Visual Studio 2005：Visual CG++ / View2D 利用の Windows プログラミング（その１）

映像メディア研究室
降旗 隆
（VS2005_View2D マニュアル 1.jtd）

Visual CG++ / View2D を利用した Windows プログラム(その１)－ SDI プログラム－を開発する手順を以下に示す.

SDI：Single Document Interface　単一のウィンドウで構成されるプログラム

【プログラミング例題】

プログラム名：View2D_SDI_Test

処理内容：① SDI 形式でウィンドウズ画面上に "Hello! Image Media Lab." と"映像メディア研究室へようこそ！" を表示する.

②メニュー「ファイル」－「開く」により選択したビットマップ原画像を表示.

③原画像をモノクロ画像に変換しファイルに保存してから、その保存画像を表示.

④その変換画像を、メニュー「ファイル」－「名前を付けて保存」によりファイルに保存.

１．　Visual C++ の起動－ Windows プログラムのスケルトン作成

** まず最初に Visual C++ プログラミング開発ツール[Visual Studio 2005]を起動する.

－ OS：Windows 2000（or XP / Vista etc.）を起動

－ Visual C++ プログラミング開発ツール［Visual Studio 2005］を起動（図１）.

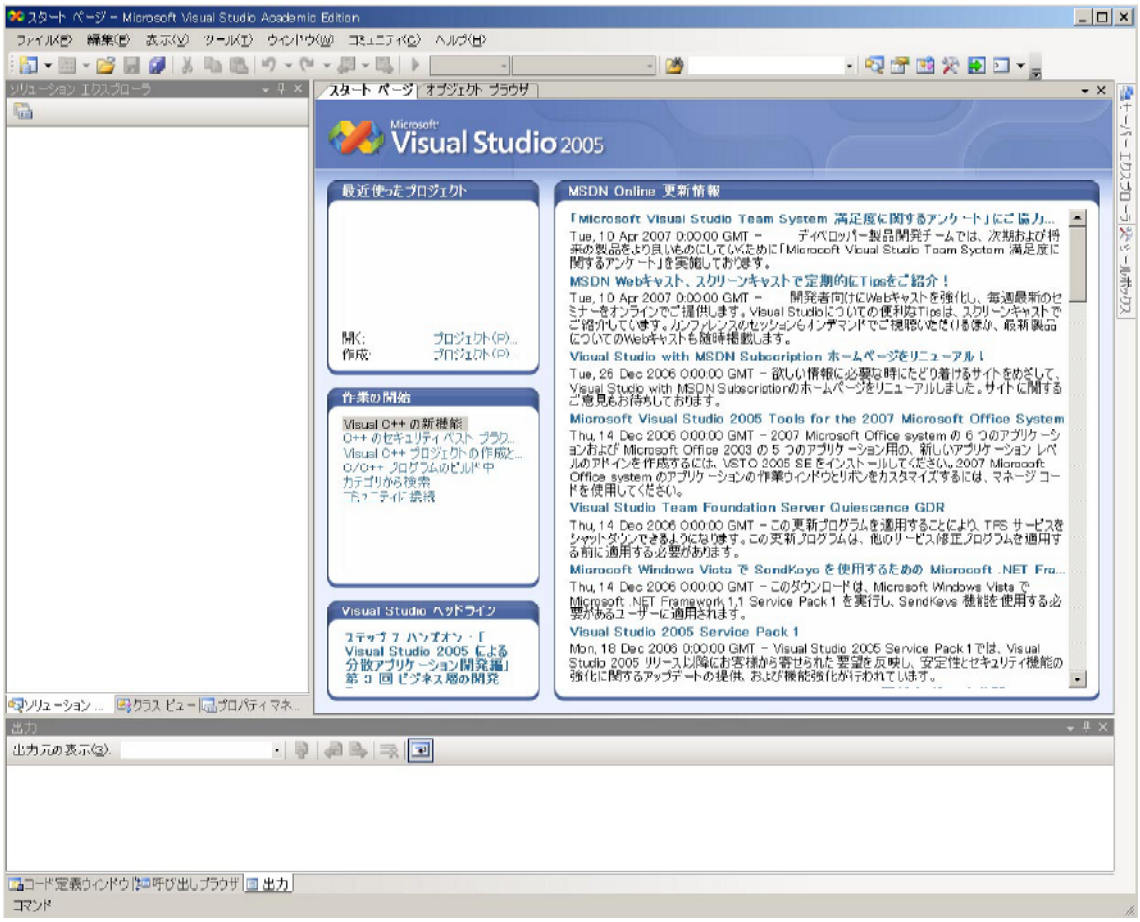

図１　Visual Studio 2005 の起動画面

** 「スタートページ― Microsoft Visual Studio Academic Edition」の画面上で
―「ファイル」―「新規作成」―「プロジェクト」を開く（図２）.

図２ 新規作成プロジェクトを開く

―「新しいプロジェクト」のサブ画面（図３）が開かれ、
［プロジェクトの種類］の欄より［Visual C++― MFC］を選択(ハイライト)し、
［テンプレート］の欄より［MFC アプリケーション］を選択してから、下欄で
［プロジェクト名］と［場所］（プロジェクトを置くディレクトリ）を適宜入力する.
［ソリューション名］は［プロジェクト名］と同じものが自動入力される

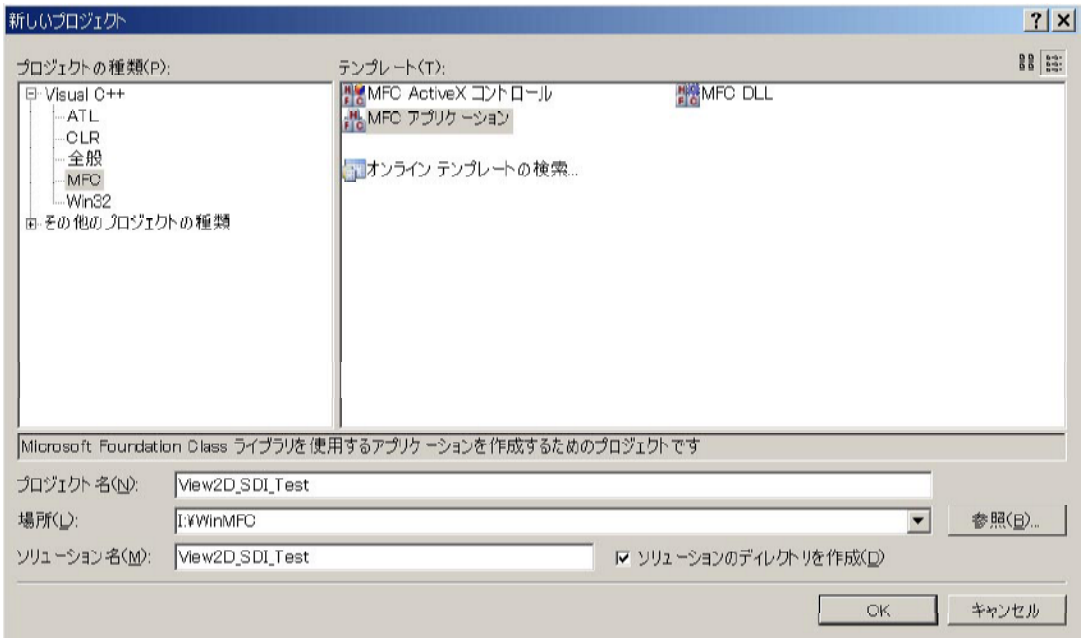

図３ Visual C++ / MFC のプロジェクト名（ソリューション名）と場所を指定

例えば、図３に示すように、
［プロジェクト名］:［View2D_SDI_Test］.
［場所］:［I:¥WinMFC］
［ソリューション名］:［View2D_SDI_Test］ を入力して「OK」.

―「MFC アプリケーションウィザード」のサブ画面（図４）が開かれ
左欄の［概要］より［アプリケーションの種類］を選択（クリック）してから、
右欄の［アプリケーションの種類］より［ｼﾝｸﾞﾙﾄﾞｷｭﾒﾝﾄ］と
［ﾄﾞｷｭﾒﾝﾄ ﾋﾞｭｰｱｰｷﾃｸﾁｬｻﾎﾟｰﾄ］を選択（チェック）してから「完了」.

→ ここで［ｼﾝｸﾞﾙﾄﾞｷｭﾒﾝﾄ］を選択することにより SDI 形式のプログラミングが可能になる.

