NintendoWare for CTR

Layout Programmer's Guide

2011/02/28

PROVISIONAL TRANSLATION

The content of this document is highly confidential and should be handled accordingly.

CTR-06-0044-001-A
Released: Review

Table of Contents

Code

Tables

Figures

1 Introduction

The lyt library, which is included in the NintendoWare for CTR (NintendoWare) LayoutLib package, is a library designed for CTR game layout programming. The library enables you to display the layout data that was created with the NintendoWare layout development tool, NW4C_LayoutEditor, on a CTR.

This document describes how to use the lyt library to put together a layout program. First, it details how to build the layout program development environment, then it uses sample demos to explain in concrete terms how to use the layout program. Finally, it describes the structure of the lyt library, the classes that have been prepared, and how they interrelate.

For details about the individual classes, see the Function Reference.

2 Layout Program Development Environment

This chapter explains how to build the layout program development environment.

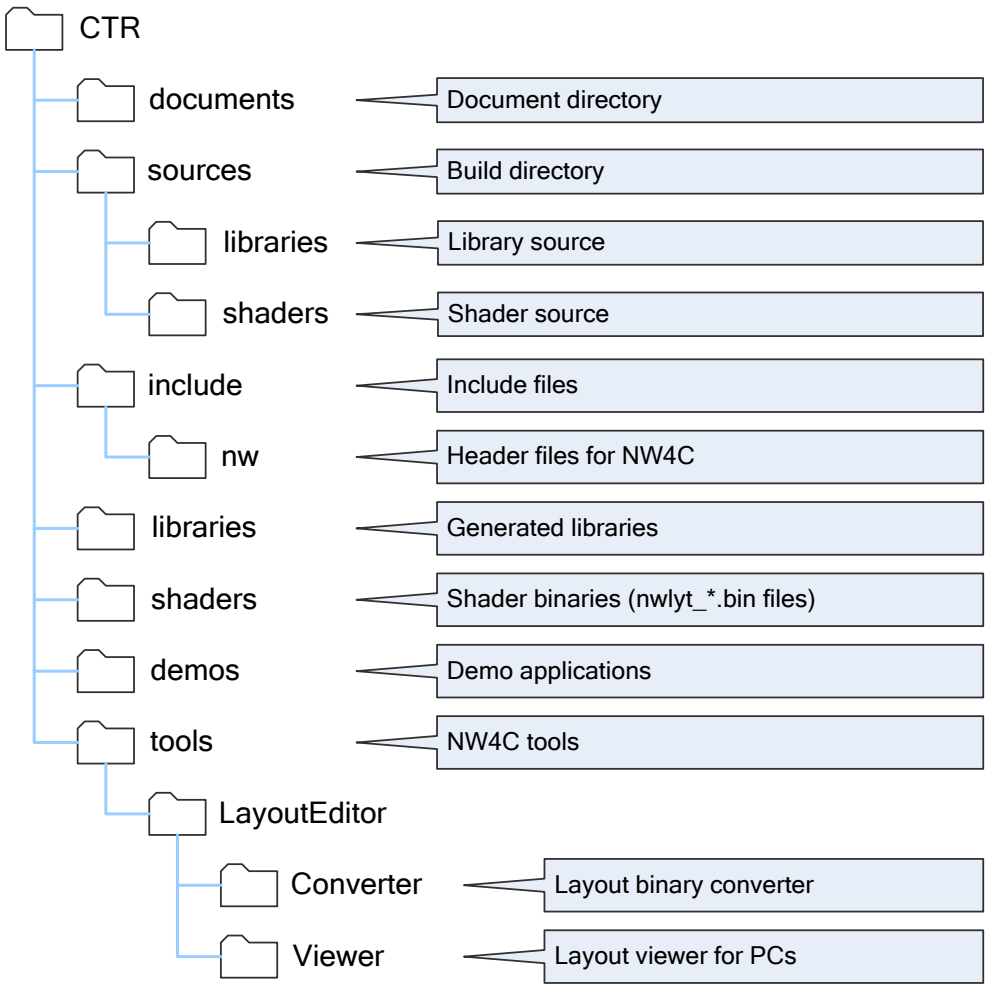
2.1 lyt Library

The layout program is put together using the lyt library, which is included in the NintendoWare package.

2.2 Directory Structure

In the directory structure of the lyt library, `$NW4C_ROOT` represents the NintendoWare install directory, `NintendoWare/CTR`.

Figure 2-1 Directory Structure



2.3 Development Using the lyt Library

Before using the lyt library for development, you must first configure the settings described below.

2.3.1 Configure the NintendoWare Build Environment

First, you need to construct the NintendoWare build environment. For details, see the *NintendoWare for CTR Programmer's Guide*.

2.3.2 Include the Build Definitions

The CTR_SDK contains a definitions file for building CTR programs using the OMake build system.

