YaneSDK3rdセットアップ for Visual C++ 6.0

はじめに

YaneSDKを使用する際に、複数プロジェクトに切り分けてゲームプロジェクトを作成する方法をまとめておきます。

基本的にはInsideYaneSDKさんのVC++7.0用の解説と同じ手順を踏もうと思います。

また、ここに書いてあることはあくまで「僕は」こうやっているという方法です。

知識不足等から、予期せぬ嘘が混ざっていることもありますので、ご注意ください。

もしそういった記述を見つけられた場合は適宜修正していきますので、管理人までご報告していただけると助かります。

YaneSDK3rdのダウンロードと設定

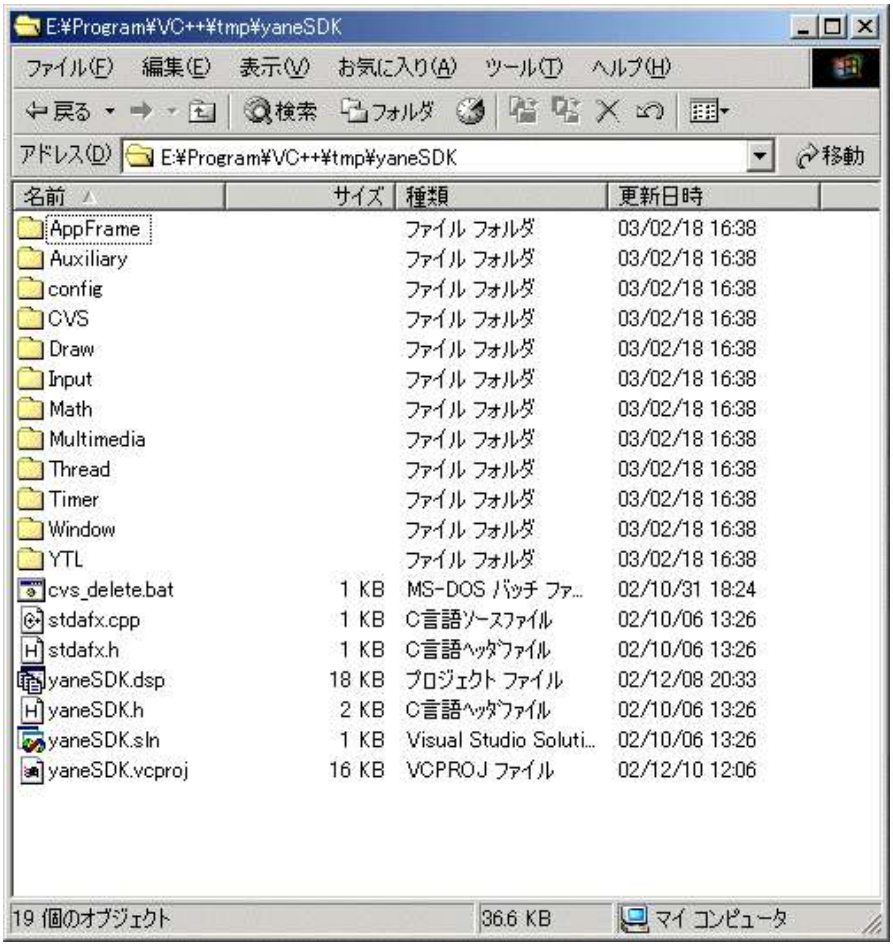
まずはソースをダウンロードします。

やねうらおさんのページのダウンロードコーナーから、最新版のソースを入手しましょう。

http://yaneurao.hp.infoseek.co.jp/yaneSDK3rd/

YaneSDK3rdはいまだ開発中の段階なので、バージョンアップがないかどうか定期的に調べたほうがいいかもしれませんね。

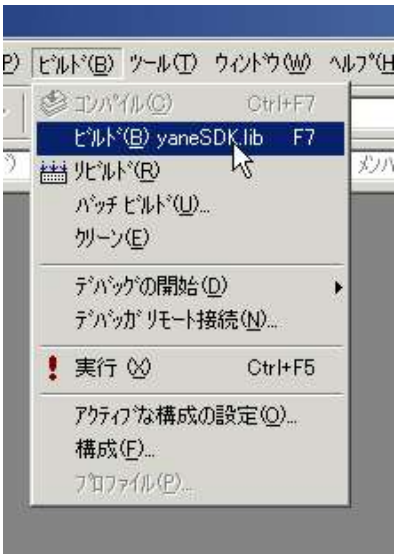
さて、ダウンロードしたファイルを適当なフォルダに展開すると、以下のようになるはずです。



