps:

1. Setup Visual Studio Solution "Player.sln"
   - Start CMake
   - Point "Where is the source code" to *<cgi-studio-root>/cmake/Candera/Player.*
   - Point "Where to build the binaries" to *<cgi-studio-root>/cgi_studio_player/build*. Avoid blanks in the path.
   - Click "Configure", select "Visual Studio 15" and specify "x64" as optional platform for generator. Confirm the dialog with OK. Some lines in the main area appears
   - Set "COURIER_PLATFORM" to the Device Package you want to use - e.g. "iMX6Platform"
   - Set "COURIER_PLATFORM_CONFIG" to build for host environment (default)
   - Set "CGIAPP_PLAYER_DEBUG_PATH" to the location, where the debug version of CGIPanel.exe resides (per default initialized to the path *<cgi-studio-root>bin/Player/CGIPanel.exe*).
   - Set "CGIAPP_PLAYER_RELEASE_PATH" to the location, where the release version of CGIPanel.exe resides (per default initialized to the path *<cgi-studio-root>bin/Player/CGIPanel.exe*).

- Set "CGIAPP_SCENE_COMPOSER_PATH" to *<cgi-studio-root>/bin/SceneComposer/*. This is the location where SCHost.dll will be copied after building.
- Click "Configure" again
- Click "Generate", the solution is generated

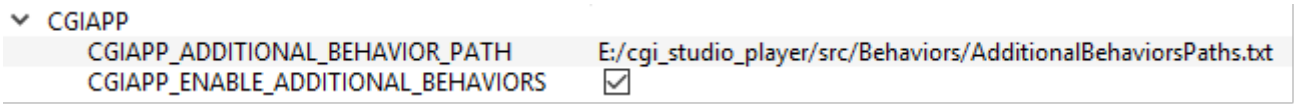
2. Start Visual Studio 2015 C++ and load generated solution from *<cgi-studio-root>/cgi_studio_player/build/Player.sln*.
3. Set Project "Player" as startup project in VS
4. Following project properties of project "Player" will be preconfigured by CMake, hence usually there is no need to adjust this manually:
   - Configuration Properties/Debugging/Command pointed to *<cgi-studio-root>/bin/Player/Device/<DeviceName>/Player.exe*
   - Configuration Properties/Debugging/Command Arguments pointed to *--run* "*<cgi-studio-root>/bin/Player/CGIPanel.exe --debug*" *-w*
   - Configuration Properties/Debugging/Working Directory pointed to *<cgi-studio-root>bin/Player*
5. In Player solution, press F5 to execute the application. After Player is started, load an asset generated by SceneComposer for the proper device platform to see if the build is working.

> If CGIPanel pops up but the asset content is not shown use the File->Attach menu to connect the panel to the Player.

Activating Additional Behaviors for Player

To include the behaviors part of CGI Studio in the Player, simple enable the CMake option **CGIAPP_ENABLE_ADDITIONAL_BEHAVIORS** before generating the CGI Application solution with CMake.

The CMake option **CGIAPP_ADDITIONAL_BEHAVIOR_PATH** is already preconfigured to include widgets part of *<cgi-studio-root>/cgi_studio_player/src/Behaviors/AdditionalBehaviorsPaths.txt*.



Following projects will be created in the solution with above cmake configuration:

- **BehaviorSet** for compiling WidgetSet.cpp
- **Behaviors_Example** for compiling example behaviors defined at *<cgi_studio_root>/cgi_studio_player/src/Behaviors*
- **ControlBehaviors_1** for compiling behaviors defined at *<cgi_studio_root>/cgi_studio_controls/src/Behaviors*

Building SCHost for SceneComposer

To make the same set of widgets available for SceneComposer, as they are for the Player, a new SCHost.dll needs to be created and deployed to SceneComposer.

> Enable COURIER_ADD_SCHOST_PROJECT in the CMake project in order to add the SCHost project to the Player solution.

