DiskCleaner - A Windows service

Clément Nicolas

September 18, 2014

1 Project

DiskCleaner is a project for ETNA school. The aim is to install a Windows service that cleans a directory, which path is stored in a registry key.

1.1 Technology used

For this, we use the Windows API.

2 ReadMe

The file with the logs of the last call is "C:\AutoClean.log"
Three commands are available :

2.1 Commands

- **setpath** : AutoClean.exe -p path_to_clean

It sets the path to clean in the register, for example if you put "D:\Users\me\test", this directory will be cleaned

- **install** : AutoClean.exe -i

It installs the service on you system. It is now available in your services.msc

- **delete** : AutoClean.exe -d

It deletes the service of your system. It will disappear from you services.msc

2.2 Install The Programm

- Run the **AutoClean_setup.exe**

2.3 Install The Service

- Go into the setup directory, and run a cmd.exe with administrator rights
- Call the **setpath** and **install** (the order doesn't matter).

2.4 Run AutoClean

- Go in the services.msc
- Find the description **Service de nettoyage ETNA**
- Click on the **start** button, or right-click – start
- Then you can stop by clicking on the **stop** button, or right-click – stop
- The directory you entered is cleaned recursively !

2.5 Delete The Service

- Call the **delete** command to remove the service.

2.6 Uninstall The Programm

- In the setup directory, simply run uninst*.exe

3 Functions

3.1 Main

The main function.

```
int     main(int argc, char* argv[])
{
        BOOL logfileready = TRUE;
        SERVICE_TABLE_ENTRY table[] = { { SERVICE_NAME, ServiceMain }, { NULL, NULL } };

        g_LogFile = CreateFile(SERVICE_LOG_FILE, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
            FILE_ATTRIBUTE_NORMAL, NULL);
        if (g_LogFile == INVALID_HANDLE_VALUE)
        {
                logfileready = FALSE;
                fprintf(stderr, "Main => CreateFile() failed with : %d\nThe service won't
                    log correctly...\n", GetLastError());
        }

        if (argc > 1)
        {
                if (strcmp(argv[1], "-i") == 0)
                        return (launchInstall(logfileready));
                else if ((strcmp(argv[1], "-p") == 0) && (argc > 2))
                        return (launchSetPath(argv[2], logfileready));
                else if (strcmp(argv[1], "-d") == 0)
                        return (launchDelete(logfileready));
                else
                {
                        fprintf(stderr, "Bad argument. Expected :\n\t[-i] to
                            install\n\t[-d] to delete\n\t[-p path] to set a path to
                            clean");
                        return (EXIT_FAILURE);
                }
        }
        else
                StartServiceCtrlDispatcher(table);

        if (!CloseHandle(g_LogFile))
                fprintf(stderr, "Main => CloseHandle failed with : %d\n", GetLastError());

        return (EXIT_SUCCESS);
}
```

3.2 launchInstall

launchInstall. This function launch the installation of the service.

```
int launchInstall(BOOL logfileready)
{
	if (!InstallMyService())
	{
		if (logfileready)
		{
			writeInLogFile("Main = > InstallMyService() failed with : ",
			    GetLastError());
			writeInLogFile("\r\n", ERROR_SUCCESS);
		}
		else
			fprintf(stderr, "Main => InstallMyService() failed with : %d\n",
			    GetLastError());
		return(EXIT_FAILURE);
	}
	if (logfileready)
		writeInLogFile("Main => InstallMyService() succeeded\r\n", ERROR_SUCCESS);
	else
		fprintf(stdout, "Main => InstallMyService() succeeded.\n");
	return (EXIT_SUCCESS);
}
```

3.3 launchSetPath

launchSetPath. This function launch the treatment to set the path.

```
int	launchSetPath(char *path, BOOL logfileready)
{
	if (!setPathToClean(path))
	{
		if (logfileready)
		{
			writeInLogFile("Main = > setPathToClean() failed with : ",
			    GetLastError());
			writeInLogFile("\r\n", ERROR_SUCCESS);
		}
		else
			fprintf(stderr, "Main => setPathToClean() failed with : %d\n",
			    GetLastError());
		return(EXIT_FAILURE);
	}
	if (logfileready)
		writeInLogFile("Main => setPathToClean() succeeded\r\n", ERROR_SUCCESS);
	else
		fprintf(stdout, "Main => setPathToClean() succeeded.\n");
	return (EXIT_SUCCESS);
}
```

