Taking Control of the File Based Write Filter with the FBWF API Set

By Sean D. Liming and John R. Malin
SJJ Embedded Micro Solutions

First Printing: October 2006

Published in the United States by

SJJ Embedded Micro Solutions, LLC.
11921 Tivoli Park Row #5
San Diego, CA 92128 USA

www.sjjmicro.com

Table of Contents

1 Introducing FBWF APIs

The Enhanced Write Filter (EWF) offers the developer two ways to control the state of EWF: EWFMGR.EXE utility and the EWF API set. The EWF API set was introduced after the release of XPe SP1. In that time, there have been multiple third party control solution developed and a .NET implementation that provide developers with different choices to control EWF.

The new File Based Write Filter (FBWF) offers the same solution set: a command line utility and a FBWF API set designed for native C++ applications. The FBWFMGR.EXE command line utility is discussed in the online help and in our article introducing FBWF. In this article, we will focus our attention on using the FBWF API set. We will provide a sample application that demonstrates the use of the different FBWF API functions.

Certainly, it is possible to shell out to the FBWFMGR. EXE command line utility to perform the basic actions, but the FBWF APIs can be integrated into your custom application when you what to provide more interactive control and data display for the user or administrator. Most important you want to monitor

2 A High Level Review of FBWF

FBWF provides protection on a file level instead of the whole volume or partition like EWF. The developer can set specific files or directories to be unprotected. Any writes made to protected files are sent to RAM overlay (cache). Any writes to unprotected (exclusion) files are passed through to the disk.

FBWF working at a file level provides some features that EWF doesn't provide, like the ability to add or remove protected volumes, to allow file writes to pass through to the protected volume, and to control the amount of RAM that FBWF will use. Here are the main features:

- File and Directory Management Transparency
- Selective Write-Through
- Selective Commits and Restores
- Dynamic Protection
- Improved Overlay Memory Use

3 The FBWF API Set

The FBWF API set consists of a separate installable FBWFAPI.H and FBWFLIB.LIB. The FBWFLIB.DLL is required for the custom FBWF API application to run, and the DLL is part of the File Based Write Filter component.

The FBWF API function set is divided into 4 groups:

- Management of the System Wide Cache
- Manage a Specific Volume
- Manage Files and Directories in a Specific protect volume
- Control individual file commit or restore

Many of these functions when called will require a reboot to be performed. Some of the status function provides information on the current state and what the state will be upon reboot. You can cancel the next state during the current session. For example, if a call is made to FbwfDisableFilter, but later in the session the desire is to keep FBWF enabled upon re-boot, the FbwfEnableFilter can be called to keep the FBWF enabled, thus cancelling the FbwfDisableFilter call.

Function	Description
FbwfEnableFilter	Enables write filtering in the next session.
FbwfDisableFilter	Disables write filtering for the next session.
FbwfIsFilterEnabled	Queries the filter state for the current and next session.
FbwfSetCacheThreshold	Sets the maximum amount of RAM the write cache may use. The values must be an integer value between 16 (16MB) and 1024 (1024MB).
FbwfCacheThresholdNotification	Allows applications to be signaled when the remaining cache memory falls below a specified size.
FbwfEnableCachePreAllocation	Causes the write filter to allocate the entire cache size at start up instead of allocating memory as needed.
FbwfDisableCachePreAllocation	Turns off cache pre-allocation so that cache memory is allocated only as needed.
FbwfIsCachePreAllocationEnabled	Retrieves the state of the pre-allocation flag for the current and next sessions.
FbwfGetMemoryUsage	Retrieves information about memory currently used by the cache.
FbwfEnableCompression	Allows the write filter to compress the memory cache.
FbwfDisableCompression	Disables cache compression.
FbwfIsCompressionEnabled	Retrieves the status of the compression flag.

Table 1 - FBWF functions used to manage the system-wide write cache.