The NintendoWare package provides a definitions file for adding the definitions needed to use the NintendoWare library.

There are two types of OMake configuration files: the OMakeroot file, which is placed in the root directory of the project, and the OMakefile files placed in subdirectories. Usually the NintendoWare definitions file is included in the OMakeroot file.

Code 2-1 Including the Build Definitions File

```
include $(getenv NW4C_ROOT)/build/omake/commondefs
```

By adding this line of code you can link library files and specify include paths required for use of the lyt library.

2.3.3 Provide the Shader Binaries

In the ROM file system, the `shaders` directory includes the shader binary files needed for the lyt library.

The names of the shader binary files required by the lyt library are defined by `NW_LYT_SHADER_NAMES` in the build definitions file.

Note: Files in the directory specified as `ROMFS_ROOT` by OMakefile are located in the ROM file system.

2.3.4 Provide the Layout Data

The layout data created by the layout designer is passed to the programmer in folders containing the layout and animation data, and binary font and texture data.

The programmer must place this data in the ROM file system to use it.

Note: In the lyt library samples, these files are first archived and then placed in the file system.

3 Quick Start

This chapter uses a sample demo to illustrate how to put together a simple layout program.

First, we build and execute the sample demo. Then, we look at the source code to see how the layout program is put together.

3.1 Executing the Sample Demo

3.1.1 Sample Demo Location

Sample demos are stored in the `$NW4C_ROOT/demos/lyt` directory. For this example, we use the demo named `simple`.

3.1.2 Building the Sample Demo

To build the sample lyt demo, move to the directory where the sample resides and then run `omake`.

Code 3-1 Demo Build

```
$ cd $NW4C_ROOT/demos/lyt/simple
$ omake
```

If the build is successful, the file `simple.cci` is created in the `simple/images/CTR-TS.Process.MPCore.fast/Development/ directory`.

Note: The actual directory name will differ depending on the build option.

3.1.3 Running the Sample Demo

To run the sample demo, load the created file `simple.cci` into the PARTNER-CTR debugger. To do this, from the PARTNER-CTR debugger, select **File (F)** > **Load (L),** and then open `simple.cci.`

Next, select **Run (R)** > **Run Program (G)** to start the sample program. If the sample demo starts normally, the layout panes will display as they are animated in the screen of the actual hardware.

3.2 Source Code

This section describes the layout program by looking at the source code. This source code is from the file `main.cpp`, which is located in the `$NW4C_ROOT/ demos/lyt/simple/sources` directory. Comments have been omitted as have portions of the code that are irrelevant to an explanation of the layout program.