3.4 launchDelete

launchDelete. This function launch the deletion of the service.

```
int     launchDelete(BOOL logfileready)
{
        if (!DeleteMyService())
        {
                if (logfileready)
                {
                        writeInLogFile("Main = > DeleteMyService() failed with : ",
                            GetLastError());
                        writeInLogFile("\r\n", ERROR_SUCCESS);
                }
                else
                        fprintf(stderr, "Main => DeleteMyService() failed with : %d\n",
                            GetLastError());
                return(EXIT_FAILURE);
        }
        if (logfileready)
                writeInLogFile("Main => DeleteMyService() succeeded\r\n", ERROR_SUCCESS);
        else
                fprintf(stdout, "Main => DeleteMyService() succeeded.\n");
        return (EXIT_SUCCESS);
}
```

3.5 InstallMyService

InstallMyService. This function sets up the service.

```
BOOL                    InstallMyService()
{
        char            strDir[MAX_PATH + 1];
        SC_HANDLE       schSCManager;
        SC_HANDLE       schService;

        if (!GetCurrentDirectory(MAX_PATH, strDir))
        {
                writeInLogFile("InstallMyService => GetCurrentDirectory() failed with : ",
                    GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return (FALSE);
        }
        lstrcat(strDir, "\\"SERVICE_BIN_NAME);

        if ((schSCManager = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS)) == NULL)
        {
                writeInLogFile("InstallMyService => OpenSCManager() failed with : ",
                    GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return (FALSE);
        }

        schService = CreateService(schSCManager, SERVICE_NAME, SERVICE_DESCRIPTOR,
                SERVICE_ALL_ACCESS, SERVICE_WIN32_OWN_PROCESS, SERVICE_DEMAND_START,
                    SERVICE_ERROR_NORMAL,
                (LPCSTR)strDir, NULL, NULL, NULL, NULL, NULL);

        if (schService == NULL)
        {
                writeInLogFile("InstallMyService => CreateService() failed with : ",
                    GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return (FALSE);
        }

        if (!CloseServiceHandle(schService))
        {
                writeInLogFile("InstallMyService => CloseServiceHandle() failed with : ",
                    GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return (FALSE);
        }
        return (TRUE);
}
```

3.6 DeleteMyService

DeleteMyService. This function deletes the service.

```
BOOL                    DeleteMyService()
{
        SC_HANDLE       schSCManager;
        SC_HANDLE       hService;

        if ((schSCManager = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS)) == NULL)
        {
                writeInLogFile("DeleteMyService => OpenSCManager() failed with : ",
                    GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return (FALSE);
        }
        if ((hService = OpenService(schSCManager, SERVICE_NAME, SC_MANAGER_ALL_ACCESS))
            == NULL)
        {
                writeInLogFile("DeleteMyService => OpenService() failed with : ",
                    GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return (FALSE);
        }
        if (!DeleteService(hService))
        {
                writeInLogFile("DeleteMyService => DeleteService() failed with : ",
                    GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return (FALSE);
        }
        if (!CloseServiceHandle(hService))
        {
                writeInLogFile("DeleteMyService => CloseServiceHandle() failed with : ",
                    GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return (FALSE);
        }
        return (TRUE);
}
```

3.7 ServiceCtrlHandler

ServiceCtrlHandler. This function is called when an event is received by the service.

```
void WINAPI ServiceCtrlHandler(DWORD Opcode)
{
	switch (Opcode)
	{
	case SERVICE_CONTROL_PAUSE:
		g_ServiceStatus.dwCurrentState = SERVICE_PAUSED;
		break;
	case SERVICE_CONTROL_CONTINUE:
		g_ServiceStatus.dwCurrentState = SERVICE_RUNNING;
		break;
	case SERVICE_CONTROL_STOP:
		g_ServiceStatus.dwWin32ExitCode = 0;
		g_ServiceStatus.dwCurrentState = SERVICE_STOPPED;
		g_ServiceStatus.dwCheckPoint = 0;
		g_ServiceStatus.dwWaitHint = 0;
		if (!SetServiceStatus(g_ServiceStatusHandle, &g_ServiceStatus))
		{
			writeInLogFile("ServiceCtrlHandler => SetServiceStatus() failed
			    with : ", GetLastError());
			writeInLogFile("\r\n", ERROR_SUCCESS);
			return;
		}
		break;
	case SERVICE_CONTROL_INTERROGATE:
		break;
	default:
		break;
	}
}
```