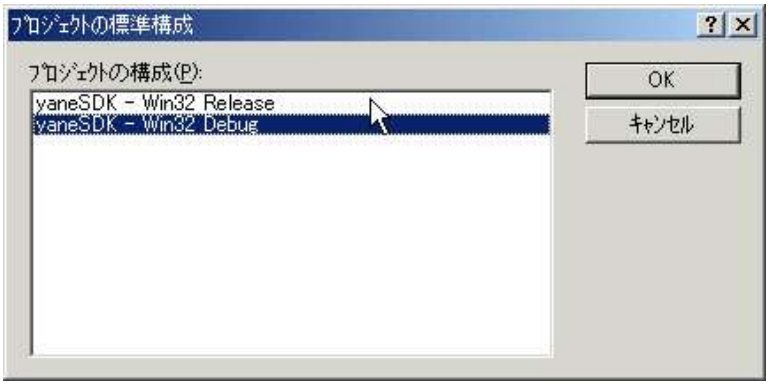
このうち、"cvs_delete.bat"というのは、展開した中にあるcvs(ソースのバージョン管理用ソフトらしい)用のファイルを削除するバッチファイルです。
別にcvsファイルが残っていても悪さはしないと思いますが、気持ち悪い人は"cvs_delete.bat"を実行して消してしまいましょう。

解凍した中に、すでにYaneSDK3rd用のプロジェクトが含まれていますので、"yaneSDK.dsp"というファイルをダブルクリックしてVisualC++の開発環境を立ち上げます。
下のような警告が出ると思いますが、気にしなくて結構です。

これが、YaneSDK本体のプロジェクトになります。
YaneSDK2ndのマニュアルに書いてあるようなややこしい設定はすべてやってありますので、"Debug"・"Release"両構成でビルドするだけでOKです。
初期状態では"Debug"構成になっているはずですので、このままビルドしましょう。
F7を押すか、メニューの「ビルド」>「ビルド」を選択します。



これで、プロジェクトを展開したフォルダに"Debug"というサブフォルダが作られ、その中に"YaneSDK.lib"が作成されたはずです。
続いて、メニューの「ビルド」>「アクティブな構成の設定」で、アクティブな構成を"Release"に切り替えます。



先ほどと同じ容量でプロジェクトをビルドすると、"Release"というサブフォルダ内に"YaneSDK.lib"が作成されます。

これで、第一段階は終了となります。

自分のプロジェクトへYaneSDKを関連付ける

続いて、自分のプロジェクトを作ります。

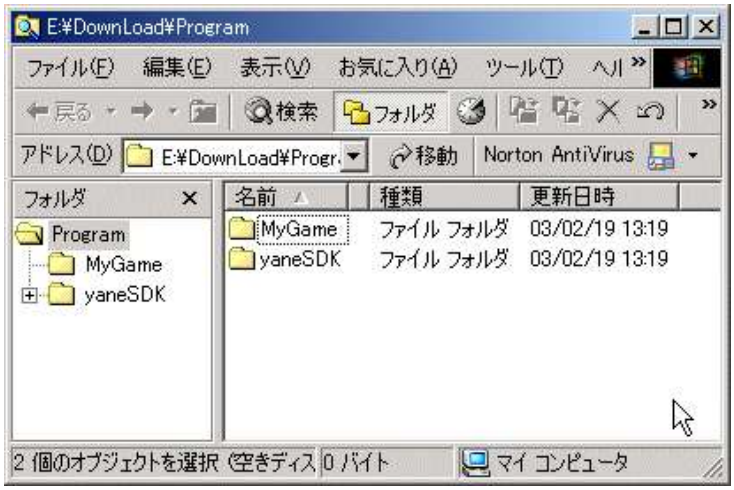
メニューの「ファイル」>「新規作成」から、"Win32 Applcation"用のワークスペースを新規作成してください。

