

Nindows Tutorial

1997/12/10

VERSION : 1.03

目次

第 1章 Nindows 入門

- 1.1 概要
- 1.2 Nindows の特徴

第 2章 最も簡単な Nindows アプリケーションの作成

- 2.1 Nindows の組み込み
- 2.2 Nindows の組み込みに関する関数の説明

第 3章 Nindows の操作とNindows ユーティリティ

- 3.1 Nindows の操作
- 3.2 Nindows ユーティリティ
- 3.3 フォントの変更

第 4章 ウィンドウ

- 4.1 概要
- 4.2 ウィンドウの作成
- 4.3 子ウィンドウの作成
- 4.4 ウィンドウの作成に関する関数の説明
- 4.5 ウィンドウサポート関数の説明

第 5章 スクロールウィンドウ

- 5.1 概要
- 5.2 スクロールウィンドウの作成
- 5.3 スクロールウィンドウの作成に関する関数の説明

第 6章 エディットウィンドウ

- 6.1 概要
- 6.2 エディットウィンドウの作成
- 6.3 エディットウィンドウの作成に関する関数の説明
- 6.4 Nindows ユーティリティのデバッグウィンドウに関する関数の説明

第 7章 スクロールバーコントロール

- 7.1 概要
- 7.2 スクロールバーコントロールの作成
- 7.3 スクロールバーコントロールの作成に関する関数の説明
- 7.4 低レベルスクロールバー関数を利用したスクロールバーコントロールの作成
- 7.5 低レベルスクロールバー関数の説明

第 8章 ボタンコントロール

- 8.1 概要
- 8.2 ボタンコントロールの作成
- 8.3 ボタンの有効化と無効化
- 8.4 ボタンコントロールの作成に関する関数の説明

第 9章 メニュー

- 9.1 概要
- 9.2 メニューテーブルの作成とシステムメニューへの登録
- 9.3 メニューコールバック
- 9.4 チェックマーク

- 9.5 システムメニュー登録に関する関数の説明
- 9.6 ポップアップメニューの作成
- 9.7 ポップアップメニューの作成に関する関数の説明

第10章 マウス

- 10.1 概要
- 10.2 マウス情報の取得
- 10.3 マウス情報の取得に関する関数の説明

第11章 フォント

- 11.1 概要
- 11.2 フォントに関する関数の説明
- 11.3 フォントの変更に関する問題

第 1 章

Nindows 入門

1.1 概要

Nindows とは、ゲームを開発する上で欠かせないデバッグやパラメータの調整などを実機およびホストマシン上で行うための、簡易的に使用できる GUI システムです。

1.2 Nindows の特徴

- Windows などの一般的な GUI と同じコントロールが使用できます。

- Nindows API により、アプリケーションで自由にウィンドウを作成することができます。

- 複雑なプログラミングをすることなく、Texture Viewer 等のデバッグに便利なユーティリティが利用できます。

- Nindows で調整したパラメータ等はリアルタイムに確認できるため、ゲームバランス等の調整が迅速に行えます。

- 調整したパラメータを、ホストマシンではファイルに、実機ではバックアップメモリ等に保存できます。(本バージョンではサポートされていません)

第2章

最も簡単な Nindows アプリケーションの作成

ここでは、既存の Ninja アプリケーションに Nindows を組み込む方法を解説します。

ソースファイルにわずか数行の変更を加えるだけで、簡単に Nindows の機能が利用可能になります。

2.1 Nindows の組み込み

0 Ninja アプリケーションを準備する。

`njUserInit()`, `njUserMain()`, `njUserExit()` の各関数を持つソースファイルを準備します。

1 Nindows のヘッダファイルをインクルードする

ソースファイルに次の 1 行を追加します。

```
#include <Nindows.h>
```

2 Nindows の初期化関数をコールする

`njInitTexture()` の呼び出し後に、次の 1 行を追加します。

```
nwInitSystem(numTextures);
```

`numTextures` はテクスチャメモリリストの数です。

`njInitTexture()` で指定した値を指定します。

Nindows のフォント用に 3 枚のテクスチャを使用するので、アプリケーションで使用する数に 3 を足した値を指定してください。

3 Nindows の実行関数をコールする

`njUserMain` の最後の `return NJD_USER_CONTINUE` を、次のように変更します。

```
return nwExecute();
```

4 Nindows の終了関数をコールする

`njExitSystem()` の呼び出し前に、次の 1 行を追加します。

```
nwExitSystem();
```

5 Nindows のライブラリをリンクする

プロジェクトに `Nindows.lib` を加えます。

以上の作業により、

1. Nindows の Nindows ユーティリティの使用

2. Nindows API 関数の呼び出し

が可能になりました。

これらをまとめると、次のようなソースになります。

```
#include <NinjaWin.h>
#include <Nindows.h>

NJS_TEXMEMLIST tex[1000];

void njUserInit(void)
{
    njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1);
    njInitVertexBuffer(500000, 0, 500000, 0);
    njInitTexture(tex, 1000);
    nwInitSystem(1000);
}

Sint32 njUserMain(void)
{
    return nwExecute();
}

void njUserExit(void)
{
    nwExitSystem();
    njExitSystem();
}
```

2.2 Nindows の組み込みに関する関数の説明

| 関数 | 機能 |
|----------------|--------------------------|
| nwInitSystem | Nindows システムを初期化する |
| nwExitSystem | Nindows システムを終了する |
| nwExecute | 全てのウィンドウを描画する |
| nwInitResource | Nindows で使用するテクスチャをロードする |

表 2-1 Nindows の組み込みに関する関数の一覧

| nwInitSystem | | 初期化関数 |
|--------------|--|---------------|
| 書式 | void nwInitSystem (UInt32 numTextures) | |
| パラメータ | numTextures | テクスチャメモリリストの数 |
| 戻り値 | なし | |
| 機能 | <p>Nindows システムを初期化し、Nindows ユーティリティおよび Nindows API 関数を使用可能にします。</p> <p>numTextures には、njInitTexture() に指定した値と同じ値を指定してください。</p> | |
| 参照 | nwExecute(), nwExitSystem(), nwInitResource(), njInitTexture(), njCpZ() | |
| 備考 | <p>Nindows 内部で使用するテクスチャを自動的にロードします。</p> <p>Ninja 関数の nReleaseAllTexture() を使用した場合は、nwInitResource() を呼び出してテクスチャのリロードが必要です。</p> <p>Nindows は、テクスチャのグローバルインデックス番号の 0xffffffff0 ~ 0xfffffffffe を予約します。アプリケーションでこの範囲のグローバルインデックス番号を持つテクスチャを使用しないでください。</p> | |

例

```
#define MAX_TEXTURE 1000

static NJS_TEXMEMLIST texlist[MAX_TEXTURE];

void njUserInit(void)
{
    njInitSystem(NJD_RESOLUTION_VGA,    NJD_FRAMEBUFFER_MODE_RGB555,
1);
    njInitVertexBuffer(500000, 0, 500000, 0);
    njInitTexture(texlist, MAX_TEXTURE);
    nwInitSystem(MAX_TEXTURE);
}
```

nwExitSystem

初期化関数

| | |
|-------|---|
| 書式 | void nwExitSystem(void) |
| パラメータ | なし |
| 戻り値 | なし |
| 機能 | Windows を終了します。 |
| 参照 | nwExitSystem() |
| 備考 | |
| 例 | <pre>void njUserExit(void) { nwExitSystem(); nwExitSystem(); }</pre> |

nwExecute

実行関数

| | |
|-------|--|
| 書式 | Sint32 nwExecute(void) |
| パラメータ | なし |
| 戻り値 | システムメニューの「Exit」が選択された場合は NJD_USER_EXIT、その他の場合は NJD_USER_CONTINUE を返します。 |
| 機能 | Windows のすべての描画を行います |
| 参照 | njUserMain() |
| 備考 | ・1フレームに 1 回呼び出してください。 Windows システムはこの関数呼び出しによってすべてのウィンドウの描画を行います。 例のように、この関数の戻り値をそのまま njUserMain() の戻り値とすることにより「Exit」メニューの選択でアプリケーションを終了することができます。 |
| 例 | <pre>Sint32 njUserMain(void) { : : : return nwExecute(); }</pre> |

nwInitResource

初期化関数

| | |
|-------|---------------------------|
| 書式 | void nwInitResource(void) |
| パラメータ | なし |
| 戻り値 | なし |

| | |
|----|--|
| 機能 | N dows で使用するテクスチャをロードします |
| 参照 | <code>nwInitSystem()</code> , <code>njhitTexture()</code> , <code>nReleaseTexture()</code> |
| 備考 | 通常は使用する必要はありませんが、例のように <code>nReleaseTextureAll()</code> を使用した場合、N dows の使用するテクスチャも開放されてしまいます。よって、 <code>nReleaseTextureAll()</code> の呼び出し後には必ずこの関数を呼び出してください。 |
| 例 | <pre> njReleaseTextureAll(); nwInitResource(); </pre> |

第 3 章

Nindows の操作とNindows ユーティリティ

3.1 Nindows の操作

第 2 章で Nindows の組み込みが終了しました。Nindows を組み込んだアプリケーション (以下 Nindows アプリ)を実行すると、画面上にマウスカーソルが表示され、マウスの操作に応じて画面上を動きます。

システムメニュー

デスクトップ上でマウスの右ボタンをクリックすると、ポップアップメニューが表示されます。これをシステムメニューと呼び、ここからメニューを選択して Nindows の持つ Nindows ユーティリティを使用します。システムメニューには以下の項目があります。

| メニュー項目 | 説明 |
|-----------------|--|
| Debug | Nindows ユーティリティメニューをポップアップします。 |
| User(Undefined) | ユーザー定義のメニューをポップアップします。Nindows 初期化時には、ユーザーメニューは登録されていないため、淡色表示となり選択できません。 |
| Font | フォントを変更します。 |
| Exit | アプリケーションを終了します。 |

表 3-1 システムメニューのメニュー項目一覧

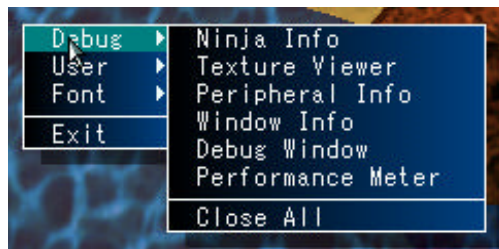


図 3-1 右ボタンでシステムメニューを表示し、「Debug」を選択した状態

3.2 Nindows ユーティリティ

システムメニューから「Debug」を選択した時に表示されるメニュー項目が、Nindows の持つ Nindows ユーティリティです。Nindows ユーティリティには以下のものがあります。

| 名称 | 説明 |
|-------------------|-----------------------------------|
| Ninja Info | Ninja ライブラリのバージョン番号、その他情報が表示されます。 |
| Texture Viewer | 読み込まれているテクスチャを全て表示することができます。 |
| Peripheral Info | ペリフェラルの情報が表示されます。 |
| Window Info | アクティブなウィンドウの情報が表示されます。 |
| Debug Window | デバッグ文字列を表示するのに便利なウィンドウです。 |
| Performance Meter | アプリケーションの描画パフォーマンスがわかります。 |

表 3-2 Nindows ユーティティ一覧

3.2.1 Ninja Info ウィンドウ



図 3-2 Ninja Info ウィンドウ

Ninja Info ウィンドウには以下の情報が表示されます。

| 表示 | 内容 |
|----------------|---------------------|
| Ninja Ver. | Ninja ライブラリのバージョン番号 |
| Nindows Ver. | Nindows のバージョン番号 |
| Vertex | 投入頂点数 |
| Calc polygon | 投入ポリゴン数 |
| Draw polygon | 描画ポリゴン数 |
| Texture Memory | テクスチャメモリの空き容量と全容量 |

表 3-3 Ninja Info ウィンドウの情報

3.2.2 Texture Viewer ウィンドウ

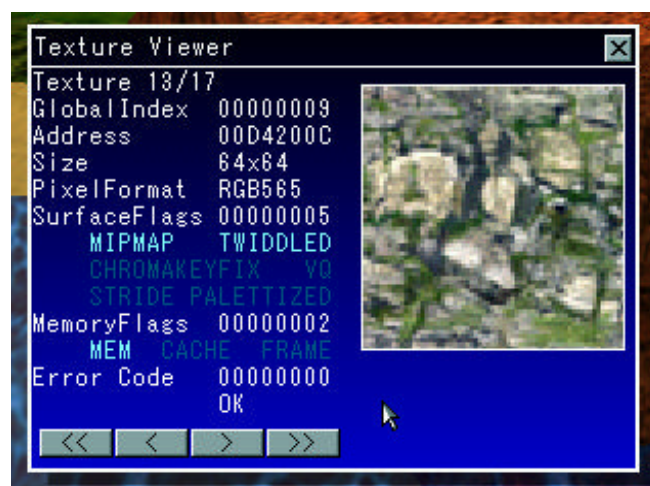


図 3-3 Texture Viewer ウィンドウ

登録されているテクスチャをすべて見ることができます。

4つのボタンでテクスチャを切り替えることができます。

Texture Viewer ウィンドウには以下の情報が表示されます。

| 表示 | 内容 |
|-------------|-----------------|
| Texture | テクスチャ番号と総テクスチャ数 |
| GlobalIndex | グローバルインデックス |

| | |
|--------------|--|
| Address | テクスチャアドレス |
| Size | テクスチャサイズ |
| PikeFormat | ピクセルフォーマット |
| SurfaceFlags | サーフェスフラグ ハイライト表示されている項目が、そのテクスチャに設定されているフラグです。フラグの詳細はテクスチャ関連ドキュメントを参照してください。 |
| MemoryFlags | メモリフラグ 同じく、そのテクスチャに設定されているフラグです。フラグの詳細はテクスチャ関連ドキュメントを参照してください。 |
| Error Code | テクスチャをロードする際に発生したエラーコードです。ロードに成功した場合は「0K」が表示されます。エラーコードの詳細はテクスチャ関連ドキュメントを参照してください。 |