Code 3-2 main.cpp for simple demo

```
#include <GLES2/gl2.h>
#include <nn.h>
#include <nn/fs.h>
```

```
#include <nw/sdk.h>
#include <nw/demo.h>
#include <nw/lyt.h>

namespace
{

void
InitGX()
{
    glClearColor(0.3f, 0.3f, 0.3f, 1.0f);
}

void
InitDraw(int width, int height)
{
    // Color buffer information
    // Face toward LCD and substitute width and height
    const nw::font::ColorBufferInfo colBufInfo =
    {
        height, width, PICA_DATA_DEPTH24_STENCIL8_EXT
    };

    const u32 commands[] =
    {
        // View port settings
        NW_FONT_CMD_SET_VIEWPORT(0, 0, colBufInfo.width, colBufInfo.height),

        // Disable scissoring
        NW_FONT_CMD_SET_DISABLE_SCISSOR(colBufInfo),

        // Disable w buffer
        // Set depth range
        // Disable polygon offset
        NW_FONT_CMD_SET_WBUFFER_DEPTHRANGE_POLYGONOFFSET(
            0.0f,           // wScale : 0.0 disables w buffer
            0.0f,           // depth range near
            1.0f,           // depth range far
            0,              // polygon offset units : 0.0 disables polygon offset
            colBufInfo),
    };
```

```
    nngxAdd3DCommand(commands, sizeof(commands), true);

    static const u32 constCommands[] =
    {
        // Disable culling
        NW_FONT_CMD_SET_CULL_FACE(NW_FONT_CMD_CULL_FACE_DISABLE),

        // Disable stencil test
        NW_FONT_CMD_SET_DISABLE_STENCIL_TEST(),

        // Disable depth test
        // All components of color buffer can be overwritten
        NW_FONT_CMD_SET_DEPTH_FUNC_COLOR_MASK(
            false,  // isDepthTestEnabled
            0,      // depthFunc
            true,   // depthMask
            true,   // red
            true,   // green
            true,   // blue
            true),  // alpha

        // Disable early depth test
        NW_FONT_CMD_SET_ENABLE_EARLY_DEPTH_TEST(false),

        // Restrict framebuffer access
        NW_FONT_CMD_SET_FBACCESS(
            true,   // colorRead
            true,   // colorWrite
            false,  // depthRead
            false,  // depthWrite
            false,  // stencilRead
            false), // stencilWrite
    };

    nngxAdd3DCommand(constCommands, sizeof(constCommands), true);
}

void
SetupCamera(nw::lyt::DrawInfo& drawInfo, const nw::lyt::Layout& layout)
{
    nw::ut::Rect layoutRect = layout.GetLayoutRect();
```

```
    f32 znear = 0.f;
    f32 zfar = 500.f;

    nw::math::MTX44 projMtx;
    nw::math::MTX44OrthoPivot(
        &projMtx,
        layoutRect.left,   // left
        layoutRect.right,  // right
        layoutRect.bottom, // bottom
        layoutRect.top,    // top
        znear,
        zfar,
        nw::math::PIVOT_UPSIDE_TO_TOP);
    drawInfo.SetProjectionMtx(projMtx);

    nw::math::VEC3 pos(0, 0, 1);
    nw::math::VEC3 up(0, 1, 0);
    nw::math::VEC3 target(0, 0, 0);

    nw::math::MTX34 viewMtx;
    nw::math::MTX34LookAt(&viewMtx, &pos, &up, &target);
    drawInfo.SetViewMtx(viewMtx);
}

}   // namespace {anonymous}

/*---------------------------------------------------------------------------*
  Main functions of @brief sample
 *---------------------------------------------------------------------------*/
void
nnMain()
{
    // Initialize the basic library
    nn::os::Initialize();
    nn::fs::Initialize();

    // Initialize application
    nw::demo::SimpleApp& demoApp = nw::demo::SimpleApp::GetInstance();
    demoApp.Initialize();
```

```
// Initialize the layout library
nw::lyt::Initialize(&demoApp.GetAllocator(), &demoApp.GetDeviceAllocator());

// Create layout
nw::lyt::Layout* pLayout = new nw::lyt::Layout();

// Load layout’s binary resource (archive)
File fileLayout;
if (!fileLayout.Read(
          NW_DEMO_FILE_PATH(L"layout.arc"),
          demoApp.GetDeviceAllocator()))

{
    NW_FATAL_ERROR("cannot open layout archive.\n");
}

// Specify root directory of binary resource and create resource accessor
nw::lyt::ArcResourceAccessor* pResAccessor
 = new nw::lyt::ArcResourceAccessor;
if (!pResAccessor->Attach(fileLayout.Buffer(), "."))
{
    NW_FATAL_ERROR("cannot attach layout archive.\n");
}

// Load layout resource
{
    const void* lytRes = pResAccessor->GetResource(0, "simple.bclyt");
    NW_NULL_ASSERT(lytRes);
    pLayout->Build(lytRes, pResAccessor);
}

nw::lyt::AnimTransform* pAnimTrans = 0;
// Load animation resource
{
    const void* lpaRes = pResAccessor->GetResource(0, "simple.bclan");
    NW_NULL_ASSERT(lpaRes);
    pAnimTrans = pLayout->CreateAnimTransform(lpaRes, pResAccessor);
    pLayout->BindAnimation(pAnimTrans);
}

f32 animFrame = 0;
```

```
nw::lyt::GraphicsResource graphicsResource;

// Load global resource file
{
    graphicsResource.StartSetup();
    const wchar_t* resourcePath = 0;
    for (int i = 0;
         (resourcePath = graphicsResource.GetResourcePath(i)) != NULL;
         ++i)
    {
        File file;
        if (!file.Read(resourcePath, demoApp.GetAllocator()))
        {
            NW_FATAL_ERROR("cannot read lyt resource file.");
        }
        graphicsResource.SetResource(i, file.Buffer(), file.Size());
    }
    graphicsResource.FinishSetup();
}

nw::lyt::DrawInfo drawInfo;
drawInfo.SetGraphicsResource(&graphicsResource);

nw::lyt::Drawer drawer;
drawer.Initialize(graphicsResource);


InitGX();

bool mainloop = true;
while (mainloop)
{
    demoApp.SetRenderingTarget(demoApp.DISPLAY0);
    {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        SetupCamera(drawInfo, *pLayout);

        InitDraw(demoApp.DISPLAY0_WIDTH, demoApp.DISPLAY0_HEIGHT);
        pLayout->Animate();
        pLayout->CalculateMtx(drawInfo);
```

```
            drawer.DrawBegin(drawInfo);
            drawer.Draw(pLayout, drawInfo);
            drawer.DrawEnd(drawInfo);

        }

        demoApp.SetRenderingTarget(demoApp.DISPLAY1);
        {
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        }

        // Update framebufffer
        animFrame += 1.0f;
        while (animFrame >= pAnimTrans->GetFrameSize())
        {
            animFrame -= pAnimTrans->GetFrameSize();
        }
        pAnimTrans->SetFrame(animFrame);

        demoApp.SwapBuffer(demoApp.DISPLAY_BOTH);
    }

    delete pLayout;
    delete pResAccessor;

    demoApp.GetDeviceAllocator().Free(fileLayout.Buffer());
}
```

3.2.1 Referenced Header Files

The header files are referenced as shown below.

