

---

## The MPLAB® IDE Debug Tool API

---

### INTRODUCTION

In some circumstances, it is either necessary or desirable to use an MPLAB® IDE debug tool, such as an in-circuit emulator, from an application other than MPLAB IDE. Frequently, this is in an automated test environment, where a testing application executes Microchip device code under the control of the debug tool in a prototype circuit board. This type of environment requires streamlined, low-level programmatic access to the debug tool without the overhead and interference of the MPLAB IDE graphical user interface. For such applications the MPLAB PTI (Production Test Interface), also known as the MPLAB IDE Debug Tool API, provides the best solution.

### OVERVIEW

A Microsoft® Windows® OS application interfaces with an MPLAB IDE debug tool through the MPLAB IDE Debug Tool API. It accesses the API by linking with the Windows DLL that supports the required tool. A different API DLL is provided for each MPLAB IDE debug tool, specifically:

- MPLAB REAL ICE™ in-circuit emulator
- MPLAB ICD 2 and 3 in-circuit debuggers
- MPLAB ICE 2000 and 4000 in-circuit emulators
- MPLAB SIM simulator

In addition to debug operation, the MPLAB REAL ICE in-circuit emulator and the MPLAB ICD 3 in-circuit debugger may be used for production programming. The MPLAB ICD 2 in-circuit debugger may not.

All APIs are essentially the same, in that they all have a set of generic functions. These functions will be described in **API Functions**, with the prefix DBT for debug functions and PGM for programming functions. Specific tools may provide additional functions; these functions will have a tool-specific tag as their prefix.

### OPERATIONAL DETAIL

Before a debug tool can be used, its drivers must be installed and it must be physically connected to the PC. Installing MPLAB IDE preinstalls the drivers so that when the tool is connected, Windows OS can find and install the drivers.

Before a debug tool can be used with the library, it will have to be properly configured. Currently, Microchip's debug tools are configured by an INI file. The actual contents of this INI file depend upon the particular tool and can be quite cryptic. Although this file can be edited directly, the best way to configure the tool is to use MPLAB IDE. With MPLAB IDE, the tool can be loaded, configured and tested. Then, when the API DLL is loaded, it will use the INI file that was set up with MPLAB IDE.

If using MPLAB ICE 2000, it will be necessary to directly edit the `MPIce2K.ini` file to disable the reporting of error messages. Add the value `SuppressMessages=1` to the `[Display_Info]` section. Please note that this will also suppress error reporting in MPLAB IDE. If using MPLAB ICD 2, a utility called `ICDConfig.exe` is available to perform the minimum configuration necessary to use this tool.

As mentioned previously, the API provides streamlined, low-level programmatic access to an MPLAB IDE debug tool, such as an ICE or ICD. After the DLL is loaded by the application, the application needs to acquire a software “handle” to the tool in order to control it. This requires configuring the tool for a specific target device so that the tool will know proper code and data memory sizes, addresses and other such minutiae. Once a handle has been obtained, the application may perform a variety of operations with it, such as running the target, stepping it, setting breakpoints and reading and writing to target memory.

In a typical application, the debug tool will be used to execute a compiled program on a Microchip device in a circuit board. Therefore, after the debug tool has been initialized for the correct device and its handle has been acquired, the next step would be to load the device program into the debug tool. The load file function that is provided will read in a file in either Hex, COFF, or ELF format, download the program code contained in it to the debug tool and then program the target device with it. Once the device program is loaded into the target, it can be reset, run, stepped and halted with the respective functions. Usually, after the target has been halted, an application will want to check some memory or registers on the target; a read memory function is provided to perform this operation, as well as a write memory function to write back to the target. Another common operation is to check the current PC value, and an appropriate function is available to do this, too. Finally, when the application is done with the debug tool, the application will need to pass the tool’s handle to the uninitialize function.

## API FUNCTIONS

A description of each function in the API follows. When using the various functions, it is crucial that the return codes be considered for robust operation. Since the library communicates with hardware devices, a communication error can occur during any operation. Furthermore, most debug tools will not allow access to the target while it is running; therefore, most API functions, other than Halt or Reset, should not be called until the target halts. Finally, note that certain functions, such as Run, can be configured to block further execution of the application until the function completes. The effects of this behavior upon the application's user interface will have to be carefully considered by the developer.