表 3-4.Texture Viewer ウィンドウの情報

3.2.3 Peripheral Info ウィンドウ



図 3-4 Peripheral Info ウィンドウ

ペリフェラル (入力装置) の情報を表示するウィンドウです。
 「<」、「>」ボタンで表示するペリフェラルポートを選択できます。
 Peripheral Info ウィンドウには以下の情報が表示されます。

| 表示 | 内容 |
|---------|--------------------|
| Port | ペリフェラルポート名 |
| Dev | ポートに接続されているペリフェラル名 |
| ON | 押されているボタン情報 |
| OFF | 押されていないボタン情報 |
| PRESS | 押された瞬間のボタン情報 |
| RELEASE | 放された瞬間のボタン情報 |
| X | X 軸値 |
| Y | Y 軸値 |

表 3-5 Peripheral Info ウィンドウの情報

3.2.4 Window Info ウィンドウ



図 3-4 Peripheral Info ウィンドウ

マウスカーソルの下にあるウィンドウ (アクティブウィンドウ) の情報が表示されます。
Nindows API を使用してアプリケーション独自のウィンドウを作成する場合などのデバッグに利用できます。

Peripheral Info ウィンドウには以下の情報が表示されます。

| 表示 | 内容 |
|-----|---------------------|
| | アクティブウィンドウのタイトル文字列 |
| x,y | ウィンドウのクライアント領域の左上座標 |
| w,h | ウィンドウのクライアント領域のサイズ |

表 3-5 Peripheral Info ウィンドウの情報

x, y 座標は必ずしも画面上の絶対座標ではなく、ウィンドウスタイルに依存します。

3.2.5 デバッグ用ウィンドウ



図 3-5 Debug 用ウィンドウ

このウィンドウには、最初は何も表示されません。
nwDebugPrintf()関数を用いて、デバッグ文字を表示してください。
この関数は 標準の printf()関数と同様に使用できます。
詳細はエディットウィンドウの章を参照してください。

3.2.6 Performance Meter ウィンドウ



図 3-6 Performance Meter ウィンドウ

アプリケーションのパフォーマンス (計算、描画速度) が直感的にわかります。
メーターが 1秒で 1周すれば、フレームレートは 60fps です。

3.3 フォントの変更

システムメニューから「Font」を選択することで、通常サイズと大きいサイズのフォントを選択することができます。NTSC モニタに出力した場合など、通常サイズでは文字が読みにくい場合には大きいサイズのフォントを選択してください。

本バージョンでは、フォントサイズ変更に伴うウィンドウのリサイズには対応していません。フォントを変更する場合、なるべくアプリケーション実行後、他のウィンドウを開く前にフォントを変更するようにしてください。

第4章 ウィンドウ

4.1 概要

ウィンドウは、Windows の最も基本的な要素です。アプリケーションは、まずウィンドウを作成し、描画コールバック関数を指定することにより、ウィンドウのクライアント領域への描画を行います。また、子ウィンドウとして別のウィンドウや、ボタン、スクロールバーなどのコントロールを作成し、制御することができます。

4.1.1 ウィンドウの種類とウィンドウクラス

ウィンドウには次の種類があり、ウィンドウクラスとして区別します。

| ウィンドウクラス | ウィンドウの種類 |
|------------------|------------------------|
| NWD_WC_WIN | 通常のウィンドウ |
| NWD_WC_SCROLLW | クライアント領域がスクロール可能なウィンドウ |
| NWD_WC_EDITW | エディットウィンドウ |
| NWD_WC_SCROLLBAR | スクロールバーコントロール |
| NWD_WC_BUTTON | ボタンコントロール |
| NWD_WC_MENUW | メニューウィンドウ |

表 4-1.ウィンドウの種類とウィンドウクラス

ここでは、主に通常のウィンドウについて解説します。

4.2 ウィンドウの作成

例として、クライアント領域にカウンタを表示するウィンドウを、デスクトップに作成します。ウィンドウの作成には、関数 `nwCreateWindow()` を使用します。

```
static void test_window_callback(NWHWND hWnd)
{
    static int count = 0;
    char buf[256];
    sprintf(buf, "Count = %d", count++);
    nwTextOut(hWnd, 1, 1, buf);
}

Bool create_test_window(void)
{
    NWHWND hWnd = nwCreateWindow(NWD_WC_WIN,
                                "Test Window",
                                NWD_WS_CAPTION | NWD_WS_BORDER |
                                NWD_WS_SHADOW,
                                50, 50, 100, 100,
                                NULL);

    if (!hWnd) return FALSE;

    hWnd->clientDraw = test_window_callback;
    return TRUE;
}

void njUserInit(void)
{
    :
    nwInitSystem(...);
    create_test_window();
}
```

これで、画面の指定位置に次のようなウィンドウが表示され、カウントの表示がインクリメントされていくはずです。なお、ウィンドウが表示されるのは `nwExecute()` の呼び出し、および `njsrMain()` からのリターン後、次のフレームからになります。

また、作成したウィンドウを破棄するには、関数 `nwDestroyWindow()` を呼び出すか、あるいはマウスを操作して、キャプションバーにあるクローズボックスをクリックします。(クローズボックスはウィンドウスタイルに `NWD_WS_CONTROL` を指定したウィンドウのみが持つことができます。)

```
nwDestroyWindow(hWnd);
```

4.3 子ウィンドウの作成

`nwCreateWindow()` の最後の引数は、親ウィンドウのウィンドウハンドルです。先程の例では、この引数に `NULL` を指定したため、親を持たない (正確にはデスクトップウィンドウを親に持つ) ウィンドウを作成しました。ここでは、親ウィンドウと、その子ウィンドウを作成する例を示します。

```
static void test_window_callback(NWHWND hWnd)
{
    static int count = 0;
    char buf[256];
    sprintf(buf, "Count = %d", count++);
    nwTextOut(hWnd, 1, 1, buf);
}

Bool create_test_window(void)
{
    NWHWND hWndParent, hWnd;
    hWndParent = nwCreateWindow(NWD_WC_WIN,
                                "Parent Window",
                                NWD_WS_CAPTION | NWD_WS_BORDER |
NWD_WS_SHADOW,
                                50, 50, 100, 100,
                                NULL);
    if (!hWndParent) return FALSE;

    hWndParent->clientDraw = test_window_callback;

    //以下は子ウィンドウの作成
    hWnd = nwCreateWindow(NWD_WC_WIN,
                           "Child Window",
                           NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_OFFSET
20, 20, 60, 60,
                           hWndParent);
    if (!hWnd) return FALSE;

    return TRUE;
}
```

親ウィンドウを、関数 `nwDestroyWindow()` (またはマウスの操作などにより) 破棄すると、そのウィンドウを親に持つ全ての子ウィンドウも自動的に破棄されます。この例では親ウィンドウ `hWndParent` を破棄すれば、子ウィンドウ `hWnd` も破棄されます。

```
nwDestroyWindow(hWndParent);
```

なお、スクロールバー(クラス `NW D_W C_S C R B A R`)、ボタン(クラス `NW D_W C_B U T T O N`)、およびメニュー(クラス `NW D_W C_M E N U W I N`)は、親ウィンドウとして指定することはできません(子ウィンドウを持つことはできません)。ただし、メニューの場合は子メニュー(サブメニュー)を持つことは可能です。詳細は第9章「メニュー」を参照してください。

4.4 ウィンドウに関連するパラメータ

ウィンドウハンドル `NW H W N D` は、実際には構造体 `NW S_W I N` へのポインタです。

作成したウィンドウのウィンドウハンドルを介してこの構造体のメンバを直接設定することにより、ウィンドウにさまざまな動作をさせることができます。ここでは、知っておくと便利なメンバとその代表的な利用方法について解説します。

・クライアント描画コールバック関数 `hW nd->clientD raw`

ウィンドウ作成例でも使用した、最も代表的なメンバです。

通常、このメンバに何等かのコールバック関数アドレスを設定し、そのコールバック関数の中でクライアント領域への描画等の処理を行います。

・デストラクタ `hW nd->destructor`

このメンバに関数アドレスを設定しておくと、ウィンドウが破棄される時にコールバックされます。

・ユーザーデータ `hW nd->param 1, hW nd->param 2`

アプリケーションで自由に設定、参照できる、`Sint32` 型のメンバです。

・ユーザーデータ `hW nd->userBuf`

よ多くのユーザーデータを格納したい場合、そのデータのアドレスを指定します。

データバッファは、アプリケーションで独自に確保してください。

4.5 ウィンドウ作成に関する関数の説明

| 関数 | 機能 |
|--------------------------------|------------|
| <code>nwCreateW indow</code> | ウィンドウを作成する |
| <code>nwDestroy W indow</code> | ウィンドウを破棄する |

表 4-1.ウィンドウ作成に関する関数の一覧

Windows API

nwCreateWindow

ウィンドウ作成関数

| | | |
|-------|--|-------------------|
| 書式 | <code>NW H W N D nwC reateW indow (S int32 wC lass, S int8* captio n, S int32 sty le, S int32 x, S int32 y, S int32 w, S int32 h, NW H W N D hW ndParent)</code> | |
| パラメータ | <code>wC lass</code> | ウィンドウクラス |
| | <code>captio n</code> | ウィンドウ名(キャプション)文字列 |
| | <code>sty le</code> | ウィンドウスタイル |
| | <code>x,y</code> | クライアント領域の左上座標 |
| | <code>w,h</code> | クライアント領域の幅と高さ |
| | <code>hW ndParent</code> | 親ウィンドウハンドル |

| | |
|------|---|
| 戻り値 | 成功すれば、作成されたウィンドウのハンドルが、ウィンドウを作成できなかった場合は NULL が返されます。 |
| 機能参照 | ・ウィンドウを作成します。 nwDestroyWindow(), nwCreateMenuWindow(), nwCreateEditWindow(), nwCreateScrollBar(), nwCreateButton() |
| 備考 | NWS_WINDOW 構造体 ・メニューウィンドウ、エディットウィンドウ、スクロールバー、ボタンを作成する場合には、より便利な nwCreateMenuWindow(), nwCreateEditWindow(), nwCreateScrollBarArray(), nwCreateButton() を使用することをおすすめします。 |
| 例 | // ウィンドウを作成します NHHWND hWnd; hWnd = nwCreateWindow(NWD_WC_WIN, "Test Window", NWD_WS_CAPTION NWD_WS_BORDER NWD_WS_SHADOW, 50, 50, 100, 100, NULL); |

ウィンドウスタイル style には、以下のフラグを組み合わせで指定します。

| ウィンドウスタイル | 意味 |
|-------------------|-----------------------------|
| NWD_WS_CAPTION | キャプションあり |
| NWD_WS_BORDER | 細い境界線を持つ |
| NWD_WS_THICKFRAME | サイズ変更可能な太い境界線を持つ |
| NWD_WS_SHADING | ウィンドウの色を頂点ごとに指定できる |
| NWD_WS_CONTROL | クローズボックスを持つ |
| NWD_WS_SHADOW | 影付きウィンドウ |
| NWD_WS_INVISIBLE | 見えないウィンドウを作成 |
| NWD_WS_NOMOVE | マウスで移動できない |
| NWD_WS_OFFSET | 親ウィンドウからの相対位置(x,y)にウィンドウを作成 |

また、ウィンドウクラスに NWD_WC_SCROLLBAR を指定した場合、以下のフラグのどちらかを必ず指定してください。

| ウィンドウスタイル | 意味 |
|----------------|----------------|
| NWD_WS_SB_HORZ | 水平スクロールバーを作成する |
| NWD_WS_SB_VERT | 垂直スクロールバーを作成する |

nwDestroyWindow

ウィンドウ作成関数

| | |
|-------|--|
| 書式 | void nwDestroyWindow(NHHWND hWnd) |
| パラメータ | hWnd 破棄したいウィンドウのウィンドウハンドル |
| 戻り値 | なし |
| 機能 | ・ウィンドウを破棄します。 |
| 参照 | nwCreateWindow(), NWS_WINDOW 構造体 |
| 備考 | hWnd->destructor にコールバック関数が設定されている場合、その関数をコールバックします。 |
| 例 | // ウィンドウを破棄します nwDestroyWindow(hWnd); |

・コールバック関数

ClientDrawCallback

ウィンドウコールバック関数

| | |
|-------|--------------------------------------|
| 書式 | void ClientDrawCallback(NHHWND hWnd) |
| パラメータ | hWnd コールバック元のウィンドウハンドル |
| 戻り値 | なし |

| | |
|----|--|
| 機能 | ・ウィンドウが描画のためにコールバックする、アプリケーション定義の関数です。 |
| 参照 | nwCreateWindow(), NWS_WN 構造体 |
| 備考 | |

