クラス： wxDocument, wxView, wxDocTemplate, wxDocManager, wxDocParentFrame, wxDocChildFrame, wxDocMDIParentFrame, wxDocMDIChildFrame, wxCommand, wxCommandProcessor

ドキュメント／ビューフレームワークは、多くの種類のアプリケーションをビルドするために要求されるコードを劇的にシンプルにすることができるため、多くのアプリケーションフレームワークで見受けられる。

その考えは、アプリケーションを、データの保存やデータへのインターフェース非依存な操作のためのドキュメントという観点と、視覚化とデータの操作のためのビューという観点でアプリケーションをモデル化することができる。ドキュメントは、与えられたストリームオブジェクトの入出力がどのように扱うかを管理し、ビューは、物理的なウィンドウからの入力受付やドキュメントデータを取り扱う。ドキュメントのデータが変更されれば、全てのビューはその変更を更新すべきである。

フレームワークは、このモデルをベースに、多くのユーザインタフェース要素を提供する。一旦自身のクラスとそれらの関係を定義すれば、フレームワークは、ファイル選択の表示、ファイルクローズ操作、変更を保存するかどうかの問い合わせ、適切な（場合によってはデフォルトの）コードへのメニューコマンドの割り当て、印刷／印刷プレビューや undo/redo でさえも取り扱う。フレームワークは高度にモジュール化されており、デフォルトの動作以上の動作をするよう、関数や変数のオーバーライドや多重定義を認められている。

ドキュメント／ビューフレームワークの元でアプリケーションを作成するために必要な、大まかな手順がある。：

1. 自身のドキュメントクラスとビュークラスを定義し、最低限のメンバ関数、例えば、入出力、描画や初期化をオーバーライドする。
2. ビューで必要とされる幾つかのサブウィンドウ（例えば、スクロールウィンドウ）を定義する。ビューやドキュメントへのイベントを送らなければならず、例えば、OnPaint は wxView::OnDraw へ送られなければならない。
3. どのようなウィンドウスタイルを使用するか決定する：Microsoft の MDI（１つのフレームによって囲まれた複数のドキュメントの子フレーム）、SDI（ドキュメントごとに独立したフレーム）、あるいは、単一ウィンドウ（Window Write に見られるような、１度に１つのドキュメントが開く）
4. 適切な wxDocParentFrame と wxDocChildFrame クラスを使用する。wxApp:OnInit で wxDocParentFrame インスタンスを作成し、ビューを初期化するときに wxDocChildFrame（単一ウィンドウで無い場合）を作成する。標準メニュー識別子（wxID_OPEN、wxID_PRINT など）を使用してメニューを作成する。
5. wxApp::OnInit の始めに１つの wxDocManager インスタンスを作成し、ドキュメントとビューの間の関係を定義するために幾つかの wxDocTemplate インスタンスを定義する。単純なアプリケーションでは、wxDocTemplate はただ１つだけである。

undo/redo を実装したい場合には、wxCommand を自身のクラスに継承し、直接コードを実行するのではなく、wxCommandProcessor::Submit を使用する。フレームワークは、wxID_UNDO と wxID_REDO メニューアイテムがビューメニューで定義されている場合には、必要に応じて undo/redo の関数を呼ぶことを管理する。

デフォルトのフレームワーク動作を超えて使用するための例を挙げる。：

- 異なる undo/redo やコマンド履歴エディタを定義するために、wxDocument::OnCreateCommandProcessor をオーバーライドする。
- 複数ページのドキュメントを使用するために、継承された wxPrintout クラスのインスタンスを作成し wxView::OnCreatePrintout をオーバーライドする。
- 異なるファイルセレクタを使用するために、wxDocManager::SelectDocumentPath をオーバーライドする。
- 開くことができるドキュメントの最大数と undo コマンドの最大数を制限する。

フレームワーク機能を起動する際に特筆すべきは、メニューで既に定義された wxWidgets のコマンド識別子の幾つか、または、全てを使用する必要がある、ということである。

wxPerl note: The document/view framework is available in wxPerl. To use it, you will need the following statements in your application code:

```
use Wx::DocView;
use Wx ':docview';   # import constants (optional)
```

---

wxDocument 概要

クラス : wxDocument

wxDocument クラスは、アプリケーションのファイルベースデータの雛形を作るために使用される。それは、wxView, wxDocTemplate と wxDocManager クラスと共に wxWidgets でサポートされているドキュメント／ビュー フレームワークの一部である。

一連のメニューコマンド（オープン、保存、Save-as など）が自動的にサポートされるため、このフレームワークを使用することにより、多くのユーザインタフェースプログラムを省略することができる。プログラマは、フレームワークのために、最小限のクラスとメンバ関数を必要に応じて定義するだけである。データと、データを表示・編集することの意味は、このモデルでは、明確に切り分けられており、また、同じデータに対して複数のビューを持つ概念がサポートされている。