That's the reason why the **SCHostDll** project is part of the C++ Player.sln, because this way the widget set configuration for the Player is identical to the one for SCHost.dll .

However, in difference to the Player, SceneComposer requires a specific set of Candera static libraries to link together with the widgets part of the solution. These specific libraries are created as part of the compilation process and collected into a library names **SCHostLib.lib**. The corresponding Visual Studio project is named **SCHostLib**.
When the SCHostDll project is built, **SCHostLib.lib** will be linked together with the widgets and the Candera Device lib part of the solution into the **SCHostDll.dll** library.

To automatically copy the generated **SCHostDll.dll** to your SceneComposer executable location as **SCHost.dll** after the build, take care to set the following CMake variable:

- Make sure to select "x64" as optional platform for generator during CMake configuration for Visual Studio versions lower than Visual Studio 2019 (version 16) in order to build with 64-bit compiler. Building SCHost.dll with 32-bit will result in an CMake error "*SCHost.dll will not work in 32-bit mode, please use CMake with a 64-bit platform*"
- "CGIAPP_SCENE_COMPOSER_PATH" to *<SC-Install-Dir>/SceneComposer/* .

- Close SceneComposer before building SCHost.dll, otherwise SCHost.dll cannot be copied to the SceneComposer location.
- SCHost.dll will contain exactly the Candera Device which is selected in CMake during Player solution generation.

>

> For using several platforms alternatively, it is also possible to specify the name of the SCHost dynamic link library as command line argument to SceneComposer. Example: "SceneComposer /schost device\iMX6\SCHost.dll"

SCHost and Memory Pools

As memory pools are introduced in FeatStd V1.2.0 resp. V1.1.0.3, and an SCHost build using this 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 a consequence, the build system for cgi_studio_dev will not add the SCHost project when Memory Pools are enabled.

- Disable FEATSTD_MEMORYPOOL_ENABLED to integrate your widgets to SceneComposer
- Enabled FEATSTD_MEMORYPOOL_ENABLED to integrate and test memory pool configurations on host simulation as preparation for the target

Customizing Widgetset Generation

Customizing Widgetset Generation

For adding your own behaviors and widgets to the default widget set, it is required to specify them in the widget set definition (WidgetSet.cpp), which is finally accessed by SceneComposer to provide all exported behaviors and widgets in the user interface.

The Candera CMake build system automatically generates this widget set definition (WidgetSet.cpp), so it is only required to provide behavior and widget implementations at any location and to specify this additional behavior and widget location for CMake.

This chapter explains how the widget set generation works.

Specifying Widget Set (WidgetSet.cpp)

The main widget set is a project to generate a library containing definitions for all behaviors and widgets to be used. Behavior and widget implementations can be part of other libraries which of course also need to be linked together with the widget set definition.

Refer to the CMake configuration of the Player demo widget set:

- *[CGI-Studio-Root]/cgi_studio_player/src/Behaviors/CMakeLists.txt* Following configurations are important:
- The WidgetSet project source path must point to "${PRV_CGI_GENERATED_CANDERA_DIR}/WidgetSet.cpp". This file will be generated by CMake including all widget implementations. No other files are required as they are added to a separate project.
- CMake macro CgiAddWidgetSet must be called to register the WidgetSet project. There is no need to call CgiCollectListedFiles as the project should only contain "WidgetSet.cpp".
- Take care to include the CMake file "${FeatStd_DIR}/cmake/cgistudio/cgistudio_widgetset_generator.cmake". For the Player Widgets, this is already done in *<cgi-studio-root>/cgi_studio_player/src/Behaviors/CMakeLists.txt*.
- CgiFetchHeaderFiles must be called to register the default widgets project. It has two parameters, the project name and the variable containing the include paths.

- CgiCreateWidgetSet must be called to generated the WidgetSet.cpp file. It has one parameter containing the CdaDescription which is written to the file.