DestroyCallback ウィンドウコールバック関数

| | |
|-------|---|
| 書式 | void DestroyCallback(NHWND hwnd) |
| パラメータ | hwnd コールバック元のウィンドウハンドル |
| 戻り値 | なし |
| 機能 | ・ウィンドウが破棄されるときにコールバックする、アプリケーション定義の関数です。 |
| 参照 | nwDestroyWindow(), NWS_WN 構造体 |
| 備考 | |

4.4 ウィンドウサポート関数のサンプルと説明

NWindows API には、上記の nwCreateWindow()と nwDestroyWindow()以外にも、ウィンドウの管理をサポートする関数が数多く用意されています。以下は、ジョイスティックを使ってウィンドウを動かすサンプルです。

```
Sint32 njUserMain(void)
{
    Sint32 x, y, w, h, style;
    NHWND hwnd;
    NJS_PERIPHERAL* per = njGetPeripheral(NJD_PORT_JOYSTICK1);

    // ウィンドウが存在するかどうか調べる
    if (!(hwnd = nwFindWindow(NULL, "Test Window"))) goto skip;

    // ウィンドウのクライアント領域の座標とサイズを取得する
    nwGetWindowPos(hwnd, &x, &y);
    nwGetWindowSize(hwnd, &w, &h);
    if (per->press & NJD_DGT_ST) {
        // スタートボタンを押すと ウィンドウを画面の中心に移動する。
        nwSetWindowPos(hwnd, 320 - w / 2, 240 - h / 2);
    }
    if (per->on & NJD_DGT_TA) {
        // A ボタンを押しているとき 十字キーでウィンドウを移動できる。
        if (per->on & NJD_DGT_KU) y--;
        if (per->on & NJD_DGT_KD) y++;
        if (per->on & NJD_DGT_KL) x--;
        if (per->on & NJD_DGT_KR) x++;
        nwSetWindowPos(hwnd, x, y);
    }
    if (per->on & NJD_DGT_TB) {
        // B ボタンを押しているとき 十字キーでウィンドウのサイズを変更できる。
        if (per->on & NJD_DGT_KU) h--;
        if (per->on & NJD_DGT_KD) h++;
        if (per->on & NJD_DGT_KL) w--;
        if (per->on & NJD_DGT_KR) w++;
        nwSetWindowSize(hwnd, w, h);
    }
    if (per->press & NJD_DGT_TC) {
        // C ボタンを押すと ウィンドウのキャプションバーの ON/OFF を切り替える。
        nwGetWindowStyle(hwnd, &style);
        if (style & NWD_WS_CAPTION) {
            nwSetWindowStyle(hwnd, ~NWD_WS_CAPTION, 0);
        } else {
            nwSetWindowStyle(hwnd, 0xffffffff, NWD_WS_CAPTION);
        }
    }
}

skip:
return nwExecute();
}
```

| 関数 | 機能 |
|--------------------------------|---------------------------------|
| <code>nwFindWindow</code> | 指定のキャプション文字列を持つウィンドウを検索する |
| <code>nwFindWindowByPos</code> | 指定位置にあるウィンドウを検索する |
| <code>nwGetClientRect</code> | 指定ウィンドウのクライアント領域の矩形を取得する |
| <code>nwGetWindowColor</code> | 指定ウィンドウのカラーを取得する |
| <code>nwGetWindowPos</code> | 指定ウィンドウのクライアント領域の左上座標を取得する |
| <code>nwGetWindowRect</code> | 指定ウィンドウのウィンドウ全体の矩形を取得する |
| <code>nwGetWindowSize</code> | 指定ウィンドウのクライアント領域の幅と高さを取得する |
| <code>nwGetWindowStyle</code> | 指定ウィンドウのウィンドウスタイルを取得する |
| <code>nwGetWindowText</code> | 指定ウィンドウのキャプション文字列を取得する |
| <code>nwSetWindowColor</code> | 指定ウィンドウのカラーを変更する |
| <code>nwSetWindowPos</code> | 指定ウィンドウを、クライアント領域の左上座標を指定して移動する |
| <code>nwSetWindowSize</code> | 指定ウィンドウのクライアント領域の幅と高さを変更する |
| <code>nwSetWindowStyle</code> | 指定ウィンドウのウィンドウスタイルを変更する |
| <code>nwSetWindowText</code> | 指定ウィンドウのキャプション文字列を変更する |

表 4-1. ウィンドウ作成に関する関数の一覧

Windows API

nwFindWindow

ウィンドウサポート関数

| | |
|-------|--|
| 書式 | <code>NHWND nwFindWindow(NHWND hwnd, Sint8* caption)</code> |
| パラメータ | <code>hwnd</code> ウィンドウを探し始める親ウィンドウ <code>caption</code> 探したいウィンドウのキャプション文字列 |
| 戻り値 | ウィンドウが見つければそのウィンドウのウィンドウハンドルを、見つからなければ <code>NULL</code> を返します。 |
| 機能 | 指定のキャプション文字列を持つウィンドウを、指定された親ウィンドウの子ウィンドウの中から探します。 すべてのウィンドウを検索の対象としたい場合は、親ウィンドウに <code>NULL</code> を指定してください。 |
| 参照 | <code>nwFindWindowByPos()</code> |
| 備考 | |
| 例 | <pre>// ウィンドウ「Material Window」を、すべてのウィンドウの中から探します。 hwnd = nwFindWindow(NULL, "Material Window");</pre> |

nwFindWindowByPos

ウィンドウサポート関数

| | |
|-------|--|
| 書式 | <code>NHWND nwFindWindowByPos(Sint16 x, Sint16 y)</code> |
| パラメータ | <code>x, y</code> スクリーン座標 |
| 戻り値 | ウィンドウが見つければそのウィンドウのウィンドウハンドルを、見つからなければ <code>NULL</code> を返します。 |
| 機能 | 指定のスクリーン座標 <code>(x, y)</code> に、ウィンドウが表示されているかどうかを調べます。 |
| 参照 | <code>nwFindWindow</code> |
| 備考 | |
| 例 | <pre>// マウスカーソルの座標に、ウィンドウが表示されているかどうか調べます。 NJS_PERIPHERAL* mouse = njGetPeripheral(NJD_PORT_SYSMOUSE); if (nwFindWindowByPos(mouse->x, mouse->y) { // ウィンドウが表示されている } else { // ウィンドウは表示されていない }</pre> |

nwGetClientRect

ウィンドウサポート関数

| | |
|-------|---|
| 書式 | <code>Bool nwGetClientRect(NHWND hwnd, NWS_RECT* rect)</code> |
| パラメータ | <code>hwnd</code> ウィンドウハンドル |

| | | |
|-----|---|---------------|
| 戻り値 | rect | 矩形情報を格納するアドレス |
| 機能 | 成功すれば TRUE、失敗すれば FALSE を返します。 | |
| 参照 | ウインドウのクライアント領域の矩形を取得します。 | |
| 備考 | ウインドウスタイルの <code>NWD_WS_OFFSET</code> フラグに関係なく、取得した矩形は画面上の絶対座標です。 | |
| 例 | <pre>NWS_RECT rect; nwGetClientRect(hWnd, &rect);</pre> | |

| nwGetWindowColor | | ウインドウサポート関数 |
|------------------|---|--|
| 書式 | <code>BoolInnwGetWindowColor(NWHWND hWnd, NWS_RGBA col[4])</code> | |
| パラメータ | hWnd | ウインドウハンドル |
| | col | カラーを取得する <code>NWS_RGBA</code> 構造体の配列のアドレス |
| 戻り値 | 成功すれば TRUE、失敗すれば FALSE を返します。 | |
| 機能 | ウインドウの色を取得します。 | |
| | col[0]から順番左上頂点、右上頂点、右下頂点、左下頂点の色が格納されます。 | |
| 参照 | | |
| 備考 | | |
| 例 | <pre>NWS_RGBA col[4]; nwGetWindowColor(hWnd, col);</pre> | |

| nwGetWindowPos | | ウインドウサポート関数 |
|----------------|---|-------------|
| 書式 | <code>BoolInnwGetWindowPos(NWHWND hWnd, Sint32* x, Sint32* y)</code> | |
| パラメータ | hWnd | ウインドウハンドル |
| | x, y | 座標を格納するアドレス |
| 戻り値 | 成功すれば TRUE、失敗すれば FALSE を返します。 | |
| 機能 | ウインドウのクライアント領域の左上座標を取得します。 | |
| 参照 | | |
| 備考 | ウインドウスタイルに <code>NWD_WS_OFFSET</code> が指定されている場合、座標は親ウインドウからの相対座標となります。 | |
| 例 | <pre>Sint32 x, y; nwGetWindowPos(hWnd, &x, &y);</pre> | |

| nwGetWindowRect | | ウインドウサポート関数 |
|-----------------|---|---------------|
| 書式 | <code>BoolInnwGetWindowRect(NWHWND hWnd, NWS_RECT* rect)</code> | |
| パラメータ | hWnd | ウインドウハンドル |
| | rect | 矩形情報を格納するアドレス |
| 戻り値 | 成功すれば TRUE、失敗すれば FALSE を返します。 | |
| 機能 | キャプションや境界線を含むウインドウ全体の矩形を取得します。 | |
| 参照 | | |
| 備考 | ウインドウスタイルの <code>NWD_WS_OFFSET</code> フラグに関係なく、取得した矩形は画面上の絶対座標です。 | |
| 例 | <pre>NWS_RECT rect; nwGetWindowRect(hWnd, &rect);</pre> | |

| nwGetWindowSize | | ウインドウサポート関数 |
|-----------------|---|---------------|
| 書式 | <code>BoolInnwGetWindowSize(NWHWND hWnd, Sint32* w, Sint32* h)</code> | |
| パラメータ | hWnd | ウインドウハンドル |
| | w, h | 幅と高さを格納するアドレス |

戻り値 成功すれば TRUE、失敗すれば FALSE を返します。
 機能 ウィンドウのクライアント領域の幅と高さを取得します。

参照
 備考
 例

```
Sint32 width, height;
nwGetWindowSize(hWnd, &width, &height);
```

nwGetWindowStyle

ウィンドウサポート関数

書式 Bool nwGetWindowStyle(NHHWND hWnd, Sint32* style)

パラメータ hWnd ウィンドウハンドル
 style スタイルを取得するアドレス

戻り値 成功すれば TRUE、失敗すれば FALSE を返します。
 機能 ウィンドウスタイルを取得します。

参照
 備考
 例

```
Sint32 style;
nwGetWindowStyle(hWnd, &style);
if (style & NWD_WS_SHADOW) {
    // もし影付きウィンドウだったら
}
```

nwGetWindowText

ウィンドウサポート関数

書式 Bool nwGetWindowText(NHHWND hWnd, Sint8* caption, Sint32 size)

パラメータ hWnd ウィンドウハンドル
 caption ウィンドウキャプション文字列を格納するバッファアドレス
 size バッファサイズ

戻り値 成功すれば TRUE、失敗すれば FALSE を返します。
 機能 ウィンドウのキャプションバーに表示されるキャプション文字列を取得し、指定のバッファにコピーします。

参照
 備考
 例

```
// ウィンドウ hWnd のキャプション文字列を buf に取得します
Sint8 buf[256];
nwGetWindowText(hWnd, buf, sizeof(buf));
```

nwSetWindowColor

ウィンドウサポート関数

書式 Bool nwSetWindowColor(NHHWND hWnd, NWS_RGBA col[4])

パラメータ hWnd ウィンドウハンドル
 col ウィンドウの各頂点の色を格納した配列アドレス

戻り値 成功すれば TRUE、失敗すれば FALSE を返します。
 機能 ウィンドウの色を変更します。
 col[0]から順番左上頂点、右上頂点、右下頂点、左下頂点の色を指定してください。