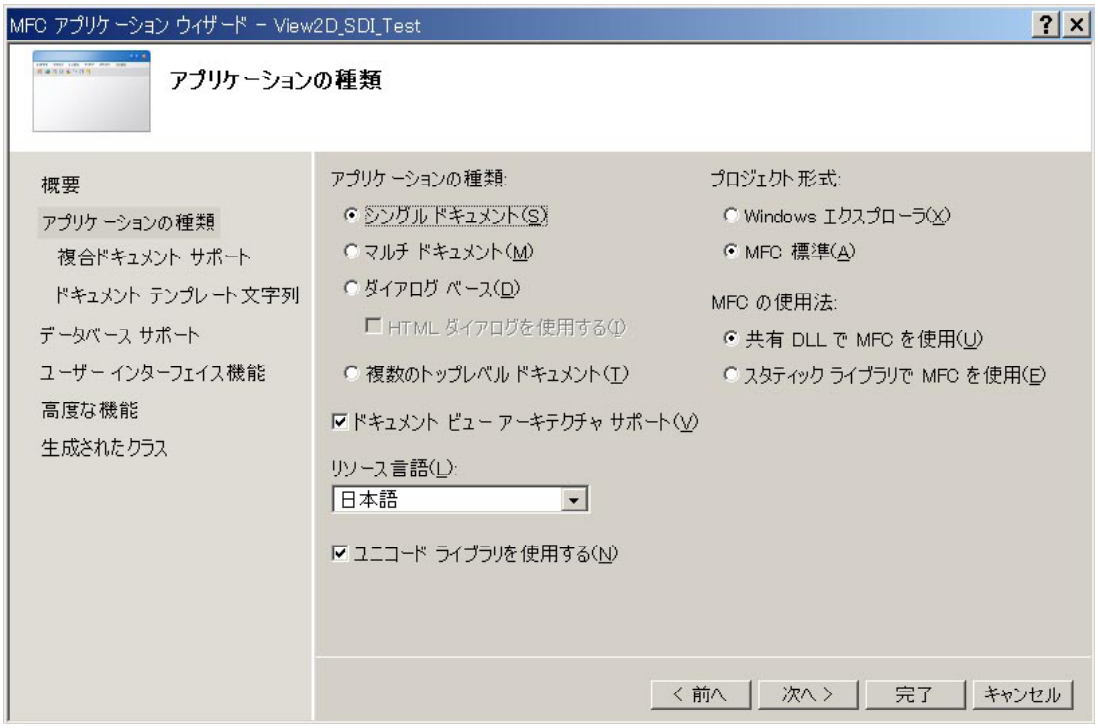

図４ SDI プログラム開発のためシングルドキュメントを指定

－「View2D_SDI_Test － Microsoft Visual Studio Academic Edition」の画面(図５)に戻り
左欄の［ソリューションエクスプローラ］タブを開き
ソリューション 'View2D_SDI_Test' (1 プロジェクト) が生成されていることを確認.
→ これにより、プロジェクト用のディレクトリ I:¥WinMFC¥View2D_SDI_Test が生成される.

** Windows プログラムを記述するためのソースファイルと構成クラスの確認
－「View2D_SDI_Test － Microsoft Visual Studio Academic Edition」の画面で
左欄の［ソリューションエクスプローラ］タブを開き
［View2D_SDI_Test］の＋マークをクリックして Windows プログラムの骨格（スケルトン）に必要なソースファイル、ヘッダーファイル、リソースファイル他が自動生成されていることを確認（図５）.
→ この中で、コード記述に必要な主なソースファイルは View2D_SDI_TestView.cpp と View2D_SDI_TestDoc.cpp であり、これらをダブルクリックすると、Windows プログラムのコード記述に必要なテキストエディタが右欄に開かれる.
－続いて、左欄の［クラスビュー］タブを開き
［View2D_SDI_Test］の＋マークをクリックして Windows プログラムの骨格（スケルトン）を構成しているクラスが自動生成されていることを確認（図６）.
→ この中で、例えばビュークラス CView2D_SDI_TestView をクリックすると、そのクラスで使われているメンバ変数やメンバ関数が下欄に表示される.
この CView2D_SDI_TestView クラスのメンバ関数 OnDraw(CDC* pDC) が実質的なエントリーポイントとなるため、この OnDraw() 関数内に最初のコードを記述して行くことになる. またこの例題では、CView2D_SDI_TestDoc クラスのメンバ関数 Serialize(CArchive &ar) にもコードを記述することになる.

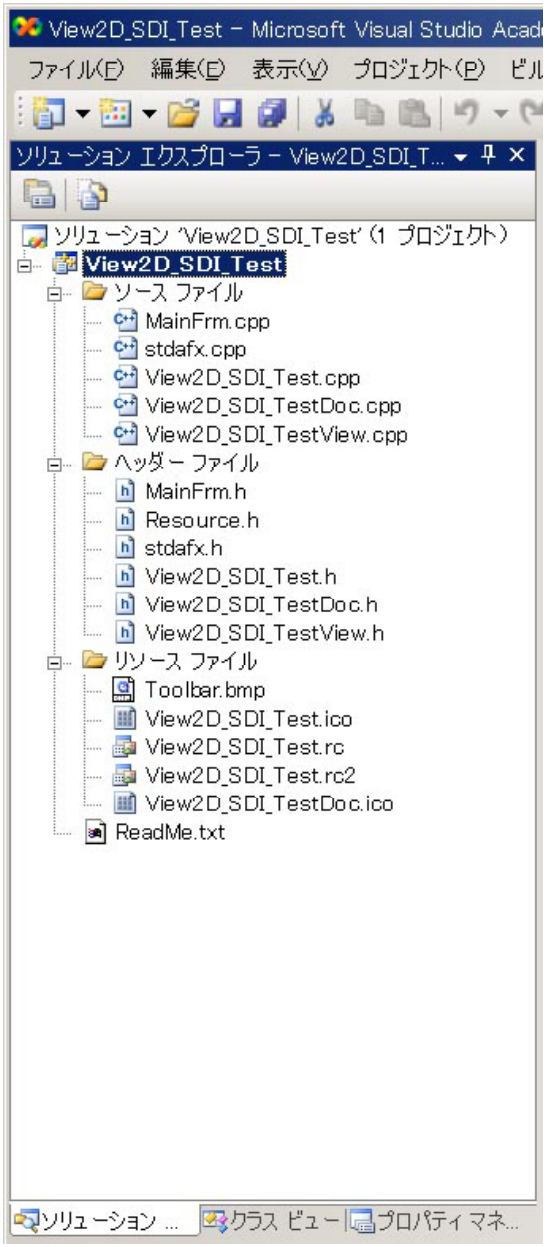

図５ 自動生成ファイルの確認

図６ 自動生成クラスの確認

** 以上の MFC アプリケーションウィザードによって

ビュークラス：	CView2D_SDI_TestView	← C***View
ドキュメントクラス：	CView2D_SDI_TestDoc	← C***Doc
アプリケーションクラス：	CView2D_SDI_TesApp	← C***App
ウィンドウフレームクラス：	CMainFrame	(*** はプロジェクト名)
ダイアログクラス：	CAboutDlg	

の５クラスが自動生成される.

この中でコード記述に必要なクラスは、

CView2D_SDI_TestView（と CView2D_SDI_TestDoc）であり、これらをダブルクリックすると、そのクラスを定義しているヘッダーファイル CView2D_SDI_TestView.h（と CView2D_SDI_TestDoc.h）が右欄に開かれる.

また、この CView2D_SDI_TestView クラスの中のメンバ関数 OnDraw() をダブルクリックすると、その関数を定義するソースファイル View2D_SDI_TestView.cpp が右

欄に開かれ、コード記述が可能になる.

** 既に作成済みの Windows プログラム（ソリューション：プロジェクトのコレクションを指す）を読み込む場合は、
「ファイル」－「開く」－「プロジェクト/ソリューション」を開いて作成済みのソリューションファイル ***.sln をロードする（*** はプロジェクト名）.
この例題では、プロジェクトディレクトリ I:¥WinMFC¥View2D_SDI_Test 内に生成された View2D_SDI_Test.sln を開く.

2． 「何もしない Windows プログラム」のビルド

** 以上の MFC アプリケーションウィザードにより Windows プログラムの骨格（スケルトン）が出来上がり、これ自体で「何もしない Windows プログラム」が完成される.

** この「何もしない Windows プログラム」をチェックするために、ビルドを実行する.
－「ビルド」－「View2D_SDI_Test のビルド」をクリック（図７）.

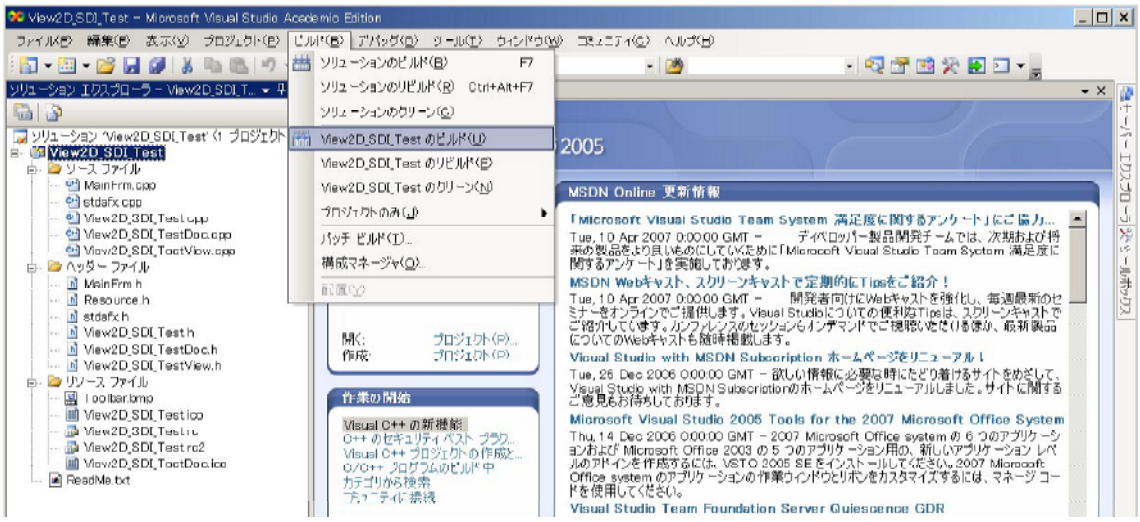

図７ ビルドの実行

－ウィンドウ下段の［出力］欄にエラーと警告のメッセージが表示される（図８）.
－プログラムにエラー（と警告）がなければ［ビルド：１正常終了］が表示され、プログラムの実行が可能になる.
→ ビルド（コンパイル＆リンク）に成功すると、プロジェクトディレクトリ I:¥WinMFC¥View2D_SDI_Test 内のフォルダ ¥debug 内に、実行可能ファイル View2D_SDI_Test.exe が生成される.