ソースがごちゃごちゃになってしまうと大変なので、先ほどYaneSDKを展開したのとは別のフォルダに作成した方がいいでしょう。

この後の説明のために、"YaneSDK"フォルダと同じ階層にプロジェクト用のフォルダを作成することにします。

プロジェクトは、"空のプロジェクト"を選択します。

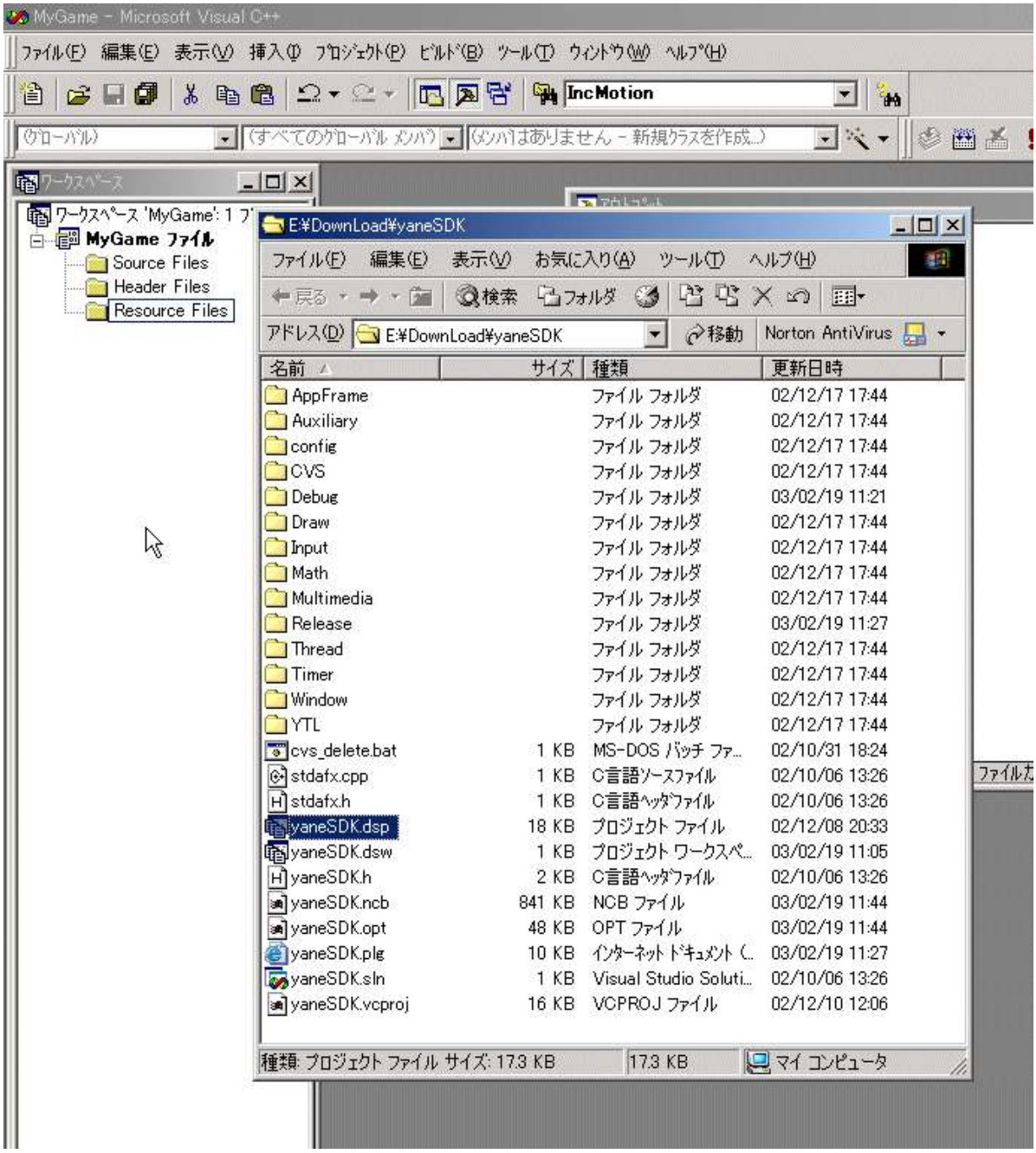
プロジェクトフォルダの階層構造は、このようになっています。



では、このプロジェクトにYaneSDKをインポートしましょう。

方法はとても簡単。エクスプローラを立ち上げて、先ほどの"YaneSDK.dsp"というファイルを、VisualC++のワークスペースウィンドウにドラッグ＆ドロップするだけです。

