Target setup for iMX6 with QNX OS

Description
This document describes the setup required to build and run CGI Studio applications with iMX6 device package on Freescale iMX6 hardware, as well as on Linux PCs.

Instructions are provided how to setup the system.
On iMX6 hardware this means how to update with the required BSP and Graphics Driver.
On Linux PCs this means how to install needed libs, headers and graphics drivers.
Next, it is explained how to run the iMX6 target demo application part of the release package.
Finally, the document provides instructions how to setup the build environment and how to build own applications for iMX6 target (Freescale iMX6 or Linux PC).

Shader Compiler Plugin
In SceneComposer there is a Shader Compiler Plugin available. Please refer to iMX6 Shader Compiler for more information.

- iMX6 QNX
  - iMX6 QNX ARM: Prerequisites
  - Running the Player Demo Application on iMX6 QNX
  - Building Own Application for iMX6 QNX
- iMX6 Prerequisites
- iMX6 Various Device Specific Content
  - iMX6 Specific Best Practices
  - Candera Device Packages
  - FeatStd Overview
  - Compile Shaders
  - Render Targets

iMX6 QNX

iMX6 QNX ARM: Prerequisites

Hardware

CGI Studio has been tested on Freescale i.MX6Q Sabre Board for Smart Devices

QNX Board Support Package (BSP)

QNX Neutrino for: BSP_freescale-imx6q-sabresmart_br-660_be-660 SVN Revision Number: 773534 Build Number: 313

For documentation related to this BSP please refer to:

http://community.qnx.com/sf/wiki/do/viewPage/projects.bsp/wiki/BSPAndDrivers

CMake

In order to generate makefiles for building Candera applications, the Meta build system CMake needs to be installed with at least version 3.5.2.

Running the Player Demo Application on iMX6 QNX

iMX6 System Preparation

For documentation related to how to build the BSP and prepare the bootable SDCard, please refer to the BSP User Guide:

http://community.qnx.com/sf/frs/do/viewRelease/projects.bsp/frs.freescale_i_mx6q_sabre_board_for.sdp_6_6_bsp_user_guide_for_sabre

Executing Target Player from the SD Card

The application and asset can be copied into the folder /usr/bin on the SD-Card with the target root file system, and executed like this:

```
Welcome to QNX Neutrino Initial Program Loader for Freescale i.MX6Q Sabre-Smart (ARM Cortex-A9 M
SDMMC download...
load image done.
Found image               @ 0x18000008
Jumping to startup        @ 0x10807364
ÿWelcome to QNX Neutrino 6.6.0 on the i.mx6 Smart-Device (ARM Cortex-A9 MPCore)
Starting watchdog...

Starting SD3 memory card driver...
Starting Ethernet driver
Starting screen
Running startup script...
# Mempool Map addr range[28100000-37f00000]
Mempool Map paddr range[3fd00000-4fb00000]
[Interrupt] Attached irqLine 41 with id 5.
[Interrupt] Attached irqLine 42 with id 6.
[Interrupt] Attached irqLine 43 with id 7.
Attached resmgr to /dev/galcore with id:1.

# cd /usr/bin
# CGIApplication_ARM_QNX_iMX6_Target_GCC iMX6_GettingStarted_3D_GLES20.bin

Starting standard application ...
```

Building Own Application for iMX6 QNX

Interfaces

The Feat-Standard public interface is covered by the namespaces FeatStd and various subnamespaces of namespace FeatStd. All namespaces containing either **Internal** or **Private** in the name cover private namespaces, which are not recommended to be used in any application, as it is not guaranteed that these interfaces remain stable respectively exist in any future release. Changes in any of these private namespaces can be applied without any (prior) notifications.

