Automated import / export for CMDFlash slave files.

On Microsoft IIS Platforms it's now possible to automate the packing/unpacking of slave's files.

CmdFlash.ini file is automatically created at the cmdflash.exe start (if the plugin is enabled on your tool)
CmdFlash.ini file will be created in the same folder as the cmdflash.exe (if you are using Vista or Win7 run cmdflash.exe as administrator, this will avoid folder virtualization)

You need:
-to ask us for the plugin activation (free of charge)
-provide us a fixed IP address of your server that will manage this service
-master and slaves subscriptions must be valid

Only Mater and Slave ID Information are dispatched to the CMD server, nothing else. So we can not see your modified files.

CmdFlash.ini is a text file and can be edited
Cmdflash interface MUST be connected to the computer
Cmdflash.exe MUST be running

Every second the CmdFlash.exe application will poll information from the ini file and executes the commands.
Results and eventual errors will be reported in the same ini file.

Flow Chart

Your server side…
-Fill the Export or Import fields inside the cmdflash.ini file
-Fill the Command field with EXPORT or IMPORT inside the cmdflash.ini file
-Start to POLL the Command field until a TRUE o FALSE will be returned from the cmdflash application
-if FALSE you can get the error message reading the LASTERROR filed

CMDFLASH.INI layout

```
[FileInfo]
Version = 3.2.0.76
UserId = 3

[Status]
Command =
LastError =

[FileExport]
InputFilePath =
OutputFileE2RomPath =
OutputFileIRomPath =
OutputFileXRomPath =
OutputFileInfoPath =

[FileImport]
InputFilePath =
OutputFilePath =
InputFileE2RomPath =
InputFileIRomPath =
InputFileXRomPath =
```

[FileInfo] Version and UserId MUST NOT Be modified
[Status] is the main section, it's used to post commands and to get back results and errors
Command : EXPORT, IMPORT Results : TRUE, FALSE
LastError is validated only if result is FALSE and describes the error

EXPORT command:
You must provide all the files' paths into the [FileExport] section then put the EXPORT code word into the Command field.
i.e

```
[FileExport]
InputFilePath =  c:\test\from_the_slave_ori.dat    ; original packed file coming from the slave user
OutputFileE2RomPath = c:\test\out\e2rom.ori     ; e2prom.ori will be created is the EEPROM section is present
OutputFileIRomPath =  c:\test\out\Irom.ori         ;  Irom.ori will be created is the IROM section is present
OutputFileXRomPath =  c:\test\out\Xrom.ori       ;   Xrom.ori will be created is the XROM section is present
OutputFileInfoPath =  c:\test\out\file_info.txt      ;  file_info.txt will be created, this file contains info provided by the slave user
```

IMPORT command:
You must provide all the files' paths into the [FileIport] section then put the IMPORT code word into the Command field.
i.e

```
[FileImport]
InputFilePath = c:\test\from_the_slave_ori.dat    ; original packed file coming from the slave user
OutputFilePath = c:\test\new_slave_mod.dat      ; modified packed file name that will be created
InputFileE2RomPath =  c:\test\out\e2rom.ori     ; e2prom.ori  if EEPROM section is present
InputFileIRomPath =  c:\test\out\Irom.ori      ;  Irom.ori if IROM section is present
InputFileXRomPath = c:\test\out\Xrom.MOD     ;   Xrom.mod if XROM section is present and calibration maps are stored in it
```

In order to read/write fields you can use GetPrivateProfileString and WritePrivateProfileString Please take a look at your language development user manual.

From Microsoft MSDN:

GetPrivateProfileString

The **GetPrivateProfileString** function retrieves a string from the specified section in an initialization file.