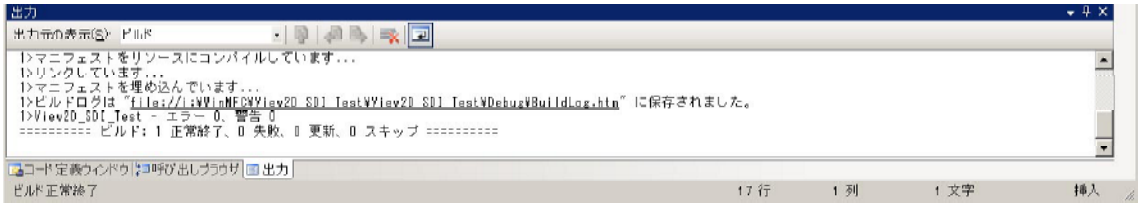

図８ ビルドの実行結果

** ビルドに成功したら、プログラムを実行する.
　－「デバッグ」－「デバッグなしで開始」を選択（図９）.

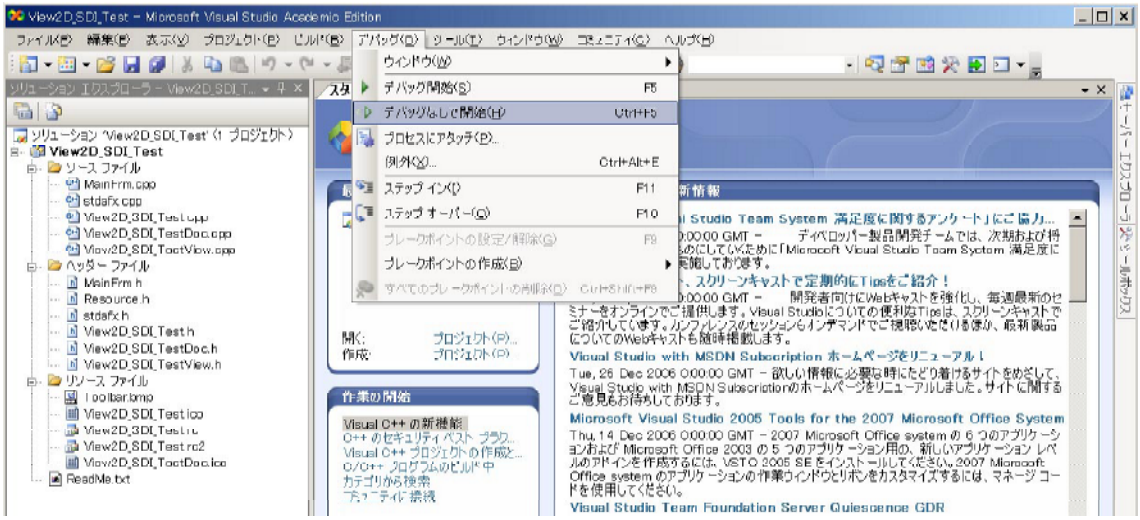

図９　プログラムの実行

　－この「何もしない Windows プログラム」を実行すると、ブランクのウィンドウズ画面が開かれる（図１０）.

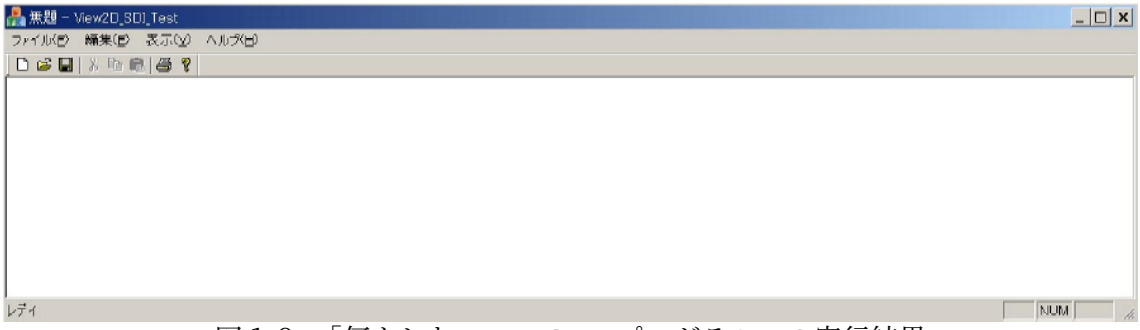

図１０　「何もしない Windows プログラム」の実行結果

** これが MFC アプリケーションウィザードにより生成された Windows プログラムの骨格（スケルトン）となる.

３．　Windows プログラム・ソースコードの記述

** まず最初に、ウィンドウズ画面上に "Hello! Image Media Lab." と "映像メディア研究室へようこそ！" を表示させる Windows プログラムを記述する.

　－左欄の［クラスビュー］タブを開き［View2D_SDI_Test］の＋マークをクリックして、Windows プログラムの骨格（スケルトン）を構成しているクラスの中から、CView2D_SDI_TestView クラスをクリック（図６）.
　－下欄に CView2D_SDI_TestView クラスに実装されているメンバ関数が表示され(図６）その中からプログラムの記述に必要なメンバ関数 OnDraw() をダブルクリック.
　－このメンバ関数 OnDraw() を定義するソースファイル View2D_SDI_TestView.cpp が右欄の「テキストエディタ」に開かれ（図１１）、コード記述（と編集）が可能になる.
　　→ SDI プログラムの場合は、この OnDraw() 関数が実質的なエントリーポイントとなり、この OnDraw() 関数内にソースコードを記述して行く.

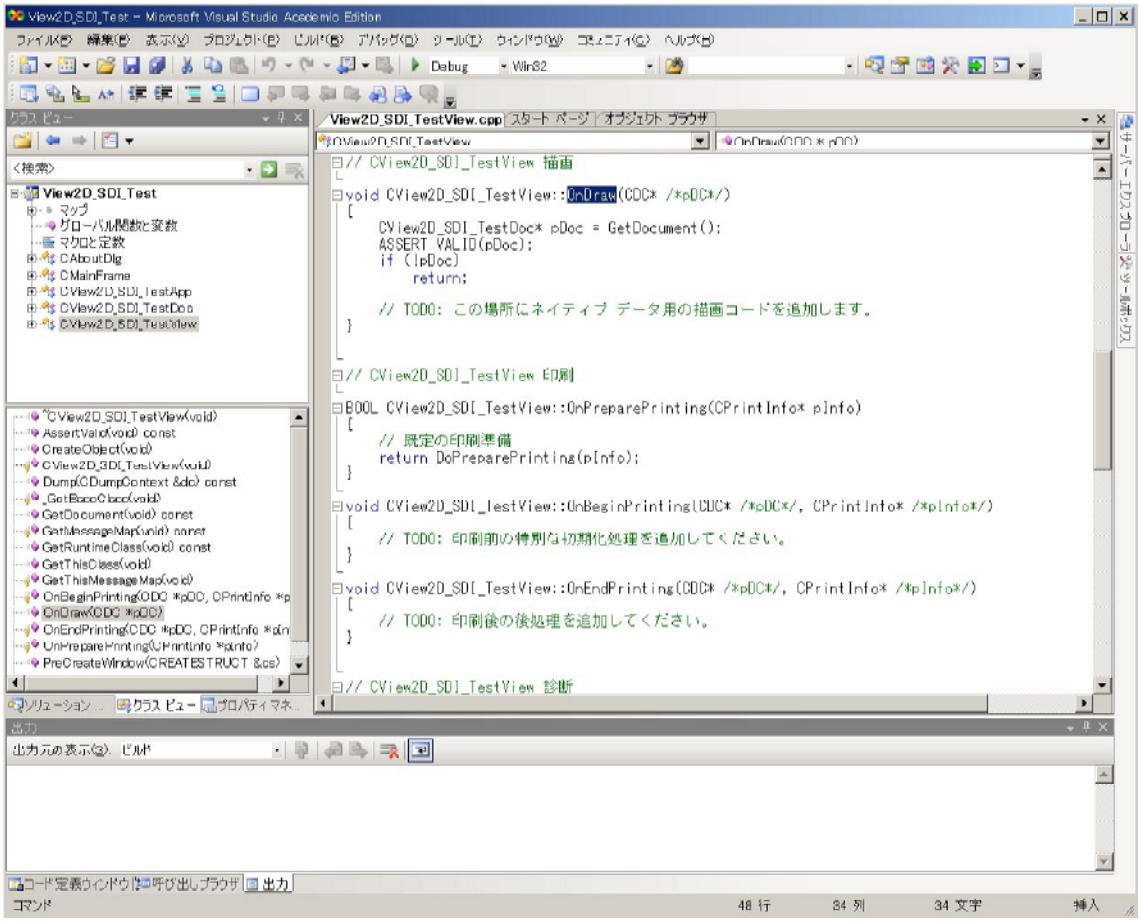

図１１　View2D_SDI_TestView.cpp のテキストエディタを開く

－右欄のテキストエディタ画面より、メンバ関数 OnDraw() を定義しているコード内に、以下のコードを追加記述する（図１２）.

```
// CView2D_SDI_TestView 描画
void CView2D_SDI_TestView::OnDraw(CDC* pDC)  // ** pDC を有効にしておく(注 1)
{
    CView2D_SDI_TestDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;
    // TODO: この場所にネイティブ データ用の描画コードを追加します。
    // ******** コード入力の開始 ****************************************
    pDC->TextOut(10, 10, "Hello! Image Media Lab.");
    pDC->TextOut(10, 40, "映像メディア研究室へようこそ！");
    // ******** コード入力の終了 ****************************************
}
```