Namespace FeatStd

- Container
- Utilities

Namespace FeatStd::Diagnostics

- Diagnostics

Namespace FeatStd::MemoryManagement

- Memory Management

Namespace FeatStd::PerfMon

- MonitorPerformanceReporting

Namespace FeatStd::Platform

This interface defines the access to platform dependent objects. The implementation of this platform dependent code is not part of FeatStd, though there are a few reference platforms, like SimulationPlatform (implemented basically for Win32) or GenericPlatform (providing generic

respectively stub implementations), which can used as templates for the customer.

This platform interface can be seen as an official porting interface of FeatStd to any customer specific platform setup, which is the reason for outlining it in the separate namespace FeatStd::Platform.

- Platform

iMX6 Prerequisites

vCompiler

The default shader compiler for iMX6, vCompiler.exe, is part of the VTK (Vivante Tool Kit) provided by Vivante Corporation. The VTK version needs to match exactly the iMX6 hardware version. For further information please check the NXP Semiconductors iMX6 documentation for graphic APIs and driver support information.

After installing vCompiler, please verify that its location has been added to the PATH environment variable. Add manually if necessary.

iMX6 Various Device Specific Content

On this page there is various content specific to the iMX6 Device.

iMX6 Specific Best Practices

Description

CGI Studio supports 2D and 3D content creation, composition, and rendering. For iMX6 the Candera graphics engine renders both 2D and 3D content via OpenGL ES API.

This chapter includes design and programming guidelines. It is strongly recommended to follow those hints to utilize *iMX6* in a performance-optimized manner.

Candera Graphics Engine supports both OpenGL ES 2.0 and OpenGL ES 3.0. Selected performance features of OpenGL ES 3.0 are available in OpenGL ES 2.0 as extensions.

Those OpenGL ES 2.0 extensions that are supported by iMX6 and Candera are listed below:

- Binary shader programs: Reduces start up times significantly.
- Direct texture: Vivante extension to access external image and video data.
- Texture compression: Latest ETC2 compression minimizes required memory bandwidth and storage for image data.
- Anisotropic filtering: Increased texture quality for steep angles of view.
- Broad range of buffer formats: Depth 24/32, Stencil 1/4/8, etc.
- 24-bit vertex element index: Support for higher polygon count in 3D models.
- Blending equations minimum and maximum: Enables color clamping.
- Synchronization: Fences support CPU / GPU synchronization.
- And many more.

> Many of those extensions have been introduced in OpenGL ES 3.0 API.

Multi Frequency Rendering

Multi frequency rendering requires either multiple buffered hardware layers (display planes) or software layers. While one layer is rendered and swapped with a higher frequency (e.g. swapping each frame results in 60 FPS), another layer is drawn partially and not swapped unless rendering is complete (e.g. swapping each second frame results in 30 FPS).

When software-layers are used, different update rates of the software-layers can be achieved through multi frequency rendering. It should be considered that the software layers still need to be drawn and blended in every frame and will consume memory bandwidth even when rendering to

the software-layer is done at a lower rate.

Special Alpha Blending Techniques

MSAA Blending

The iMX6 platform supports multi-sampling anti-aliasing (MSAA) to avoid jagged edges. MSAA renders multiple samples per pixel and combines them to a final color. This circumstance can be used to render transparent objects in a very performant manner, too. Instead of alpha blending via blend-mode, MSAA can be used to render only a selected percentage of samples per pixel. With that approach blend operations and read/write operations, which are typically required for blending, can be omitted.

Example:
MSAA is defined to accommodate 4 samples per pixel. The alpha value can define the coverage of samples to be written into frame buffer. Coverage of 50% means that half the samples are written, resulting in 2 of 4 samples being rendered. When the samples are finally combined to one pixel, the rendered object contributes only with 2 of 4 samples, which results in 50 % transparency.

Layer Blending

When blending multiple layers a correct alpha value must be written to the layer. By default objects overwrite the current alpha value of a pixel in the layer. In case of layer blending the alpha value of a pixel must be correctly combined with the existing alpha value.

Default alpha blend setting (optimized for performance):

- Source Blend Factor: One
- Destination Blend Factor: Zero

Alpha blend setting for blending multiple layers:

- Source Blend Factor: Inverse Destination Alpha
- Destination Blend Factor: One

Shader

Shader Precision

It is *strongly* recommended to use precision "*high*" in iMX6 shaders.