Note This function is provided only for compatibility with 16-bit Windows-based applications. Applications should store initialization information in the registry.

```
DWORD GetPrivateProfileString(
  LPCTSTR lpAppName,
  LPCTSTR lpKeyName,
  LPCTSTR lpDefault,
  LPTSTR lpReturnedString,
  DWORD nSize,
  LPCTSTR lpFileName
);
```

Parameters

lpAppName

[in] Pointer to a null-terminated string that specifies the name of the section containing the key name. If this parameter is NULL, the **GetPrivateProfileString** function copies all section names in the file to the supplied buffer.

lpKeyName

[in] Pointer to the null-terminated string specifying the name of the key whose associated string is to be retrieved. If this parameter is NULL, all key names in the section specified by the *lpAppName* parameter are copied to the buffer specified by the *lpReturnedString* parameter.

lpDefault

[in] Pointer to a null-terminated default string. If the *lpKeyName* key cannot be found in the initialization file, **GetPrivateProfileString** copies the default string to the *lpReturnedString* buffer. If this parameter is NULL, the default is an empty string, "".

Avoid specifying a default string with trailing blank characters. The function inserts a null character in the *lpReturnedString* buffer to strip any trailing blanks.

> **Windows Me/98/95:** Although *lpDefault* is declared as a constant parameter, the system strips any trailing blanks by inserting a null character into the *lpDefault* string before copying it to the *lpReturnedString* buffer.

lpReturnedString

[out] Pointer to the buffer that receives the retrieved string.

> **Windows Me/98/95:** The string cannot contain control characters (character code less than 32). Strings containing control characters may be truncated.

nSize

[in] Size of the buffer pointed to by the *lpReturnedString* parameter, in characters.

lpFileName

[in] Pointer to a null-terminated string that specifies the name of the initialization file. If this parameter does not contain a full path to the file, the system searches for the file in the Windows directory.

Return Values

The return value is the number of characters copied to the buffer, not including the terminating null character.

If neither *lpAppName* nor *lpKeyName* is NULL and the supplied destination buffer is too small to hold the requested string, the string is truncated and followed by a null character, and the return value is equal to *nSize* minus one.

If either *lpAppName* or *lpKeyName* is NULL and the supplied destination buffer is too small to hold all the strings, the last string is truncated and followed by two null characters. In this case, the return value is equal to *nSize* minus two.

Remarks

The **GetPrivateProfileString** function searches the specified initialization file for a key that matches the name specified by the *lpKeyName* parameter under the section heading specified by the *lpAppName* parameter. If it finds the key, the function copies the corresponding string to the buffer. If the key does not exist, the function copies the default character string specified by the *lpDefault* parameter. A section in the initialization file must have the following form:

```
[section]
key=string
    .
    .
    .
```

If *lpAppName* is NULL, **GetPrivateProfileString** copies all section names in the specified file to the supplied buffer. If *lpKeyName* is NULL, the function copies all key names in the specified section to the supplied buffer. An application can use this method to enumerate all of the sections and keys in a file. In either case, each string is followed by a null

character and the final string is followed by a second null character. If the supplied destination buffer is too small to hold all the strings, the last string is truncated and followed by two null characters.

If the string associated with *lpKeyName* is enclosed in single or double quotation marks, the marks are discarded when the **GetPrivateProfileString** function retrieves the string.

The **GetPrivateProfileString** function is not case-sensitive; the strings can be a combination of uppercase and lowercase letters.

To retrieve a string from the Win.ini file, use the **GetProfileString** function.

The system maps most .ini file references to the registry, using the mapping defined under the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\IniFileMapping

This mapping is likely if an application modifies system-component initialization files, such as Control.ini, System.ini, and Winfile.ini. In these cases, the function retrieves information from the registry, not from the initialization file; the change in the storage location has no effect on the function's behavior.

The profile functions use the following steps to locate initialization information:

1. Look in the registry for the name of the initialization file under the **IniFileMapping** key.
2. Look for the section name specified by *lpAppName*. This will be a named value under the key that has the name of the initialization file, or a subkey with this name, or the name will not exist as either a value or subkey.
3. If the section name specified by *lpAppName* is a named value, then that value specifies where in the registry you will find the keys for the section.
4. If the section name specified by *lpAppName* is a subkey, then named values under that subkey specify where in the registry you will find the keys for the section. If the key you are looking for does not exist as a named value, then there will be an unnamed value (shown as **<No Name>**) that specifies the default location in the registry where you will find the key.
5. If the section name specified by *lpAppName* does not exist as a named value or as a subkey, then there will be an unnamed value (shown as **<No Name>**) that specifies the default location in the registry where you will find the keys for the section.
6. If there is no subkey or entry for the section name, then look for the actual initialization file on the disk and read its contents.