参照
 備考

ウィンドウスタイルに NWD_WS_SHADOW が指定されていない場合、左上頂点の色が全頂点に適用されます。

例

```
NWS_RGBA col[4] = {
    {255, 0, 0, 255}, // 左上頂点の色
    { 0, 255, 0, 255}, // 右上頂点の色
    { 0, 0, 255, 255}, // 右下頂点の色
    { 0, 0, 0, 255}, // 左下頂点の色
};
nwSetWindowColor(hWnd, col);
```

| nwSetWindowPos | | ウィンドウサポート関数 |
|----------------|--|---------------|
| 書式 | Bool nwSetWindowPos(HWND hwnd, Sint32 x, Sint32 y) | |
| パラメータ | hwnd | ウィンドウハンドル |
| | x, y | クライアント領域の左上座標 |
| 戻り値 | 成功すれば TRUE、失敗すれば FALSE を返します。 | |
| 機能 | ウィンドウの表示座標を変更します。 座標(x, y) がクライアント領域の左上座標となるよう ウィンドウが移動します。 ウィンドウスタイルに NWD_WS_OFFSET が指定されている場合、座標は親ウィンドウからの相対座標となります。 | |
| 参照 | | |
| 備考 | | |
| 例 | <pre>// ウィンドウを座標(100, 50)に移動します。 nwSetWindowPos(hwnd, 100, 50);</pre> | |

| nwSetWindowSize | | ウィンドウサポート関数 |
|-----------------|--|---------------|
| 書式 | Bool nwSetWindowSize(HWND hwnd, Sint32 w, Sint32 h) | |
| パラメータ | hwnd | ウィンドウハンドル |
| | w, h | クライアント領域の幅と高さ |
| 戻り値 | 成功すれば TRUE、失敗すれば FALSE を返します。 | |
| 機能 | ウィンドウの大きさを変更します。 (w, h)がクライアント領域の幅と高さになります。 | |
| 参照 | | |
| 備考 | | |
| 例 | <pre>// ウィンドウのクライアント領域の幅と高さを(128, 64)に変更します。 nwSetWindowSize(hwnd, 128, 64);</pre> | |

| nwSetWindowStyle | | ウィンドウサポート関数 |
|------------------|---|-------------|
| 書式 | Bool nwSetWindowStyle(HWND hwnd, Sint32 and_style, Sint32 or_style) | |
| パラメータ | hwnd | ウィンドウハンドル |
| | and_style | アンド・スタイル |
| | or_style | オア・スタイル |
| 戻り値 | 成功すれば TRUE、失敗すれば FALSE を返します。 | |
| 機能 | ・ウィンドウスタイルを変更します。 | |
| 参照 | | |
| 備考 | ・ウィンドウクラスや、他のパラメータと矛盾する設定をした場合の動作は保証しません。 ・アンド・スタイルには、例のように"~"を付けることを忘れないようにしてください。 ・ウィンドウ作成時に NWD_WS_CONTROL フラグを指定しなかったウィンドウに対して、この関数で NWD_WS_CONTROL フラグを設定した場合の動作は保証しません。 | |
| 例 | <pre>// ウィンドウから影を外し、キャプションを付けます。 nwSetWindowStyle(hwnd, ~NWD_WS_SHADOW, NWD_WS_CAPTION);</pre> | |

| nwSetWindowText | | ウィンドウサポート関数 |
|-----------------|---|------------------------|
| 書式 | Bool nwSetWindowText(HWND hwnd, Sint8* caption) | |
| パラメータ | hwnd | ウィンドウハンドル |
| | caption | NULL 終端キャプション文字列へのポインタ |
| 戻り値 | 成功すれば TRUE、失敗すれば FALSE を返します。 | |
| 機能 | ウィンドウのキャプションバーに表示されるキャプションを変更します。 | |

参照
備考
例

```
// ウィンドウのキャプションを"New Caption"に変更します。  
nwSetWindowText(hWnd, "New Caption");
```

構造体

NWS_WIN

構造体

定義

```
typedef struct _NWS_WIN {  
    Sint32 style;  
    Sint32 wClass;  
    Sint8 *caption;  
    Sint32 font;  
    struct _NWS_WIN *parent;  
    struct _NWS_WIN *child;  
    struct _NWS_WIN *before;  
    struct _NWS_WIN *next;  
    Sint32 x, y;  
    Sint32 w, h;  
    NWS_RGBA col[4];  
    NWS_MSGHANDLE *msgHandle;  
    void *menuTable;  
    void *userBuf;  
    void (*clientDraw)(struct _NWS_WIN *NWFUNC);  
    void (*execFunc)(struct _NWS_WIN *NWFUNC);  
    void (*destructor)(struct _NWS_WIN* NWFUNC);  
    Sint32 param1, param2;  
    struct _NWS_WIN* hClose;  
    struct _NWS_WIN* hMaximize;  
    struct _NWS_WIN* hMinimize;  
} NWS_WIN;
```

メンバ

| | |
|------------------|----------------------|
| style | ウィンドウスタイル |
| wClass | ウィンドウクラス |
| caption | キャプション文字列 |
| font | フォント種別 |
| parent | 親ウィンドウハンドル |
| child | 子ウィンドウハンドル |
| before | 前のウィンドウハンドル |
| next | 次のウィンドウハンドル |
| x, y | クライアント領域の左上座標 |
| w, h | クライアント領域の幅と高さ |
| col | 4頂点の色 |
| msgHandle | 未使用 (予約) |
| menuTable | メニューテーブル |
| userBuf | ユーザー用バッファ |
| clientDraw | クライアント描画コールバック関数アドレス |
| execFunc | ウィンドウ実行関数アドレス |
| destructor | ウィンドウ破棄コールバック関数アドレス |
| param 1, param 2 | ユーザーパラメータ |
| hClose | クローズボックスのウィンドウハンドル |
| hMaximize | 予約 |
| hMinimize | 予約 |

説明

全てのウィンドウの基本となる構造体です。ウィンドウハンドルはこの構造体へのポインタです。

参照

NWS_RGBA

構造体

| | |
|-----|--|
| 定義 | typedef struct _NWS_RGBA { Uint8 r; Uint8 g; Uint8 b; Uint8 a; } NWS_RGBA; |
| 説明 | 主にウィンドウのカラーを定義する構造体です。 |
| メンバ | r 赤 (0-255) g 緑 (0-255) b 青 (0-255) a 透明度 (0-255) 0 で完全透明、255 で不透明です。 |
| 参照 | nwGetWindowColor(),nwSetWindowColor() |

NWS_RECT

構造体

| | |
|-----|---|
| 定義 | typedef struct _NWS_RECT { Sint32 left; Sint32 top; Sint32 right; Sint32 bottom; } NWS_RECT; |
| 説明 | 画面上の矩形領域を定義する構造体です。 |
| メンバ | left 左端座標 top 上端座標 right 右端座標 bottom 下端座標 |
| 参照 | nwGetWindowRect() |

第5章 スクロールウィンドウ

5.1 概要

スクロールウィンドウは、ウィンドウクラス `NWD_WC_SCROLLWIN` を持つウィンドウです。スクロールウィンドウは、通常のウィンドウにはない、クライアント領域の表示内容をスクロールできるという機能を持っています。

5.2 スクロールウィンドウの作成

スクロールウィンドウの作成には、通常のウィンドウの作成と同じ関数 `nwCreateWindow()` を使います。ウィンドウクラスに `NWD_WC_SCROLLWIN` を指定することにより、スクロールウィンドウが作成できます。

```
static void scroll_window_callback(NWHWND hWnd)
{
    nwTextOut(hWnd, 1, 1, "Scroll Window Sample");
}

Bool create_scroll_window(void)
{
    NWHWND hWnd = nwCreateWindow(NWD_WC_SCROLLWIN,
                                "Scroll Window",
                                NWD_WS_CAPTION | NWD_WS_BORDER |
                                NWD_WS_SHADOW,
                                50, 50, 100, 100,
                                NULL);

    if (!hWnd) return FALSE;

    hWnd->clientDraw = scroll_window_callback;
    return TRUE;
}

void njUserInit(void)
{
    :
    nwInitSystem(...);
    create_scroll_window();
}
```

こうして作成したスクロールウィンドウの外観は通常のウィンドウと同じですが、クライアント領域をマウス操作でスクロールできます。クライアント領域にマウスカーソルを置いて左ボタンを押し、そのままマウスを動かすことにより、自由にスクロールさせることができます。

ウィンドウ作成直後は、クライアント領域は上下左右自由にスクロールできますが、上下方向のみ、左右方向のみスクロール可能にすることもできます。

上下方向のみスクロール可能にする設定

```
nwScrollWinEnableScroll(hWnd, NWD_ES_VERTICAL);
```

左右方向のみスクロール可能にする設定

```
nwScrollWinEnableScroll(hWnd, NWD_ES_HORIZONTAL);
```

上下左右にスクロール可能にする設定

```
nwScrWinEnableScroll(hWnd, NWD_ES_VERTICAL | NWD_ES_HORIZONTAL);
```

どの方向にもスクロール不可にする設定

```
nwScrWinEnableScroll(hWnd, 0);
```

5.3 スクロールウィンドウに関する関数の説明

| 関数 | 機能 |
|-----------------------------------|-----------------------------|
| <code>nwScrWinEnableScroll</code> | スクロールウィンドウのスクロールの許可、禁止を設定する |
| <code>nwScrWinSetClip</code> | スクロールウィンドウのスクロール範囲を設定する |
| <code>nwScrWinScroll</code> | スクロールウィンドウをスクロールする |

表 5-1. ウィンドウ作成に関する関数の一覧

Windows API

nwScrWinEnableScroll

スクロールウィンドウ関数

| | |
|-------|---|
| 書式 | <code>Bool nwScrWinEnableScroll(NHWND hWnd, long flag)</code> |
| パラメータ | <code>hWnd</code> スクロールウィンドウのウィンドウハンドル <code>flag</code> スクロール可能にしたい方向 <code>NWD_ES_VERTICAL</code> 上下方向にスクロール可能 <code>NWD_ES_HORIZONTAL</code> 左右方向にスクロール可能 |
| 戻り値 | 設定に成功した場合は TRUE、失敗した場合は FALSE を返します。 |
| 機能 | ・スクロールウィンドウのクライアント領域が、どの方向にスクロール可能にするかを設定します。 ・どの方向にもスクロールさせたくない場合は <code>flag</code> に 0 を指定します。 ・上下左右にスクロールさせたい場合はフラグを で組み合わせて指定します。 |
| 参照 | <code>nwScrWinSetClip()</code> , <code>nwScrWinScroll()</code> |
| 備考 | ・この関数による設定はマウス操作でのスクロールに対してのみ有効です。 <code>nwScrWinScroll()</code> によるスクロールは常に全方向に有効です。 |
| 例 | // すべての方向にスクロール可能にします <code>nwScrWinEnableScroll(hWnd, NWD_ES_VERTICAL NWD_ES_HORIZONTAL);</code> |

nwScrWinSetClip

スクロールウィンドウ関数

| | |
|-------|---|
| 書式 | <code>Bool nwScrWinSetClip(NHWND hWnd, NWS_RECT* rect)</code> |
| パラメータ | <code>hWnd</code> スクロールウィンドウのウィンドウハンドル <code>rect</code> スクロール可能矩形領域 |
| 戻り値 | 設定に成功した場合は TRUE、失敗した場合は FALSE を返します。 |
| 機能 | ・スクロールウィンドウのクライアント領域のスクロール範囲を設定します。 |
| 参照 | <code>nwScrWinEnableScroll()</code> , <code>nwScrWinScroll()</code> |
| 備考 | ・この関数による設定はマウス操作でのスクロールに対してのみ有効です。 ・クリッピング領域の初期値は、クライアント領域を中心とした、クライアント領域 9 個分 (3x3) のサイズの領域です。 |
| 例 | // スクロール範囲を (-10, -10) - (10, 10) に設定します。 <code>NWS_RECT rect = {-10, -10, 10, 10};</code> <code>nwScrWinSetClip(hWnd, &rect);</code> |

| nwScrWinScroll | | スクロールウィンドウ関数 |
|----------------|--|--------------|
|----------------|--|--------------|

| | |
|-------|---|
| 書式 | BoolInwScrW nScroll(NWHWND hWnd, Sint32 x, Sint32 y) |
| パラメータ | hWnd スクロールウィンドウのウィンドウハンドル x,y スクロール値 |
| 戻り値 | 設定に成功した場合は TRUE、失敗した場合は FALSE を返します。 |
| 機能 | ・スクロールウィンドウのクライアント領域を指定ドット数スクロールします。 |
| 参照 | nwScrW nEnableScroll(),nwScrW nSetClip() |
| 備考 | |
| 例 | <pre>//上に1ドットずつスクロール Sint32 njUserMain(void) { NWHWND hWnd = nwFindWindow(NULL, " Scroll Window"); if (hWnd) { nwScrWinScroll(hWnd, 0, 1); } : : return nwExecute(); }</pre> |

| nwScrWinGetScroll | | スクロールウィンドウ関数 |
|-------------------|--|--------------|
|-------------------|--|--------------|

| | |
|-------|---|
| 書式 | BoolInwScrW nGetScroll(NWHWND hWnd, Sint32* x, Sint32* y) |
| パラメータ | hWnd スクロールウィンドウのウィンドウハンドル x,y スクロール座標を取得するポインタ |
| 戻り値 | 設定に成功した場合は TRUE、失敗した場合は FALSE を返します。 |
| 機能 | ・スクロールウィンドウの現在のスクロール座標を取得します。 |
| 参照 | nwScrW nScroll() |
| 備考 | |
| 例 | <pre>//スクロールをリセットします Sint32 x, y; nwScrWinGetScroll(hWnd, &x, &y); nwScrWinSetScroll(hWnd, -x, -y);</pre> |

第6章 エディットウィンドウ

6.1 概要

エディットウィンドウは、ウィンドウクラス `NWD_WC_EDITWIN` を持つウィンドウであり、スクロールウィンドウ(ウィンドウクラス `NWD_WC_SCROLLWIN`)の機能を含んでいます。

エディットウィンドウには、以下の特徴があります。

- ・テキストバッファを持ち、テキストを設定して自動表示させることができる。
- ・複数行のテキストを表示させることができる。
- ・ウィンドウのクライアント領域をスクロールさせることができる。

