CodeXL Plug-In Development Guide

Gilad Yarnitzky, Doron Ofek

Jul-29, 2014

//devtools/main/CodeXL/Documents/CodeXL PlugIn Development Guide.docx

Contents

[CodeXL Plug-In Development Guide 1](#_Toc395194549)

[1 About CodeXL Development 2](#_Toc395194550)

[1.1 CodeXL Standalone Application and Visual Studio Extension 2](#_Toc395194551)

[1.2 CodeXL Plug-Ins and CodeXL Framework 2](#_Toc395194552)

[1.3 CodeXL and 3rd Party Libraries 2](#_Toc395194553)

[1.4 Perforce Repository 2](#_Toc395194554)

[2 Creating the Plug-In Project 3](#_Toc395194555)

[2.1 Create a new Visual Studio Project 3](#_Toc395194556)

[2.2 Set Project Settings 3](#_Toc395194557)

[3 Create Plug-In skeleton 5](#_Toc395194558)

[3.1 Header File for Export definitions 5](#_Toc395194559)

[3.2 Export the 3 Mandatory C Functions 5](#_Toc395194560)

[4 Platform Agnostic Development 6](#_Toc395194561)

[5 Extending CodeXL 6](#_Toc395194562)

[5.1 Extending the CodeXL Menu 6](#_Toc395194563)

[5.2 Extending the CodeXL Global Options 6](#_Toc395194564)

[5.3 Extending the CodeXL Project Settings 6](#_Toc395194565)

[5.4 Handling Events 7](#_Toc395194566)

[5.5 Adding Views 7](#_Toc395194567)

[5.6 Linux build with SCons 7](#_Toc395194568)

# About CodeXL Development

## CodeXL Standalone Application and Visual Studio Extension

CodeXL can be run as a:

* Standalone application  
  Available on Windows and Linux.
* Visual Studio Extension  
  Available on Windows for VS2010, VS2012 and VS2013.

The standalone application and the Visual Studio extension both provide the same functionality and load the same CodeXL plug-ins.

## CodeXL Plug-Ins and CodeXL Framework

CodeXL supports expansion of its functionality via Plug-Ins. A Plug-In is a DLL (or SO on Linux) that can be loaded by the CodeXL application and interacts with CodeXL Framework using the framework classes API.

CodeXL Framework is a collection of classes. Types of framework classes:

* **Interfaces**  
  Plug-In classes inherit interface classes and implement their methods.
* **Base Classes**  
  Plug-In classes inherit base classes, use and extend their functionality.
* **Managers**  
  Plug-In classes register themselves with framework managers. This causes the framework to activate the Plug-In classes when events occur.
* **Utils**Plug-In instanciate and use Utils classes.

## CodeXL and 3rd Party Libraries

CodeXL makes use of the following 3rd party libraries:

|  |  |  |  |
| --- | --- | --- | --- |
| **Library** | **Version** | **Usage** | **License** |
| Qt | 5.3 | UI framework. | LGPL |
| TinyXML | 2.6.2 | XML Parser, read/write XML files. | zLib |
| QSCintilla | 2.8 | Text editing UI control. | GPL + exceptions |
| zLib | 1.2.8 | Compression library. | zLib |
| QCustomPlot | 1.2.1 | Render graphs and charts. | Commercial |

## Perforce Repository

Developer Tools source files are stored in this Perforce repository:

atlvp4s01.amd.com, port 1660

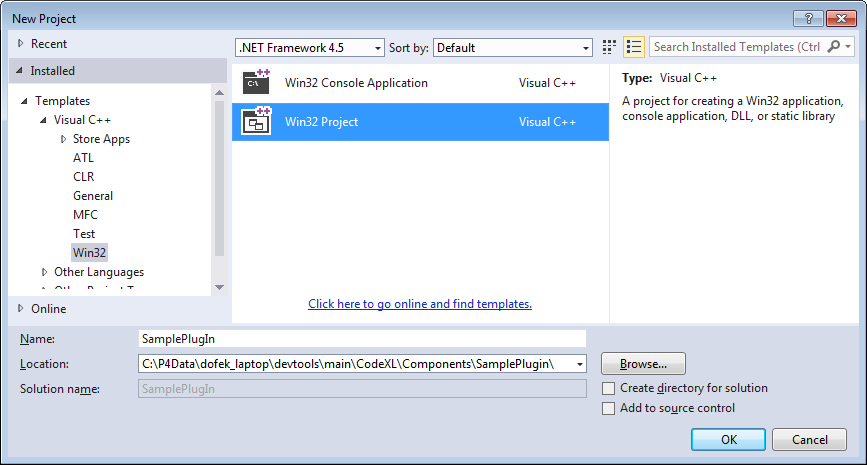
# Creating the Plug-In Project

## Create a new Visual Studio Project

Use Visual Studio 2013.

Create a new project of type Win32.

If you’re working in the DevTools Perforce repository, the path for the project should be in a new folder under the \devtools\main\CodeXL\Components folder, such as “\devtools\main\CodeXL\Components\SamplePlugIn”.



