

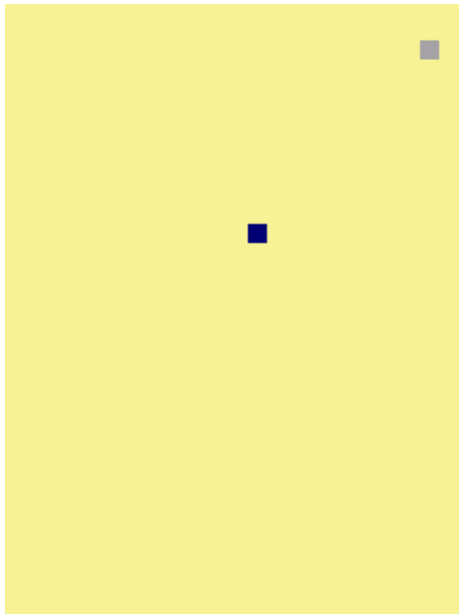


Contents

- [1 Introduction](#)
- [2 Main loop](#)
- [3 Using RGA API](#)
- [4 Initialize application environment](#)
- [5 Initialize display](#)
- [6 Initialize graphics](#)
- [7 Initialize keypad](#)
- [8 MyApplication.cpp](#)
- [9 MyRGAUI.h](#)
- [10 MyRGAUI.cpp](#)
- [11 Sources](#)
- [12 Links](#)

Introduction

This application shows how to create a simple application using the [RGA](#) (Real Time Graphics and Audio) APIs. The application draws two squares on the screen - one controlled with user input and one drawn randomly.



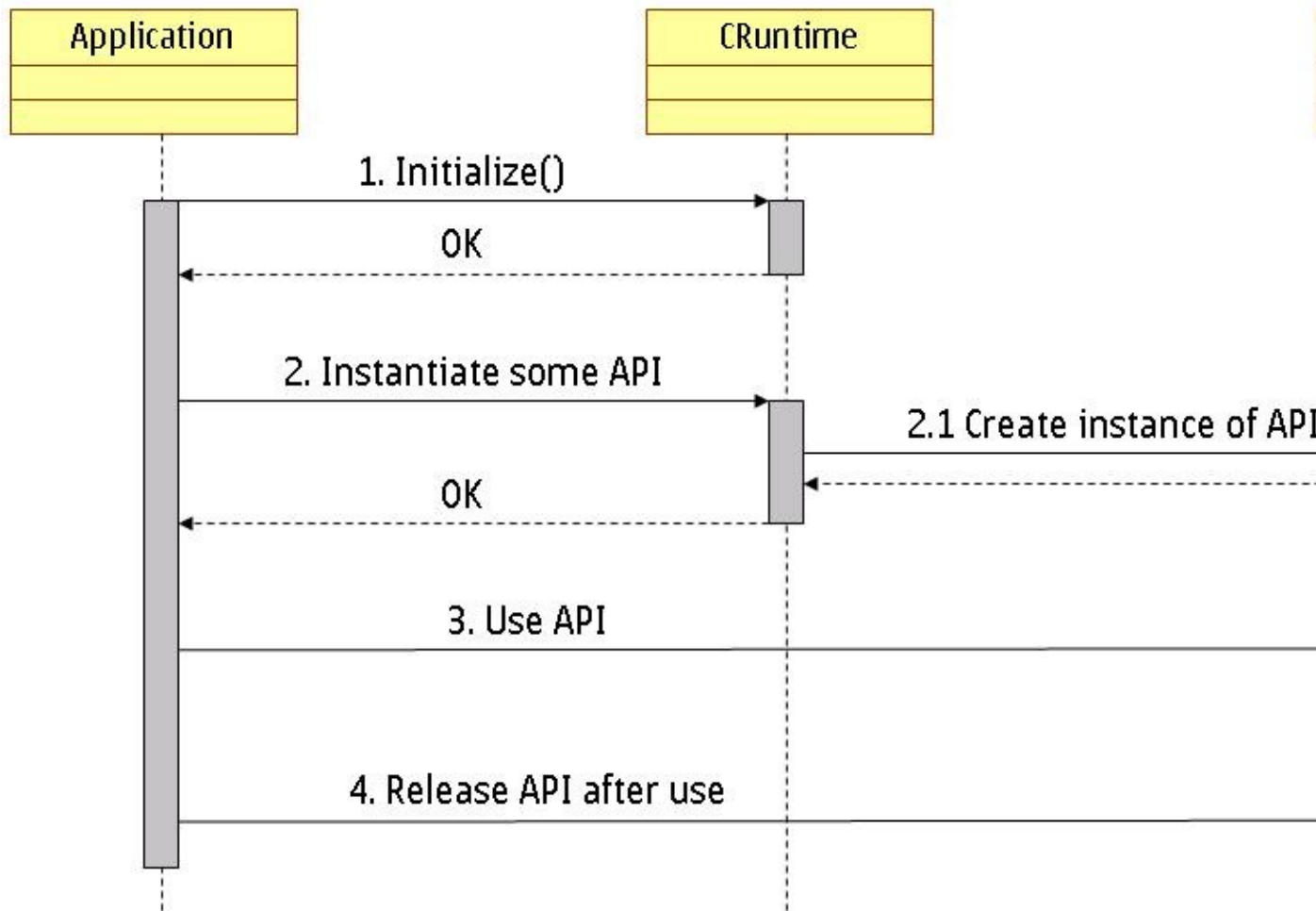
Main loop

```
void CMyRGAUI::Run()  
{  
    while ( mExit == FALSE )  
    {
```