"YaneSDK.dsw"と間違えないようにしてください。



すると、以下のような警告がでます。「OK」を選択してください。

これで、自分のプロジェクトにYaneSDKプロジェクトが追加されました。

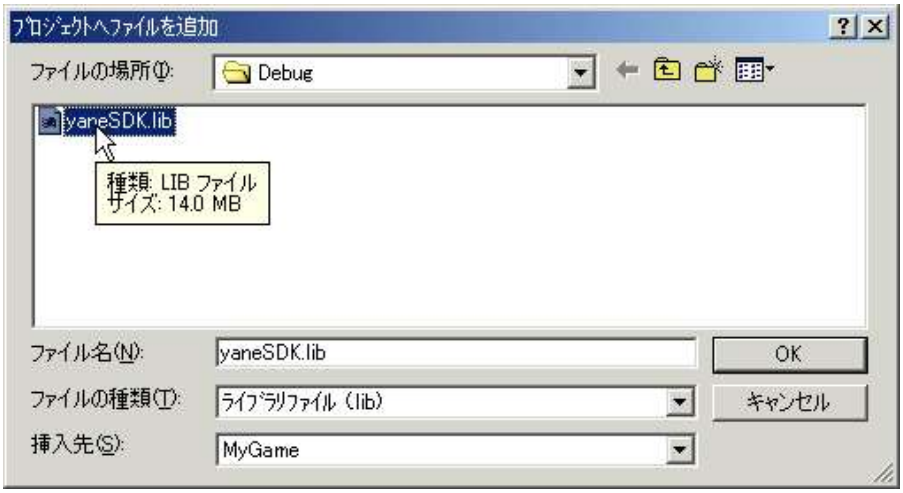
次に、自分のプロジェクトをYaneSDKに関連付けしましょう。

簡単に言えば、「このプロジェクトにはこっちのプロジェクトが必要だよ」という設定をします。

ワークスペースウィンドウで自分のプロジェクト(例では"MyGame")の右クリックメニュー「アクティブプロジェクトに設定」を選択して、自分のプロジェクトをアクティブにします(名前が太字で表示されます)。

メニューの「プロジェクト」>「プロジェクトに追加」>「ファイル」から、先ほど作成した"Debug"・"Release"両方の"YaneSDK.lib"を追加します。

この両ファイルはファイル名が一緒なので、先にどちらのファイルを追加したかを覚えておきましょう(ここでは、"Debug"ビルドしたものを先に追加します)。



次に、どの構成のときにどのライブラリをビルドするかを設定します。

ここからがちょっとややこしいです。

"YaneSDK.lib"が2つあるので、自分のプロジェクトが"Debug"ビルドの時は"Debug"用の"YaneSDK.lib"、"Release"ビルドの時は"Release"用の"YaneSDK.lib"を使用するようにします。