Function	Description
FbwfProtectVolume	Enables write protection for a specified volume. This can be used to add new volumes.
FbwfUnprotectVolume	Removes write protection for the specified volume.
FbwfGetVolumeList	Retrieves the list of protected volumes.
FbwfIsVolumeProtected	Retrieves the protection status for a specified volume in both the current and next sessions.
FbwfFindFirst	Retrieves information about the first file in the FBWF memory cache.
FbwfFindNext	Retrieves information about the next file in the FBWF memory cache.
FbwfFindClose	Closes the FbwfFindFirst/FbwfFindNext search.

Table 2 - Manage the protection of a specific volume.

Function	Description
FbwfAddExclusion	Adds a file or directory to the exclusion list.
FbwfRemoveExclusion	Removes a file or directory from the exclusion list.
FbwfGetExclusionList	Retrieves the list of files and directories in the exclusion list.

Table 3 - Manage files and Directories excluded within a protected volume.

Function	Description
FbwfCommitFile	Writes the cached file overlay to the physical disk file.
FbwfRestoreFile	Clears the cached view of the specified file, thus protect file on disk is viewed

Table 4 - Functions used for file commit and restore.

4 Creating a Custom Command Line Application

The help files provide details of the detailed information about each function, the syntax, and any results. Now lets see how we can put these functions into action. Using visual Studio 2005, we will create a custom command line application that demonstrates who many of these functions operate.

4.1 Part 1 Project Setup

1. **Open** Visual Studio .2005
2. From the Menu select **File->New Project**
3. The New Project dialog box appears, under Project Types select **Visual C++** .
4. On the right under templates select **Win32 Console Application**.
5. Enter name **FBWFCMD** as show in Fig 1.

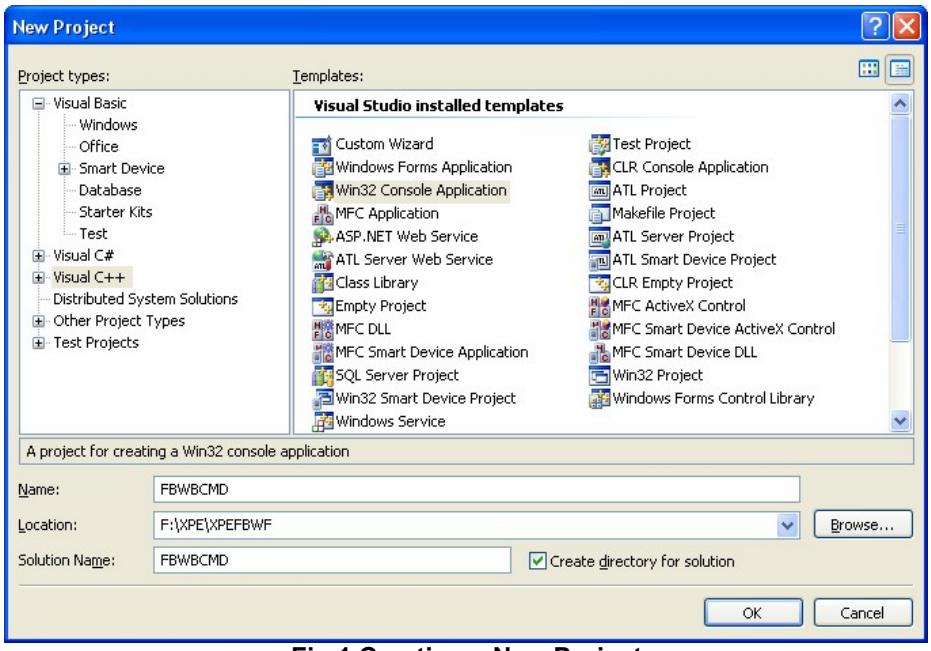

Fig 1 Creating a New Project

6. Click **OK**.
7. Click on **Finish** in the Win32 Application Wizard.
8. **Copy fbwfapi.h** to the directory of the project so that it exists within the same directory as the other project files.
9. In the Solutions Explorer, right click on **Header Files**, and select **Add Existing Item** from the pop-up context menu.
10. The Open Add Existing Item dialog box should open to the project directory. Highlight the **ewfapi.h** file and click **OPEN**. Fig 2 shows the result.