API Functions are:

[DBTInitialize](#)

[DBTUninitialize](#)

[DBTLoadFile](#)

[DBTRun](#)

[DBTStep](#)

[DBTHalt](#)

[DBTReset](#)

[DBTGetPC](#)

[DBTSetPC](#)

[DBTSetBreakpoint](#)

[DBTClearBreakpoint](#)

[DBTReadMemory](#)

[DBTWriteMemory](#)

[DBTGetMemoryParameters](#)

[DBTSetOnHaltCallback](#)

[DBTGetSymbolAddress](#)

[MPSIMLoadStimulusFile](#)

[MPSIMCloseStimulusFile](#)

[PGMProgram](#)

[PGMVerify](#)

[PGMBlankCheck](#)

[PGMErase](#)

## DBTInitialize

### SYNTAX

```
int DBTInitialize(LPCTSTR lpszDevice, DWORD dwFlags,  
void* pParameters, HTOOL* hTool)
```

### PARAMETERS

#### **lpszDevice**

[in] Pointer to a NULL-terminated string containing the Microchip device name, e.g., PIC18F452.

#### **dwFlags**

[in] Currently two flags are used for the MPLAB ICE 4000. These are MPF\_ABIDE\_DBGEXEC and MPF\_UPDATE\_DBGEXEC.

There are two flags for the MPLAB REAL ICE in-circuit emulator and MPLAB ICD 3 in-circuit debugger. These are MPF\_DEBUGGER and MPF\_PROGRAMMER.

#### **pParameters**

[in] Currently unused; must be NULL.

#### **hTool**

[out] Pointer to an HTOOL to contain the handle to the debug tool that may be used to call the other functions in the library.

### RETURN VALUES

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

### REMARKS

Call this function to initialize the MPLAB Hardware Tool library. This function creates and initializes all of the MPLAB COM objects that are necessary to use the hardware tool. When done with the tool, the handle must be passed to DBTUninitialize.

For the MPLAB ICE 4000, the MPF\_ABIDE\_DBGEXEC flag informs the API to run using an out- of-date debug executive. Typically, the DBTInitialize call will fail if the debug executive on the disk, either I4KDSCDE.bin or DE4K30B0.bin, is a newer version than the one currently programmed into the ICE firmware. Specifying this flag will indicate DBTInitialize to accept and use the older version. The MPF\_UPDATE\_DBGEXEC flag, on the other hand, will cause MPLAB ICE 4000 to program its firmware with the newer version of the debug exec.

For debug tools that can also program, use the MPF\_DEBUGGER flag to use the debug tool as a debugger and the MPF\_PROGRAMMER flag to use the debug tool as a programmer.

## DBTUninitialize

### SYNTAX

```
void DBTUninitialize(HTOOL hTool)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be uninitialized.

## RETURN VALUES

None

## REMARKS

All MPLAB IDE objects that were created to use the tool will be destroyed. The handle will no longer be usable after a call to DBTUninitialize; using it may cause an error.

## DBTLoadFile

### SYNTAX

```
int DBTLoadFile(HTOOL hTool, DWORD dwFlags, LPCTSTR lpszFileName
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Currently there is only one flag defined: MPF\_NO\_PROGRAM.

#### **lpszFileName**

[in] Pointer to a constant null-terminated string containing the full path and file name of the file to be loaded.

## RETURN VALUES

If an error occurred, the return value will be negative. If the file was successfully loaded, then the function will return zero.

## REMARKS

This function loads the program code from the file specified by `lpszFileName` into memory and automatically downloads the code to target memory. Hex, COFF and ELF formats are supported.

For the MPLAB ICD 2, MPLAB ICD 3 and MPLAB REAL ICE in-circuit emulator, this involves automatically programming the target with the loaded code, unless the `MPF_NO_PROGRAM` flag is used.

