![Texas Instruments]

XgxPerf Toolkit

Frangline Jose

ABSTRACT

This application report shows how to use the XgxPerf toolkit to build and benchmark a Qt™ based application. It is assumed that you are familiar with installing and writing applications based on Qt.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www-s.ti.com/sc/techlit/sprabf4.zip.

1   Introduction

Section 5 includes information on getting started on Qt and the links to download XgxPerf.

A sample application is also provided with this document.

2   XgxPerf

XgxPerf is a light-weight toolkit that you can use to build modular Qt applications that can be profiled. XgxPerf has two main components: the *Application Manager* and the *Application Base*.

The main uses of this toolkit are to profile, benchmark, and provide an example for optimized usage of Qt. It also adds tools like *livemem* used to capture screens at any point in time when the application is being executed. Optimized SVG classes for animation are provided.

XgxPerf is dependant on SVG and Webkit; Qt needs to be built with these options.
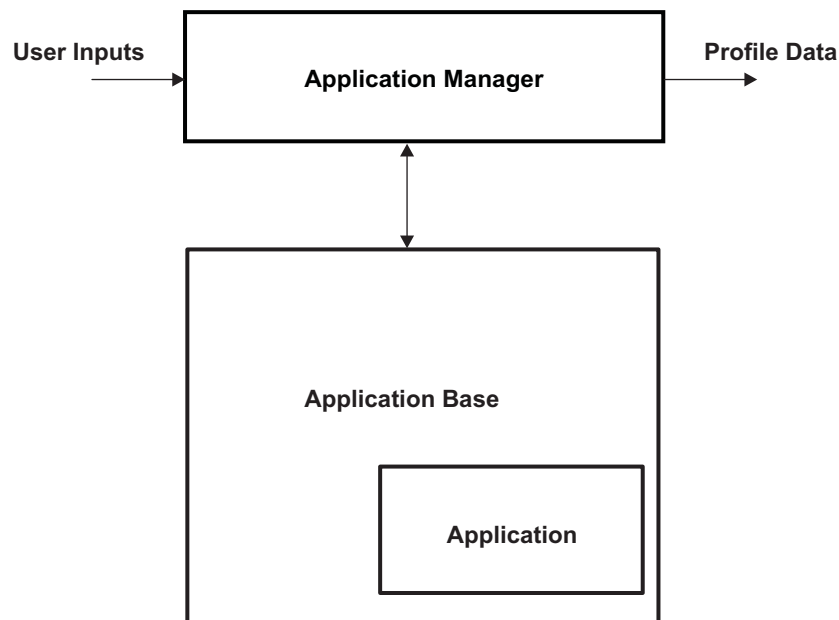

Figure 1. High-Level View of XgxPerf

2.1 Application Manager

Application manager acts as an interface between you and the actual Qt-based application. It has the option to load the actual Qt application and also to accept the relevant parameters via command line.

2.2 Application Base

Application base is a class used to profile the application's CPU bandwidth requirements. This class can be used in providing the startup time of the application, as well as the animation time.

`applicationmanager\common\applicationbase.h` provides the application base class. The base class provides Init, Process, GetAppName, GetAppIcon, GetSwitchingTime and calls.

- Init: initialize the widgets
- Process: timer-based process calls
  The application manager configures the timer based on the fps set.
- GetAppName: returns the application name (deprecated)
- GetAppIcon: returns the icon for application (deprecated)
- GetSwitchingTime: returns the time interval for whichever process is called (deprecated)

3 How to Create an Application With XgxPerf

3.1 XgxPerf Package

The toolkit comes with an application manager package with the following components:

`<INSTALL_PATH>\xgxperf\trunk\Packages\OMAP3_Graphics_SDK\xgxperf\xgxperf\application manager`

```
--common
      --applicationbase.h
--qt_e
      --applicationmanager.cpp
      --applicationmanager.h
      --main.cpp
      --qt_e.pro
```

3.2 Creating a New Application

The application is built as a library. The application manager loads the application library into the target and waits for the execution to complete.