Simple_RGA_application

```
if( mFocus )
{
    Draw();
    mIdle->Process( 2000, TRUE );
}
else
{
    mIdle->Process( 0, TRUE );
}
}
```

Using RGA API



Initialize application environment

This code initializes the execution environment. The **IApplicationState** (mApplicationState) object allows the application to get events when the application comes to foreground or goes to background. The **Idle**

Simple_RGA_application

(mIdle) object enables the waiting between application events.

```
int32 CMyRGAUI::Initialize()
{
    int32 errorCode;

    errorCode = CRuntime::CreateInstance( mIdle );
    if ( errorCode != OK ) return errorCode;

    errorCode = CRuntime::CreateInstance( mApplicationState );
    if ( errorCode != OK ) return errorCode;
    mApplicationState->SetObserver( this );

    ...
}
```

Initialize display

The **IDisplay** (mDisplay) object provides the interface to the devices screen. The **IDisplay** object is created using the **IDisplayManager** object.

```
/// Display number
const uint8 DISPLAY_NUMBER = 0;

/// Display configuration
const uint8 DISPLAY_CONFIGURATION = 0;

int32 CMyRGAUI::Initialize()
{
    int32 errorCode;

    ...

    IDisplayManager* displayManager = NULL;
    errorCode = CRuntime::CreateInstance( displayManager );
    if ( errorCode != OK ) return errorCode;

    errorCode = displayManager->CreateDisplay( DISPLAY_NUMBER, mDisplay );
    displayManager->Release();
    if( errorCode != OK ) return errorCode;

    IDisplayConfiguration* displayConfig = NULL;
    errorCode = mDisplay->GetDisplayConfiguration( DISPLAY_CONFIGURATION,
                                                    displayConfig );
    if ( errorCode != OK ) return errorCode;

    errorCode = mDisplay->CreateWindow( *displayConfig, mWindow );
    displayConfig->Release();
    if ( errorCode != OK ) return errorCode;
    mDisplay->SetObserver( this );

    ...
}
```

Initialize graphics

```

int32 CMyRGAUI::Initialize()
{
    int32 errorCode;

    ...

    IBackBufferFactory* bBFactory = NULL;

    errorCode = CRuntime::CreateInstance( bBFactory );
    if ( errorCode != OK ) return errorCode;

    //set the window handle to the required display mode
    //later this will be done by the Display API
    CBackBufferResolution bBResolution;
    uint32 colorDepthMask;
    errorCode = bBFactory->GetBackBufferConfiguration(
        BACKBUFFER_CONFIG, BACKBUFFER_RESOLUTION_MASK,
        BACKBUFFER_COLOR_MASK, bBResolution, colorDepthMask );

    if ( errorCode != OK )
    {
        bBFactory->Release();
        return errorCode;
    }
    errorCode = bBFactory->CreateBackBuffer( mWindow, bBResolution, colorDepthMask,
        BACKBUFFER_FLAG_ANTI_TEARING_DISABLED |
        BACKBUFFER_FLAG_CONSISTENT, mBackBuffer );

    bBFactory->Release();
    if ( errorCode != OK ) return errorCode;

    errorCode = CRuntime::CreateInstance( mGraphicsContext );
    if ( errorCode != OK ) return errorCode;
    mGraphicsContext->SetGraphicsDevice( *mBackBuffer );

    GraphicsOrientationType type;
    GraphicsOrientationAngle angle;
    mDisplay->GetOrientation( type, angle );
    mBackBuffer->Transpose( type, angle );

    ...
}