Windows の Windows ユーティリティ `Debug Window` は、このエディットウィンドウとして実現されています。

6.2 エディットウィンドウの作成と使用方法

エディットウィンドウを作成するには、関数 `nwCreateEditWindow()` を呼び出します。

```
NHWND hWnd;  
  
hWnd = nwCreateEditWindow(500, "Edit Window 1",  
                           NWD_WS_CAPTION | NWD_WS_CONTROL | NWD_WS_THICKFRAME |  
                           NWD_WS_SHADING,  
                           50, 50, 150, 100, NULL);
```

次に、このウィンドウに文字列を追加してみましょう。それには、関数 `nwEditWinAddString()` を使います。

```
nwEditWinAddString(hWnd, "ABCD¥n");  
nwEditWinAddString(hWnd, "0123¥n");
```

さらに、`printf()`と同様に使用できる関数として、`nwEditWinPrintf()`も用意されています。

```
nwEditWinPrintf(hWnd, "ADDRESS : %08X¥n", 0);
```

ここまでの作業で、画面上に次のようなウィンドウが表示されます。

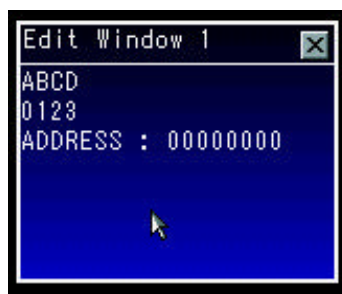


図 6-1 エディットウィンドウの作成例

エディットウィンドウに次々に文字列を追加していくと、ウィンドウのクライアント領域に表示しきれなく

なった時点で、自動的にスクロールします。
 また、エディットウィンドウはスクロールウィンドウの機能も持っているため、マウスドラッグによりクライアント領域の表示内容を自由にスクロールさせることができます。
 追加した文字列がバッファに入りきらなくなると、バッファの先頭から消去されていきます。

エディットウィンドウを破棄するには、他のウィンドウ同様、関数 `nwDestroyWindow()` を使います。

```
nwDestroyWindow(hWnd);
```

現バージョンの `NtWindows` では、関数 `nwEditWinAddString()`、`nwEditWinPrint()` で設定する文字列の最後には改行文字 `'\n'` が必要です。改行文字を付加しなかった場合、次にこれらの関数を用いて改行文字を含む文字列を設定するまで、先に設定した文字列は表示されませんので注意してください。

6.3 エディットウィンドウに関する関数の説明

| 関数 | 機能 |
|---------------------------------|---|
| <code>nwCreateEditWindow</code> | エディットウィンドウを作成する |
| <code>nwEditWinAddString</code> | エディットウィンドウにテキストを追加する |
| <code>nwEditWinPrintf</code> | <code>printf()</code> 書式でエディットウィンドウにテキストを追加する |

表 6-1. ウィンドウ作成に関する関数の一覧

NtWindows API

| nwCreateEditWindow | | ウィンドウ作成関数 |
|--------------------|--|---------------------|
| 書式 | <code>NWHWND nwCreateEditWindow(S int32 lines, S int8* caption, S int32 style, S int32 x, S int32 y, S int32 w, S int32 h, NWHWND hWndParent)</code> | |
| パラメータ | lines | テキストの最大行数 |
| | caption | ウィンドウ名 (キャプション) 文字列 |
| | style | ウィンドウスタイル |
| | x, y | クライアント領域の左上座標 |
| | w, h | クライアント領域の幅と高さ |
| | hWndParent | 親ウィンドウハンドル |
| 戻り値 | 成功した場合は作成されたエディットウィンドウのハンドルが、エディットウィンドウを作成できなかった場合は <code>NULL</code> が返されます。 | |
| 機能 | ・エディットウィンドウを作成し、テキストバッファを確保します。 | |
| 参照 | <code>nwDestroyWindow()</code> , <code>nwCreateMenuWindow()</code> , <code>nwCreateEditWindow()</code> , <code>nwCreateScrollBar()</code> , <code>nwCreateButton()</code> | |
| | <code>NWS_WN</code> 構造体 | |
| 備考 | | |
| 例 | <pre>NWHWND hWnd; hWnd = nwCreateEditWindow(500, "Edit Window", NWD_WS_CAPTION NWD_WS_CONTROL NWD_WS_BORDER NWD_WS_SHADING, 50, 50, 150, 100, NULL);</pre> | |

| nwEditWinAddString | | エディットウィンドウ関数 |
|--------------------|---|------------------------------|
| 書式 | <code>Bool nwEditWinAddString(NWHWND hWnd, S int8* string)</code> | |
| パラメータ | hWnd | 文字列を追加するエディットウィンドウのウィンドウハンドル |

| | |
|-----|---|
| 戻り値 | string 追加する文字列へのポインタ |
| 機能 | 文字列の追加に成功すれば TRUE、失敗した場合は FALSE を返します。 |
| 参照 | ・エディットウィンドウに文字列を追加し、表示します。 |
| 備考 | nwCreateEditWindow(), nwEditWinPrintf() |
| 例 | nwEditWinAddString(hWnd, "AddText¥n"); |

nwEditWinPrintf

エディットウィンドウ関数

| | |
|-------|--|
| 書式 | Bool nwEditWinPrintf(NHWND hWnd, Sint8* fmt, ...) |
| パラメータ | hWnd 文字列を追加するエディットウィンドウのウィンドウハンドル fmt printf()書式文字列 |
| 戻り値 | 文字列の追加に成功すれば TRUE、失敗した場合は FALSE を返します。 |
| 機能 | ・エディットウィンドウに文字列を追加し、表示します。 printf()書式が利用できます。 |
| 参照 | nwCreateEditWindow(), nwEditWinAddString() |
| 備考 | |
| 例 | nwEditWinPrintf(hWnd, "i = %d¥n", i); |

6.4 Nindows ユーティリティのデバッグウィンドウに関する関数の説明

nwDebugPrintf

エディットウィンドウ関数

| | |
|-------|---|
| 書式 | Bool nwDebugPrintf(Sint8* fmt, ...) |
| パラメータ | fmt printf()書式文字列 |
| 戻り値 | 文字列の追加に成功すれば TRUE、失敗した場合は FALSE を返します。 |
| 機能 | ・デバッグウィンドウに文字列を追加し、表示します。 printf()書式が利用できます。 |
| 参照 | nwEditWinPrintf() |
| 備考 | |
| 例 | nwDebugPrintf("i = %d¥n", i); |

第7章

スクロールバーコントロール

7.1 概要

スクロールバーは、各種の数値パラメータを調整するのに非常に適したコントロールです。例えば、背景やモデルの色を変更したり、オブジェクトの移動スピードを調整するなど、さまざまな用途で使えます。

7.2 スクロールバーコントロールの作成

モデルのマテリアル(NJS_ARGB 構造体)を、スクロールバーを使って変更する例を示します。

```
static NJS_ARGB argb = { 0.f, 0.f, 0.f, 0.f};

static NWS_SCROLLBARINFO material_scroll_info[] = {
    /* caption      data      min      max      line page pos */
    {"Alpha Scroll", {&argb.a, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
    {"Red Scroll",   {&argb.r, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
    {"Green Scroll", {&argb.g, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
    {"Blue Scroll",  {&argb.b, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
};

static NWS_SCROLLBARLIST material_scroll_list = {
    sizeof(material_scroll_info) / sizeof(NWS_SCROLLBARINFO),
    NWD_WS_SB_HORZ,      50, 3, 200, 14,
    material_scroll_info,
};

static void draw_material_window_callback(NWHWND hWnd)
{
    nwTextOut(hWnd, 2, 2, "A %.3f", argb.a);
    nwTextOut(hWnd, 2, 16, "R %.3f", argb.r);
    nwTextOut(hWnd, 2, 30, "G %.3f", argb.g);
    nwTextOut(hWnd, 2, 44, "B %.3f", argb.b);
}

static void create_material_window(void)
{
    NWHWND hWnd = nwCreateWindow(NWD_WC_WIN, "Material Window",
                                NWD_WS_CAPTION | NWD_WS_SHADING |
                                NWD_WS_BORDER | NWD_WS_CONTROL,
                                40, 382, 260, 64, NULL);
    hWnd->clientDraw = draw_material_window_callback;

    nwCreateScrollBarArray(&material_scroll_list, hWnd);
}

void njUserInit(void)
{
    :
    create_material_window();
}

Sint32 njUserMain(void)
{
    :
    :
    njSetConstantMaterial(&argb);
    njDrawObject(OBJECT);
    return nwExecute();
}
```

この例では、まずスクロールバーの親となるウィンドウ「Material Window」を作成し、4つのスクロールバーを子ウィンドウとして作成しています。このようにするのが一般的と言えます。これにより、次のようなウィンドウが表示されます。

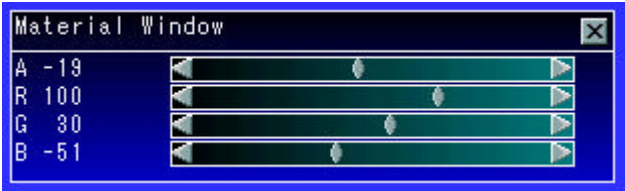


図 7-1 スクロールバーコントロールの作成例

ここで、スクロールバーつまみを操作すると それに応じて `NJS_ARGB` 構造体のメンバ `a,r,g,b` の値も変化します。

「Material Window」を破棄すると、子ウィンドウである4つのスクロールバーも自動的に破棄されます。

7.3 スクロールバーコントロールの作成に関する関数の説明

| 関数 | 機能 |
|-------------------------------------|---------------------------|
| <code>nwCreateScrollBarArray</code> | 複数のスクロールバーコントロールを一括して作成する |

表 7-1 スクロールバーコントロール作成に関する関数の一覧

Windows API

| <code>nwCreateScrollBarArray</code> | | スクロールバー関数 |
|-------------------------------------|--|------------------|
| 書式 | <code>Bool nwCreateScrollBarArray (NWS_SCROLLBARLIST* list, NHHWND hwndParent)</code> | |
| パラメータ | <code>list</code> | スクロールバーリストへのポインタ |
| | <code>hwndParent</code> | 親ウィンドウハンドル |
| 戻り値 | 全てのスクロールバーコントロールを作成できた場合は <code>TRUE</code> 、作成できなかった場合は <code>FALSE</code> を返します。 | |
| 機能 | 複数のスクロールバーコントロールを一括して作成し、パラメータを設定します。 | |
| 参照 | 低レベルスクロールバー関数: <code>nwCreateScrollBar()</code> , <code>nwSetScrollBarPos()</code> , <code>nwSetScrollBarRange()</code> , <code>nwSetScrollBarData()</code> , <code>nwSetScrollBarLineMove()</code> , <code>nwSetScrollBarPageMove()</code> | |
| 備考 | スクロールバーを1つずつ作成してパラメータを設定するよりずっと簡単に使用できますが、作成したスクロールバーのハンドルを取得することはできません。 通常はスクロールバーのハンドルを取得する必要はないと思いますが、必要な場合は低レベルスクロールバー関数を使用するか、 <code>nwFindWindow()</code> でウィンドウを探すようにしてください。 | |
| 例 | | |

構造体

| <code>NWS_SCROLLBARLIST</code> | | 構造体 |
|--------------------------------|---|-----|
| 定義 | <pre>typedef struct { Sint32 n; Sint32 style; Sint16 x, y; Sint16 w, h;</pre> | |

| | |
|-----|--|
| 説明 | <pre> NWS_SCROLLBARINFO* info; } NWS_SCROLLBARLIST; nwCreateScrollBarArray ()関数を用いてスクロールバーを作成する時に NWS_SCROLLBARINFO 構造体とともに必要となります。 </pre> |
| メンバ | <pre> n style x,y w,h info </pre> <p>NWS_SCROLLBARINFO の配列の要素数です。</p> <p>水平スクロールバーの場合は NWD_WS_SB_HORZ、 垂直スクロールバーの場合は NWD_WS_SB_VERT です。</p> <p>最初のスクロールバーを表示する座標 (親ウィンドウ相対) です。</p> <p>スクロールバー 1つの幅と高さです。</p> <p>NWS_SCROLLBARINFO 構造体の配列アドレスです。</p> |
| 参照 | NWS_SCROLLBARLIST NWS_DATA, nwCreateScrollBarArray () |

NWS_SCROLLBARINFO

構造体

| | |
|-----|--|
| 定義 | <pre> typedef struct { Sint8* caption; NWS_DATA data; Float min, max; Float line, page; Float pos; } NWS_SCROLLBARINFO; </pre> |
| 説明 | <p>nwCreateScrollBarArray ()関数を用いてスクロールバーを作成する時に NWS_SCROLLBARLIST 構造体とともに必要となります。この構造体 1つがスクロールバー 1つに対応します。通常は配列として用い、複数のスクロールバーを一括作成します。</p> |
| メンバ | <pre> caption data min max line page pos </pre> <p>スクロールバーのキャプション文字列です。</p> <p>スクロールバーに関連付けるデータ構造体へのポインタです。</p> <p>関連付けされたデータのとりうる値の最小値です。</p> <p>関連付けされたデータのとりうる値の最大値です。</p> <p>スクロールバー矢印をクリックした時のデータの変化量です。</p> <p>スクロールエリアをクリックしたときのデータの変化量です。</p> <p>関連付けされたデータの初期値です。</p> |
| 参照 | NWS_SCROLLBARLIST NWS_DATA, nwCreateScrollBarArray () |