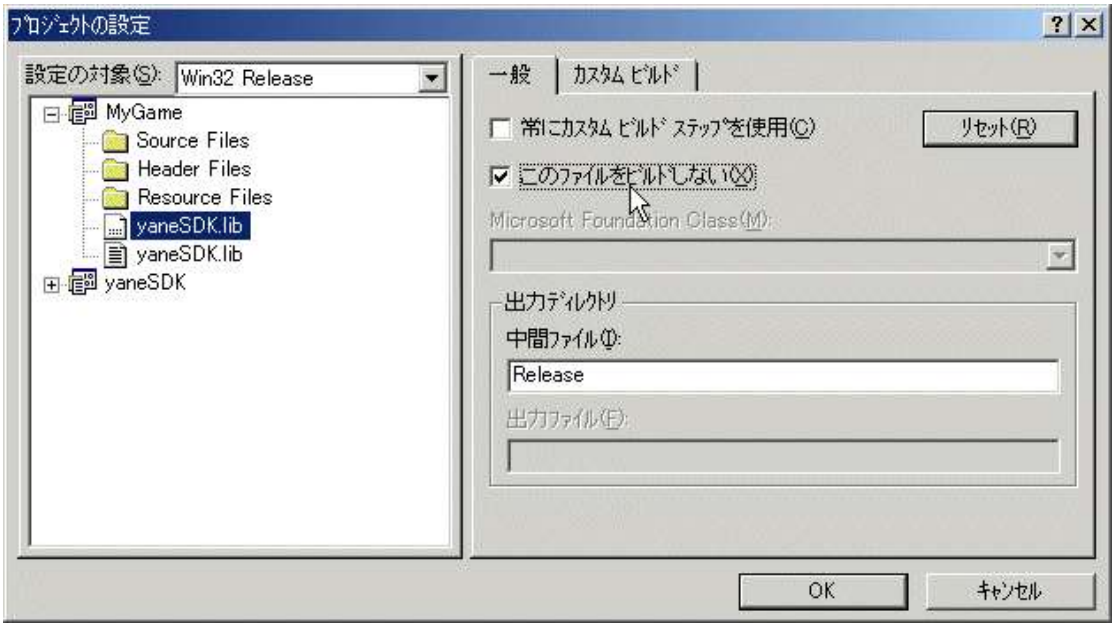
現時点では、両方ともビルドされるようになっていますので、必要ない方をビルドしないようにすればいい、ということになります。