As medium and low precisions are converted to 32 bit internally, work load for conversion is

required, while only saving little amount of memory compared to textures.

Utilization

One strength of iMX6 is the fast shader computation unit. Utilize it best, by computing color values instead of texture fetches. Take care of pixel throughput and bandwidth utilization. Reduction of pixel coverage in frame-buffer is of utter importance.

Example:
Do not use full-screen backgrounds or big textures. In 2D reduce coverage by tessellated, smaller shaped images or in 3D even better by right-shaped geometry, with least pixel coverage.

Binary Shaders

Candera supports both binary shaders (pre-compiled) and binary shader programs (pre-compiled and linked) on the iMX6 platform. Use them to reduce start-up time significantly.

Textures

Procedural Textures

For iMX6 it is advised to use procedural textures whenever possible in order to both, overcome limited bandwidth and to utilize fast shader units.

Example:
Instead of storing a color gradient in a texture, calculate color in shader related to the texture coordinate given.

Texture Compression

Use **ETC2** texture compression on iMX6. This measure saves bandwidth and memory consumed.

> Textures with color gradients shall not be compressed, due to noticeable decrease in visual appearance.

Consider Tile Size

The iMX6 platform internally uses a tile-based rendering approach. Make sure to align render-area with the optimal tile size of 16x8 pixels. Small intersections of adjacent tiles shall be avoided. Partial coverage of tiles results in wasted performance.

Scene Partitioning

Recommended Partitioning of a 2D Scene

Attempt to reduce dimension of rectangles by omitting areas that are invisible (e.g. same color as background) or occluded (another opaque image is rendered on top of it). This spares pixel coverage.
Furthermore, only those regions need to be redrawn that are not covered by moving needle.
Example: A classical, round tachometer with a needle shall be rendered. The tachometer basically consists of a tachometer image and a needle image. Except of the tachometer's dial most pixels within tachometer's image have same color as background. Tip: Scissor the tachometer into multiple smaller rectangles to avoid rendering of a huge rectangular image, which renders big portion of same color on top of background. This measure spares pixel coverage. Furthermore, only those clipped tachometer image rectangles must be redrawn that are not covered by moving needle.
Advantages:

- Less pixel coverage.
- Reduced rendering updates.

Recommended Partitioning of a 3D Scene

Use materials instead of textures, no image data required.
Material can be defined directly in shader to reduce number of shader parameters.
Geometry allows occluding exactly those areas visible; no wasted pixels are rendered into framebuffer.
To mitigate aliasing effects (jagged edges), fade out silhouette with transparency.
Advantages:

- Utilize computation strength of iMX6 (shader units), avoid bandwidth utilization (read/write of image data), and spare memory allocation (image data).
- Ease of transitions.
- Advanced visual effects are possible.

Layer Configuration

The iMX6 hardware and layer capabilities are reflected in SceneComposer configuration.

Hardware Layers via Planes:
One IPU (Image Processing Unit) blends two planes efficiently with Display Processor (DP).
Restrictions of the planes:

- Background plane must be full-screen
- Foreground (Overlay) plane can have any size and position

Split of content in two layers allows rendering in multiple frequencies. See chapter Multi Frequency Rendering for more details.

Software Layers:
Software Layers are supported via off-screen surfaces. Up to eight software layers can be blended in one single pass. In successive blend-passes even more software layers are possible. The use of software layers is fully integrated in SceneComposer.

> Off-screen surfaces do not support anti-aliasing in OpenGL ES 2.0.

Arbitrary Warping:
Full support for arbitrary warping as required for Head-up display.
CGI Studio supports arbitrary warping for Head-up displays on iMX6. Warping is internally realized via morphed warping mesh.