Refer to the following example from the Player behaviors:

```
# Declare widget set class files
set(WidgetSet_SRC
    ${PRV_CGI_GENERATED_CANDERA_DIR}/WidgetSet.cpp
)
source_group("WidgetSet" FILES ${WidgetSet_SRC})

# Declare behavior set class files
set(BehaviorSet_SRC
    ${PRV_CGI_GENERATED_CANDERA_DIR}/WidgetSet.cpp
)
source_group("BehaviorSet" FILES ${BehaviorSet_SRC})


# Register widget set
CgiAddWidgetSet(PRV_WIDGET_TARGET_NAME_WIDGETSET WidgetsSet ${WidgetSet_SRC})

# Register behavior set
CgiAddWidgetSet(PRV_WIDGET_TARGET_NAME_WIDGETSET BehaviorSet ${BehaviorSet_SRC})


# Declare default widget directories (include here all your subdirectories
# containing behavior and/or widget implementations
include(${CGISTUDIO_BEHAVIORSET_CMAKE_INCLUDE})
if (CGIAPP_ENABLE_ADDITIONAL_WIDGETS)
    if ((EXISTS "${CGIAPP_ADDITIONAL_WIDGET_PATH}"))
        CgiAddAdditionalWidgets(${CGIAPP_ADDITIONAL_WIDGET_PATH} "Widgets_CgiStudioWidgetsLib")
    endif()
endif()



if (CGIAPP_ENABLE_ADDITIONAL_BEHAVIORS)
    if ((EXISTS "${CGIAPP_ADDITIONAL_BEHAVIOR_PATH}"))
        CgiGetCurrentDir(PRV_CUR_DIR)
        CgiAddAdditionalBehaviors(${CGIAPP_ADDITIONAL_BEHAVIOR_PATH} "ControlBehaviors")
    endif()
endif()
#add to WidgetSet.cpp

if (CGIAPP_ENABLE_TUTORIAL)
    source_group(PRV_ALL_SRCS FILES ${PRV_CGIAPP_TUTORIALWIDGET_SUBDIRS})
    CgiCollectListedFiles(PRV_ALL_SRCS "" "" LIST ${PRV_CGIAPP_TUTORIALWIDGET_SUBDIRS})
    CgiFetchHeaderFiles("Widgets_Tutorial" "${PRV_CGIAPP_TUTORIALWIDGET_SUBDIRS}")
endif(CGIAPP_ENABLE_TUTORIAL)
```

```
source_group(PRV_ALL_SRCS FILES ${PRV_CGIAPP_BEHAVIORS_SUBDIRS})
CgiCollectListedFiles(PRV_ALL_SRCS "" "" LIST ${PRV_CGIAPP_BEHAVIORS_SUBDIRS})
#CgiFetchHeaderFiles("Behaviors_Examples" "${PRV_CGIAPP_BEHAVIORS_SUBDIRS}")
CgiFetchBehaviorsHeaderFiles("Behaviors_Examples" "${PRV_CGIAPP_BEHAVIORS_SUBDIRS}
```

In this example, following projects will be created in the solution:

- **BehaviorSet** for compiling WidgetSet.cpp
- **ControlBehaviors_1** for compiling behaviors defined at *<cgi_studio_root>/cgi_studio_controls/src/Behaviors*

Adding Widget Sets

If other locations than in the example above shall be included for widget set generation, following steps are required:

1. Specify the paths to additional widget locations in the file *<cgi-studio-root>/cgi_studio_player/src/Behaviors/AdditionalBehaviorsPaths.txt and/or AdditionalWidgetsPaths.txt.* These paths must be relative to the default widget folder.
2. Each directory must - as always - provide a "FileList.txt" which lists all files to include. All subdirectories of the given directories which contain a "FileList.txt" file will also be included.
3. Setup Visual Studio Solution "Player.sln"
   - Start CMake again
   - Point "Where is the source code" to *<cgi-studio-root>/cmake/Candera/Player*
   - Point "Where to build the binaries" to *<cgi-studio-root>/cgi_studio_player/build*.
   - Enable option "CGIAPP_ENABLE_ADDITIONAL_BEHAVIORS" and/or "CGIAPP_ENABLE_ADDITIONAL_WIDGETS"
   - After pressing configure the new option "CGIAPP_ADDITIONAL_BEHAVIOR_PATH" and/or "CGIAPP_ADDITIONAL_WIDGET_PATH" is shown. Cmake should automatically detect "AdditionalBehaviorsPaths.txt" and/or "AdditionalWidgetsPaths.txt"
   - Click "Configure" again
   - Click "Generate", the solution is generated

