TN030: Customizing Printing and Print Preview

Article • 08/03/2021 • 6 minutes to read

> **Note**
>
> The following technical note has not been updated since it was first included in the online documentation. As a result, some procedures and topics might be out of date or incorrect. For the latest information, it is recommended that you search for the topic of interest in the online documentation index.

This note describes the process of customizing printing and print preview and describes the purposes of the callback routines used in `CView` and the callback routines and member functions of `CPreviewView`.

The Problem

MFC provides a complete solution for most printing and print preview needs. In most cases, little additional code is required to have a view able to print and preview. However, there are ways to optimize printing that require significant effort on the part of the developer, and some applications need to add specific user interface elements to the print preview mode.

Efficient Printing

When an MFC application prints using the standard methods, Windows directs all Graphical Device Interface (GDI) output calls to an in-memory metafile. When `EndPage` is called, Windows plays the metafile once for each physical band that the printer requires to print one page. During this rendering, GDI frequently queries the Abort Procedure to determine if it should continue. Typically the abort procedure allows messages to be processed so that the user may abort the print job using a printing dialog.

Unfortunately, this can slow the printing process. If the printing in your application must be faster than can be achieved using the standard technique, you must implement

manual banding.

Print Banding

In order to manually band, you must re implement the print loop such that `OnPrint` is called multiple times per page (once per band). The print loop is implemented in the `OnFilePrint` function in viewprnt.cpp. In your `CView`-derived class, you overload this function so that the message map entry for handling the print command calls your print function. Copy the `OnFilePrint` routine and change the print loop to implement banding. You will probably also want to pass the banding rectangle to your printing functions so that you can optimize drawing based on the section of the page being printed.

Second, you must frequently call `QueryAbort` while drawing the band. Otherwise, the Abort Procedure will not get called and the user will be unable to cancel the print job.

Print Preview: Electronic Paper with User Interface

Print Preview, in essence, tries to turn the display into an emulation of a printer. By default, the client area of the main window is used to display one or two pages fully within the window. The user is able to zoom in on an area of the page to see it in more detail. With additional support, the user may even be allowed to edit the document in preview mode.

Customizing Print Preview

This note only deals with one aspect of modifying print preview: Adding UI to preview mode. Other modifications are possible, but such changes are out of the scope of this discussion.

To add UI to the preview mode

1. Derive a view class from `CPreviewView`.

2. Add command handlers for the UI aspects you desire.

3. If you are adding visual aspects to the display, override `OnDraw` and perform your drawing after calling `CPreviewView::OnDraw`.

OnFilePrintPreview

This is the command handler for print preview. Its default implementation is:

```
C++

void CView::OnFilePrintPreview()
{
    // In derived classes, implement special window handling here
    // Be sure to Unhook Frame Window close if hooked.

    // must not create this on the frame. Must outlive this function
    CPrintPreviewState* pState = new CPrintPreviewState;

    if (!DoPrintPreview(AFX_IDD_PREVIEW_TOOLBAR, this,
        RUNTIME_CLASS(CPreviewView), pState))
    {
        // In derived classes, reverse special window handling
        // here for Preview failure case

        TRACE0("Error: DoPrintPreview failed");
        AfxMessageBox(AFX_IDP_COMMAND_FAILURE);
        delete pState;  // preview failed to initialize, delete State now
    }
}
```

`DoPrintPreview` will hide the main pane of the application. Control Bars, such as the status bar, can be retained by specifying them in the pState->*dwStates* member (This is a bit mask and the bits for individual control bars are defined by AFX_CONTROLBAR_MASK(AFX_IDW_MYBAR)). The window pState->*nIDMainPane* is the window that will be automatically hidden and reshown. `DoPrintPreview` will then create a button bar for the standard Preview UI. If special window handling is needed, such as to hide or show other windows, that should be done before `DoPrintPreview` is called.

By default, when print preview finishes, it returns the control bars to their original states and the main pane to visible. If special handling is needed, it should be done in an override of `EndPrintPreview`. If `DoPrintPreview` fails, also provide special handling.

DoPrintPreview is called with:

- The Resource ID of the dialog template for the preview toolbar.
- A pointer to the view to perform the printing for the print preview.
- The run-time class of the Preview View class. This will be dynamically created in DoPrintPreview.
- The CPrintPreviewState pointer. Note that the CPrintPreviewState structure (or the derived structure if the application needs more state preserved) must *not* be created on the frame. DoPrintPreview is modeless and this structure must survive until EndPrintPreview is called.

> **Note**
>
> If a separate view or view class is needed for printing support, a pointer to that object should be passed as the second parameter.

EndPrintPreview

This is called to terminate the print preview mode. It is often desirable to move to the page in the document that was last displayed in print preview. `EndPrintPreview` is the application's chance to do that. The pInfo->*m_nCurPage* member is the page that was last displayed (leftmost if two pages were displayed), and the pointer is a hint as to where on the page the user was interested. Since the structure of the application's view is unknown to the framework, then you must provide the code to move to the chosen point.

You should perform most actions before calling `CView::EndPrintPreview`. This call reverses the effects of `DoPrintPreview` and deletes pView, pDC, and pInfo.

C++

```
// Any further cleanup should be done here.
CView::EndPrintPreview(pDC, pInfo, point, pView);
```

CWinApp::OnFilePrintSetup

This must be mapped for the Print Setup menu item. In most cases, it is not necessary to override the implementation.

Page Nomenclature

Another issue is that of page numbering and order. For simple word processor type applications, this is a straightforward issue. Most print preview systems assume that each printed page corresponds to one page in the document.

In trying to provide a generalized solution, there are several things to consider. Imagine a CAD system. The user has a drawing that covers several E-size sheets. On an E-size (or a smaller, scaled) plotter, page numbering would be as in the simple case. But on a laser printer, printing 16 A-size pages per sheet, what does print preview consider a "page"

As the introductory paragraph states, Print Preview is acting like a printer. Therefore, the user will see what would come out of the particular printer that is selected. It is up to the view to determine what image is printed on each page.

The page description string in the `CPrintInfo` structure provides a means of displaying the page number to the user if it can be represented as one number per page (as in "Page 1" or "Pages 1-2"). This string is used by the default implementation of `CPreviewView::OnDisplayPageNumber`. If a different display is needed, one may override this virtual function to provide, for example, "Sheet1, Sections A, B".

See also

Technical Notes by Number
Technical Notes by Category