ワークスペースウィンドウの上の方の"YaneSDK.lib"の右クリックメニュー「設定」を選択してください。

ここで選択した"YaneSDK.lib"は"Debug"用のライブラリです(先に追加した方のライブラリ)。

そこで、"Release"ビルドをする時には、このファイルをビルドしないようにします。

左上にあるコンボボックスから"Win32 Release"を選択し、右の方にある「このファイルをビルドしない」にチェックをつけます。



これで、"Debug"用のライブラリの設定は完成です。

"Release"用のライブラリに対しては、いまと逆のことをしてやります。

右クリックメニュー「設定」のウィンドウで、コンボボックスを"Win32 Debug"にして、「このファイルをビルドしない」をチェックします。

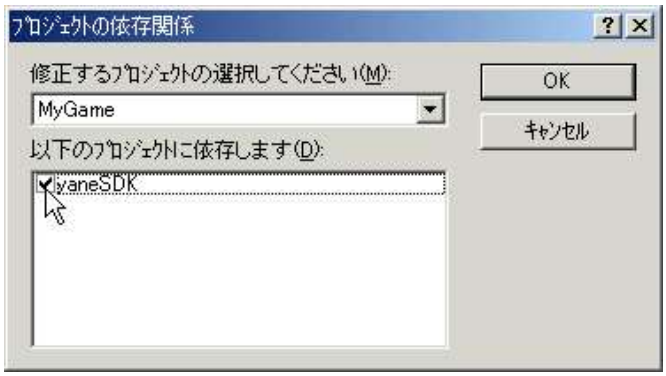
最後に、依存関係を設定しましょう。

依存関係というのは、「このプロジェクトをビルドする時は、まずこっちのプロジェクトをビルドしてね」という設定のことです。

これをやっておかないと、バージョンアップ等でYaneSDKのソースを変更した場合、上記の作業をもう一度やり直さなくてはならなくなります。

依存関係を設定するには、メニューの「プロジェクト」>「依存関係」を選択します。

コンボボックスが自分のプロジェクト名(ここではMyGame)になっているのを確認して、下のリスト中の"YaneSDK"をチェックします。

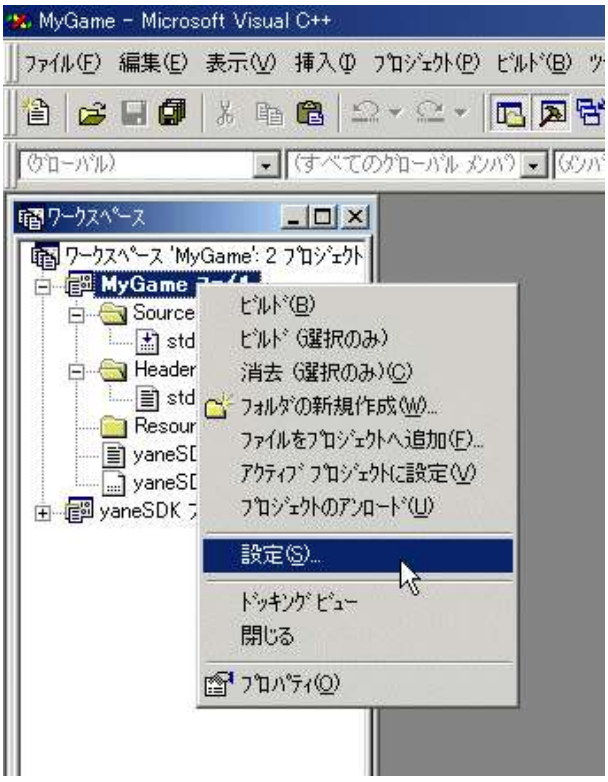


自分のプロジェクトとリンクするライブラリの設定を変えます。

VC++6.0ではデフォルトではシングルスレッドのライブラリとリンクするようになっているんですが、YaneSDKの方はマルチスレッドのライブラリを使用するようになっています。

このままだとライブラリリンク設定の整合性がとれていないというエラー(warning LINK4098)が出るので、自分のプロジェクトの設定を変更します。

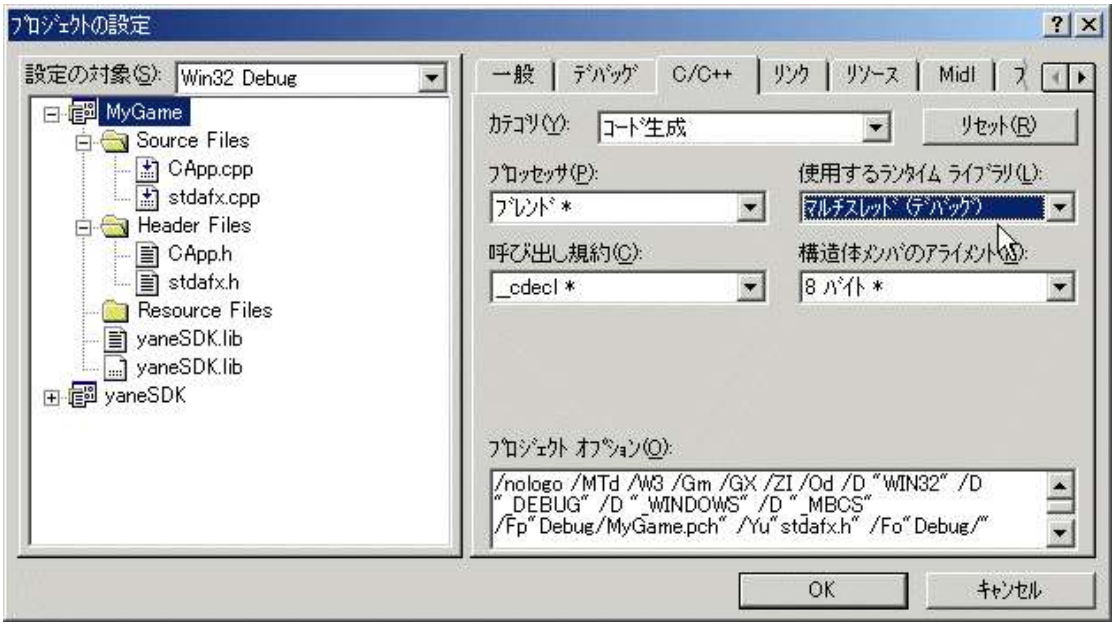
ワークスペースウィンドウの自分のプロジェクトの右クリックメニュー「設定」を選択してください。