ドキュメント／ビューモデルは、多くのアプリケーションに適用できるが、全てのアプリケーションに向いている訳ではない。例えば、ビュー関連の関数を呼び出すことが無かったり、ファイルのオープン、編集、保存などの機能が不要な、単純なファイル変換ユーティリティである。しかし、恐らく大部分のアプリケーションはドキュメントベースである。

See the example application in samples/docview.

wxDocument 抽象クラスを使用するために、新しい派生クラスを作成し、少なくともメンバ関数 SaveObject と LoadObject をオーバーライドしなければならない。SaveObject と LoadObject は、ドキュメントを保存、または読み込む際にフレームワークから呼ばれる。

フレームワークに対して、要求に応じてドキュメントオブジェクトを作成することを許可するために、マクロ DECLARE_DYNAMIC_CLASS、及び IMPLEMENT_DYNAMIC_CLASS を使用する。

アプリケーションの初期化で wxDocTemplate オブジェクトを作成するときに、wxDocTemplate クラスが wxDocument インスタンスの作成方法を知り得るよう、wxDocTemplate コンストラクタの引数に CLASSINFO(YourDocumentClass) を渡さなければならない。

ドキュメントオブジェクトを動的に作成する方法で wxWidgets を使用したくない場合には、wxDocTemplate::CreateDocument を適切なクラスのインスタンスを返すようにオーバーライドしなければならない。

wxView 概要

クラス : wxView

wxView クラスは、アプリケーションのファイルベースデータの要素の表示や編集をモデル化するために使用される。wxDocument, wxDocTemplate, wxDocManager と共に、wxWidgets でサポートされているドキュメント／ビューフレームワークの一部である。

See the example application in samples/docview.

wxView 抽象クラスを使用するために、新しい派生クラスを作成し、少なくともメンバ関数 OnCleate, OnDraw, OnUpdate, 及び OnClose をオーバーライドしなければならない。ビューに含まれるフレームからにメニューコマンドに対応する OnMenuCommand をオーバーライドする場合があるかもしれない。

フレームワークに対して、要求に応じてビューオブジェクトを作成することを許可するために、マクロ DECLARE_DYNAMIC_CLASS、及び IMPLEMENT_DYNAMIC_CLASS を使用する。アプリケーションの初期化で wxDocTemplate オブジェクトを作成する際、wxView インスタンスを作成する方法を wxDocTemplate が知り得るよう、wxDocTemplate コンストラクタの引数に CLASSINFO(YourViewClass) を渡す必要がある。

ビューオブジェクトを動的に作成する方法で wxWidgets を使用したくない場合には、適切なクラスのインスタンスを返すよう wxDocTemplate::CreateView をオーバーライドしなければならない。

wxDocTemplate 概要

クラス : wxDocTemplate

wxDocTemplate クラスは、ドキュメントクラスとビュークラスの間の関係をモデル化するために使用される。アプリケーションは、ドキュメント／ビューの対ごとにドキュメントテンプレートオブジェクトを作成する。ドキュメントテンプレートのリストは、ドキュメントとビューを作成するときに使用される wxDocManager インスタンスにより管理される。各ドキュメントテンプレートは、ドキュメント／ビューの組み合わせに対して、ファイルフィルターとデフォルト拡張子に何が適切か、また、どのようにドキュメントやビューを作成するのかを把握している。

例えば、行単位のリストを読み込み・保存するようなアプリケーションを作成しようとしたとす

る。データに対して２つのビュー（グラフィックと行のリスト）がある場合、１つのドキュメントクラス DoodleDocument と、２つのビュークラス（DoodleGraphicView と DoodleListView）を作成する。さらに、２つのドキュメントテンプレートを作成する。１つはグラフィック、もう１つはリストビューのためのものである。そして、両方のドキュメントテンプレートに対し、１つの同じドキュメントクラスとデフォルト拡張子を渡すが、異なるビュークラスを渡す。ユーザがオープンメニューをクリックすると、ファイル選択ダイアログが可能なファイルフィルタ（各 wxDocTemplate に対して１つ）のリスト共に表示される。wxDocTemplate を選択するようなフィルタを選択し、あるファイルが選択されると、そのテンプレートはドキュメントとビューを作成する。

アプリケーションが１つのドキュメントと１つのビューを持つ場合、１つのドキュメントテンプレートが作成され、ダイアログは適当に簡略化される。

wxDocTemplate は、wxView, wxDocument, 及び wxDocManager クラスと共に、wxWidgets でサポートされているドキュメント／ビューフレームワークの一部である。

See the example application in samples/docview.

wxDocTemplate クラスを使用するために、新しい派生クラスは必要ない。動的なインスタンス生成を許可するために、CLASSINFO(YourDocumentClass) と CLASSINFO(YourViewClass) を含めて、関連する情報をコンストラクタに渡すだけである。動的にドキュメントオブジェクトを作成する方法で wxWidgets を使用したくない場合には、適切なクラスのインスタンスを返すように wxDocTemplate::CreateDocument と wxDocTemplate::CreateView をオーバーライドしなければならない。