Code 3-3 Referenced Header Files

```
#include <GLES2/gl2.h>
#include <nn.h>
#include <nn/fs.h>
  #include <nw/sdk.h>
  #include <nw/demo.h>
#include <nw/lyt.h>
```

To use the lyt library, you must reference its header file, `nw/lyt.h`.

3.2.2 Namespace

The namespace for the lyt library is defined as `nw::lyt`, which for simplicity is shortened to `lyt`.

3.3 Initialization

To use the lyt library, you must first initialize it as follows:

Code 3-4 Initializing the Library

```
nw::lyt::Initialize(&demoApp.GetAllocator(), &demoApp.GetDeviceAllocator());
```

The initialization function specifies two memory allocators as arguments.

The first allocator allocates memory blocks from regular memory, while the second allocates memory blocks from device memory.

3.4 Creating a Layout

Create a `Lyt::Layout` class object as follows.

Code 3-5 Creating Layout

```
nw::lyt::Layout* pLayout = new nw::lyt::Layout();
```

3.5 Loading the Layout Data

Before displaying the layout you must first load the necessary layout data. The data is loaded in units of layout resources and animation resources.

To load a layout resource, pass the starting address of the resource and the resource accessor to the `Build` function of the `lyt::Layout` class. The binary font and texture data required for layout display is loaded at this time.

To load an animation resource, pass the starting address of the resource and the resource accessor to the `CreateAnimTransform` function of the `lyt::Layout` class, and then retrieve the pointer to the `lyt::AnimTransform` class that is created. Next, pass this pointer to the `BindAnimation` function of the `lyt::Layout` class to bind the animation to the pane and the material. The texture data used for texture pattern animation is loaded at this time.

Code 3-6 Loading the Layout Data

```
// Create a resource accessor specifying the root directory
// for binary resources
nw::lyt::ArcResourceAccessor* pResAccessor
    = new nw::lyt::ArcResourceAccessor;
if (!pResAccessor->Attach(fileLayout.Buffer(), "."))
{
    NW_FATAL_ERROR("cannot attach layout archive.\n");
```

```
}
// Read the layout resource
{
    const void* lytRes = pResAccessor->GetResource(0, "simple.bclyt");
    NW_NULL_ASSERT(lytRes);
    pLayout->Build(lytRes, pResAccessor);
}

nw::lyt::AnimTransform* pAnimTrans = 0;
// Read the animation resource
{
    const void* lpaRes = pResAccessor->GetResource(0, "simple.bclan");
    NW_NULL_ASSERT(lpaRes);
    pAnimTrans = pLayout->CreateAnimTransform(lpaRes, pResAccessor);
    pLayout->BindAnimation(pAnimTrans);
}
```

3.6 Initializing Graphics Resources

The `GraphicsResource` class maintains objects used by shader (and other) programs for rendering. Graphics resources are initialized as shown in Code 3-7.

Code 3-7 Initializing Graphics Resources

```
nw::lyt::GraphicsResource graphicsResource;
    // Read the global resource files
    {
        graphicsResource.StartSetup();
        const wchar_t* resourcePath = 0;
        for (int i = 0;
             (resourcePath = graphicsResource.GetResourcePath(i)) != NULL;
             ++i)
        {
            File file;
            if (!file.Read(resourcePath, demoApp.GetAllocator()))           {
                NW_FATAL_ERROR("cannot read lyt resource file.");
            }
            graphicsResource.SetResource(i, file.Buffer(), file.Size());
        }
        graphicsResource.FinishSetup();
    }
```

An initialized graphics resource is configured in the rendering information (`DrawInfo`).

Code 3-8 Initializing Graphics Resources

```
nw::lyt::DrawInfo drawInfo;
drawInfo.SetGraphicsResource(&graphicsResource);
```

3.7 Initializing Drawer

Create a `Drawer` class object for rendering the layout as follows.

Code 3-9 Initialization of Drawer

```
    nw::lyt::Drawer drawer;
    drawer.Initialize(graphicsResource);
```

3.8 Setting Up the Camera

The lyt library uses polygons and textures to display layouts. The application must configure the rectangular region in which the layout will be displayed as well as the projection and model view matrix, which are required for rendering.