```

Initialize keypad

```

int32 CMyRGAUI::Initialize()
{
    ...

    IInput* input = NULL;
    errorCode = CRuntime::CreateInstance( input );
    if ( errorCode != OK ) return errorCode;

    int32 number = input->GetDeviceCount();
    for( int32 i = 0; i < number && !mKeypadDevice; ++i )
    {
        CInputDeviceInfo deviceInfo;
        errorCode = input->GetDeviceInfo( i, deviceInfo );
    }
}

```

Simple_RGA_application

```
    if ( errorCode != OK )
    {
        input->Release();
        return errorCode;
    }
    if( ( deviceInfo.mDeviceType == INPUT_DEVICE_TYPE_KEYPAD ) &&
        ( deviceInfo.mInterfaceType == INPUT_INTERFACE_TYPE_INTERNAL ) )
    {
        errorCode = input->CreateDevice( i, mKeypadDevice );
        if ( errorCode != OK )
        {
            input->Release();
            return errorCode;
        }
    }
    input->Release();

    if( mKeypadDevice == NULL ) return ERROR_UNEXPECTED;

    errorCode = mKeypadDevice->Start();
    if ( errorCode != OK ) return errorCode;
    mKeypadDevice->SetObserver( this );

    return OK;
}
```

MyApplication.cpp

```
// INCLUDE FILES
#include <errorcodes.h>           // ngi
#include <runtime.h>             // ngi

// This is a GCCE toolchain workaround needed when compiling with GCCE
// and using main() entry point
#ifdef __GCCE__
#include <staticlibinit_gcce.h>
#endif

#include "MyRGAUI.h"

using namespace ngi;

namespace MyExample
{
    /**
     * ExampleMain()
     *
     * Initializes the runtime, creates the application
     * instance and runs the application.
     */
    int ExampleMain()
    {
        int32 error;
        // Initializes CRuntime
        error = CRuntime::Initialize();
        if ( error == OK )
```

Simple_RGA_application

```
{
    CMyRGAUI* rgaUI = new CMyRGAUI();// create environment
    if ( rgaUI == NULL )
    {
        CRuntime::Close();
        return ERROR_OUT_OF_MEMORY;
    }
    error = rgaUI->Initialize();// initialize environment
    if ( error == OK )
    {
        rgaUI->Run();          // main loop
    }
    delete rgaUI;
    CRuntime::Close();
}
return error;
}
}

int main(void)
{
    return MyExample::ExampleMain();
}
```

MyRGAUI.h

```
#ifndef MYRGAUI_H_
#define MYRGAUI_H_

#include <applicationstate.h>
#include <backbuffer.h>
#include <display.h>
#include <graphicscontext.h>
#include <idle.h>
#include <input.h>

using namespace ngi;

namespace MyExample
{
    /// Display number
    const uint8 DISPLAY_NUMBER = 0;

    /// Display configuration
    const uint8 DISPLAY_CONFIGURATION = 0;

    /// Background color definition
    const uint32 COLOR_BACKGROUND = 0x00f0f090;

    /// Blue color definition
    const uint32 COLOR_BLUE = 0x00000077;

    /// Gray color definition
    const uint32 COLOR_GRAY = 0x00a0a0a0;

    /// Back buffer configuration to use
    const uint32 BACKBUFFER_CONFIG = 0;

    /// Possible back buffer resolutions
    const uint32 BACKBUFFER_RESOLUTION_MASK = BACKBUFFER_RESOLUTION_176x208 |
```