→ // は MFC アプリケーションウィザードが生成したコメントを示し、
それとは別にユーザ（プログラマ）が適宜コメントを追加する場合は、
// ** で記述する.
// ** で挟まれる部分は、ユーザが独自にコード記述する部分を示す.
// **（注 1）OnDraw(CDC* /*pDC*/) の /* */ を外して pDC を有効にしておく.

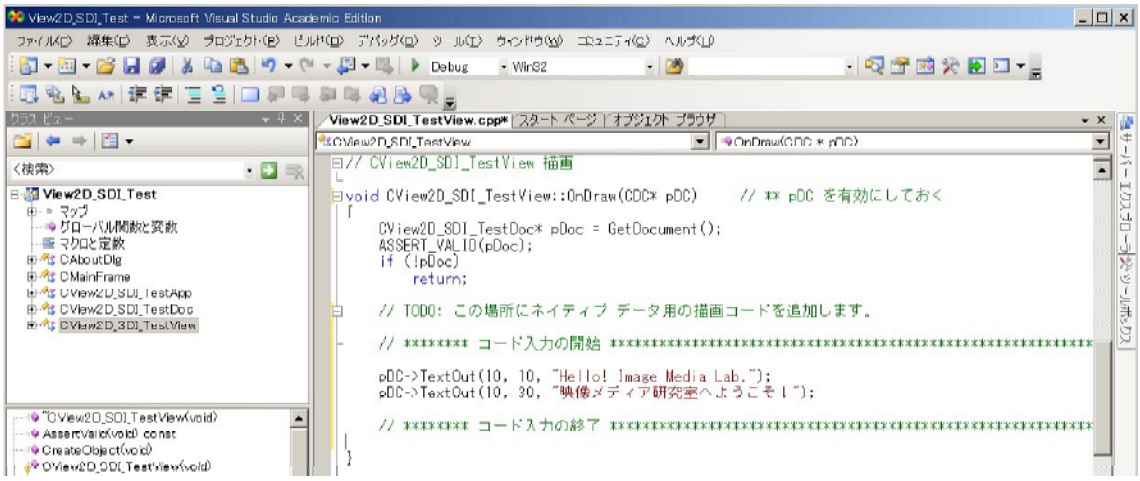

図１２　エントリーポイント OnDraw() にコード記述

** 以上の Windows プログラム（View2D_SDI_TestView.cpp のソースコード）の記述が終了したら、
－「ファイル」－「View2D_SDI_TestView.cpp の保存」をクリックして保存.
→ ただし、下記の「ビルド」－「View2D_SDI_Test のビルド」を実行すると、修正したファイルは自動的に上書き保存されるため、これは実行しなくてもよい.

４．　Windows プログラムのビルド

** Windows プログラムをチェックするために、最初のビルドを実行する.
－「ビルド」－「View2D_SDI_Test のビルド」を選択（図１３）.

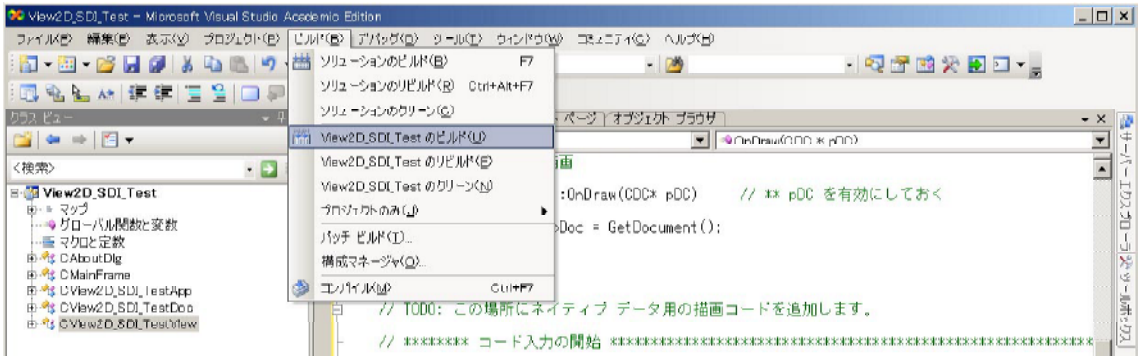

図１３　最初のビルドの実行

－プログラムにエラー(と警告) があった場合は、ウィンドウ下段の［出力］欄に、図１４に示すように、そのエラー因子と［ビルド：１失敗］が表示される.

図１４　ビルドの実行結果（失敗）

５．　Windows プログラムのデバッグ

** ビルド時にエラーが発生したら、エラー因子を取り除くためのプログラムの修正（デバッグ）を行う.

－上記例題では、

```
pDC->TextOut(10, 10, "Hello! Image Media Lab.");
pDC->TextOut(10, 40, "映像メディア研究室へようこそ！");
```

の引数にエラーがあり、

上段の「テキストエディタ」に戻り、以下（図１５）のように修正する.

```
CString str1, str2;
str1 = "Hello! Image Media Lab.";
str2 = "映像メディア研究室へようこそ！";
pDC->TextOut(10, 10, str1);          ← MFC クラスのメンバー関数
pDC->TextOut(10, 40, str2);          ←          〃
```

→ Visual C++ 6.0 / MFC（旧バージョン）では

```
CDC::TextOut(int, int, unsigned short *, int);
```

の使用が許されていたが,

Visual C++ 2005 / MFC（新バージョン）では

```
BOOL CDC::TextOutW(int,int,const CString &);
#define TextOut TextOutW;
```

に限られている.

** このように Visual C++ のバージョンによって MFC 等の互換性が無くなることがあるので注意が必要.

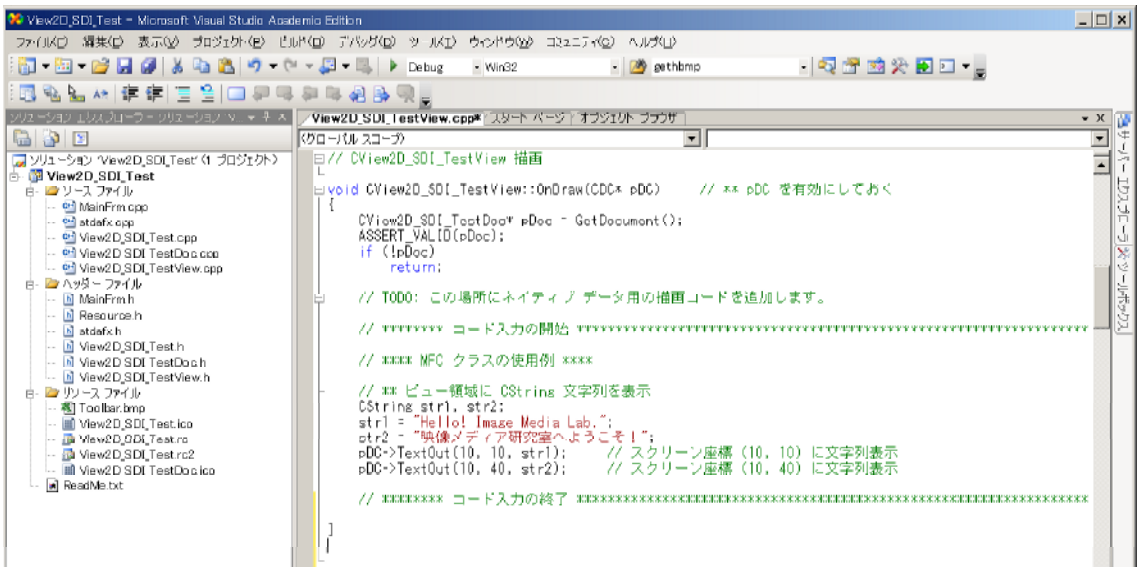

図１５　プログラムの修正

** プログラムの修正後、

－「ビルド」－「View2D_SDI_Test のビルド」を再度実行する..

これを繰り返し、修正によりエラーがなくなれば、ビルドに成功し、

下段の［出力］欄に［ビルド：１正常終了］が表示される.

図１６　ビルドの再実行結果（成功）

－ビルドに成功すると、プロジェクトディレクトリ I:¥WinMFC¥View2D_SDI_Test 内のフォルダ ¥debug に実行可能ファイル View2D_SDI_Test.exe が生成される.