Code 3-10 Setting Up the Camera

```
void
SetupCamera(nw::lyt:: DrawInfo& drawInfo, const nw::lyt::Layout& layout)
{
    nw::ut::Rect layoutRect = layout.GetLayoutRect();

    f32 znear = 0.f;
    f32 zfar = 500.f;

    // Set the projection matrix to an orthogonal projection
    //(The Layout data is sideways, so the direction changes)
    nw::math::MTX44 projMtx;
    nw::math:: MTX44OrthoPivot (
        &projMtx,
        layoutRect.left,    // left
        layoutRect.right,   // right
        layoutRect.bottom, // bottom
        layoutRect.top,     // top
              znear,

        znear,
        zfar,
       nw::math::PIVOT_UPSIDE_TO_TOP);
      drawInfo.SetProjectionMtx(projMtx);
```

```
    // Set the model view matrix
   // (The Layout data is sideways, so the upward direction of the screen is the x-
direction of the layout)
      nw::math::VEC3 pos(0, 0, 1);
      nw::math::VEC3 up(0, 1, 0);
      nw::math::VEC3 target(0, 0, 0);

     nw::math::MTX34 viewMtx;
     nw::math::MTX34LookAt(&viewMtx, &pos, &up, &target);
     drawInfo.SetViewMtx(viewMtx);
}
```

3.9 Displaying the Layout

To display the layout, first calculate the animation results for the current playback frame. Once the matrix calculations necessary for display are done, render the layout.

The `Drawer` class is used for layout drawing.

Code 3-11 Displaying the Layout

```
InitDraw(demoApp.DISPLAY0_WIDTH, demoApp.DISPLAY0_HEIGHT);
pLayout->Animate();
pLayout->CalculateMtx(drawInfo);

drawer.DrawBegin(drawInfo);
drawer.Draw(pLayout, drawInfo);
drawer.DrawEnd(drawInfo);
```

The lyt library does not configure the following GL settings; it leaves that up the application:

- Culling
- Depth test
- Stencil test
- Scissoring
- Masking
- Polygon offset
- Early depth test

In the sample demo, the `InitDraw` function configures the appropriate drawing settings for 2D expression.

Code 3-12 Configuring the Render Mode

```
void
InitDraw(int width, int height)
{
    // Color buffer information
    // Face toward LCD and substitute width and height
    const nw::font::ColorBufferInfo colBufInfo =
    {
        height, width, PICA_DATA_DEPTH24_STENCIL8_EXT
    };

    const u32 commands[] =
    {
        // Set Viewport
        NW_FONT_CMD_SET_VIEWPORT(0, 0, colBufInfo.width, colBufInfo.height),

        // Disable scissoring
        NW_FONT_CMD_SET_DISABLE_SCISSOR(colBufInfo),

        // Disable w buffer
        // Set depth range
        // Disable polygom offset
        NW_FONT_CMD_SET_WBUFFER_DEPTHRANGE_POLYGONOFFSET(
            0.0f,           // wScale : 0.0 disables w buffer
            0.0f,           // depth range near
            1.0f,           // depth range far
            0,              // polygon offset units : 0.0 disables polygon offset
            colBufInfo),
    };

    nngxAdd3DCommand(commands, sizeof(commands), true);

    static const u32 constCommands[] =
    {
        // Disable culling
        NW_FONT_CMD_SET_CULL_FACE(NW_FONT_CMD_CULL_FACE_DISABLE),

        // Disable stencil test
        NW_FONT_CMD_SET_DISABLE_STENCIL_TEST(),

        // Disable depth test
        // All color buffer components can be overwritten
```

```
        NW_FONT_CMD_SET_DEPTH_FUNC_COLOR_MASK(
            false,  // isDepthTestEnabled
            0,      // depthFunc
            true,   // depthMask
            true,   // red
            true,   // green
            true,   // blue
            true),  // alpha

        // Disable early depth test
        NW_FONT_CMD_SET_ENABLE_EARLY_DEPTH_TEST(false),

        // Restrict framebuffer access
        NW_FONT_CMD_SET_FBACCESS(
            true,   // colorRead
            true,   // colorWrite
            false,  // depthRead
            false,  // depthWrite
            false,  // stencilRead
            false), // stencilWrite
    };

    nngxAdd3DCommand(constCommands, sizeof(constCommands), true);
}
```

3.10 Animation Process

The animation's playback frame is updated to the frame that will play when it is time for the next drawing process to occur.

Code 3-13 Frame Processing

```
// Update frame
animFrame += 1.0f;
while (animFrame >= pAnimTrans->GetFrameSize())
{
    animFrame -= pAnimTrans->GetFrameSize();
}
pAnimTrans->SetFrame(animFrame);
```

The lyt library does not automatically update the playback frame, so the program must set the next frame that will be played. If you are playing animation to match the video frames, your program should update the playback frame in every video frame.

4 Basic Operations in the Pane

This chapter explains a number of basic operations that can be performed in the pane in which the layout is being displayed.