To create a new application:

1. Create a new cpp file say helloworld.cpp,
   create a new class from applicationbase.

```
class HelloWrld: public ApplicationBase
{
public:
    HelloWrld(QWidget* parent, QRectF inSceneRect, QGraphicsScene *scene, void* data);
    void Init();
    void Process();
    QString GetAppName();
    QIcon GetAppIcon();
    int GetSwitchingTime();
private:
    TestTargetParamsStruct* m_widgetTestInStruct;
    QGraphicsTextItem *pHelloVal;
};
extern "C"
HelloWrld *
```

```
createNewApp(QWidget* parent,
      QRectF inSceneRect,
      QGraphicsScene *scene,
      void* data
      )
{
  return new HelloWrld(parent, inSceneRect, scene, data);
}

HelloWrld::HelloWrld(QWidget* parent, QRectF inSceneRect,
    QGraphicsScene *scene, void* data):
ApplicationBase(parent, inSceneRect,scene,data)
{
    //Store the test data locally
    m_widgetTestInStruct = (TestTargetParamsStruct*)data;
}
QString HelloWrld::GetAppName(){return QString("HelloWrld");};
int HelloWrld::GetSwitchingTime()
{
  return (1000/m_widgetTestInStruct->rate_screen_refresh_fps);
};
```

2. Add the Init and process functions.

```
void HelloWrld::Init()
{
    pHelloVal =  m_scene->addText("Hello World");
    pHelloVal->moveBy(200,200);
}

void HelloWrld::Process()
{
   pHelloVal->moveBy(5,5);
   //Indicate that the processing is complete
   SetProcessCallComplete();
}
```

3. Create a .pro file with the following:

```
TEMPLATE = lib
SOURCES = helloworld.cpp
HEADERS += ../applicationmanager/common/applicationbase.h
```

4. Build the application as follows:

```
qmake qt_e.pro
        This will create a Makefile based on the Qt configuration.
make
```

The libraries need to be installed in the `/usr/lib` folder. If you want to install the libraries in a different path, use `LD_LIBRARY_PATH` to set the library path.

3.3 Building the Application Manager

The application manager can be copied to the target folder where the actual application resides. To build the application manager for a particular target:

1. Go to the `applicationmanager\qt_e` folder
2. `qmake qt_e.pro`
3. make

The *qt_e* binary created at this step is the applicationmanager.

An example folder structure will look like:

```
--applicationmanager
--application_name
        --application_name.cpp
        --application_name.pro
```

3.4 *Executing the Application*

To execute the application:

```
#qt_e  -qws <params>
```

params = Cookie, numWdgts, fx, transparency, texture, fontsize, fps, prebuiltID, wdgtsize, numFrames, appstring

example: /qt_e –qws 123 3 0 0 0 0 10 99 512 100 "helloworld"

Mandatory params:
- Cookie: number to be appended to the output file – example: 0,1,2
- fps: number of frames per second – example: 1,2
- numFrames: number of frames to run – example: 100,200
- appstring: name of the application library to run

Optional params:
- numWdgts – number of widgets in the application
- fx=0,1: enable/disable shadow to widgets when fx=1
- transparency: enable/disable transparency
- texture = 0,1,2: add a background texture 0-no; 1-PNG; 2-SVG
- fontsize – set font size: example: 8,16
- fontsize – set font size: example: 8,16 – set to 99 while executing custom application.
    - 0 – Industrial Automation blocks (Turbines, Boilers, Pipes, Text Info classes)
    - 1 – Text + Pictures (PNG/VG/Colors)
    - 2 – Involves GLWidget, needs powervr (not on HawkBoard)
    - 3 – Webkit Browser
    - 4 – Automotive Tacho
    - 5 – Medical ECG Monitor
    - 5 – Medical ECG Monitor
    - 7 – Sewing Machine UI
- wdgtsize – size of the widget: example:100,256,512

livemem

If a screen capture of the application is required, copy get_screen_dump.sh file from
\xgxperf\trunk\Packages\OMAP3_Graphics_SDK\xgxperf\server to the folder where the application is installed.

4   Results and Interpreting

Once the Process call returns, the application manager saves the results in an xml file.

Table 1 shows an example result of the ECG monitor application on the OMAP3530 device.

Table 1. XgxPerf_Profile_Test_Inputs

Preconfigured_Test_Name	ecgmonitor
Num_Widgets	3
Effects	0
Transparency	0
Texture_type	0
Font_Size	0
Screen_Refresh_Rate	10
Size_of_widgets	30
Number_of_frames_run	300

Table 2 shows that it took 3 mS to start the application, once it was initiated. Prior to that, it was taking about 35 mS per frame with an average CPU load of 29% on OMAP3530.

Table 2. XgxPerf_Profile_Test_Output_Profile_Parameters

Screen Init Time (mS)	3
Screen Update Time/Frame (mS)	35
Average CPU Load	29

5   References

- Building Qt, http://processors.wiki.ti.com/index.php/Building_Qt
- Download XgxPerf:
  - XgxPerf project page: https://gforge.ti.com/gf/project/gleslayer/
  - svn checkout: http://processors.wiki.ti.com/index.php/SVN_Access
- XgxPerf Support:
  - Linux support forum at http://e2e.ti.com/forums/

IMPORTANT NOTICE