注意：ドキュメントテンプレートは、C++ テンプレートとは無関係である。

wxDocManager **概要**

クラス : wxDocManager

wxDocManager クラスは、wxView, wxDocument, 及び wxDocTemplate クラスと共に、wxWidgets でサポートされているドキュメント／ビューフレームワークの一部である。

wxDocManager インスタンスはドキュメント、ビュー、及びドキュメントテンプレートを統合する。ドキュメントとテンプレートのインスタンスを保持し、ファイルダイアログなどの多くの機能はこの変数を通じて行われる。アプリケーションは、このクラスを ' そのまま ' 使用するか、クラスを継承して機能の拡張や変更のために幾つかのメンバ関数をオーバーライドすることができす。ドキュメント、ビュー、及びテンプレートを使用する前に、アプリケーション初期化の初期の部分でこのクラスのインスタンスを作成する必要がある。

１つのアプリケーションで、複数の wxDocManager インスタンスを持つことも可能である。

See the example application in samples/docview.

wxCommand 概要

クラス : wxCommand, wxCommandProcessor

wxCommand は、メニューの選択、ツールバーボタンのクリック、データやビューを変更するためにアプリケーションによって用意されたような、アプリケーションコマンドを形成するための基本クラスである。

アプリケーションの機能を、読みにくい、あるいはメンテナンスしにくい方法で関数化したり switch 文で分散させるよりも、コマンド機能をフレームワークやアプリケーションにより扱えるオブジェクトとして明示的に表現したほうが良い。ユーザインタフェースイベントが発生すると、アプリケーションは wxCommandProcessor に対して実行やストアするようなコマンドを発行する。

wxWidgets のドキュメント／ビューフレームワークは、wxCommand オブジェクトと wxCommandProcessor オブジェクトを使用することにより、Undo と Redo を操作する。コマンドをストア、ロード、リプレイするようなマクロ機能を実装するなど、より進んだ wxCommand の使用方法を見つけるかもしれない。

アプリケーションは、コマンドごとに新しいクラスを継承したり、あるいは、整数型や文字列型のコマンド識別子を引数として１つのクラスだけ使用することもできる。

wxCommandProcessor 概要

クラス : wxCommandProcessor, wxCommand

wxCommandProcessor は、wxCommand インスタンスの履歴を管理するクラスであり、undo/redo を機能的に内蔵している。異なる動作をさせたい場合には、このクラスを継承させること。

wxFileHistory 概要

クラス : wxFileHistory, wxDocManager

wxFileHistory は、最後に使用した幾つかのファイルを記録する機能をカプセル化し、ユーザに対して、ファイルメニューに追加された使用したことのあるファイルを素早くロードできるようにする。

wxFileHistory は wxDocManager により使用されるが、独立に使用される事もできる。ファイルのスクロールリストをポップアップ表示したり、異なる動作をさせたい場合には、このクラスを継承させても良い。

wxFileHistory::FileHistoryUseMenu を呼ぶことにより、ファイル履歴とファイルメニューを関連付けることができる。wxID_FILE1 から wxID_FILE9 の範囲で、メニュー識別子を使用することができる。

これら識別子のうち１つのファイルロードコマンドに応答するために、イベントハンドらを使用してそれら識別子を操作する必要がある。例えば：

```
BEGIN_EVENT_TABLE(wxDocParentFrame, wxFrame)
    EVT_MENU(wxID_EXIT, wxDocParentFrame::OnExit)
    EVT_MENU_RANGE(wxID_FILE1, wxID_FILE9, wxDocParentFrame::OnMRUFile)
END_EVENT_TABLE()

void wxDocParentFrame::OnExit(wxCommandEvent& WXUNUSED(event))
{
    Close();
}

void wxDocParentFrame::OnMRUFile(wxCommandEvent& event)
{
      wxString f(m_docManager->GetHistoryFile(event.GetId() - wxID_FILE1));
      if (f != "")
        (void)m_docManager->CreateDocument(f, wxDOC_SILENT);
}
```

wxWidgets **定義済みコマンド識別子**

アプリケーションのメニューとドキュメント／ビューフレームワークとの間のコミュニケーションのために、幾つかのコマンド識別子がメニューの中で定義されている。フレームワークは、その識別子を認識し、wxFrame::OnMenuCommand から（または、ツールバーや、その他のユーザインタフェースから）コマンドが転送された時にそのコマンドを実行する。

- wxID_OPEN (5000)
- wxID_CLOSE (5001)
- wxID_NEW (5002)
- wxID_SAVE (5003)
- wxID_SAVEAS (5004)
- wxID_REVERT (5005)
- wxID_EXIT (5006)
- wxID_UNDO (5007)
- wxID_REDO (5008)
- wxID_HELP (5009)
- wxID_PRINT (5010)
- wxID_PRINT_SETUP (5011)
- wxID_PREVIEW (5012)