６． Windows プログラムの実行

** ビルドに成功したら、プログラムを実行する.
－「デバッグ」－「デバッグなしで開始」を選択（図１７）.

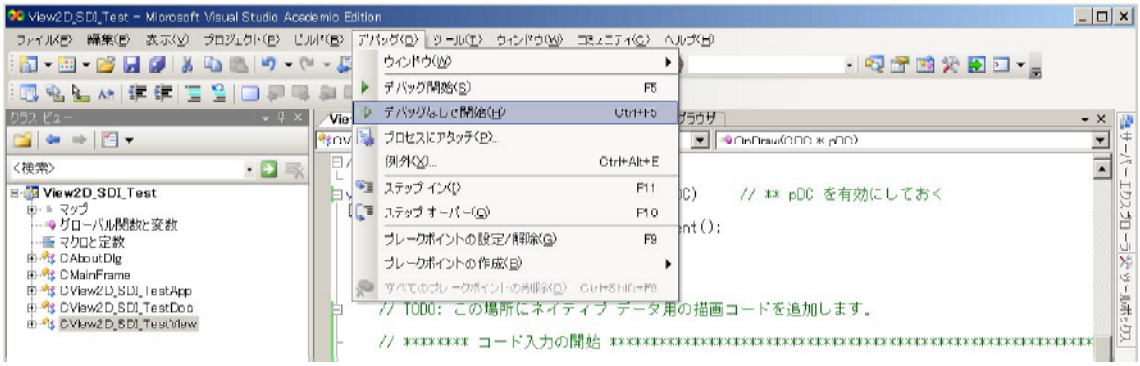

図１７ プログラム（View2D_SDI_Test.exe）の実行

－この View2D_SDI_Test プログラムを実行すると以下のウィンドウ画面が開かれる.

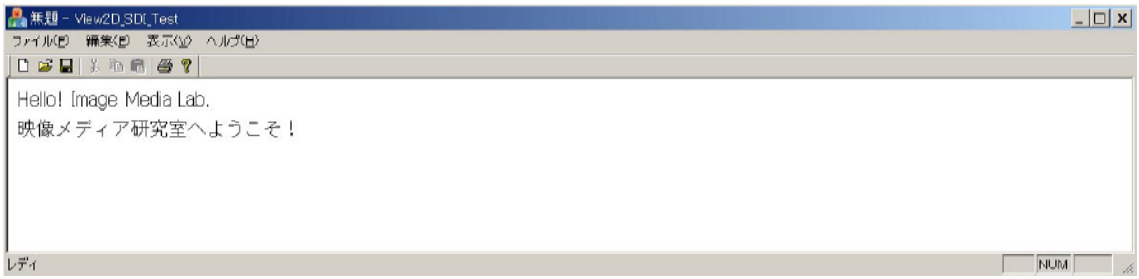

図１８ SDI 形式 Windows プログラムの実行結果

** この例題は SDI(Single Document Interface) を指定して作成したプログラムである.
このため図１８のように単一のウィンドウ画面で構成される.
－通常のウィンドウズアプリケーションと同様に「×」をクリックすればプログラムを終了.

** 以上の MFC アプリケーションウィザードにより構築される Windows プログラムに対し、同じ Visual C++ の環境下でも、Win32 Console Application により構築されるＣプログラムでは、以下の MS-DOS 画面が表示される.

図１９ Win32 Console Application によるＣプログラムの実行結果

** ちなみに、この C プログラムのソースコードは以下の通り.

```
/*  C テストプログラム（c_test.c）  */
#include <stdio.h>
#include <conio.h>
void main(void)
{
    printf("Hello! Media System Lab. ¥n");
    printf("映像メディア研究室へようこそ！¥n");
    getch();
}
```

** 次に、映像メディア研究室で開発した２次元描画クラス View2D を用いて、ビットマップ画像を表示する Windows プログラムを記述する.

７．　ユーザ定義２次元描画クラス View2D のプロジェクトへの追加

－まず最初に、ユーザ定義の２次元描画クラス View2D の
　　ヘッダファイル View2D.h
　　ソースファイル View2D.cpp
をソリューションディレクトリ I:¥WinMFC¥View2D_SDI_Test¥View2D_SDI_Test 内にコピーしておく.

－続いて、「プロジェクト」－「既存項目の追加」を開き（図２０）

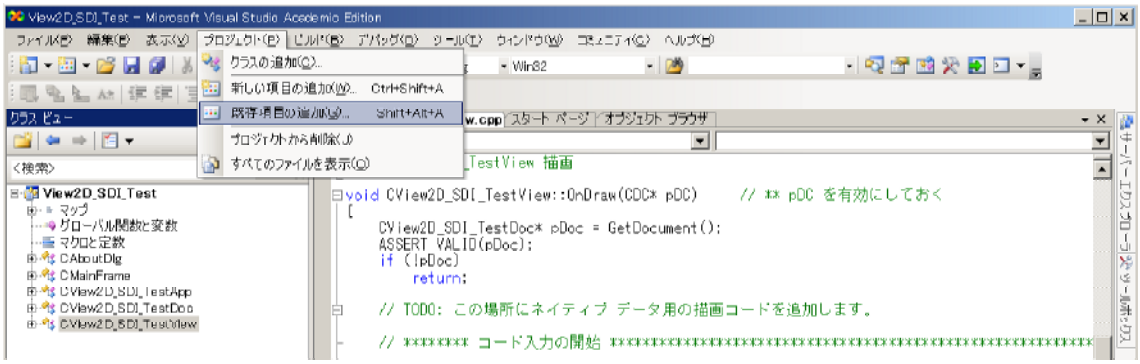

図２０　新しいユーザ定義クラスをプロジェクトに追加

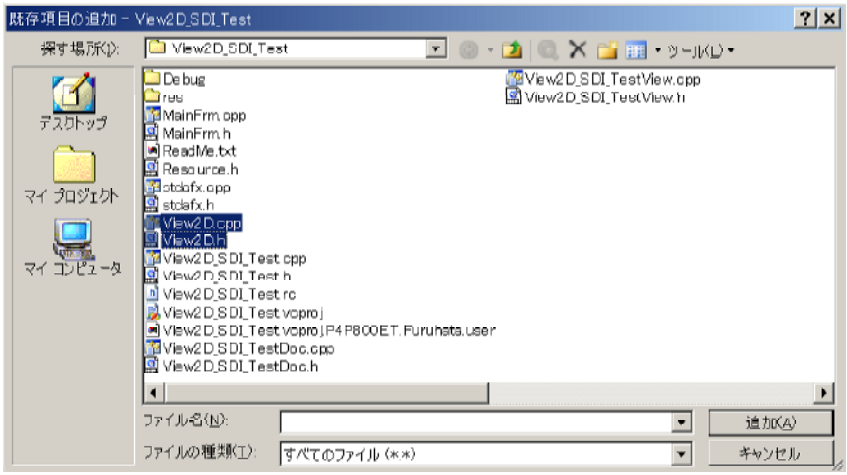

図２１　２次元描画クラス View2D をプロジェクトに追加

－［既存項目の追加］のサブ画面（図２１）で、
ソリューションディレクトリ I:¥WinMFC¥View2D_SDI_Test¥View2D_SDI_Test 内にコピーしておいた２次元描画クラス View2D 用のファイル（View2D.h, View2D.cpp）を選択して「追加」をクリック.

８． Windows / View2D プログラミングの開発準備

－［クラスビュー］タブを開き、ユーザ定義の２次元描画クラス View2D がプロジェクトに追加されたことを確認（図２２左欄）.
－続いて、この２次元描画クラス View2D をビュークラス C**View で使用できるようにするために、ビュークラス C**View が実装されているソースファイル View2D_SDI_TestView.cpp を開き、そのトップで View2D.h を以下のようにインクルードしておく（図２２右欄）

```
// View2D_SDI_TestView.cpp : CView2D_SDI_TestView クラスの実装
・・・・・・・・
#include   "View2D.h"
```

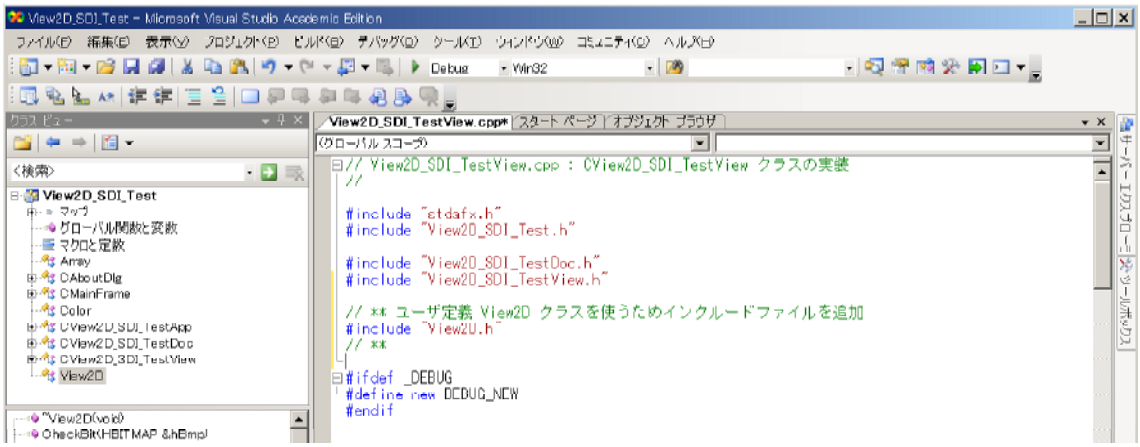

図２２ View2D クラスのプロジェクト追加確認と
ビュークラスへのインクルードファイルの追加記述

－ View2D.h のインクルードを終了したら、一度「View2D_SDI_Test のビルド」を実行し、プログラムにエラーがないことを確認しておく.
→ もし、エラー(と警告)があったら View2D.h, View2D.cpp を修正しておかなければならない.