When looking at values in the registry that specify other registry locations, there are several prefixes that change the behavior of the .ini file mapping:

- ! - this character forces all writes to go both to the registry and to the .ini file on disk.
- # - this character causes the registry value to be set to the value in the Windows 3.1 .ini file when a new user logs in for the first time after setup.

- ?? @ - this character prevents any reads from going to the .ini file on disk if the requested data is not found in the registry.
- ?? USR: - this prefix stands for **HKEY_CURRENT_USER**, and the text after the prefix is relative to that key.
- ?? SYS: - this prefix stands for **HKEY_LOCAL_MACHINE\SOFTWARE**, and the text after the prefix is relative to that key.

Requirements

Client	Requires Windows "Longhorn", Windows XP, Windows 2000 Professional, Windows NT Workstation, Windows Me, Windows 98, or Windows 95.
Server	Requires Windows Server "Longhorn", Windows Server 2003, Windows 2000 Server, or Windows NT Server.
Header	Declared in Winbase.h; include Windows.h.
Library	Link to Kernel32.lib.
DLL	Requires Kernel32.dll.
Unicode	Implemented as **GetPrivateProfileStringW** (Unicode) and **GetPrivateProfileStringA** (ANSI). Note that Unicode support on Windows Me/98/95 requires Microsoft Layer for Unicode.

WritePrivateProfileString

The **WritePrivateProfileString** function copies a string into the specified section of an initialization file.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should store initialization information in the registry.

```
BOOL WritePrivateProfileString(
  LPCTSTR lpAppName,
  LPCTSTR lpKeyName,
  LPCTSTR lpString,
  LPCTSTR lpFileName
);
```

Parameters

lpAppName

[in] Pointer to a null-terminated string containing the name of the section to which the string will be copied. If the section does not exist, it is created. The name of the section is case-independent; the string can be any combination of uppercase and lowercase letters.

lpKeyName

[in] Pointer to the null-terminated string containing the name of the key to be associated with a string. If the key does not exist in the specified section, it is created. If this parameter is NULL, the entire section, including all entries within the section, is deleted.

lpString

[in] Pointer to a null-terminated string to be written to the file. If this parameter is NULL, the key pointed to by the *lpKeyName* parameter is deleted.

> **Windows Me/98/95:** The system does not support the use of the TAB (\t) character as part of this parameter.

lpFileName

[in] Pointer to a null-terminated string that specifies the name of the initialization file. If the file was created using Unicode characters, the function writes Unicode characters to the file. Otherwise, the function writes ANSI characters.

Return Values

If the function successfully copies the string to the initialization file, the return value is nonzero.

If the function fails, or if it flushes the cached version of the most recently accessed initialization file, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

A section in the initialization file must have the following form:

```
[section]
key=string
.
.
.
```

If the *lpFileName* parameter does not contain a full path and file name for the file, **WritePrivateProfileString** searches the Windows directory for the file. If the file does not exist, this function creates the file in the Windows directory.

If *lpFileName* contains a full path and file name and the file does not exist, **WritePrivateProfileString** creates the file. The specified directory must already exist.
The system keeps a cached version of the most recent registry file mapping to improve performance. If all parameters are NULL, the function flushes the cache. While the system is editing the cached version of the file, processes that edit the file itself will use the original file until the cache has been cleared.

Requirements

Client	Requires Windows "Longhorn", Windows XP, Windows 2000 Professional, Windows NT Workstation, Windows Me, Windows 98, or Windows 95.
Server	Requires Windows Server "Longhorn", Windows Server 2003, Windows 2000 Server, or Windows NT Server.
Header	Declared in Winbase.h; include Windows.h.
Library	Link to Kernel32.lib.
DLL	Requires Kernel32.dll.
Unicode	Implemented as **WritePrivateProfileStringW** (Unicode) and **WritePrivateProfileStringA** (ANSI). Note that Unicode support on Windows Me/98/95 requires Microsoft Layer for Unicode.