## DBTRun

### SYNTAX

```
int DBTRun(HTOOL hTool, DWORD dwFlags)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Flag bits to control the run operation. Currently, there is only one flag defined, `MPF_NO_WAIT`.

## RETURN VALUES

If an error occurred, the return value will be negative. If the `MPF_NO_WAIT` flag was specified, and the run began successfully, then the return value will be zero. Otherwise, the return value will be one of the following:

- `MPS_HALT` – The target processor halted, usually due to an instruction

- **MPS\_USER** – The halt caused as a result of the DBTHalt function being called
- **MPS\_BREAK** – The halt was caused by a breakpoint being executed
- **MPS\_WDT** – The halt was caused by a Watchdog Timer time-out

## REMARKS

Calling this function causes the debug tool to run the target processor. If the **MPF\_NO\_WAIT** flag is given for **dwFlags**, **DBTRun** returns immediately after starting the debug tool to run. However, this application requires that the calling process use a Windows OS message loop to translate and dispatch any Windows OS messages. This is necessary because the debug tools may use Windows OS timer messages that need to be processed in the main thread. The callback function that is specified in a call to **DBTSetOnHaltCallback** will then be called when the debug tool halts. If this flag is not specified, then this function will not return until the debug tool halts.

For debug tools that can also program, **DBTInitialize** must have been called with the **MPF\_DEBUGGER** flag or this function will fail.

## DBTStep

### SYNTAX

```
int DBTStep(HTOOL hTool, DWORD dwFlags)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Currently unused; must be 0.

### RETURN VALUES

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

## REMARKS

This function causes the debug tool to execute the next instruction, i.e. the instruction at the current value of the PC.

For debug tools that can also program, **DBTInitialize** must have been called with the **MPF\_DEBUGGER** flag or this function will fail.

## DBTHalt

### SYNTAX

```
int DBTHalt(HTOOL hTool, DWORD dwFlags)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Flag bits to control the halt operation. Currently, there is only one flag defined, **MPF\_NO\_WAIT**.

## RETURN VALUES

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

## REMARKS

Calling this function causes the debug tool to halt the target processor. If the `MPF_NO_WAIT` flag is given for `dwFlags`, `DBTHalt` returns immediately after issuing the halt command to the debug tool. The callback function that is specified in a call to `DBTSetOnHaltCallback` will then be called when the target processor actually halts. If this flag is not specified, then this function will not return until the target acknowledges halting.

For debug tools that can also program, `DBTInitialize` must have been called with the `MPF_DEBUGGER` flag or this function will fail.

## DBTReset

### SYNTAX

```
int DBTReset(HTOOL hTool, DWORD dwFlags)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Flag bits to control the reset. The currently defined flags are `MPF_FULL` and `MPF_MCLR`.

## RETURN VALUES

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

## REMARKS

This function performs a reset of the target processor. The behavior of the various types of resets is dependent upon the debug tool in use. Typically, if the `MPF_FULL` flag is specified, all registers will be set to their reset values. If the `MPF_MCLR` flag is specified, a master clear only reset will be performed.

For debug tools that can also program, `DBTInitialize` must have been called with the `MPF_DEBUGGER` flag or this function will fail.

## DBTGetPC

### SYNTAX

```
int DBTGetPC(HTOOL hTool, DWORD* dwPC)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwPC**

[out] Pointer to a `DWORD` value to hold the current PC value.

## RETURN VALUES

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

## REMARKS

This function retrieves the current value of the PC and returns it in `dwPC`. Currently, it should be called only when the target is halted.

For debug tools that can also program, `DBTInitialize` must have been called with the `MPF_DEBUGGER` flag or this function will fail.

## DBTSetPC

### SYNTAX

```
int DBTSetPC(HTOOL hTool, DWORD dwPC)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwPC**

[in] Address at which to set the target's PC.

## RETURN VALUES

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

## REMARKS

This function will change the target's PC value to the specified address. No instructions are executed, and no other registers are changed. This should be called only when the target is halted.

For debug tools that can also program, `DBTInitialize` must have been called with the `MPF_DEBUGGER` flag or this function will fail.

## DBTSetBreakpoint

### SYNTAX

```
int DBTSetBreakpoint(HTOOL hTool, DWORD dwAddress)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwAddress**

[in] DWORD value containing the address at which to set the breakpoint.

## RETURN VALUES

If the breakpoint was set successfully, the return value will be zero. If an error occurred, the return value will be negative. The following error codes have special meaning for `DBTSetBreakpoint`:

- `MPE_NO_BREAKS` - There are no breakpoints available

**REMARKS**

This function may fail if the debug tool has a limited number of breakpoints, and all of them are currently used. In this case, it will return `MPE_NO_BREAKS`.

For debug tools that can also program, `DBTInitialize` must have been called with the `MPF_DEBUGGER` flag or this function will fail.

**DBTClearBreakpoint****SYNTAX**