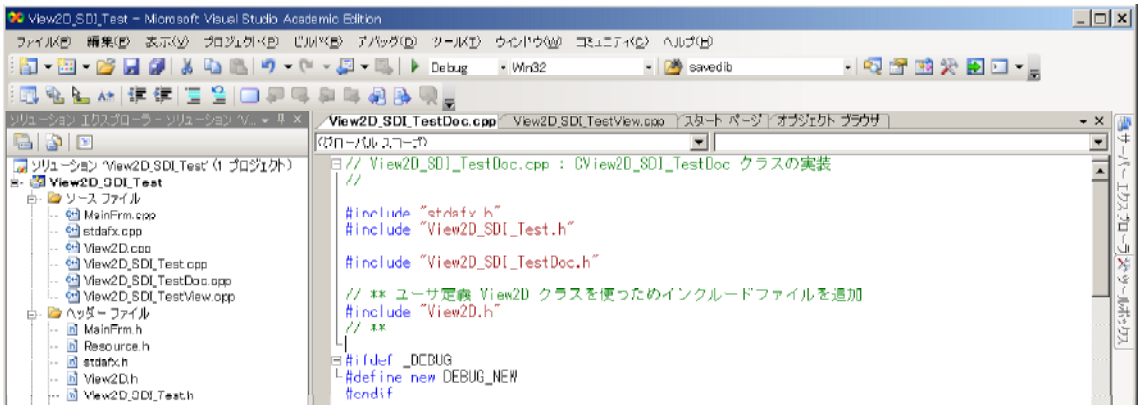

図２３ View2D.h のドキュメントクラスへのインクルード追加記述

－このプログラム例題では、ドキュメントクラス C**Doc でも、この２次元描画クラス View2D を使うため、ドキュメントクラス C**Doc が実装されているソースファイル CView2D_SDI_TestDoc.cpp を開き、そのトップで View2D.h をインクルードしておく（図２３右欄）

```
// View2D_SDI_TestDoc.cpp : CView2D_SDI_TestDoc クラスの実装
・・・・・・・・
#include   "View2D.h"
```

－更に、この例題では、ドキュメントクラス C**Doc でビットマップ画像(**.bmp）を扱えるようにするために、C**Doc クラスを定義しているヘッダファイル View2D_SDI_TestDoc.h を開き、ビットマップ画像用のメンバー変数 m_hbmpw, m_hbmpr（ビットマップ情報を格納する DIBSECTION のハンドル）を C**Doc クラスの public （外部からアクセス可能な）変数として、以下のように追加登録する（図２４右欄）

```
// View2D_SDI_TestDoc.h : CView2D_SDI_TestDoc クラスのインターフェイス
・・・・・・・・
// 操作
public:
    HBITMAP m_hbmpw;          ← 書き込み用ビットマップハンドルの登録
    HBITMAP m_hbmpr;          ← 読み取り用ビットマップハンドルの登録
```

－ドキュメントクラス C**Doc を開き、メンバー変数 m_hbmpw, m_hbmpr が登録されていることを確認（図２４左欄）

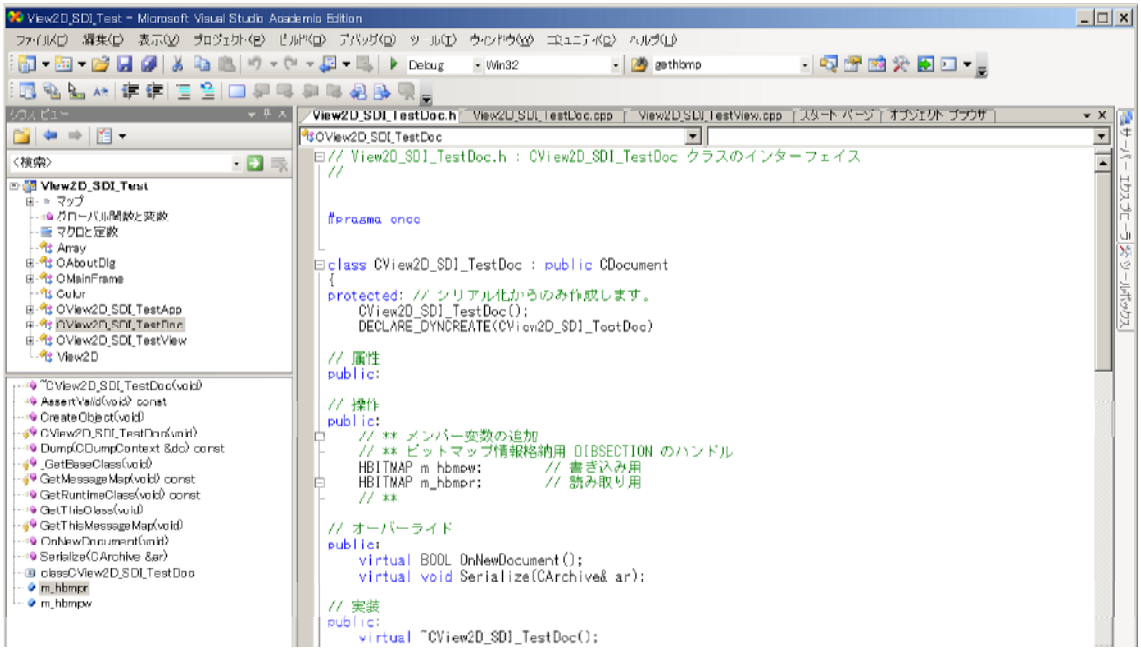

図２４　ドキュメントクラス・メンバー変数 m_hbmpw, m_hbmpr の追加登録

－以上の View2D.h のインクルードとメンバー変数 m_hbmpw, m_hbmpr の登録を終了したら、「ビルド」－「View2D_SDI_Test のビルド」を実行し、プログラムにエラーがないことを確認しておく.

** 以上で、２次元描画クラス View2D を用いた Windows プログラミングの準備が完了したので、いよいよ例題のプログラムのソースコードを記述して行くことになる.
** ここまでのソースコードは Windows/View2D プログラミングの雛形（スケルトン）として利用可能.

９．　Windows / View2D プログラム・ソースコードの記述

** この例題では、ウィンドウメニューの「ファイル」－「開く」より選択した任意のビットマップ画像（xxx.bmp）をウィンドウのビュー領域左側に表示させ、続いて、そのビットマップ画像をモノクロ画像に変換してから同じビュー領域右側に表示させた後で、メニューの「ファイル」－「名前を付けて保存」により、任意のファイル名（yyy.bmp）で任意のフォルダに保存できるようにする．

－ウィンドウメニューからのファイル操作は、ドキュメントクラス C**Doc にて行わせることができ、そのため最初に［クラスビュー］タブを開き CView2D_SDI_TestDoc クラスのメンバー関数 Serialize(CArchive &ar) をダブルクリック（図２５左欄）
　→ 右欄に Serialize(CArchive &ar) 関数が表示され、コード入力が可能となる．