Simple_RGA_application

```
BACKBUFFER_RESOLUTION_208x176 |
BACKBUFFER_RESOLUTION_352x416 |
BACKBUFFER_RESOLUTION_416x352 |
BACKBUFFER_RESOLUTION_240x320 |
BACKBUFFER_RESOLUTION_320x240 |
BACKBUFFER_RESOLUTION_320x480 |
BACKBUFFER_RESOLUTION_480x320 |
BACKBUFFER_RESOLUTION_480x640 |
BACKBUFFER_RESOLUTION_640x480 |
BACKBUFFER_RESOLUTION_160x240 |
BACKBUFFER_RESOLUTION_240x160 |
BACKBUFFER_RESOLUTION_480x848 |
BACKBUFFER_RESOLUTION_848x480 |
BACKBUFFER_RESOLUTION_120x160 |
BACKBUFFER_RESOLUTION_160x120 |
BACKBUFFER_RESOLUTION_360x640 |
BACKBUFFER_RESOLUTION_640x360 |
BACKBUFFER_RESOLUTION_352x800 |
BACKBUFFER_RESOLUTION_800x352;

// Usable back buffer color resolutions
const uint32 BACKBUFFER_COLOR_MASK = GRAPHICS_FORMAT_XRGB4444 |
    GRAPHICS_FORMAT_XRGB1555 |
    GRAPHICS_FORMAT_RGB565 |
    GRAPHICS_FORMAT_RGB888 |
    GRAPHICS_FORMAT_XRGB8888;

class CMyRGAUI : public IApplicationStateObserver,
                public IDisplayObserver,
                public IInputDeviceObserver
{
public: // Constructor & Destructor

    CMyRGAUI      ();
    virtual ~CMyRGAUI();

public: // CMyRGAUI interface

    virtual int32 Initialize();
    virtual void Run();

public: // IApplicationStateObserver

    virtual void FocusGained() NO_THROW;
    virtual void FocusLost() NO_THROW;
    virtual void ExitRequested() NO_THROW;

public: // IDisplayObserver

    virtual void DisplayOrientationChanged( uint32 aDisplayIndex,
        GraphicsOrientationType aType, GraphicsOrientationAngle aAngle )
        NO_THROW;

    virtual void RedrawWindow( const IWindow& aWindow ) NO_THROW;

public: // IInputDeviceObserver

    virtual void InputKeyPressed( IInputDevice& aDevice,
        uint64 aTimeStamp, uint32 aKeyCode ) NO_THROW;

    virtual void InputKeyReleased( IInputDevice& aDevice,
        uint64 aTimeStamp, uint32 aKeyCode ) NO_THROW;
```

Simple_RGA_application

```
virtual void InputAxisMoved( IInputDevice& aDevice,
    uint64 aTimeStamp,  uint32 aAxisNumber, int32 aNewAxisValue )
    NO_THROW;

virtual void DeviceDisconnected( IInputDevice& aDevice ) NO_THROW;

virtual void DeviceConnected( IInputDevice& aDevice ) NO_THROW;

private:
    virtual void Draw();
    virtual void Cleanup();

private:
    /// Idle process
    IIdle* mIdle;

    /// Pointer to the application state object.
    IApplicationState* mApplicationState;

    /// Application display object
    IDisplay* mDisplay;

    /// Application window
    IWindow* mWindow;

    /// Pointer to the graphics context object.
    IGraphicsContext* mGraphicsContext;

    /// Pointer to the backbuffer object.
    IBackBuffer* mBackBuffer;

    /// Pointer to the keypad device object.
    IInputDevice* mKeypadDevice;

    /// Application has focus or not
    bool32 mFocus;

    /// Indicates, that the application has stopped.
    bool32 mExit;

    /// My point
    CPoint mPoint;

    /// Screen size
    CSize mResolution;
};

#endif /*MYRGAUI_H_*/
```

MyRGAUI.cpp

```
#include <stdlib.h>

#include <errorcodes.h>          // ngi
#include <runtime.h>            // ngi
#include <displaymanager.h>     // ngi
```

MyRGAUI.cpp

Simple_RGA_application

```
#include "MyRGAUI.h"

using namespace ngi;
using namespace MyExample;

CMyRGAUI::CMyRGAUI() :
    mIdle( NULL ),
    mApplicationState( NULL ),
    mDisplay( NULL ),
    mWindow( NULL ),
    mGraphicsContext( NULL ),
    mBackBuffer( NULL ),
    mKeypadDevice( NULL ),
    mFocus( TRUE ),
    mExit( FALSE )
{
}

CMyRGAUI::~CMyRGAUI()
{
    Cleanup();
}

int32 CMyRGAUI::Initialize()
{
    int32 errorCode;

    errorCode = CRuntime::CreateInstance( mIdle );
    if ( errorCode != OK ) return errorCode;

    errorCode = CRuntime::CreateInstance( mApplicationState );
    if ( errorCode != OK ) return errorCode;
    mApplicationState->SetObserver( this );

    //*****

    IDisplayManager* displayManager = NULL;
    errorCode = CRuntime::CreateInstance( displayManager );
    if ( errorCode != OK ) return errorCode;

    errorCode = displayManager->CreateDisplay( DISPLAY_NUMBER, mDisplay );
    displayManager->Release();
    if( errorCode != OK ) return errorCode;

    IDisplayConfiguration* displayConfig = NULL;
    errorCode = mDisplay->GetDisplayConfiguration( DISPLAY_CONFIGURATION,
        displayConfig );
    if ( errorCode != OK ) return errorCode;

    errorCode = mDisplay->CreateWindow( *displayConfig, mWindow );
    displayConfig->Release();
    if ( errorCode != OK ) return errorCode;
    mDisplay->SetObserver( this );

    //*****

    IBackBufferFactory* bBFactory = NULL;

    errorCode = CRuntime::CreateInstance( bBFactory );
    if ( errorCode != OK ) return errorCode;
}
```