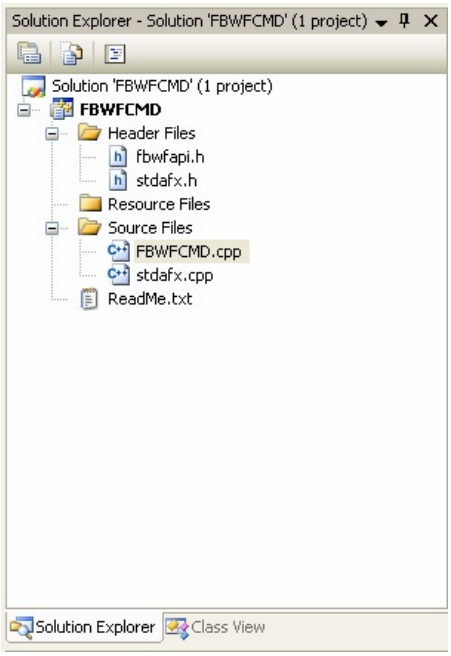

Fig 2 Adding the FBWFAPI.H Resource

11. Now, make sure that **FBWFCMD.cpp** has focus. We need to add the FBWFLIB.LIB to the project. From the menu, select **Project**.
12. Select **FBWFCMD Properties** from the submenu.
13. The properties dialog provides the ability to custom control the building of the application. From the Configuration drop down, select **All Configurations**.
14. In the Configuration Properties tree on the left, expand **Linker** and select **Input**.
15. Select **Additional Dependencies**, and click on the box with three dots.
16. Type in the path and name to the **fbwflib.lib**, as shown in Fig 3, and click **OK**.