```
int DBTClearBreakpoint(HTOOL hTool, DWORD dwAddress)
```

**PARAMETERS****hTool**

[in] Handle to the hardware tool to be used.

**dwAddress**

[in] `DWORD` value containing the address at which to clear the breakpoint.

**RETURN VALUES**

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

**REMARKS**

This function clears a breakpoint set at the specified address. If there is no breakpoint at that address, it simply returns zero.

For debug tools that can also program, `DBTInitialize` must have been called with the `MPF_DEBUGGER` flag or this function will fail.

**DBTReadMemory****SYNTAX**

```
int DBTReadMemory(HTOOL hTool, DWORD dwFlags, DWORD dwAddress,  
    DWORD dwSize, BYTE* pData, DWORD* pdwRead)
```

**PARAMETERS****hTool**

[in] Handle to the hardware tool to be used.

**dwFlags**

[in] Flags for controlling the read operation. Currently, the following flags are defined: `MPF_DATA_MEMORY`, `MPF_CODE_MEMORY`, `MPF_EE_MEMORY`, `MPF_CONFIG_MEMORY`, `MPF_REFRESH_IMAGE`, `MPF_USER_ID_MEMORY`, `MPF_DEVICE_ID_MEMORY` and `MPF_BOOT_MEMORY` (PIC32 MCUs).

**dwAddress**

[in] The target device address at which the read will begin.

**dwSize**

[in] The number of bytes to be read.

**pData**

[out] Pointer to a buffer of `BYTE` to hold the data to be read. This buffer must be at least `dwSize` bytes large.



## **pdwRead**

[out] Pointer to a `DWORD` value that contains the number of bytes that were actually read. This parameter may be `NULL` if the number of bytes read is not needed.

## **RETURN VALUES**

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

## **REMARKS**

This function will read a number of bytes, starting at a specified target address, from the specified memory. The target address is a program memory word address.

If the `MPF_REFRESH_IMAGE` flag is specified, the internal image of memory will be refreshed first by uploading the requested addresses directly from the hardware device. This flag should be used judiciously, because there are serious performance ramifications resulting from the large data transfers.

If the function succeeds, it will specify the number of bytes that were actually read in `pdwRead`; this may be different from the number of bytes that were requested. If the `pData` buffer is not large enough to hold the requested number of bytes, an exception may be thrown.

## **DBTWriteMemory**

### **SYNTAX**

```
int DBTWriteMemory(HTOOL hTool, DWORD dwFlags, DWORD dwAddress,
    DWORD dwSize, BYTE* pData, DWORD* pdwWritten)
```

### **PARAMETERS**

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Flags for controlling the write operation. Currently, the following flags are defined: `MPF_DATA_MEMORY`, `MPF_CODE_MEMORY`, `MPF_EE_MEMORY`, `MPF_CONFIG_MEMORY`, `MPF_NO_PROGRAM`, `MPF_USER_ID_MEMORY`, and `MPF_BOOT_MEMORY` (PIC32 MCUs)

#### **dwAddress**

[in] The target device address at which the write will begin.

#### **dwSize**

[in] The number of bytes to be written.

#### **pData**

[in] Pointer to a buffer of `BYTE` that holds the data to be written. This buffer must be at least `dwSize` bytes large.

#### **pdwWritten**

[out] Pointer to a `DWORD` value that contains the number of bytes that were actually written. This parameter may be `NULL` if the number of bytes written is not needed.

## **RETURN VALUES**

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

**REMARKS**

This function will write a number of bytes, starting at a specified target address, to the specified memory. The target address is a program memory word address.

If the function succeeds, it will specify the number of bytes that were actually written in `pdwWritten`; this may be different from the number of bytes that were requested.

For the MPLAB ICD 2, MPLAB ICD 3 and MPLAB REAL ICE in-circuit emulator, writing to program memory will cause the tool to program the device, unless the `MPF_NO_PROGRAM` flag is used.

**DBTGetMemoryParameters****SYNTAX**