Candera Device Packages

Supported Device Packages

Candera provides device implementations for following device types:

Name	Description	Possible values
Freescale iMX6		ON/OFF
Other proprietary devices	Can be provided upon special request	ON/OFF

Device-Specific CMake Settings

Name	Description	Possible values
CGIDEVICE_NAME	The name of the device package	Each device package following the Candera device CMake rules provided in the folder <CANDERA_SRC>/CanderaPlatform/Device
CGIDEVICE_TARGET_BUILD	Enable or disable building for target	ON/OFF
CGISTUDIO_PATH_3RD_PARTY	Path to the 3rd party repository location	A string value containing the system file location of 3rd party software, auto-detected by finding possible locations of the name "[ic01_]cgi_studio_3psw"
CGIDEVICE_TARGET_ <CGIDEVICE_NAME> _DRIVER_PATH_INCLUDE	If you don't use the default driver path, the path to the driver include directory can be changed with this setting.	Path to driver include directory (by default set to include directory in 3rd party repository)
CGIDEVICE_TARGET_ <CGIDEVICE_NAME> _DRIVER_PATH_LIB	If you don't use the default driver path, the path to the driver library directory can be changed with this setting.	Path to driver library directory (by default set to library directory in 3rd party repository)

To build a certain device package selected it from the drop-down list of the CMake property **"CGIDEVICE_NAME"**:

For each device package, either building OpenGL based host simulation or target binaries can be selected via the Boolean CMake property **"*CGIDEVICE_TARGET_BUILD*"**.

The following table illustrates configurations available for the supported device package:

CGIDEVICE_NAME	CGIDEVICE_TARGET_BUILD	PUB_CGI_CPU	PUB_CGI_OS	Default driver path following <CGISTUDIO_PATH_3RD_PARTY> /[src/lib]/
iMX6	Off	X86	Win32	OpenGLES/Win32/OpenGLAdapter
iMX6	On	ARM	Linux	OpenGLES/Linux/iMX6

The CMake properties "PUB_CGI_CPU", "PUB_CGI_OS" usually cannot be configured via the CMake user interface, but are autonomously detected by
the build system. The same applies for the variable "PUB_CGI_COMPILER".

If you don't use the CGI Studio 3rd party software folder structure, the driver paths can be specified in the variables

- CGIDEVICE_TARGET_<CGIDEVICE_NAME>_DRIVER_PATH_INCLUDE and
- CGIDEVICE_TARGET_<CGIDEVICE_NAME >_DRIVER_PATH_LIB

Depending on these properties, either MS Visual Studio solutions or Unix makefiles will be generated into the specified build
directory.

FeatStd Overview

Introduction

For general information and an overview of FeatStd please refer to FeatStd Overview.

See here a typical output of a CMake configuration run of FeatStd:

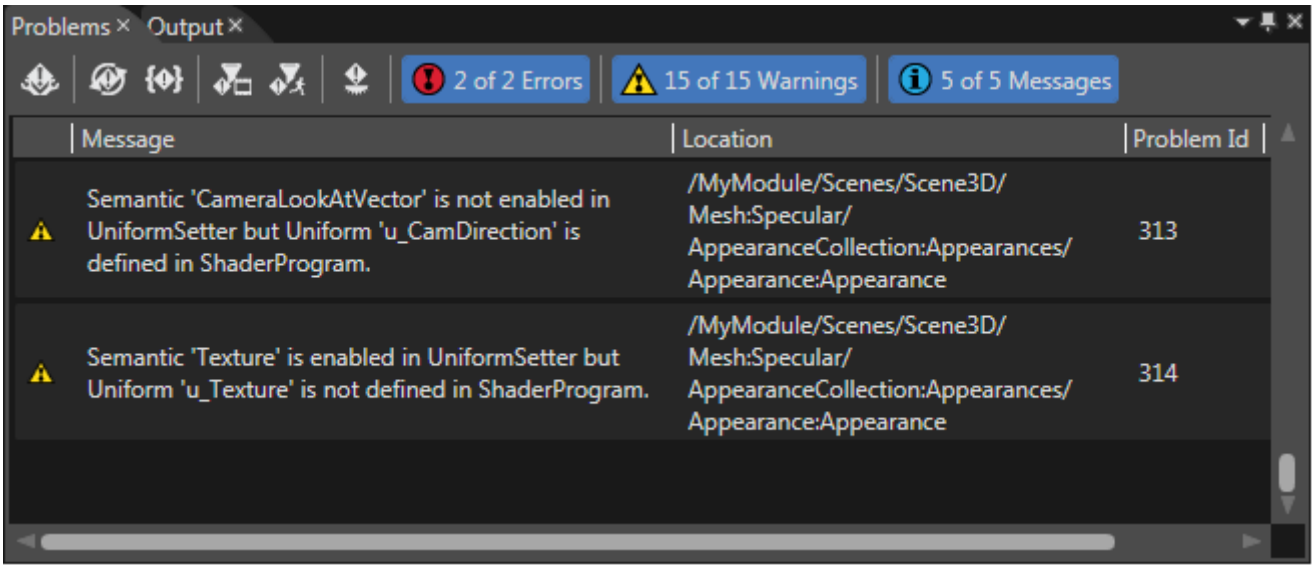
```
Building project: FeatStd vW.X.Y.Z
---------------------------------
Pre-sets from calling project FeatStd:
    Pre-set include-dirs:
    Pre-set definitions:
---------------------------------
FeatStd external configuration CMake variables:
    BUILD_SHARED_LIBS: OFF
    FEATSTD_3PSW_DIR: D:/cgi-studio/cgi_studio_3psw
    FEATSTD_ADD_BUILD_MODE_DEFINE:
    FEATSTD_USE_PREBUILT_LIBS: OFF
    FEATSTD_PREBUILT_LIB_DIR:
---------------------------------
FeatStd feature settings:
    FEATSTD_IPC_ENABLED=OFF
    FEATSTD_LOG_ENABLED=ON
    FEATSTD_MONITOR_ENABLED=OFF
    FEATSTD_THREADSAFETY_ENABLED=OFF
    FEATSTD_UNITTEST_FRAMEWORK_ENABLED=OFF
    FEATSTD_FRACTIONAL_NUMBER_TYPE=Float
    FEATSTD_MEMORYPOOL_ENABLED=OFF
    FEATSTD_SYSTEM_MEMORY_STATISTIC_ENABLED=OFF
    FEATSTD_SYSTEM_MEMORY_STATISTIC_FILE_AND_LINE_TRACKING_ENABLED=ON
---------------------------------
FeatStd compiler defines:
    ;_CRT_SECURE_NO_WARNINGS;FEATSTD__FILE__=__FILE__
---------------------------------
FeatStd configuration file:
    Generated to D:/cgi-studio/build/gensrc/FeatStd/Config.h
---------------------------------
Exported FeatStd CMake variables:
    FEATSTD_EXPORTED_CMAKE_DIR: D:/cgi-studio/featstd/cmake
    FEATSTD_EXPORTED_DEFINITIONS: -D_CRT_SECURE_NO_WARNINGS;-DFEATSTD__FILE__=__FILE__
    FEATSTD_EXPORTED_INCLUDE_DIRECTORIES: D:/cgi-studio/featstd/src;D:/cgi-studio/build/gensrc
    FEATSTD_EXPORTED_LIB_NAME: FeatStd_x86_Win32_iMX6_Simulation_MSVC
    FEATSTD_EXPORTED_LINK_DIRECTORIES:
    FEATSTD_EXPORTED_LINK_LIBRARIES: winmm;Ws2_32;FeatStd_x86_Win32_iMX6_Simulation_MSVC
    FEATSTD_EXPORTED_OS: Win32
    FEATSTD_EXPORTED_ROOT_DIR: D:/cgi-studio/featstd
```

Compile Shaders

Shader compilation is triggered in the following situations:

1. During shader editing, if requested via menu item "Shader" > "Compile Shader" or by pressing F7 key, the selected shader or shader program will be compiled using the platform's shader compiler.
2. During scene editing, when the scene dependencies are generated, the shaders will be compiled using an internal shader compiler (so called "builtin://DefaultShaderCompiler").
3. During asset generation, depending on the selected shader compiler type, the shaders will be compiled using platform's shader compiler (Target) or internal shader compiler (Simulation).

