010 Editor

Edit Anything

v10.0

Reference Manual

SWEETSCAPE SOFTWARE

www.sweetscape.com

Copyright © 2003-2019

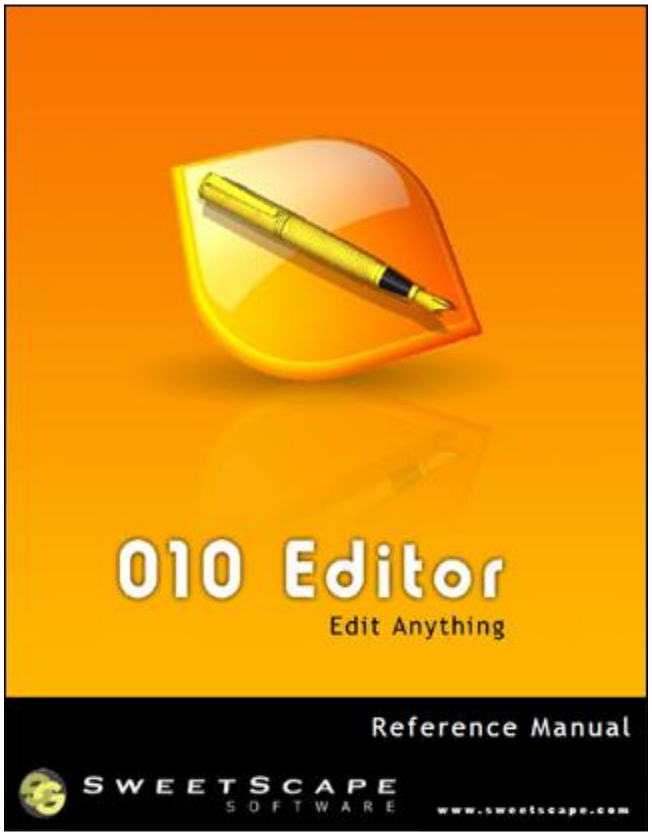
© 2003-2019 SweetScape Software
Charlottetown, PEI, Canada

No part of this book may be used or reproduced in any form or by any means, electronic or otherwise, without the prior written permission of SweetScape Software.

While every precaution has been taken in the preparation of this book, the publisher and the author assume no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information or instructions contained herein. It is further stated that the publisher and author are not responsible for any damage or loss to your data or your equipment that results directly or indirectly from your use of this book.

Table of Contents

Copyright © 2003-2019 SweetScape Software



What is 010 Editor?

010 Editor is a professional text editor and hex editor designed to quickly and easily edit the contents of any file on your computer. This software can edit text files including Unicode files, batch files, C/C++, XML, etc. but where 010 Editor excels is in editing binary files. A **binary file** is a file that is computer-readable but not human-readable (a binary file will appear as garbled characters if opened in a text editor). A **hex editor** is a program that allows you to view and edit the individual bytes of binary files and advanced hex editors including 010 Editor also allow you to edit the bytes of hard drives, floppy drives, memory keys, flash drives, CD-ROMs, processes, etc. Here are just some of the benefits of using 010 Editor:

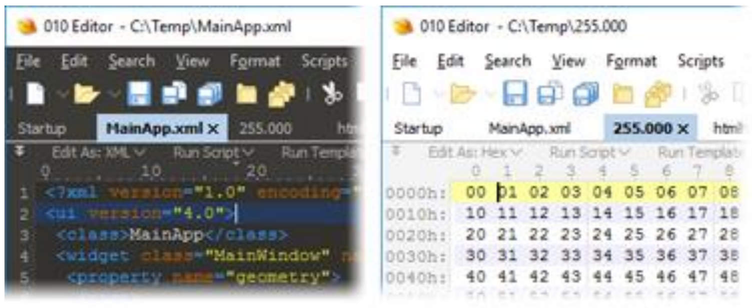
- View and edit **any binary file** on your hard drive (unlimited file size) and **text files** including Unicode files, C/C++, XML, PHP, etc.
- Unique **Binary Templates** technology allows you to understand any binary file format.
- Find and fix problems with **hard drives**, floppy drives, memory keys, flash drives, CD-ROMs, **processes**, etc.
- **Analyze** and edit text and binary data with powerful tools including Find, Replace, Find in Files, Replace in Files, Binary Comparisons, Checksum/Hash Algorithms, Histograms, etc.
- Powerful **scripting** engine allows automation of many tasks (language is very similar to C).

- Easily download and install Binary Templates and Scripts others have shared using the 010 Editor Repository.
- **Import** and **export** your binary data in a number of different formats.

010 Editor's unique **Binary Templates** technology allows you to *understand* the bytes of a binary file by presenting you with the file parsed into an easy-to-use structure. For an example of how Binary Templates work, open any ZIP, BMP, or WAV file on your computer and a Binary Template will automatically be run on the file. Binary Templates are easy to write and look similar to C/C++ structures except they are *very* powerful and can be configured to parse *any* binary format. A repository of Templates that other people have written is available using the Repository Dialog. For more information on Binary Templates see Introduction to Templates and Scripts.

The hex editor built into 010 Editor can load files of any size instantly, and features unlimited undo and redo on all editing operations. The editor can even copy or paste huge blocks of data between files instantly. A Portable version of 010 Editor is also available for Windows for running 010 Editor from USB keys. Try 010 Editor and we're sure you'll agree that 010 Editor is the most powerful hex editor available today!

Themes


010 Editor - C:\Temp\MainApp.xml
File Edit Search View Format Scripts
Startup MainApp.xml 255.000
Edit As: XML Run Script Run Templ
010 Editor - C:\Temp\255.000
File Edit Search View Format Scripts
Startup MainApp.xml 255.000 html
Edit As: Hex Run Script Run Templat

```
<?xml version="1.0" encoding=
<ui version="4.0">
  <class>MainApp</class>
  <widget class="MainWindow" n
   <property name="geometry">
```

```
0000h:  00 01 02 03 04 05 06 07 08
0010h:  10 11 12 13 14 15 16 17 18
0020h:  20 21 22 23 24 25 26 27 28
0030h:  30 31 32 33 34 35 36 37 38
0040h:  40 41 42 43 44 45 46 47 48
```

010 Editor is available in a number of Themes including a dark theme (shown above left) and a light theme (shown above right). Themes can be chosen on the Welcome dialog or on the Theme/Color Options dialog.

Getting Started

For more information on how 010 Editor works see:

- Introduction to Number Systems
- Introduction to Byte Ordering
- Introduction to the Data Engine

To begin editing files see:

- Introduction to Editing

For information on how Binary Templates and Scripts can be used to edit files see:

- Introduction to Templates and Scripts

To easily install Templates and Scripts other people have shared see:

- Introduction to the Repository

010 Editor can be downloaded and used free for 30 days. After that time, the program must be registered. For more information see:

- How to Buy 010 Editor

If you have any questions or problems, information on contacting SweetScape Software can be found here:

- How to Get Support

Thank you and we hope you enjoy using 010 Editor!

Related Topics:
Introduction to the Repository
Introduction to Templates and Scripts
Introduction to Number Systems
Introduction to the Data Engine
Introduction to Editing
How to Buy 010 Editor
How to Get Support
Theme/Color Options
Using the Portable Version

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Introduction to Number Systems

When editing raw hex data, 010 Editor uses a variety of different number systems including decimal, hexadecimal, octal, and binary. Each number system has a different '*base*' that is used to convert from a set of digits to a numeric value. For example, the digits '246' can be converted to a number using base 10 by $2*10^2 + 4*10 + 6 = 246$. In general, if the *n* digits of a number *A* are numbered where A_0 is the right-most digit, A_1 is the digit to the left and so on, then the value of a number of base *B* is calculated:

$$A_{n-1}*B^{n-1} + A_{n-2}*B^{n-2} + \ldots + A_1*B + A_0$$

The following is a list of the 4 number systems used:

- **Decimal -** Numbers are represented as base 10. The digits may be any number from '0' to '9'. For example, in decimal $153 = 1*10^2 + 5*10^1 + 3$.
- **Hexadecimal -** Numbers are represented as base 16. All the decimal digits are used, plus the letters 'A', 'B', 'C', 'D', 'E', and 'F' are used to represent the numbers 10 through 15. For example, in hexadecimal $3d7 = 3*16^2 + 13*16^1 + 7 = 983$. This system is commonly referred to as *Hex*.
- **Octal -** Numbers are represented as base 8. Only the digits '0' through '7' are used ('8' or '9' is not allowed). For example, the number $2740 = 2*8^3 + 7*8^2 + 4*8^1 + 0 = 1504$.
- **Binary -** Numbers are represented as base 2. Only the digits '0' or '1' can be used. For example, the number $10110 = 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0 = 22$.

Bits and Bytes

A digit of a binary number is also called a '*bit*'. When 8 bits are grouped together, the result is called a '*byte*'. Since a byte has 8 binary digits, it can represent any value from 0 up to 255 inclusive. Every file stored on a disk is stored as a set of bytes. Note that when 4 bits are grouped together (base 2), this can also be represented as a single hexadecimal digit (base 16). For example, binary 0101 = '5' hexadecimal, or binary 1111 = 'F' hexadecimal.

010 Editor is designed specifically for editing the individual bytes of a file. When a file is opened for editing, a Hex Editor Window shows the representation of each byte as a hexadecimal number and as a character (see Introduction to Editing for more information).

Entering Numbers in Text Fields

Almost anywhere a number is entered in 010 Editor (in most text fields, the Inspector, etc.), the program supports input in a number of different formats. Usually, the format of the number is assumed to be decimal (some fields have a *Decimal* and *Hex* toggle beside them - clicking the *Hex* toggle will set the default to be hex). However, the other formats can be entered with a special syntax:

- **Hex -** Use '0x' before the number or ',h', ',x', or 'h' after the number. For example, '0x100' or '3f,h', 'd2,x', or 'FFh' represent hexadecimal numbers.
- **Octal -** Enter ',o' after the number. For example, '377,o' is an octal number.
- **Binary -** Enter ',b' after the number. For example, '0101,b' is a binary number.
- **Decimal -** Enter ',d' after the number. For example, '123,d' is a decimal number.
- **Characters -** Characters can be entered by placing single quotes around the character. For example, 'A' would be converted to the number 65. Most standard C escape sequences are also supported using '\'.

For example, '\n' would be converted to the number 10.

See Introduction to Byte Ordering for information on how to group bytes together to make numbers larger than 255.

The number formats supported in Scripts and Templates are slightly different. See Introduction to Templates and Scripts for more information.

Related Topics:
Introduction to Editing
Introduction to Templates and Scripts
Introduction to Byte Ordering

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Introduction to Byte Ordering

Data on a computer is usually divided into sets of 8 bits, called a *byte* (see Introduction to Number Systems). A byte can store 256 different values but to store larger numbers, a set of bytes must be grouped together. The term '*Endian*' refers to how these bytes are grouped together.

- **Little Endian -** In little-endian systems (for example, Intel machines), bytes are stored with the least significant byte first. For example, the hex bytes '2f 75 05' actually represent the number 0x05752f (357679 in decimal). '2f' is the least significant byte and '05' is the most significant byte.
- **Big Endian -** In big-endian systems (for example, Motorola machines), bytes are stored with the most significant byte first. In the same example, the hex bytes '2f 75 05' would represent the number 0x2f7505 (3110149) in decimal.

Which endian is used to convert bytes to numbers is very important and every file in 010 Editor has an endian setting. *LIT* will appear in the Status Bar when the current file is in little-endian mode, and *BIG* will appear in big-endian mode. Also, when in big-endian mode the Toggle Endian button in the Tool Bar will be highlighted. Most tools and the Inspector use this endian setting. To change the default endian used for files, use the '*View > Endian*' menu. 010 Editor can be configured to automatically set the endian based on the file extension (see Working with File Interfaces).

Related Topics:
Introduction to Number Systems
Status Bar
Using Tool Bars
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Introduction to the Data Engine

010 Editor contains a powerful data engine and this engine is used on all file and disk operations including reading data, editing, undo, redo, cut, copy, paste, inserting files, etc. As a result of using this engine, the program gains some extremely powerful functionality but also has a few limitations.

Features

One of the post powerful features of the engine is the ability to open any hex file or drive instantly. There is no limit to the size of files that can be opened, but some file systems (including FAT) limit the file size to 2 gigabytes.

Many clipboard operations including Cut, Copy or Paste can usually (but not always) copy hex data instantly (see Using the Clipboard for more information). For example, open the largest file available on the disk, press *Ctrl+A* to select all, *Ctrl+C* to copy, *Ctrl+N* to create a new file, *Ctrl+H* to switch into hex-editing mode, and *Ctrl+V* to paste the data.

Other file operations, such as Insert Bytes, Insert File, Overwrite Bytes, Overwrite File, and Set File Size can operate on hex files instantly as well. For example, create a new hex file using *Ctrl+N* and *Ctrl+H* and then click '*Edit > Set File Size...*'. Type '10000000000' and press *Enter* to create a huge file (NOTE: do not try to save this file unless enough disk space is free).

Another feature gained by using the data engine is unlimited Undo and Redo on all operations. Regardless of the size of data copied, pasted, or deleted, the operation can be undone or redone using *Ctrl+Z* or *Ctrl+Shift+Z*.

Limitations

The limitations of the data engine only apply when copying or pasting large blocks of data to or from the clipboard (larger than 16KB).

The data engine uses a read-on-demand system, meaning data is not read into the editor until it is required. When Copy and Paste operate on large blocks, sometimes only pointers to the data are transferred. As a result, if a large block of memory is copied and then the file is deleted or modified by another program, the copied data may become corrupted (you will be warned when this happens). If a file is open in the editor, make sure to close the file before deleting it from disk using another program.

As stated earlier, when large blocks of data are copied to the clipboard, sometimes only pointers are copied. When the file the data is copied from is closed or modifications are saved, the actual data is then copied to the clipboard. This process is called *Unlinking* and may take some time if very large blocks have been copied. The Status Bar will display a progress bar while the file is being saved or closed. The '*Edit > Clipboard > Clear Clipboards*' menu option can be used to clear the clipboards, removing any large blocks from memory.

Related Topics:
Inserting Files
Using the Clipboard

010 Editor v10.0 Manual - Windows Edition

Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

What's new in Version 10.0?

Version 10.0 - December 6th, 2019

The following is an overview of the new functionality in version 10.0 of 010 Editor:

- A full debugger is now available for finding and fixing problems with Templates and Scripts.
- The debugger can be accessed using the *Debug* menu and includes stepping, breakpoints, watches and a call stack.
- Templates and Scripts are now threaded, meaning other editing operations can be done while a Template or Script is running.
- When using the Text Editor, line numbers and ruler labels are now hidden by default (they can be shown with '*View > Addresses > Show Addresses*' or '*View > Ruler > Show Labels*').
- When line numbers or ruler labels are hidden, hover the mouse over the address column or ruler for a second to display a hint popup with the hidden information.
- '*View > Tabs/Whitespace > Show Whitespace*' now can be used to visualize linefeed types for each line.

The following is a list of all new features in version 10.0 of 010 Editor:

- **Debugger**
  - A full debugger is now included for finding and fixing problems with 010 Editor Templates and Scripts.
  - Added a new *Debug* menu for controlling the debugger.
  - Debugging can be turned on or off using the '*Debug > Debugging Enabled*' menu option.
- **Debugger Program Flow**
  - Scripts or Templates can be run the usual way (for example with '*Scripts > Run Script*' or '*Templates > Run Template*') or by selecting a Script or Template and clicking '*Debug > Start Debugging*'.
  - If debugging is enabled and a breakpoint is hit in the Script or Template, program execution will pause (see the next section for information on breakpoints).
  - When paused a yellow arrow will indicate the current debug active line in the Text Editor.
  - Use '*Debug > Step Over*' to step to the next line of the file, jumping over any functions or structs that are called.
  - Use '*Debug > Step Into*' to step to the next line of the file and step into any functions or structs that are called.
  - Use '*Debug > Step Out*' to execute the rest of the current function or struct and stop at the first statement outside the function or struct.
  - To continue running a paused Script or Template click '*Debug > Continue*', '*Scripts > Continue Script or Template*' or '*Templates > Continue Script or Template*'.
  - To pause a running Script or Template click '*Debug > Pause*'.
  - To stop a running or paused Script or Template click '*Debug > Stop*' or press Shift+Esc (note this has changed from the Esc key in previous versions).
  - Scripts or Templates are now run threaded meaning other editing operations can take place when a Script or Template is running.
  - If stepping to a line in an include file, the include file is automatically opened in the editor.
  - Right-click on a Script or Template and choose *Run to Cursor* from the right-click menu. The Script or Template will run (or continue) and execution will stop at the chosen line or at the first breakpoint encountered.
  - When a Script or Template is stopped, click '*Debug > Step Into*' to start the program and stop at the first executable line.
  - When stepping through a Template and the last line of the Template Results or Variables tab is selected, if any new variables are appended to the table then the selection will be moved to the last created variable.
- **Breakpoints**

- A breakpoint marks a line to stop in the Script or Template and is marked by a red arrow in the left-hand column of the Text Editor.
- Set or clear a breakpoint for the current line using '*Debug > Toggle Breakpoint*' or by left-clicking the left-hand column in the Text Editor.
- If a breakpoint is set on a non-executable line then the breakpoint will be moved to the next line that is executable when the Script or Template executes.
- Breakpoints are persistent (saved to disk) but this can be changed using the Compiling page of the Options dialog.
- If debugging is disabled then no breakpoints will be hit and the breakpoints are displayed as red outlines in the Text Editor.
- If the Script or Template is modified when program execution is paused then breakpoints will be disabled. The breakpoints will be displayed as outlines with a solid arrow head.
- A list of all breakpoints can be found in the Breakpoints tab, which is found in the Inspector tab group or by clicking '*Debug > View Breakpoints*'.
- In the Breakpoints tab, right-click on the table and select *Add Breakpoint* to set a breakpoint by line number.
- All breakpoints in all files can be deleted by clicking '*Debug > Delete All Breakpoints*'.
- The color of breakpoints or the active line marker can be controlled using the Theme/Colors page of the Options dialog.
- Note that breakpoints are not hit when the application is starting up and any files are being reloaded.

- **Variable Hints**
  - When program execution is paused and the mouse is placed over a variable name in the Script or Template, a hint popup will display the value of the variable.
  - When a selection is made in the Script or Template and the mouse is placed over the selection, the selection will be evaluated and the results displayed in a hint popup.
  - Currently only simple functions (sizeof, startof, exists, etc) can be evaluated in a selection and open the Quick Watch dialog to evaluate a selection which contains more complex functions.
  - Note that how variables are scoped can be affected by the Call Stack tab.
  - Variable hints can be turned off using the Compiling page of the Options dialog.
- **Watches**
  - Watches can be set in the *Watch* tab found in the Inspector tab group or by clicking '*Debug > View Watches*'.
  - Add a watch by double-clicking on the first empty line in the Name column or by right-clicking on the Watch tab and choosing *Add Watch*.
  - A watch can be almost any expression or variable name (for example, 'FileSize()-1000' or 'blocks[i].data[10]').
  - Watches are evaluated every time program execution is paused (e.g. a breakpoint is hit) or when the program is stepped to the next line.
  - If the result of a watch is a struct, the struct can be opened and explored similar to the Template Results.
  - To delete a watch use the Delete key or right-click on a watch and choose *Remove Watch*.
  - A single list of watches is kept for the entire application.
  - Note that how variables are scoped can be affected by the Call Stack tab.
- **Quick Watch**
  - Expressions can also be evaluated without creating a watch using the Quick Watch dialog ('*Debug > Quick Watch*').
  - Enter an expression in the *Expression* field and click *Evaluate*.
  - The result of the expression or variable is displayed in the *Value* column.
  - A list of recent expressions is available by clicking the Down arrow in the dialog.
  - Click the *Add Watch* button to add the current expression to the *Watch* tab.
  - If a selection is made in the Text Editor before the Quick Watch dialog is opened, the selection is copied to the Expression field and evaluated.
- **Debugging Runtime Errors**
  - If a runtime error occurs in a Template or Script a popup dialog box will be displayed asking to start the debugger.
  - When debugging errors the cursor is placed on the line that caused the error.
  - Variables can be investigated with Variable Hints in the Text Editor or with watches.
  - Clicking *Continue* or stepping to the next line will stop the Script or Template.
  - Select the *Always use this action* toggle in the popup dialog box to always start the debugger or never start the debugger.
  - Whether the debugger starts on an error can also be controlled with the Compiling page of the Options dialog.

- **Call Stack**
  - The Call Stack is available in the *Call Stack* tab which can be found in the Inspector tab group or by clicking '*Debug > View Call Stack*'.
  - When program execution is paused, the Call Stack lists the functions or structs that were called to reach the current execution point.
  - The current function or struct is listed at the top of the call stack and the function or struct which called that function or struct is listed below it.
  - If execution is not inside a function or struct then *(Main Program)* is listed in the call stack.
  - Double-clicking on a function or struct jumps to the last position inside that function or struct.
  - Double-clicking on a function or struct also makes any local variables inside the function or struct in local scope (this affects any watches or Variable hints in the Text Editor).
- **Debugger Limitations**
  - Currently breakpoints are not hit inside custom read/write/name/comment functions that are called from the Template Results or Variables tab. To debug these functions call them directly inside the Template.
  - Currently breakpoints in on-demand structures are not hit when the structure is created by opening it in the Template Results. To debug these functions trigger creation of the struct directly in the Template by accessing a variable inside the struct.
  - Currently breakpoints are not hit inside the HighlightLineRealtime or HighlightBytesRealtime functions. To debug these functions see the *Using the Debugger* help topic in the manual for sample code to call.
- **Templates and Scripts**
  - A full debugger including breakpoints, watches and call stack is now available for Templates and Scripts.
  - Templates and Scripts are now threaded, meaning other editing operations can be done while a Template or Script is running.
  - When a Template is running click '*Templates > Stop Template*' or press Shift+Esc to cancel the Template.
  - When a Script is running click '*Scripts > Stop Script*' or press Shift+Esc to cancel the Script.
  - On-demand Structures which have arguments are now supported.
  - Custom read functions can now be called on structs with zero size.
  - Custom name/comment functions now work for local variables.
  - After selecting a Script or a Template that has been run, the Variables tab now shows the list of variables created by that Script or Template.
  - If an included file is opened and modified in the editor, the modified version is used when compiling instead of the disk version.
  - Which warnings are displayed in the Output panel can be configured using the *Compiling* page of the Options dialog.
  - When the application is starting up and files are being reloaded, the Output panel shows the results from all Templates that were run.
  - The Template Results panel only shows the results from a syntax highlighting template if the Template was run directly (not as the result of opening a file).
  - Can right-click on the Variables tab and select *Clear* to clear the results from a Script or Template.
  - The InputString function now returns a UTF-8 string.
  - Jump to Template Variable is now only shown on the Editor right-click menu when editing a hex file.
- **Editor**
  - In the Text Editor, line numbers are now hidden by default and can be displayed by clicking '*View > Addresses > Show Addresses*'.
  - When addresses are hidden, place the mouse cursor over the address column for a second to see the line number in a hint popup.
  - When addresses are hidden, a triangle marker indicates the last line in a file (this can be turned off by setting the *Address End Marker* to None in the Theme/Colors Options dialog).
  - When addresses are hidden, a '-' marker indicates lines that are created by word-wrap.
  - In the Text Editor, ruler labels are now hidden by default and can be shown using '*View > Ruler > Show Labels*'.
  - When ruler labels are hidden, place the mouse cursor over the ruler for a second to view the mouse and cursor position in a hint popup.
  - In the Hex Editor, small arrows in the ruler show the current cursor position and can be turned off using '*View > Ruler > Show Arrows*'.
  - '*View > Tabs/Whitespace > Show Whitespace*' now can be used to visualize linefeed types for each line.

    - The different symbols drawn for *Show Whitespace* can be configured using the *Text Editor* page of the Options dialog.
    - Breakpoints can be toggled by clicking the left-most column when editing a Script or Template.
    - When right-clicking on the editor, the cursor is now moved before the right-click menu is shown.
- **General**
    - The shortcut for opening the Base Converter was changed to Ctrl+F11.
    - Updated the visual style of the Windows installer.
    - Using *Import Hex* with Hex Text or *Paste from Hex Text* now supports data with more types of formatting.
- **Options**
    - On the Text Editor page added the *Show Whitespace* section to control how linefeeds are drawn.
    - On the Text Editor page added the *Change Whitespace Symbols* button to control which symbols are drawn for the different types of whitespace.
    - On the Theme/Colors page added an option to control colors of breakpoints and the debug active line.
    - On the Theme/Colors page added an option to control colors of the Address Hover Marker and Address End Marker (a triangle marker displayed on the last line when Show Addresses is turned off).
    - On the Compiling page added the *Configure* button to control which warnings are displayed in the Output panel.
    - On the Compiling page added the *Breakpoints are Persistent* toggle to control whether breakpoints are automatically saved to disk.
    - On the Compiling page added the *Show Variable Hints when Debugging* option to display the value of variables when the mouse is placed over a variable name in the Text Editor.
    - On the Compiling page added the *When errors occur* drop-down menu to control what action is taken when an error occurs in a Script or Template.
    - On the Inspector page added the default date format 'dd/MM/yyyy'.
- **Bugs**
    - Fixed scripts were not given permission to execute functions in DLLs in some cases.
    - Fixed incorrect error message 'Incorrect function' when trying to load a file that does not exist on some machines.
    - Fixed a crash replacing certain empty regular expressions with nothing.
    - Fixed a possible crash editing a text file which contains a very long line.
    - Fixed incorrect size of tabs in the Preferences dialog of the Help application.
    - Fixed Save All does not try to save text in the Calculator to a file.
    - Fixed the Inspector would sometimes not update properly after clicking a Floating Tab Group file and then a Template Results panel in the main window.
    - Fixed possible crash with the Memset function.
    - Fixed possible crash with ReadWString/ReadString functions and very large files.
    - Fixed '*Format > Comment Selection*' now works with Python commenting.
    - Fixed permission issue with the FileSaveRange function.
    - Fixed up some inaccurate error messages when using invalid name/comment functions.
    - Fixed a text color issue with the Output pane after calling the OutputPaneClear function.
    - Fixed when replacing with nothing, sometimes not all replacements were listed in the Replace Results when 2 or more occurrences were found together.
    - Fixed an empty struct could be executed twice in some cases.

For a full list of changes in other versions, see the Release Notes.

Related Topics:
Release Notes

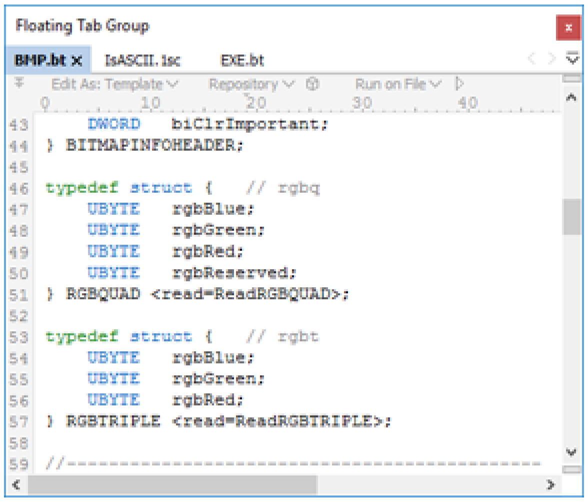
010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Notes for Users of 010 Editor v2.1

This section lists some important notes for users of 010 Editor v2.1 and all previous versions. Some of the functionality has changed in version 3 and 4 and this section should help describe the major changes. See What's New for a full list of changes.

There is no more Code Editor

The Code Editor of 010 Editor has been removed. Now when a script or template is loaded, it is placed into a tab in the *Floating Tab Group*. Whenever a file in the main interface or in the *Floating Tab Group* tab group is marked **bold**, that is the active file in the program and all file operations (save, close, find, select all, etc.) operate on that file. Note that now multiple scripts and templates can be loaded into the interface whereas the old program was limited to one script or template at a time. See Using File Tabs for more information on the Floating Tab Group.



```
    DWORD    biClrImportant;
} BITMAPINFOHEADER;

typedef struct {    // rgbq
    UBYTE    rgbBlue;
    UBYTE    rgbGreen;
    UBYTE    rgbRed;
    UBYTE    rgbReserved;
} RGBQUAD <read=ReadRGBQUAD>;

typedef struct {    // rgbt
    UBYTE    rgbBlue;
    UBYTE    rgbGreen;
    UBYTE    rgbRed;
} RGBTRIPLE <read=ReadRGBTRIPLE>;

//-----------------------------------------------
```

All the functionality of the Code Editor has been moved to other places in the application. New scripts and templates can be generated directly from the Scripts or Templates menu. Any output from the Printf function is now listed in the *Output* tab of the *Output* panel (press *Alt+3* to show the Output panel and press *Alt+3* or Esc to hide the Output panel). After a script is run, any defined variables can be viewed in the *Variables* tab of the Inspector. The list of functions that can be inserted into Scripts or Templates can be accessed on the *Functions* tab of the Inspector (you may have to use the scroll arrows beside the tabs to locate the *Functions* tab). Because multiple Scripts and Templates can now be loaded, how files are executed has changed slightly (see the next section).

Running Scripts and Templates

How Scripts and Templates are run in version 4.0 has changed from previous versions. Scripts and Templates can be run as usual from the menu or command line; however, Scripts and Templates can also be run using the drop-down lists located in the File Bar above each editor. See Running Templates and Scripts for more information.

Using File Tabs

010 Editor version 4 contains more advanced file tabs than version 2. Tabs can be dragged to new positions and can even be dragged to a special *Floating Tab Group*. Click the 'x' icon on each tab to close a file or middle-click on the tab. See Using File Tabs for more information.

Editing text files

Starting with version 3, 010 Editor can now edit text files. Just open a text file with the regular '*File > Open File*' command and 010 Editor should automatically detect whether the file is text or binary. If 010 Editor incorrectly identifies a file, the type of editor can be changed using the *Edit As* drop-down list in the Tool Bar. Also, pressing *Ctrl+H* toggles between hex and text editing mode. See Working with File Interfaces for more information.

Location of Scripts and Templates

In version 3, Scripts and Templates are now stored in the following directories in Windows 7:

- **Scripts** - "Documents\SweetScape\010 Scripts"
- **Templates** - "Documents\SweetScape\010 Templates"

instead of in the 'Program Files' directory in version 2. The locations will differ slightly depending upon which operating system is used.

A complete list of the changes is listed in the What's New help topic.

Related Topics:
Using File Tabs
Using the Inspector
What's New in Version 4.0
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Introduction to Editing

010 Editor is a powerful editor for text and binary files. This section describes the basic methods used to edit files. Typically, editing is performed by opening a file, applying changes, and then saving the changes to the disk. To see how to load files into the editor click on:

- Opening Files

To begin editing the files see either:

- Using the Text Editor or
- Using the Hex Editor

To learn how to use the clipboard to copy and paste data see:

- Using the Clipboard

The editor can be configured to display data in a number of formats. For more information see:

- Working with File Interfaces

Finally, to save files see:

- Saving Files

A number of other ways exist to edit files. See Using the Inspector for a method of interpreting binary data as a number of different data types. For a more powerful way of editing files see Introduction to Templates and Scripts.

010 Editor can also be used to edit hard drives and processes. See the Editing Drives or Editing Processes help topics for more information.

Related Topics:
Introduction to Templates and Scripts
Editing Drives
Editing Processes
Opening Files
Saving Files
Selecting Bytes
Status Bar
Using the Clipboard
Using the Hex Editor
Using the Text Editor
Using the Inspector
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Opening Files

010 Editor contains a number of methods for opening files. Clicking the '*File > Open File...*' menu option, or pressing *Ctrl+O* opens a file dialog box. After selecting a file and clicking the *Open* button, the file will be displayed as either a Text Editor Window or a Hex Editor Window in the main display. See Using the Text Editor or Using the Hex Editor for information on editing the file. When opening a file, if the *Open as read-only* toggle is selected in the dialog, the file will be opened in read-only mode. To control which directory is used when the Open File dialog is shown see the Directory Options dialog.

The editor stores a list of files that have been recently opened. This list can be accessed from the '*File > Open Recent*' menu option, from the Workspace (see Using the Workspace), or from the Startup Page (see the Using the Startup Page). Click a file from the *Open Recent* menu to load it into memory. The number of files in the list can be controlled from the General Options dialog.

Another way of opening files is to use the standard Windows Explorer program. When 010 Editor was installed, enabling the '*Add 010 Editor to explorer right-click menu*' toggle in the install program will allow files to be opened from the Explorer. Right click on a file and select the '*010 Editor*' option. Another method of opening files is to drag-and-drop a file from the Windows Explorer onto a running 010 Editor program. If dragging and dropping an Intel Hex or Motorola S-Record file, the file will automatically be imported (this functionality can be turned off using the Importing Options dialog). Files can also be associated with 010 Editor, meaning that double-clicking a file in the Windows Explorer will open the file in 010 Editor. Intel Hex files or Motorola S-Record files can be automatically associated with 010 Editor by selecting the '*Associate Intel Hex files with 010 Editor*' or '*Associate Motorola S-Records with 010 Editor*' toggle in the install program. For more information on Intel Hex or Motorola S-Record files, see Importing/Exporting Files.

The Workspace also contains a simplified browser that can be used to open files. Files can also be loaded from the *Favorite Files* list or the *Bookmarked Files* list in the Workspace.

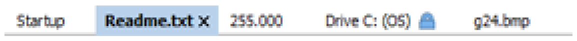
Files can be opened when starting 010 Editor from the command line. See Command Line Parameters for more information.

010 Editor can also be used to edit hard drives and processes. See the Editing Drives or Editing Processes help topics for more information.

Related Topics:
Command Line Parameters
Directory Options
Editing Drives
Editing Processes
General Options
Importing/Exporting Files
Importing Options
Using the Hex Editor
Using the Startup Page
Using the Text Editor
Using the Workspace

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

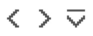
Using File Tabs



Each file, process, or drive that is loaded in 010 Editor is placed into a *File Tab* as shown above. A collection of File Tabs in a single line is referred to as a *Tab Group*. By default all regular files that are loaded are placed into a single Tab Group in the main interface but all Scripts and Templates are loaded into a Floating Tab Group (see the Floating Tab Group section below).

The currently active File Tab will be displayed as **bold** and all file operations (save, close, find, select all, etc.) will be applied to that file. Each File Tab displays a shortened version of the full file name but the file name can be viewed in a hint popup by placing the mouse cursor over top of the Tab (the file name of the active file can also be viewed in the application title bar). Clicking on a File Tab sets that file as the active file in the application.

To close a File Tab either click the 'x' button on the Tab or click the middle mouse button on the Tab (some computer mice have a scroll wheel and a middle click can be performed by pressing down on the scroll wheel). Alternately, File Tabs can be closed using the '*File > Close*' functionality. A popup menu can be accessed by *right-clicking* on the Tab. From the popup menu, click the '*Close*' menu option to close the currently selected file or the '*Close All Except This*' option to close all files except the file that is currently selected. Clicking the '*Copy File Path*' option will place the full directory path of the current file onto the clipboard. See the File Menu for an explanation of other options on the popup menu.

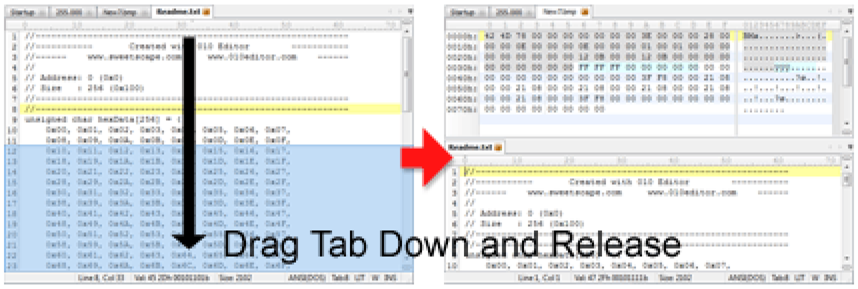


If too many File Tabs are loaded to be viewed all at once, the File Tabs can be viewed by clicking on the left and right arrows (shown above) to scroll through the tabs. Clicking the down arrow displays a list of all File Tabs in that Tab Group. Another way to scroll through the tabs is to position the mouse over the File Tabs and then roll the mouse scroll wheel forward or backward.

File Tabs may be rearranged by clicking and dragging the tabs to new positions. If the File Tab is dragged far enough the tab will tear off and an arrow will appear to indicate where a Tab will be positioned when dropped. Then the File Tab may be dropped onto other Tab Groups (see Using Multiple Tab Groups below).

