

A 5 step guide to build Image Apprentice Plugins



This documentation is copyrighted and rights on the content are reserved.
© 2005-2006. All Rights reserved by ADISL.

Image Apprentice (Version 0.5.0.1)

A 5 step guide to build Image Apprentice Plugins

In the Plugin Development Kit, the files of concern so as to write your own code are the pair of files 'Processor.cpp' and 'Processor.h'. The simplicity of using the Image Apprentice Plugin Development Kit lies in the fact that you have to concentrate on these two files only.

Processor.h contains the prototype of the C++ class CProcessor and Processor.cpp contains the implementation (actual source code) of the class. For the details of the member functions of class CProcessor, see Appendix-B.

For a short note on how to use Visual C++ 6.0 Development Environment and the PDK, see Appendix-A.

Let's begin with the steps to create a plugin for Image Apprentice Application from the ground up. We will create a plugin to invert the colors of an image so that the darker shades become lighter and vice versa. This is basically the Example-1 provided with the Plugin Development Kit. These are the steps involved:

STEP 1: In the file Processor.cpp, let's add the related menu names in the constructor CProcessor():

```
CProcessor::CProcessor()
{
    m_params.MenuTitle = "Invert Color";
    m_params.SubMenuTitle[0] = "Process";
    m_params.SubMenuTitle[1] = "Authors";
    m_numSubMenus = 2; // 0 - 1 as above.
}
```

STEP 2: In the file Processor.h, write the prototype of the function InvertColors() as follows:

```
class CProcessor
{
public:
    CProcessor();
    ~CProcessor();
    int m_numSubMenus;
    PARAMETERS m_params;
    int NumOfMenus();
    bool Process(int n, BYTE *buffer, int width,
                int height, int bytesPerPixel);

    // Add your Image Processing functions here:
    void InvertColors(BYTE *buffer, int width,
                    int height, int bytesPerPixel);
};
```

STEP 3: In the file Processor.cpp, write the implementation of the newly declared member function InvertColors() as follows:

```
void CProcessor::InvertColors(BYTE *buffer, int width,
                             int height, int bytesPerPixel)
{
    for(int i=0; i < width*bytesPerPixel*height; i++)
    {
        buffer[i] = 255 - buffer[i];
    }
}
```

STEP 4: In the file Processor.cpp, now that the `InvertColors()` function has been written, we must call it so that it gets executed at the press of a menu item in the Image Apprentice Application. We do this inside the body of the member function `Process()`. This goes inside the switch-case statements. Keep in mind that the integer `n` being passed as argument is the index of the sub-menu that is pressed by the user.

```
bool CProcessor::Process(int n, BYTE *m_buf, int width,
                        int height, int bytesPerPixel)
{
    switch(n)
    {
        case 0:
            InvertColors(m_buf, width, height, bytesPerPixel);
            break;
        case 1:
            ::MessageBox(NULL, "Invert Colors Sample Plugin",
                        "Jan 06, 2006", MB_ICONINFORMATION | MB_OK);
            break;
        default:
            break;
    }
    return true;
}
```

We note that `Process` is the function that gets the submenu index, the image data pointer, the width, the height and the bytes-per-pixel information from the Image Apprentice Application. We pass on these same values to `InvertColors()` as shown above in case 0. We also add a small message displaying window (called `MessageBox`) indicating some small text about the Plugin as case 1.

STEP 5: That's it! Go to Build menu of VC++ 6.0 and click on 'Rebuild All' menu item. If everything was written correctly, the compiler should give you a 0 error message. The plugin file (`IAPPlugin.dll`) is generated in the Release folder in the directory where the Plugin source code is placed. Copy `IAPPlugin.dll` (you might like to rename this file as per the functionality) and paste it in Plugins folder near the Image Apprentice Application (`ia.exe`). Your plugin is ready to use! Execute `ia.exe` and test your plugin.

Appendix A

A short note on how to use Visual C++ 6.0 Development Environment and the PDK:

Q: How do I open the Plugin Development Toolkit in VC++?

A: Within the folder where you unzipped the downloaded file (<http://home.iitk.ac.in/~rksr/adisl/ia/ImageApprentice.zip>), go to the path "IA v0.5.0.1 Plugin Development Kit\PDK". This is the VC++ project that contains the Plugin Development Kit. You can either double-click the "IAPlugin.dsw" file or you can open VC++ from the Start Menu of Windows and Go to File >> Open Workspace... and browse for the file "IAPlugin.dsw" in the above folder.