The project will contain a sub project for the generated widget set, a sub project for the main widget set, and further sub projects for each additional widget location specified in AdditionalBehaviorsPaths.txt and/or AdditionalWidgetsPaths.txt.

>

: ATTENTION: The last part of a path in AdditionalBehaviorsPaths.txt and AdditionalBehaviorsWidgetsPaths.txt can not be the same.

Fine	**Wrong (last part is Behaviors for both paths)**
cgi_studio_controls/src/Behaviors	cgi_studio_controls/src/Behaviors
CustomBehaviors/src/CustomBehaviors	CustomBehaviors/src/Behaviors

BehaviorSet Library with Behavior Set Parts

With the new CMake environment, a new CMake Flag **CANDERA_DEPRECATED_WIDGETSET_ENABLED** is available (Default setting is **ON**). When this flag is turned **OFF**, then instead of a single file "WidgetSet.cpp", several files are generated for every Widget or Behavior library.

This might resolve known issues with too many symbols generated during compilation of single "WidgetSet.cpp" file.

Candera Version Validation

To ensure, that an application like the Player always refers to the proper Candera version, following CMake configurations can be used:

<CANDERA_DIR>/CgiStudioCanderaConfig.cmake: This file specifies the actual version of Candera, e.g.:

```
set(CGI_CANDERA_VERSION "3.12.0.0")
```

Accordingly, proper Candera build defines will be generated, accessible for applications via *<CANDERA_DIR>/src/Candera/CanderaVersion.h*

<CGI-Player_DIR>/src/CgiStudioPrereqs.txt: This file is an example how an application will request a specific Candera version:

```
set(CGI_CANDERA_VERSION_REQUIRED "3.12.0.0")
```

<CGI-Player_DIR>/cmake/cgiapp_cmake_common.cmake: This cmake file contains an example how an application can detect a Candera folder matching a specific Candera version (via cmake find_package command).