NWS_DATA

構造体

| | |
|-----|---|
| 定義 | <pre> typedef struct _NWS_DATA { void *dt; int type; } NWS_DATA; </pre> |
| 説明 | スクロールバーに関連付けるデータを表す構造体です。 |
| メンバ | <pre> dt type </pre> <p>データへのポインタです。</p> <p>データタイプです。下の表の中から指定します。</p> |
| 参照 | 低レベルスクロールバー関数 |

| データタイプ | 意味 |
|---------------|------------------------------|
| NWD_DT_CHAR | データは char(Sint8)型。 |
| NWD_DT_SHORT | データは short(Sint16)型。 |
| NWD_DT_LONG | データは long(Sint32)型。 |
| NWD_DT_FLOAT | データは float(Float)型。 |
| NWD_DT_UCHAR | データは unsigned char(Uint8)型 |
| NWD_DT_USHORT | データは unsigned short(Uint16)型 |
| NWD_DT_ULONG | データは unsigned long(Uint32)型 |

7.4 低レベルスクロールバー関数を利用したスクロールバーコントロールの作成

ここでは、既に説明した `nwC reateScrollBarArray` (ではなく、もっと低レベルの関数を使用して、全く同じマテリアルウィンドウを作成する方法を解説します。必要ないと思われる方は次章に進んでください。

```
static NJS_ARGB argb = { 0.f, 0.f, 0.f, 0.f};

static NWS_DATA dt[] = {
    {(void*)&argb.a, NWD_DT_FLOAT},
    {(void*)&argb.r, NWD_DT_FLOAT},
    {(void*)&argb.g, NWD_DT_FLOAT},
    {(void*)&argb.b, NWD_DT_FLOAT},
};

static void draw_material_window_callback(NWHWND hWnd)
{
    nwTextOut(hWnd, 2, 2, "A %3.0f", argb.a);
    nwTextOut(hWnd, 2, 16, "R %3.0f", argb.r);
    nwTextOut(hWnd, 2, 30, "G %3.0f", argb.g);
    nwTextOut(hWnd, 2, 44, "B %3.0f", argb.b);
}

static void create_material_window(void)
{
    NWHWND hScl;
    NWHWND hWnd = nwCreateWindow(NWD_WC_WIN, "Material Window",
                                NWD_WS_CAPTION | NWD_WS_SHADING |
                                NWD_WS_BORDER | NWD_WS_CONTROL,
                                40, 382, 260, 64, NULL);
    hWnd->clientDraw = draw_material_window_callback;

    hScl = nwCreateScrollBar(NWD_WS_SB_HORZ, "Alpha Scroll", 80, 3, 200, 11, hWnd);
    nwSetScrollBarData(hScl, &dt[0]);
    nwSetScrollBarRange(hScl, -256.0f, 255.0f);
    nwSetScrollBarPos(hScl, 0.f);
    nwSetScrollBarLineMove(hScl, 1.f);
    nwSetScrollBarPageMove(hScl, 10.f);

    hScl = nwCreateScrollBar(NWD_WS_SB_HORZ, "Red Scroll", 80, 17, 200, 25, hWnd);
    nwSetScrollBarData(hScl, &dt[1]);
    nwSetScrollBarRange(hScl, -256.0f, 255.0f);
    nwSetScrollBarLineMove(hScl, 1.f);
    nwSetScrollBarPageMove(hScl, 10.f);
    nwSetScrollBarPos(hScl, 0.f);

    hScl = nwCreateScrollBar(NWD_WS_SB_HORZ, "Green Scroll", 80, 31, 200, 39, hWnd);
    nwSetScrollBarData(hScl, &dt[2]);
    nwSetScrollBarRange(hScl, -256.0f, 255.0f);
    nwSetScrollBarPos(hScl, 0.f);
    nwSetScrollBarLineMove(hScl, 1.f);
    nwSetScrollBarPageMove(hScl, 10.f);

    hScl = nwCreateScrollBar(NWD_WS_SB_HORZ, "Blue Scroll", 80, 45, 200, 53, hWnd);
    nwSetScrollBarData(hScl, &dt[3]);
    nwSetScrollBarRange(hScl, -256.0f, 255.0f);
    nwSetScrollBarPos(hScl, 0.f);
    nwSetScrollBarLineMove(hScl, 1.f);
    nwSetScrollBarPageMove(hScl, 10.f);
}
```

以上で、図 7-1 と全く同じ機能、外観を持つ『Material Window』が作成できます。

7.5 低レベルスクロールバー関数の説明

| 関数 | 機能 |
|-------------------------------------|---------------------------|
| <code>nwCreateScrollBar</code> | スクロールバーコントロールを作成する |
| <code>nwSetScrollBarData</code> | スクロールバーにデータを関連付ける |
| <code>nwSetScrollBarRange</code> | スクロールバーの範囲を設定する |
| <code>nwSetScrollBarPos</code> | スクロールバーつまみの位置を設定する |
| <code>nwSetScrollBarLineMove</code> | スクロールバーアロークリックでの移動量を設定する |
| <code>nwSetScrollBarPageMove</code> | スクロールバーエリアクリックでの移動量を設定する。 |

表 7-3 低レベルスクロールバー関数の一覧

Windows API

nwCreateScrollBar

低レベルスクロールバー関数

| | |
|-------|--|
| 書式 | <code>NW_HWND nwCreateScrollBar(S int32 type, S int8 *caption, S int32 x, S int32 y, S int32 w, S int32 h, NW_HWND hWndParent);</code> |
| パラメータ | <p><code>type</code> 水平スクロールバーの場合は <code>NWD_WS_SB_HORZ</code>、垂直スクロールバーの場合は <code>NWD_WS_SB_VERT</code></p> <p><code>caption</code> スクロールバーのキャプション(文字列)</p> <p><code>x, y</code> スクロールバーを作成する座標(親ウィンドウ相対)</p> <p><code>w, h</code> スクロールバー幅と高さ</p> <p><code>hWndParent</code> 親ウィンドウのウィンドウハンドル</p> |
| 戻り値 | スクロールバーコントロールを作成できた場合は作成したスクロールバーコントロールのウィンドウハンドルを、作成できなかった場合は <code>NULL</code> を返します |
| 機能 | スクロールバーコントロールを作成します |
| 参照 | <code>nwCreateScrollBarArray()</code> 、 低レベルスクロールバー関数: |
| 備考 | <code>nwSetScrollBarPos()</code> <code>nwSetScrollBarRange()</code> <code>nwSetScrollBarData()</code> <code>nwSetScrollBarLineMove()</code> <code>nwSetScrollBarPageMove()</code> 作成したスクロールバーは、必ず <code>nwSetScrollBarPos()</code> <code>nwSetScrollBarRange()</code> <code>nwSetScrollBarData()</code> を使用して初期設定を行ってください。 |
| 例 | <pre>NW_HWND hSc1 = nwCreateScrollBar(NWD_WS_SB_HORZ, "Alpha Scroll", 80, 3, 200, 11, hWndParent);</pre> |

nwSetScrollBarData

低レベルスクロールバー関数

| | |
|-------|---|
| 書式 | <code>void nwSetScrollBarData(NW_HWND hSc1, NWS_DATA * data)</code> |
| パラメータ | <p><code>hSc1</code> スクロールバーコントロールのウィンドウハンドル</p> <p><code>data</code> 関連付けするデータ構造体</p> |
| 戻り値 | なし |
| 機能 | スクロールバーコントロールにデータを関連付けます |
| 参照 | <code>nwSetScrollBarPos()</code> <code>nwSetScrollBarRange()</code> <code>nwSetScrollBarLineMove()</code> <code>nwSetScrollBarPageMove()</code> |
| 備考 | |
| 例 | <pre>// bng 変数 a をスクロールバーに関連付けます bng a; NWS_DATA data = {&a, NWD_DT_LONG}; nwSetScrollBarData(hWnd, &data);</pre> |

nwSetScrollBarRange

低レベルスクロールバー関数

| | |
|-------|---|
| 書式 | void nwSetScrollBarRange(NWHWND hScI, F bat min, F bat max) |
| パラメータ | hScI スクロールバーコントロールのウィンドウハンドル min スクロールバーに関連付けたデータのとりうる最小値 max スクロールバーに関連付けたデータのとりうる最大値 |
| 戻り値 | なし |
| 機能 | スクロールバーのレンジを設定します |
| 参照 | nwSetScrollBarPos(),nwSetScrollBarData(),nwSetScrollBarLineMove(),nwSetScrollBarPageMove() |
| 備考 | |
| 例 | // レンジを(-30 ~ 30)に設定します nwSetScrollBarRange(hW nd, -30.f, 30.f); |

nwSetScrollBarPos 低レベルスクロールバー関数

| | |
|-------|--|
| 書式 | void nwSetScrollBarPos(NWHWND hScI, F bat pos) |
| パラメータ | hScI スクロールバーコントロールのウィンドウハンドル pos スクロールバーに関連付けたデータの値 |
| 戻り値 | なし |
| 機能 | スクロールバーに関連付けたデータの値を設定します |
| 参照 | nwSetScrollBarRange(),nwSetScrollBarData(),nwSetScrollBarLineMove(),nwSetScrollBarPageMove() |
| 備考 | スクロールバーを作成した際のデータの初期値を設定する際に使用します。 |
| 例 | // データの初期値を 0 にします nwSetScrollBarPos(hW nd, 0.f); |

nwSetScrollBarLineMove 低レベルスクロールバー関数

| | |
|-------|---|
| 書式 | void nwSetScrollBarLineMove(NWHWND hScI, F bat step) |
| パラメータ | hScI スクロールバーコントロールのウィンドウハンドル step ステップ値 |
| 戻り値 | なし |
| 機能 | スクロールバー両端の矢印 (スクロールバーアロー)を押したときのデータの変化量を設定します |
| 参照 | nwSetScrollBarRange(),nwSetScrollBarData(),nwSetScrollBarPos(),nwSetScrollBarPageMove() |
| 備考 | デフォルトは 1.0 となっています。 |
| 例 | // スクロールバーアローを押したときのデータの変化量を 2 にします nwSetScrollBarLineMove(2.f); |

nwSetScrollBarPageMove 低レベルスクロールバー関数

| | |
|-------|---|
| 書式 | void nwSetScrollBarPageMove(NWHWND hScI, F bat step) |
| パラメータ | hScI スクロールバーコントロールのウィンドウハンドル step ステップ値 |
| 戻り値 | なし |
| 機能 | スクロールバーエリアをクリックしたときのデータの変化量を設定します |
| 参照 | nwSetScrollBarRange(),nwSetScrollBarData(),nwSetScrollBarPos(),nwSetScrollBarLineMove() |
| 備考 | デフォルトは 10.0 となっています。 |

例

```
// スクロールバーエリアをクリックした時のデータの変化量を5 にします  
nwSetScrollBarLineMove(5.f);
```

第8章 ボタンコントロール

8.1 概要

ボタンは、クリックすることによりコールバックを発生する、様々な目的に利用できるコントロールです。アプリケーションで保持しているフラグ変数のトグル切り替え、複数のオブジェクトから1つを選択する際のインタフェース等、工夫しだいで便利に利用できます。

8.2 ボタンコントロールの作成

例として、Back、Nextボタンにより環境マッピングに使用するテクスチャを選択するサンプルを作成します。

```
#define TEXTURES 4
static int texno = 0;

static void test_window_callback(NHWND hWnd)
{
    char buf[256];
    sprintf(buf, "Texture=%d", texno);
    nwTextOut(hWnd, 1, 1, buf);
}

static void button_callback_back(NHWND hWnd)
{
    texno--;
    if (texno < 0) texno = TEXTURES - 1;
}

static void button_callback_next(NHWND hWnd)
{
    texno++;
    if (texno >= TEXTURES) texno = 0;
}

Bool create_test_window(void)
{
    NHWND hWndParent;
    hWndParent = nwCreateWindow(NWD_WC_WIN,
                                "Texture Select",
                                NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
                                50, 50, 128, 48,
                                NULL);
    if (!hWndParent) return FALSE;
    hWndParent->clientDraw = test_window_callback;
    nwCreateButton(button_callback_back, "Back", 3, 20, 48, 13, hWndParent);
    nwCreateButton(button_callback_next, "Next", 56, 20, 48, 13, hWndParent);
    return TRUE;
}

Sint32 njUserMain(void)
{
    :
    :
    njSetTexture(&texlist[texno]);
    njDrawObject(OBJECT);
    return nwExecute();
}
```

以上で、次のようなテクスチャ選択ウィンドウが表示されます。

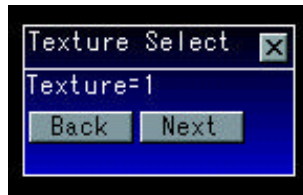


図 8-1. テクスチャ選択ウィンドウの作成例

「Back」、「Next」ボタンをクリックすると、それぞれ指定のコールバック関数が呼び出され、変数 `texno` の値が変更されます。それに応じて環境マッピングに使用されるテクスチャも変更されます。テクスチャ選択ウィンドウを破棄すると、子ウィンドウである2つのボタンも自動的に破棄されます。