3.8 ServiceMain

ServiceMain. This function is the entering point of the service. It starts the ServiceCtrlHandler and calls the autoClean function.

```
void WINAPI ServiceMain(DWORD argc, LPTSTR *argv)
{
	writeInLogFile("ServiceMain => Starting the service\r\n", ERROR_SUCCESS);

	g_ServiceStatus.dwServiceType = SERVICE_WIN32;
	g_ServiceStatus.dwCurrentState = SERVICE_START_PENDING;
	g_ServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;
	g_ServiceStatus.dwWin32ExitCode = 0;
	g_ServiceStatus.dwServiceSpecificExitCode = 0;

	g_ServiceStatusHandle = RegisterServiceCtrlHandler(SERVICE_NAME,
	    ServiceCtrlHandler);
	if (g_ServiceStatusHandle == (SERVICE_STATUS_HANDLE)0)
	{
		writeInLogFile("ServiceMain => RegisterServiceCtrlHandler() failed with :
		    ", GetLastError());
		writeInLogFile("\r\n", ERROR_SUCCESS);
		return;
	}

	g_ServiceStatus.dwCurrentState = SERVICE_RUNNING;
	g_ServiceStatus.dwCheckPoint = 0;
	g_ServiceStatus.dwWaitHint = 0;
	if (!SetServiceStatus(g_ServiceStatusHandle, &g_ServiceStatus))
	{
		writeInLogFile("ServiceMain => SetServiceStatus() failed with : ",
		    GetLastError());
		writeInLogFile("\r\n", ERROR_SUCCESS);
		return;
	}


	writeInLogFile("ServiceMain => Starting to clean\r\n", ERROR_SUCCESS);
	/*
	** ALGORITHME ICI
	*/
	char *path;

	path = malloc(MAX_PATH * sizeof(char));
	getPathToClean(&path);
	if (path == NULL)
		return;
	if (!autoClean(path))
		return;
	writeInLogFile("ServiceMain => ended without problem\r\n", ERROR_SUCCESS);
}
```

3.9 autoClean

autoClean. This function is the main algorithm of cleaning. It is called recursively in the path given to delete *.tmp or temporary files.

```
BOOL                        autoClean(char *path)
{
        WIN32_FIND_DATA FindFileData;
        HANDLE                  hFind;
        char                    *fullpath = malloc(MAX_PATH * sizeof(char));

        fullpath = _strdup(path);
        lstrcat(fullpath, "\\*.*");
        if ((hFind = FindFirstFile(fullpath, &FindFileData)) == INVALID_HANDLE_VALUE)
        {
                writeInLogFile("autoClean => FindFirstFile() failed with : ",
                    GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return (FALSE);
        }
        do
        {
                if ((FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
                        && (strcmp(FindFileData.cFileName, ".") != 0)
                        && (strcmp(FindFileData.cFileName, "..") != 0))
                {
                        char    *newpath = goInto(path, FindFileData.cFileName);
                        autoClean(newpath);
                }
                else if (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_TEMPORARY)
                {
                        if (!deleteFoundFile(path, FindFileData.cFileName))
                                return(FALSE);
                }
                else if (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_ARCHIVE)
                {
                        if (isExtensionTMP(FindFileData.cFileName))
                                if (!deleteFoundFile(path, FindFileData.cFileName))
                                        return(FALSE);
                }
        } while (FindNextFile(hFind, &FindFileData) != 0);

        if (!FindClose(hFind))
        {
                writeInLogFile("autoClean => FindClose() failed with : ", GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return (FALSE);
        }

        return (TRUE);
}
```

3.10 goInto

goInto. This function is just a cleaner way to build the path to a sub-directory, with the addition of ”
”.

```
char            *goInto(char *path, char *filename) {
        char    *newpath = malloc(MAX_PATH * sizeof(char));

        newpath = _strdup(path);
        lstrcat(newpath, "\\");
        lstrcat(newpath, filename);

        return (newpath);
}
```

3.11 isExtensionTMP

isExtensionTMP. This function tests if a file has a .tmp extension.

```
BOOL            isExtensionTMP(char *filename)
{
        char    ext[5];

        memcpy(ext, &filename[strlen(filename) - 4], 4);
        ext[4] = 0;
        if (strcmp(ext, ".tmp") == 0)
                return (TRUE);
        else
                return (FALSE);
}
```

3.12 deleteFoundFile

deleteFoundFile. This function deletes a file.

```
BOOL    deleteFoundFile(char *path, char *cFileName)
{
        char    *ficToDele = goInto(path, cFileName);

        if (!DeleteFile(ficToDele))
        {
                writeInLogFile("deleteFoundFile => DeleteFile() failed with : ",
                    GetLastError());
                return (FALSE);
        }
        writeInLogFile("deleteFoundFile => \"", ERROR_SUCCESS);
        writeInLogFile(ficToDele, ERROR_SUCCESS);
        writeInLogFile("\" deleted without problem.\r\n", GetLastError());
        return (TRUE);
}
```