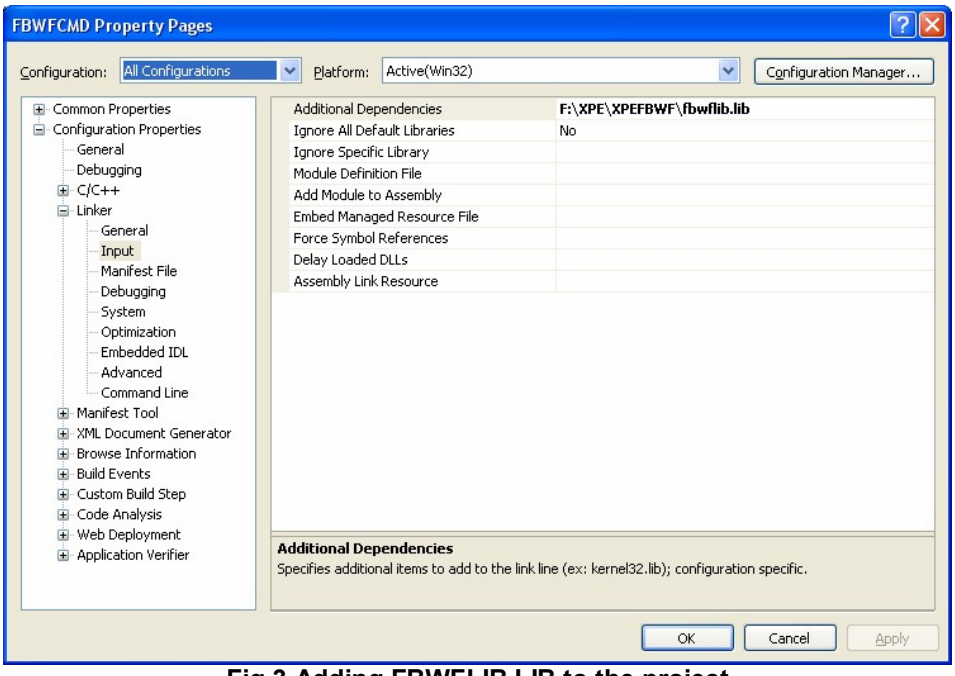

Fig 3 Adding FBWFLIB.LIB to the project

17. Click **OK** again to close the Properties dialog.

4.2 Part 2 Writing the Code

1. Enter the code below for the FBWFCMD.CPP file

```
// FBWFCMD.cpp : Defines the entry point for the console application.
//
// Copyright (c) 2006 SJJ Embedded Micro Solutions, LLC. All Rights Reserved
//
// Code is provided AS IS without any warranty
//
// Description: Command line application demonstrates how to implement
//                            the FBWF APIs for several of the core functions.
//
//


#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include "fbwfapi.h"
void DisplayHelp();

int main(int argc, char *argv[])
{
        WCHAR szDrive[20];
        WCHAR szPath[128];
        DWORD dwStatus;
        ULONG IscsEnabled = 0;
        ULONG IsnsEnabled = 0;
        CHAR *csResult;
        CHAR *nsResult;
        ULONG cacheThreshold = 64;


        printf("\nFBWFCMD - FBWF API Example\n");
        printf("Copyright (c) 2006 SJJ Embedded Micro Solutions LLC., All Rights
Reserved\n\n");


        if      ((argc >= 2))
        {
                //-----------------------------------------------
                // FbwfEnableFilter
                //Enable FBWF
                //-----------------------------------------------
                if (strcmp(argv[1], "-enable")==0) {
                        if(!FbwfEnableFilter()){
                                printf("FBWF Enabled.  ");
                                printf("Reboot system for change to take effect\n");
                        }
                        else{
                                printf("Enable Error");
                        }

                }

                //-----------------------------------------------
                //FbwfDisableFilter
                //Disable FBWF
                //-----------------------------------------------
                if (strcmp(argv[1], "-disable")==0) {
                        if(!FbwfDisableFilter()){
                                        printf("FBWF Disabled. ");
                                        printf("Reboot system for change to take effect\n");
                        }
                        else{
                                printf("Disable Error");
                        }
```

```
}

//-----------------------------------------------
//FbwfIsFilterEnabled
//Check to see if FBWF is enabled
//-----------------------------------------------
if (strcmp(argv[1], "-fbwfstate")==0) {
        if(!FbwfIsFilterEnabled(&IscsEnabled, &IsnsEnabled )){
                if(IscsEnabled){

                        csResult = "ENABLED";
                }
                else{
                        csResult = "DISABLED";
                }
                if(IsnsEnabled){

                        nsResult = "ENABLED";
                }
                else{
                        nsResult = "DISABLED";
                }
                printf("FBWF is %s\n", csResult);
                printf("FBWF for the next session is %s\n", nsResult);
        }
        else{
                printf("State Error");
        }
}

//-----------------------------------------------
//FbwfEnableCompression
//Enable FBWF Compression
//-----------------------------------------------
if (strcmp(argv[1], "-enablecompression")==0) {
        if(!FbwfEnableCompression( )){
                printf("FBWF compression enabled. ");
                        printf("Reboot system for change to take effect\n");
        }
        else{
                printf("Enable Compression Error");
        }
}

//-----------------------------------------------
//FbwfDisableCompression
//Disable FBWF Compression
//-----------------------------------------------
if (strcmp(argv[1], "-disablecompression")==0) {
        if(!FbwfDisableCompression( )){
                printf("FBWF compression disabled. ");
                        printf("Reboot system for change to take effect\n");
        }
        else{
                printf("Disable Compression Error");
        }
}

//-----------------------------------------------
//FbwfEnableCachePreAllocation
//Enable FBWF Cache PreAllocation
//-----------------------------------------------
if (strcmp(argv[1], "-enablecache")==0) {
        if(!FbwfEnableCachePreAllocation( )){
                printf("FBWF Cache PreAllocation enabled. ");
                        printf("Reboot system for change to take effect\n");
        }
        else{
                printf("EnableCachePreAllocation Error");
        }
}

//-----------------------------------------------
//FbwfDisableCachePreAllocation
```

```
                //Disable FBWF Cache PreAllocation
                //-------------------------------------------
                if (strcmp(argv[1], "-disablecache")==0) {
                        if(!FbwfDisableCachePreAllocation( )){
                                printf("FBWF Cache PreAllocation disabled. ");
                                        printf("Reboot system for change to take effect\n");
                        }
                        else{
                                printf("DisableCachePreAllocation Error");
                        }
                }

                //-------------------------------------------
                //FbwfProtectVolume
                //Add a volume to be protected by FBWF Cache
                //-------------------------------------------
                if (strcmp(argv[1], "-protectvolume")==0) {