```
int DBTGetMemoryParameters(HTOOL hTool, DWORD dwFlags,
    DWORD* pdwBaseAddress, DWORD* pdwSize, BYTE* pbWordSize,
    BYTE* pbAddressIncrement)
```

**PARAMETERS****hTool**

[in] Handle to the hardware tool to be used.

**dwFlags**

[in] Flags to specify for which memory to retrieve the parameters. Currently, the following flags are defined: `MPF_DATA_MEMORY`, `MPF_CODE_MEMORY`, `MPF_EE_MEMORY`, `MPF_CONFIG_MEMORY`, `MPF_USER_ID_MEMORY` and `MPF_BOOT_MEMORY` (PIC32 MCUs).

**pdwBaseAddress**

[out] The base address of the specified memory. This parameter may be `NULL` if it is not needed.

**pdwSize**

[out] The size, in bytes, of the specified memory. This parameter may be `NULL` if it is not needed.

**pbWordSize**

[out] The size of a word, in bytes, of the specified memory. This parameter may be `NULL` if it is not needed.

**pbAddressIncrement**

[out] The amount by which the addresses of successive program memory words increases. This will be different depending on the device; that is, for PIC16F devices the increment would be 1, for PIC18F devices the increment would be 2, etc.

This parameter may be `NULL` if it is not needed.

**RETURN VALUES**

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

**REMARKS**

This function retrieves the indicated memory information from the database. The memory for which the information is retrieved is specified with the `dwFlags` parameter.

This function may be unnecessary if you already know the start address for the memory and its addressing scheme from a data sheet.

## DBTSetOnHaltCallback

### SYNTAX

```
int DBTSetOnHaltCallback(HTOOL hTool, int (*pCallback)(HTOOL, DWORD))
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **pCallback**

[in] Pointer to the callback function.

### RETURN VALUES

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

### REMARKS

Call this function to pass to the library a callback function to be called when the target halts. This callback will be called when the `MPF_NO_WAIT` flag is specified to a `DBTRun` or `DBTHalt` call. The callback function takes two parameters, an `HTOOL` that identifies the tool that is making the callback, and a `DWORD`, which is tool-specific parameter. The callback function should return zero if it succeeded, or a negative value if some sort of error occurred. What happens if an error occurs depends upon the debug tool that is being used. Note that as mentioned in the description of `DBTRun`, when using the halt callback, the application is required to process Windows OS messages.

For debug tools that can also program, `DBTInitialize` must have been called with the `MPF_DEBUGGER` flag or this function will fail.

## DBTGetSymbolAddress

### SYNTAX

```
int DBTGetSymbolAddress(HTOOL hTool, LPCTSTR lpszSymbol, DWORD dwPC,
    DWORD* pdwAddress)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **lpszSymbol**

[in] A string containing the symbol name to resolve. Symbol names are case sensitive.

#### **dwPC**

[in] The PC value at which to resolve the scope of the symbol.

#### **pdwAddress**

[out] Pointer to a `DWORD` value to hold the symbol address.

### RETURN VALUES

If an error occurred, the function returns a negative value. If the function succeeds, it will return zero.

## REMARKS

This function takes the name of a symbol and determines its address, based upon the passed in PC value. A PC value of zero can be used to find global symbols. If the symbol was not found, `MPE_INVALID_SYMBOL` is returned in `pdwAddress`. In order for this function to work, debugging symbols must have been loaded with the COFF or ELF file format.

## MPSIMLoadStimulusFile

### SYNTAX

```
int MPSIMLoadStimulusFile(HTOOL hTool, DWORD dwFlags,  
    LPCTSTR lpszFileName)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Currently unused; must be 0.

#### **lpszFileName**

[in] Pointer to a constant null-terminated string containing the full path and file name of the file to be loaded.

### RETURN VALUES

If an error occurred, the return value will be negative. If the file was successfully loaded, then the function will return zero.

## REMARKS

This function loads a stimulus control language file ( `.SCL` ) and attaches it to the simulator to direct stimulus input, the simulation of hardware signals. The SCL file can instruct the simulator to inject data into or collect data from the simulation. The simplest method to create a SCL file is to use the simulator in MPLAB IDE. Documentation of the stimulus language can be found on the Web at:

<http://forum.microchip.com/tm.aspx?m=111255>.

