クラス : wxCloseEvent, wxWindow

ウィンドウの削除は分かりにくい項目である。そのためこの概要では、ウィンドウの破棄とクローズの方法について説明する。

ウィンドウ削除のときに発生するイベントの順番は？

ユーザが、フレームやダイアログに組み込まれているクローズボタンやクローズコマンドをクリックすると、wxWidgets は wxWindow::Close を呼ぶ。このとき、EVT_CLOSE イベントを順番に生成する。：wxCloseEvent を参照のこと。

適切なイベントハンドラを定義し、ウィンドウを破棄するかどうか決定することは、アプリケーションの義務である。もし何らかの理由によりアプリケーションがクローズさせられる（wxCloseEvent::CanVeto が false を返す）場合には、別の方法でその要求を無視しウィンドウを常に破棄しなければならないか、もしくは、安全にクローズしてよいかどうかを決定する前の問い合わせに対してユーザが解凍するのを持つことも考えられる。EVT_CLOSE に対するハンドラは、ウィンドウを破棄しない場合には、wxCloseEvent::Veto を呼ぶことにより呼び出し元に通知しなければならない。これを呼ぶことは、呼び出し元に有意な情報を提供する。

wxCloseEvent ハンドラは、ウィンドウ削除のために wxWindow::Destroy を呼ぶだけであり、delete 演算子を使用すべきではない。それは、幾つかのウィンドウクラスに関して、存在しないウィンドウに対してイベントが送信されるのは危険であるため、wxWidgets が全てのイベントが処理されるまでウィンドウを実際に削除するのを遅らせるためである。

次のセクションでの補足されている通り、Close を呼ぶことはウィンドウが破棄されることを保障するものではない。ウィンドウを確実に削除する場合には wxWindow::Destroy を呼ばなければならない。

アプリケーションはどのようにウィンドウ自身をクローズしたらよいか？

アプリケーションは、フレームワークが行うのと同じように wxWindow::Close イベントを使用するか、wxWindow::Destroy を直接呼ぶことができる。Close() を使用すると、フレームを削除し、それが拒否されないようにイベントハンドラに対して通知するために、この関数に引数 true を渡すことができる。

Destroy の代わりに Close を使用する利点は、EVT_CLOSE ハンドラによって定義されたクリーンアップのためのコードを呼ぶことにある。例えば、ユーザに対してドキュメントを保存するかどうかをまず問い合わせてから、ウィンドウに含まれるドキュメントをクローズさせることができる。Destroy が確実にウィンドウを破棄するのに対して、Close はこの関数（false を返す）によって拒否することができる。

デフォルトの動作は何か？

wxDialog のデフォルトクローズイベントハンドラは、キャンセルコマンドと同様で、wxID_CANCEL を発行する。キャンセルイベントのハンドラは、それ自身で Close を呼ぶため、無限ループチェックをしている。wxID_CANCEL のデフォルトハンドラは、(モードレスでは) ダイアログを非表示にし、(モーダルでは) EndModal (wxID_CANCEL) を呼ぶ。言い換えれば、デフォルトでは、ダイアログは破棄されない (スタック上には作成されるかもしれないが、動的生成という仮定では作成されない)。

wxFrame のデフォルトクローズイベントハンドラは、Destroy() を使用してフレームを破棄する。

ユーザがメニューから終了を呼んだ場合には何をすべきか？

フレームで wxWindow::Close を呼ぶだけでよい。これにより、フレームを破棄するような自身のクローズイベントが発生される。

アプリケーションが現時点で安全に終了できるか、クローズイベントハンドラで安全に終了できるか、または、終了メニューコマンドで安全に終了できるかどうかを確認することができる。例えば、全てのファイルが保存されたかどうかを確認したい場合がある。ユーザに対して、保存して終了、保存しないで終了、または、終了コマンドをキャンセルするかどうかの機会を与えなければならない。

1.xx の OnClose を 2.0 にアップグレードするには、何をすべきか？

wxWidgets 1.xx では、OnClose 関数は実際には 'this' を削除しないが、その代わり、呼び出し元(Close または、wxWidgets フレームワーク) に対してウィンドウを削除するか、削除しないかを通知していた。

コードをアップグレードするために、フレームやダイアログに EVT_CLOSE マクロを使用してイベントテーブルエントリを準備しなければならない。そのイベントハンドラ関数は、以下のようになる。：

```
void MyFrame::OnCloseWindow(wxCloseEvent& event)
{
  if (MyDataHasBeenModified())
  {
    wxMessageDialog* dialog = new wxMessageDialog(this,
      "Save changed data?", "My app", wxYES_NO|wxCANCEL);

    int ans = dialog->ShowModal();
    dialog->Destroy();

    switch (ans)
    {
      case wxID_YES:          // 保存したあと、破棄とアプリ終了
        SaveMyData();
        this->Destroy();
        break;
      case wxID_NO:           // 保存せず、破棄とアプリ終了のみ
        this->Destroy();
        break;
      case wxID_CANCEL:       // 何もせず、アプリを終了しない
      default:
        if (!event.CanVeto()) // 削除を棄却できるか確認
          this->Destroy();    // できなければ、ウィンドウを破棄
        else
          event.Veto();       // 呼び出し元にフレームを破棄しないことを通知
```

```
            break;
        }
    }
}
```

どのようにアプリケーションを終了させたらよいか？

指定されたトップウィンドウや最後のフレーム、またはダイアログが破棄されるとき、wxWidgets アプリケーションは自動的に終了する。wxApp::OnExit（これは仮想関数であり、イベントハンドラではない）にアプリケーションレベルのクリーンアップコードを記述すればよい。

子ウィンドウは自動的に削除されるのか？

はい。子ウィンドウは、親のデストラクタで削除される。フレームやダイアログの子フレームや子ダイアログウィンドウは親のクローズハンドラで確実にクローズするために、これにはフレームやダイアログであるような子も含まれる。

他の種類のウィンドウではどうか？

上記では、' 管理された ' ウィンドウ、即ち、フレームとダイアログについて言及した。親を持つウィンドウ、例えばコントロールは、遅れて破棄されることはなく、また通常、クローズイベントハンドラを持たない。しかし、望む場合には、クローズイベントハンドラを実装することもできる。一貫性のために、これらのウィンドウを確実に削除するときには、delete 演算子よりも wxWindow::Destroy 関数を使うほうが良い。