左上のコンボボックスから"Win32 Debug"を選択して、「C/C++」タブを開きます。

「カテゴリ」コンボボックスの「コード生成」を選択します。

「使用するランタイムライブラリ」の欄を「マルチスレッド(デバッグ)」にします。



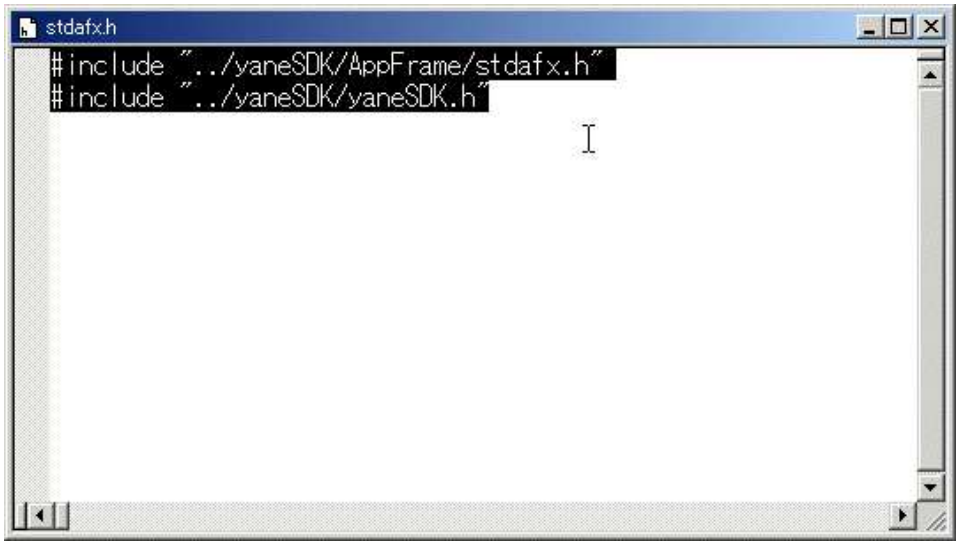
左上のコンボボックスから"Win32 Release"を選択して、同じ欄を「マルチスレッド」にします。

以上で自分のプロジェクトとYaneSDKの関連付けは終了です。

自分のプロジェクトの設定

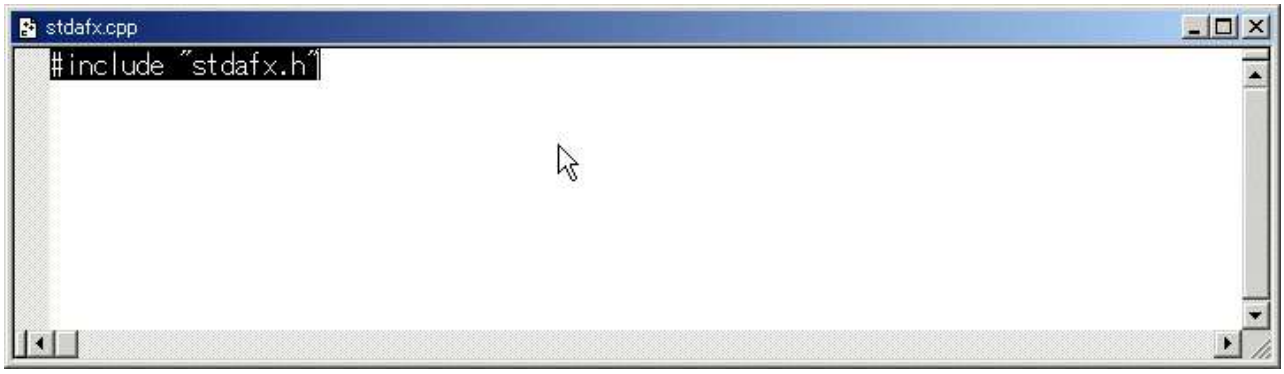
プロジェクトの設定はできましたが、プロジェクトの中身は空っぽのままです。
YaneSDKを使用するには、ライブラリ用のヘッダファイルをインクルードする必要があります。
ついでに、プリコンパイルヘッダの設定もしてしまいましょう。
まず、プロジェクトに"stdafx.h"というファイルを追加して、以下の2行を書いてください。

```
#include "../yaneSDK/AppFrame/stdafx.h"
#include "../yaneSDK/YaneSDK.h"
```