                        mbstowcs(szDrive, argv[2], 10);

                        dwStatus=FbwfProtectVolume(szDrive );
                        if(!dwStatus ){
                                printf("Volume %s will be protected in the next session.
",argv[2]);
                                printf("Reboot system for change to take effect\n");
                        }
                        else{
                                printf("Error %d \n", dwStatus);
                                printf("FBWF must be enabled for the next session");
                                DisplayHelp();
                        }
                }

                //-------------------------------------------
                //FbwfUnprotectVolume
                //Remove a volume to be protected by FBWF Cache
                //-------------------------------------------
                if (strcmp(argv[1], "-unprotectvolume")==0) {

                        mbstowcs(szDrive, argv[2], 10);

                        dwStatus=FbwfUnprotectVolume(szDrive,0 );
                        if(!dwStatus ){
                                printf("Volume %s  will be unprotected in the next session.
", argv[2]);
                                printf("Reboot system for change to take effect\n");
                        }
                        else{
                                printf("Error %d \n", dwStatus);

                                DisplayHelp();
                        }
                }

                //-------------------------------------------
                //FbwfIsCompressionEnabled
                //Check to see if FBWF Compression is enabled
                //-------------------------------------------
                if (strcmp(argv[1], "-fbwfcompression")==0) {
                        if(!FbwfIsCompressionEnabled(&IscsEnabled, &IsnsEnabled )){
                                if(IscsEnabled){

                                        csResult = "ENABLED";
                                }
                                else{
                                        csResult = "DISABLED";
                                }
                                if(IsnsEnabled){

                                        nsResult = "ENABLED";
                                }
                                else{
                                        nsResult = "DISABLED";
                                }
```

```
                                printf("FBWF compresspon is %s\n", csResult);
                                printf("FBWF compression for the next session is %s\n",
nsResult);
                        }
                        else{
                                printf("Compression Error");
                        }
                }
                //-----------------------------------------------
                //FbwfIsCachePreAllocationEnabled
                //Check to see if FBWF Cache PreAllocation is enabled
                //-----------------------------------------------
                if (strcmp(argv[1], "-fbwfprealloc")==0) {
                        if(!FbwfIsCachePreAllocationEnabled(&IscsEnabled, &IsnsEnabled )){
                                if(IscsEnabled){

                                        csResult = "ENABLED";
                                }
                                else{
                                        csResult = "DISABLED";
                                }
                                if(IsnsEnabled){

                                        nsResult = "ENABLED";
                                }
                                else{
                                        nsResult = "DISABLED";
                                }
                                printf("FBWF Cache PreAlloc is %s\n", csResult);
                                printf("FBWF Cache PreAlloc for the next session is %s\n",
nsResult);
                        }
                        else{
                                printf("Cache PreAlloc Error");
                        }
                }

                //-----------------------------------------------
                //FbwfIsVolumeProtected
                //Check if FBWF is portecting a specific volume
                //-----------------------------------------------
                if (strcmp(argv[1], "-volumestatus")==0) {

                        mbstowcs(szDrive, argv[2], 10);

                        dwStatus=FbwfIsVolumeProtected(szDrive,&IscsEnabled,&IsnsEnabled);

                        if(!dwStatus ){
                                if(IscsEnabled){

                                        csResult = "ENABLED";
                                }
                                else{
                                        csResult = "DISABLED";
                                }
                                if(IsnsEnabled){

                                        nsResult = "ENABLED";
                                }
                                else{
                                        nsResult = "DISABLED";
                                }
                                printf("Volume %s  protection is %s\n", argv[2],csResult);
                                printf("Volume %s  protection for next session is %s\n",
argv[2],nsResult);
                        }
                        else{
                                printf("Error %d \n", dwStatus);

                                DisplayHelp();