For details about the functions introduced here (as well as some other functions not mentioned here), see the Function Reference Manual.

4.1 Search for Panes

The panes in which layouts are displayed are all child panes of a single root pane. To search for a pane, use the `lyt::Layout` class function `GetRootPane` to get the pointer to the root pane, then search using the `lyt::Layout` class function `FindPaneByName` to get the target pane.

Code 4-1 Searching for Panes

```
// Get the root pane
nw::lyt::Pane* pRoot = pLayout->GetRootPane();

// Search for pane using name
nw::lyt::Pane* pTarget = pRoot->FindPaneByName("Pane_0");
```

4.2 Move, Rotate, and Enlarge Panes

Use the functions `SetTranslate`, `SetRotate`, and `SetScale` to move, rotate, and enlarge panes. Each modification affects all child panes.

Use the `SetSize` function to change the size of the pane (and not the scale). Use the `SetSRTElement` function to change only specified elements of the pane.

Code 4-2 Move, Rotate, and Enlarge Panes

```
// Move the pane to local coordinates (120.0, 120.0, 0.0)
pPane->SetTranslate(nw::math::VEC3(120.0f, 120.0f, 0.0f));

// Rotate the pane 30 degrees clockwise
pPane->SetRotate(nw::math::VEC3(0.0f, 0.0f, -30.0f));

// Show the pane at double the original size
pPane->SetScale(nw::math::VEC2(2.0f, 2.0f));

// Change the size of the pane to (128.0, 128.0)
pPane->SetSize(nw::lyt::Size(128.0f, 128.0f));
```

4.3 Show and Hide Panes

Use the `SetVisible` function to toggle between showing and hiding the display of panes. The change also affects all child panes.

Code 4-3 Show and Hide Panes

```
// Hide the pane and all its children
pPane->SetVisible(false);

// Show the pane and all its children
pPane->SetVisible(true);
```

4.4 Control Animation

There are also functions to bind and unbind animation to only a specific pane, or to it and its child panes, and to enable and disable animations.

Code 4-4 Control Animation

```
// Bind an animation (only to the specified pane)
pPane->BindAnimation(pAnm1, false);

// Bind an animation (To the specified pane and its children)
pPane->BindAnimation(pAnm2, true);

// Disable animation
pPane->SetAnimationEnable(pAnm2, false, true);

// Unbind animation
pPane->UnbindAnimation(pAnm2);
```

5 Group Features

This chapter explains how to use the layout library's Group features.

5.1 Structure of Group Features

The Group features are comprised of these two classes:

- `Group`
- `GroupContainer`

5.1.1 Group

The `Group` class is for grouping multiple panes and processing them as a group. This is used to control groups of panes for specific purposes.

5.1.2 GroupContainer

The `GroupContainer` class is for managing the multiple groups that exist in the layout. For the layout to access groups, it must search for them via `GroupContainer`.

5.2 Executing the Group Sample Demo

The sample demo named `Group` makes use of the `Group` features, so let's execute that sample demo.

5.2.1 Starting the Sample Demo

The sample demo is stored in the directory `$NW4C_ROOT/demos/lyt/group`, so you can start it the same way you started the `simple` demo.

5.2.2 Explanation of Display

The screen displays nine picture panes. These nine panes are divided into three groups (left, center, and right) of three vertical panes each. The panes that belong to the current group are all shown rotated 30 degrees counterclockwise.

5.2.3 Controlling the Display

Press the A Button of Controller 1 to switch the current pane.

5.3 Searching for Groups

To search for a group, use the `GroupContainer` class function `FindGroupByName` and search using the group name.

Code 5-1 Searching for a Group

```
pCurrentGroup =
    pLayout->GetGroupContainer()->FindGroupByName(groupName[currentGroup]);
```

Instead of searching by name, you can also get a list of the groups in `GroupContainer` by using the `GetGroupList` function.

Code 5-2 Getting the List of Groups

```
nw::lyt::GroupList& groupList =
        pLayout->GetGroupContainer()->GetGroupList();

for (nw::lyt::GroupList::Iterator it = groupList.GetBeginIter();
        it != groupList.GetEndIter(); ++it)
{
    // Here is where you would describe processes for each group (it->mTarget).
}
```

5.4 Getting Panes That Belong to a Group

Use the `GetPaneList` function to get the list of panes that belong to a group (`lyt::PaneLinkList`) and access them via an iterator.

Code 5-3 Getting Panes That Belong to a Group

```
nw::lyt::PaneLinkList& paneList = pCurrentGroup->GetPaneList();

for (nw::lyt::PaneLinkList::Iterator it = paneList.GetBeginIter();
        it != paneList.GetEndIter(); ++it)
{
    it->mTarget->SetRotate(nw::math::VEC3(0.f, 0.f, 30.f));
}
```

6 Accessing Resources Processed with arc

The sample programs use layout data that has been archived.