## MPSIMCloseStimulusFile

### SYNTAX

```
int MPSIMCloseStimulusFile(HTOOL hTool)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

### RETURN VALUES

If an error occurred, the return value will be negative. If the file was successfully closed, then the function will return zero.

## REMARKS

This function closes the stimulus control language file that was opened by the `MPSIMLoadStimulusFile` function and is currently in use by the simulator.

## PGMProgram

### SYNTAX

```
int PGMProgram(HTOOL hTool, DWORD dwFlags, DWORD dwPgmMemStart,  
               DWORD dwPgmMemEnd)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Flags to control the programming operation and which memories are programmed. These may be either MPF\_AUTO\_PROGRAM, or any combination of MPF\_CODE\_MEMORY, MPF\_EE\_MEMORY, MPF\_CONFIG\_MEMORY, MPF\_USER\_ID\_MEMORY, MPF\_BOOT\_MEMORY (PIC32 MCUs), and MPF\_ERASE.

#### **dwPgmMemStart**

[in] The program memory (code) address at which to start programming.

#### **dwPgmMemEnd**

[in] The program memory (code) address at which to end programming.

### RETURN VALUES

If an error occurred, the return value will be negative. If the programming operation succeeded, then the function will return zero.

### REMARKS

This function programs the target device with the contents of the memory image, usually from a DBTLoadFile. DBTInitialize must have been called with the MPF\_PROGRAMMER flag or this function will fail. To have the programmer choose which memories to program and the code memory start and end addresses, pass the MPF\_AUTO\_PROGRAM value to dwFlags. In this case, the dwPgmMemStart and dwPgmMemEnd parameters will be ignored. To program only specific memories, pass the appropriate flags, OR'd together, to dwFlags, and pass appropriate values to dwPgmMemStart and dwPgmMemEnd. Note that dwPgmMemStart and dwPgmMemEnd must be whole pages and aligned on page boundaries. The MPF\_ERASE flag may also be specified to instruct the programmer to erase the target's memory before programming.

## PGMVerify

### SYNTAX

```
int PGMVerify(HTOOL hTool, DWORD dwFlags)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Currently unused; must be 0.

### RETURN VALUES

If the target memory image is identical to the image that was last sent by the programmer, S\_TRUE is returned. Otherwise, S\_FALSE is returned.

## REMARKS

This function verifies that the image in the target device's memory matches what was last sent by the programmer. DBTInitialize must have been called with the MPF\_PROGRAMMER flag or this function will fail.

## PGMBlankCheck

### SYNTAX

```
int PGMBlankCheck(HTOOL hTool, DWORD dwFlags)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Currently unused; must be 0.

### RETURN VALUES

If the target memory is blank, S\_TRUE is returned. Otherwise, S\_FALSE is returned.

## REMARKS

DBTInitialize must have been called with the MPF\_PROGRAMMER flag or this function will fail.

## PGMErase

### SYNTAX

```
int PGMErase(HTOOL hTool, DWORD dwFlags)
```

### PARAMETERS

#### **hTool**

[in] Handle to the hardware tool to be used.

#### **dwFlags**

[in] Currently unused; must be 0.

### RETURN VALUES

If an error occurred, the return value will be negative. If the target was successfully erased, then the function will return zero.

## REMARKS

DBTInitialize must have been called with the MPF\_PROGRAMMER flag or this function will fail.

---

**Note the following details of the code protection feature on Microchip devices:**

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

---

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

#### **Trademarks**

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, rfPIC, SmartShunt and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Hampshire, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, nanoWatt XLP, PICkit, PICDEM, PICDEM.net, PICtail, PIC<sup>32</sup> logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Total Endurance, TSHARC, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2009, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

**QUALITY MANAGEMENT SYSTEM**  
**CERTIFIED BY DNV**  
**== ISO/TS 16949:2002 ==**

*Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*



---

## WORLDWIDE SALES AND SERVICE

---

### AMERICAS

#### Corporate Office

2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 480-792-7200  
Fax: 480-792-7277  
Technical Support:  
<http://support.microchip.com>  
Web Address:  
[www.microchip.com](http://www.microchip.com)