                        }
                }
```

```
		//-------------------------------------------
		//FbwfSetCacheThreshold
		//Set the Cache threshold for FBWF
		//-------------------------------------------
		if (strcmp(argv[1], "-setcache")==0) {

			cacheThreshold = atoi(argv[2]);

			dwStatus=FbwfSetCacheThreshold(cacheThreshold);

			if(!dwStatus ){
				printf("Cache Threshold set to %u\n ", cacheThreshold);
				printf("Reboot system for change to take effect\n");
			}
			else{
				printf("Error %d \n", dwStatus);
				if(dwStatus = ERROR_INVALID_PARAMETER){
					printf("Error: Threshold must be between 16 to
1024\n");
				}
				DisplayHelp();
			}

		}

		//-------------------------------------------
		// FbwfAddExclusion
		// Add a file or path to the exclusion list
		//-------------------------------------------
		if (strcmp(argv[1], "-addex")==0) {

			mbstowcs(szDrive, argv[2], 10);
			mbstowcs(szPath, argv[3], 128);

			dwStatus=FbwfAddExclusion(szDrive,szPath);
			if(!dwStatus ){
				printf("%s on drive %s is now excluded.\n
",argv[3],argv[2]);
				printf("Reboot system for change to take effect\n");
			}
			else{
				printf("Error %d \n", dwStatus);
				DisplayHelp();
			}
		}

		//-------------------------------------------
		//FbwfRemoveExclusion
		//Remove a file or path to the exclusion list
		//-------------------------------------------
		if (strcmp(argv[1], "-remex")==0) {

			mbstowcs(szDrive, argv[2], 10);
			mbstowcs(szPath, argv[3], 128);

			dwStatus=FbwfRemoveExclusion(szDrive,szPath);
			if(!dwStatus ){
				printf("%s on drive %s is now removed from exclusion.\n
",argv[3],argv[2]);
				printf("Reboot system for change to take effect\n");
			}
			else{
				printf("Error %d \n", dwStatus);
				DisplayHelp();
			}
		}

		//-------------------------------------------
		// FbwfCommitFile
		// Commit a file in cache to the disk
		//-------------------------------------------
		if (strcmp(argv[1], "-commit")==0) {

			mbstowcs(szDrive, argv[2], 10);
```

```
                        mbstowcs(szPath, argv[3], 128);

                        dwStatus=FbwfCommitFile(szDrive,szPath);
                        if(!dwStatus ){
                                printf("%s on drive %s has been committed to the disk.\n
",argv[3],argv[2]);
                        }
                        else{
                                printf("Error %d \n", dwStatus);
                                DisplayHelp();
                        }
                }

                //--------------------------------------------
                //FbwfRestoreFile
                //Restore a file in cache from the version on the disk
                //--------------------------------------------
                if (strcmp(argv[1], "-restore")==0) {

                        mbstowcs(szDrive, argv[2], 10);
                        mbstowcs(szPath, argv[3], 128);

                        dwStatus=FbwfRestoreFile(szDrive,szPath);
                        if(!dwStatus ){
                                printf("%s on drive %s has been restored from the disk.\n
",argv[3],argv[2]);
                        }
                        else{
                                printf("Error %d \n", dwStatus);
                                if(dwStatus = ERROR_ACCESS_DENIED){
                                        printf("Error: File not in cache\n");
                                }
                                if(dwStatus = ERROR_FILE_NOT_FOUND){
                                        printf("Error: File not found\n");
                                }

                        }
                }

                //--------------------------------------------
                // Display help
                //--------------------------------------------
                if ((strcmp(argv[1], "-?")==0) || (strcmp( argv[1],"-help")==0)) {

                                        DisplayHelp();
                }

        }
        else
        {
                // If the parameter conditions fails display help
                DisplayHelp();
                return 1;
        }

        return 0;
}

void DisplayHelp()
{
        printf("\nFBWFCMD <Option>\n");
        printf("-enable                         Enable FBWF\n");
        printf("-disable                Disable FBWF\n");
        printf("-enablecompression          Enable Compression\n");
        printf("-disablecompression         Disable Compression\n");
        printf("-enablecache            Enable Cache PreAllocate\n");
        printf("-disablecache           Disable Cache PreAllocate \n");
        printf("-protectvolume <volume>       Protect a new volume\n");
    printf("-unprotectvolume <volume> Unprotect a new volume\n");
        printf("-fbwfstate              Gets the filter status\n");
        printf("-fbwfcompression        Gets the compresstion status\n");
        printf("-fbwfprealloc           Gets the Cache PreAllocation status\n");
        printf("-volumestatus <volume>        Gets the volume status\n");
        printf("-setcache <cache size>        cache size must be between 16 and 1024\n");
```

```
        printf("-addex <drive> <file or path> add a file or path to the exclusion
list\n");
        printf("-remex <drive> <file or path> remove a file or path to the exclusion
list\n");
        printf("-commit <drive> <file or path>       commits a file in cache to the
disk\n");
        printf("-restore <drive> <file or path>      restores a file in cache from the
disk\n");
}
```