The lyt library can access resources that have been archived. Unless you want to manage resources on your own, you can use the library's accessor for easy management of resources.

6.1 ArcResourceAccessor

This class manages a single `arc` file and accesses resources.

The resource type determines which directory is searched. Consequently, when you archive the output from the layout converter (`NW4C_LayoutConverter`), do not make any modifications, such as changing the file locations or directory names.

6.2 MultiArcResourceAccessor

This class manages multiple `arc` files and accesses resources.

If multiple layouts share use of the same texture images and font data, you can separate these layout resources from other resources and archive them separately. Then, when you load the resources, you can use this class to reduce the volume of content on the optical disc drive that holds the layout data.

For more details on how to use this class, see the `multiArc` sample demo.

6.3 Building Your Own Way to Access Resources

Instead of inheriting the `ResourceAccessor` class, you can build your own procedure for accessing resources.

To do this, override the `GetResource` function to return the resource's starting address and size from its type and name. The resource types are declared in the `nw/lyt/lyt_Resources.h` header file. If you are going to use the binary files that get output from the layout converter, be aware that the directory where these files get placed differ depending on the resource type.

To access your own resources, you also need to override the various `GetFont` and `GetTexture` functions. To implement this, use the `lyt::FontContainer` and `lyt::TextureContainer` classes.

7 Loading Textures in VRAM

Sometimes rendering performance can be improved by loading textures in VRAM.

This chapter describes library functions for loading textures in VRAM.

7.1 Loading Textures in VRAM

The resource assessor calls the LoadTexture function in the case of the first texture requested. The LoadTexture function loads the texture in VRAM.

There are two ways to specify that textures be loaded into VRAM using the LoadTexture function.

- Specification by argument
  Specify NN_GX_MEM_VRAMA | GL_NO_COPY_FCRAM_DMP or NN_GX_MEM_VRAMB | GL_NO_COPY_FCRAM_DMP using the `texLoadFlag` argument.
  Use this method when customizing the operations of a resource accessor-derived class,
- Specification by texture image resource
  Make VRAM layout settings for texture image resources ahead of time.

Texture image resources must be put in FCRAM. The same is true for textures loaded in VRAM

7.2 Texture Image Resource Settings

Use the `TexResource` class to make settings for loading textures into VRAM for a texture image resource.

The `TexResource` class is an accessor to texture image resources.

The `SetImageArea` member function specifies whether to load the texture into VRAMA or VRAMB.

Textures having a 4-bit format cannot be loaded into the same VRAM as another texture and used together. Use the `Is4BitFormat` member function to find if a texture resource has a 4-bit format.

To access a texture image resource inside an archive file, you must use the ARC function to get the resource inside that archive file.

You can use the `ArcUtl` class to get resources accessed by the `ArcResourceAccessor` class.

You can open the directory in which to search for texture images using the `OpenTextureDir` member function. The `ReadTextureDir` member function gets the texture image resource directory entry.

Code 7-1 Configuring the VRAM Layout for a Texture Image Resource

```
void SetTexAreaToVRAM(void* pLayoutArc)
{
    nw::lyt::ARCHandle arcHandle;
    nw::lyt::ARCInitHandle(pLayoutArc, &arcHandle);

    nw::lyt::ARCDir arcDir;
    if (!nw::lyt::ArcUtil::OpenTextureDir(&arcHandle, L"/", &arcDir))
    {
        return;
    }

    nw::lyt::ARCDirEntry arcDirEnt;
    while (nw::lyt::ArcUtil::ReadTextureDir(&arcDir, &arcDirEnt))
    {
        nw::lyt::ARCFileInfo arcFileInfo;
        if (!nw::lyt::ARCOpen(&arcHandle, &arcDirEnt, &arcFileInfo))
        {
            continue;
        }

        nw::lyt::TexResource texResource;
        if (!texResource.Set(arcFileInfo))
        {
            continue;
        }

        if (texResource.Is4bitFormat())
        {
            texResource.SetImageArea(nw::lyt::MEMAREA_VRAMA);
        }
        else
        {
            texResource.SetImageArea(nw::lyt::MEMAREA_VRAMB);
        }

        nw::lyt::ARCClose(&arcFileInfo);
    }

    nw::lyt::ARCCloseDir(&arcDir);
}
```