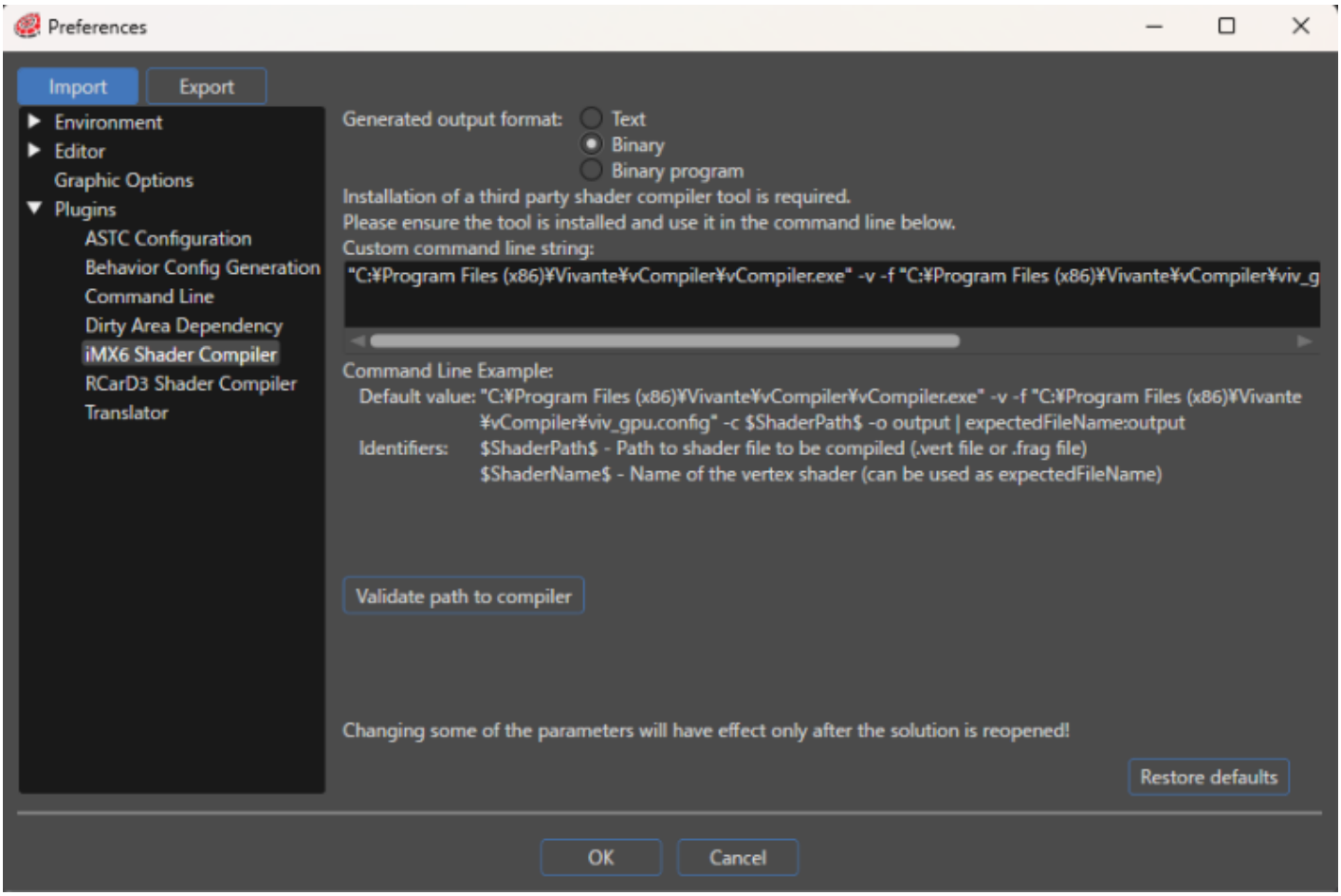
The compilation result can be seen in the "Problems Browser" panel. For further information please refer to section Problems Browser.