2. **Build** the application.
3. Using Component Designer, **create** a component for the FBWFCMD.EXE application. Place the new component in the Embedded Enabling Features category.
4. **Import** the new component into the database.
5. Finally, create an XPE configuration using Target Designer and be sure to include the following components:

    - File Based Write Filter
    - .NET 2.0 Framework
    - Task Manager
    - CMD – Windows Command Processor
    - FAT
    - NTFS

6. Include the component created for the FBWFCMD application.
7. Make sure that you have **imported** the PMQ information for the target platform.
8. Using the Extra Files in the configuration, add a folder called **FBWFTEST.**
9. **Modify** the File Bases Write Filer settings so that it will protect the C: partition and that FBWFTEST is in the exclusion list.
10. Check dependencies, build the configuration, download to the targets first partition, and boot to XP Embedded.
11. Once the Explorer Shell appears, **click** on new task and type CMD in the edit box.
12. Click **OK**.
13. In the Command Window, use **FBWFMGR** to make sure that EWF was setup properly during FBA.
14. Now, test FBWFCMD to test the different functions. You may have to reboot the system after some of the functions. Try disable and then check the state. Re-enable FBWF, thus cancelling the disable call. Add \Windows to the exclusion list, and the run FBWFMGR to see that \Windows will be part of the exclusion list in the next session. Edit the FBA log, and commit the changed file to the disk.

    C:\>**FBWFCMD -disable**

    FBWFCMD - FBWF API Example
    Copyright (c) 2006 SJJ Embedded Micro Solutions LLC., All Rights Reserved

    FBWF Disabled. Reboot system for change to take effect

    C:\>**FBWFCMD -fbwfstate**

    FBWFCMD - FBWF API Example
    Copyright (c) 2006 SJJ Embedded Micro Solutions LLC., All Rights Reserved

    FBWF is ENABLED
    FBWF for the next session is DISABLED

    C:\>**FBWFCMD -enable**

    FBWFCMD - FBWF API Example

Copyright (c) 2006 SJJ Embedded Micro Solutions LLC., All Rights Reserved

FBWF Enabled. Reboot system for change to take effect

C:\>**FBWFCMD –addex c: \windows**

FBWFCMD - FBWF API Example
Copyright (c) 2006 SJJ Embedded Micro Solutions LLC., All Rights Reserved

\windows on drive c: is now excluded.
Reboot system for change to take effect

C:\>**FBWFCMD –commit c: \windows\fba\fbalog.txt**

FBWFCMD - FBWF API Example
Copyright (c) 2006 SJJ Embedded Micro Solutions LLC., All Rights Reserved

\windows\fba\fbalog.txt on drive c: has been committed to the disk.

15. Reboot the target, the FBALOG.TXT file should contain the change.

5 Summary

FBWF adds a brand new solution to architecting an XP Embedded image with the ability to allow selected files to write-through to the disk. Taking control of the Filter is an important part of the OS image design to allow for updates and administrative setup. Controlling FBWF can be performed via a pre-built command line utility or through the FBWF API. The FBWF API allows you to add FBWF management function into your custom applications.