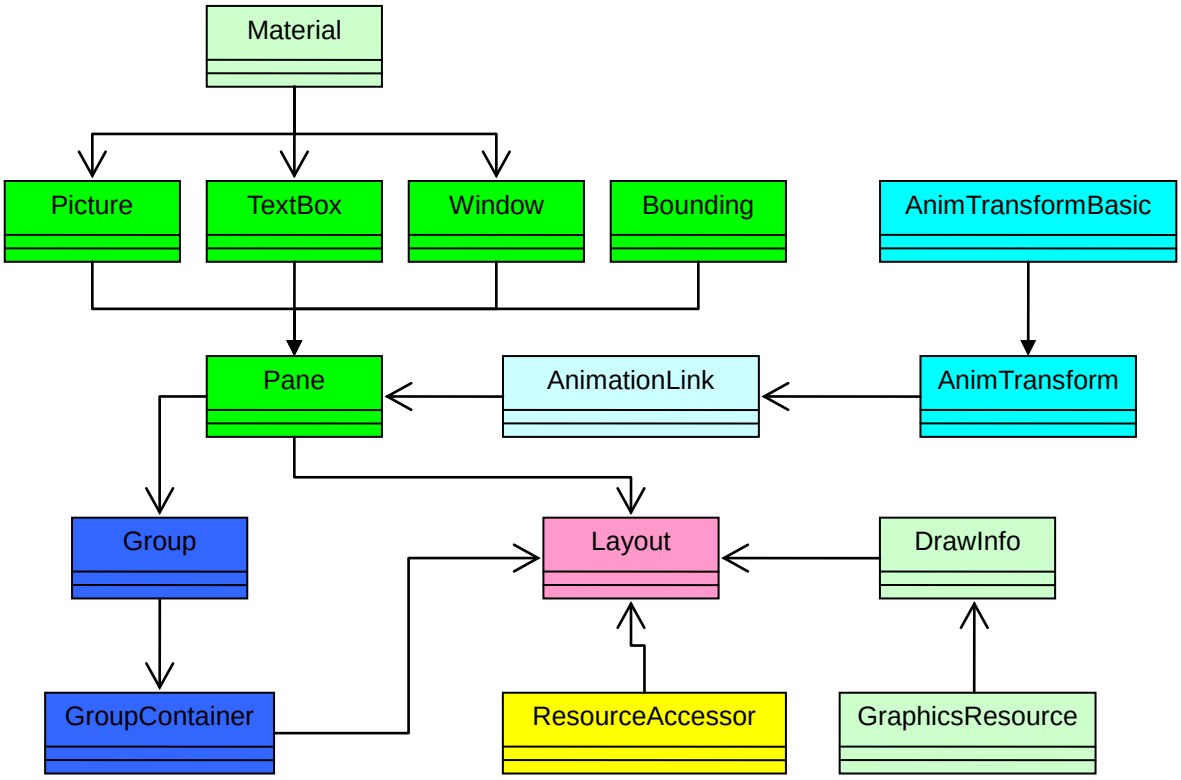
8 Structure of the lyt Library

This chapter looks at the structure of the lyt library.

8.1 Class Schematic

The schematic shows the structure of the `lyt` library classes.

Figure 7-1 Class Schematic



8.2 Descriptions of Classes

Table 8-1 presents an overview of the various classes.

Table 8-1 Description of Classes

Category	Class	Description
System	`Layout`	Class for managing the overall layout system.
Resources	`ResourceAccessor`	Abstract class for entering layout data.
	`ArcResourceAccessor`	Concrete class of the `ResourceAccessor` class for handling archived layout data.

Category	Class	Description
	MultiArcResourceAccessor	Concrete class of the ResourceAccessor class for handling multiple sets of archived layout data.
Group features	Group	Class for managing multiple panes.
	GroupContainer	Class for managing multiple groups in the layout data.
Panes	Pane	Base class for panes.
	Picture	Class for displaying pictures.
	TextBox	Class for displaying text.
	Window	Class for displaying windows.
	Bounding	Class used for hit checks.
Animations	AnimationLink	Class for linking panes and animations.
	AnimTransform	Base class for animations.
	AnimTransformBasic	Class for storing animation information.
Material	Material	Class for storing material information needed to display panes.
Drawing	Drawer	Class for drawing panes.
	DrawInfo	Class for storing the layout's drawing information.
	GraphicsResource	Class for maintaining global objects used when rendering.

Revision History

Version	Revision Date	Description
1.6	2011-02-28	• This document • Added Chapter 7 Loading Textures in VRAM.
1.5	2010/09/30	• Revised text to reflect changes in the library's API. • Updated the sample code to be current.
1.4	2010/07/29	• Revised text to reflect changes in the library's API. • Updated the description of the build method. • Updated the sample code to be current.
1.3	2010/06/28	• Revised text to reflect changes in the library's API. • Updated the sample code to be current.
1.2	2010/02/19	• Revised text to reflect changes in the library's API. • Revised sample code to be current. • Changed the format of the Revision History and moved it to the end of the document.
1.1	2009/12/22	• Revised sections that described "NintendoWare for Revolution."
1.0	2009/10/30	• Initial version.