8.3 ボタンの有効化と無効化

先程のサンプルでは、「Back」、「Next」のボタンは常に有効で、クリックすると必ずコールバックが働きます。しかし実際には、状況に応じてボタンを使用不可にしたりする必要があるでしょう。では、先程のサンプルを改造して、そのような処理を追加してみましょう。

追加する処理は、ボタン作成時に、「Back」ボタンを無効にする処理、ボタンの親ウィンドウのコールバック関数中でテクスチャ番号を調べ、それに応じて2つのボタンの有効、無効を切り替える処理です。そしてそのためには、作成したボタンのウィンドウハンドルをグローバル変数に格納しておくといでしょう。

ボタンの有効、無効を設定するには、関数 `nwEnableButton()` を使用します。

ボタンを有効にする処理

```
nwEnableButton(button, TRUE);
```

ボタンを無効にする処理

```
nwEnableButton(button, FALSE);
```

無効化されたボタンは、ボタンの文字が淡色表示され、クリックしてもボタンアニメーションやコールバックが働かなくなります。

```

#define TEXTURES 4
static int texno = 0;
static NWHWND button_back;
static NWHWND button_next;

static void test_window_callback(NWHWND hWnd)
{
    char buf[256];
    sprintf(buf, "Texture=%d", texno);
    nwTextOut(hWnd, 1, 1, buf);

    // テクスチャ番号が0なら、Backボタンを使用不可にする
    if (texno == 0) nwEnableButton(button_back, FALSE);
    else nwEnableButton(button_back, TRUE);

    // テクスチャ番号が3なら、Nextボタンを使用不可にする
    if (texno == TEXTURES - 1) nwEnableButton(button_next, FALSE);
    else nwEnableButton(button_next, TRUE);
}

static void button_callback_back(NWHWND hWnd)
{
    texno--;
    if (texno < 0) texno = 0;
}

static void button_callback_next(NWHWND hWnd)
{
    texno++;
    if (texno >= TEXTURES) texno = TEXTURES - 1;
}

Bool create_test_window(void)
{
    NWHWND hWndParent;
    hWndParent = nwCreateWindow(NWD_WC_WIN,
                                "Texture Select",
                                NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
                                50, 50, 128, 48,
                                NULL);
    if (!hWndParent) return FALSE;

    hWndParent->clientDraw = test_window_callback;

    button_back = nwCreateButton(button_callback_back,
                                  "Back", 3, 20, 48, 13, hWndParent);
    button_next = nwCreateButton(button_callback_next,
                                  "Next", 56, 20, 48, 13, hWndParent);

    // 最初はテクスチャ番号が0なので、Backボタンを使用不可にする
    nwEnableButton(button_back, FALSE);
    return TRUE;
}

```

8.4 ボタンコントロールに関する関数の説明

| 関数 | 機能 |
|-----------------------------|-----------------|
| <code>nwCreateButton</code> | ボタンコントロールを作成する |
| <code>nwEnableButton</code> | ボタンの有効、無効を切り替える |

表 8-1 ボタンコントロール作成に関する関数の一覧

| nwCreateButton | | ボタン関数 |
|----------------|--|-----------------------|
| 書式 | NHHWND nwCreateButton(NWF_BUTTONFUNC func,S int8 *caption, S int32 x,S int32 y ,S int32 w ,S int32 h,NHHWND hW ndParent); | |
| パラメータ | func | ボタンコールバック関数 |
| | caption | ボタン表面に表示する文字列 |
| | x,y | ボタンを作成する座標 (親ウィンドウ相対) |
| | w ,h | ボタンの幅と高さ |
| | hW ndParent | 親ウィンドウのウィンドウハンドル |
| 戻り値 | ボタンを作成できた場合はボタンのウィンドウハンドルを、作成できなかった場合はNULL を返します。 | |
| 機能 | ボタンコントロールを作成します。 | |
| 参照 | nwEnableButton(),nwDestroyWindow(), | |
| 備考 | 作成されたばかりのボタンは有効状態です。 | |
| 例 | // OK ボタンを作成します | |
| | NHHWND button = nwCreateButton(button_callback_back, "OK", 3, 20, 48, 13, hW ndParent); | |

| nwEnableButton | | ボタン関数 |
|----------------|--|--------------------------------|
| 書式 | void nwEnableButton(NHHWND hW nd,Bool flag) | |
| パラメータ | hW nd | ボタンのウィンドウハンドル |
| | flag | ボタンを有効化の場合は TRUE、無効化の場合は FALSE |
| 戻り値 | なし | |
| 機能 | ボタンの有効、無効を設定します。無効化されたボタンは、ボタンの文字が淡色表示され、ボタンをクリックしてもコールバックが働きなくなります。 | |
| 参照 | nwCreateButton() | |
| 備考 | 作成されたばかりのボタンは有効状態です。 | |
| 例 | // ボタンを無効化します | |
| | nwEnableButton(button,FALSE); | |

コールバック関数

| ButtonCallback | | ボタンコールバック関数 |
|----------------|---|------------------|
| 書式 | void ButtonCallback(NHHWND hW nd) | |
| パラメータ | hW nd | コールバック元のボタンのハンドル |
| 戻り値 | なし | |
| 機能 | ボタンがクリックされた時にコールバックされる、アプリケーション定義の関数です。 | |
| 参照 | nwCreateButton(),nwEnableButton() | |
| 備考 | | |

第9章 メニュー

9.1 概要

Nindows は、一般的な GUI システムにおけるポップアップメニューを実現するための API を備えています。

Nindows において、最も代表的なメニューはシステムメニューですが、このメニュー項目の中に、「User(undefined)」という 淡色表示されている項目があります。



図 9-1 システムメニューの「User(undefined)」項目

この項目は、ユーザー定義のメニューを設定するための項目です。メニューテーブルを作成してこの項目に登録することにより、簡単にユーザー定義のメニューが使用できるようになります。この章では、メニューテーブルの作成方法と、作成したメニューテーブルの登録方法を解説します。また、システムメニューに登録せずに直接ポップアップウィンドウを作成する方法も合わせて解説します。

9.2 メニューテーブルの作成と登録

メニューテーブルは、構造体 `NWS_MENU_TABLE` の配列として実現します。以下は、1つの項目からなる、最も簡単なメニューテーブルの例です。

```
static void menu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

static NWS_MENU_TABLE menu_table[] = {
    // 項目 1
    {
        NWD_MF_NORMAL,                // ノーマルメニュー項目であることを示すフラグ
        "Test Menu 1",                // 項目文字列
        menu_callback,                // 項目が選択された時に呼び出されるコールバ
        0,                            // コールバック関数に渡されるパラ
    },
    // テーブルの終了を表す
    {
        NWD_MF_NULL,                  // テーブル終了
        "",
        NULL,
        0
    },
};
```

このメニューテーブルをシステムメニューの「User(undefined)」項目に登録するには、関数 `nwSetUserMenu()` を使います。

```

void njUserInit(void)
{
    :
    nwInitSystem(...);
    :
    nwSetUserMenu(test_menu);
}

```

この関数の呼び出しにより、淡色だった「User(undefined)」項目は、「User >」という表示に変わり、登録した「Test Menu 1」がサブメニューとしてポップアップします。



図 9-2 ユーザーメニューを登録した状態(1)

このメニューが選択されると、メニューテーブルで設定したコールバック関数 `menu_callback()` がコールバックされます。`menu_callback()` では特に処理をしていないため、このメニューを選択しても何も起こりません。このコールバック関数については次節で解説します。

もう少し複雑なメニューテーブルの例も見ましょう

```

static void menu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

static NWS_MENUTABLE menu_table[] = {
    // 項目 1
    {
        NWD_MF_NORMAL,                // ノーマルメニュー項目であることを示すフラグ
        "Test Menu 1",                // 項目文字列
        menu_callback,                // 項目が選択された時に呼び出されるコールバ
        0,                            // コールバック関数に渡されるパラ
    },
    // 項目 2
    {
        NWD_MF_SEPARATOR,             // セパレータ
        "",
        NULL,
        0,
    },
    // 項目 3
    {
        NWD_MF_POPUP,                 // ポップアップメニューであることを示すフラグ
        "Test Popup 1",               // 項目文字列
        NULL,
        (Sint32)submenu_table,       // サブメニューのメニューテーブルアドレス
    },
    // テーブルの終了を表す
    {
        NWD_MF_NULL,                  // テーブル終了
        "",
        NULL,
        0,
    }
}

```

```

static void submenu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

static NWS_MENUTABLE submenu_table[] = {
    // 項目 1
    {
        NWD_MF_NORMAL | NWD_MF_GRAYED,          // ノーマルメニュー項目、選択不可
        "Sub Menu 1",                             // 項目文字列
        submenu_callback,                         // コールバック関数
        0,                                         // コール
        バック関数に渡されるパラメータ
    },
    // 項目 2
    {
        NWD_MF_NULL,                             // メニュー終了
        "",
        NULL,
        0
    },
};
};

```

このテーブルを同じように `nwSetUserMenu()` を使って登録すると、次のようなメニューになります。
 なお、`nwSetUserMenu()` で新しいメニューテーブルを登録すると、今まで登録されていたメニューテーブルは上書きされ、新しいメニューが使用可能になります。



図 9-3 ユーザーメニューを登録した状態(2)

ユーザーメニューの登録を取り消すには、以下のようになります。

```

nwSetUserMenu(NULL);

```

これでふたたび「User(undefined)」の淡色表示となり、ユーザーメニューは選択できなくなります。

9.3 メニューコールバック関数

メニューコールバック関数は、メニューが選択された場合にコールバックされるユーザー定義の関数で、メニューテーブルに登録しておきます。先程の例では、関数 `menu_callback()` がそれに当たります。

```
static void menu_callback(NHWND hWndMenu, Sint32 idx, Sint32 param)
```

パラメータ `hWndMenu` には、コールバック元のメニューウィンドウのウィンドウハンドルが渡されてきます。通常は使用する必要はないでしょう。

パラメータ `idx` は、選択されたメニュー項目が、その項目を含むメニューの何番目の項目であるかを示す、0から始まるインデックス番号です。1つのコールバック関数で複数のメニュー項目の処理を行いたい場合などに、どのメニュー項目が選択されたかを判定するのに用いることができます。

`param` は、メニューテーブルでユーザーが定義したパラメータです。これも同様に1つのコールバック関数で複数のメニュー項目の処理を行いたい場合などに用いることができます。

9.4 チェックマーク

メニュー項目文字列の左側に、チェックマークを表示することができます。チェックマークは、ユーザーに対して、今その項目が有効である、その項目が選択中であることを示すために便利です。図は、Nindowsユーティリティの「Ninja Info」を選択した状態です。「Ninja Info」ウィンドウが表示中であることを示すために、「Ninja Info」の項目名の左にチェックマークを表示しています。「Ninja Info」ウィンドウを閉じると、チェックマークも表示されなくなります。

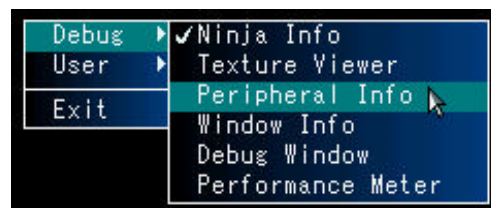


図 9-4.チェックマークの例

チェックマークの表示、非表示を切り替えるには、メニューテーブルのメンバ `type` を直接設定します。例は、メニューテーブル `menu_table` の先頭項目のチェックマークを切り替える例です。

チェックマークを表示する

```
menu_table[0].type |= NWD_MF_CHECKED;
```

チェックマークを消去する

```
menu_table[0].type &= ~NWD_MF_CHECKED;
```

では、具体的な例を見てみましょう

```

static NWS_MENUTABLE menu_table[] = {
    {NWD_MF_NORMAL, "Test Window", create_test_window, 0},
    {NWD_MF_NULL, "", NULL, 0},
};

static void destroy_test_window_callback(NHWND hWnd)
{
    menu_table[0].type &= ~NWD_MF_CHECKED;
}

static void create_test_window(NHWND hWndMenu, Sint32 idx, Sint32 param)
{
    NHWND hWnd;
    // ウィンドウがすでに作成されていたら破棄する
    // ウィンドウデストラクタによりチェックマークが消去される
    if (hWnd = nwFindWindow(NULL, "Test Window")) {
        nwDestroyWindow(hWnd);
    } else {
        hWnd = nwCreateWindow(NWD_WC_WIN, "Test Window",
                               NWD_WS_CAPTION | NWD_WS_CONTROL | NWD_WS_SHADING |
NWD_WS_BORDER,
                               50, 50, 100, 100, NULL);
        // デストラクタの設定
        hWndNjInfo->destructor = destroy_test_window_callback;
        // 「Test Window 1」を作成したので、メニュー項目にチェックマークを付ける
        menu_table[0].type |= NWD_MF_CHECKED;
    }
}

void njUserInit(void)
{
    :
    nwInitSystem(...);
    :
    nwSetUserMenu(menu_table);
}