Simple_RGA_application

```
//set the window handle to the required display mode
//later this will be done by the Display API
CBackBufferResolution bBResolution;
uint32 colorDepthMask;
errorCode = bBFactory->GetBackBufferConfiguration(
    BACKBUFFER_CONFIG, BACKBUFFER_RESOLUTION_MASK,
    BACKBUFFER_COLOR_MASK, bBResolution, colorDepthMask );
if ( errorCode != OK )
{
    bBFactory->Release();
    return errorCode;
}
errorCode = bBFactory->CreateBackBuffer( mWindow, bBResolution, colorDepthMask,
    BACKBUFFER_FLAG_ANTI_TEARING_DISABLED |
    BACKBUFFER_FLAG_CONSISTENT, mBackBuffer );

bBFactory->Release();
if ( errorCode != OK ) return errorCode;

errorCode = CRuntime::CreateInstance( mGraphicsContext );
if ( errorCode != OK ) return errorCode;
mGraphicsContext->SetGraphicsDevice( *mBackBuffer );

//*****

GraphicsOrientationType type;
GraphicsOrientationAngle angle;
mDisplay->GetOrientation( type, angle );
mBackBuffer->Transpose( type, angle );

//*****

mResolution = mGraphicsContext->GetGraphicsDevice()->GetSize();
mPoint.mX = mResolution.mX / 2;
mPoint.mY = mResolution.mY / 2;

//*****

IInput* input = NULL;
errorCode = CRuntime::CreateInstance( input );
if ( errorCode != OK ) return errorCode;

int32 number = input->GetDeviceCount();
for( int32 i = 0; i < number && !mKeypadDevice; ++i )
{
    CInputDeviceInfo deviceInfo;
    errorCode = input->GetDeviceInfo( i, deviceInfo );
    if ( errorCode != OK )
    {
        input->Release();
        return errorCode;
    }
    if( ( deviceInfo.mDeviceType == INPUT_DEVICE_TYPE_KEYPAD ) &&
        ( deviceInfo.mInterfaceType == INPUT_INTERFACE_TYPE_INTERNAL ) )
    {
        errorCode = input->CreateDevice( i, mKeypadDevice );
        if ( errorCode != OK )
        {
            input->Release();
            return errorCode;
        }
    }
}
}
```

Simple_RGA_application

```
input->Release();

if( mKeypadDevice == NULL ) return ERROR_UNEXPECTED;

errorCode = mKeypadDevice->Start();
if ( errorCode != OK ) return errorCode;
mKeypadDevice->SetObserver( this );

return OK;
}

void CMyRGAUI::Draw()
{
// copy the content of the graphics context to the ngi backbuffer
if( mBackBuffer && ( mBackBuffer->Lock() == OK ) )
{
CRect clearRegion = CRect( 0, 0, mResolution.mX, mResolution.mY );
mGraphicsContext->ClearRegion( COLOR_BACKGROUND, clearRegion,
GRAPHICS_FORMAT_RGB888 );

CRect square = CRect( mPoint.mX - 5, mPoint.mY - 5, 10, 10);
mGraphicsContext->ClearRegion( COLOR_BLUE, square,
GRAPHICS_FORMAT_RGB888 );

int32 x = random() % mResolution.mX;
int32 y = random() % mResolution.mY;

CRect rand = CRect( x - 5, y - 5, 10, 10);
mGraphicsContext->ClearRegion( COLOR_GRAY, rand,
GRAPHICS_FORMAT_RGB888 );

mBackBuffer->Unlock();
mBackBuffer->Swap( TRUE );
}
}

void CMyRGAUI::Run()
{
while ( mExit == FALSE )
{
if( mFocus )
{
(); Draw
->Pmōdēs( 2000, TRUE );
}
else
{
->Pmōdēs( 0, TRUE );
}
}
}

void CMyRGAUI::Cleanup()
{
if( mBackBuffer )
{
mBackBuffer->Release();
mBackBuffer = NULL;
}
if( mGraphicsContext )
{
mGraphicsContext->Release();
mGraphicsContext = NULL;
}
```