#### Atlanta

Duluth, GA  
Tel: 678-957-9614  
Fax: 678-957-1455

#### Boston

Westborough, MA  
Tel: 774-760-0087  
Fax: 774-760-0088

#### Chicago

Itasca, IL  
Tel: 630-285-0071  
Fax: 630-285-0075

#### Cleveland

Independence, OH  
Tel: 216-447-0464  
Fax: 216-447-0643

#### Dallas

Addison, TX  
Tel: 972-818-7423  
Fax: 972-818-2924

#### Detroit

Farmington Hills, MI  
Tel: 248-538-2250  
Fax: 248-538-2260

#### Kokomo

Kokomo, IN  
Tel: 765-864-8360  
Fax: 765-864-8387

#### Los Angeles

Mission Viejo, CA  
Tel: 949-462-9523  
Fax: 949-462-9608

#### Santa Clara

Santa Clara, CA  
Tel: 408-961-6444  
Fax: 408-961-6445

#### Toronto

Mississauga, Ontario,  
Canada  
Tel: 905-673-0699  
Fax: 905-673-6509

### ASIA/PACIFIC

#### Asia Pacific Office

Suites 3707-14, 37th Floor  
Tower 6, The Gateway  
Harbour City, Kowloon  
Hong Kong  
Tel: 852-2401-1200  
Fax: 852-2401-3431

#### Australia - Sydney

Tel: 61-2-9868-6733  
Fax: 61-2-9868-6755

#### China - Beijing

Tel: 86-10-8528-2100  
Fax: 86-10-8528-2104

#### China - Chengdu

Tel: 86-28-8665-5511  
Fax: 86-28-8665-7889

#### China - Hong Kong SAR

Tel: 852-2401-1200  
Fax: 852-2401-3431

#### China - Nanjing

Tel: 86-25-8473-2460  
Fax: 86-25-8473-2470

#### China - Qingdao

Tel: 86-532-8502-7355  
Fax: 86-532-8502-7205

#### China - Shanghai

Tel: 86-21-5407-5533  
Fax: 86-21-5407-5066

#### China - Shenyang

Tel: 86-24-2334-2829  
Fax: 86-24-2334-2393

#### China - Shenzhen

Tel: 86-755-8203-2660  
Fax: 86-755-8203-1760

#### China - Wuhan

Tel: 86-27-5980-5300  
Fax: 86-27-5980-5118

#### China - Xiamen

Tel: 86-592-2388138  
Fax: 86-592-2388130

#### China - Xian

Tel: 86-29-8833-7252  
Fax: 86-29-8833-7256

#### China - Zhuhai

Tel: 86-756-3210040  
Fax: 86-756-3210049

### ASIA/PACIFIC

#### India - Bangalore

Tel: 91-80-3090-4444  
Fax: 91-80-3090-4080

#### India - New Delhi

Tel: 91-11-4160-8631  
Fax: 91-11-4160-8632

#### India - Pune

Tel: 91-20-2566-1512  
Fax: 91-20-2566-1513

#### Japan - Yokohama

Tel: 81-45-471- 6166  
Fax: 81-45-471-6122

#### Korea - Daegu

Tel: 82-53-744-4301  
Fax: 82-53-744-4302

#### Korea - Seoul

Tel: 82-2-554-7200  
Fax: 82-2-558-5932 or  
82-2-558-5934

#### Malaysia - Kuala Lumpur

Tel: 60-3-6201-9857  
Fax: 60-3-6201-9859

#### Malaysia - Penang

Tel: 60-4-227-8870  
Fax: 60-4-227-4068

#### Philippines - Manila

Tel: 63-2-634-9065  
Fax: 63-2-634-9069

#### Singapore

Tel: 65-6334-8870  
Fax: 65-6334-8850

#### Taiwan - Hsin Chu

Tel: 886-3-6578-300  
Fax: 886-3-6578-370

#### Taiwan - Kaohsiung

Tel: 886-7-536-4818  
Fax: 886-7-536-4803

#### Taiwan - Taipei

Tel: 886-2-2500-6610  
Fax: 886-2-2508-0102

#### Thailand - Bangkok

Tel: 66-2-694-1351  
Fax: 66-2-694-1350

### EUROPE

#### Austria - Wels

Tel: 43-7242-2244-39  
Fax: 43-7242-2244-393

#### Denmark - Copenhagen

Tel: 45-4450-2828  
Fax: 45-4485-2829

#### France - Paris

Tel: 33-1-69-53-63-20  
Fax: 33-1-69-30-90-79

#### Germany - Munich

Tel: 49-89-627-144-0  
Fax: 49-89-627-144-44

#### Italy - Milan

Tel: 39-0331-742611  
Fax: 39-0331-466781

#### Netherlands - Drunen

Tel: 31-416-690399  
Fax: 31-416-690340

#### Spain - Madrid

Tel: 34-91-708-08-90  
Fax: 34-91-708-08-91

#### UK - Wokingham

Tel: 44-118-921-5869  
Fax: 44-118-921-5820