Using Multiple Tab Groups

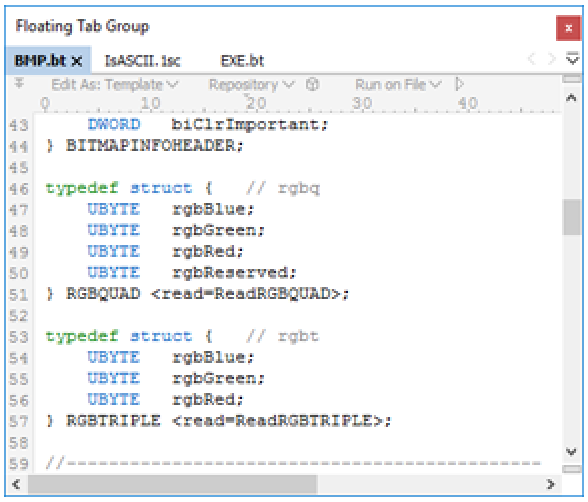
By default all regular files are loaded into a single Tab Group in the main interface and all Scripts and Templates are loaded into a special Floating Tab Group (see the Floating Tab Group section below). The main interface can be split into multiple Tab Groups oriented either horizontally or vertically. When multiple files are loaded, multiple horizontal Tab Groups can be created by right-clicking on a File Tab and selecting *Move to New Horizontal Tab Group*. Another way to generate horizontal Tab Groups is to drag a File Tab towards the bottom of the Window until the blue selection area indicates the lower half of the dialog (see the dialog below).



Similarly, multiple vertical Tab Groups can be generated by right-clicking on a Tab and selecting *Move to New Vertical Tab Group* or by dragging a File Tab to the right until the blue selection area indicates the right side of the window.

Once multiple Tab Groups have been created simply drag a File Tab between the groups. Clicking the '*Window > Merge All Tab Groups*' will place all tabs into a single Tab Group again. See the Window Menu for more information.

Floating Tab Group



```
    DWORD    biClrImportant;
} BITMAPINFOHEADER;

typedef struct {    // rgbq
    UBYTE    rgbBlue;
    UBYTE    rgbGreen;
    UBYTE    rgbRed;
    UBYTE    rgbReserved;
} RGBQUAD <read=ReadRGBQUAD>;

typedef struct {    // rgbt
    UBYTE    rgbBlue;
    UBYTE    rgbGreen;
    UBYTE    rgbRed;
} RGBTRIPLE <read=ReadRGBTRIPLE>;

//---------------------------------------------
```

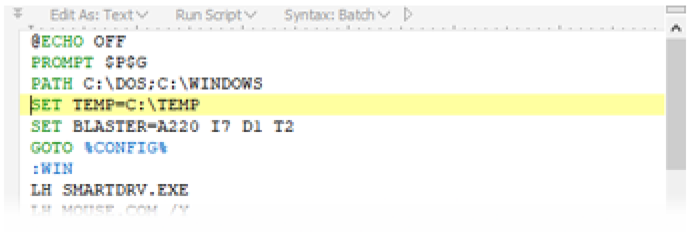
The *Floating Tab Group* is just a floating window that contains a set of File Tabs and can be displayed by clicking the '*View > Floating Tab Group*'. Any file can be moved to the Floating Tab Group by dragging the File Tab to the window until a blue selection area appears and then dropping the Tab. Note that the active file for the application will be marked as **bold** and only one file at a time can be the active file. Hide the Floating Tab Group easily by pressing the Esc key (the Esc key is also used to hide Output windows). When all tabs in the Floating Tag Group are closed, the Floating Tab Group will automatically be hidden but this behavior can be turned off using the Editor Options dialog.

By default all Scripts and Templates that are loaded are placed into this Tab Group but they can be set to load into the main interface by using the Compiling Options. Usually the Floating Tab Group is displayed on top of the application but it can be docked by right-clicking on the window and enabling the *Allow Docking* menu option.

Related Topics:
Editor Options
File Menu
Opening Files
Window Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using the Text Editor



```
@ECHO OFF
PROMPT $P$G
PATH C:\DOS;C:\WINDOWS
SET TEMP=C:\TEMP
SET BLASTER=A220 I7 D1 T2
GOTO %CONFIG%
:WIN
LH SMARTDRV.EXE
```

The Text Editor Window (shown above) is the main method of viewing and editing text files in 010 Editor. Each text file that is loaded is displayed in a File Tab that shows a shortened form of the file name (the full file name can be viewed in the application title bar or in a hint popup displayed by placing the mouse cursor over the File Tab). The main area of the text editor shows the file interpreted as a series of characters (if a byte cannot be shown as a character a square symbol or cross will be displayed). On the left side of the Text Editor Window is a list of addresses and each address indicates the line number or the file position of the first byte on the line. A Ruler, located above the main area, displays the column offset from the start of each line and a small arrow indicates the current cursor position. The File Bar, located above the Ruler, is used to choose how data is displayed in the editor using the *Edit As* section (see Working with File Interfaces). Also, the File Bar can be used to execute Scripts or Templates (see Running Templates and Scripts).

The Cursor

A cursor is shown in the Text Editor Window as a vertical, flashing line and this cursor indicates the current position for inserting, deleting, or editing data. Left-click with the mouse to move the cursor or use the cursor keys on your keyboard (see Editor Keys below). When the Text Editor Window is not active, a vertical gray line, called the shadow cursor, will indicate where the cursor was located. When the editor is in *Overwrite* mode (see Editing Data below) the cursor will be displayed as a thick vertical line and when the editor is in *Insert* mode the cursor will be displayed as a thin vertical line.

Editing Data

To edit data in the editor, position the cursor over the character to edit and type on the keyboard. The result of editing depends on whether the editor is in *Insert* or *Overwrite* mode. In Overwrite mode (*OVR* appears in the Status Bar) the characters typed will replace any existing characters. In Insert mode (*INS* appears in the Status Bar) a new character will be inserted into the file. Note that the current Insert/Overwrite mode is stored separately for text and hex files. The current Insert/Overwrite mode can be changing using the Insert Key (see Editor Keys below) or by clicking INS/OVR in the status bar. Pressing the *Delete* key will delete the current character from the file.

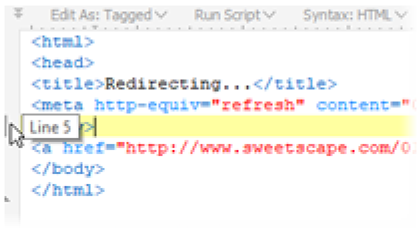
After any edits are made a '*' character will appear in the File Tab to indicate that the file has been modified. Also, if bytes have been inserted a '*' character will appear by the file size in the Status Bar. Use the '*Edit > Undo*' and '*Edit > Redo*' menu options to undo or redo any modifications made to the file. The file can also be edited using the clipboard (see Using the Clipboard).

Editor Keys

The following keys can be used while editing a text file:

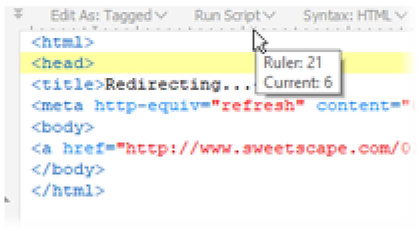
- **Left, Right, Up, Down -** move the cursor in any direction.
- ***Ctrl+Left, Ctrl+Right* -** move the cursor to the next or last word
- ***Ctrl+Up, Ctrl+Down* -** scroll the editor up or down without moving the cursor.
- **Enter -** insert a new line.
- **Home -** move the cursor to the first character on a line.
- **End -** move the cursor to the last character on a line.
- **Ctrl+Home -** move the cursor to the top of the file.
- **Ctrl+End -** move the cursor to the end of the file.
- **Insert -** toggle Insert and Overwrite mode.
- **Delete -** deletes the current character from the file.
- **Ctrl+Shift+Backspace -** deletes the current line if no bytes are selected. If bytes are selected all lines that contain the selection are deleted.
- **Ctrl+Backspace -** deletes the previous word if no bytes are selected.
- **Ctrl+Delete -** deletes the next word if no bytes are selected.
- **Tab -** inserts a tab character or indents a blocks of lines if more that one line is selected at a time.
- **Shift+Tab -** if more than one line is selected, this option unindents the selected lines.

Line Numbers and Ruler



```
<html>
<head>
<title>Redirecting...</title>
<meta http-equiv="refresh" content="
          >
<a href="http://www.sweetscape.com/0
</body>
</html>
```

By default the line numbers for text files are hidden; however, placing the mouse over the left-most address column in the Text Editor shows the current line number in a hint popup. Any lines that were created by word-wrap as described below are marked in the address column with a '-' symbol. When line numbers are hidden a small triangle marker in the column indicates the last line of the file. To show line numbers click '*View > Addresses > Show Addresses*' and to change what is displayed in the address column use the '*View > Addresses*' menu.

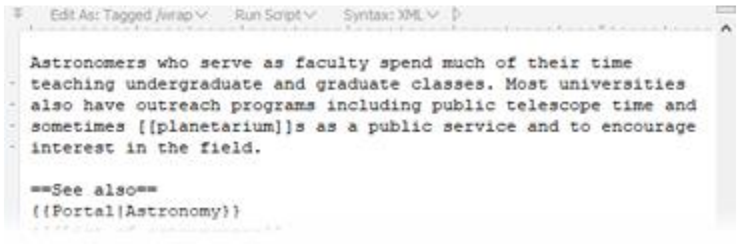


```
<html>
<head>
<title>Redirecting...
<meta http-equiv="refresh" content="
<body>
<a href="http://www.sweetscape.com/0
</body>
</html>
```

A ruler is displayed at the top of the Text Editor to indicate the different columns of the file. For text files the column labels are hidden by default but can be shown by clicking '*View > Ruler > Show Labels*'. When labels are hidden and the mouse is placed over the ruler for a second, a hint popup is displayed showing the column the mouse is over (shown as *Ruler:*) and the column the cursor is on (shown as *Current:*). Small arrows are drawn to indicate the current cursor position and the column the mouse is over and the arrows can be turned off using

'*View > Ruler > Show Arrows*'.

Word Wrap



```
Astronomers who serve as faculty spend much of their time
teaching undergraduate and graduate classes. Most universities
also have outreach programs including public telescope time and
sometimes [[planetarium]]s as a public service and to encourage
interest in the field.

==See also==
{{Portal|Astronomy}}
```

When editing text files that contain long lines, 010 Editor has the ability to automatically wrap lines that extend beyond the edge of the Text Editor Window (see the above image as an example). This is called *Word Wrap* and can be turned on or off by clicking the '*View > Linefeeds > Word Wrap*' menu option, clicking the Word Wrap icon in the Tool Bar, or typing *Ctrl+;*. When Word Wrap is enabled, the text **/wrap** will appear in the *Edit As* section of the File Bar beside the File Interface name and the Word Wrap icon in the Tool Bar will be highlighted. Each line in the text editor that is generated because of a word wrap will have a '-' mark displayed in the Address column on the left side of the editor. As data is edited, the word wraps will automatically be updated.

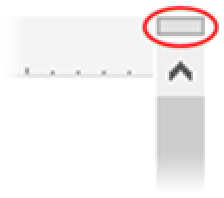
A number of options exist to control how Word Wrap is performed. 010 Editor can automatically turn on Word Wrap when a file is opened that contains long lines. To turn this ability on or off see the '*View > Linefeeds > Initial Wrap State*' menu option. By default wrapping is performed at the edge of the Text Editor Window; however, wrapping can be performed at a specific column using the '*View > Linefeeds > Wrap Width*' menu option. Usually wrapping keeps whole words together but to allow wrapping on any letter within a word use the '*View > Linefeeds > Wrap Method*' menu option. See the View Menu for more information.

When editing a word wrapped line, pressing the *Home* key once will go to the beginning of the line generated by wrapping (marked with '-' in the Address column) and pressing *Home* again will go the beginning of the whole line. Similarly pressing the *End* key once will go the end of the current generated line and pressing *End* again will go the very end of the line. Even when Word Wrap is turned off, 010 Editor will wrap any lines that are longer than the maximum allowed line length. This maximum line length can be controlled in the Text Editor Options dialog.

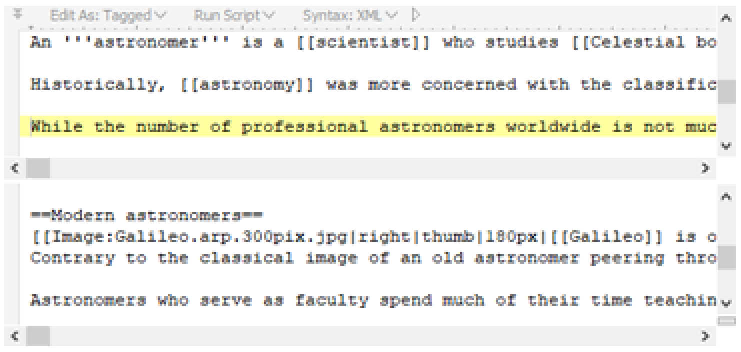
Right-Click Menu

Right-clicking on the Text Editor Window displays a popup-menu of options. This menu is a sub-set of the Edit Menu and see the Edit Menu help topic for more information on each menu option. The Right-Click menu also provides a way of setting the current selection and see Selecting Bytes for more information. When right-clicking on a character that is not selected, note that the cursor is moved to the clicked character before the menu is displayed. To customize the Right-Click Menu click the *Customize...* menu item at the bottom of the Right-Click Menu (see the Menu Options dialog for more information).

Splitting the Text Editor Window



The Text Editor Window may be split into two different areas by clicking the small button above the horizontal scroll bar (see the diagram above) and dragging the mouse down. Once the mouse is released the window will be split into a top and a bottom region (see the diagram below). Having two different regions is useful for editing two different areas of the file at the same time. The areas may be resized by moving the mouse over the line between the areas and dragging the line up or down (the mouse cursor will appear as an up-down arrow). Double-click on the dividing line to return to having a single area. Alternately, the Text Editor Window may be split or un-split using the '*Window > Split Window*' menu option.



```
An '''astronomer''' is a [[scientist]] who studies [[Celestial bo

Historically, [[astronomy]] was more concerned with the classific

While the number of professional astronomers worldwide is not muc
```

```

==Modern astronomers==
[[Image:Galileo.arp.300pix.jpg|right|thumb|180px|[[Galileo]] is o
Contrary to the classical image of an old astronomer peering thro

Astronomers who serve as faculty spend much of their time teachin
```

Column Mode

Text data can be selected by column using the special Column Mode. A quick way to create a column selection is to hold down the *Ctrl* key while dragging the mouse. See the separate Column Mode help topic for more information.

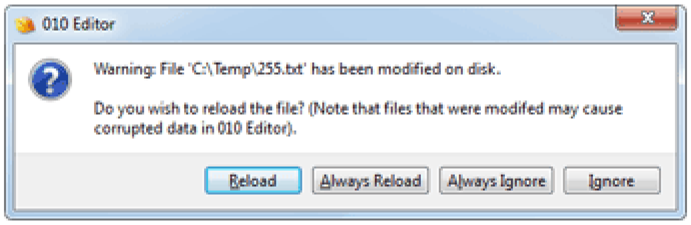
Byte-Order Marks

A *Byte-Order Mark* or *BOM* is a set of special bytes at the beginning of a text file that indicate the file contains data in a certain character set. 010 Editor supports the following byte-order marks:

- **0xFF 0xFE** - Unicode Little-Endian
- **0xFE 0xFF** - Unicode Big-Endian
- **0xEF 0xBB 0xBF** - UTF-8

If a text file starts with any of these BOMs 010 Editor will automatically set the correct File Interface when the file is loaded. When a BOM is present in a file, '+B' will be displayed in the Status Bar and BOMs can be added or removed using the Converting Files dialog. To automatically add byte-order marks to files when they are created, see the File Interface Options dialog.

Reloading Changes


010 Editor

Warning: File 'C:\Temp\255.txt' has been modified on disk.

Do you wish to reload the file? (Note that files that were modifed may cause corrupted data in 010 Editor).

010 Editor checks if a file that is open in the editor has been changed by an external process. If a change is detected the above dialog is displayed. Click *Reload* to load the changes from the external process and note that any current edits on the file will be lost. Click *Ignore* to continue editing the file as usual without loading any external changes. After either of these buttons is clicked and 010 Editor detects new changes to the file, the above dialog will be displayed again. Click *Always Reload* to automatically reload any changes that are detected and 010 Editor checks for new changes every few seconds (this option is useful for editing a log file that is being written to disk from another process). Click *Always Ignore* to ignore any future changes to this file until the file is closed and opened again. Note that the notice about possible corrupted data only applies if a large block of data has been inserted into another file and then that block was modified by an external process (see Introduction to the Data Engine for more information).

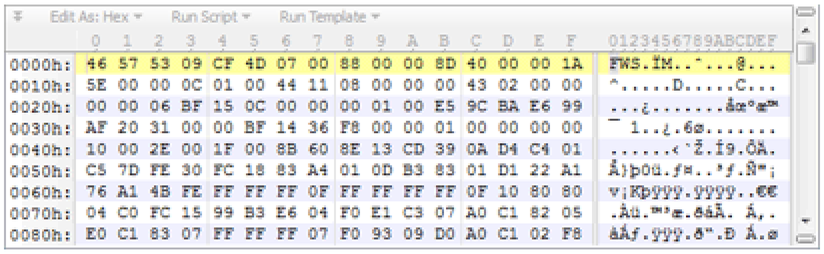
Editor Options

The Editor Options and Text Editor Options dialogs can be used to control a number of options for the Text Editor Window.

Related Topics:
Converting Files
Edit Menu
Editor Options
File Interface Options
Introduction to the Data Engine
Menu Options
Status Bar
Text Editor Options
Running Templates and Scripts
Selecting Bytes
Status Bar
Using Column Mode
Using the Clipboard
View Menu
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using the Hex Editor



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 46 57 53 09 CF 4D 07 00 88 00 00 8D 40 00 00 1A   FWS.ÏM..ˆ...@...
0010h: 5E 00 00 0C 01 00 44 11 08 00 00 00 43 02 00 00   ^.....D.....C...
0020h: 00 00 06 BF 15 0C 00 00 00 01 00 E5 9C BA E6 99   ...¿.......åœºæ™
0030h: AF 20 31 00 00 BF 14 36 F8 00 00 01 00 00 00 00   ¯ 1..¿.6ø.......
0040h: 10 00 2E 00 1F 00 8B 60 8E 13 CD 39 0A D4 C4 01   ......‹`Ž.Í9.ÔÄ.
0050h: C5 7D FE 30 FC 18 83 A4 01 0D B3 83 01 D1 22 A1   Å}þ0ü.ƒ¤..³ƒ.Ñ"¡
0060h: 76 A1 4B FE FF FF FF 0F FF FF FF FF 0F 10 80 80   v¡Kþÿÿÿ.ÿÿÿÿ..€€
0070h: 04 C0 FC 15 99 B3 E6 04 F0 E1 C3 07 A0 C1 82 05   .Àü.™³æ.ðáÃ. Á‚.
0080h: E0 C1 83 07 FF FF FF 07 F0 93 09 D0 A0 C1 02 F8   àÁƒ.ÿÿÿ.ð".Ð Á.ø
```

The Hex Editor Window (shown above) is the main method of viewing and editing binary files in 010 Editor (to edit text files see Using the Text Editor). A Hex Editor Window is displayed for each binary file that is loaded in the editor. Each file is displayed in a File Tab that displays a shortened form of the file name but the full file name can be viewed in the application title bar or in a hint popup displayed by placing the mouse cursor over the File Tab. The Hex Editor Window is divided into a left and a right area. By default, the left area displays the bytes of the file as a series of hexadecimal bytes and the right area displays the bytes as a series of characters (if a byte cannot be shown as a character a '.' will be displayed). To the left of the Hex Editor Window is a list of addresses. Each address indicates the file position of the first byte on the line. At the top of the window a Ruler indicates the byte offsets from the address on that line. The editor can be changed to display data in a number of different formats and to modify how the Hex Editor Window displays data see Working with File Interfaces.

The Cursor

A cursor is displayed in the Hex Editor Window as a vertical, flashing line. The cursor indicates the current position for inserting, deleting, or editing data. Move the cursor with the mouse by clicking anywhere in the main display with the left mouse button. Alternately, the cursor keys can be used to move the cursor (see Editor Keys below). When the cursor is in the left or right areas, the byte the cursor is currently over will be highlighted gray in the other area. Switch between areas by pressing the *Tab* key. When the Hex Editor Window is not focused, a vertical gray line, called the shadow cursor, will indicate where the cursor was located. When the editor is in *Overwrite* mode (see Editing Data below) the cursor will be displayed as a thick vertical line and when the editor is in *Insert* mode the cursor will be displayed as a thin vertical line.

Editing Data

To edit data in the editor, position the cursor over the byte to edit. When the cursor is in the left area (hexadecimal data) enter a valid hexadecimal digit (0 to 9 or A to F) to edit the data. When the cursor is in the right area (character data) enter any character to edit the data.

The result of editing depends on whether the editor is in *Insert* or *Overwrite* mode. In Overwrite mode (*OVR* appears in the Status Bar) the characters typed will replace any existing characters. In Insert mode (*INS* appears in the Status Bar) a new byte will be inserted in the file (NOTE: when editing hexadecimal data, a byte is inserted only when the cursor is over the first digit in the hexadecimal byte). The current Insert/Overwrite mode is stored separately for text and hex files and the current mode can be changing using the Insert Key (see Editor Keys below) or by clicking INS/OVR in the status bar. Pressing the *Delete* key will delete the current byte from the file.

When any edits are made to the file, a '*' character will appear in the title bar to indicate that the file has been modified. If bytes have been inserted, a '*' character will appear by the file size in the Status Bar. The '*Edit >*

Copyright © 2003-2019 SweetScape Software

Undo' and '*Edit > Redo*' menu options can be used to undo or redo any changes made to the file. The file can also be edited using the clipboard (see Using the Clipboard for more information).

Editor Keys

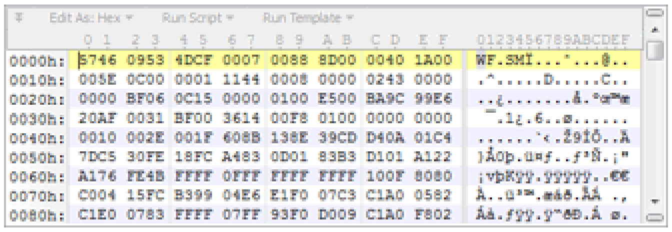
The following keys are available when editing the file:

- **Left, Right, Up, Down -** move the cursor in any direction.
- ***Ctrl+Left, Ctrl+Right* -** move the cursor to the next or last group of bytes.
- ***Ctrl+Up, Ctrl+Down* -** scroll the editor up or down without moving the cursor.
- **Home -** move the cursor to the first byte on a line.
- **End -** move the cursor to the last byte on a line.
- **Ctrl+Home -** move the cursor to the first byte in the file.
- **Ctrl+End -** move the cursor to the end of the file.
- **Insert -** toggle Insert and Overwrite mode.
- **Delete -** deletes the current byte from the file.
- **Tab -** switches between the left and right editing areas.
- ***Alt+Up* -** moves to the previous sector in a hard drive.
- ***Alt+Down* -** moves to the next sector in a hard drive.

Right-Click Menu

A menu of editing options can be accessed by right-clicking on the Hex Editor Window. This menu is sub-set of the Edit Menu (see the Edit Menu for an explanation of each menu option). The Right-Click menu can also be used to set the current selection and see Selecting Bytes for more information. The Right-Click Menu can be customized by right-clicking the editor and selecting the *Customize...* menu option (see the Menu Options dialog for more information).

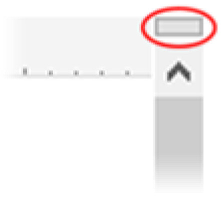
Swapping Bytes



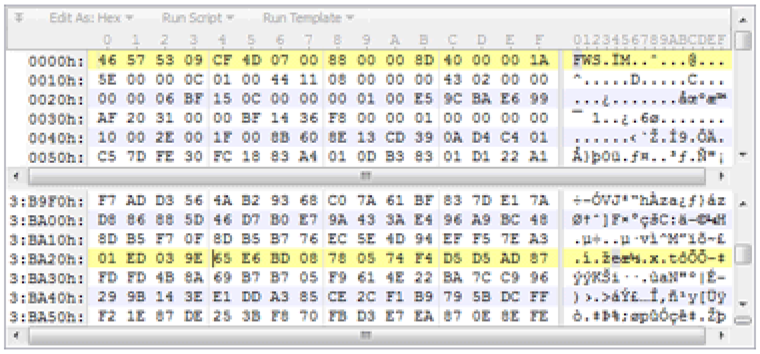
```
        0 1  2 3  4 5  6 7  8 9  A B  C D  E F   0123456789ABCDEF
0000h: 5746 0953 4DCF 0007 0088 8D00 0040 1A00  WF.SMÏ...ˆ...@..
0010h: 005E 0C00 0001 1144 0008 0000 0243 0000  .^.....D.....C..
0020h: 0000 BF06 0C15 0000 0100 E500 BA9C 99E6  ..¿.......å.ºœ™æ
0030h: 20AF 0031 BF00 3614 00F8 0100 0000 0000   ¯.1¿.6..ø......
0040h: 0010 002E 001F 608B 138E 39CD D40A 01C4  ......`‹.Ž9ÍÔ..Ä
0050h: 7DC5 30FE 18FC A483 0D01 83B3 D101 A122  }Å0þ.ü¤ƒ..ƒ³Ñ.¡"
0060h: A176 FE4B FFFF 0FFF FFFF FFFF 100F 8080  ¡vþKÿÿ.ÿÿÿÿ..€€
0070h: C004 15FC B399 04E6 E1F0 07C3 C1A0 0582  À..ü³™.æáð.ÃÁ .‚
0080h: C1E0 0783 FFFF 07FF 93F0 D009 C1A0 F802  Áà.ƒÿÿ.ÿ“ðÐ.Á ø.
```

010 Editor has the ability to visually swap bytes of data in the Hex Editor without modifying the underlying data (for example, compare the image above with the image at the top of this page). Data can be swapped in groups of 2 bytes, 4 bytes, 8 bytes, etc. and the number of bytes is controlled using the '*View > Group By*' menu. To swap data, choose a byte grouping in the '*View > Group By*' other than *Byte* and then enable the '*View > Group By > Swap Little-Endian Bytes by Group*' option. Note that swapping is only performed when the current file is in Little Endian mode and when swapping is enabled '*LIT<>*' will appear in the status bar. When bytes are swapped in the Hex Editor, the selection behaves differently because 010 Editor only supports selecting a contiguous range of bytes. Therefore, the selection may sometimes appear disjointed because of which bytes are selected. Hold down the *Ctrl* key when selecting using the keyboard to ensure that a full group is selected. To swap the bytes in the actual data file, see the Hex Operations dialog.

Splitting the Hex Editor Window



The Hex Editor Window can be split into two different regions by clicking the small button above the horizontal scroll bar (see the diagram above) and dragging the mouse down. After releasing the mouse, the window will be split into a top and a bottom area (see the diagram below). This feature is useful if you are editing two different areas of the file at the same time. Press the *Tab* or *Shift-Tab* keys to move the cursor between the different areas. Move the mouse over the line separating the areas and click and drag the line up or down to resize the areas (the mouse cursor should change to an up-down arrow). Double-click on the separating line to remove the separator and return to having just one area. The Hex Editor Window can also be split or un-split by clicking the '*Window > Split Window*' menu option.



```
            0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
   0000h:  46 57 53 09 CF 4D 07 00 88 00 00 8D 40 00 00 1A   FWS.ÏM..ˆ...@...
   0010h:  5E 00 00 0C 01 00 44 11 08 00 00 00 43 02 00 00   ^.....D.....C...
   0020h:  00 00 06 BF 15 0C 00 00 00 01 00 E5 9C BA E6 99   ...¿.......åœºæ™
   0030h:  AF 20 31 00 00 BF 14 36 F8 00 00 01 00 00 00 00   ¯ 1..¿.6ø.......
   0040h:  10 00 2E 00 1F 00 8B 60 8E 13 CD 39 0A D4 C4 01   ......‹`Ž.Í9.ÔÄ.
   0050h:  C5 7D FE 30 FC 18 83 A4 01 0D B3 83 01 D1 22 A1   Å}þ0ü.ƒ¤..³ƒ.Ñ"¡