After successful shader compilation, a regarding message is displayed.

iMX6 Shader Compiler

iMX6ShaderCompiler provides support for exporting compiled (and/or linked) shaders to the asset file.



It can be configured from File/Preferences section and provides some parameters which can be changed in order to achieve the desired behavior:

1. Output type:

- a. Binary - exports compiled shaders (separate compiled vertex and fragment shaders)
- b. Binary program - exports a linked shader pair (in the asset file, the vertex shader will contain the compiled and linked pair and the fragment shader will contain an empty buffer)
- c. Text - exports the source code of the shaders (separate source code for vertex and fragment shaders)

2. Command line - a configurable string of the following format:

```
<path_to_executable_file> [arg0] [arg1] [arg2] ... [argN] | expectedFileName:<output_file_name>
```

- a. The path to executable to be run in order to generate the compiled/linked shaders along. The executable itself can be specified alone (no path) if the path to it's parent directory is found in the system environment path. Paths which cointain spaces need to be delimited by double quotes. Example: "C:\My Projects\Vivante\vCompiler.exe"

> If vCompiler.exe is not available, in order to avoid an error related to the executable file missing, for host assets the output format for the generated shaders can be set to text. For more information related to vCompiler please

> check vCompiler

- b. The arguments (parameters to be passed to execution). Some special parameters which identify the current processed shader can be specified (they are detailed below). Examples: $ShaderPath$, $VsPath$
- c. The expected file name specifier: specifies the name (without extension) of the output file and is separated from the rest of the command (a and b) by a pipe '|'. iMX6 shader compiler expects output files with the following extensions: i.) .vgcSL - compiled vertex shader (applies for Binary output only) ii.) .pgcSL - compiled fragment shader (applies for Binary output only) iii.) .gcPGM - compiled and linked shader pair (applies for Binary program output only)

Example: For binary output, "expectedFileName:output" - requires that a compiled vertex shader file will be named output.vgcSL and a compiled fragment shader file will be named output.pgcSL

The command line for "Binary" output is a little bit different than the one for "Binary program" output.

Binary output command line: this command line is invoked separately for each vertex and fragments shader.

```
Example: "C:\Vivante\vCompiler.exe" -s4 -c $ShaderPath$ -o output | expectedFileName:output
```

Identifiers

- $ShaderPath$ - can be used as argument only and this identifier will be replaced with actual path to shader file when the command is invoked
- $ShaderName$ - can be used as expectedFileName specifier and will be replaced with the shader name (without extension) when the command is invoked

Binary program output command line:

This command is invoked for a shader pair (both vertex and fragment shader)

```
Example: "C:\Vivante\vCompiler.exe" -v $VsPath$ $FsPath$ -o output | expectedFileName:output
```

Identifiers

- $VsPath$ - can be used as argument only and will be replaced with actual path to vertex shader file when the command is invoked
- $FsPath$ - can be used as argument only and will be replaced with actual path to fragment shader file when the command is invoked
- $VsName$ - can be used as expectedFileName specifier and will be replaced with the vertex shader name (without extension) when the command is invoked

Render Targets

Render Targets Device Capabilities

Device Capabilities

Following is an overview about what kind of render targets are supported on which device:

Device	Number of Displays Supported	Types of Layers Supported	Unit Types
iMX6	4	1 color layer	Mixed2D3DDisplayTarget: Candera::iMX6WindowSurfaceRenderTarget Mixed2D3DOffscreenTarget: Candera::iMX6FrameBufferRenderTarget Candera::SoftwareLayer