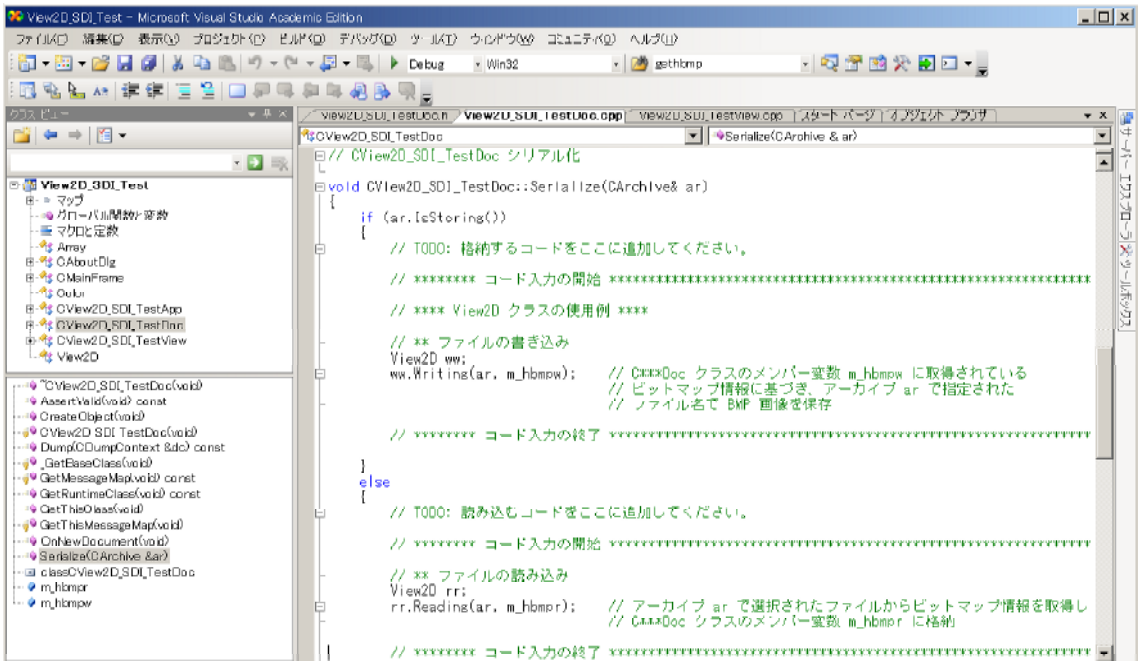

図２５　ドキュメントクラスのメンバー関数 Serialize() にコード記述

－この Serialize(CArchive &ar) 関数内に、以下のコードを記述する(図２５右欄)

```
// CView2D_SDI_TestDoc シリアル化
void CView2D_SDI_TestDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        View2D ww;                      ← View2D クラス ww の宣言
        ww.Writing(ar, m_hbmpw);        ← View2D クラスのメンバー関数 Writing
    }                                       の呼び出し
    else
    {
        View2D rr;                      ← View2D クラス rr の宣言
        rr.Reading(ar, m_hbmpr);        ← View2D クラスのメンバー関数 Reading
    }                                       の呼び出し
}
```

　→ 関数の詳細については、View2D.h / View2D.cpp のドキュメントと

ソースファイル View2D_SDI_TestDoc.cpp のコメントを参照.

－ビュー領域での描画は、ビュークラス C**View で行わせることができる.
ここでは［クラスビュー］タブを開き CView2D_SDI_TestView クラスのメンバー関数 OnDraw(CDC* pDC) をダブルクリック（図２６左欄）
→ 右欄に OnDraw(CDC* pDC) 関数が表示され、コード入力が可能となる.

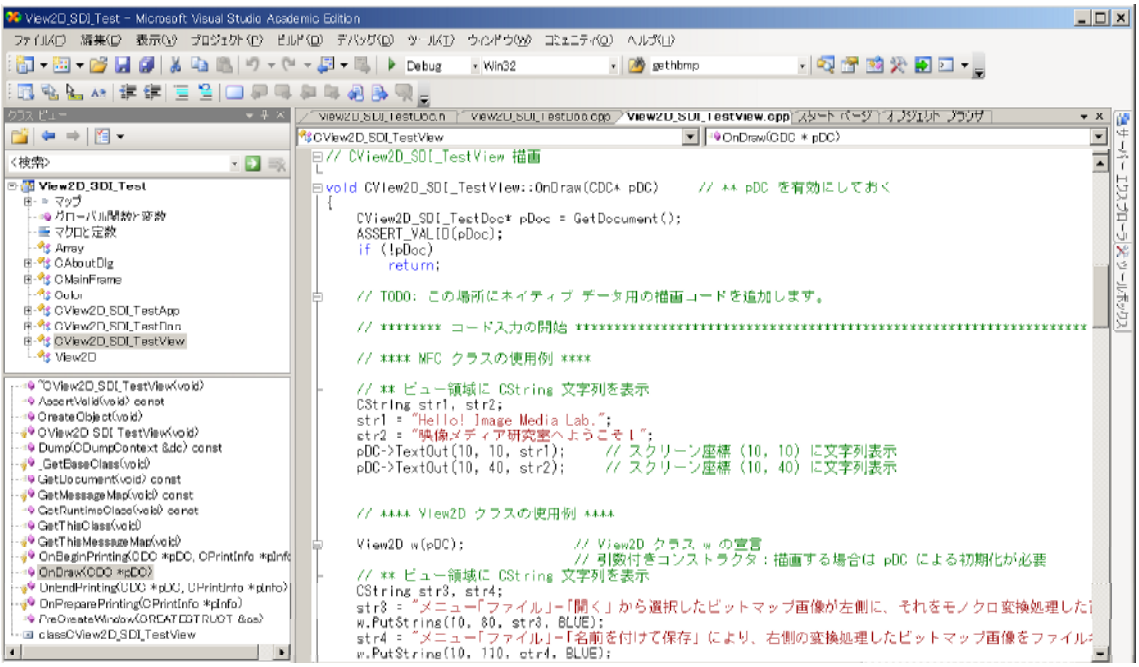

図２６ ビュークラスのメンバー関数 OnDraw() にコード記述

－この OnDraw(CDC* pDC) 関数内に、以下のコードを追加記述する(図２６右欄)

```
// CView2D_SDI_TestView 描画
void CView2D_SDI_TestView::OnDraw(CDC* pDC)        // ** pDC を有効にしておく
 {
      ........
      // ******** コード入力の開始 **********************************************
      // **** View2D クラスの使用例 ****
      View2D w(pDC);                                    ← View2D クラス w の宣言
      // ** ビュー領域に CString 文字列を表示
      CString str3, str4;
      str3 = "メニュー「ファイル」－「開く」から選択したビットマップ画像が左側に、
      それをモノクロ変換処理した画像が右側に表示されます.";
      w.PutString(10, 80, str3, BLUE);              ← View2D クラスのメンバー関数
      str4 = "メニュー「ファイル」－「名前を付けて保存」により、右側の変換処理した
      ビットマップ画像をファイル名を指定して保存出来ます.";
      w.PutString(10, 110, str4, BLUE);             ←          〃
      // ** ドキュメントクラスで読み込んだ BMP 画像を表示する **
      // ビュー領域に char* 文字列を表示
      w.PutString(15, 155, "処理前の画像を表示します", RED);    ←        〃
      w.PutString(605,155, "処理後の画像を表示します", RED);    ←        〃
      // ドキュメントクラスのメンバー変数の取得
      HBITMAP hbmp = pDoc->m_hbmpr;                     ← MFC クラス
      // イメージサイズの取得
      int lx, ly;
```

```
    CSize imagesize;
    imagesize = w.GetImageSize(hbmp);              ← View2D クラスのメンバー関数
    lx = imagesize.cx;
    ly = imagesize.cy;
    if(lx < 0 || ly < 0) {
        lx = 0;
        ly = 0;
    }
    // ビュー座標に画像表示
    w.DrawBmp(10, 150, hbmp);                      ← View2D クラスのメンバー関数
    w.PutString(15, 155, "処理前の画像", RED);     ←                  〃
    // 画像処理用一次元動的配列を生成
    double *rr = new double[lx*ly];
    double *gg = new double[lx*ly];
    double *bb = new double[lx*ly];
    // 配列にイメージデータをロード
    w.LoadBmp(hbmp, rr, gg, bb);
    // 一次元配列上で画像処理
    int i, j;
    double pr, pg, pb, px;
    for(j=0; j<ly; j++) {
        for(i=0; i<lx; i++) {
            pr = rr[i+lx*j];
            pg = gg[i+lx*j];
            pb = bb[i+lx*j];
            px = (pr + pg + pb) / 3.0;
            rr[i+lx*j] = px;
            gg[i+lx*j] = px;
            bb[i+lx*j] = px;
        }
    }
    // 一次元配列上の画像を表示
    HBITMAP hbmp1;
    w.SaveDib(imagesize, rr, gg, bb, hbmp1);       ← View2D クラスのメンバー関数
    w.DrawBmp(600, 150, hbmp1);                    ←                  〃
    w.PutString(605,155, "処理後の画像", RED);     ←                  〃
    // ドキュメントクラスでのファイル名指定保存に備える
    pDoc->m_hbmpw = hbmp1;                         ← MFC クラス
    // 処理した一次元配列上の画像を直接ファイルに保存
    CString savefile;
    savefile = "Img00.bmp";
    w.SaveBmp(savefile, imagesize, rr, gg, bb);    ← View2D クラスのメンバー関数
    // ファイル保存した画像の確認
    w.ReadDrawBmp(savefile, 600, 600);             ←                  〃
    w.PutString(605,605, "保存後の画像", BLUE);    ←                  〃
    delete [] rr, gg, bb;
    // ******** コード入力の終了 **********************************************
}
```

→ 関数の詳細については、View2D.h / View2D.cpp のドキュメントと
ソースファイル View2D_SDI_TestView.cpp のコメントを参照.

－以上でソースコードの入力は完了.