3:B9F0h:  F7 AD D3 56 4A B2 93 68 C0 7A 61 BF 83 7D E1 7A   ÷-ÓVJ²"hÀza¿ƒ}áz
3:BA00h:  D8 86 88 5D 46 D7 B0 E7 9A 43 3A E4 96 A9 BC 48   Ø†ˆ]F×°çšC:ä–©¼H
3:BA10h:  8D B5 F7 0F 8D B5 B7 76 EC 5E 4D 94 EF F5 7E A3   .µ÷..µ·vì^M"ïõ~£
3:BA20h:  01 ED 03 9E 65 E6 BD 08 78 05 74 F4 D5 D5 AD 87   .í.žeæ½.x.tôÕÕ-‡
3:BA30h:  FD FD 4B 8A 69 B7 B7 05 F9 61 4E 22 BA 7C C9 96   ýýKŠi··.ùaN"º|É–
3:BA40h:  29 9B 14 3E E1 DD A3 85 CE 2C F1 B9 79 5B DC FF   )›.>áÝ£…Î,ñ¹y[Üÿ
3:BA50h:  F2 1E 87 DE 25 3B F8 70 FB D3 E7 EA 87 0E 8E FE   ò.‡Þ%;øpûÓçê‡.Žþ
```

Template Results

After a Binary Template has been run on the current file, the results will be displayed in the Template Results panel at the bottom of the Hex Editor Window. This panel is sometimes hidden and can be shown by clicking the small button below the horizontal scroll bar and dragging upwards. See Working with Template Results for more information on using the Template Results panel.

Column Mode

Hex data can also be edited using a special Column Mode where columns of bytes can be selected. One way to create a column selection is to hold down the *Ctrl* key while dragging the mouse. See the separate Column Mode help topic for more information.

Reloading Changes

When the editor detects that an open file has been modified by an external process, a dialog is displayed asking to reload the file. See the Reloading Changes section of the Text Editor for more information.

Copyright © 2003-2019 SweetScape Software

Editor Options

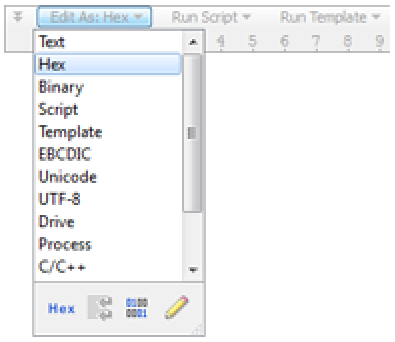
See the Editor Options and Hex Editor Options dialogs for a list of options that can be controlled for the Hex Editor Window.

Related Topics:
Edit Menu
Editor Options
Hex Editor Options
Hex Operations
Introduction to Byte Ordering
Menu Options
Selecting Bytes
Status Bar
Using Column Mode
Using the Clipboard
Using the Text Editor
Working with File Interfaces
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Working with File Interfaces

A *File Interface* is an important concept in 010 Editor. Every file that is loaded is assigned a File Interface and this interface controls how that file is displayed and edited. A File Interface includes the Font, Character Set, Linefeeds/Line Width, Tabs, Addresses, Group By, Division Lines, Areas, Highlighting, Ruler, Status Bar, and Endian settings for a file (basically all options listed at the top of the View Menu). The name of the current file's File Interface is displayed in the *Edit As* area of the File Bar above each file. A different File Interface can be applied to the current file by clicking the *Edit As* area and choosing from the drop-down list or using the '*View > Edit As*' menu.



File Interfaces may be either *text-based*, where the Text Editor Window is used to edit the file, or *hex-based*, where the Hex Editor Window is used to the edit the file. Note that some options are only applicable to one type of File Interface. The *Toggle Hex Interface* button in the Tool Bar will be highlighted when the current file is using a hex-based file interface. Edit the current interface from the View Menu and some, but not all, options can be edited using the File Interface Options dialog as well. Modifying an option for one file modifies that option for all files that use that interface. Any changes to the interface are automatically saved when 010 Editor exits so the interface will appear the same the next time the program loads.

Multiple interfaces can be generated for files and assigned automatically when a file loads. For example, 010 Editor can be configured to always open a file with a certain file extension in Big Endian mode or with the EBCDIC character set. To create new interfaces use the '*View > Edit As*' menu (see the View Menu for more information) or the File Interface Options dialog (see File Interface Options dialog). The '*View > Edit As*' menu contains a list of all available File Interfaces and a checkmark will be placed beside the currently active interface. For more options on how File Interfaces are assigned to files see the Editor Options dialog.

The following list indicates all attributes that are stored with the File Interface:

- **Font -** Controls the font type, size, and style that are used in the editor. File Interfaces may use either the standard Text Editor font, the standard Hex Editor font, or their own custom font.
- **Character Set -** Sets which character set is used to display data in a character area (ASCII, ASCII+OEM, ASCII+ANSI, EBCDIC, Unicode, UTF-8, Macintosh, or various international character sets). See the Character Set Options dialog for more information about character sets.
- **Linefeeds (text only) -** Controls how 010 Editor breaks a text file into a set of lines. Can be used to turn on Word Wrap for a file and control how the wrapping is performed.
- **Line Width (hex only) -** Specifies how many bytes are displayed on each line of the hex editor. The width can be set to 4, 8, 12, 16, or 20 bytes per line or any custom value. Also the editor can be set to automatically determine the maximum number of bytes to display in the window by choosing the *Auto Width* option.
- **Tabs/Whitespace (text only) -** When editing text files the Tabs setting controls how far apart the tab stops are in a file. When a tab character is encountered in a file the next character is displayed at the next tab stop position. The Tabs option also controls how many characters are inserted when the Tab key is pressed. 010 Editor can be set to insert spaces instead of tabs (the default) as well.
- **Addresses -** Controls whether the addresses on the left side display the byte number, sector number,

line number or short number. The address can also be displayed in hex format, decimal format, or octal format or hidden altogether.

- **Group By (hex only)-** Sets how many bytes are grouped together in the Hex Editor Window (the default is one).
- **Division Lines (hex only) -** 010 Editor can draw a set of lines on the Hex Editor Window to visualize how different bytes are grouped together. There are two types of lines that can be drawn: *Division Lines* and *Sector Lines*. By default, a division line is drawn every 4 bytes in a light-gray color and a sector line is drawn for each sector in a dark-gray color when editing hard drives. See the View Menu for more information on Division Lines.
- **Left Area (hex only) -** Controls which numeric format is displayed in the left area of the Hex Editor (Hex, Char, Octal, Binary, or Decimal).
- **Right Area (hex only) -** Controls which numeric format is displayed in the right area of the Hex Editor (Hex, Char, Octal, Binary, Decimal, or Hide).
- **Highlighting -** Controls which bytes are highlighted in the editor. A number of highlighting schemes are available by default but more can be generated through the Highlight Options dialog. Note that Syntax Highlighting is no longer applied using this menu and see the Syntax Highlighting help topic for more information.
- **Ruler -** Specifies whether the Ruler is displayed at the top of the Editor Window and the units of the Ruler can be controlled as well.
- **Status Bar -** Sets what format to display the current file position, file size, and selection size as seen in the status bar.
- **Endian -** Controls which byte-ordering is used for the current editor (see Introduction to Byte Ordering).

See the View Menu for more details about each option.

Related Topics:
Character Set Options
Highlight Options
Using Syntax Highlighting
Using the Hex Editor
Using the Text Editor
Using Tool Bars
View Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Selecting Bytes

Before using any of the clipboard operations such as Cut, Copy, or Paste, a selection must be made on the file (see Using the Clipboard). Selections can be made with either the mouse, the keyboard, or through the Select bar.

To select bytes with the mouse, click a file with the left mouse button and drag the mouse over the file. To select bytes with the keyboard, hold down the *Shift* key and move the cursor with any of the cursor movement keys (see Using the Text Editor or Using the Hex Editor for a list of keys). The bytes that are selected will be displayed with a blue background. The number of bytes selected and the start address of the selection are displayed in the Status Bar along the bottom of the application.

Double-click the mouse and drag on a text file to select by words instead of characters. Triple-click the mouse to select a single line of text or triple-click and drag to select multiple lines of text.

All bytes in the file can be selected at once using the '*Edit > Select All*' menu option. To move the cursor to the start of the selection, right-click on the Editor Window and choose '*Selection > Goto Selection Start*' from the right-click menu. Similarly, to move the cursor to the end of the selection select '*Selection > Goto Selection End*' from the right-click menu.

Bytes can also be selected with the Select bar (see Selecting a Range). The start address and number of bytes of the selection can be viewed at any time by opening the Select bar.

Marking a Selection

An alternate way of setting the selection involves right-clicking on a byte in an editor and then choosing either the '*Selection > Mark Selection Start*' or '*Selection > Mark Selection End*' menu options from the right-click menu. When *Mark Selection Start* or *Mark Selection End* is clicked while no selection has been made, 010 Editor will remember the selection mark for the current file (note that there is no visual indication where the selection marks are). If both the start and end of the selection have been marked, 010 Editor will select those bytes. If a selection is currently active and then either *Mark Selection Start* or *Mark Selection End* is clicked, the selection will be expanded or contracted so that the start or end lies at the correct position. This is a useful way to either enlarge or shrink a selection on a file.

Related Topics:
Selecting a Range
Status Bar
Using the Clipboard
Using the Hex Editor
Using the Text Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using the Clipboard

The clipboard is a temporary scratch pad that can be used to store a block of bytes. The clipboard also makes possible moving data between applications. Most clipboard operations require that a set of bytes be selected in a file (see Selecting Bytes). The following clipboard operations are supported:

- **Copy -** Accessed from the '*Edit > Copy*' menu option, this operation copies the currently selected bytes onto the clipboard. This operation does not modify the file. The keyboard shortcuts *Ctrl+C* and *Ctrl+Ins* can also be used to copy data.
- **Cut -** The '*Edit > Cut*' menu option copies the selected bytes onto the clipboard and then deletes the bytes from the file. The keyboard shortcuts *Ctrl+X* and *Shift+Del* can also be used to cut data.
- **Delete -** Clicking the '*Edit > Delete*' menu option will delete the currently selected bytes from the file. The data on the clipboard will not be modified.
- **Paste -** The '*Edit > Paste*' menu option has two possible effects: When editing text data or when in Insert mode (*INS* will appear in the Status Bar), the bytes on the clipboard are inserted into the file at the current cursor position. When editing hex data in Overwrite mode (*OVR* will appear in the Status Bar), the bytes on the clipboard are written over the bytes in the file, starting at the cursor position. If a selection is made when a Paste operation is performed, the selection will first be deleted from the file and then the bytes will be inserted. The *Insert* key can be used to toggle between Insert and Overwrite mode. Note that the functionality of the Paste command can be changed in the Hex Editor Options dialog. The keyboard shortcuts *Ctrl+V* and *Shift+Ins* can also be used to paste data.
- **Paste Special -** Some applications copy data to the clipboard in a number of different formats. The '*Edit > Paste Special*' menu option is similar to the Paste menu option except that the format to paste can be chosen explicitly. See Using Paste Special for more information.
- **Copy as Hex Text -** Click the '*Edit > Copy As > Copy as Hex Text*' menu option to convert the selected bytes into text characters and copy the result onto the clipboard. For example, copying the bytes 0x2F and 0xB7 as text would result in the string "2FB7" being placed on the clipboard. Use this option to transfer binary data into a text editor or to search for a set of hex bytes in the Find tool.
- **Copy As (*export_type*) -** This set of options provides a quick way to export data to any of the supported export formats (see Importing/Exporting Files) and then copying the exported data to the clipboard.
- **Paste from Hex Text -** The '*Edit > Paste From > Paste from Hex Text*' command is similar to the Paste command, except that the data on the clipboard is automatically converted from hex characters to binary bytes before pasting. For example, if the four characters "17D4" were copied onto the clipboard, then the two binary bytes 0x17 and 0xD4 would be pasted into the file. Note that any characters that are not valid hexadecimal digits will be ignored in the conversion. Use this option to transfer hex bytes from a text editor into 010 Editor.
- **Paste From (*import_type*) -** This set of commands provides a method of importing data that is on the clipboard in any of the supported import formats (see Importing/Exporting Files for a list of all import types). The data is imported and then inserted at the current cursor position.

Multiple Clipboards

010 Editor contains a total of 10 clipboards: the standard Windows clipboard plus 9 custom clipboards. Only one clipboard is active at a time and all Cut, Copy, and Paste commands will operate on that clipboard. The active clipboard will be checked in the '*Edit > Clipboard*' menu. Select another clipboard by clicking an item in the Clipboard menu, or use *Ctrl+0* up to *Ctrl+9*. The active clipboard is also displayed in the Status Bar (see Status Bar for more information).

Clearing Clipboards

Large blocks of memory can easily be copied to or from the clipboard (see Introduction to the Data Engine). Use the '*Edit > Clipboard > Clear Clipboards*' menu option to remove all data from the 10 clipboards. This command is useful to prevent large blocks of data from being copied into memory when a file is saved or closed.

Linux Selection Clipboard

Unix systems have the concept of a selection clipboard where every time a selection is made in any program, the contents are automatically copied to a special clipboard. 010 Editor supports this clipboard on Linux systems and data can be pasted at the current mouse position by clicking the middle mouse button. Note that the position of the cursor does not change after data is pasted using this method. Any data selected in 010 Editor is available to be pasted in other programs using the middle-click technique. Note that the selection clipboard is independent of the regular clipboard used for copy and paste above.

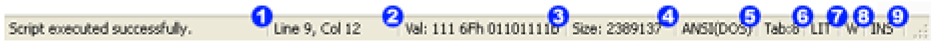
Exiting with Large Blocks

Note that when the program exits, a large block of memory copied to the Windows clipboard is deleted unless the *Leave Large Blocks on Clipboard on Exit* toggle is set in the General Options.

Related Topics:
Edit Menu
Editor Options
General Options
Importing/Exporting Files
Introduction to the Data Engine
Selecting Bytes
Status Bar
Using Find
Using Paste Special

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Status Bar



The Status Bar, located along the bottom of the application, contains useful information about the current file and the current status of the editor. The Status Bar is divided up into a number of panels as listed below:

- **Message Area (1) -** Displays information about the last completed operation. When an important message such as an error or warning is displayed in this field, the message will be displayed with an orange background.
- **File Position / Selection Start (2) -** When no bytes are selected, this panel shows the current address of the cursor in the file. When bytes are selected, this panel displays 'Start:' followed by the starting address of the selection. The address can display the current byte number, sector number, line number or short number and the address can be displayed in hex format, decimal format, or octal format. Set the address format using the '*View > Status Bar > File Position Units*' menu option or the popup menu accessed by right-clicking the Status Bar. Left-click on this panel to bring up the Goto bar.
- **Current Byte / Selection Size (3) -** When no bytes are selected, this panel displays the current value of the byte the cursor is over. The value of the byte is displayed in hex, decimal, and binary formats. The byte value is displayed as an Unsigned Byte but to convert to other formats see the Inspector. When bytes are selected in the file, this panel displays 'Sel:' followed by the size of the current selection. The display format used for the selection size can be controlled using the '*View > Status Bar > Selection Size Units*' menu option or by right-clicking on the Status Bar. Note that when displaying the selection size as a number of lines and less than one line is selected, the number of selected bytes will be displayed instead.
- **File Size (4) -** Shows the size of the file being edited. The size can be displayed in number of bytes, sectors, lines or shorts and the size can be shown in hex format, decimal format, or octal format. To control the displayed format, see the '*View > Status Bar > File Size Units*' menu option or the popup menu accessed by right-clicking the Status Bar. Note that a '*' will appear beside the file size if the size has been changed since the file was opened. Left-click on this panel to open the Set File Size dialog.
- **Character Set and Linefeeds (5) -** Shows which character set is being used to display the current file. The character set depends upon the current File Interface (see Working with File Interfaces) and can be controlled through the '*View > Character Set*' menu option. The following character sets are available by default:
  - *ASCII* - ASCII Character Set
  - *ANSI* - ASCII+ANSI Character Set
  - *OEM* - ASCII+OEM Character Set
  - *EBC* - EBCDIC Character Set
  - *UNI* - Unicode Character Set
  - *UTF8* - UTF-8 Character Set
  - *MAC* - Macintosh Character Set
  - *ARA* - Arabic Windows Character Set
  - *ARA-I* - Arabic ISO Character Set
  - *BAL* - Baltic Windows Character Set
  - *BAL-I* - Baltic ISO Character Set
  - *CH-S* - Chinese Simplified Character Set
  - *CH-T* - Chinese Traditional Character Set
  - *CYR* - Cyrillic Windows Character Set
  - *CYR-R* - Cyrillic KOI8-R Character Set
  - *CYR-U* - Cyrillic KOI8-U Character Set
  - *CYR-I* - Cyrillic ISO Character Set
  - *EEUR* - Eastern Europe Windows Character Set
  - *EEUR-I* - Eastern Europe ISO Character Set
  - *GRE* - Greek Windows Character Set
  - *GRE-I* - Greek ISO Character Set
  - *HEB* - Hebrew Windows Character Set

- *HEB-I* - Hebrew ISO Character Set
- *JAP* - Japanese Shift_JIS Character Set
- *JAP-E* - Japanese EUC-JP Character Set
- *KOR* - Korean EUC-KR Character Set
- *THAI* - Thai Character Set
- *TURK* - Turkish Windows Character Set
- *TURK-I* - Turkish ISO Character Set
- *VIET* - Vietnamese Character Set

Note that the character set list and the text displayed in the status bar for each character set can be modified using the Character Set Options dialog. If the current file is a text file, the current type of linefeeds is displayed in brackets after the character set name. The following linefeed types are supported:

- *DOS* - DOS Linefeeds (CR+LF - 0x0D0A)
- *UNIX* - Unix Linefeeds (LF - 0x0A)
- *Mac* - Macintosh Linefeeds (CR - 0x0D)
- *NEL* - Next Line (0x15 in EBCDIC or 0x0085 in Unicode)
- *FF* - Form Feed (0x000C in Unicode)
- *LS* - Line Separator (0x2028 in Unicode)
- *PS* - Paragraph Separator (0x2029 in Unicode)

If the current file contains a Byte-Order Mark (BOM), this panel will contain '*+B*' after the linefeed type. Click the Character Set panel in the Status Bar to open the Convert dialog to translate the current file to a different character set or linefeed type.

- **Tabs (6) -** If the current file is being edited as a text file, this panel of the Status Bar lists the number of characters per tab stop in the file. Clicking this panel brings up a popup-menu which can be used to control the tab settings (this menu can also be accessed by clicking the '*View > Tabs/Whitespace*' menu option). See the View Menu for more information on other tab settings.
- **Endian (7) -** Indicates which endian is used to interpret the current file. *LIT* means little endian (e.g. Intel machines) and *BIG* means big endian (e.g. Motorola machines). See Introduction to Byte Ordering for more information. 010 Editor can visually swap bytes in the Hex Editor Window without modifying the underlying data. This swapping only occurs in little endian mode and when swapping is enabled, this field will display *LIT<>* (see Swapping Bytes for more information).
- **Clipboard (8) -** A total of 10 clipboards are available for copying and pasting data. This field indicates which clipboard is currently selected. A '*W*' means the default Windows clipboard is active and the numbers '1' through '9' indicate that a user clipboard is active. See Using the Clipboard for more information.
- **Insert Mode (9) -** Displays whether the editor is in Insert (*INS*), or Overwrite (*OVR*) mode. Press the *Insert* key or click this status bar area to toggle between the two states. This mode is used when editing data in the editor, or when pasting data from the clipboard (see Using the Text Editor, Using the Hex Editor and Using the Clipboard for more information). The cursor in the Editor Window will display as a thick line when in Overwrite mode, or a thin line when in Insert mode.

For long operations, a progress bar will usually be displayed in the Status Bar. In most cases, pressing the *Esc* key will cancel the operation.

Related Topics:
Character Set Options
Setting the File Size
Using Goto
Using the Clipboard
Using the Hex Editor
Using the Text Editor
View Menu
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Saving Files

The last step in editing files is to save any modifications to disk. 010 Editor contains a number of different options for saving files:

- **Save -** Save a file using the '*File > Save*' menu option, or by pressing *Ctrl+S*. If an existing file was opened for editing, the changes will be written back to the original file. If a new file was created using '*File > New*' or '*File > Import Hex...*' a file dialog box will be shown to select a file name for the file.
- **Save As -** Saves the current file to a new file name. Click the '*File > Save As...*' menu option or press *Ctrl+Shift+S* to access this command. A new file name must be chosen using the file dialog box that is displayed. The name of the file in the editor will change to the new file name (the name will change in the Title Bar of the application, the File Tab, and anywhere else the file is referenced). To control which directory is used when the Save As dialog is shown see the Directory Options dialog.
- **Save a Copy -** Saves a copy of the current file to a new file. The '*File > Save a Copy...*' menu option can be used to access this command. Select a file name in the file dialog box that is displayed. This command is useful for creating a backup copy of a file that is being edited. The name of the file in the editor will not be changed.
- **Save Selection -** If a selection has been made on the current file (see Selecting Bytes), click the '*File > Save Selection*' menu option to save just the selected bytes to a file. Select a new file to save with the file dialog box that is displayed. The file name of the current file will not be changed.
- **Save All -** Click the '*File > Save All*' menu option to save all modified files to the disk and files marked as read-only will be skipped. Any parameters specified with the '*Tools > Options...*' dialog box will also be saved to disk.

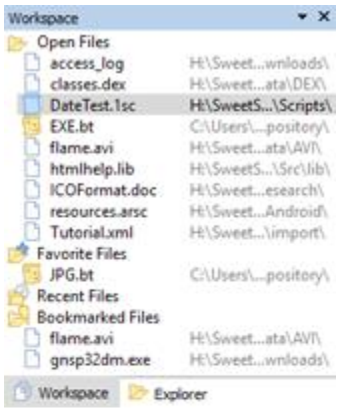
Related Topics:
Directory Options
Opening Files
Selecting Bytes
Using the Hex Editor
Using the Text Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Dock Windows

The 010 Editor main application contains a number of special panels called *Dock Windows*, for example the Workspace, Explorer, Inspector, and the Output Windows. These Dock Windows can be moved to other locations in the application, docked together as a set of tabs, or moved to their own floating windows.

Rearranging Dock Windows



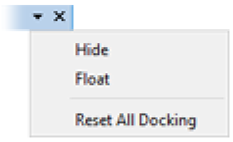
Open Files	
access_log	H:\Sweet...wnloads\
classes.dex	H:\Sweet...ata\DEX\
DateTest.1sc	H:\SweetS...\Scripts\
EXE.bt	C:\Users\...pository\
flame.avi	H:\Sweet...ata\AVI\
htmlhelp.lib	H:\SweetS...\Src\lib\
ICOFormat.doc	H:\Sweet...esearch\
resources.arsc	H:\Sweet...Android\
Tutorial.xml	H:\Sweet...\import\
Favorite Files	
JPG.bt	C:\Users\...pository\
Recent Files	
Bookmarked Files	
flame.avi	H:\Sweet...ata\AVI\
gnsp32dm.exe	H:\Sweet...wnloads\

There are two main ways to rearrange Dock Windows. The first way is to click and drag on the Dock Header at the top of the window. For example for the Workspace above click and drag on the blue bar and doing so will move all of the windows that are docked together as tabs. An animation will show where the Dock Window will be moved to and release the mouse button to complete the docking. The second way is to click and drag on the tab name (in the above diagram the gray Workspace tab). Doing so allows rearranging the tab in the list and if the tab is dragged far enough it will be pulled off and become a separate window. Another way to move a Dock Window to a separate window is to click the down arrow in the Dock Header to access the Dock Menu (shown below) and click *Float*.

Many Dock Windows have an *Allow Docking* option that can be accessed by right-clicking on the window. When this option is turned off the Dock Window will always be a separate floating window. When this option is turned on then the Dock Window can be dragged and docked with the application.

Hiding and Showing Dock Windows

When a Dock Window is docked with the main application the Dock Header is displayed as blue on Windows and Linux or gray in macOS and a down arrow appears in the Dock Header. Clicking on the X button beside the down arrow hides all tabs beneath the Dock Header. To hide an individual tab click the down arrow to access the Dock Menu (shown below) and click the *Hide* menu option.



Individual tabs can also be hidden by clicking and dragging on the tab until it tears off as a separate window and then clicking the X button to hide the separate window. Tabs can also be shown or hidden using the View Menu and the Esc key can be used to hide the Output tabs.

Resetting Dock Windows

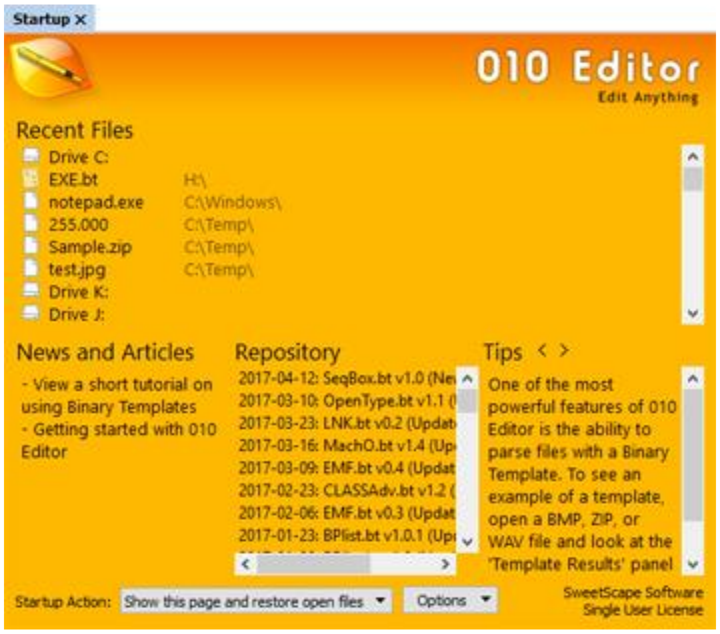
To return the Dock Windows to their original positions when 010 Editor was first installed, click the down arrow in the Dock Header and select '*Reset All Docking*' from the drop-down menu (see the diagram in the above section). Alternately the docks can be reset by using the -resetdocks command line option when 010 Editor is run.

Related Topics:
Command Line Parameters
Using the Inspector
Using the Workspace
View Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using the Startup Page

The Startup page is shown by default when 010 Editor is opened or when all files are closed, but can also be displayed by clicking on the 'View > Other Windows > Startup Page' menu option. This page provides an easy way to open recent files, view application news and updates, and view tips for the application.



Recent Files

Drive C:	
EXE.bt	H:\
notepad.exe	C:\Windows\
255.000	C:\Temp\
Sample.zip	C:\Temp\
test.jpg	C:\Temp\
Drive K:	
Drive J:	

News and Articles

- View a short tutorial on using Binary Templates
- Getting started with 010 Editor

Repository

2017-04-12: SeqBox.bt v1.0 (Ne
2017-03-10: OpenType.bt v1.1 (
2017-03-23: LNK.bt v0.2 (Updat
2017-03-16: MachO.bt v1.4 (Up
2017-03-09: EMF.bt v0.4 (Updat
2017-02-23: CLASSAdv.bt v1.2 (
2017-02-06: EMF.bt v0.3 (Updat
2017-01-23: BPlist.bt v1.0.1 (Up

Tips < >

One of the most powerful features of 010 Editor is the ability to parse files with a Binary Template. To see an example of a template, open a BMP, ZIP, or WAV file and look at the 'Template Results' panel

Startup Action: Show this page and restore open files

The following list describes each of the four areas of the Startup page:

- **Recent Files** - Displays a list of all files, drives, and processes that have been recently opened in 010 Editor. Files are displayed with the file name on the left side and the file path on the right side in a lighter color. Click on an item in the list to load that file. If the file is already opened in the interface the tab containing that file will be activated. To resize the columns of the *Recent Files* list move the mouse cursor between the areas until a resize icon is shown and then click and drag the mouse. To remove a file from the Recent Files list right-click on the file and select the *Remove from Recent Files* menu option, or to remove all files from the list right-click on a file and select *Clear Recent Files List*. Right-clicking on a file a selecting *Copy Path* from the popup menu will copy the full file name including path to the clipboard.
- **News and Articles** - This section lists any application news or updates that the application downloads periodically from the SweetScape Software website. To control how often news and updates are downloaded see the General Options dialog. Clicking on a news item will either load a webpage, help topic, or other window in the application.
- **Repository** - Lists the recent updates to the Repository. See the Updating the Repository help topic for more information about when the Repository is updated. Double-clicking on an item in the list displays more information in the Repository Dialog.
- **Tips** - Displays some hints and tricks to better using 010 Editor. A new tip is shown every time the application is opened or the tips can be viewed by clicking the left or right arrow buttons.

Use the *Startup Action* drop-down list at the bottom of the Startup page to control what the application displays when opened. The default value is *Show this page and restore open files* meaning that all open files are reloaded from the last time 010 Editor was run but the Startup page is focused. Choose *Show this page* to just display the Startup page and not load files, or choose *Restore all open files* to reload all files and focus the last file that was being edited. Choose *Create new file* to automatically create a new file using '*File > New*' (the Startup page is not displayed in this case), or choose *Display empty interface* to not load any files or pages on startup. The *Startup Action* can also be controlled using the General Options dialog.

The *Options* button can be used to control which of the above four areas are displayed. Click the *Options* button and check or uncheck one of the entries to show or hide that section of the page. Click the 'x' button in the *Startup* tab to close the tab or press *Ctrl+W*. Also in the *Options* menu is the toggle *Show Startup Page when All Files are Closed*. When this toggle is enabled and all file tabs are closed, the Startup Page will be shown. If the toggle is disabled a blank interface will be shown and right-click on the interface and choose '*Startup Page*' to display the Startup Page. This option is also available on the Editor Options page. To adjust the theme or colors of the Startup page choose *Change Theme/Colors* from the *Options* menu.

The current license or trial status is displayed in the bottom-right corner of this page. Clicking on the license text displays the Register Dialog.

Related Topics:
Editor Options
General Options
How to Buy 010 Editor
Introduction to the Repository
Opening Files
Updating the Repository
Using the Repository Dialog

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using the Portable Version

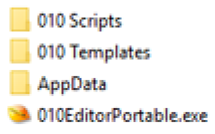
010 Editor is available as a *portable* version meaning the 010 Editor directory can be placed onto a USB key and moved between different computers without having to run an installer on each computer. Currently the portable version is only available on Windows and must be installed using a separate installer found on:

http://www.sweetscape.com/download/010editor/

When the portable version of 010 Editor is being run the title of the application is '010 Editor Portable' and the version will include 'Portable' in the '*Help > About*' dialog. Note that with the portable version of 010 Editor no desktop icon will be created and the '010 Editor' entry will not be added to the Windows Explorer right-click menu.

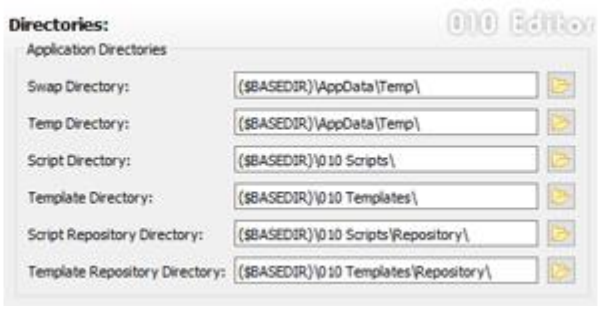
Directory Structure

When 010 Editor Portable is installed the following directory structure will be found on disk:



```
010 Scripts
010 Templates
AppData
010EditorPortable.exe
```

This directory structure can be moved to different locations or different computers. Start the application by running the '010EditorPortable.exe' program (this is equivalent to running the '010Editor.exe' program in the 'AppData' directory). All Scripts and Templates installed from the Script or Template Repository are automatically installed into the '010 Scripts' and '010 Templates' directories. Any other custom Scripts or Templates should be installed into these directories if they are going to be used on different systems. The directory options for the portable version can be controlled by clicking '*Tools > Options...*' and selecting '*Directories*' from the list:



Directories:

Application Directories

Swap Directory:	($BASEDIR)\AppData\Temp\
Temp Directory:	($BASEDIR)\AppData\Temp\
Script Directory:	($BASEDIR)\010 Scripts\
Template Directory:	($BASEDIR)\010 Templates\
Script Repository Directory:	($BASEDIR)\010 Scripts\Repository\
Template Repository Directory:	($BASEDIR)\010 Templates\Repository\

By default all directories are specified offset from ($BASEDIR) which is the directory which contains '010 Scripts', '010 Templates', 'AppData' and '010EditorPortable.exe'. Note that this constant does not exist on non-portable versions of 010 Editor. If a large amount of room is needed for a swap or temporary file these directories could be changed to a location on the local hard-drive.

Copyright © 2003-2019 SweetScape Software

Licensing Issues

The portable version of 010 Editor uses the same license as the regular version of 010 Editor and a special license is not required to run the portable version; however, the license information for the regular version of 010 Editor is stored in the Windows registry but the license information for the portable version is stored in the 'AppData\Config' directory. If the portable version is run with no license installed but a license exists in the Windows registry, you will be asked to copy the license to the portable version if you are the owner of the license. If the license is changed in either the portable or standard version of 010 Editor the license will not automatically be copied over and must be entered using the Register dialog.

Uninstalling

Note that there is also no uninstaller for the portable version of 010 Editor and to uninstall simply delete the directory structure that was created.

Related Topics:
Directory Options
How to Buy 010 Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Editing Drives

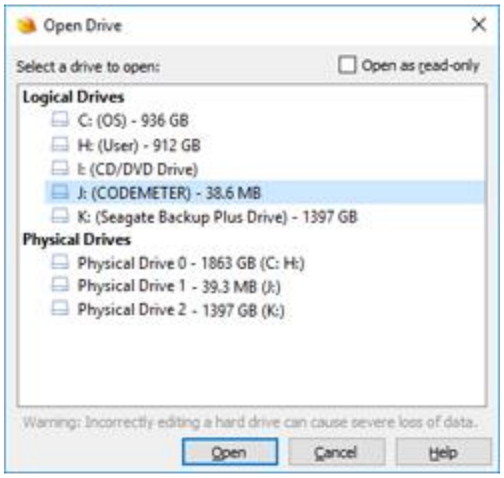
010 Editor can edit the individual bytes of hard drives, floppy drives, memory keys, flash drives, CD-ROMs, etc. To open a drive for editing, click the '*File > Open Drive...*' menu option, or press *Ctrl+D* to display the Open Drive dialog box. Drives can also be opened from the Command Line.

Two main types of drives can be edited in 010 Editor: *Logical Drives* and *Physical Drives*. A Logical Drive is a device or a partition that has been assigned a drive letter such as 'C:' or 'A:'. A Physical Drive corresponds to an actual device inside the computer such as a hard drive or memory key. Physical Drives may be divided into multiple Logical Drives using partitions.

When editing drives on Windows Vista and all later versions of Windows (7, 8, etc.), Windows requires that all file handles on the drive be closed before changes can be written to the drive. If writing to a Physical Drive, all file handles on the Logical Drives within the Physical Drive must first be closed (a list of Logical Drives within each Physical Drive is displayed in the Open Drive dialog below). When a drive is first opened, 010 Editor displays a warning if any file handles on the drive are open and the drive is marked as read only. To enable editing of the drive close all open files on the drive, right-click on the Editor Window and click '*Read Only*' from the pop-up menu. Note that the Windows boot drive (e.g. C:) cannot be modified unless the OS is booted from a different drive.

WARNING: Incorrectly editing a hard drive can cause severe loss of data. SweetScape Software will not be held liable for data loss as the result of incorrect editing. Edit your drives at your own risk.

Open Drive Dialog


Open Drive
Select a drive to open:
Open as read-only
Logical Drives
C: (OS) - 936 GB
H: (User) - 912 GB
I: (CD/DVD Drive)
J: (CODEMETER) - 38.6 MB
K: (Seagate Backup Plus Drive) - 1397 GB
Physical Drives
Physical Drive 0 - 1863 GB (C: H:)
Physical Drive 1 - 39.3 MB (J:)
Physical Drive 2 - 1397 GB (K:)
Warning: Incorrectly editing a hard drive can cause severe loss of data.
Open
Cancel
Help

In the Open Drive dialog box a list of all Logical Drives is displayed at the top of the dialog and a list of Physical Drives is displayed at the bottom of the dialog. If the drive size can be calculated, the size will be displayed to the right of each drive name. To the right of each Physical Drive name is a list of Logical Drives that are contained within that drive. Select a logical or physical drive and click the *Open* button to open the drive as a file in the editor. Double-clicking on a drive name in the list also opens the drive. Click the *Cancel* button to dismiss the dialog without opening a drive. If the *Open as read-only* toggle is enabled in the upper-right corner of the dialog, the drive will be opened as a read-only file.

Editing Drives

Once a drive is opened in 010 Editor, it can be edited can like any other file (see Using the Hex Editor). Bytes can be modified and blocks of data can be copied or pasted using the clipboard. The only limitation is because the size of drives cannot be modified, bytes cannot be inserted or deleted from a drive. Once modifications have been made to the drive, click the '*File > Save*' menu option to commit the changes to disk. On Windows Vista and all later Windows Versions (7, 8, etc.) all files on the disk must be closed before changes can be written to the drive. See the introduction to this help topic for more information.

Making Disk Images

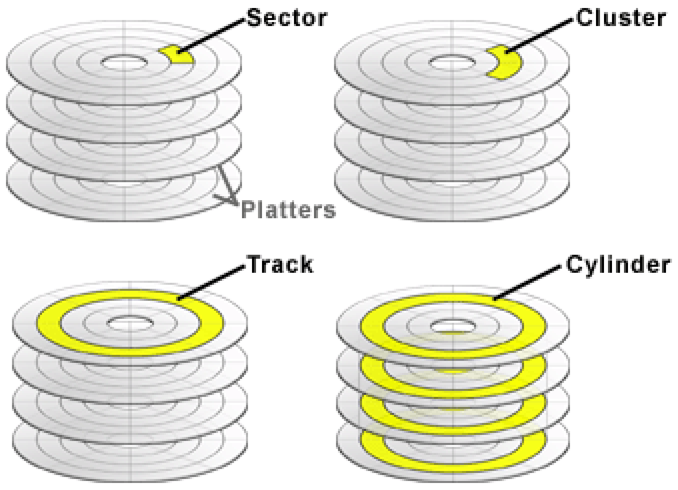
Once a drive has been loaded in 010 Editor, use the '*File > Save As...*' or '*File > Save a Copy...*' menu option to save a byte-by-byte copy of the drive to a file (called a *disk image*). A portion of the drive can be saved to disk by selecting the desired bytes and using the '*File > Save Selection...*' menu option.

Viewing Drive Properties

To view the properties of the current logical or physical drive, click the '*Edit > Properties...*' menu option. See the File Properties help topic for more information. The properties includes information about sectors, clusters, tracks, and cylinders of the drive.

Sectors, Clusters, Tracks, and Cylinders

A typical hard drive is an electromagnetic device made up of a number of disk-shaped pieces called *platters* that are stacked on top of each other (see the figure below). Each platter can store data on both sides and has a read/write *head* that transfers data from the computer to the disk. To find information on these platters, drives are divided into a number of sections called *Sectors*, *Clusters*, *Tracks*, and *Cylinders*.



The following lists each type of section:

- **Sector** - A *sector* is the smallest unit of data that can be read or written from a disk. Typically, sectors are 512 bytes in size, but other sizes including 1024 and 2048 are common.
- **Cluster** - A *cluster* is the smallest unit of data that a file system can allocate for a file. Each cluster has

a fixed size that is always a multiple of the sector size. Older file systems (FAT16) often allocated large cluster sizes of 32K or more, meaning that even small files of 1K would take up 32K of disk space. More modern file systems (FAT32 and NTFS) allow smaller cluster sizes. A file is stored optimally on disk as a series of contiguous clusters (clusters that are in order on disk). However, a file can be split into multiple clusters on different areas of the disk and this is called **fragmentation**.

- **Track** - A *track* is a concentric ring of sectors on a platter. A read/write head can read all the data from a certain track by moving to a position and then rotating the platter.
- **Cylinder** - A *cylinder* is a group of tracks in all the platters that are on top of each other.

By default, 010 Editor displays Sector Lines in the Hex Editor Window that indicate where sectors start and stop on the drive. For more information, see the View Menu help topic. To jump to the previous or next sector, use the *Alt+Up* or *Alt+Down* keys respectively.

Related Topics:
Command Line Parameters
File Properties
Using the Hex Editor
View Menu

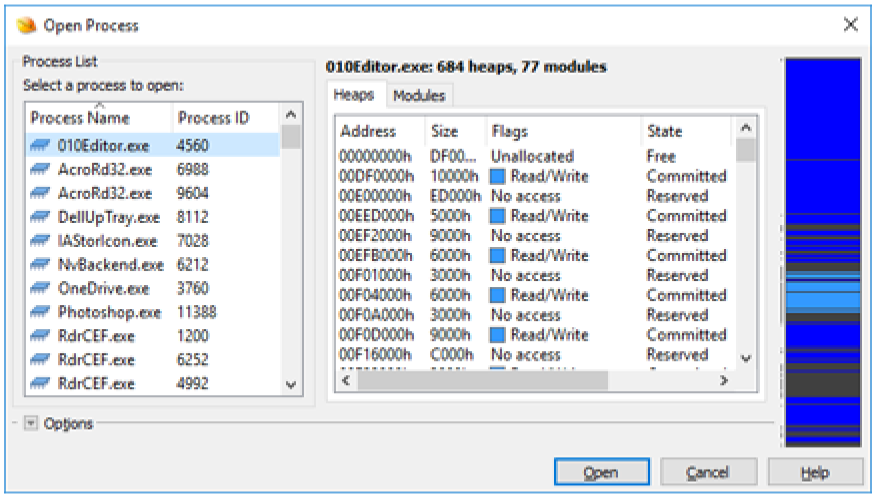
010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Editing Processes

010 Editor can open any currently running process as a file in the interface. The individual bytes of memory used by process can then be edited and saved back to the process. To open a process, use the Open Process dialog box which can be accessed by clicking the '*File > Open Process...*' menu option, or pressing *Ctrl+Shift+O*. Processes can also be opened using the Command Line.

Note: Incorrectly editing processes can lead to programs performing incorrectly or system crashes.

Open Process Dialog



Process Name	Process ID
010Editor.exe	4560
AcroRd32.exe	6988
AcroRd32.exe	9604
DellUpTray.exe	8112
IAStorIcon.exe	7028
NvBackend.exe	6212
OneDrive.exe	3760
Photoshop.exe	11388
RdrCEF.exe	1200
RdrCEF.exe	6252
RdrCEF.exe	4992

010Editor.exe: 684 heaps, 77 modules

Address	Size	Flags	State
00000000h	DF00...	Unallocated	Free
00DF0000h	10000h	🟦 Read/Write	Committed
00E00000h	ED000h	No access	Reserved
00EED000h	5000h	🟦 Read/Write	Committed
00EF2000h	9000h	No access	Reserved
00EFB000h	6000h	🟦 Read/Write	Committed
00F01000h	3000h	No access	Reserved
00F04000h	6000h	🟦 Read/Write	Committed
00F0A000h	3000h	No access	Reserved
00F0D000h	9000h	🟦 Read/Write	Committed
00F16000h	C000h	No access	Reserved

Choose which process to open in the *Process List* on the left side of the dialog. Each process has a name and an ID number associated with it. The process list can be sorted by clicking on the *Process Name* or *Process ID* headings.

When a process is selected, a number of heaps and modules are displayed for the process in the tabbed section of the dialog. A *heap* is simply a block of memory that has been assigned to a process. A list of all heaps for the process can be viewed by clicking on the *Heaps* tab in the center of the dialog. Each heap has a position in memory, indicated by the *Address* and *Size* columns. As well, each heap has a number of *Flags*, a *State* and a *Type*. The *Flags* indicate the access restrictions for the heap and may include Read, Write, WriteCopy, Execute, No access, or Unallocated. The *State* column may be Free (unallocated), Committed (allocated), or Reserved (allocated but not available). The *Type* column may be Image, Mapped, or Private.

A *module* is a block of memory that is associated with an executable, DLL, or other dynamically linked library. Each module may have a number of heaps contained within it. Click the *Modules* tab to view a list of all modules for the current process. Each module has a starting *Address*, a *Size*, and a *Name* which is usually the name of the executable or DLL associated with the module.

On the right-hand side of the display is a graph of all readable bytes of the process. Areas of the process that can be modified are drawn in green, and read-only areas are drawn in gray. If heaps or modules are selected in the heaps or modules list, those areas will be highlighted in the graph. A gray vertical line just to the left of the graph indicates which bytes will be opened in the editor when the *Open* button is clicked. Which bytes are opened is determined by the toggles in the *Options* box.

The *Open Options* box allows control of how the process is opened and is accessed by clicking the *Options* button. If the *Open only Writeable Bytes* toggle is enabled, then only the areas of the process that can be modified are opened (note that a vertical gray line just to the left of the process graph indicates which bytes will be opened). Unclick the toggle to open all readable bytes. Note that if you modify a read-only portion of the file and try to save the changes, you will receive an error. If the *Open Custom Range* toggle is enabled, a custom memory start address and size to open can be specified in the *Start* and *Size* combo boxes. If the tabs in the middle of the dialog are displaying heaps, then the last option will indicate *Open Selected Heaps*. When this option is enabled, only those heaps that are selected in the table will be opened for editing. Likewise, if the Modules tab is displayed, the last option will indicate *Open Selected Modules* and only the selected modules will be opened for editing. Enable the *Open as Read Only* toggle to open the file in read-only mode.

When a process is opened, each heap is mapped to an area of a file. See the Output Window section below for more information.

Editing Processes

Once a process has been opened in the editor, it can be edited just like any other file (see Using the Hex Editor). Since the size of a process is fixed, bytes cannot be inserted or deleted from the process. Once modifications have been made to the process, click the '*File > Save*' menu option to write the changes back to the process. If the process has changed, or you are attempting to modify a read-only area, you may receive an error when saving data.

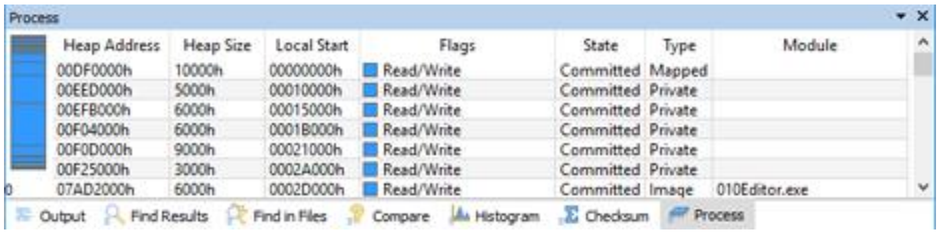
Making Process Images

After a process is loaded in 010 Editor, a byte-by-byte copy of the process can be saved to disk using the '*File > Save As...*' or '*File > Save a Copy...*' menu options (this is called a *process image*). Save a portion of the process to disk by selecting bytes in the editor and clicking the '*File > Save Selection...*' menu option.

Viewing Process Properties

Various properties of the current process can be viewed via the Properties dialog. Click the '*Edit > Properties...*' menu option to view the properties (see File Properties for more information).

Output Window



Heap Address	Heap Size	Local Start	Flags	State	Type	Module
00DF0000h	10000h	00000000h	🟦 Read/Write	Committed	Mapped	
00EED000h	5000h	00010000h	🟦 Read/Write	Committed	Private	
00EFB000h	6000h	00015000h	🟦 Read/Write	Committed	Private	
00F04000h	6000h	0001B000h	🟦 Read/Write	Committed	Private	
00F0D000h	9000h	00021000h	🟦 Read/Write	Committed	Private	
00F25000h	3000h	0002A000h	🟦 Read/Write	Committed	Private	
07AD2000h	6000h	0002D000h	🟦 Read/Write	Committed	Image	010Editor.exe

When a process is opened in the editor, clicking on the *Process* tab of the Output Window displays a list of all heaps that are currently open for the process. If the Output Window is hidden, it can be shown using the '*View > Output*' menu option. Each heap of the process will be mapped to an area of the file. The *Heap Address* and *Heap*

Size columns list the address of the heap in actual memory. The *Local Start* column lists where the heap is mapped to in the file. Right-clicking on any column allows setting the display format via the *Column Display Format* menu option. The *Flags*, *State*, *Type*, and *Module* columns are identical to the columns in the Open Process dialog as discussed above.

A graph of the process is displayed on the left side of the Output Window. This graph is similar to the graph of the Open Process dialog, but only displays those heaps that were opened. The total number of opened heaps is displayed below the graph. Selecting a heap from the list highlights the heap in the graph and also selects the bytes for the heap in the Hex Editor Window.

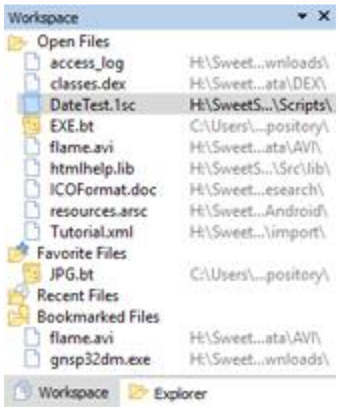
Related Topics:
Command Line Parameters
File Properties
Using the Hex Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using the Workspace

The Workspace is the main control center for managing files in 010 Editor and includes two tabs: *Workspace* and *Explorer*. Use the Workspace tab to manage open, favorite, recent and bookmarked files and use the Explorer tab to browse for files on disk. Show or hide the Workspace tabs using the '*View > Workspace Windows*' menu. The following sections describe the *Workspace* and the *Explorer* tabs.

Workspace



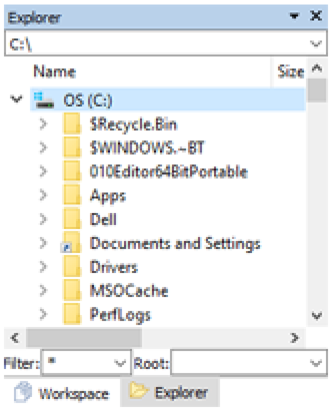
Workspace	
Open Files	
access_log	H:\Sweet...wnloads\
classes.dex	H:\Sweet...ata\DEX\
DateTest.1sc	H:\SweetS...\Scripts\
EXE.bt	C:\Users\...pository\
flame.avi	H:\Sweet...ata\AVI\
htmlhelp.lib	H:\SweetS...\Src\lib\
ICOFormat.doc	H:\Sweet...esearch\
resources.arsc	H:\Sweet...Android\
Tutorial.xml	H:\Sweet...\import\
Favorite Files	
JPG.bt	C:\Users\...pository\
Recent Files	
Bookmarked Files	
flame.avi	H:\Sweet...ata\AVI\
gnsp32dm.exe	H:\Sweet...wnloads\
Workspace	Explorer

The *Workspace* tab is used to open files or manage files in the editor. Four lists of files are displayed with a folder icon besides each heading. Double-click the folder title to expand or hide the list of files (the folder icon will change from opened to closed or vice-versa). The files are displayed with the file name on the left side of the Workspace and the path on the right side in a lighter color. The areas can be resized by moving the mouse cursor between the areas until a resize icon appears. Each list of files is sorted by file name, but only part of the file name will be displayed if the name is too long. To see the full file name, place the mouse cursor over the name in the list. The four lists are as follows:

- **Open Files -** Displays a list of all files currently open in the editor. Double-clicking on a file name will activate that file and bring the corresponding Editor Window to the front. Right clicking the file will display a menu with 5 options for the file: '*Activate*' brings the file to the front, '*Close*' closes the file, '*Save*' saves the file to disk, '*Add to Favorites*' adds the file to the Favorites list (see below), and '*Properties...*' displays the file properties.
- **Favorite Files -** Displays a list of favorite files for easy access. Files can be added to the list by right-clicking on a file in the *Open Files*, *Recent Files*, or *Bookmarked Files* lists and clicking the '*Add to Favorites*' menu option. Remove files from the list by right-clicking on a file in the Favorites list and selecting the '*Remove from Favorites*' menu item. Select '*Open*' from the right-click menu or double-click the file name to open the file.
- **Recent Files -** Displays a list of all files which have been edited recently, but are not currently open. Double-click the file name or click '*Open*' from the right-click menu to open the file in the editor. Right-clicking the file and selecting '*Add to Favorites*' will add the file to the Favorites list (see above). The list of recent files can be sorted either by the order that they were accessed or sorted by file name. To sort the list in access order, right-click the Recent Files and click '*Sort Recent Files > By Time*' or to sort by file name choose '*Sort Recent Files > By Name*'. A file can be deleted from the Recent Files list by right-clicking on the file and selecting the *Remove from Recent Files* menu option, and all files can be removed from the list by right-clicking on a file and selecting *Clear Recent Files List*. The number of items in the *Recent Files* list can be configured in the General Options dialog.
- **Bookmarked Files -** Displays a list of all files that contain bookmarks (see Using Bookmarks for more

information). Bookmarks are saved and loaded automatically when the program exits and restarts. Double-click the file, or select '*Open*' from the right-click menu to open the file in 010 Editor. From the right-click menu, select '*Add to Favorites*' to add the file to the Favorites list (see above), or click '*Clear Bookmarks*' to remove all bookmarks from the file.

Explorer



A simplifier File Explorer is integrated directly into 010 Editor. This explorer provides an easy way to locate files on a hard drive and can be accessed by clicking the *Explorer* tab. The Explorer is divided into 4 areas:

- **Directory Name -** The drop-down list at the top of the window displays the current directory. To jump to another directory, enter the directory name in the field and press the Enter key. Click the down arrow to the right of the drop-down list to access a list of recent directories.
- **Directory Tree -** A list of all drives and directories is displayed in a tree below the Directory Name. Open or close directory items by double-clicking the directory name, or single-clicking the '+' or '-' symbols. Double-clicking on a file opens that file in the editor. Files can also be opened by right-clicking on a file name and selecting '*Open*' from the popup menu. The Name, Size, Type, and Date Modified for each file are displayed in the list by scrolling to the right.
- **Filter -** A Filter field is displayed at the bottom of the dialog. Only the files that match this filter will be displayed in the Directory Tree. Enter a new filter in the field and press the Enter key to refresh the display. A list of previous filters can be accessed by clicking on the down-arrow located to the right of the Filter field.
- **Root -** The root directory for the Explorer can be specified in the *Root* field. When a root directory is entered the Explorer will only show the files and folders in that root directory or its subdirectories. After entering a root directory press the Enter key to refresh the list. To disable the use of a root directory and show all available directories, delete the string in the *Root* field or enter '(none)' and press the Enter key. The down-arrow to the right of the *Root* field is used to access a list of recently used root directories.

NOTE: The Workspace is a Dock Window. The tabs can be moved to other locations by clicking and dragging on the dock header bar and the individual tabs can be dragged to rearrange them or move them to their own floating window. Right-click the window and deselect the '*Allow Docking*' toggle to prevent a tab from docking.

Related Topics:
General Options
Opening Files
Saving Files
Using Bookmarks

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

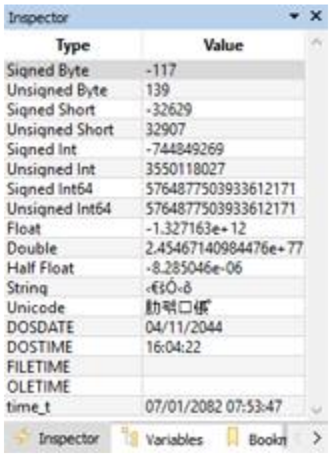
Using the Inspector

The *Inspector* is a powerful tool for examining and editing binary data as a number of different data types. Grouped with the Inspector are six other tabs: *Variables*, *Bookmarks*, *Functions*, *Watch*, *Call Stack* and *Breakpoints*. The *Variables* tab shows a list of created variables after running a Binary Template or a Script. The variables are also displayed in the Template Results panel but the *Variables* tab is an alternate location that can be undocked. The Bookmarks tab displays a list of all created Bookmarks for the current file (see Using Bookmarks) and the *Functions* tab shows a list of all built-in functions that can be used in Scripts or Templates. Show or hide the tabs by using the '*View > Inspector Windows*' menu. The *Watch*, *Call Stack* and *Breakpoints* tabs are discussed in the Using the Debugger help topic.

Some options for the different tabs are accessed by right-clicking with the mouse on the window. Select the '*Copy*' option from the menu to copy the contents of the current cell to the clipboard. Clicking the '*Copy Row*' or '*Copy Column*' option will copy the entire row or column to the clipboard, and the '*Copy Table*' option will copy all data (the data will be copied in a tab-delimited format). For some columns of the Inspector, the numbers in the column can be displayed in a number of different numeric formats by selecting '*Column Display Format*' from the right-click menu and then choosing '*Hex*', '*Decimal*', '*Octal*', or '*Binary*'. Click the '*Export CSV*' menu option to write the current table to a CSV file. A CSV file (which stands for Comma Separated Value) is a text file where each cell is written separated by commas and CSV files are written in UTF-8 format.

When viewing the tabs (*Inspector*, *Variables*, *Bookmarks*, etc.) note that the small left and right arrows beside the tabs may need to be clicked to view all the tabs. The following sections discuss each tab:

Inspector Tab



Type	Value
Signed Byte	-117
Unsigned Byte	139
Signed Short	-32629
Unsigned Short	32907
Signed Int	-744849269
Unsigned Int	3550118027
Signed Int64	5764877503933612171
Unsigned Int64	5764877503933612171
Float	-1.327163e+12
Double	2.45467140984476e+77
Half Float	-8.285046e-06
String	‹€šÕ‹ð
Unicode	劻뫂口㤊
DOSDATE	04/11/2044
DOSTIME	16:04:22
FILETIME	
OLETIME	
time_t	07/01/2082 07:53:47

When the *Inspector* tab is selected a list of data types will be displayed in a table. When a file is opened, the binary data starting at the cursor is converted to each of the different data types and displayed in the table. As the cursor is moved around the file, the Inspector will change to display the converted data. If a selection is made in the current file, the data is converted starting at the beginning of the selection. The following formats are supported in the Inspector:

- **Signed Byte -** 8-bit number between -128 and 127
- **Unsigned Byte -** 8-bit number between 0 and 255
- **Signed Short -** 16-bit number between -32768 and 32767

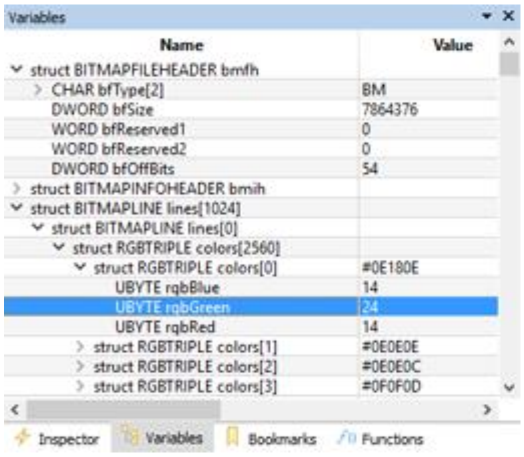
Copyright © 2003-2019 SweetScape Software

- **Unsigned Short -** 16-bit number between 0 and 65535
- **Signed Int -** 32-bit number between -2147483648 and 2147483647
- **Unsigned Int -** 32-bit number between 0 and 4294967295
- **Signed Int64 -** 64-bit number -9223372036854775808 and 9223372036854775807
- **Unsigned Int64 -** 64-bit number between 0 and 18446744073709551615
- **Float -** 32-bit floating-point number between 1.175494351e-38 and 3.402823466e38
- **Double -** 64-bit floating-point number between 2.2250738585072014e-308 and 1.7976931348623158e+308
- **Half Float -** 16-bit floating-point number between 5.960464e-08 and 65504
- **String -** Displays a null-terminated ASCII character string (limit of 256 characters). If a string is edited and characters are inserted or deleted, when in *Insert* mode, bytes will be inserted or deleted from the file but when in *Overwrite* mode, null bytes will be written to the file so that the file size does not change.
- **Unicode -** Displays a null-terminated Unicode character string (limit of 128 characters). Strings are edited in a similar manner to the *String* data type.
- **DOSDATE -** 16-bit value representing the date in DOS using the format 'MM/dd/yyyy' (note that M means month, d means day, and y means year). The date format can be controlled in the Inspector Options dialog.
- **DOSTIME -** 16-bit value representing the time in DOS using the format 'hh:mm:ss' (note that h means hour, m means minute, and s means second). The time format can be controlled in the Inspector Options dialog.
- **FILETIME -** 64-bit value representing date and time in Windows using the format 'MM/dd/yyyy hh:mm:ss'. FILETIME is a 64-bit integer representing the number of 100-nanosecond intervals since 01/01/1601 12:00 AM. The date format can be controlled in the Inspector Options dialog.
- **OLETIME -** 64-bit value representing date and time in OLE and Delphi using the format 'MM/dd/yyyy hh:mm:ss'. OLETIME is a 64-bit double representing the number of days since 12/30/1899 12:00 AM. The date format can be controlled in the Inspector Options dialog.
- **time_t -** 32-bit value representing date and time in C using the format 'MM/dd/yyyy hh:mm:ss'. time_t is a 32-bit integer representing the number of seconds since 01/01/1970 12:00 AM. The date format can be controlled in the Inspector Options dialog.
- **time64_t -** Similar to time_t except data is stored as a 64-bit value. time64_t represents the number of seconds since 01/01/1970 12:00 AM. Use the Inspector Options dialog the change the date format.

To edit a value, left-click the number with the mouse or press the *Enter* key. Edit the value (see Introduction to Number Systems for a list of supported formats) and press *Enter* to commit the change or *Esc* to cancel. Note that changed bytes will be displayed as orange when editing data using the Hex Editor Window. The '*Edit > Undo*' or '*Edit > Redo*' commands can be used to undo and redo changes made with the Inspector.

The different data types in Inspector may be reordered or deleted or your own custom data formats may be added. To customize the Inspector tab, right-click on the table and click the '*Customize...*' menu option or see the Inspector Options dialog.

Variables Tab



Name	Value
struct BITMAPFILEHEADER bmfh	
CHAR bfType[2]	BM
DWORD bfSize	7864376
WORD bfReserved1	0
WORD bfReserved2	0
DWORD bfOffBits	54
struct BITMAPINFOHEADER bmih	
struct BITMAPLINE lines[1024]	
struct BITMAPLINE lines[0]	
struct RGBTRIPLE colors[2560]	
struct RGBTRIPLE colors[0]	#0E180E
UBYTE rgbBlue	14
UBYTE rgbGreen	24
UBYTE rgbRed	14
struct RGBTRIPLE colors[1]	#0E0E0E
struct RGBTRIPLE colors[2]	#0E0E0C
struct RGBTRIPLE colors[3]	#0F0F0D

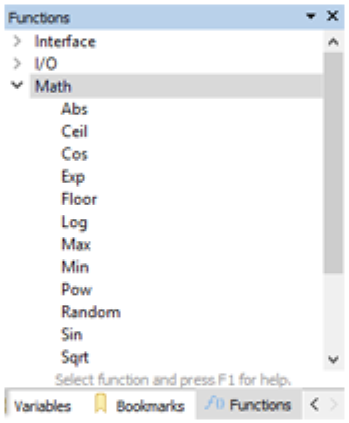
The *Variables* tab displays variables that were generated by running a Script or a Binary Template (see Introduction to Templates and Scripts for more information). Usually the variables generated by a template are edited in the Template Results panel (see Working with Template Results) but this tab provides an alternate place to view and edit variables that can be undocked. The functionality of this tab is the same as the Template Results panel and is discussed in the Working with Template Results help topic.

Bookmarks Tab

The *Bookmarks* tab displays a list of all bookmarks for the current file (see Using Bookmarks). Bookmarks are displayed and edited similar to the Template Results panel mentioned above. The *Name* column displays a combination of the *Name* and *Type* fields in the Add/Edit Bookmark dialog. The *Value* column shows the bytes of the bookmark interpreted according to the data type. The *Start* and *Size* columns display the position of the bookmark, and the *Color* column shows the *Foreground* (*Fg:*) and *Background* (*Bg:*) colors from the dialog.

To edit a bookmark, select a bookmark from the list and select the '*Edit Bookmark*' menu option from the right-click menu. Selecting the '*Remove Bookmark*' menu option from the right-click menu will delete the bookmark.

Functions Tab



Interface
I/O
Math
Abs
Ceil
Cos
Exp
Floor
Log
Max
Min
Pow
Random
Sin
Sqrt
Select function and press F1 for help.

The *Functions* tab lists all built-in functions that can be used when writing 010 Editor Scripts or Binary Templates (see Introduction to Templates and Scripts). All functions are sorted into five categories: Interface Functions, I/O

Functions, String Functions, Math Functions, and Tool Functions. Click the arrow beside each function group to list all functions in that group. The *All* group lists every function in alphabetical order. Double-clicking on a function name will insert that function into the text editor at the cursor position. Click a function and press the F1 key to view help on that function.

The other tabs in the Inspector tab group are discussed in the Using the Debugger help topic.

NOTE: The Inspector is a Dock Window. The tabs can be moved to other locations as a group by clicking and dragging on the dock header bar and the individual tabs can be dragged to rearrange them or move them to their own floating window. Right-click the window and deselect the '*Allow Docking*' toggle to prevent a tab from docking.

Related Topics:
Inspector Options
Interface Functions
Introduction to Number Systems
Introduction to Templates and Scripts
Using Bookmarks
Using the Debugger
Using the Hex Editor
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

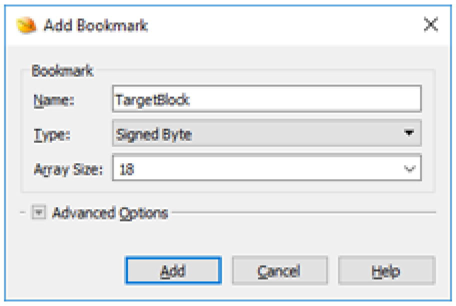
Using Bookmarks

A Bookmark is a set of bytes in a file that are marked as having special significance. There are two types of bookmarks in 010 Editor: Quick Bookmarks and Advanced Bookmarks. A Quick Bookmark just marks a position at a particular byte but an Advanced Bookmark may be given a name and may be interpreted as any of the standard data types or data types defined in a Binary Template. All bookmarks are persistent, meaning that created bookmarks will still exist after exiting and restarting 010 Editor. The bookmarks for the current file will be displayed in the *Bookmarks* tab (see Using the Inspector).

Adding Quick Bookmarks

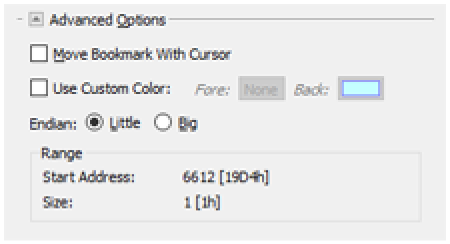
Quick bookmarks provide an easy way to mark an important position or range in a file. To set a quick bookmark, move the cursor to the position to mark or make a selection and then press *Ctrl+F2* or click the '*Search > Toggle Bookmark*' menu option. The marked byte or selection will be displayed in a different color which is controlled by the *Bookmarks* color in the Theme/Color Options dialog. To move the cursor to a bookmark, press the *F2* or *Shift+F2* keys (see the Searching for Bookmarks section below for more information). To remove a bookmark move the cursor to the bookmark you want to delete and press *Ctrl+F2* or click the '*Search > Toggle Bookmark*' menu option again (see the Removing Bookmarks section below for more information).

Adding Advanced Bookmarks



Name: TargetBlock
Type: Signed Byte
Array Size: 18

To add an advanced bookmark, click the '*Search > Add/Edit Bookmark*' menu option, or press *Ctrl+B*. The '*Add Bookmark*' dialog will be displayed, which lists the attributes for the bookmark to create. To keep track of different bookmarks, a name can be assigned using the *Name* field, but this field is optional. Select the data type to interpret the bytes of the bookmark using the *Type* drop-down list. If a Binary Template has been run on the current file, the *Type* drop-down list will also include custom data types defined in the template (note that all bookmarks for a file using custom data types must come from the same Binary Template). If the bookmark to create is not an array, enter '*(none)*' or nothing in the *Array Size* field. To interpret the bookmark bytes as an array, enter the size in the *Array Size* field (note that a list of previously used sizes can be accessed by clicking the down arrow to the right of field).



Start Address: 6612 [19D4h]
Size: 1 [1h]

Other options are available by clicking the *Advanced Options* section. If the *Move Bookmark with Cursor* toggle is enabled, the bookmark will move to the current cursor position every time the Editor Window is clicked with the mouse or the cursor is moved with the cursor keys. This feature is useful to apply structures from a Binary Template to a file when the exact file format is not known.

By default Bookmarks use the *Bookmarks* color in the Theme/Color Options dialog. To use a different color than the default enable the *Use Custom Color* toggle and choose a foreground color (text color) using the *Fore* color rectangle and a background color using the *Back* color rectangle. Clicking a color rectangle shows a drop-down list of colors and a new color for the bookmark can be chosen by clicking one of the colors in the list. Selecting *None* from the list means that the bookmark will not change the color. Click the *More Colors...* button from the drop-down list to select a different color using a standard color selection dialog.

Select which endian should be used when interpreting the data (see Introduction to Byte Ordering) using the *Little* or *Big* toggles. By default, the endian will be the same endian as the file.

The start address and size of bookmark to be created will be listed beside the *Start Address* and *Size* labels respectively. If no bytes were selected in the file when the '*Add Bookmark*' dialog was opened, the *Start Address* will be the cursor position in the file, and the *Size* will be calculated from the *Type* and *Array Size*. If a selection was made when opening the dialog, the *Start Address* will be the start of the selection, and the *Size* will be the size of the selection (note that the *Array Size* will be adjusted automatically to try to fit inside the selection). When defining a bookmark using a custom data type from a Binary Template, sometimes the size cannot be calculated so the size will be displayed as '???'.

Click the *Add* button to create the bookmark or the *Cancel* button to dismiss the dialog. The generated bookmarks will color the current file and be displayed in the *Bookmarks* tab of the Inspector. Note that when bookmarks are added to a file, that file will be displayed in the *Bookmarked Files* list in the Workspace (see Using the Workspace).

Editing Bookmarks

To edit a bookmark, position the cursor over a bookmark in a file, or select the bookmark from the *Bookmarks* tab of the Inspector. Clicking the '*Search > Add/Edit Bookmark*' menu option or pressing *Ctrl+B* will display the above dialog with all the values from the selected bookmark. Change any values and click *Update* to apply the changes or *Cancel* to ignore the changes.

Searching for Bookmarks

To search for the next bookmark after the current cursor position, click the '*Search > Next Bookmark*' menu option or press *F2*. If a bookmark is found it will be selected in the file. The search will wrap to the beginning of the file if no more bookmarks are found. Similarly, to search for the previous bookmark before the current cursor position, click the '*Search > Previous Bookmark*' menu option or press *Shift+F2*.

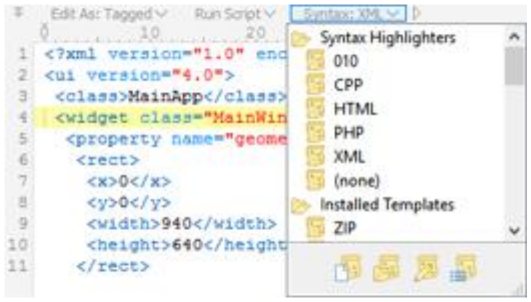
Removing Bookmarks

To remove a bookmark, place the cursor over the bookmark in the file or select the bookmark from the *Bookmarks* tab of the Inspector. Click the '*Search > Toggle Bookmark*' menu option or press *Ctrl+F2* to remove the bookmark from the file. Alternately, all bookmarks can be removed from the file at once by clicking the '*Search > Clear All Bookmarks*' menu option.

Related Topics:
Introduction to Byte Ordering
Theme/Color Options
Using the Inspector
Using the Workspace

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Syntax Highlighting



```
<?xml version="1.0" enc
<ui version="4.0">
 <class>MainApp</class>
 <widget class="MainWin
  <property name="geome
   <rect>
    <x>0</x>
    <y>0</y>
    <width>940</width>
    <height>640</height
   </rect>
```

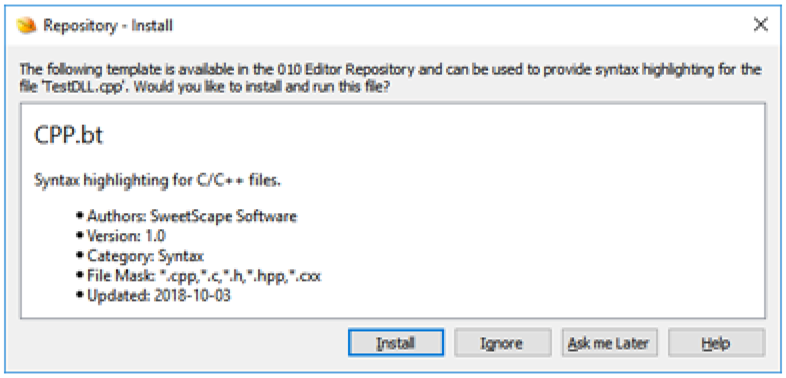
Syntax Highlighting allows applying colors to text in the Text Editor to make certain information easier to see. The easiest way to apply Syntax Highlighting is to click the *Syntax* section of the File Bar above each Text Editor as shown in the above figure. Syntax Highlighters can also be applied using the '*Templates > Syntax*' menu. To remove Syntax Highlighting from a file, click on the *Syntax* section of the File Bar and select '*(none)*'.

Syntax Highlighters can be set to run automatically when files are opened based on their extension or file name. To control which Syntax Highlighters are run on load see the *File Mask* field of the Template Options dialog, accessed on the menu under '*Templates > View Installed Templates*'. To control which Syntax Highlighter is applied when a new file is created see the *Manage New File Types* button in the Editor Options dialog.

The colors applied for each Syntax Highlighter depend upon the current Theme. Different themes can be selected using the Theme/Color Options dialog. The individual colors for Syntax Highlighters can also be customized by scrolling down to the *Syntax Styles* section of the Theme/Color Options dialog.

Installing Syntax Highlighters

Syntax Highlighters now use 010 Editor's Binary Templates technology. Each Syntax Highlighter is contained inside of a separate Binary Template bt file. New Syntax Highlighters can be installed from the Template Repository by clicking '*Templates > Template Repository*' and locating any Templates in the *Syntax* category. If a file is opened in 010 Editor and a Syntax Highlighter exists in the Repository which is not already installed, a dialog will be displayed asking to install the Template:


Repository - Install

The following template is available in the 010 Editor Repository and can be used to provide syntax highlighting for the file 'TestDLL.cpp'. Would you like to install and run this file?

CPP.bt

Syntax highlighting for C/C++ files.

- Authors: SweetScape Software
- Version: 1.0
- Category: Syntax
- File Mask: *.cpp,*.c,*.h,*.hpp,*.cxx
- Updated: 2018-10-03

The Syntax Highlighter can be installed by clicking the *Install* button or ignored by clicking the *Ignore* button. See

Installing Files on Open from the Repository for more information on installing templates on open.

The list of installed Syntax Highlighters is available in the Template Options dialog. All Templates that have the category *Syntax* are assumed to be Syntax Highlighters and these Templates will appear in the *Syntax* drop-down section of the File Bar. A Binary Template which contains a Syntax Highlighter can be installed in the Template Options dialog by clicking the *Add...* button and then selecting the Template.

Writing Syntax Highlighters

Starting in version 9 of 010 Editor, Syntax Highlighters are now written as a function inside a Binary Template. This means that some programming is required to create Syntax Highlighters but that they can be very powerful and handle a huge variety of different formats. A number of functions have been provided to make the process of writing Syntax Highlighters easier. New Syntax Highlighters can be created the same way new Binary Templates are created, for example by using the '*Templates > New Template*' menu option. Syntax Highlighters should also have their Category set to Syntax in order to be displayed in the *Syntax* drop-down menu of the File Bar when they are installed.

A Binary Template which is used to perform syntax highlighting must implement the special function HighlightLineRealtime. This function is called every time a line of text is about to be displayed in the editor to apply coloring to the line. Note that this function only works on a single line but to handle multi-line syntax highlighting such as multi-line comments, see the special *flags* parameter of the HighlightLineRealtime function which can be used to pass the status to the next line.

Most Syntax Highlighters make use of Syntax Styles which allow a single style to be used in multiple Syntax Highlighters. For example the Syntax Style "code-comment" is used to specify the color of comments both in C/C++ and PHP. The colors can then be changed using the Theme/Color Options dialog. See the HighlightFindStyle function to connect to styles and the HighlightApplyStyle to apply colors to text.

A set of rule functions is provided for convenience when writing Syntax Highlighters but note that these functions are not required to be used. These functions assume that the current rule (highlighting method) is stored in the flags parameter of the HighlightLineRealtime function. Highlighting rules are provided to color single-line comments, keywords, multi-line comments, strings, tags, etc. See the functions HighlightCheckCommentRule, HighlightCheckKeywordRule, HighlightCheckMultiLineRule, HighlightCheckSingleLineRule, or HighlightCheckTagRule for more information.

In order to save time and memory, Syntax Highlighters allow *instance sharing*. This means that all open files in the editor can share a single copy of a Binary Template to do syntax highlighting for a particular type. See the HighlightAllowInstanceSharing function for more information.

Syntax Highlighting is only used for text files; however, realtime highlighting can be applied to hex files as well by implementing the HighlightBytesRealtime function. More information is found in the help topic for this function.

Old Syntax Highlighters

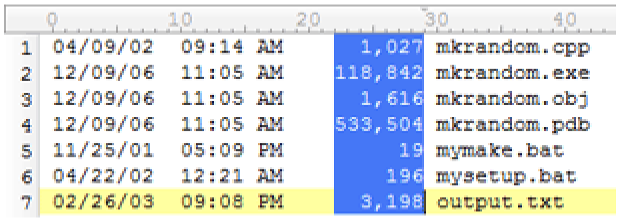
In 010 Editor version 8 and previous versions, syntax highlighting was controlled using the '*Tools > Options...*' dialog. These type of syntax highlighters are no longer supported. When a newer version of 010 Editor is first run and there exist old custom syntax highlighters, those syntax highlighters will automatically be exported as an XML file to the '*Documents\SweetScape\Old Syntax Highlighters*' directory. The old syntax highlighters can also be exported by clicking on the *Export Old Syntax Highlighter* button in the Highlight Options dialog. If this button is not visible in the dialog then no syntax highlighters could be found to export.

Currently there does not exist any automatic way to convert from the XML file into a Binary Template which can be used for Syntax Highlighting and the conversion must be done by hand. SweetScape Software may be available to help in the conversion of common text formats time permitting and contact SweetScape Software for more information.

Related Topics:
Editor Options
Highlight Options
Installing Files on Open from the Repository
Introduction to Templates and Scripts
Introduction to the Repository
Template Options
Theme/Color Options
View Menu
Using the Text Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Column Mode



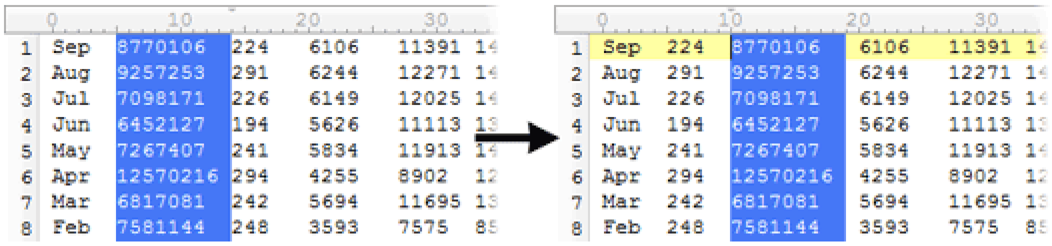
```
04/09/02  09:14 AM      1,027 mkrandom.cpp
12/09/06  11:05 AM    118,842 mkrandom.exe
12/09/06  11:05 AM      1,616 mkrandom.obj
12/09/06  11:05 AM    533,504 mkrandom.pdb
11/25/01  05:09 PM         19 mymake.bat
04/22/02  12:21 AM        196 mysetup.bat
02/26/03  09:08 PM      3,198 output.txt
```

Column Mode is a special editing mode when using the Text Editor or Hex Editor where columns of data are selected instead of a contiguous set of bytes (see the image above for an example). To enter Column Mode click the '*View > Linefeeds > Column Mode*' menu option, type *Alt+3*, or click the Column Mode icon in the Toolbar. Column selections can also be made without entering Column Mode by holding down the *Ctrl* key while dragging the mouse.

Columns Selections and the Clipboard

Once a column selection is made in the Text Editor, use the regular clipboard operations to manipulate the data. For example, *Ctrl+C* will copy the column selection to the clipboard and the Delete key will delete the column selection. When a column selection has been copied to the clipboard and Paste is used, the data is pasted as columns even if the editor is not in Column Mode (this is useful so that *Ctrl* plus dragging can be used to make a column selection and the selection can be copied and pasted all without entering Column Mode). Pasting a column selection has a few different possibilities:

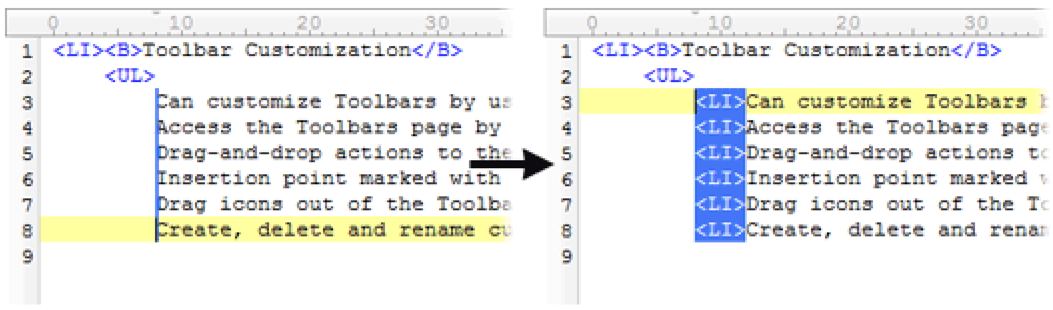
- If a column selection is on the clipboard and Paste is used, the columns will be inserted into the file and spaces will be inserted into the file so that the columns line up properly. The following image shows an example of selecting a set of columns, cutting them to the clipboard and pasting them to the right 5 columns:



```
Sep  8770106  224   6106   11391  14
Aug  9257253  291   6244   12271  14
Jul  7098171  226   6149   12025  14
Jun  6452127  194   5626   11113  13
May  7267407  241   5834   11913  14
Apr  12570216 294   4255   8902   12
Mar  6817081  242   5694   11695  13
Feb  7581144  248   3593   7575   85
```