Simple_RGA_application

```
    }
    if( mWindow )
    {
        mWindow->Release();
        mWindow = NULL;
    }
    if( mDisplay )
    {
        mDisplay->SetObserver( NULL );
        mDisplay->Release();
        mDisplay = NULL;
    }
    if( mKeypadDevice )
    {
        mKeypadDevice->SetObserver( NULL );
        mKeypadDevice->Release();
        mKeypadDevice = NULL;
    }
    if( mApplicationState )
    {
        mApplicationState->SetObserver( NULL );
        mApplicationState->Release();
        mApplicationState = NULL;
    }
    if( mIdle )
    {
        mIdle->Release();
        mIdle = NULL;
    }
}

void CMyRGAUI::FocusGained() NO_THROW
{
    mFocus = TRUE;
}

void CMyRGAUI::FocusLost() NO_THROW
{
    mFocus = FALSE;
}

void CMyRGAUI::ExitRequested() NO_THROW
{
    mExit = TRUE;
}

void CMyRGAUI::DisplayOrientationChanged(
    uint32 aDisplayIndex,
    GraphicsOrientationType aType,
    GraphicsOrientationAngle aAngle ) NO_THROW
{
    if( aDisplayIndex == DISPLAY_NUMBER )
    {
        mBackBuffer->Transpose( aType, aAngle );

        mResolution = mGraphicsContext->GetGraphicsDevice()->GetSize();
        if ( mPoint.mX > mResolution.mX ) mPoint.mX = mResolution.mX;
        if ( mPoint.mY > mResolution.mY ) mPoint.mY = mResolution.mY;
    }
}

void CMyRGAUI::RedrawWindow( const IWindow& /*aWindow*/ ) NO_THROW
```

Simple_RGA_application

```
{
    (); Draw
}

void CMyRGAUI::InputKeyPressed(
    IInputDevice& /*aDevice*/,
    uint64 /*aTimeStamp*/,
    uint32 aKeyCode ) NO_THROW
{
    switch ( aKeyCode )
    {
        case INPUT_KEY_LSK:
        case INPUT_KEY_RSK:
            {
                =mFALSE;
                mExit = TRUE;
            }
            break;
        case INPUT_KEY_DIGITAL_UP:
            {
                if ( mPoint.mY > 0 ) mPoint.mY--;
                break;
            }
        case INPUT_KEY_DIGITAL_DOWN:
            {
                if ( mPoint.mY < mResolution.mY ) mPoint.mY++;
                break;
            }
        case INPUT_KEY_DIGITAL_LEFT:
            {
                if ( mPoint.mX > 0 ) mPoint.mX--;
                break;
            }
        case INPUT_KEY_DIGITAL_RIGHT:
            {
                if ( mPoint.mX < mResolution.mX ) mPoint.mX++;
                break;
            }
    }
}

void CMyRGAUI::InputKeyReleased(
    IInputDevice& /*aDevice*/,
    uint64 /*aTimeStamp*/,
    uint32 /*aKeyCode*/ ) NO_THROW
{
}

void CMyRGAUI::InputAxisMoved(
    IInputDevice& /*aDevice*/,
    uint64 /*aTimeStamp*/,
    uint32 /*aAxisNumber*/,
    int32 /*aNewAxisValue*/ ) NO_THROW
{
}

void CMyRGAUI::DeviceDisconnected( IInputDevice& /*aDevice*/ ) NO_THROW
{
}

void CMyRGAUI::DeviceConnected( IInputDevice& /*aDevice*/ ) NO_THROW
{
}
```

}

Sources

- [File:MyApplication.zip](#)

Here are the sources for a more advanced example also:

- [File:RGA Morse Code Example.zip](#)

Links

- [Introduction to RGA](#)
- [Using RGA with Carbide.c++](#)
- [Open C/C++ Plug-ins for S60 3rd Edition](#)

[Media:Example.ogg](#)