3.13 setPathToClean

setPathToClean. This function sets the path to clean in the service registry key.

```
BOOL          setPathToClean(char *path)
{
        HKEY    hKey;

        if (PathFileExists(path))
        {
                if (RegCreateKeyEx(SERVICE_ROOT_KEY, SERVICE_PATH_TO_CLEAN, 0, NULL,
                    REG_OPTION_NON_VOLATILE, KEY_WRITE, NULL, &hKey, NULL) !=
                    ERROR_SUCCESS)
                {
                        writeInLogFile("setPathToClean => RegCreateKey() failed to create
                            \"HKEY_LOCAL_MACHINE", ERROR_SUCCESS);
                        writeInLogFile(SERVICE_PATH_TO_CLEAN, ERROR_SUCCESS);
                        writeInLogFile("\" with : ", GetLastError());
                        writeInLogFile("\"\r\n", ERROR_SUCCESS);
                        return (FALSE);
                }
                if (RegSetValueEx(hKey, "path", 0, REG_SZ, path, strlen(path)) !=
                    ERROR_SUCCESS)
                {
                        writeInLogFile("setPathToClean => RegSetValueEx() failed with : ",
                            GetLastError());
                        writeInLogFile("\r\n", ERROR_SUCCESS);
                        return (FALSE);
                }
                if (RegCloseKey(hKey) != ERROR_SUCCESS)
                {
                        writeInLogFile("setPathToClean => RegCloseKey() failed with : ",
                            GetLastError());
                        writeInLogFile("\r\n", ERROR_SUCCESS);
                        return (FALSE);
                }
                writeInLogFile("setPathToClean => The key has been successfully created
                    with path : \"", ERROR_SUCCESS);
                writeInLogFile(path, ERROR_SUCCESS);
                writeInLogFile("\"\r\n", ERROR_SUCCESS);
                return (TRUE);
        }
        else
        {
                writeInLogFile("setPathToClean => \"", ERROR_SUCCESS);
                writeInLogFile(path, ERROR_SUCCESS);
                writeInLogFile("\" is not a valid path, PathFileExists() failed with : ",
                    GetLastError());
                writeInLogFile("\r\n", ERROR_SUCCESS);
                return(FALSE);
        }
}
```

3.14 getPathToClean

getPathToClean. This function gets the path from the service registry key.

```
BOOL          getPathToClean(char **path) {
        HKEY    hKey;
        BYTE    buf[255];
        DWORD   dwType;
        DWORD   dwBufSize;

        if (RegOpenKey(SERVICE_ROOT_KEY, SERVICE_PATH_TO_CLEAN, &hKey) == ERROR_SUCCESS)
        {
                dwBufSize = sizeof(buf);
                dwType = REG_SZ;
                if (RegQueryValueEx(hKey, "path", 0, &dwType, buf, &dwBufSize) ==
                    ERROR_SUCCESS)
                {
                        if (RegCloseKey(hKey) != ERROR_SUCCESS)
                        {
                                writeInLogFile("getPathToClean => RegCloseKey() failed with
                                    : ", GetLastError());
                                return (FALSE);
                        }
                        *path = _strdup(buf);
                        printf("");
                        return (TRUE);
                }
                else
                {
                        writeInLogFile("getPathToClean => RegQueryValueEx() failed with :
                            ", GetLastError());
                        return (FALSE);
                }
        }
        else
        {
                writeInLogFile("getPathToClean => RegOpenKey() failed to open
                    \"HKEY_LOCAL_MACHINE", ERROR_SUCCESS);
                writeInLogFile(SERVICE_PATH_TO_CLEAN, ERROR_SUCCESS);
                writeInLogFile("\" with : ", GetLastError());
                writeInLogFile("\"\r\n", ERROR_SUCCESS);
                return(FALSE);
        }
}
```

3.15 writeInLogFile

writeInLogFile. This function's aim is to log what happened during the execution.

```
BOOL          writeInLogFile(char *log, int errorCode)
{
       int           len;
       int           loglen;
       char    err[6];
       char    *newlog;

       loglen = strlen(log);
       newlog = _strdup(log);
       if (errorCode != ERROR_SUCCESS)
       {
              if (_itoa_s(errorCode, err, 6, 10) != ERROR_SUCCESS)
              {
                     fprintf(stderr, "writeInLogFile => _itoa_s failed with : %d\n",
                         GetLastError());
                     return (FALSE);
              }
              lstrcat(newlog, err);
       }
       loglen = strlen(newlog);

       if (!WriteFile(g_LogFile, newlog, loglen, &len, NULL))
       {
              fprintf(stderr, "writeInLogFile => WriteFile failed with : %d\n",
                  GetLastError());
              return (FALSE);
       }
       return (TRUE);
}
```