Using Player on 64 bit Systems

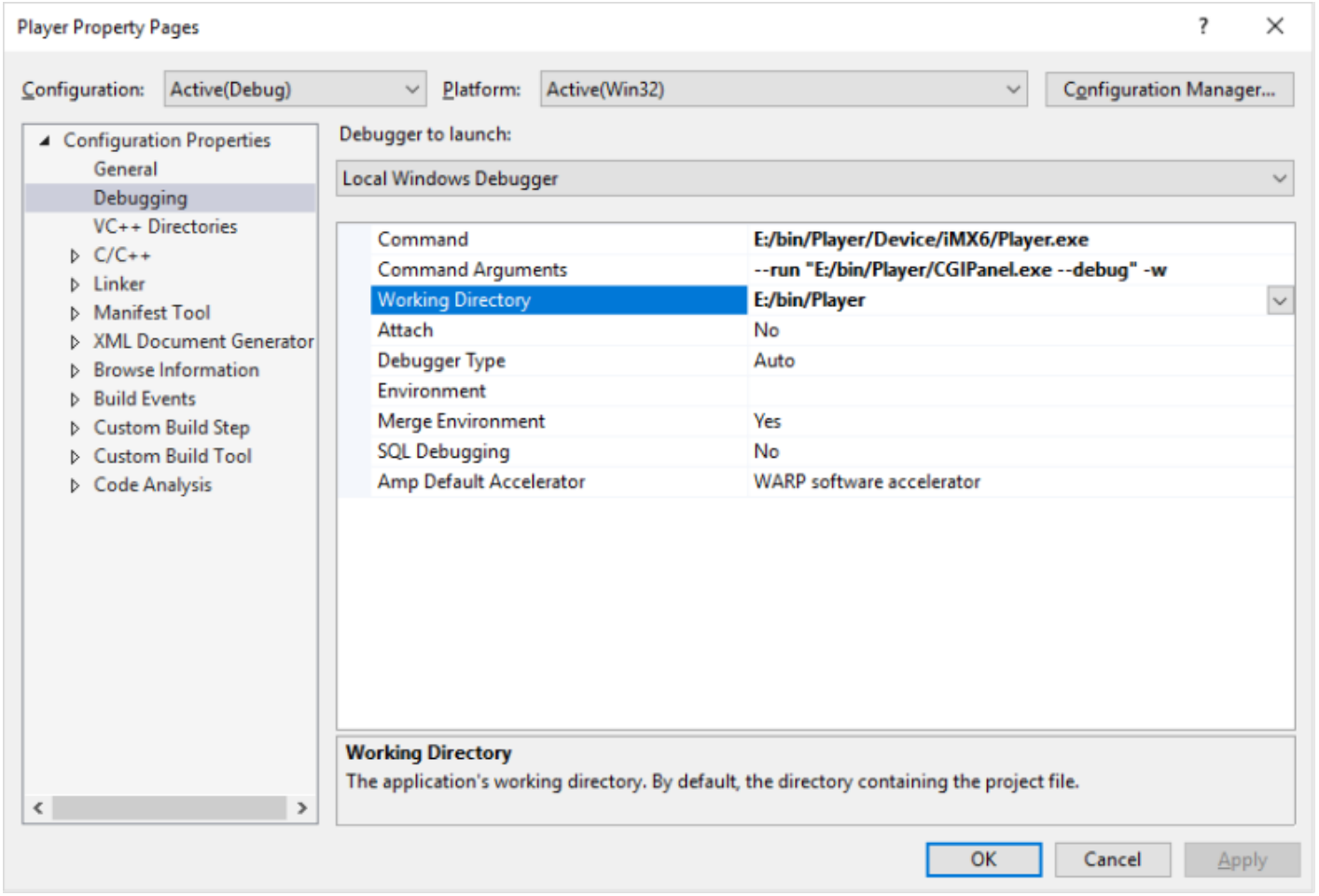
For Player, the CMake project generator should be of type *Win64* (e.g. *Visual Studio 15 2017 Win64*) and it should be built with the "COURIER_PLATFORM_CONFIG" setting set to "x64". This is the only way to build a SCHost.dll that can be used in SceneComposer

Debugging Hints

Debugging Player

For debugging the Player set "Player" as startup project in the Player solution and start debugging.