** Windows プログラムをチェックするために、ビルドを実行する.
 －「ビルド」－「View2D_SDI_Test のビルド」を実行.
** ビルドに成功したら、いよいよプログラムを実行する.
 －「デバッグ」－「デバッグなしで開始」を実行.
 －この例題では、プログラムを実行すると、まず最初にプログラムのコメントを表示するウィンドウが開かれる（図２７）

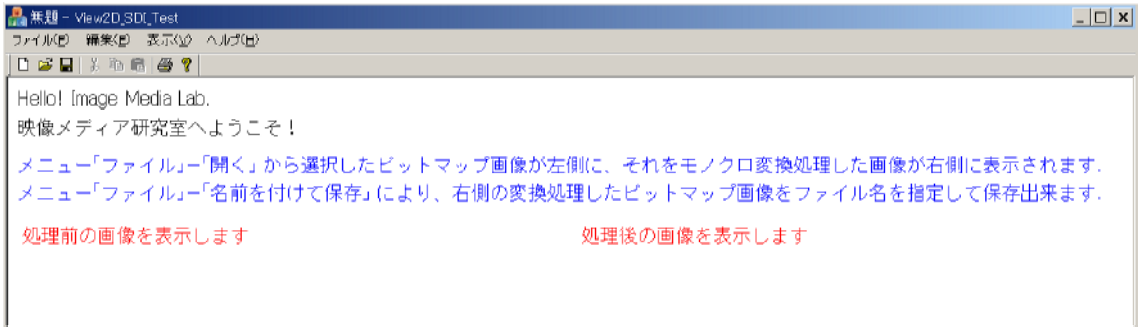

図２７ Windows/View2D プログラム(View2D_SDI_Test.exe）の最初の実行画面

 －コメントの指示に従い、メニュー「ファイル」－「開く」を選択（クリック）
 －サブ画面「開く」が開かれ、処理したいビットマップ画像を、適宜選択してから「開く」をクリック（図２８）

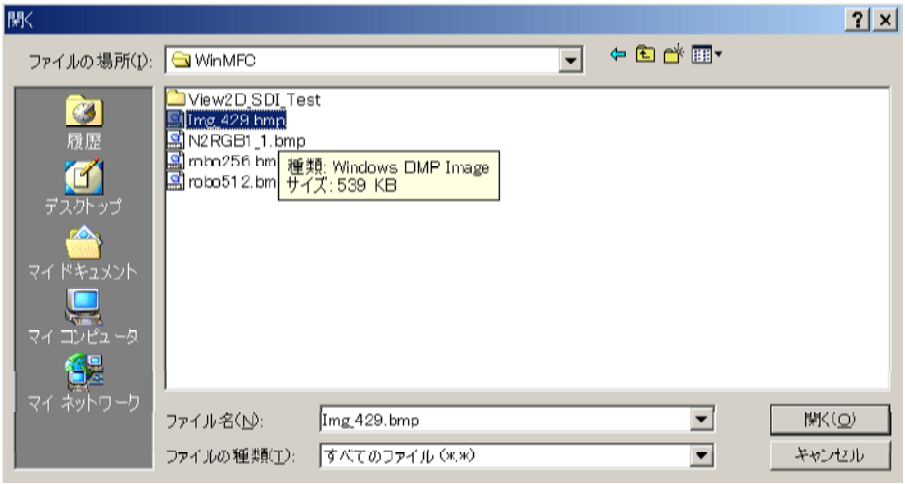

図２８ 処理させるビットマップ画像（**.bmp）を選択する

 －ここでは、ビットマップ画像 Img_429.bmp が選択され、「処理前の画像」（原画像）として左側に表示され、それをモノクロに変換した画像が「処理後の画像」（処理画像）として右側に表示される.
 更に、その処理された画像は原画像と同じディレクトリ内にファイル名 Img00.bmp で自動保存された後、「保存後の画像」として右下に表示される（図２９）
 －なお、このプログラム例では、メニュー「ファイル」－「名前を付けて保存」を選択すると「名前を付けて保存」のサブ画面（図３０）が表示され、保存先のディレクトリを指定して、適宜名前を付けて保存させることができる.

図２９ Windows/View2D プログラムの実行で表示されるビットマップ画像

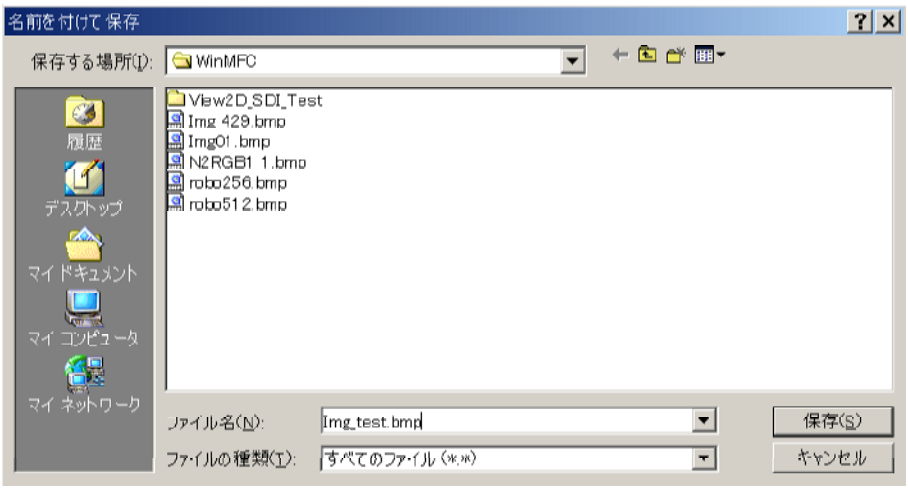

図３０ 処理画像の保存先ディレクトリとファイル名を指定する

１０．Visual CG++ / View2D プラットホームの特徴

** Windows プログラムの開発には、Microsoft 社の提供する MFC（Microsoft Foundation Class)を利用したプログラミングが不可欠となっている.
しかし、この MFC は利便性に優れ拡張性に富む反面、その膨大なライブラリと複雑な階層構造を有するがために、プログラミング開発の初心者には難解なものとなっており、導入の大きな障害になっている.

** 例えば「ウィンドウに画像表示するだけの簡単な Windows プログラミング」を考えると、MFC を利用したプログラムコードは以下のようになる.

```
// **** MFC 利用の Windows プログラミング例 ****
// ** filepath のビットマップ画像ファイルを読み込み、そのビットマップ情報をメンバー
//    変数 mhBmp に取得
    CFile file;
    if(!file.Open(filepath, CFile::modeRead | CFile::typeBinary)) {
    return;
    }
    BITMAPFILEHEADER bmfh;
    if(file.Read(&bmfh, sizeof(bmfh)) != sizeof(bmfh)) {
        AfxThrowArchiveException(CArchiveException::endOfFile);
    }
    if(bmfh.bfType != 0x4d42){
        AfxThrowArchiveException(CArchiveException::badIndex);
    }
    UINT sz = bmfh.bfSize - sizeof(bmfh);
    BYTE *buf = new BYTE[sz];
    try{
        if(file.Read(buf, sz) != sz){
            AfxThrowArchiveException(CArchiveException::endOfFile);
        }
        BITMAPINFOHEADER *pbmih = (BITMAPINFOHEADER *)buf;
        if(pbmih->biCompression != BI_RGB) {
            AfxThrowArchiveException(CArchiveException::badIndex);
        }
        RGBQUAD *pcolortbl = NULL;
        if(pbmih->biBitCount <= 8) {
            pcolortbl = (RGBQUAD *)(buf + pbmih->biSize);
        }
        BYTE *pdata = buf + bmfh.bfOffBits - sizeof(bmfh);
        BYTE *pbits;
        mhBmp = ::CreateDIBSection(0, (BITMAPINFO *)pbmih, DIB_RGB_COLORS,
                (void **)&pbits, 0, 0);
        if(mhBmp == 0 || pbits == NULL) {
            AfxThrowMemoryException();
        }
        long szImage = (pbmih->biWidth * pbmih->biBitCount + 31)
                        / 32 * 4 * pbmih->biHeight;
        memcpy(pbits, pdata, szImage);
        delete []buf;
```

```
    }
    catch(CException *e){
        delete []buf;
        throw e;
    }
// ** メンバー変数 mhBmp に取得されているビットマップ情報に基づき
//     ビュー座標（x, y）に BMP 画像を描画
    CBitmap *pbmp = CBitmap::FromHandle(mhBmp);
    CDC dcMem;
    dcMem.CreateCompatibleDC(pdc);
    CBitmap *pbmpOrig = dcMem.SelectObject(pbmp);
    CSize ImageSize;
    ImageSize = GetImageSize();
    pdc->BitBlt(x, y, ImageSize.cx, ImageSize.cy, &dcMem, 0, 0, SRCCOPY);
    dcMem.SelectObject(pbmpOrig);
    dcMem.DeleteDC();
    ::DeleteObject(mhBmp);
```

** このように MFC クラス利用の Windows プログラミングでは、ステートメント数にして 51 行にも及び、しかも難解な MFC が多用されているためプログラミング初心者には、恐らく理解不能となるだろう.

** これに対し View2D クラスを利用した Windows プログラミングでは、そのプログラムコードは次のように大幅に縮小される（ステートメント数にして 2 行）

```
// **** View2D 利用のプログラミング例 ****
// ** filepath のビットマップ画像ファイルを読み込み、ビュー座標（x, y）に描画
    View2D w(pDC);
    w.ReadDrawBmp(filepath, x, y);
```

** このように、新開発の Visual CG++ / View2D プラットホームでは、Windows プログラミングに不可欠であった MFC は View2D クラスの中にモジュール化されて覆い隠されるため、ユーザ（プログラミング初心者）はこの難解な MFC から解放され、MFC を意識することなく、従来の C / C++ プログラミングの感覚で Windows プログラムを構築することが可能となる.

→ これらの詳細は、
「降旗･鈴木･本田･田村："Windows / CG プログラミング・プラットホームの一考察", 画像電子学会誌, Vol.36, No.2, pp.131-140（2007）」
を参照されたい.