```

このメニューテーブルを `nwSetUserMenu()` で登録し、メニューから「Test Window」を選択すると選択するたびにウィンドウの作成、破棄を行い、それに応じてチェックマークを切り替えます。ウィンドウデストラクタ(`hWnd->destructor`)を設定して、その中でチェックマークの消去を行っているのは、メニュー選択以外の方法でウィンドウが破棄される場合があるためです。次のコードは、一見正しいようですが、クローズボックスをクリックしてウィンドウを破棄した場合などに、チェックマークが消去されません。

```

static void create_test_window(NHWND hWndMenu, Sint32 idx, Sint32 param)
{
    NHWND hWnd;

    if (hWnd = nwFindWindow(NULL, "Test Window")) {
        nwDestroyWindow(hWnd);
        menu_table[0].type &= ~NWD_MF_CHECKED;
    } else {
        hWnd = nwCreateWindow(NWD_WC_WIN, "Test Window",
                               NWD_WS_CAPTION | NWD_WS_CONTROL | NWD_WS_SHADING |
NWD_WS_BORDER,
                               50, 50, 100, 100, NULL);
        menu_table[0].type |= NWD_MF_CHECKED;
    }
}

```

95 ユーザーメニューの登録に関する関数の説明

| 関数 | 機能 |
|---------------|------------------------|
| nwSetUserMenu | システムメニューにユーザーメニューを登録する |

表 9-1 ユーザーメニューの登録に関する関数の一覧

Windows API

| nwSetUserMenu | | メニュー関数 |
|---------------|--|--------------------|
| 書式 | void nwSetUserMenu(NWS_MENUTABLE* menuTbl) | |
| パラメータ | menuTbl | メニューテーブル構造体の配列アドレス |
| 戻り値 | なし | |
| 機能 | ユーザーメニューを、システムメニューの「User」項目に、ポップアップメニューとして登録します。引数に NULL を指定した場合、登録してあるユーザーメニューを破棄します。 | |
| 参照 | | |
| 備考 | | |
| 例 | nwSetUserMenu(user_menu); | |

コールバック関数

| MenuCallback | | メニュー関数 |
|--------------|---|-------------------------------|
| 書式 | void MenuCallback(NHWND hwnd, Sint32 idx, Sint32 param) | |
| パラメータ | hwnd | コールバック元のメニューウィンドウのウィンドウハンドル |
| | idx | 選択されたメニュー項目の、メニューテーブル上のインデックス |
| | param | メニューテーブルで設定したパラメータ |
| 戻り値 | なし | |
| 機能 | メニューが選択された場合にメニューウィンドウによってコールバックされる、ユーザー定義の関数です。 | |
| 参照 | | |
| 備考 | | |
| 例 | | |

構造体

| NWS_MENUTABLE | | 構造体 |
|---------------|---|-----------------------|
| 定義 | <pre>typedef struct _NWS_MENUTABLE { Sint32 type; Sint8 *title; NWF_MENUHANDLE func; Sint32 param; } NWS_MENUTABLE;</pre> | |
| 説明 | nwSetUserMenu()関数でユーザーメニューを登録したり、nwCreateMenu()関数でメニューウィンドウを作成する場合に、メニュー内容を定義します。 | |
| メンバ | type | メニュー項目タイプ |
| | title | メニュー項目文字列 |
| | func | メニューが選択された場合のコールバック関数 |
| | param | コールバック関数に渡されるパラメータ |
| 参照 | | |

| メニュータイプフラグ | 意味 |
|------------------|---|
| NWD_MF_NORMAL | ノーマルメニュー項目。NWD_MF_POPUP、NWD_MF_SEPARATOR と同時には指定できない。 |
| NWD_MF_POPUP | ポップアップするサブメニューを持つ。NWD_MF_NORMAL、NWD_MF_SEPARATOR と同時には指定できない。 |
| NWD_MF_SEPARATOR | セパレータ。NWD_MF_NORMAL、NWD_MF_POPUP と同時には指定できない。 |

| | |
|----------------|------------------|
| NWD_MF_CHECKED | チェックマークを持つ。 |
| NWD_MF_GRAYED | 淡色表示され、選択できない項目。 |

96 ポップアップメニューの作成

96.1 簡単なポップアップメニューの作成

これまでは、システムメニューにユーザーメニューを設定する方法を解説してきましたが、システムメニューに登録せずに、画面にポップアップメニューを作成することもできます。それには関数 `nwCreateMenuWindow()` を使用します。

```
static void submenu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

static NWS_MENUTABLE submenu_table[] = {
    {NWD_MF_NORMAL | NWD_MF_GRAYED, "Sub Menu 1", submenu_callback,
    0},
    {NWD_MF_NULL, "",
    NULL, 0},
};

static void menu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

static NWS_MENUTABLE menu_table[] = {
    {NWD_MF_NORMAL, "Test Menu 1", menu_callback, 0},
    {NWD_MF_SEPARATOR, "", NULL,
    0},
    {NWD_MF_POPUP, "Test Popup 1", NULL,
    (Sint32)submenu_table},
    {NWD_MF_NULL, "", NULL,
    0},
};

void create_popup_menu(void)
{
    NWHWND hWnd;

    hWnd = nwCreateMenuWindow(menu_table, "Test Menu", 10, 10, NULL);
}

void njUserInit(void)
{
    :
```

これで、画面に次のようなポップアップメニューが表示されます。



図 9-5 作成したポップアップメニュー

作成したメニューウィンドウは、メニュー項目が選択されるか、メニューウィンドウ領域外がクリックされると自動的に破棄されます。

96.2 画面上に常駐するポップアップメニューの作成

メニューウィンドウは、メニュー項目が選択されるか、メニューウィンドウ領域外がクリックされると自動的に破棄されるため、このままでは最大 1 回しかメニューを選択できません。ふたたび `nwCreateMenuWindow()` を使用して同じメニューを作成することが必要となります。

画面に常駐するメニューを作成したい場合は、次のようにします。

```
void create_popup_menu(void)
{
    NWHWND hWnd;

    hWnd = nwFindWindow(NULL, "Test Menu");
    if (!hWnd)
        hWnd = nwCreateMenuWindow(menu_table, "Test Menu", 10, 10, NULL);
}

Sint32 njUserMain(void)
{
    :
    :
    create_popup_menu();
    :
    :
    return nwExecute();
}
```

すでにメニューウィンドウがあるかどうかを毎フレーム調べて、もしなければメニューウィンドウを作成しなおします。これで、一見画面に常駐しているように見えるポップアップメニューが実現できます。

9.7 ポップアップメニューの作成に関する関数の説明

| nwCreateMenuWindow | | メニュー関数 |
|--------------------|---|--------------------|
| 書式 | <code>NWHWND nwCreateMenuWindow(NWS_MENU_TABLE *menuTbl, Sint8 *caption, Sint32 x, Sint32 y, NWHWND hWndParent);</code> | |
| パラメータ | <code>menu</code> | メニューテーブル構造体の配列アドレス |
| 戻り値 | 成功した場合には、新しく作成したメニューウィンドウのウィンドウハンドル、メニューウィンドウを作成できなかった場合には <code>NULL</code> を返します。 | |
| 機能 | ポップアップメニューウィンドウを作成します。 | |
| 参照 | <code>nwDestroyWindow()</code> , <code>NWS_MENU_TABLE</code> 構造体 | |
| 備考 | | |
| 例 | <code>NWHWND hWnd = nwCreateMenu(menu_tbl, "MENU", 100, 100, NULL);</code> | |

第 10 章 マウス

10.1 概要

N indows では、特にマウスに関する関数を用意していません。マウスカーソルの座標、ボタン情報の取得は N inja 関数を使用して行います。

10.2 マウス情報の取得

マウス情報の取得には、N inja 関数 `njGetPeripheral()` を使用します。次のサンプルを参考にしてください。

```
Sint32 njUserMain(void)
{
    Sint16 x, y;
    NJS_PERIPHERAL* mouse = njGetPeripheral(NJD_PORT_SYSMOUSE);

    x = mouse->x;           // マウスカーソルの x 座標
    y = mouse->y;           // マウスカーソルの y 座標

    if (mouse->on & NJD_DGT_B0) {
        // 左ボタンが押されている
    }
    if (mouse->on & NJD_DGT_B1) {
        // 右ボタンが押されている
    }
    if (mouse->press & NJD_DGT_B0) {
        // 左ボタンが押された
    }
    if (mouse->press & NJD_DGT_B1) {
        // 右ボタンが押された
    }
    if (mouse->release & NJD_DGT_B0) {
        // 左ボタンが放された
    }
    if (mouse->release & NJD_DGT_B1) {
        // 右ボタンが放された
    }

    :
    :
    return nwExecute();
}
```

マウスカーソルの座標にあるウィンドウを知りたい場合は、以下のようになります。

```
Sint32 njUserMain(void)
{
    Sint16 x, y;
    NHHWND hWnd;
    NJS_PERIPHERAL* mouse;
    static Sint16 old_x = 640 / 2, old_y = 480 / 2;

    mouse = njGetPeripheral(NJD_PORT_SYSMOUSE);

    hWnd = nwFindWindowByPos(old_x, old_y);

    if (hWnd) {
        // マウスカーソルのある座標に、何かウィンドウが表示されている
    } else {
        // マウスカーソルのある座標には、ウィンドウは表示されていない
    }

    :
    old_x = mouse->x;
    old_y = mouse->y;

    return nwExecute();
}
```

変数 `old_x`, `old_y` に前フレームでのマウス座標を保存しておき、その座標にあるウィンドウを調べます。これは、ウィンドウの位置やその他の情報が、`nwExecute()`呼び出しのタイミングで更新されるためです。`njGetPeripheral()`で得られたマウス座標に対して、`nwExecute()`呼び出し前のウィンドウ状態には1フレーム分の遅れがあるため、`nwFindWindowByPos()`は正しく動作しない場合があります。

10.3 マウス情報の取得に関する関数の説明

ここでは、Ninjaのペリフェラル関数および構造体を、特にマウスに特化して解説します。

| 関数 | 機能 |
|------------------------------|---------------|
| <code>njGetPeripheral</code> | ペリフェラル情報を取得する |

表 10-1 ユーザーメニューの登録に関する関数の一覧

Windows API

`njGetPeripheral`

Ninja 関数

| | |
|-------|--|
| 書式 | <code>NJS_PERIPHERAL* njGetPeripheral(long port)</code> |
| パラメータ | <code>port</code> ペリフェラルポート番号 マウスの情報を取得するには、 <code>NJD_PORT_SYSMOUSE</code> を指定してください。 |
| 戻り値 | マウス情報を格納した構造体のアドレス |
| 機能 | マウス情報を取得します。 |
| 参照 | <code>NJS_PERIPHERAL</code> 構造体 |
| 備考 | 1フレーム中何度でも呼び出すことができますが、情報が更新されるのはフレームの更新タイミングです。 |

| | |
|---|---|
| 例 | <pre> Sint32 nJserMain(void) { NJS_PERIPHERAL* mouse = nJgetPeripheral(NJD_PORT_SYSMOUSE); : } </pre> |
|---|---|

構造体

| NJS_PERIPHERAL | | Ninja 構造体 |
|----------------|--|---|
| 定義 | <pre> typedef struct { Uint32 id; Uint32 on; Uint32 off; union { Uint32 push; Uint32 press; }; union { Uint32 pull; Uint32 release; }; Sint16 x; Sint16 y; Sint16 z; Sint16 r; Sint16 u; Sint16 v; Sint8* name; void* extend; Uint32 old; } NJS_PERIPHERAL; </pre> | |
| 説明 | この構造体は N Jndows による定義ではなく N Ninja の持つ構造体です。 ジョイスティック、キーボード、マウス等、入力装置の情報が格納されます。 | |
| メンバ | id on off push, press pull, release x, y z, r, u, v name extend old | ペリフェラル ID (NJD_DEV_SYSMOUSE) です。 押されているボタンに対応したビットが 1 になります。 押されているボタンに対応したビットが 0 になります。 押した瞬間のボタンに対応したビットが 1 になります。 押した瞬間のボタンに対応したビットが 0 になります。 マウスの座標が格納されます。 未使用 (予約) ペリフェラル名称です。 未使用 (予約) 予約 |
| 参照 | nJgetPeripheral() | |

第 11 章 フォント

11.1 概要

Windows では、システムメニューの「Font」項目の選択によってのみ、フォントを切り替えることができます。本バージョンでは、選択されているフォントの種類、幅、高さを取得する関数のみがサポートされています。

11.1 フォントに関する関数の説明

| 関数 | 機能 |
|----------------------------|--------------|
| <code>nwGetFontSize</code> | フォントの幅と高さの取得 |

表 11-1 ユーザーメニューの登録に関する関数の一覧

Windows API

| <code>nwGetFontSize</code> | フォント関数 |
|----------------------------|--|
| 書式 | <code>Sint32 nwGetFontSize(Sint32* width, Sint32* height)</code> |
| パラメータ | <code>width, height</code> フォントの幅と高さを取得するポインタ |
| 戻り値 | 選択されているフォントの種類 |
| 機能 | システムメニューの「Font」で選択されているフォントの種類、幅、高さを取得します |
| 参照 | |
| 備考 | |
| 例 | <code>Sint32 width, height;</code> <code>nwGetFontSize(&width, &height);</code> |

11.2 フォントの変更に関する問題

Windows ではフォントの変更に伴うウィンドウのリサイズには対応していません。また、一部の特殊なウィンドウとコントロール「Property」などは、大きいサイズのフォントでは正しく表示されません。