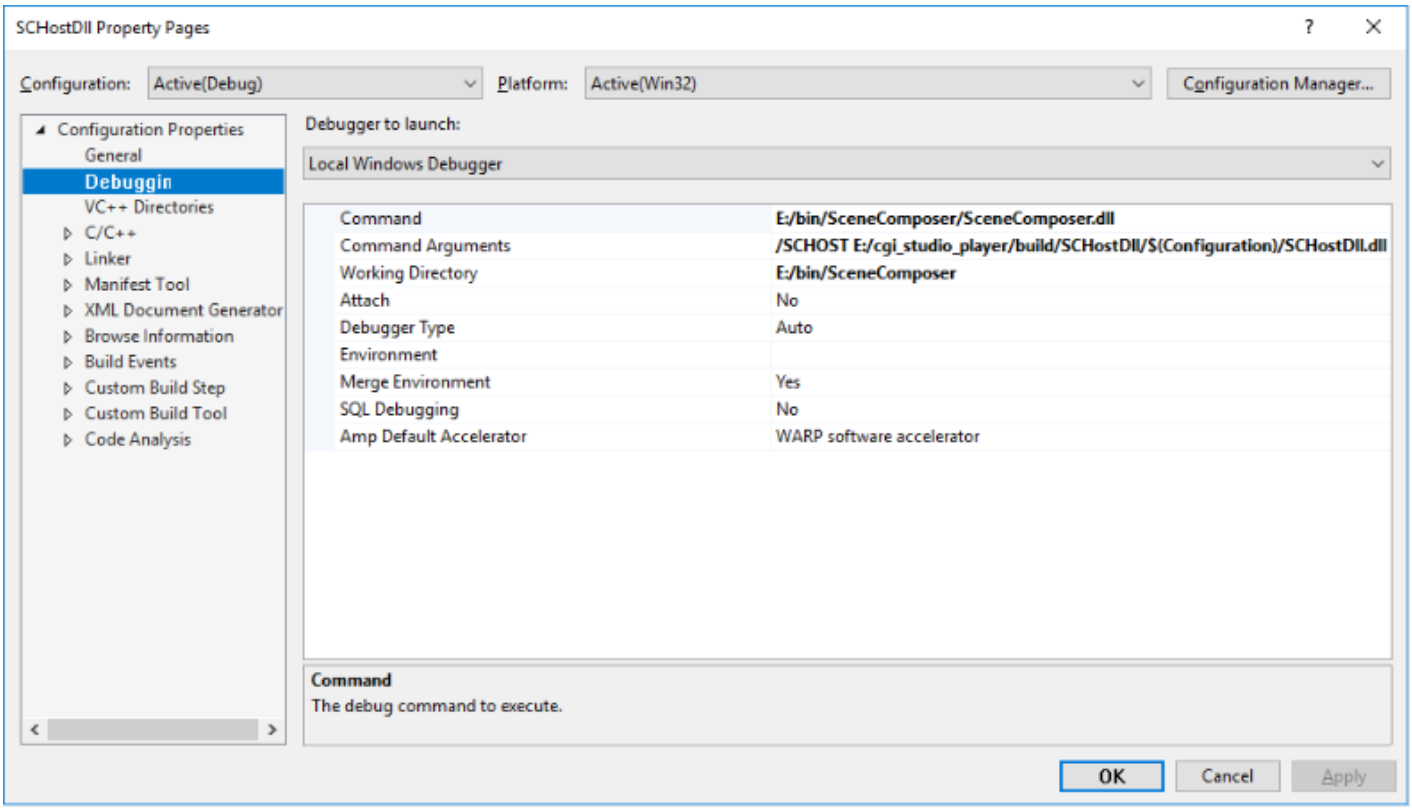
With CMake, the required Debugging configurations are preconfigured as following:



Debugging Widget Life Cycle in SceneComposer

For debugging widgets used in SceneComposer please execute the following steps:

- Set SCHost as startup project in the dev solution and edit the properties as shown below.



Set:

- Debugger Type to Native Only
- Command: The path to the file 'SceneComposer.dll' in your SceneComposer installation.

- After setting a breakpoint in a widget in the dev solution and pressing F5 (Debug), the breakpoint will get hit as soon as the widget is used by SceneComposer.

> Debugging with SceneComposer only works while using a SCHost.dll including debug information! For this, take care that the CMake variable "CGIAPP_SCENE_COMPOSER_PATH" is set to your SceneComposer executable, and compile in debug mode. All dll files required by SceneComposer will then be automatically deployed to your SceneComposer executable directory.

Used Behaviors Plugin

The "Used Behaviors Plugin" extracts the type names of all Behaviors referenced in a solution. At asset generation time, it generates a CMake file as given in the plugin configuration.

When configuring a build with CMake, there is a new option "CONTROLS_PLUGIN_CONFIGURATION_ENABLED".

> This option is hidden, when "COURIER_ADD_SCHOST_PROJECT" flag is set to ON.

When the option "CONTROLS_PLUGIN_CONFIGURATION_ENABLED" is turned on, then another option "CONTROLS_USED_BEHAVIORS_CONFIG_PATH" is exposed, which must be set to the file as configured in the plugin.

Then the build is configured by CMake to only contain the Behaviors referenced in the solution, and some base behaviors and functionality, which is always required.