Q: How do I add source code to the CProcessor class?

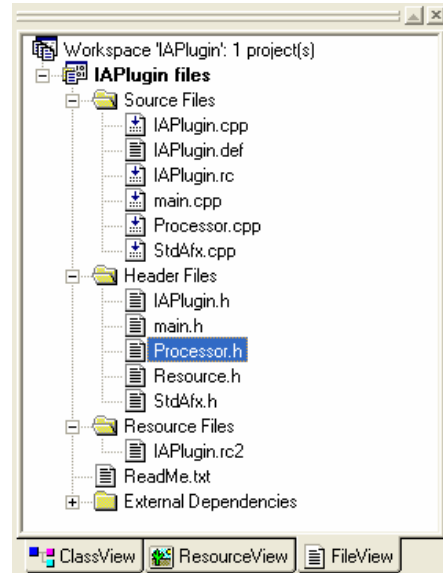
A: CProcessor class is contained in two files, Processor.h and Processor.cpp. If you open the PDK in Visual C++ Integrated Development Environment and look at the 'Workspace' window, you will see that it has three tabs named as 'Class View', 'Resource View', and 'File View'. Click on 'File View' Tab (see adjacent figure) and expand the 'Header Files' item. There you will find the file Processor.h. Just double click over it and it opens for you in the adjacent window. Edit your source file there. Similarly edit the Processor.cpp file which is listed under the 'Source Files' item in 'File View' Tab.

Q: How do I compile/build the PDK?

A: In the VC++ Development Environment, click on 'Build >> 'Rebuild All'. This does the compilation as well as building of the plugin.

Q: Where is this newly build plugin created?

A: Go to Build >> Set Active Configuration and check the type of Project Configuration. If it is 'IAPlugin – Win32 Release', then your plugin is created in the Release folder inside the PDK files. If the configuration type is 'IAPlugin – Win32 Debug', then it is inside the Debug folder of your PDK files. In both of these cases, the default filename of the plugin is 'IAPlugin.dll'. Copy and paste this .dll file in the 'Plugins' folder near your Image Apprentice Application executable (IA.exe).



Appendix B

Description of class CProcessor:

1. Processor.h

- a. A datatype PARAMS (a C++ structure) is defined as:

```
typedef struct PARAMS
{
    CString MenuTitle;
    CString SubMenuTitle[10]; // Update this as per no. of
                               // sub-menus in CProcessor().
                               // Default is 10.
} PARAMETERS;
```

- The member variable MenuTitle is the name of the Menu Option that shows up when you click the 'Plugins' Menu in the Image Apprentice Application.
- The member variable SubMenuTitle is an array of the names of submenus that show up on expanding the MenuTitle of concern in the Image Apprentice Application.

- b. The class CProcessor is defined as:

```
class CProcessor
{
public:
    CProcessor();
    ~CProcessor();
    int m_numSubMenus;
    PARAMETERS m_params;
    int NumOfMenus();
    bool Process(int n, BYTE *buffer, int width,
                 int height, int bytesPerPixel);

    // Add your Image Processing functions here:

};
```

- CProcessor() and ~CProcessor() are respectively the constructors and destructors of the class CProcessor.
- m_params is a member variable of type PARAMETERS. It is used to name the menu (and its submenus) that show up under the the 'Plugins' menu when the plugin is detected by Image Apprentice Application. You have to fill this up in the constructor CProcessor() appropriately.
- NumOfMenus() is a member function that returns the no. of menus to the Image Apprentice Application. You needn't touch this function as the architecture of Image Apprentice Application takes care of calling it at appropriate place.
- Process(int n, BYTE *buffer, int width, int height, int bytesPerPixel) is the function that is doing the real work for you. It accepts, from the Image Apprentice Application, the following as arguments:
 - int n - index of the submenu that was clicked by the user in the Image Apprentice Application.
 - BYTE *buffer – pointer to the first BYTE pointing to the image data. This is the image that was opened using the Image Apprentice Application.
 - int width – width of the image opened in pixels.

- o `int` `height` – height of the image opened in pixels.
- o `int` `bytesPerPixel` – Number of bytes used per pixel. It is 3 for RGB images and 1 for grayscale or 256-colormapped images.