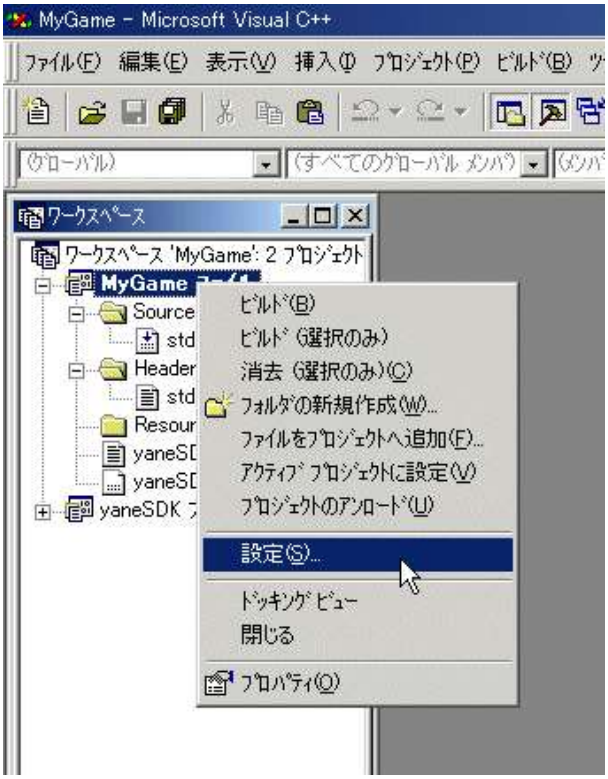
次に、プロジェクトに"stdafx.cpp"というファイルを追加して、以下の1行を書いてください。

```
#include "stdafx.h"
```



続いてプリコンパイルヘッダの設定をします。

ワークスペースウィンドウの自分のプロジェクトの右クリックメニュー「設定」を選択してください。



左上のコンボボックスから"すべての構成"を選択し、「C/C++」タブを開きます。
「カテゴリ」コンボボックスの「プリコンパイル済みヘッダー」を選択します。
「プリコンパイル済みヘッダーファイル(pch)を使用」を選択して、ファイル名に"stdafx.h"を指定します。

続いて、ワークスペースウィンドウの"stdafx.cpp"の右クリックメニュー「設定」を選択します。
左上のコンボボックスから"すべての構成"を選択し、「C/C++」タブを開きます。
「カテゴリ」コンボボックスの「プリコンパイル済みヘッダー」を選択します。
「プリコンパイル済みヘッダーファイル(pch)を作成」を選択して、ファイル名に"stdafx.h"を指定します。

これで一通り設定は終わったんですが、このままではWinMain関数すらないので、コンパイルが通りません。
せっかくですから、簡単なアプリケーションを作ってみましょう。
プロジェクトに"CApp.h"というファイルを追加して、以下のソースを打ち込んでください。

list-1 CApp.h

```
#pragma once

class CApp : public CAppFrame {
    virtual void MainThread();
};
```

続いて、"CApp.cpp"というファイルを追加して、以下のソースを打ち込んでください。

list-2 CApp.cpp

```
#include "stdafx.h"
#include "CApp.h"

void    CApp::MainThread() {

    CFastDraw draw;
    draw.SetDisplay();

    CFPSTimer t;

    //  これをメインアプリにする（終了するときに、他のウィンドゥをすべて閉じる）
    SetMainApp(true);
    CKey1 key;

    while (IsThreadValid()){

        draw.GetSecondary()->Clear();
        draw.OnDraw();

        key.Input();
        if (key.IsKeyPress(0)) return ;
        //  ESCキーが押されたら終了

        t.WaitFrame();
    }
}

//  これがmain windowのためのクラス。
class CAppMainWindow : public CAppBase {    //  アプリケーションクラスから派生
    virtual void MainThread(){              //  これがワーカースレッド
        CApp().Start();
    }
};

//  言わずと知れたWinMain
int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow)
{
    CAppInitializer init(hInstance,hPrevInstance,lpCmdLine,nCmdShow);
    //  必ず書いてね

    CSingleApp sapp;
    if (sapp.IsValid()) {
        CThreadManager::CreateThread(new CAppMainWindow);
        //  上で定義したメインのウィンドゥを作成
    }

    //  ここでCAppInitializerがスコープアウトするのだが、このときに
    //  すべてのスレッドの終了を待つことになる
    return 0;
}
```

早速ビルドして実行してみましょう。
何もないウィンドウが表示されれば、設定は成功です。

公開：2003-03-06

プログラム関連記事に戻る
管理人：そい