In the Project’s Application Settings, select

* Application Type = DLL
* Additional Options
  + Check ‘Empty Project’
  + Uncheck ‘SDL’

## Set Project Settings

Create a new props file for the Plug-In project. The props file should contain the following:

* Define the CommonDir macro  
  This is the path to the \devtools\main\Common folder, starting with the current project directory, i.e. $(ProjectDir)\*path\_to\_common\_folder*
* Import the CodeXLComponent props file

See the sample props file below. The areas that require editing are highlighted:

<?xml version="1.0" encoding="utf-8"?>

<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<PropertyGroup Label="UserMacros">

<CommonDir>$(ProjectDir)\..\..\..\Common</CommonDir>

</PropertyGroup>

<ItemGroup>

<BuildMacro Include="CommonDir">

<Value>$(CommonDir)</Value>

</BuildMacro>

</ItemGroup>

<ImportGroup Label="PropertySheets">

<Import Project="$(ProjectDir)\..\CodeXLComponent.props" />

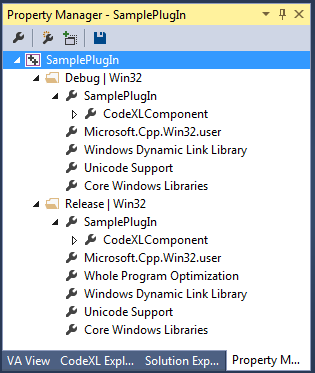
</ImportGroup>

</Project>

Place this file in the same folder as the project file. For example: CodeXL\Components\SamplePlugIn\SamplePlugIn.props

Add the props file to the project using Right-click -> “Add Existing Property Sheet…”

Your project’s Property Manager tree should look like this:



# Create Plug-In skeleton

## Header File for Export definitions

Create a new header file (e.g. SamplePlugInExport.h) for defining dll export macros. Place the file in the same folder as the project file, and add the file to the project.

See the following example:

//=============================================================

// (c) 2014 Advanced Micro Devices, Inc.

//

/// \author Doron Ofek

/// \version $Revision: $

/// \brief Sample Plug-In export macros

//

//=============================================================

// $Id: $

// Last checkin: $DateTime: $

// Last edited by: $Author: $

// Change list: $Change: $

//=============================================================

#ifndef \_\_SAMPLE\_PLUGIN\_EXPORT\_H\_

#define \_\_SAMPLE\_PLUGIN\_EXPORT\_H\_

// Under Win32 builds - define: SAMPLE\_PLUGIN\_API to be:

// - When building SamplePlugIn.dll: \_\_declspec(dllexport).

// - When building other projects: \_\_declspec(dllimport).

#if defined(\_WIN32)

# if defined(AMDTSAMPLEPLUGIN\_EXPORTS)

# define SAMPLE\_PLUGIN\_API \_\_declspec(dllexport)

# else

# define SAMPLE\_PLUGIN\_API \_\_declspec(dllimport)

# endif

#else

# define SAMPLE\_PLUGIN\_API

#endif

#endif // \_\_SAMPLE\_PLUGIN\_EXPORT\_H\_

## Export the 3 Mandatory C Functions

Each CodeXL plug-in must export 3 C functions, as follows

* **CheckValidity**  
  Signature: int CheckValidity(gtString& errString)  
  Purpose: Verify that the plug-in’s prerequisites are met. This is the first function that is called.  
  Description: This function checks if the preconditions that the plug-in relies on are met. For example, plug-ins that require OpenCL should verify that OpenCL is installed.  
  This function should return zero if successful. Otherwise an error code should be returned and a description of the error should be assigned to the errString parameter.
* **Initialize**  
  Signature: void initialize()  
  Purpose: Create objects that interact with the CodeXL framework manager objects.  
  Description: Typical initializations that occur in this function:
  + Create and register the plug-in's specific main menu actions creator
  + Initialize backend modules specific to this plug-in
  + Create and register the plug-in's specific node in the global settings page
  + Create and register the plug-in's specific node in the project settings page
  + Create and register the plug-in's specific execution mode
  + Create event observers and register them with framework managers
  + Create, init and register the plug-in's specific views creator
* **InitializeIndependentWidgets**  
  Signature: void initializeIndependentWidgets()  
  Purpose: Initialize other items after main window creation.  
  Description: This function initializes all the widget items that are not registered with the creators mechanism. These widgets are responsible for their own callbacks and strings.

# Platform Agnostic Development

CodeXL plug-ins should be platform agnostic, just like all other CodeXL components. For this purpose we use the following OS abstraction modules:

* BaseTools - contains macros, STL-like containers and the gtString class.
* OSWrappers - contains a set of classes for File System access, Inter-Process Communication, Threading and Synchronization, and many others.

# Extending CodeXL

## Extending the CodeXL Menu

TODO: add content to this section

## Extending the CodeXL Global Options

TODO: add content to this section

## Extending the CodeXL Project Settings

TODO: add content to this section

## Handling Events

TODO: add content to this section

## Adding Views

TODO: add content to this section

## Linux build with SCons

TODO: add content to this section

CodeXL Linux build uses gcc 4.7.2 and Scons on a CentOS 6.2 machine.

SCons is similar to make but is python based.