```
Sep  224  8770106   6106   11391  1
Aug  291  9257253   6244   12271  1
Jul  226  7098171   6149   12025  1
Jun  194  6452127   5626   11113  1
May  241  7267407   5834   11913  1
Apr  294  12570216  4255   8902   1
Mar  242  6817081   5694   11695  1
Feb  248  7581144   3593   7575   8
```

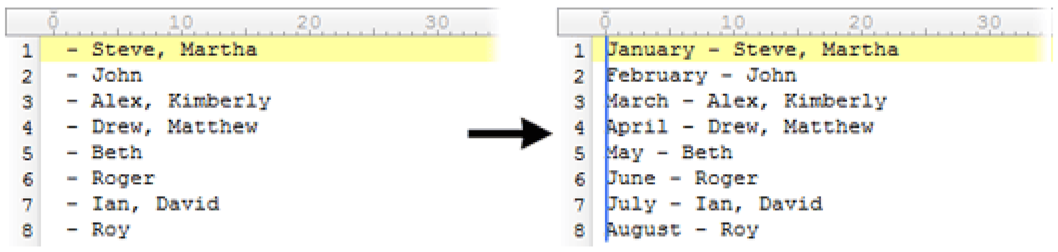
- If only a single line of text data containing no linefeeds is on the clipboard and either a column selection has been made or a Column Insert Line has been drawn (see *Typing with Column Selections* below), pasting will cause the clipboard contents to be pasted on each line. For example, the following image shows the result of pasting the text "<LI>" when a Column Insert Line has been made:



```
<LI><B>Toolbar Customization</B>
    <UL>
        Can customize Toolbars by us
        Access the Toolbars page by
        Drag-and-drop actions to the
        Insertion point marked with
        Drag icons out of the Toolba
        Create, delete and rename cu
```

```
<LI><B>Toolbar Customization</B>
    <UL>
        <LI>Can customize Toolbars b
        <LI>Access the Toolbars page
        <LI>Drag-and-drop actions tc
        <LI>Insertion point marked w
        <LI>Drag icons out of the Tc
        <LI>Create, delete and renan
```

- If multiple lines of text data (not a column selection) are on the clipboard and either the editor is in Column Mode or a column selection is currently made, the text data will be inserted line-by-line into the editor (note in this case the columns may not line up properly). The following figure is an example of pasting a text file containing a list of months into a file in Column Mode:



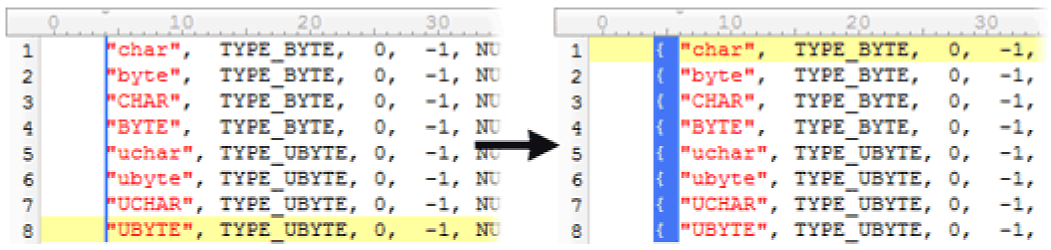
```
 - Steve, Martha
 - John
 - Alex, Kimberly
 - Drew, Matthew
 - Beth
 - Roger
 - Ian, David
 - Roy
```

```
January - Steve, Martha
February - John
March - Alex, Kimberly
April - Drew, Matthew
May - Beth
June - Roger
July - Ian, David
August - Roy
```

Note that when pasting and a column selection has been made, the column selection will be deleted before the paste occurs.

Typing with Column Selections

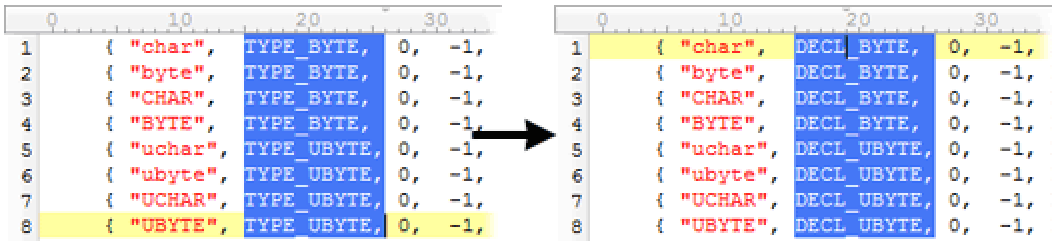
010 Editor has a special Column Insert Line which can be created by clicking and dragging straight down when in Column Mode or when the *Ctrl* key is held down. The Column Insert Line is displayed as a vertical blue line as shown in the figure below left. When a Column Insert Line exists and a key is typed on the keyboard, that key will be inserted onto each line at the same time. The following figure shows an example of typing the characters '{' and ' ' when a Column Selection Line is made:



```
        "char",  TYPE_BYTE,  0,  -1, NU
        "byte",  TYPE_BYTE,  0,  -1, NU
        "CHAR",  TYPE_BYTE,  0,  -1, NU
        "BYTE",  TYPE_BYTE,  0,  -1, NU
        "uchar", TYPE_UBYTE, 0,  -1, NU
        "ubyte", TYPE_UBYTE, 0,  -1, NU
        "UCHAR", TYPE_UBYTE, 0,  -1, NU
        "UBYTE", TYPE_UBYTE, 0,  -1, NU
```

```
        { "char",  TYPE_BYTE,  0,  -1,
        { "byte",  TYPE_BYTE,  0,  -1,
        { "CHAR",  TYPE_BYTE,  0,  -1,
        { "BYTE",  TYPE_BYTE,  0,  -1,
        { "uchar", TYPE_UBYTE, 0,  -1,
        { "ubyte", TYPE_UBYTE, 0,  -1,
        { "UCHAR", TYPE_UBYTE, 0,  -1,
        { "UBYTE", TYPE_UBYTE, 0,  -1,
```

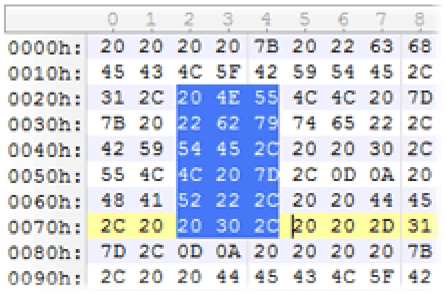
This technique also works when regular column selections are made and keys are typed on the keyboard, except in this case the typed keys are written over the column selection until the cursor reaches the right side of the column selection. Typing additional keys when the cursor is at the right side of the column selection will insert that key on each line. For example, the following image shows making a column selection and then typing the characters 'DECL' on the keyboard (note the position of the cursor on the right-hand image):



```
    { "char",  TYPE_BYTE,  0,  -1,
    { "byte",  TYPE_BYTE,  0,  -1,
    { "CHAR",  TYPE_BYTE,  0,  -1,
    { "BYTE",  TYPE_BYTE,  0,  -1,
    { "uchar", TYPE_UBYTE, 0,  -1,
    { "ubyte", TYPE_UBYTE, 0,  -1,
    { "UCHAR", TYPE_UBYTE, 0,  -1,
    { "UBYTE", TYPE_UBYTE, 0,  -1,
```

```
    { "char",  DECL_BYTE,  0,  -1,
    { "byte",  DECL_BYTE,  0,  -1,
    { "CHAR",  DECL_BYTE,  0,  -1,
    { "BYTE",  DECL_BYTE,  0,  -1,
    { "uchar", DECL_UBYTE, 0,  -1,
    { "ubyte", DECL_UBYTE, 0,  -1,
    { "UCHAR", DECL_UBYTE, 0,  -1,
    { "UBYTE", DECL_UBYTE, 0,  -1,
```

Hex Editor and Column Mode



```
        0  1  2  3  4  5  6  7  8
0000h: 20 20 20 20 7B 20 22 63 68
0010h: 45 43 4C 5F 42 59 54 45 2C
0020h: 31 2C 20 4E 55 4C 4C 20 7D
0030h: 7B 20 22 62 79 74 65 22 2C
0040h: 42 59 54 45 2C 20 20 30 2C
0050h: 55 4C 4C 20 7D 2C 0D 0A 20
0060h: 48 41 52 22 2C 20 20 44 45
0070h: 2C 20 20 30 2C 20 20 2D 31
0080h: 7D 2C 0D 0A 20 20 20 20 7B
0090h: 2C 20 20 44 45 43 4C 5F 42
```

Column mode also works with the Hex Editor except all selections are made by byte as shown above. Note that deleting a column selection in the Hex Editor can cause columns to be unsynchronized. Pasting works similar to the Text Editor Column Mode except that if a column selection exists while pasting, the selection will not be deleted before the paste occurs. Typing on a Column Selection works the same way as indicated above and typing 0's is a quick way to blank out a column selection.

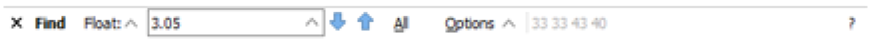
Related Topics:
Using the Clipboard
Using the Hex Editor
Using the Text Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Find

The Find Bar can be used to find a string, a set of hex bytes, or a number of different data types within a file. The Find Bar can be accessed from the '*Search > Find...*' menu option, the Tool Bar, or by pressing *Ctrl+F*.

Find Bar



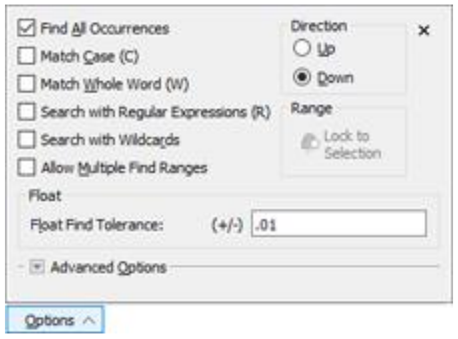
When performing a search, the Find Bar will be displayed at the bottom of the editor. Different data types may be searched by clicking the type name to the right of the *Find* label and choosing from the popup list. The following types are supported:

- **Text** (t) - searches for a string using the character set of the current file (see File Interfaces). Can be used to search for ASCII strings, Unicode strings, UTF-8 strings, etc.
- **Hex Bytes** (h) - searches for a set of hex bytes (for example: 'FF 00 CB').
- **ASCII String** (a) - searches for an ASCII+ANSI string.
- **Unicode String** (u)
- **EBCDIC String** (e)
- **Signed Byte** (i8)
- **Unsigned Byte** (ui8)
- **Signed Short** (i16)
- **Unsigned Short** (ui16)
- **Signed Int** (i32)
- **Unsigned Int** (ui32)
- **Signed Quad** (i64)
- **Unsigned Quad** (ui64)
- **Float** (f)
- **Double** (lf)
- **Variable Name** (n) - see Finding Variables below
- **Variable Value** (v) - see Finding Variables below

The value in brackets is the *type specifier* and can be used as a quick way to search for different data types (see the Type Specifiers section below). Enter a string, value, or hex bytes to search for in the text field and the field will automatically be converted to hexadecimal bytes and displayed to the right of the *Options* button. Note that when converting numbers or Unicode strings, the endian of the current file is used (see Introduction to Byte Ordering). Note also that the text field can be resized by dragging the line to the right of the *Options* button.

Clicking the ⬇ icon will find the next occurrence of the value in the current file or clicking the ⬆ icon will find the previous occurrence. Clicking the *All* button will find all instances of the target value in the file and display the results in the Output Window (see below). Pressing the Enter key in the text field has three different possible outcomes depending upon which options are set in the *Options* dialog: If the *Find All Occurrences* toggle is set, pressing Enter will perform a Find All operation but if it is not set, pressing Enter will either find the next occurrence of the find value if the *Down* toggle is selected, or find the previous occurrence if the *Up* toggle is selected. If no occurrences of the find value could be found, the find text field will be displayed in an orange color. Press the Esc key to hide the Find Bar and return to editing the file.

Find Options



- [✓] Find All Occurrences
- [ ] Match Case (C)
- [ ] Match Whole Word (W)
- [ ] Search with Regular Expressions (R)
- [ ] Search with Wildcards
- [ ] Allow Multiple Find Ranges

Direction: Up [✓ Down]

Range: Lock to Selection

Float

Float Find Tolerance: (+/-) .01

Advanced Options

Options

Different options for the dialog can be displayed by clicking the *Options* button. When the *Find All Occurrences* toggle is set, pressing the Enter key in the text field will find all occurrences of the target value. When this toggle is not set, pressing the Enter key will either search for the next or previous occurrence of the target value depending upon if the *Down* or *Up* toggle is set in the *Direction* box.

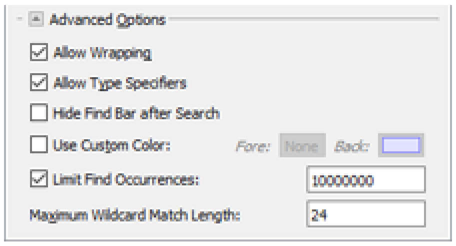
By default all find operations search the entire file but it is possible to limit finding to one part of the file by using the *Range* box. To limit the find, first select some bytes in the file and then click the *Options* button and then the *Lock to Selection* button. The *Lock to Selection* button will be disabled if no selection exists. After this button is clicked, all subsequent find operations will be limited to the selection and the limited area will be highlighted brown in the editor (see Theme/Color Options to change the color). To return to searching the whole file, click the *Unlock Selection* button or the icon.

When searching for strings, two options are available: If the *Match Case* toggle is enabled, the target string will only match if the bytes match exactly. When the toggle is disabled, characters that are not the same case will match. If the *Match Whole Word* toggle is set, the target string will not match partial words. When *Match Case* is enabled, the *Options* button will contain the 'C' character in brackets and when the *Match Whole Word* toggle is enabled, the *Options* button will contain the 'W' character in brackets.

To search for regular expressions enable the *Search with Regular Expressions* toggle and see the separate Regular Expressions help topic. When regular expression searching is turned on the *Options* button will contain the 'R' character in brackets. When searching for strings or hex bytes, enable the *Search with Wildcards* toggle to allow the characters '*' and '?' to be used as wildcards in the text field. The '?' wildcard must match exactly one byte, and the '*' wildcard can match zero up to a certain maximum number of bytes. For example, the value 'adv*e?' would match both the strings 'advantages' and 'advised'. The maximum number of bytes for a match can be specified using the Advanced Options section as described below. When the *Search with Wildcards* toggle is turned on, the special syntax '*' and '\?' can be used to literally search for the characters '*' (ASCII code 0x2A) or '?' (ASCII code 0x3F).

When searching for floats or doubles, the *Float Find Tolerance* field will be displayed. This field can be used to search for numbers that are very close to other numbers. Because of numerical precision, sometimes floating-point numbers are stored as 2.00000001 instead of 2. Enter a tolerance value in the *Float Find Tolerance* field. Numbers that are within the tolerance above or below the target value will match.

If the *Allow Multiple Find Ranges* toggle is enabled, 010 Editor can store the search results from a number of different queries and can optionally color each query by a different color in the Editor Window. When this toggle is enabled, each query that is performed will add another section of results to the Find tab of the Output Window (see Output Window below for more information). Also, the coloring of the file can be controlled with the Advanced Options section as discussed below.


Advanced Options

- [✓] Allow Wrapping
- [✓] Allow Type Specifiers
- [ ] Hide Find Bar after Search
- [ ] Use Custom Color: Fore: None Back:
- [✓] Limit Find Occurrences: 10000000

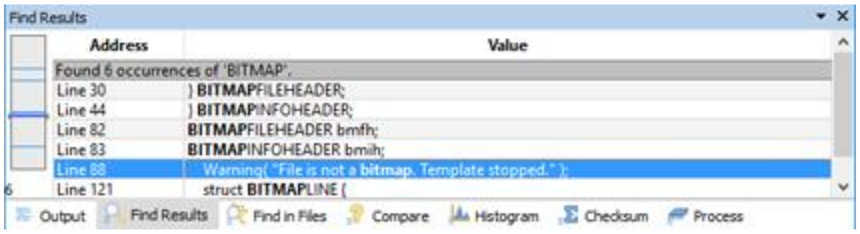
Maximum Wildcard Match Length: 24

Click the *Advanced Options* section to display further options for the find tool. By default, when searching for the next occurrence of a find value and no values are found, the search will start over at the beginning of the file or when searching up and no values are found, the search will start again at the bottom of the file. This is called *wrapping* and can be disabled by turning off the *Allow Wrapping* toggle. Clicking the *Allow Type Specifiers* toggle allows turning off the use of type specifiers (see the Type Specifiers section below). If the *Hide Find Bar after Search* toggle is enabled the Find Bar will be hidden after a Find All operation, but if this toggle is turned off the Find Bar will remained focused after a search.

By default, after a find is executed the found values are colored in the main Editor Window according to the *Find* color listed in the Theme/Color Options dialog. To override this coloring enable the *Use Custom Color* toggle and enter a new color using the *Fore* and *Back* color boxes. The *Fore* color indicates the text color to be displayed, and the *Back* color indicates the background color behind the text. Specifying custom colors is useful with using *Allow Multiple Find Ranges* to color different ranges different colors.

When finding a large number of find values it is possible to run out of memory. To prevent this 010 Editor by default has a limit of 10000000 find occurrences (10000000 occurrences uses approximately 1 GB of storage). This limit can be disabled by turning off the *Limit Find Occurrences* toggle or the limit can be changed using the field to the right of the toggle. If more find occurrences are found than the limit, the text "Too many to display" will be displayed at the bottom of the Find results. When the *Search with Wildcards* toggle is enabled, specify the maximum number of bytes that the wildcard character '*' can match in the *Maximum Wildcard Match Length*' (default of 24).

Output Window



Address	Value
Found 6 occurrences of 'BITMAP'.	
Line 30	} **BITMAP**FILEHEADER;
Line 44	} **BITMAP**INFOHEADER;
Line 82	**BITMAP**FILEHEADER bmfh;
Line 83	**BITMAP**INFOHEADER bmih;
Line 88	Warning("File is not a **bitmap**. Template stopped.");
Line 121	struct **BITMAP**LINE {

6

When searching for all occurrences of a target value using the *Find All* button, the results are displayed in the Output Window. The top row will be a dark gray header line that indicates the search target and number of occurrences found. Each row of the table indicates a match with the target value. The starting address and value searched for are displayed in the *Address* and *Value* fields respectively. When searching within text files, the full line where the find occurrence was found is displayed in the *Value* field with the found value marked in bold. On the left side of the Output Window is a graph representing the file. Blue lines indicate the position of matches in the file. When a row is selected in the table, the corresponding line in the graph will be displayed as a yellow line. The number displayed below the graph indicates the number of matches that were found. Bytes in the main Editor Window will be colored blue if those bytes are part of a match. Selecting a row in the table will also select the bytes in the editor.

The display format for each column can be set to hexadecimal or decimal by right-clicking on the Output Window and selecting '*Column Display Format*'. The *Size* column, which displays the size of each find occurrence is hidden by default but can displayed by right-clicking on the table and selecting *Show Size Column*. Data can be exported or imported from the table in CSV format by right-clicking and selecting either *Export CSV* or *Import* from the menu. Right-click the table and select '*Clear*' to clear all find results, or press the *Esc* key to hide the Output

Window.

If multiple searches were performed and the *Allow Multiple Find Ranges* toggle is enabled, each search will be displayed in the Output Window in a different section of the table separated by a dark gray header line. Click the '+' or '-' buttons beside the header, or double-click the header to hide or show the results from that search. Right-click the table and click the '*Expand All*' or '*Shrink All*' menu options to show or hide the search results for all queries.

Find Next/Find Previous

After a search has been made, the '*Find > Find Next*' or '*Find > Find Previous*' menu options can be used to step through the file, searching for the next or previous occurrences of the target value even when the Find Bar is not displayed.

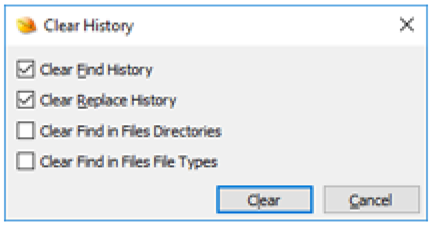
Type Specifiers

A quick way exists in the find field to specify different find types without using the popup list of types. This can be done by placing a comma followed by a type specifier at the end of the value to find in the text field. For example, the value "453f,h" would search for the hex bytes 0x45 and 0x3f, the value "0x100,i32" would search for the integer 256, or the value "2.5,lf" would search for the double '2.5'. The full list of type specifiers is shown in the list at the top of this section. To disable the use of type specifiers, click the *Allow Type Specifiers* toggle in the *Advanced Options* section.

Finding Variables

The Find Bar can also be used to find variable names or values within the Template Results panel. Set the type to search for as *Variable Name* or *Variable Value* to search for variables. When a variable is found that matches either the Name or Value, that variable will be highlighted within the Template Results. Note that find all and replace operations are not supported with finding variables.

Clearing Find History



All recent Find, Replace, and Find In Files operations are stored in a history list which can be accessed by clicking the Up arrow to the right of a text field in the Find, Replace or Find in Files bars. To clear the search history click the Up arrow in the Find field and choose the option '*(clear find history)*' located at the bottom of the list. Select which lists to clear in the dialog above and click the *Clear* button. The *Clear Find in Files Directories* option indicates the history list to the right of the *in Files* text on the Find In Files bar and the *Clear Find in Files File Types* option clears any file masks from the *File Types* field of the Find in Files bar.

NOTE: The Find Bar will be hidden automatically after a period of inactivity. To control the length of the period of inactivity before hiding see the General Options dialog.

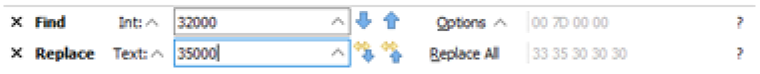
Related Topics:
Introduction to Byte Ordering
Theme/Color Options
Using Find In Files
Using Regular Expressions
Using Replace
Using Replace In Files
Working with File Interfaces
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Replace

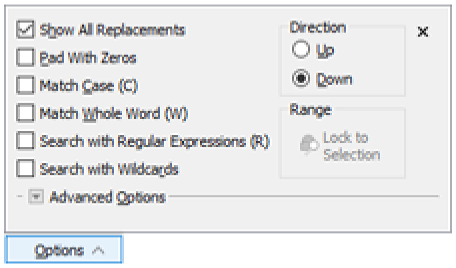
The Replace Bar is used to search and replace a set of bytes within a file. Access the Replace Bar from the '*Search > Replace...*' menu option, the Tool Bar, or by pressing *Ctrl+R*.

Replace Bar



Specify the data to find in the *Find* Bar at the bottom of the editor. The functionality of this bar is documented in the Using Find help topic. Enter the value to replace in the *Replace* text field and choose the data type to replace with the popup list to the right of the *Replace* label (see Using Find for more information). The replace value will be converted to a set of hex bytes and displayed to the right of the *Replace All* button.

Click the Find buttons or to scan through the file without making any replacements. Clicking the Replace buttons or has two possible effects: If no occurrence of the Find value has been found yet, these buttons function just like the Find buttons and search for either the next or the previous occurrence of the Find value and set the selection to the result. If, however, a Find occurrence has been found (as indicated by the selection), clicking either of these buttons will replace the current selection with the Replace value and then search for another occurrence of the Find value. Pressing the Enter key while in the Replace text field will replace the value and find either the next or the previous occurrence depending upon if the *Down* or *Up* toggle is set in the *Options* dialog. Clicking the *Replace All* button automatically replaces all occurrences of the target value in the file.



All options when performing a Replace operation are identical to the Find Bar except for the *Show All Replacements* toggle and the *Pad With Zeros* toggle. When the *Show All Replacements* toggle is set and a *Replace All* operation is done, all the replacements will be shown in the Output Window (see Using Find for more information). When the *Pad With Zeros* toggle is enabled, a set of zero bytes will be appended to the replaced hex bytes until the length is the same as the find hex bytes. This feature is useful to replace a string with a shorter string without changing the file length. The *Range* box is similar in functionality to the *Range* box of the Find Bar. The *Advanced Options* are also documented in the Using Find help topic.

When replacing a large number of values (more than the *Limit Find Occurrences* setting in the *Advanced Options*) and *Show All Replacements* is turned on, the text "Too many to display" will appear at the bottom of the Replace results. Note that all the replacements will be done but not all the replacements will be listed in the Replace results.

Replace Next/Replace Previous

After a replacement has been performed, clicking the '*Search > Replace Next*' menu option or the '*Search > Replace Previous*' menu option will perform the replacement again. These commands, along with the '*Search > Find Next*' and '*Search > Find Previous*' menu options can be used to step through a file making replacements even when the Replace Bar is hidden.

NOTE: The Replace Bar will be hidden automatically after a period of inactivity. To control the length of the period of inactivity before hiding see the General Options dialog.

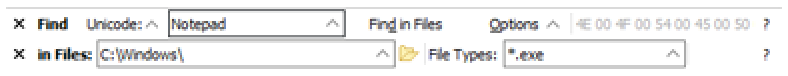
Related Topics:
Using Find

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Find in Files

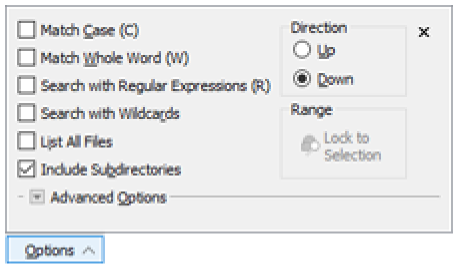
The Find in Files Bar is used to search for a set of bytes across multiple files. Open the Find in Files Bar by using the '*Search > Find in Files...*' menu option, the Tool Bar, or by pressing *Ctrl+Shift+F*.

Find in Files Bar



When searching for a set of bytes across files, the Find in Files Bar is displayed at the bottom of the editor. The top portion of the bar, the *Find* Bar, is used to specify what bytes to search for and is identical to the normal Find Bar (see Using Find for more information).

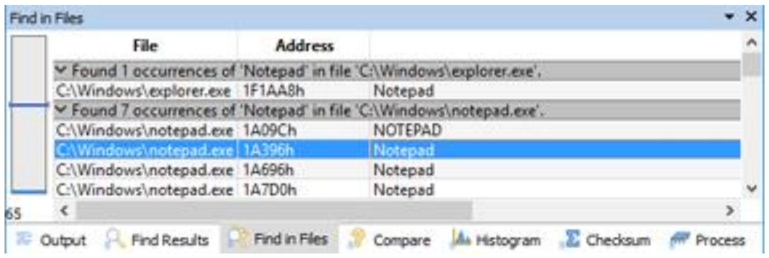
The *in Files* area is used to indicate which files should be searched. To search a directory enter a directory in the text field to the right of the *in Files* label or choose a directory by clicking the browse button to the right of the field. To search only files that are currently open in the interface, click the arrow icon on the right side of the text field and select *All Open Files* from the popup list. Enter a file name mask in the *File Types* field using the characters '*' and '?' to indicate wildcards. Multiple file masks can be entered in the field by separating them by commas or semi-colons (for example: "*.exe,*.dll"). If the *Include Subdirectories* toggle is enabled in the *Options* section, then the subdirectories of the specified directory are recursively searched as well.



Other options for the search can be set by clicking the *Options* button. Consult the Using Find help topic for information on the *Match Case*, *Match Whole Words*, *Search with Regular Expressions* and *Search with Wildcards* toggles. When a Find in Files operation is performed the results are displayed in the Output Window. By default, only those files that contain 1 or more occurrence of the find value are listed; however, all files that were searched can be listed in the Output Window by enabling the *List All Files* toggle.

Click the *Find in Files* button to start the Find in Files search and note that the Esc key can be used to cancel a long search. Note also that the text fields in the *in Files* section can be resized by dragging the line to the right of the Browse button or the line to the right of the File Types field.

Output Window



File	Address	
Found 1 occurrences of 'Notepad' in file 'C:\Windows\explorer.exe'.		
C:\Windows\explorer.exe	1F1AA8h	Notepad
Found 7 occurrences of 'Notepad' in file 'C:\Windows\notepad.exe'.		
C:\Windows\notepad.exe	1A09Ch	NOTEPAD
C:\Windows\notepad.exe	1A396h	Notepad
C:\Windows\notepad.exe	1A696h	Notepad
C:\Windows\notepad.exe	1A7D0h	Notepad

When some occurrences of the search string are located in a file, the results are displayed in the *Find in Files* tab of the Output Window. For each file which contains a match of the search string, a dark gray header line will be displayed in the Output Window indicating the name of the file and the number of occurrences found. All search occurrences for the file will be listed below the dark gray header, one per row of the table. Click the '+' or '-' button beside the header, or double-click on the header to hide or show all search results for that file. Alternately, all headers can be closed by right-clicking on the table and selecting the '*Shrink All*' menu option, or all headers can be opened by right-clicking and selecting the '*Expand All*' menu option. Each search result lists the *File*, *Address*, and *Value* and where the search occurrence was found (note that the *Size* column can be shown by right-clicking on the table and choosing '*Show Size Column*').

Along the left side of the dialog is a graph indicating where the search occurrences were found. The graph will display information for the file the currently selected search occurrence is in. The selected search occurrence will be highlighted as a yellow line and the other occurrences will be displayed as blue lines. The number below the graph indicates the total number of occurrences that were found in all files. The display format for each column can be set to hexadecimal or decimal by right-clicking on the Output Window and selecting '*Column Display Format*'. Press the *Enter* key while a find occurrence is highlighted to load that file and move the cursor to the address of the occurrence. Pressing *Ctrl+Enter* will perform a similar operation as pressing *Enter*, but the application focus will remain on the Output Window. The search results can be exported or imported from the table in CSV format by right-clicking on the results and choosing either *Export CSV* or *Import* from the right-click menu. Right-click the table and select '*Clear*' to clear all find in files results, or press the *Esc* key to hide the Output Window.

NOTE: The Find in Files Bar will be hidden automatically after a period of inactivity. To control the length of the period of inactivity before hiding see the General Options dialog.

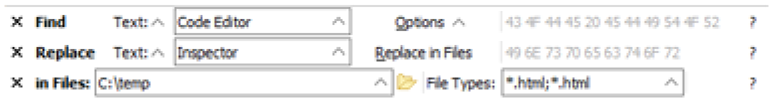
Related Topics:
Using Find

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Replace in Files

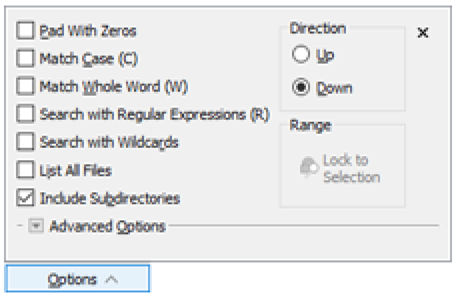
Use the Replace in Files Bar to search and replace a set of bytes across a whole range of files at a single time. The Replace in Files Bar may be accessed by the '*Search > Replace in Files...*' menu option. Note that undo is not currently supported after doing a Replace in Files operation so use this tool with caution.

Replace in Files Bar



Find Text: Code Editor — 43 4F 44 45 20 45 44 49 54 4F 52
Replace Text: Inspector — 49 6E 73 70 65 63 74 6F 72
in Files: C:\temp
File Types: *.html;*.html

Indicate which data to find using the *Find* Bar located at the bottom of the editor (see Using Find for more information). In a similar manner, enter the bytes to replace in the *Replace* Bar (see Using Replace for more information). Choose which files should be searched using the *in Files* Bar. The functionality of this box is the same as with the Find in Files Bar (see Using Find in Files).



- [ ] Pad With Zeros
- [ ] Match Case (C)
- [ ] Match Whole Word (W)
- [ ] Search with Regular Expressions (R)
- [ ] Search with Wildcards
- [ ] List All Files
- [✓] Include Subdirectories

Direction: Up [✓ Down]

Options for the bar can be controlled by clicking the *Options* button. See the Replace Bar for information on the *Pad With Zeros* toggle or the Find Bar for information on the *Match Case*, *Match Whole Word*, *Search with Regular Expressions*, or *Search with Wildcards* toggles. See the Find in Files Bar for information on the *List All Files* or *Include Subdirectories* toggles.

Click the *Replace in Files* button to search all files indicated by the *in Files* Bar and make the indicated replacements.

Output Window

After a Replace in Files operation, the *Find in Files* tab of the Output Window changes to *Replace in Files* and a list of all replacements is displayed in the window. See Using Find in Files for more information on how to use the Output Window.

NOTE: The Replace in Files Bar will be hidden automatically after a period of inactivity. To control the length of the period of inactivity before hiding see the General Options dialog.

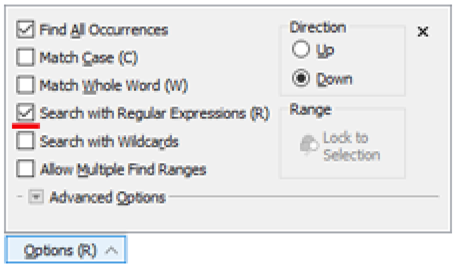
Copyright © 2003-2019 SweetScape Software

Related Topics:
Using Find
Using Find in Files
Using Replace

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Regular Expressions

Regular Expressions are a powerful syntax for finding string patterns within a file. Many different flavors of regular expressions exist and 010 Editor uses a syntax similar to Ruby/Perl. To search for a regular expression, click the *Options* button in the Find Bar and enable the *Search with Regular Expressions* toggle (see the image below). Regular expressions can be used when performing a Find, Replace, Find In Files, or Replace In Files operation. Note that the letter 'R' will appear beside the word *Options* when regular expressions are enabled. The full syntax of regular expressions are beyond the scope of this document but the following contains an introduction to the major features of regular expressions. Warning: Some regular expressions can be very complex and using certain regular expressions containing lots of repetition operators can cause searches to be performed very slowly.



- [✓] Find All Occurrences
- [ ] Match Case (C)
- [ ] Match Whole Word (W)
- [✓] Search with Regular Expressions (R)
- [ ] Search with Wildcards
- [ ] Allow Multiple Find Ranges
- Advanced Options

Direction: Up [✓ Down]

Range: Lock to Selection

Matching Characters

Regular expressions look just like regular Find strings. For example, to find a string such as 'Green' just use the regular expression:

```
Green
```

Regular expressions use a number of special control characters to control how the searches are done and the special characters are: ".[]^$()/*{}?+|". To search for any of these control characters include an extra '\' character before the control character. For example, to search for the string "5+6" use the regular expression:

```
5\+6
```

A number of special codes can be used to match characters:

- **.** - any character (except linefeeds)
- **\w** - a word character include letters, numbers, '_' and unicode characters
- **\W** - a non word character
- **\s** - a whitespace character includes tabs and spaces
- **\S** - a non whitespace chararacter
- **\d** - a decimal digit chararacter [0-9]
- **\D** - a non decimal digit character
- **\h** - a hexadecimal character [0-9a-fA-F]
- **\H** - a non hexadecimal character
- **\t** - horizontal tab (0x09)
- **\n** - newline (0x0A)
- **\r** - return (0x0D)
- **\a** - bell (0x07)

- **\e** - escape (0x1B)
- **\f** - form feed (0x0C)
- **\v** - vertical tab (0x0B)
- **\nnn** - octal character
- **\xHH** - hexadecimal character

For example, to search for all phone numbers in the form 555-5555 use the regular expression:

```
\d\d\d-\d\d\d\d
```

To search for any character (including linefeeds) in a binary file, '[\x00-\xff]' can be used instead of '.'. Note that the case-sensitivity of regular expressions is controlled by the *Match Case* toggle in the Find Bar Options.

Character Classes

A Character Class or Character Set provides a way to give a number of different options that a single character can match. Character Classes are denoted with '[' and ']' brackets where each character inside the brackets can match. For example, the regular expression:

```
defen[cs]e
```

will match both the words 'defence' and 'defense'. Inside of a character class, only the characters "]\-^" are considered control characters. The '-' character can be used to indicate a range of characters. For example the character class:

```
[0-9a-fA-F]
```

will match any of the hexadecimal characters. Using the '^' character at the beginning of a character class indicates a *negated* character class, meaning the regular expression will match any characters that are not in the character class. For example, the character class:

```
[^abc]
```

will match any characters that are not a, b, or c.

Anchors

All matching so far has worked by matching a particular character. Regular expressions also support *anchors* which work by matching a position within a file. The following anchors are supported:

- **^** - beginning of the line
- **$** - end of the line
- **\b** - word boundary
- **\B** - non word boundary

For example, the regular expression:

```
^\d\d:\d\d:\d\d
```

will match a timestamp only if it exists at the beginning of a line. The '\b' anchor can be used to ensure a regular expression matches a whole word. For example the regular expression 'Al' would match both the words 'Al' and

'Alpha' but the regular expression:

```
\bAl\B
```

would match 'Al' but not 'Alpha'. The *Match Whole Word* toggle in the Find Bar Options can be enabled as another way to limit regular expressions to matching whole words only.

Repetition

To match multiple characters in a row, a number of different operators can be used. Some operators are *greedy* meaning they match the largest number of characters they can, or *lazy* meaning they match as few characters as they can. The following operators are supported and are by default greedy:

- **?** - 1 or 0 times
- ***** - 0 or more times
- **+** - 1 or more times
- **{n,m}** - at least n but not more than m times
- **{n,}** - at least n times
- **{,n}** - at least 0 but not more than n times
- **{n}** - exactly n times

To convert a greedy operator to a lazy operator include an additional '?' after the operator (for example, '??', '*?', or '{n,m}?'. In our phone number example from above for a number such as 555-5555 we could now use:

```
\d{3}-\d{4}
```

To match both the strings 'color' and 'colour' use the regular expressions:

```
colou?r
```

For another example, to match a simple XML tag use:

```
<[A-Za-z0-9_/]+>
```

This regular expression matches one or more alphanumeric characters inside '<' and '>' brackets. Repetition operations can also be used with the '(' and ')' brackets to indicate what is repeating. For example:

```
reg(ular)? ex(pression)?
```

matches both the strings 'regular expression' and 'reg ex'. Warning: Using certain combinations of repetition operators can cause searches to be performed very slowly.

Alternation

The alternation operator '|' allows matching one out of several possible regular expressions. For example to search for the colors red, green or blue, use:

```
red|green|blue
```

Alternation can be combined with the '(' and ')' brackets to make more complex statements. For example to search for 'const int' or 'const char' use:

```
const (int|char)
```

Matching Hex Bytes

When searching for hex bytes use the syntax '\x*HH*' to denote a hex byte where *HH* is the byte to find. This syntax must be used for regular expressions even when the Find type is set to *Hex Bytes* in the Find Bar. For example, to search for the bytes '3F 4D ?? 0F' use the regular expression:

```
\x3F\x4D.\x0F
```

Hex bytes can also be used in character classes. For example to search for the first non-zero byte use:

```
[^\x00]
```

When regular expressions are enabled, the Find type is set to *Hex Bytes* and no regular expression is being editing in the Find Bar, pressing *Ctrl+F* on the keyboard will copy the currently selected hex bytes to the Find Bar using the \x notation.

Functions

Regular expressions can be used in scripts using the FindAll, FindFirst, FindInFiles or ReplaceAll functions and the 'method=FINDMETHOD_REGEX' parameter. Regular expressions can also be used to search within strings using the RegExMatch or RegExSearch functions.

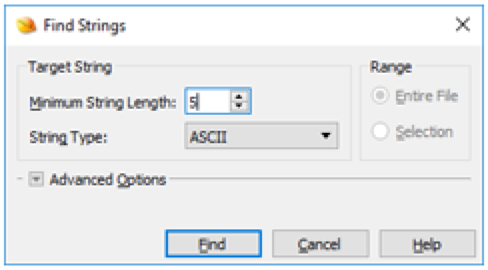
Backreferences

Backreferences are currently not supported when performing replacement operations.

Related Topics:
String Functions
Tool Functions
Using Find
Using Find In Files
Using Replace
Using Replace In Files

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Find Strings



Target String

Minimum String Length: 5

String Type: ASCII

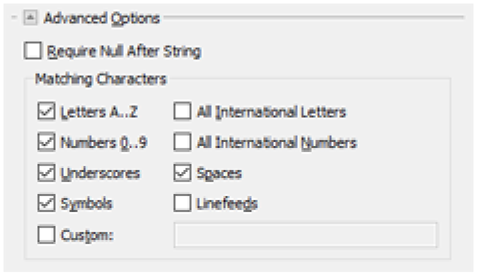
Range

[✓ Entire File] Selection

Advanced Options

The Find Strings dialog can be used to discover the location of strings within a binary file. Access the Find Strings dialog using the '*Search > Find Strings...*' menu option and note that this tool is generally not useful for text files because a text file consists entirely of a set of strings.

Use the Find Strings dialog by specifying the minimum number of characters for each string in the *Minimum String Length* field. Choose whether to search for ASCII strings, Unicode strings, or both ASCII and Unicode strings using the *String Type* field. By default, the search will take place over the whole file (*Entire File* will be selected in the *Range* group), but the search can be limited to an area of the file by making a selection before opening the Find Strings dialog and then choosing the *Selection* toggle. Click the *Find* button to search through the file and list all strings that were found and see the Output Window below. Clicking *Cancel* or pressing the Esc key will dismiss the dialog without performing a search.



Advanced Options

☐ Require Null After String

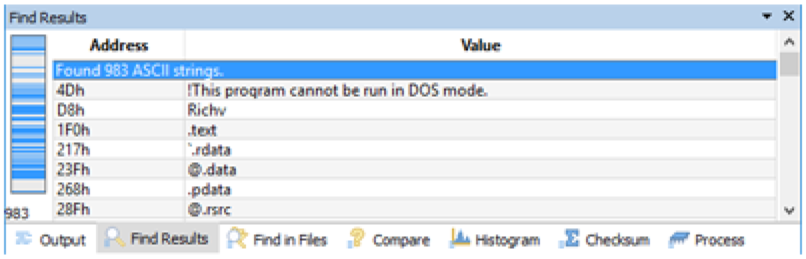
Matching Characters

☑ Letters A..Z	☐ All International Letters
☑ Numbers 0..9	☐ All International Numbers
☑ Underscores	☑ Spaces
☑ Symbols	☐ Linefeeds
☐ Custom:	

Other options are available for the Find Strings dialog by clicking the *Advanced Options* section. If the *Require Null After String* toggle is set, only those strings that have a null (zero) character immediately following the string will be listed in the Output Window. The *Matching Characters* box can be used to customize exactly which characters are considered when searching for strings. Characters are divided into a number of different groups which can be enabled or disabled by clicking the toggles. For example, if *Letters A..Z* is selected then all letters (including uppercase and lowercase) will be considered part of a string. A list of custom characters can be specified in the *Custom* field and the sequence ".." can be used to indicate a range of characters. For example, to search just for letters or the characters $, & and @, disable all toggles except the *Custom* toggle and enter "A..Za..z$&@" in the *Custom* field.

Output Window

Copyright © 2003-2019 SweetScape Software



Address	Value
Found 983 ASCII strings.	
4Dh	!This program cannot be run in DOS mode.
D8h	Richv
1F0h	.text
217h	`.rdata
23Fh	@.data
268h	.pdata
28Fh	@.rsrc

After the search is performed, all strings that were found will be displayed in the Find tab of the Output Window. The top line lists how many strings were found and clicking on a string name will highlight that string in the editor. The figure above shows an example of finding strings in the 'notepad.exe' file. A graph of where the strings were found in the file is located to the left of the list of strings. Different formats for the *Address* column can be chosen by right-clicking on a cell in the column and selecting *Column Display Format* from the popup menu. Right-click the table and select '*Clear*' to clear all the results or press the *Esc* key to hide the Output Window.

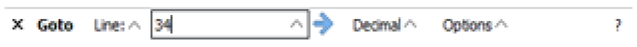
Related Topics:
Using Find

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Goto

The Goto Bar can be used to jump to any address, line, sector, or short in the current file. Access the Goto Bar by using the '*Search > Goto...*' menu option, the Tool Bar, or by pressing *Ctrl+G*.

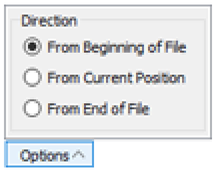
Goto Bar



When using the Goto Bar, select what type of value to jump to using the popup list to the right of the *Goto* label. The following 4 options are available:

- **Byte** - Use this setting to seek to a particular byte address in the file. Entering ',b' after a value will force the goto to jump to an address even if one of the other options is chosen in the list.
- **Line** - When *Line* is chosen, the Goto Bar is used to jump to a line in the file. Enter ',l' after a value to force the Goto Bar to jump to a line.
- **Sector** - Choose *Sector* to jump to a sector in a file or drive (see Editing Drives for more information on sectors). Enter ',s' after a value to jump to a particular sector.
- **Short** - A *Short* is a group of two bytes within a hex file. Selecting *Short* from the list will jump to the chosen short. Enter ',w' after a value to force the Goto Bar to jump to a short.

Enter a numeric value in the text field in the bar. Switch between *Decimal* and *Hex* numeric format by clicking the area just to the right of the text field, or use any of the formats described in the Introduction to Number Systems section. The position to seek will be calculated using an origin. The origin can be controlled by clicking the *Options* button and using the *Direction* radio buttons:



- **From Beginning of File -** Origin is at the beginning of the file.
- **From Current Position -** Origin is the current position and plus or minus are used to move forward or backward. For example, use '+10,l' to skip forward 10 lines or '-16,b' to skip backward 16 bytes.
- **From End of File -** Origin is at the end of the file. For example, use '<16' to jump to the 16th byte from the end of the file.

Press the *Enter* key or click the Goto icon to the right of the text field to move the cursor to the new position. Pressing the *Esc* key will hide the Goto Bar and a list of the recent Goto commands can be accessed by clicking the small up arrow to the right of the text field.

Goto Again

Once the Goto tool has been used, clicking '*Search > Goto Again*' or pressing *Ctrl+Shift+G* will jump to the address again. If the address was relative to the current position, the cursor will be moved again in the same

Copyright © 2003-2019 SweetScape Software

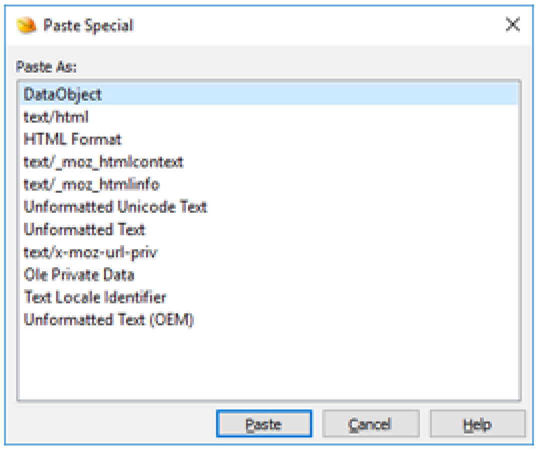
direction. For example, enter '+48,b' in the Goto Bar and press *Enter*. Then press *Ctrl+Shift+G* multiple times to step through the file by 48 bytes.

NOTE: The Goto Bar will be hidden automatically after a period of inactivity. To control the length of the period of inactivity before hiding see the General Options dialog.

Related Topics:
Editing Drives
Selecting a Range

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Paste Special



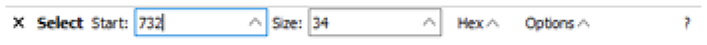
Many applications copy data to the clipboard in a variety of different formats. The *Paste Special* command allows inserting of data into the current document in any of the available formats. This command can be accessed from the '*Edit > Paste Special...*' menu option. For more information on how pasting works in general, see the Using the Clipboard topic.

Clicking the Paste Special command displays the *Paste Special* dialog show above. A list of all available data formats is displayed in the center of the dialog. Click a format name and press the *Paste* button to paste data in the selected format. Data can also be pasted by double-clicking a format name. Click the *Cancel* button to dismiss the dialog without inserting any data.

Related Topics:
Using the Clipboard

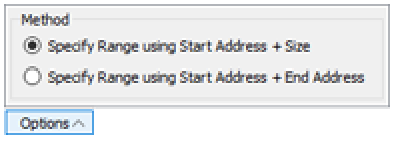
010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Selecting a Range



The Select Bar can be used to select a set of continuous bytes in a file by specifying a starting address and a number of bytes, or a start address and an end address. To open the Select Bar, click the '*Edit > Select Range...*' menu option or press *Ctrl+Shift+A*. If the Select Bar is opened when a selection is already made, the bar will display the start address and size (or end address) of the current selection. Note that when the Select Bar is displayed and the selection changes in a file, the Select bar will update to display the current selection.

By default, the Select Bar specifies selections using a *Start* and a *Size* field. The *Start* field of the bar displays the address of the first byte of the selection. If no selection is made, the address will be the current cursor position. The *Size* field displays the number of selected bytes. If no bytes are currently selected, this field shows the last number of bytes selected with this bar. Choose the numeric format for the fields by clicking either *Hex* or *Decimal* to the right of the size field, or use any of the formats described in the Introduction to Number Systems section. Press the *Enter* key to make a selection or the *Esc* key to hide the bar.


Method
● Specify Range using Start Address + Size
○ Specify Range using Start Address + End Address
Options

Alternately, selections can be controlled using a *Start* address and an *End* address. To enable this mode, click the *Options* button and enable the *Specify Range using Start Address + End Address* toggle. Enter the address of the end of the selection in the *End* field (note that the *Size* field disappears). For example, entering a *Start* address of 1000 and an *End* address of 1005 would select 5 bytes (the byte at the end address is not selected).

When a selection is made, the start address and size of the selection are displayed in the status bar. See the Status Bar help topic for more information.

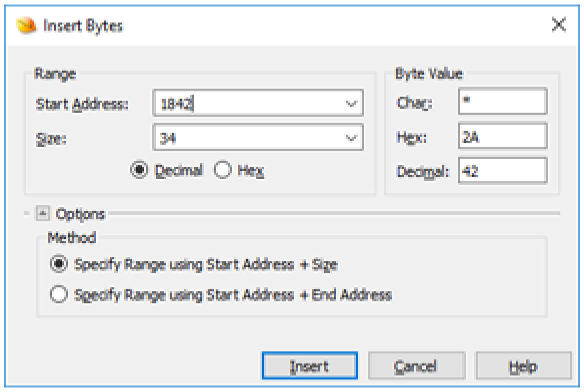
NOTE: The Select Bar will be hidden automatically after a period of inactivity. To control the length of the period of inactivity before hiding see the General Options dialog.

Related Topics:
Introduction to Number Systems
Selecting Bytes
Status Bar
Using the Clipboard

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

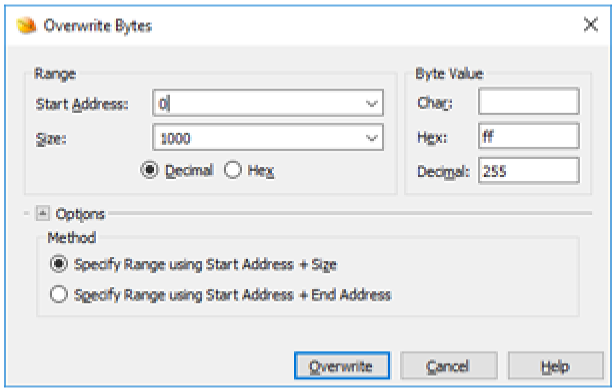
Inserting or Overwriting Bytes

Two tools are included with 010 Editor that make inserting or overwriting blocks of bytes easy (for example, if you want to insert a row of eighty '*' characters these tools make this operation simple).



Click the '*Edit > Insert/Overwrite > Insert Bytes...*' menu option to access the Insert Bytes tool. The current cursor address will be displayed in the *Start Address* field. Enter the number of bytes to insert in the *Size* field. The address and size can be displayed in decimal or hex formats by clicking on the *Decimal* or *Hex* radio buttons. Usually the range of bytes to insert is specified using the *Size* field but the range can also be specified using an *End Address* by clicking the *Options* button and enabling the *Specify Range using Start Address + End Address* toggle (note that the range does not include the byte at the end address).

The value of the bytes to be inserted can be controlled in the *Byte Value* box by entering a value in the *Char*, *Hex* or *Decimal* fields. Note that the value is automatically converted between the different formats as a number is entered in either field (the *Char* field will be left blank if there is no printable character that corresponds to the byte value). Click the *OK* button to insert the bytes, or the *Cancel* button to close the dialog without making changes. Note that bytes cannot be inserted into a drive or a process.



Click the '*Edit > Insert/Overwrite > Overwrite Bytes...*' menu option to access the Overwrite Bytes tool. If any bytes are selected in the editor, the starting address of the selection will be displayed in the *Start Address* field and the number of bytes selected will be displayed in the *Size* field; otherwise, the current cursor position will be listed in the *Start Address* field and the last number of filled bytes will be listed in the *Size* field. The range of bytes to overwrite can also be specified by using a start address and end address (see the Insert Bytes dialog above for more information). A *Byte Value* can be entered as when using the Insert Bytes tool. Click *OK* to set all the bytes specified by the *Range* to the given *Byte Value*. The *Cancel* button will close the dialog with no changes.

Related Topics:
Inserting Files
Selecting Bytes
Setting the File Size
Using the Hex Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Inserting or Overwriting Files

A file can easily be inserted into another file using the *Insert File* or *Overwrite File* tools. The *Insert File* tool inserts the bytes from another file into the current file at the current cursor position whereas the *Overwrite File* overwrites any bytes in the current file with another file starting at the current cursor position. Access the *Insert File* tool by clicking the '*Edit > Insert/Overwrite > Insert File...*' menu option or pressing *Ctrl+I*. Access the *Overwrite File* using the '*Edit > Insert/Overwrite > Overwrite File...*' menu option.

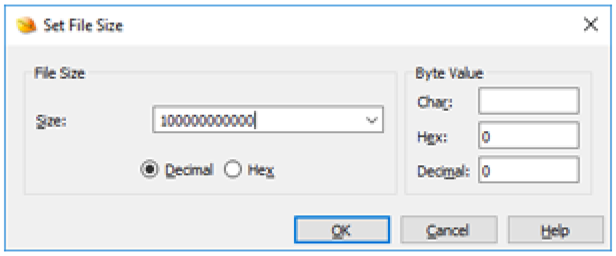
Position the cursor at the address to insert or overwrite the file and activate the desired tool. Select any file with the file dialog box that is shown and click the *Open* button. Note that files cannot be inserted into drives or processes since the file size of drives and processes is always fixed (but data can be overwritten using the *Overwrite File* tool).

NOTE: 010 Editor employs a read-on-demand data engine that allows even huge files to be instantly inserted or overwritten. As a result, the inserted file should not be deleted until the edits have been saved to disk (see Introduction to the Data Engine for more information).

Related Topics:
Inserting or Overwriting Bytes
Introduction to the Data Engine

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Setting the File Size



Size: 100000000000
[✓ Decimal] Hex
Char:
Hex: 0
Decimal: 0

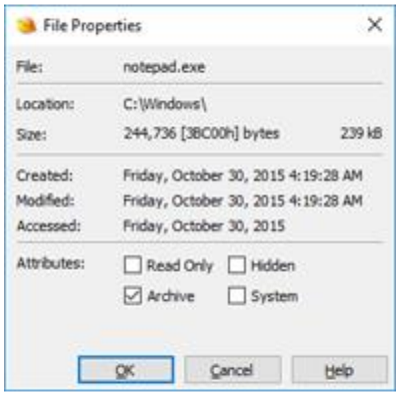
010 Editor contains a useful tool for setting the number of bytes in the current file. Note that the size of drives and processes is fixed and cannot be edited with this tool. Click the '*Edit > Set File Size...*' menu option to display the Set File Size dialog. Enter the desired file size in the *Size* field. Note that either hex or decimal formats can be used, depending on the *Hex* and *Decimal* radio buttons. If the new file size is larger than the current file size, a number of bytes will be appended to the file. The value of the inserted bytes can be controlled using the *Hex*, *Decimal*, and *Char* fields in the *Byte Value* box. Note that when typing a value in one field, the other fields will automatically display the converted value (the *Char* field is empty in the above screenshot because the byte value 0 cannot be converted to a printable character). If the new file size is less than the current size, bytes will be deleted from the end of the file. Click *OK* to perform the operation or *Cancel* to close the dialog.

Related Topics:
Inserting or Filling Bytes

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

File Properties

The File Properties dialog displays useful information about the current file. This dialog can be accessed from the '*Edit > File Properties...*' menu option or by pressing *Alt+Enter*. The Properties dialog will display different information if the current file is a logical drive, physical drive, or process. See below for the information displayed.



File Properties

File: notepad.exe

Location: C:\Windows\

Size: 244,736 [3BC00h] bytes 239 kB

Created: Friday, October 30, 2015 4:19:28 AM

Modified: Friday, October 30, 2015 4:19:28 AM

Accessed: Friday, October 30, 2015

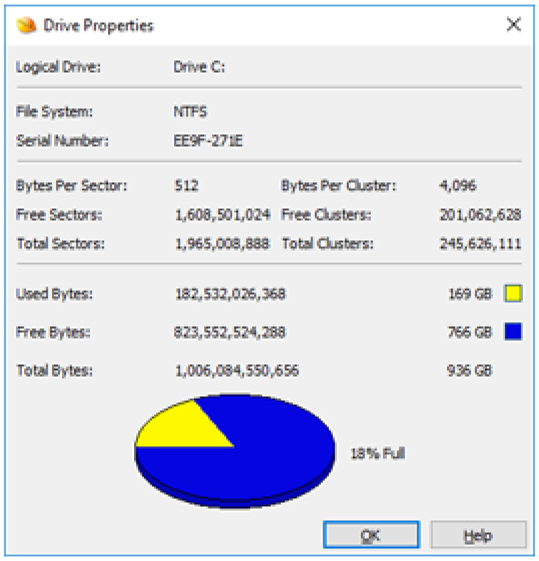
Attributes: ☐ Read Only ☐ Hidden ☑ Archive ☐ System

The File Properties dialog displays the current *File*, *Location*, and *Size* in both hex and decimal formats. The *Created*, *Modified*, and *Accessed* fields display the time and date when the file was created, last changed on disk, and accessed from disk respectively. Note that on some operating systems, these dates are not always 100% accurate.

The *Attributes* area shows the *Read Only*, *Hidden*, *Archive*, and *System* file flags from the operating system. Note that these toggles can be clicked on to change the attributes of the file on disk.

Click the *OK* button to accept any file attributes changes, or the *Cancel* button to dismiss the File Properties dialog without making any changes.

Logical Drive Properties


Drive Properties

Logical Drive:	Drive C:		
File System:	NTFS		
Serial Number:	EE9F-271E		
Bytes Per Sector:	512	Bytes Per Cluster:	4,096
Free Sectors:	1,608,501,024	Free Clusters:	201,062,628
Total Sectors:	1,965,008,888	Total Clusters:	245,626,111
Used Bytes:	182,532,026,368	169 GB	🟨
Free Bytes:	823,552,524,288	766 GB	🟦
Total Bytes:	1,006,084,550,656	936 GB	

18% Full

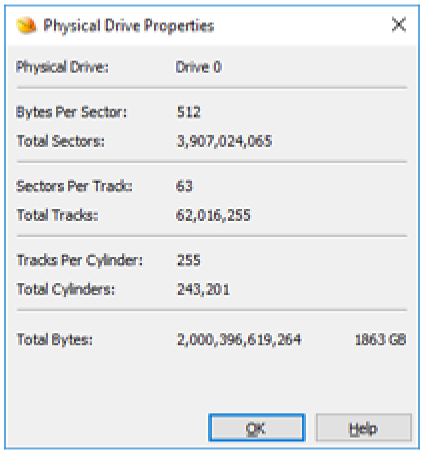
The Drive Properties dialog display information about a logical drive (for example, 'C:' or 'D:'). The dialog lists the current *File System* (NTFS, FAT, CDFS, etc.) plus the *Serial Number* of the drive.

Each drive is divided into a number of sectors and clusters (see Editing Drives for more information). The middle section of the dialog gives the size of each sector and cluster, plus the number of free and total sectors and clusters.

The bottom section of the dialog displays the total number of bytes in the drive, the number of bytes used, and the number of free bytes. As well a graph indicates what percentage of the drive is full (yellow indicates used bytes and blue indicates unused bytes).

Click the *OK* button to dismiss the Drive Properties dialog.

Physical Drive Properties


Physical Drive Properties

Physical Drive:	Drive 0	
Bytes Per Sector:	512	
Total Sectors:	3,907,024,065	
Sectors Per Track:	63	
Total Tracks:	62,016,255	
Tracks Per Cylinder:	255	
Total Cylinders:	243,201	
Total Bytes:	2,000,396,619,264	1863 GB

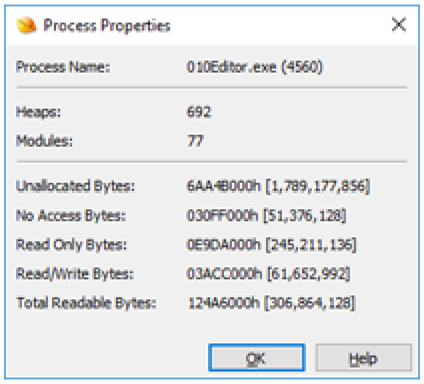
When editing a physical drive, the properties dialog will display a different set of information about the drive than a logical drive (see above). For information on the difference between a logical and a physical drive, see Editing

Drives.

A physical drive is made up of a number of sectors, tracks, and cylinders (see Editing Drives). The middle section of the dialog lists the number of bytes per sector, track, and cylinder, as well as the total number of each on the drive (the number of tracks and cylinders is not available on all devices). The bottom part of the dialog displays the total number of bytes available on the drive. Note that information on how much of the drive is free or in use is not available for physical drives.

Click the *OK* button to dismiss the Physical Drive Properties dialog.

Process Properties



Process Name:	010Editor.exe (4560)
Heaps:	692
Modules:	77
Unallocated Bytes:	6AA4B000h [1,789,177,856]
No Access Bytes:	030FF000h [51,376,128]
Read Only Bytes:	0E9DA000h [245,211,136]
Read/Write Bytes:	03ACC000h [61,652,992]
Total Readable Bytes:	124A6000h [306,864,128]

If the current file is a process, then the Process Properties dialog will be displayed that contains information on the current process. The top portion of the dialog lists the *Process Name*, and the number of *Heaps* and number of *Modules* of the process (see Editing Processes for more information).

The bottom portion of the dialog displays the number of bytes that do not belong to the process (*Unallocated Bytes*) and the number of bytes that are allocated by cannot be accessed (*No Access Bytes*). As well the total number of bytes that are marked as read only is displayed (*Read Only Bytes*), the number of bytes that are marked for reading and writing (*Read/Write Bytes*), and the total number of readable bytes (*Total Readable Bytes*).

Click the *OK* button to dismiss the Process Properties dialog.

Related Topics:
Editing Drives
Editing Processes
Using the Workspace

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Importing/Exporting Files

Importing and Exporting allows conversion between a binary file and a number of supported formats. When importing or exporting files, the following formats are supported:

- **Hex Text -** Stores a binary file as a text file containing a series of bytes in hexadecimal format. For example, the hex bytes 0x3F and 0x61 would be stored as the characters "3F 61". Hex bytes formatted as "0x31,0x0,0x8" are also supported for import.
- **Decimal Text -** Stores a binary file as a text file where each byte is converted to a decimal number. For example, the hex byte 0xFF would be stored as the characters "255".
- **Binary Text -** Stores a binary file as a text file where each byte is converted to a binary number. For example, the binary byte 0x6F would be stored as the characters "01101111".
- **C Code or Java Code -** Converts a binary file to an array of bytes that could be included in a C/C++ or Java program. (When importing data, specify the import type as 'Source Code' and 010 Editor will automatically detect whether the data is C or Java code).
- **Intel 8, 16, or 32-Bit Hex Code -** Stores a binary file in the Intel Hex format. A number of different variations of the format exist, including 8-bit, 16-bit, and 32-bit. The Intel Hex format is used in a number of different applications and is commonly used with EPROMs.
- **Motorola S19, S28, or S37 Records -** Motorola S-Record format is used for transferring binary files and is commonly used with EPROMs.
- **Text Area -** Stores the currently selected bytes as text exactly how they are displayed in the hex editor, including addresses and both the left and right areas. For example:

```
      0030h: 3031 3233 3435 3637 3839 3A3B 3C3D 3E3F
0123456789:;<=>?
      0040h: 4041 4243 4445 4647 4849 4A4B 4C4D 4E4F
@ABCDEFGHIJKLMNO
      0050h: 5051 5253 5455 5657 5859 5A5B 5C5D 5E5F
PQRSTUVWXYZ[\]^_
      0060h: 6061 6263 6465 6667 6869 6A6B 6C6D 6E6F
`abcdefghijklmno
      0070h: 7071 7273 7475 7677 7879 7A7B 7C7D 7E7F
pqrstuvwxyz{|}~
```

  Note that importing from this format is not supported.

- **Web Page (HTML) -** Similar to the Text Area option above except that the data is stored in HTML format suitable for being placed on a webpage. Any coloring of the data is also recorded in the HTML data. Note that data can only be exported into HTML format (importing is not supported).
- **Rich Text Format (RTF) -** Similar to the Text Area option above except the data is stored in Rich Text Format (also called RTF). This format is used by a number of word processors including Microsoft Word. 010 Editor includes all foreground and background coloring information with the RTF but note that some programs such as Microsoft Word do not support having background colors in RTF data. Exporting to HTML will retain both the foreground and background colors more reliably. Only export is supported for this format.
- **Base64 -** Base64 is a method of encoding binary data so that it may be transferred between different systems without losing any special characters. This format is used when transferring attachments over email and other applications on the internet. Both importing and exporting are supported for Base64 data.
- **Uuencoding -** Uuencoding is a method of encoding data similar to Base64 but with different parameters. Uuencoding is used for transferring attachments over email or newsgroups, as well as other applications. Importing and exporting are both supported for uuencoded data.

Importing Files

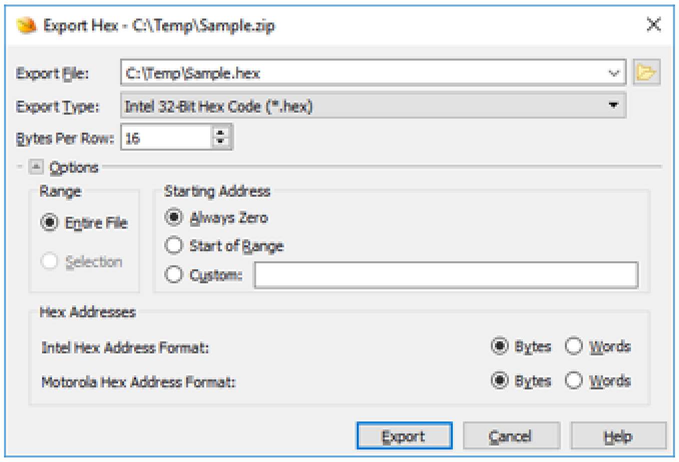
Files may be imported by clicking the '*File > Import Hex...*' menu option. Select a file to import using the displayed file dialog box. By default, all files that can be imported will be displayed and the file type will be set to '*All Supported Import Types*' but the file type can be changed to only display files of one type. Once the file is imported, it will be converted to a binary file and opened as a new file in the editor. Any of the above formats can be imported into 010 Editor except where indicated. 010 Editor contains some special functionality for importing Intel Hex or Motorola S-Records (see Opening Files for more information). The Directory Options dialog can be used to control the initial directory when the file dialog box is displayed.

When importing a file, any bytes that are skipped are set to zero value by default; however, this value can be changed using the *Default Import Byte* option in the Importing Options.

In some special Intel Hex or Motorola files, the addresses specified in the file indicate the position of the Word where the data exists. To convert from this Word-based addressing system to a Byte-based addressing system, enable the *Words* toggle in the Importing Options dialog for either Intel Hex or Motorola files (internally, the addresses are multiplied by two). Note that leaving this toggle enabled when the import file does not use Word-based addresses will cause undefined results.

Exporting Files

To export the current file to one of the above formats, open the file and click the '*File > Export Hex...*' menu option. The Export Hex dialog will be displayed.


Export Hex - C:\Temp\Sample.zip

Export File: C:\Temp\Sample.hex

Export Type: Intel 32-Bit Hex Code (*.hex)

Bytes Per Row: 16

Options

Range: [✓ Entire File] Selection

Starting Address: [✓ Always Zero] Start of Range Custom:

Hex Addresses

Intel Hex Address Format: [✓ Bytes] Words

Motorola Hex Address Format: [✓ Bytes] Words

Choose a file name for the exported file in the *Export File* field. A file name can also be chosen by clicking the Browse button to the right of the field. The default directory for the file to export can be controlled using the Directory Options dialog. Select which type of file to export using the *Export Type* drop-down list. Note that changing the export type automatically modifies the extension of the file to save. The number of bytes per line in the output file can be adjusted using the *Bytes Per Row* field.

To view advanced options for the file to export, click the *Options* button. Choose the *Entire File* radio button to export all the bytes from the current file or if a selection is made, choose the *Selection* toggle button to only export the selected bytes.

The Intel Hex and Motorola S-Records formats support specifying a starting address of the data. When the *Always Zero* toggle is selected, the address will always be written as zero. If the *Start of Range* toggle is selected, the starting address of the exported bytes will be written (this is usually zero unless a selection is exported using the *Selection* toggle). A custom address can also be specified by selecting the *Custom* toggle and entering a number in the corresponding text field.

When exporting Intel Hex or Motorola files, the output addresses can be set to Byte-based addresses by selecting the *Bytes* toggle (the default) or to Word-based addresses by clicking the *Words* toggle. See the Importing Files section above for more information on Word-based addresses. If the *Words* toggle is enabled, the addresses will be divided by two when exporting.

Click the *Export* button to create the exported file or the *Cancel* button to dismiss the dialog without exporting.

Importing or Exporting Data Through the Clipboard

A quick way of importing or exporting data exists in 010 Editor by using the clipboard. To quickly export data, select the bytes to export and click '*Edit > Copy As*' and then select the type of data to export. The data will be exported and the results copied to the clipboard. Then the data can be pasted to another application (for example, copy data to the clipboard using '*Edit > Copy As > Copy As Web Page (HTML)*' and paste the data into an HTML editor such as Microsoft Word). The data will be exporting using the same options as when the selected format was last exported using the '*File > Export Hex...*' menu option.

Data can be imported quickly into 010 Editor by copying the data to import to the clipboard, then clicking '*Edit > Paste From*' and choosing which format to import. The data will be imported and inserted into the file at the current cursor position.

Related Topics:
Converting Files
Directory Options
Edit Menu
Importing Options
Opening Files

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Command Line Parameters

Opening Files

A set of files to load can be specified on the command line when starting 010 Editor. Each file to load should be separated by a space. For example, to load two files use:

```
010editor file1.dat file2.dat
```

Multiple files can be loaded on the command line by using the wildcards '*' and '?'. For example:

```
010editor *.bin file???.dat
```

By default, when 010 Editor is installed it is placed in the system path. This means that 010 Editor can be run from any command line by entering '010editor' (no directory needs to be specified). This command line syntax can be used to load files even if 010 Editor is already running. To not place 010 Editor in the path, disable the '*Add 010 Editor to the system path*' toggle in the install program.

Opening Drives

Drives can be opened from the command line by using the **-drive:** command, followed by either a drive label or a drive number. If a drive label is specified, a logical drive is opened and if a drive number is specified, a physical drive is specified (see Editing Drives for more information). For example:

```
010editor -drive:C -drive:1
```

would open logical drive C: and physical drive 1.

Opening Processes

Processes can be opened using the **-process:** command, followed by either a process identification number or a name of a process. For more information on working with processes, see the Editing Processes help topic. For example, to open two processes from the command line use:

```
010editor -process:cmd.exe -process:1074
```

Importing Files

To import any of the available file formats, use the **-import:** command, followed by the file to load. Any of the accepted import or export types are accepted and the type used will be based from the file extension (see Importing/Exporting Files for more information). For example, to import a C file use:

```
010editor -import:array.c
```

The wildcard characters '*' and '?' can also be used to import multiple files at the same time.

Position the Cursor

The cursor can automatically be positioned within a file using the **-line:** or **-goto:** commands. Specify **-line:** followed by a number to jump to that line within the file. For example:

```
010editor file1.txt -line:100
```

To jump to a specific address within a file use the **-goto:** command followed by a number and the number may be in any of the supported numeric formats. For example:

```
010editor file1.dat -goto:0x20
```

All the options from the Goto Bar are available, including using '+' or '-' to jump relative the current location or '<' to jump relative the end of the file. For example, to skip over 32 bytes use:

```
010editor -goto:+32
```

Using **-goto:** to jump to a line, sector or word is also possible by including either ',l', ',s' or ',w' at the end of the command respectively. For example, to jump to sector 0x100 in a file use:

```
010editor -goto:0x100,s
```

Note that versions of 010 Editor before version 6 could use '@' to jump to a location. This syntax is still allowed in version 6 but will be deprecated in future versions.

Making Selections

Selections can be made in a file using the command **-goto:**<*start*>*:*<*size*>. Note that *start* and *size* refer to bytes and any of the standard numeric formats are accepted. For example, to select 4 bytes starting from address 16 use:

```
010editor -select:16:4
```

If *start* is empty, the selection is started from the current cursor position which can be controlled with the **-goto** command. For example, to select 0x200 bytes starting from the cursor position use:

```
010editor -select::0x200
```

Opening Scripts or Templates

Scripts or Templates can be run from the command line using the **-script:** or **-template:** command respectively. See Introduction to Templates and Scripts for more information on Templates and Scripts. To run the script IsASCII for example, use:

```
010editor -script:IsASCII.1sc
```

The file to run must exist in either the current directory, the current 'Scripts' directory if running a script, or the current 'Templates' directory if running a template (these directories can be modified in the Compiling Options). To open the Script or Template in the interface without running the file, specify a '@' symbol and a line number after the filename. For example, to open the ZIPTemplate.bt file and position the cursor on the 3rd line, use:

```
010editor -template:ZIPTemplate.bt@3
```

Command line arguments can be passed to a script or template by using the syntax **:(<*arg1*>,<*arg2*>,...,<*argN*>)** after the script or template command. For example:

```
010editor -script:MyScript.1sc:(15,Test)
```

would pass the two arguments "15" and "Test" to the MyScript.1sc script. To include spaces in any of the arguments, place double quotes around the whole command. The arguments can be retrieved in the script or template using the GetNumArgs, GetArg and GetArgW functions.

Saving and Closing Files

The current open file can be saved to disk by using the **-save** command. To save a file to a different file name, use the command **-save:<*filename*>**. For example:

```
010editor temp.txt -save:temp.txt.bak
```

To save all modified files use the **-saveall** command or to close the current file, use the **-close** command.

Replacing Strings or Bytes

A string or a set of bytes can be replaced from the command line using the **-replace:<*find_value*>:<*replace_value*>** command. For example:

```
010editor temp.txt -replace:apple:orange
```

would replace all occurrences of apple with orange in the file temp.txt. Note that if there are spaces in any of the replace strings, surround the whole -replace command with double quotes. The replacement options can be controlled, and special characters can be inserted into strings through the use of escape codes which start with the character '\'. The following special escape codes can be used:

\m = match case
\w = match whole word
\p = pad with zeros
\a = perform replacement on all open files
\* = search with wildcards

\\ = insert a '\' character
\: = insert a colon
\' = insert double quotes "

For example, to replace the string 'first' with 'second' using the match case and whole word options, use:

```
    010editor -replace:first:second\m\w
```

Different data types can be indicated using a comma and type specifier after the value (e.g. ',h' indicates hexidecimal bytes). For example, use

```
    010editor -replace:0D0A,h:0A,h
```

to replace Windows linefeeds with Unix linefeeds. See the Find dialog for a list of type specifiers.

Comparing Files

Two files can be compared from the command line using the **-compare:<*fileA*>::<*fileB*>** or the **-compare:<*fileA*>::<*fileB*>::<*options*>** command. For example:

```
    010editor -compare:c:\01.dat::c:\02.dat
```

would run a regular binary comparison on the two given files. Notice the double-colon '::' between the file names and if there are spaces in either file name, surround the whole -compare command with double quotes. To specify options for the comparison, use another double-colon '::' after the second file name and specify one or more of the following special escape codes (note that 'XXX' indicates a number):

\b = byte by byte comparison
\i = ignore case
\e = enable synchronized scrolling
\t = enable synchronized template results scrolling
\xXXX = max look-a-head
\nXXX = min match length
\qXXX = quick match length
\saXXX = limit start for file A
\sbXXX = limit start for file B
\zaXXX = limit size for file A
\zbXXX = limit size for file B

For example, to compare two files and ignore the case, use:

```
    010editor -compare:c:\01.dat::c:\02.dat::\i
```
or to limit the comparison to the first 16 bytes of each file, use:
```
    010editor -compare:c:\01.dat::c:\02.dat::\i\za16\zb16
```

See Comparing Files for an explanation of the different options for comparisons. Note that if a Max Look-a-head, Min Match Length, or Quick Match is not specified the values from the last time the Compare dialog was run will be used. Running '010editor -compare:' will give a list of all comparison options.

Running 010 Editor in Batch Files

If running 010 Editor from a batch file, it is possible to pass error level codes from a script or template back to the batch file. First, in a script or template call the function Exit with an error code (e.g. 'Exit(95);'). Next, in a batch file start 010 Editor by using the syntax '*start /wait 010editor ...*'. Afterwards the error code can be accessed in the batch file using the variable %ERRORLEVEL%. For example:

```
start /wait 010editor test.txt -template:test.bt -exit
echo %ERRORLEVEL%
```

Running 010 Editor without a User Interface

When running 010 Editor from the command line, the software can be executed without a user interface by specifying the **-noui** command. In this mode, the splash screen and main window of 010 Editor are not displayed and the program will exit automatically when all the command line options are executed. Note that in this mode, messages boxes may still be displayed on error messages and this may stop program execution until the message box is cleared by the user. To disable the display of any message boxes, include the **-nowarnings** option on the command line. The **-noui** command line option is very useful when running 010 Editor from a batch file (see above).

Safe Mode

The **-safe** command starts 010 Editor in safe mode. In this mode no scripts or templates are run on startup and this is useful if a script or template was crashing and causing 010 Editor to not start correctly.

Resetting the Application

Options for 010 Editor can be reset by using the command line. Specify the **-resetdocks** command on startup to reset just the docking panel positions. Specify **-resetall** to revert all application settings to their defaults. If 010 Editor will not startup, specifying **-resetdocks** or **-resetall** will usually fix the problem.

Reinstalling the Application

On macOS the **-install** command can be used to perform the installation steps done when 010 Editor was run for the first time. This currently only includes displaying the End-User License Agreement (EULA) and checking if the application has been added to the system path.

Exiting the Application

Use the **-exit** command to shut down 010 Editor from the command line (all unsaved modifications will be lost). Alternately the **-exitnoerrors** command can be used to shut down 010 Editor only if no errors occurred. Here an error is defined to be a Script or Template that did not compile or execute properly, or a Script or Template that was halted using the Exit function with a negative error code.

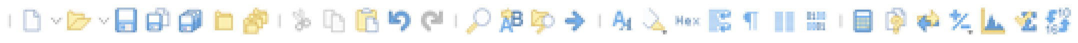
Other Command Line Parameters

The **-readonly** command can be used to set the last opened file, drive, or process to Read Only and the **-readonlyall** command sets all open files as Read Only. The **-h** command will display this manual page.

Related Topics:
Compiling Options
Editing Drives
Editing Processes
Interface Functions
Introduction to Templates and Scripts
Importing/Exporting Files
Opening Files
Using Find

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using Tool Bars



The Tool Bars are a set of icons along the top of the screen and most icons are associated with a command on the main menu (the icon will appear to the left of the menu option when browsing the main menu). Some icons have a small down arrow in the lower-right corner that indicates the icon will drop down a list of items when clicked. Placing the mouse over an icon for a second will display a hint containing the name of the icon and the keyboard shortcut. Click and drag on the small vertical bars in the Tool Bars to rearrange or undock the Tool Bars.

Five different icons in the tool bar may be highlighted (pressed in) depending upon the current file. The *Toggle Hex Interface* icon will be highlighted when the current file uses a hex-based File Interface, the *Word Wrap* icon will be highlighted when the current text file is in Word Wrap mode, the Show Whitespace icon will be highlighted when whitespace is visible in the file, the *Column Mode* icon will be highlighted when the current file is in column mode, and the *Toggle Endian* icon will be highlighted when the current file is in big-endian mode (see Byte Ordering for more information).

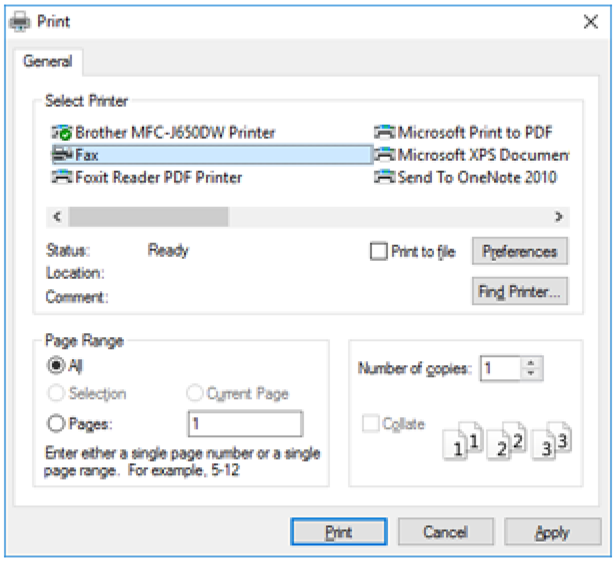
Right-click on the Tool Bar background to display a popup menu with a list of all Tool Bars. Click on a Tool Bar name in the list to either show or hide that Tool Bar and this list can also be accessed from the View Menu. Tool Bars can be customized by selecting *Customize...* from the right-click menu or clicking '*Tools > Options...*' and selecting *Toolbars* from the list (see Toolbar Options for more information).

Related Topics:
Edit Menu
File Menu
Introduction to Byte Ordering
Theme/Color Options
Toolbar Options
Tools Menu
Using the Text Editor
View Menu
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Printing

010 Editor contains a powerful printing tool, complete with print preview, headers, footers, and margins. To configure the display of the current page, see the Page Setup Dialog. To obtain a preview of how the document will appear when printed, see the Print Preview Dialog. Once the document is configured properly, click the '*File > Print...*' menu option to open the Print Dialog (note that the Print Dialog may appear slightly different than below depending on the operating system).



Select the output printer from the *Select Printer* list box. Printer-specific options can be edited by clicking the *Preferences* button.

Select which pages to print using the *Print Range* options. *All* prints the whole document. The *Pages* radio button can be used to print a range of pages. If a selection is made in the current document, the *Selection* radio toggle can be selected to print just the selected bytes.

More that one copy of the current document can be printed by entering a number into the *Number of copies* field. When printing documents of more than one page, the copies can be collated or uncollated by clicking the *Collate* toggle (if your printer supports this option). For example, if 2 copies of a document of 3 pages are printed, the page order when collated would be 1, 2, 3, 1, 2, 3, and the page order when uncollated would be 1, 1, 2, 2, 3, 3.

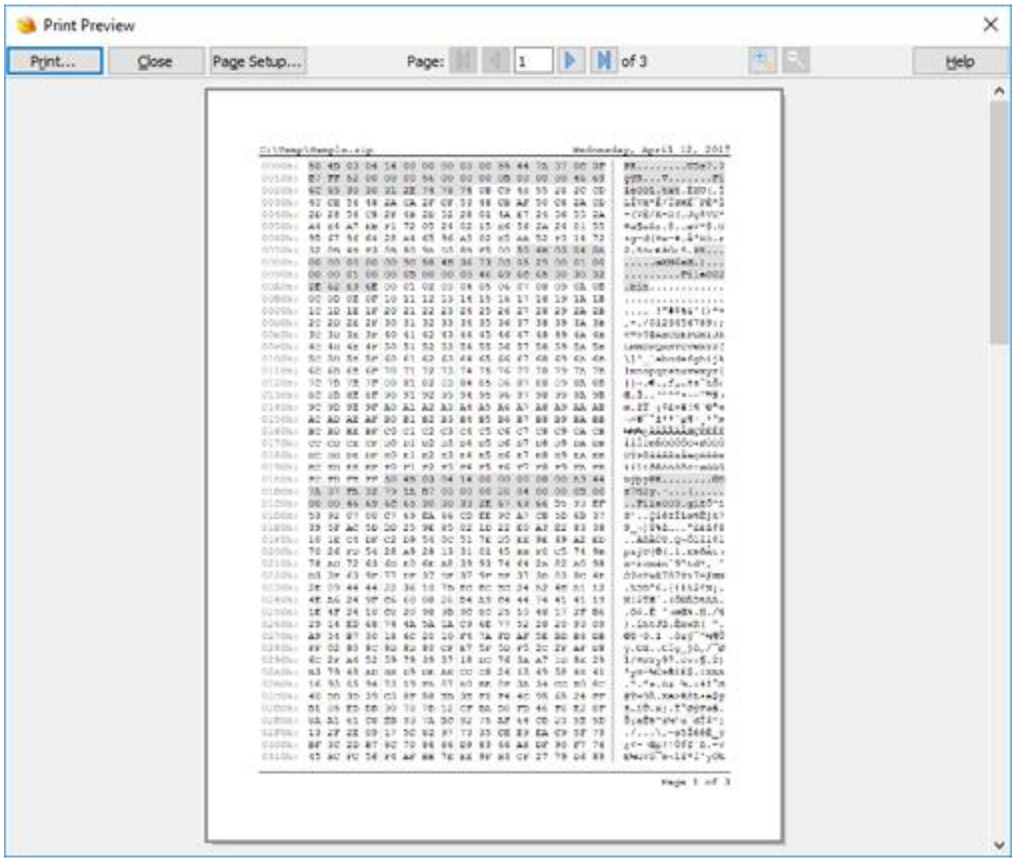
Click the *Print* button to send the document to the printer, or the *Cancel* button to close the dialog.

Related Topics:
Page Setup
Print Preview

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Print Preview

The Print Preview dialog shows an image representing how the current document would appear if printed. This dialog shows a series of pages, complete with margins, header, and footer. The '*File > Print Preview...*' menu option can be used to access the Print Preview dialog.



When the mouse cursor is positioned over the document, the cursor will change to a magnifying glass. Click the left mouse button to zoom into medium-resolution mode and click again to zoom into high-resolution mode. Clicking the mouse a third time will return to low-resolution mode. Note that the Zoom In and Zoom Out icons (located at the top-right of the dialog) can also be used to change the zoom factor.

A text field at the top of the dialog shows the current page number and the total number of pages (indicated as '*Page* <current> *of* <total>'). A value can be typed into the page number text field to jump to a particular page. Clicking the Next Page or Previous Page icons (the left and right arrows) will view the next or previous page in the document respectively. The First Page or Last Page icons (the left and right arrows with a vertical bar) will view the first or last page in the document respectively. The mouse wheel can also be used to scan through the pages to print.

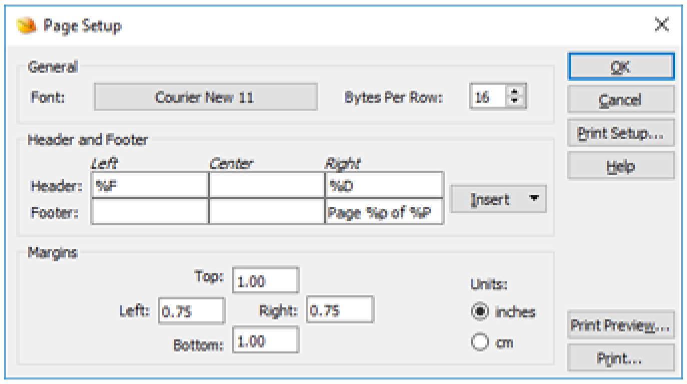
Clicking the '*Print...*' button will show the Print Dialog that can be used to send the document to the printer (see Printing for more information). The *Cancel* button will close the current dialog. Finally, the *Page Setup...* button will display the Page Setup dialog that is used to configure the margins, font, headers, and orientation for the document (see Page Setup for more information).

Related Topics:
Page Setup
Printing

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Page Setup

The Page Setup dialog controls how the page appears when printing. Use the Page Setup dialog to set margins, headers and footers, fonts, and paper orientation for printing. Click the '*File > Page Setup...*' menu option to open the dialog.


Page Setup

General
Font: Courier New 11
Bytes Per Row: 16

Header and Footer
	Left	Center	Right
Header:	%F		%D
Footer:			Page %p of %P

Margins
Top: 1.00
Left: 0.75
Right: 0.75
Bottom: 1.00
Units: [✓ inches] cm

When printing a file, most of the options from the current File Interface are used, except for the font and number of bytes per row (when printing hex data). Click the *Font* button to set the font to use for the printer. Use the *Bytes Per Row* field to control how many bytes are printed on each line of the output.

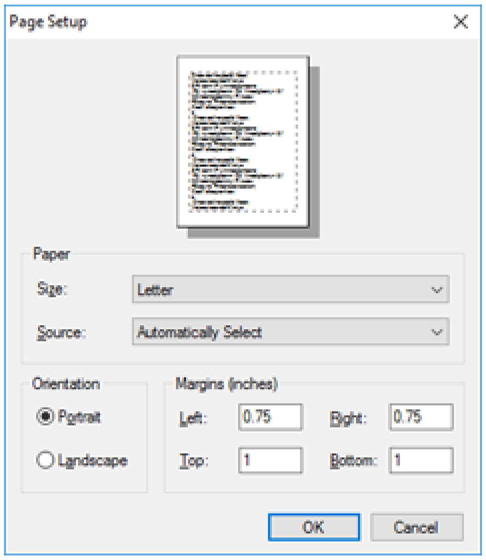
The *Header and Footer* fields control the text displayed in the top and bottom margins respectively. Text entered into either of the *Left* fields will appear left-justified and text entered into either of the *Right* fields will appear right-justified. Similarly, text in the *Center* fields will appear in the center of the page. Each field can contain a combination of regular text and special codes, indicated by a '%' symbol followed by one or two characters. Click the *Insert* > button to see a full list of available codes and click one of the codes from the list to enter it into the current text field. When the page is printed, the special codes will be replaced with the required information. The following is a list of available codes:

- **File Name (%f) -** The name of the file to be printed without path.
- **File Path (%F) -** The name of the file to be printed included path.
- **Current Page (%p) -** Current page number being printed.
- **Total Pages (%P) -** Total number of pages in document.
- **File Time - AM/PM (%t) -** Last modified time for the file, displayed with am/pm indictor.
- **File Time - 24 Hour (%T) -** Last modified time for the file, displayed in 24-hour format.
- **File Date - Short (%d) -** Last modified date for the file, displayed in 'MM/DD/YY' format.
- **File Date - Long (%D) -** Last modified date for the file, displayed in 'Weekday, Month DD, YYYY' format.
- **Current Time - AM/PM (%ct) -** Current time, displayed with am/pm indictor.
- **Current Time - 24 Hour (%cT) -** Current time, displayed in 24-hour format.
- **Current Date - Short (%cd) -** Current date, displayed in 'MM/DD/YY' format.
- **Current Date - Long (%cD) -** Current date, displayed in 'Weekday, Month DD, YYYY' format.

The *Margin* fields indicate how much space is required from the edge of the paper to the start of the hex printout. Note that the *Header* and *Footer* are printed in the margins. Enter a value in the *Top*, *Right*, *Bottom*, or *Left* fields to control the margin size in either of those directions. If the *Units* radio buttons is set to *inches*, the fields will be displayed in number of inches. If the *Units* radio button is set to *cm*, the fields will be displayed in centimeters.

Print Setup

Click the *Print Setup* button to show the Print Setup dialog. The dialog may be used to choose the printer used when printing. Also, the size or source of the paper can be chosen. Click the *Portrait* radio button to print in portrait (upright) format, or the *Landscape* button to print in landscape (sideways) format.


Page Setup

Paper
Size: Letter
Source: Automatically Select

Orientation
(● Portrait) ○ Landscape

Margins (inches)
Left: 0.75 Right: 0.75
Top: 1 Bottom: 1

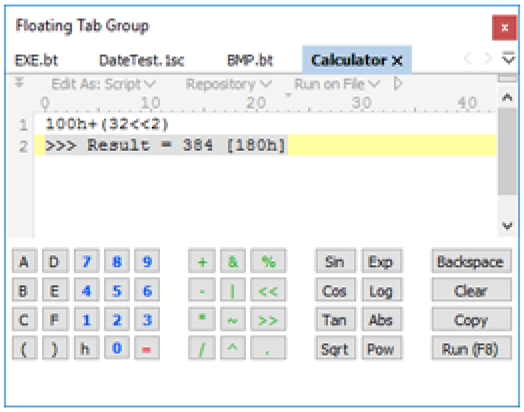
Click the *Print Preview* button in the Page Setup dialog to close the dialog and display the Print Preview (see Print Preview). Clicking the *Print* button will close the dialog and open the Print dialog (see Printing). Press the *OK* button to close the dialog with the current changes made or the *Cancel* button to discard all changes.

Related Topics:
Print Preview
Printing
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Calculator

The Calculator provided with 010 Editor is a full expression calculator using a syntax similar to C. The calculator can be loaded by clicking the '*Tools > Calculator...*' menu option or pressing *F8*.



```
100h+(32<<2)
>>> Result = 384 [180h]
```

Enter values into the calculator by clicking the buttons at the bottom of the calculator or by typing on the keyboard. Note that many common letters, numbers, and symbols can be entered by clicking the calculator buttons but some of the advanced features of the calculator are accessible only by typing commands on the keyboard.

Click the *Backspace* button to delete the character to the left of the current cursor position or press the *Clear* button to delete all information in the calculator. Clicking the *Copy* button copies the last result to the Windows clipboard and clicking the *Run* or '=' button will evaluate the current expression and display the result in the calculator window.

Expressions

For evaluating simple expressions, enter an expression in the calculator with no semi-colon (';') at the end of the line. Note that when entering hexadecimal numbers, place an 'h' after the number (for example '2Fh'). Because the calculator uses C-syntax, make sure to place a '0' before any hexadecimal numbers that begin with a letter (for example, you must use '0FFh' instead of just 'FFh'). Click the *Run* button or press *F8* again to display the results in the calculator. For example:

```
1000h+512*123
```

will display the result '67072 [10600h]'. If a semi-colon is included at the end of the line, the expression is treated as a C program and to display results the *return* keyword must be used. For example:

```
return 0x1000 + 512*123;
```

All standard C operators are supported including +, -, *, /, ~, ^, &, %, |, <<, >>, ?:, brackets, etc. Decimal, hex, octal, and binary number formats are supported. For example:

```
(312 + 013) * (0x1000 | 0b10)
```

See Script Basics and Expressions for more information on expressions.

Variables

Variables can also be declared and used in the calculator using C syntax. For example:

```
int x = 0x4210 + 512;
int y = (x << 16) + x;
return y;
```

A variable declared in the calculator will be displayed in the *Variables* tab of the Inspector. Strings and arrays are also supported. See Data Types for a full list of supported data types.

Functions

010 Editor includes a number of functions for math operations, editing files, editing strings, and interacting with the interface. Most functions are similar to their C counterparts but have a capital first letter. The Printf function is supported and can be used for displaying text in the Output tab of the Output panel. For example:

```
Printf( "Integer result = %d, String result = '%s'\n",
    0x24 << 3, "Test" );
```

See Interface Functions, I/O Functions, String Functions, Math Functions, or Tool Functions for a full list of functions.

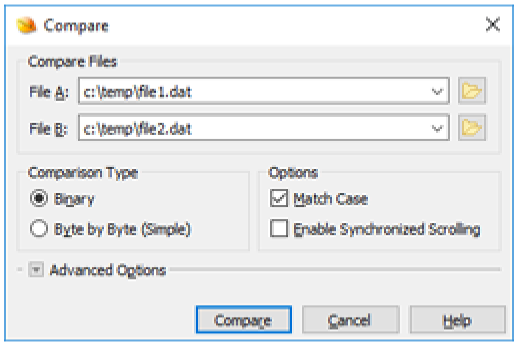
Related Topics:
Data Types
Expressions
Introduction to Templates and Scripts
Interface Functions
I/O Functions
Math Functions
Script Basics
String Functions
Tool Functions

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Comparing Files

The Compare Files tool allows the binary comparison of two files or two blocks of data for byte-by-byte differences. Note that this comparison is different that most text editors which only compare line-by-line. Access the Compare Files tool by clicking the '*Tools > Compare Files...*' menu option.

Compare Dialog



Compare

Compare Files

File A: c:\temp\file1.dat

File B: c:\temp\file2.dat

Comparison Type: [✓ Binary] Byte by Byte (Simple)

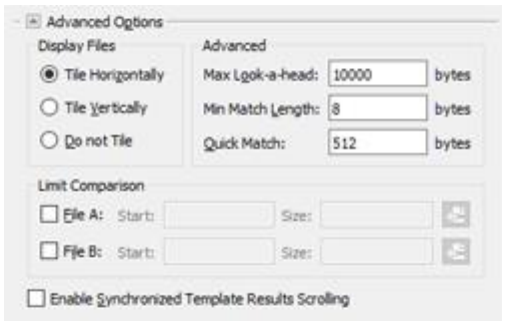
Options: [✓ Match Case] Enable Synchronized Scrolling

Advanced Options

Enter the two files to compare in the *File A* and *File B* fields. Each field contains a drop-down list of all open and recent files. Click the browse button beside either field to use a file dialog box to select a file. Note that if exactly two files are open in the main Tab Group in 010 Editor, the file names for those two files will be automatically entered in these fields.

The Comparison tool supports two different algorithms: *Binary* and *Byte by Byte*. The *Byte by Byte* algorithm compares corresponding bytes between the two files (e.g. each byte at address *n* of file A is compared only against the byte at address *n* of file B) and will usually run quickly. The *Binary* algorithm tries to identify blocks within the files that match. This algorithm is fast when the number of differences is low between the files, but slows down if a number of differences exist (the algorithm is O(d^2) where d is the number of differences). Select which algorithm to use in the *Comparison Type* box.

Two options exist for running comparisons in the *Options* box. If the *Match Case* toggle is enabled, then ASCII strings must match exactly, otherwise strings with a mixture of upper and lowercase letters will match. If the *Enable Synchronized Scrolling* toggle is enabled then after the comparison, scrolling one of the files will cause the other file to scroll as well. Synchronized scrolling can be turned off using the '*Window > Synchronize Scrolling*' menu option (see the Window Menu for more information).



Advanced Options

Display Files: [✓ Tile Horizontally] Tile Vertically, Do not Tile

Advanced

Max Look-a-head: 10000 bytes

Min Match Length: 8 bytes

Quick Match: 512 bytes

Limit Comparison

File A: Start: Size:

File B: Start: Size:

Enable Synchronized Template Results Scrolling

Other advanced options can be viewed by clicking the *Advanced Options* button. The display of the files after the comparison is determined by the *Display Files* box. If the *Tile Horizontal* toggle is set, the two files will be stacked one on top of the other. If the *Tile Vertically* toggle is selected, the files will be stacked side-by-side. The files will not be moved if the *Do Not Tile* toggle is selected. When tiling, the files may be moved from the Floating Tab Group to the main window and if this is done the Floating Tab Group will be hidden.

For the *Binary* algorithm, a limit can be put on the number of bytes the algorithm searches forward by entering a value in the *Max Look-a-head* field. The higher the value, the slower the algorithm may run. The *Min Match Length* field indicates the minimum number of consecutive bytes that must match to display a match in the final output. For higher values, random data is less likely to match, but for lower values, small matches might be ignored. The algorithm contains a heuristic that allows it to quickly accept matches of a certain size. Enter a valid in the *Quick Match* field to indicate the minimum number of bytes that must match for the heuristic. The lower the value, the faster the algorithm will run but the results may be less accurate.

If the *Enable Synchronized Template Results Scrolling* toggle is enabled then after the comparison, when the Template Results panel is scrolled in one file then it will be scrolled in the other file as well. This type of scrolling can be turned off using the '*Window > Synchronize Template Results Scrolling*' menu option and see the Window Menu for more information.

Click the *Compare* button to run the algorithm and display the results in the Output Window. The *Cancel* button will dismiss the dialog when pressed.

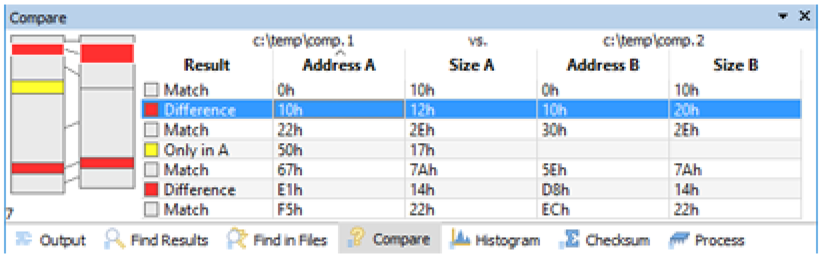
Comparing Blocks of Data

The binary comparison tool can also compare two blocks of data in two different files, or two blocks of data in the same file. Blocks to compare are specified in the *Limit Comparison* box. To compare only part of file A, click the *File A* toggle and enter the starting address of the block and number of bytes in the block in the *Start* and *Size* fields respectively. If the set of bytes to compare has been selected in the file, click the *Get Selection* icon to copy the start and size of the current selection into the proper fields. If the *File A* toggle is disabled, then the comparison will use the entire file A. Similarly, to limit the bytes compared in file B, enable the *File B* toggle and enter values in the *Start* and *Size* fields.

To compare two blocks of data in the same file, set the file name for *File A* and *File B* to be the same file name. Then specify which blocks of data to compare in the *Limit Comparison* box. Clicking the *Compare* button will cause two views of the file to be opened as could be done with the '*Window > Duplicate Window*' menu option.

Output Window

The Output Window will appear after a comparison is run. This window displays a list of all matches and differences, plus a graphical representation of how the files match.



c:\temp\comp.1 vs. c:\temp\comp.2

Result	Address A	Size A	Address B	Size B
☐ Match	0h	10h	0h	10h
🟥 Difference	10h	12h	10h	20h
☐ Match	22h	2Eh	30h	2Eh
🟨 Only in A	50h	17h		
☐ Match	67h	7Ah	5Eh	7Ah
🟥 Difference	E1h	14h	D8h	14h
☐ Match	F5h	22h	ECh	22h

The *Result* field can indicate a *Difference*, a *Match*, or may say *Only In A* or *Only In B* if a block exists only in one of the two files. *Address A* and *Start A* indicate the start and size of the block in the first file, and *Address B* and *Start B* indicate the start and size of the block in the second file. Selecting an entry from the list will highlight the block in both files and also highlight the block in the graph. The display format for each column can be set to hexadecimal or decimal by right-clicking on the Output Window and selecting '*Column Display Format*'. Data can be sorted in the display by clicking on one of the column headers. For example, to see all the matches and differences grouped together, click on the *Result* column header.

The graph contains a representation of both files, side by side. Matched areas are shown in gray, differences are shown as red, and blocks that are only in one file are displayed as yellow. When a range is selected, a white box is drawn around the range in the graph. Below the graph is a number indicating the number of ranges in the comparison.

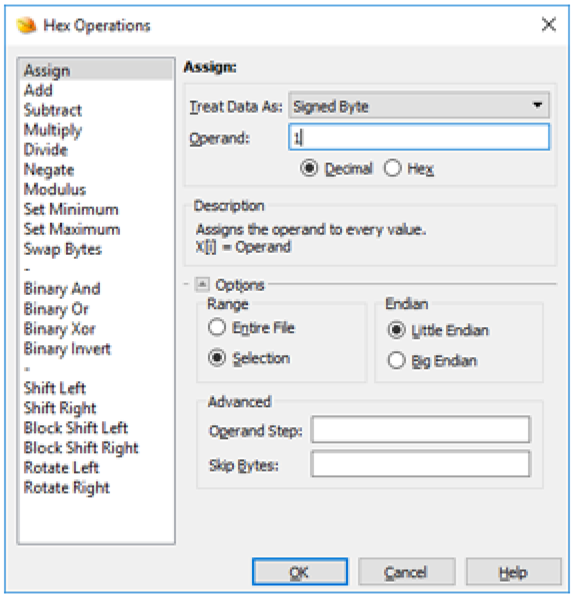
After the comparison is run, the files will be colored according to which bytes match. Bytes that are different will be displayed as light red and bytes that are only in one file will be displayed as light yellow (see Theme/Color Options to change the colors). Right-click the Output Window and select '*Clear*' to clear the results from the comparison. Press the *Esc* key in the Output Window to hide the window.

Related Topics:
Theme/Color Options
Window Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Hex Operations

The Hex Operations tool provides any easy way to apply mathematical operations to a set of bytes. Open the Hex Operations dialog by clicking the '*Tools > Hex Operations*' menu option and selecting an operation from the list.



Assign:

Treat Data As: Signed Byte

Operand: 1

[✓ Decimal] Hex

Description

Assigns the operand to every value.
X[i] = Operand

Options

Range: Entire File [✓ Selection]

Endian: [✓ Little Endian] Big Endian

Advanced

Operand Step:

Skip Bytes:

All hex operations treat the bytes in the file as an array. Select the data type of the array by choosing a type from the *Treat Data As* drop-down list. Enter a number in the *Operand* field using any of the formats described in the Introduction to Number Systems. Note that the operand is assumed to be hex if the *Hex* toggle is set, or decimal if the *Decimal* toggle is set. How the *Operand* is applied to the data is dependant upon which operation is selected. The following list describes each operation in C notation, assuming that X[i] represents each value in the file to be modified.

- **Assign:** X[i] = Operand
- **Add:** X[i] += Operand (this is equivalent to X[i] = X[i] + Operand)
- **Subtract:** X[i] -= Operand (this is equivalent to X[i] = X[i] - Operand)
- **Multiply:** X[i] *= Operand (this is equivalent to X[i] = X[i] * Operand)
- **Divide:** X[i] /= Operand (this is equivalent to X[i] = X[i] / Operand)
- **Negate:** X[i] = -X[i]
- **Modulus:** X[i] = X[i] % Operand (the modulus operator % computes the remainder after dividing X[i] by the Operand)
- **Set Minimum:** Sets a minimum limit for X[i]. If X[i] is less than the Operand, X[i] is set to the Operand.
- **Set Maximum:** Sets a maximum limit for X[i]. If X[i] is greater than the Operand, X[i] is set to the Operand.
- **Swap Bytes:** Swap the bytes of X[i]
- **Binary And:** X[i] &= Operand
- **Binary Or:** X[i] |= Operand
- **Binary Xor:** X[i] ^= Operand
- **Binary Invert:** X[i] = ~X[i]
- **Shift Left:** X[i] <<= Operand
- **Shift Right:** X[i] >>= Operand

- **Block Shift Left:** Similar to Shift Left except data is treated as one long block. Bytes shifted off of X[i+1] will be shifted onto X[i].
- **Block Shift Right:** Similar to Shift Right except data is treated as one long block. Bytes shifted off of X[i] will be shifted onto X[i+1].
- **Rotate Left:** Similar to Shift Left except that bytes shifted off of X[i] will be added to the right side of X[i].
- **Rotate Right:** Similar to Shift Right except that bytes shifted off of X[i] will be added to the left side of X[i].

Note that the *Operand* is not used for some operations and some operations can only be used on certain data types. A description of the selected operation is shown in the *Description* box and further options for the dialog can be controlled by clicking the *Options* button.

If no bytes are selected in the file, the Operation will be applied to the whole file. If a selection is made, the Operation will be applied to the selected bytes if the *Selection* toggle is set, or the whole file if the *Entire File* toggle is set. By default, the data for the operation will be assumed to have the same endian as the file (see Introduction to Byte Ordering for more information). To change the endian of the operation, click the *Little Endian* or the *Big Endian* toggle.

The *Advanced* box contains two fields: *Operand Step* and *Skip Bytes*. If a value is entered in *Operand Step*, that value will be added to the *Operand* after modifying each value in the file. The *Operand Step* can be used to easily perform a number of complex operations, including building arrays. For example, select 256 bytes in a file and perform an Assign operation on the bytes with an *Operand* of '0' and an *Operand Step* of '1'. The result will be an array with the values 0 up to 255.

If a value is entered in the *Skip Bytes* field, that number of bytes will be skipped after each value is modified in the file. This feature can be used to skip over bytes that should not be modified. For example, if a binary file contains a series of employee records containing an integer ID number followed by a 40 character Name, the *Skip Bytes* field can be used to modify the ID number without modifying the Name. Use an Add operation on integers with '1' as the Operand and '40' as the *Skip Bytes* value.

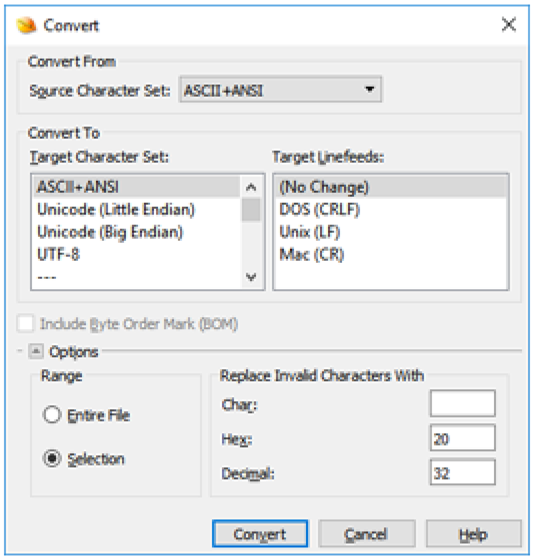
For more complex operations, scripts can be used. See Introduction to Templates and Scripts for more information.

Related Topics:
Introduction to Byte Ordering
Introduction to Number Systems
Introduction to Templates and Scripts

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Converting Files

The conversion tool supplied with 010 Editor can be used to convert bytes from one character set to another and can also transform linefeeds from one type to another. Select '*Tools > Convert*' or press *Ctrl+T* to open the Convert Dialog. The list of available character sets can be controlled with the Character Set Options dialog.



In the Convert dialog choose which character set to convert from using the *Source Character Set* drop-down list. The source character set will automatically be filled from the current File Interface or from the file's character set if '*View > Character Set > Use Default*' is turned off. Next select which character set to convert to using the *Target Character Set* list (by default the target character set is the same as the source). The *Target Character Set* lists common character sets at the top (see the *Show at Top Level* toggle on the Character Set Options dialog), followed by a list of recently used character sets, followed by '---' and then a list of all available character sets.

To not apply any special conversions to the linefeeds of a text file, leave the *Target Linefeeds* list set to '*(No Change)*'. Selecting a linefeed type from the *Target Linefeeds* list will transform *all* linefeeds found in the file to the indicated type. The Convert dialog can be used to modify only the linefeeds by leaving the source and the target character set the same. See Working with File Interfaces for more information on character sets and linefeeds.

Some text files contain a sequence of bytes at the beginning of the file to indicate which character set is used in the file and this is called a *Byte-Order Mark* or *BOM*. Byte-Order Marks are only currently used for Unicode or UTF-8 files. When the Convert dialog is opened for a Unicode or UTF-8 text file that contains a BOM, the *Include Byte Order Mark (BOM)* toggle will be set and the text '*(currently exists)*' will be included after the toggle. When converting a Unicode or UTF-8 file that does not contain a BOM, the *Include Byte Order Mark (BOM)* will be unchecked and the text '*(not present)*' will be shown. When the Target Character Set is Unicode or UTF-8, set the *Include Byte Order Mark (BOM)* toggle to add a BOM to the file or uncheck the toggle to remove the BOM. The toggle will be disabled when the Target Character Set is not Unicode or UTF-8. See Byte-Order Marks for more information on using BOMs with the editor.

If no bytes are selected in the file when the conversion is performed, the conversion will be applied to every byte in the file. If a selection is made the Conversion may be applied to just the selected bytes by clicking the *Options* button and choosing the *Selection* toggle (the default). Alternately, select the *Entire File* toggle to convert the whole file. Sometimes there is no equivalent character when converting data from one character set to another (for example when converting Unicode to ASCII). Any characters that the dialog cannot convert will be assigned the byte value listed in the *Replace Invalid Characters With* box (the default is the space character 0x20).

Clicking *Convert* will perform the conversion or clicking *Cancel* will close the dialog. Double-clicking on an item in the *Target Character Set* or *Target Linefeeds* list will also perform the conversion. Note that if converting a whole text file, 010 Editor may automatically change the current File Interface or may turn off the '*View > Character Set > Use Default*' toggle and set a per-file character set.

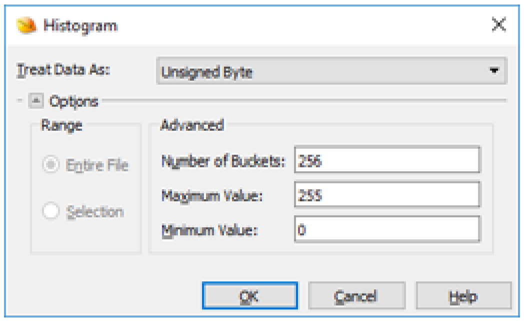
Files can also be converted to other formats using the '*File > Import Hex...*' or '*File > Export Hex...*' tools (see Importing/Exporting Files for more information).

Related Topics:
Character Set Options
Importing/Exporting Files
Introduction to Byte Ordering
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Histograms

A histogram is a graph which indicates how often certain data values occur in a file. Click the '*Tools > Histogram*' menu option to calculate and display the histogram for the current file.


Histogram

Treat Data As: Unsigned Byte

Options

Range
- Entire File
- Selection

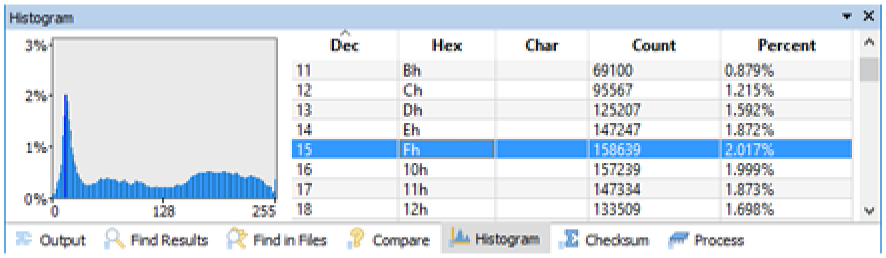
Advanced
- Number of Buckets: 256
- Maximum Value: 255
- Minimum Value: 0

A histogram is calculated by first interpreting the file as an array of data. Select the data type for the array by selecting a value from the *Treat Data As* drop-down list (see Using the Inspector for more information on different data types). Then the histogram generates a number of buckets and places each value into a bucket depending upon its value. For example, the most common histogram treats the data as unsigned bytes and uses 256 buckets. Bytes in the file with value 0 are placed into the first bucket, bytes with value 1 are placed into the second bucket, etc. The graph generated indicates how many values were placed into each bucket.

Click the *Options* button to control the range and bucket configuration for the histogram. By default, 010 Editor generates 256 buckets to place values but the number of buckets can be modified using the *Number of Buckets* field. The minimum and maximum accepted values can be edited by modifying the *Minimum Value* and *Maximum Value* fields respectively. The values that each bucket holds is calculated by dividing the range specified by the minimum and maximum values into *Number of Buckets* equal intervals. Any values in the file outside the minimum and maximum values are ignored in the histogram calculation.

If no selection is made on the file, the Histogram will be run on the entire file. If a selection is made, select the *Selection* toggle to calculate the histogram based only on the selected bytes (the default), or select *Entire File* to calculate the histogram on all bytes in the file.

Output Window



Dec	Hex	Char	Count	Percent
11	Bh		69100	0.879%
12	Ch		95567	1.215%
13	Dh		125207	1.592%
14	Eh		147247	1.872%
15	Fh		158639	2.017%
16	10h		157239	1.999%
17	11h		147334	1.873%
18	12h		133509	1.698%

The result of the histogram will be shown in the Output Window. On the right side of the window is a table of all the different buckets (see above for an explanation of the buckets). The *Dec*, *Hex*, and *Char* fields indicate the values in a particular bucket in different formats. The *Count* field indicates the number times a value was placed into a bucket and the *Percent* field indicates the percentage of all the values that were placed into the bucket.

Note that the *Char* field will display characters using the character set of the current file if the file is using a simple character set; however, if the file is using a more complex multi-byte character set such as UTF-16, UTF-8, or Chinese, this field will just display ASCII values. The table can be sorted by clicking on one of the field headings.

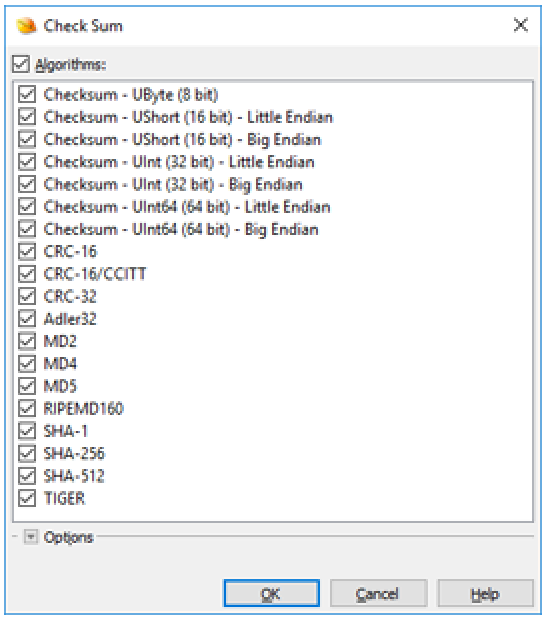
A graph is displayed in the left area with percentage running along the vertical axis and value running along the horizontal axis. A blue bar indicates the percentage of each value. Selecting a bucket in the output table will highlight the corresponding bar as yellow in the graph. To clear the histogram, right click on the Output Window and select '*Clear*'. Press the *Esc* key in the Output Window to hide the window.

Related Topics:
Using the Inspector

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Check Sum/Hash Algorithms

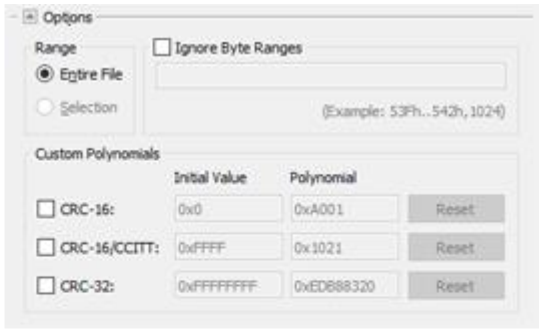
The Check Sum tool can be used to apply a number of Check Sum or Hash Algorithms to the current file. Run the Check Sum tool by clicking the '*Tools > Check Sum...*' menu option or press *Ctrl+K*.


Check Sum
☑ Algorithms:
☑ Checksum - UByte (8 bit)
☑ Checksum - UShort (16 bit) - Little Endian
☑ Checksum - UShort (16 bit) - Big Endian
☑ Checksum - UInt (32 bit) - Little Endian
☑ Checksum - UInt (32 bit) - Big Endian
☑ Checksum - UInt64 (64 bit) - Little Endian
☑ Checksum - UInt64 (64 bit) - Big Endian
☑ CRC-16
☑ CRC-16/CCITT
☑ CRC-32
☑ Adler32
☑ MD2
☑ MD4
☑ MD5
☑ RIPEMD160
☑ SHA-1
☑ SHA-256
☑ SHA-512
☑ TIGER
Options
OK
Cancel
Help

Select which Check Sum algorithms to perform on the file by checking or unchecking the algorithm name in the *Algorithms* list. Note that all algorithms can be selected or deselected at once by clicking the box beside the *Algorithms* label. The following algorithms are supported:

- **Checksum - UByte (8 bit)**
- **Checksum - UShort (16 bit) - Little Endian**
- **Checksum - UShort (16 bit) - Big Endian**
- **Checksum - UInt (32 bit) - Little Endian**
- **Checksum - UInt (32 bit) - Big Endian**
- **Checksum - UInt64 (64 bit) - Little Endian**
- **Checksum - UInt64 (64 bit) - Big Endian**
- **CRC-16**
- **CRC-16/CCITT**
- **CRC-32**
- **Adler32**
- **MD2**
- **MD4**
- **MD5**
- **RIPEMD160**
- **SHA-1**
- **SHA-256**
- **SHA-512**
- **TIGER**

Most checksum algorithms treat the data file as a list of unsigned bytes and then sum the values of those bytes (this type of algorithm can be performed by selecting the *Checkum - UByte* algorithm). However, some checksum algorithms need to treat the data file as a list of unsigned shorts, ints, or int64s (see Using the Inspector for more information on different data types). These type of checksums can be calculated by selecting the *Checksum - UShort*, *Checksum - UInt*, or *Checksum - UInt64* algorithms respectively. Note that there are two versions of these algorithms, one where the file is treated as *Little Endian* and one where the file is treated as *Big Endian* (see Introduction to Byte Ordering for more information on endianness). For unsigned short, int, or int64 checksum algorithms the data will be padded with zeros if the data size is not a multiple of the data type size. The details of the other algorithms are beyond the scope of this help file.


Options

Range: [✓ Entire File] Selection

☐ Ignore Byte Ranges

(Example: 53Fh..542h,1024)

Custom Polynomials

	Initial Value	Polynomial	
☐ CRC-16:	0x0	0xA001	Reset
☐ CRC-16/CCITT:	0xFFFF	0x1021	Reset
☐ CRC-32:	0xFFFFFFFF	0xEDB88320	Reset

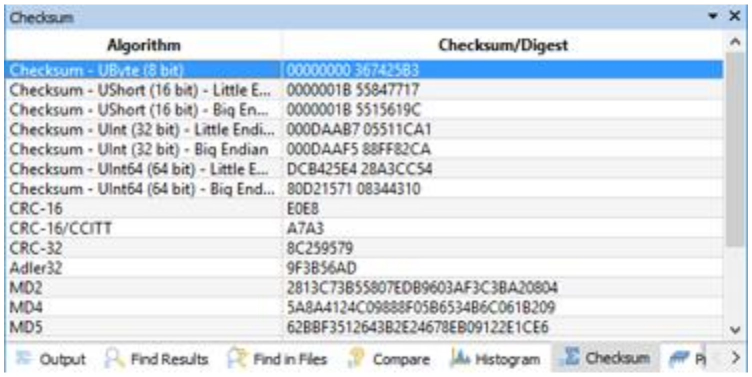
Some advanced options can be controlled by clicking the *Options* button. If no bytes are selected in the file when the Check Sum tool is run, the algorithms will be applied to all bytes in the file. When a selection is made, select the *Selection* toggle in the dialog box to apply the algorithms to only the selected bytes (the default), or the *Entire File* toggle to apply the algorithms to all bytes.

Besides limiting the checksum to the selected bytes, explicitly excluding certain byte ranges is possible by enabling the *Ignore Byte Ranges* toggle and entered ranges in the associated field. Multiple addresses can be indicated using commas, and a range of addresses can be indicated using '..' between two addresses (the range is inclusive). Any of the standard numeric formats can be used in the field. For example, entering the ranges '512..515,0x1000' would ignore the 4 bytes starting at position 512, and the single byte at 0x1000. This feature is useful to calculate the checksums for files where the actual checksum is stored in the file and thus should be excluded from the calculation.

The CRC-16, CRC-16/CCITT and CRC-32 algorithms can be customized by clicking the *Options* button and using the *Custom Polynomials* section. Enable the toggle beside the name of the algorithm to customize, and then enter the starting value for the CRC in the *Initial Value* field and the polynomial for the algorithm in the *Polynomial* field. Clicking the *Reset* button to the right of an algorithm resets all values for that algorithm to their default values. Note that different polynomials exist for some algorithms depending upon how the algorithm is implemented internally and this dialog lists the default polynomials for how 010 Editor has implemented the algorithms.

Click the *OK* button to perform the calculation or click the *Cancel* button to dismiss the dialog.

Output Window



Checksum

Algorithm	Checksum/Digest
Checksum - UByte (8 bit)	00000000 367425B3
Checksum - UShort (16 bit) - Little E...	0000001B 55847717
Checksum - UShort (16 bit) - Big En...	0000001B 5515619C
Checksum - UInt (32 bit) - Little Endi...	000DAAB7 05511CA1
Checksum - UInt (32 bit) - Big Endian	000DAAF5 88FF82CA
Checksum - UInt64 (64 bit) - Little E...	DCB425E4 28A3CC54
Checksum - UInt64 (64 bit) - Big End...	80D21571 08344310
CRC-16	E0E8
CRC-16/CCITT	A7A3
CRC-32	8C259579
Adler32	9F3B56AD
MD2	2813C73B55807EDB9603AF3C3BA20804
MD4	5A8A4124C09888F05B6534B6C061B209
MD5	62BBF3512643B2E24678EB09122E1CE6

The results of the different algorithms are displayed in the Output Window. The algorithm name is listed under the *Algorithm* column and the result is displayed in the *Check Sum/Digest* column. The result can be copied to the clipboard by right-clicking the Output Window and selecting '*Copy*'.

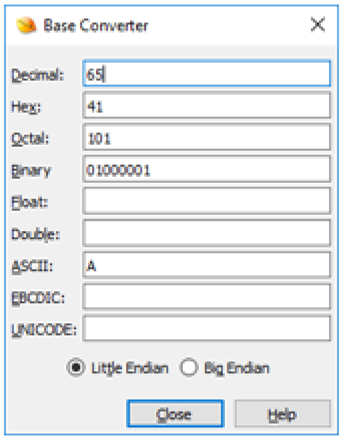
By default, all checksums will be displayed as a 64-bit number in hex notation where the first 32 bits and the last 32 bits are separated by a space (for example, "00000000 00007F80"). The checksums can be configured to only display 32, 16, or 8 bits instead of 64 by right-clicking on the *Checksum/Digest* column and selecting the *Checksum Precision* menu option. Also, the results can be displayed in decimal notation by right-clicking on the *Checksum/Digest* column and selecting the *Column Display Format* menu option. Select '*Clear*' from the right-click menu to clear all of the results. Press the *Esc* key in the Output Window to hide the window.

Related Topics:
Introduction to Byte Ordering
Using the Inspector

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Base Converter

The Base Converter is an easy-to-use tool for converting between decimal, hexadecimal, octal, and binary numeric formats, plus a number of floating point and string formats. Click the '*Tools > Base Converter...*' menu option to display the Base Converter window.



Base Converter

Decimal:	65
Hex:	41
Octal:	101
Binary	01000001
Float:	
Double:	
ASCII:	A
EBCDIC:	
UNICODE:	

[✓ Little Endian] Big Endian

Type a number into either the *Decimal*, *Hex*, *Octal*, or *Binary* field. The number will be converted into the other three formats and displayed in the corresponding fields. If an invalid number is entered, the other fields will be cleared. See the Introduction to Number Systems for more information on numeric formats.

If a number is entered into the *Float* or *Double* fields, the number will be converted from its floating point value to a binary encoding (4 bytes for a Float and 8 bytes for a Double). The binary data will be displayed as a set of 4 or 8 numbers in the *Decimal*, *Hex*, *Octal*, and *Binary* fields. Note that the endianness of the binary data can be controlled with the *Little Endian* and *Big Endian* toggles at the bottom of the dialog (see Introduction to Byte Ordering).

If a string or character is entered in the *ASCII*, *EBCDIC*, or *UNICODE* fields, the string will be converted to a set of binary bytes. These bytes will be displayed as a set of numbers in the *Hex*, *Octal*, and *Binary* fields. Note that UNICODE strings have 2 bytes per character and the endianness of the UNICODE data can be controlled using the *Little Endian* and *Big Endian* toggles at the bottom of the dialog.

The Base Converter dialog can be left open while working in other windows in the editor and the window can be resized horizontally to enlarge or shrink the fields. Click the *Close* button to dismiss the dialog.

Related Topics:
Introduction to Byte Ordering
Introduction to Number Systems

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Introduction to Templates and Scripts

One of the most powerful features of 010 Editor is the ability to run Binary Templates and Scripts. A Binary Template allows a binary file to be *understood* by parsing the file into a hierarchical structure. Templates have a similar syntax to C/C++ structs but they are run as a program. Every time a variable is declared in the Template, the variable is mapped to a set of bytes in the current file. For example, the following is a simple Template:

```
struct FILE {
    struct HEADER {
        char    type[4];
        int     version;
        int     numRecords;
    } header;

    struct RECORD {
        int     employeeId;
        char    name[40];
        float   salary;
    } record[ header.numRecords ];

} file;
```

The variable *type* is mapped to the bytes 0 to 3 in the file, *version* is mapped to the bytes 4 to 7, and *numRecords* is mapped to the bytes 8 to 11. Any time a variable is accessed, its value is read from the file, and any time the variable is assigned, its value is written to the file. These structures are different from regular C since they can contain control statements such as *if*, *for*, or *while*. Templates are executed in a similar fashion to an interpreter, where each line is executing starting from the top of the file.

A Script file also has a similar syntax to C and can be used to edit variables defined in a Template. For example, the Script:

```
int i;
for( i = 0; i < file.header.numRecords; i++ )
    file.record[i].salary *= 2.0;
```

can be used to double every employee's salary using the Template. Scripts can be used with Templates, or on their own to edit files or interact with the 010 Editor program. Scripts can also be used as macros to simplify repetitive tasks.

For an example of using Templates to parse files, open a ZIP, BMP, or BMP file and look at the *Template Results* panel below the Hex Editor Window. For more information see:

- Working with Template Results

Binary Templates are stored as text files with extension ".bt" and Scripts are stored as text files with extension ".1sc". For information on executing Scripts or Templates see:

- Running Templates and Scripts

For an introduction to writing Templates see:

- Template Basics

For information on writing Scripts see:

- Script Basics

Binary Templates and Scripts others have created can easily be downloaded and installed from the 010 Editor Repository. See:

- Introduction to the Repository

To help find and fix errors with Templates and Scripts, 010 Editor includes an advanced debugger. For more information see:

- Using the Debugger

Although Templates are initially compiled, they are executed similar to an interpreter. The execution starts at the first line of Template and continues line by line, obeying any control statements encountered.

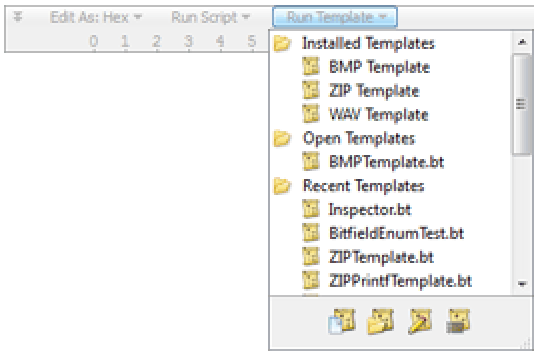
Related Topics:
Introduction to the Repository
Script Basics
Template Basics
Using the Debugger
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Running Templates and Scripts

Running Templates

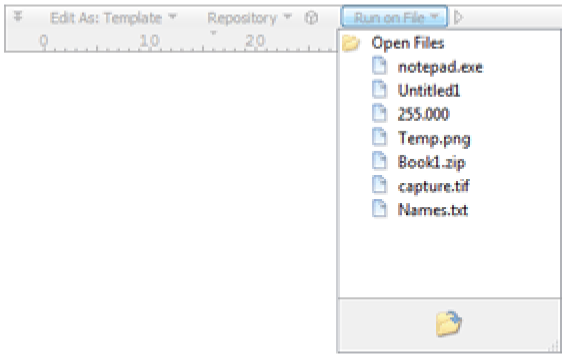
A number of ways exist to run a Binary Template. The easiest is simply to open a file and if 010 Editor has a Binary Template installed for that type of file the Template will be run automatically. 010 Editor comes preinstalled with Binary Templates for BMP, WAV, and ZIP files but other Templates can be installed (see Template Options or the Repository Dialog for more information). Templates can also be run by clicking on the name of a Template in the '*Templates*' menu (Templates can be installed on this menu using the Template Options dialog or the Repository Dialog), or by using the Debug Menu.



Another method of running Binary Templates is to use the File Bar above each editor. When editing a binary data file, the File Bar will contain sections marked *Run Script* and *Run Template* as shown above. Click the *Run Template* area to display a drop-down list of *Installed Templates*, *Open Templates*, and *Recent Templates*. Click on a Template name in the list to execute that Template on the current file. Once a Template has been selected for a file, the *Run Template* area will indicate *Run Template: <template_name>* followed by the icon ▷. Click this icon or press F5 to rerun the template.

At the bottom of the drop-down list are four icons which can be used for creating a new Template, opening a Template, editing the Template associated to this file, or editing the list of Installed Templates, respectively. Note that the drop-down list can be resized by clicking and dragging the handle at the bottom-right corner of the list and 010 Editor will remember the chosen size.

Templates are currently run in a separate thread, meaning the editor can still be used while a Template is running. Only one Template or Script can be run at a time and the current Template or Script must be stopped before another can be run. To stop a running Template click '*Templates > Stop Template*' or press Shift+Esc.



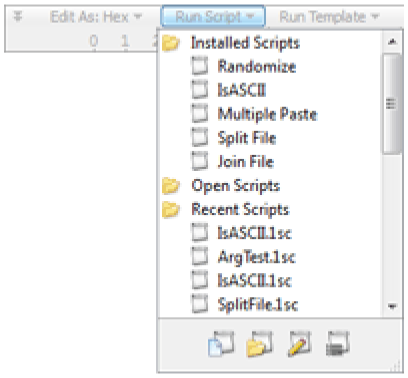
When editing a text file that is not a Template or Script, the File Bar will contain the sections *Run Script* and *Syntax*. Use the *Syntax* section to choose which Binary Template to use for Syntax Highlighting as described in the Using Syntax Highlighting help topic.

When editing a Template (the *Edit As* area will display *Edit As: Template*) the File Bar will now contain the section *Run on File* as shown above. Clicking the *Run on File* area will display a list of all files currently open, but will not include any Scripts or Templates. Click a file in the drop-down list to run the current Template on that file. Once a file has been selected, this area will indicate *Run on File: <file_name>* and clicking the ▷icon or pressing F5 will rerun the current Template. Clicking the Open icon at the bottom of the drop-down list allows opening a file and then immediately running the current Template on that file. The *Repository* section of the File Bar is discussed in the Repository Menu help topic.

If an error occurs while running a Template, an error message will be displayed in the Output tab of the Output Window and a dialog will ask to start the debugger. Double-clicking on an error message in the Output tab will move the cursor to the line where the error occurred. Templates can also be run using the Command Line. Once a Template has been run the Working with Template Results help topic describes how to use the results.

Running Scripts

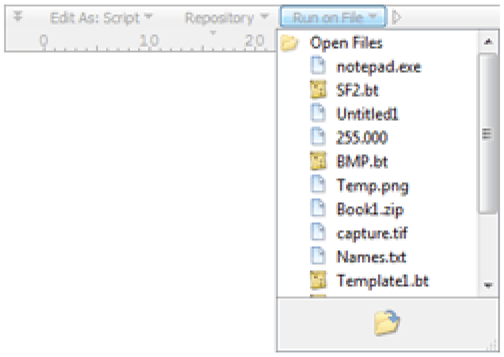
Similar to running Templates, Scripts can be run by clicking on a script name in the '*Scripts*' menu (see the Script Options dialog for information on placing a Script on this menu plus a list of all available scripts). Also, through the same dialog Scripts can be set to run when a certain file type is opened or can be set to run automatically on application startup or shutdown. See the Repository Dialog for information on installing Scripts that other people have submitted to the repository.



Alternately, Scripts can be run using the File Bar at the top of each editor. When editing a file that is not a Script or Template, the File Bar will contain two sections marked *Run Script* and *Run Template*. Click the *Run Script* area and select a script from the list of *Installed Scripts*, *Open Scripts*, or *Recent Scripts* to run that script on the current file. After a Script has been selected for a file, the *Run Script* area will display *Run Script: <script_name>* followed by the icon ▷. Click this icon or press F7 to run the script again.

At the bottom of the drop-down list four icons exist. These icons can be used for creating a new Script, opening a Script, editing the Script associated with this file, or editing the list of Installed Scripts. The drop-down list can also be resized by clicking and dragging the handle at the bottom-right corner of the list.

Scripts are run in a different thread, meaning editing can still be performed while a Script is running. Only one Template or Script can be run at a time and the current Template or Script must be stopped before another can be started. To stop a running Script click '*Scripts > Stop Script*' or press Shift+Esc.



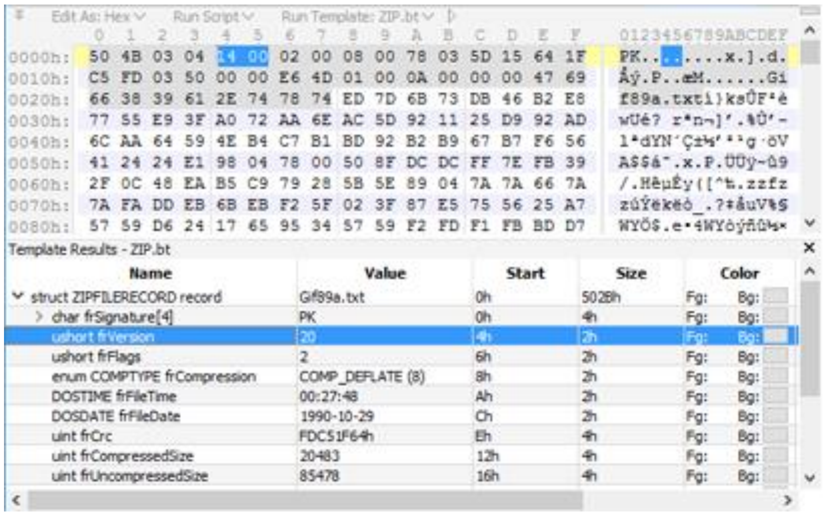
If the current file being edited is a Script (the *Edit As* area will display *Edit As: Script*) the File Bar will now display a section *Run on File*. Click the *Run on File* section and select a file from the drop-down list of all open files to run the current Script on that file. Some Scripts may be run without a target file by clicking the ▷icon or by selecting '*(none)*' from the drop-down list. After a file has been chosen, this area will show *Run on File: <file_name>* and clicking the ▷icon or pressing F7 will rerun the current Script on the selected file. The Open icon at the bottom of the drop-down list can be used to open a file and immediately run the current Script on that file.

If an error occurs while running a Script, an error message will be displayed in the Output tab of the Output Window and a dialog will ask to start the debugger. Double-click an error message to view the line where the error occurred. Scripts can be run using the Command Line as well. After a Script has been run (and assuming the Script is loaded in the interface) select the Script and click on the *Variables* tab of the Inspector to view the variables created by the Script.

Related Topics:
Command Line Parameters
Script Options
Template Basics
Template Options
Using Syntax Highlighting
Using the Debugger
Using the Inspector
Using the Repository Dialog
Using the Repository Menu
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Working with Template Results



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 50 4B 03 04 14 00 02 00 08 00 78 03 5D 15 64 1F   PK........x.].d.
0010h: C5 FD 03 50 00 00 E6 4D 01 00 0A 00 00 00 47 69   Åý.P..æM......Gi
0020h: 66 38 39 61 2E 74 78 74 ED 7D 6B 73 DB 46 B2 E8   f89a.txtí}ksÛF²è
0030h: 77 55 E9 3F A0 72 AA 6E AC 5D 92 11 25 D9 92 AD   wUé? rªn¬]'.%Ù'-
0040h: 6C AA 64 59 4E B4 C7 B1 BD 92 B2 B9 67 B7 F6 56   lªdYN´Ç±½'²¹g·öV
0050h: 41 24 24 E1 98 04 78 00 50 8F DC DC FF 7E FB 39   A$$á˜.x.P.ÜÜÿ~û9
0060h: 2F 0C 48 EA B5 C9 79 28 5B 5E 89 04 7A 7A 66 7A   /.HêµÉy([^‰.zzfz
0070h: 7A FA DD EB 6B EB F2 5F 02 3F 87 E5 75 56 25 A7   zúÝëkëò_.?‡åuV%§
0080h: 57 59 D6 24 17 65 95 34 57 59 F2 FD F1 FB BD D7   WYÖ$.e•4WYòýñû½×
```

Name	Value	Start	Size	Color
struct ZIPFILERECORD record	Gif89a.txt	0h	502Bh	Fg: Bg:
char frSignature[4]	PK	0h	4h	Fg: Bg:
ushort frVersion	20	4h	2h	Fg: Bg:
ushort frFlags	2	6h	2h	Fg: Bg:
enum COMPTYPE frCompression	COMP_DEFLATE (8)	8h	2h	Fg: Bg:
DOSTIME frFileTime	00:27:48	Ah	2h	Fg: Bg:
DOSDATE frFileDate	1990-10-29	Ch	2h	Fg: Bg:
uint frCrc	FDC51F64h	Eh	4h	Fg: Bg:
uint frCompressedSize	20483	12h	4h	Fg: Bg:
uint frUncompressedSize	85478	16h	4h	Fg: Bg:

The *Template Results* panel displays variables that were generated by running a Binary Template on a file (see Introduction to Templates and Scripts) and this panel is shown by default attached to the bottom of the Hex Editor Window (see Using the Hex Editor). To hide the *Template Results* panel click on the 'x' icon to the right of the *Template Results* title or click the '*View > Template Results*' menu option when a file is opened. If the *Template Results* panel is hidden, it can be shown by clicking the '*View > Template Results*' menu option or by clicking the button underneath the vertical scroll bar and dragging upwards (see the diagram at the bottom of this page). The Template Results can be attached to the right side of the Hex Editor Window by right-clicking on the Template Results and choosing the *Template Results Position* menu option. Alternately, the Template variables can be viewed by clicking on the *Variables* tab of the Inspector. To see an example of a Template open any ZIP, BMP, or WAV file on your computer and a Binary Template will automatically be run and the results displayed.

All variables are displayed in a hierarchal list. If a variable in the list has a right arrow or '+' beside it that variable is either a structure or array. Clicking the arrow or plus or double-clicking the variable will show all members within that variable. If a variable has a down arrow or '-' beside it click the icon or double-click the variable to hide all members. A variable can also be opened using the *Ctrl+Right Arrow* key combination or the *Right Arrow* key while the cursor is in the right-most column. A variable can be closed using the *Ctrl+Left Arrow* key combination or by pressing the *Left Arrow* when the cursor is in the left-most column. When a variable is selected, the bytes that correspond to that variable are selected in the file.

To open the entire sub-tree below a variable, right-click on a variable with the mouse and select the *Expand All Children of Node* menu option, or right-click on a variable and select *Expand All Nodes* to open all children of all nodes in the tree (note that the syntax '<*open*=suppress>' can be used after a variable to prevent it from being opened in an Expand All operation). To locate a variable in the list that corresponds to a byte position in the file, position the cursor over a byte in the Hex Editor Window and then use the '*Search > Jump to Template Variable*' menu option to try to locate the variable (see Search Menu). Variables can also be located by string value using the Find Bar.

The Template Results panel displays 6 different columns: The *Name* column lists the data type and the name of the variable, and will also include any array indices. The *Value* column displays the value of the current variable as read from the file. The *Start* column lists the starting address of the variable and the *Size* column lists the size in bytes of the data. The *Color* column lists the foreground color (*Fg:*) and background color (*Bg:*) of the variable (this can be modified by the SetForeColor, SetBackColor, or SetColor functions as listed in Interface Functions). The *Comment* column displays a string that can be set after a variable is declared using the syntax

Copyright © 2003-2019 SweetScape Software

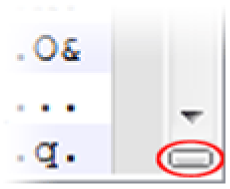
'<*comment*="<string>">' or '<*comment*=<*function_name*>>' (see Declaring Template Variables).

If the value for a variable displayed in the *Value* column is a hex or decimal number, right-click on the variable and select '*Goto > Goto Address <number>*' or '*Goto > Goto Sector <number>*' from the popup menu to jump to that byte address or sector within the file. For example, if a variable has the value '10324', right-click on the variable and choose '*Goto > Goto Address 10324*' to jump to address 10324.

The display format for the different columns can be changed by right-clicking on the window and selecting the '*Column Display Format*' option from the menu. For the *Value* column, if the '*Default*' option is chosen from the '*Column Display Format*' menu, then the display format is determined when running the Template from the functions DisplayFormatHex, DisplayFormatDecimal, DisplayFormatOctal, DisplayFormatBinary, or by using '<format=hex|decimal|octal|binary>'. The '*Column Display Format*' for the first column can be set to '*Type and Name*' to show both the variable type and name, or set to '*Name*' to just display the variable name. Through the right-click menu the address display for the *Start* column can be set to either '*Global*' or '*Local*' using the '*Start Addresses*' menu option. In '*Global*' mode, the addresses are displayed as regular addresses from the beginning of the file but the '*Local*' addresses option displays a variable's address as the offset from its parent variable.

To edit the value of a variable, click the value or press the *Enter* key while a cell in the *Value* column is selected (the current cell is indicated by a dashed box and can be moved using the cursor keys). Modify the value and press *Enter* to commit the change or *Esc* to cancel. Editing is performed similar to the Inspector and note that date formats can be controlled using the Inspector Options dialog. If no value is displayed in the *Value* column, this means that the variable cannot be edited. To clear the template results, right-click the Template Results panel or the *Variables* tab of the Inspector and click the '*Clear*' menu option.

By default, the *Template Results* does not display any local variables (see Declaring Template Variables for more information on local variables). To enable the display of local variables right-click on the panel and select the *Show Local Variables* menu option from the menu. Many of the menu options on the right-click menu are similar to the Inspector. See the Inspector help topic for information on the '*Copy*', '*Copy Row*', '*Copy Column*', '*Copy Table*', and '*Export CSV*' options. Use the '*Export XML*' menu option to export data from the Template Results in XML format (note that XML files are written in UTF-8 format).



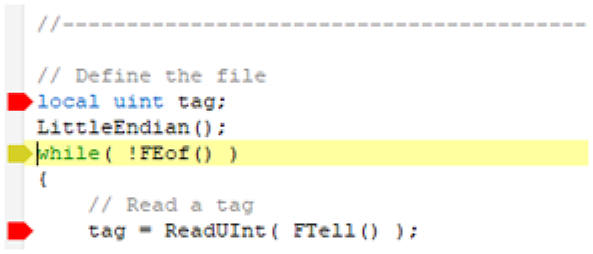
Mouse Over and Hints

After a Template has been run on a file, 010 Editor has an easy way to view the Template variables. Just position the mouse over bytes in the Hex Editor Window for a second and a Hint will popup displaying the value of the variable that uses those bytes. Brackets are also displayed on the Hex Editor Window to indicate which bytes the variable uses. The brackets and hint display can be turned off through the Hex Editor Options dialog.

Related Topics:
Declaring Template Variables
Hex Editor Options
Introduction to Templates and Scripts
Search Menu
Using Find
Using the Inspector
Using the Hex Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using the Debugger



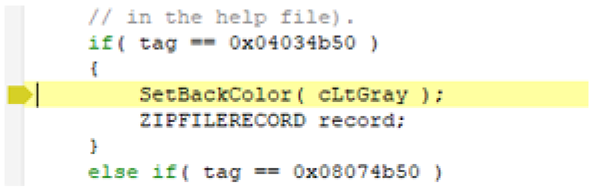
```
//----------------------------------------

// Define the file
local uint tag;
LittleEndian();
while( !FEof() )
{
    // Read a tag
    tag = ReadUInt( FTell() );
```

The *debugger* allows finding and fixing issues with Scripts or Templates written for 010 Editor. Using the debugger, execution of a Script or Template can be stepped line by line and the value of each variable examined after each line. Many of the debugging operations can be controlled using the Debug Menu.

Starting and Stopping the Debugger

By default the debugger is always enabled in 010 Editor but can be turned on or off by clicking the '*Debug > Debugging Enabled*' menu option. If a checkmark is displayed beside the *Debugging Enabled* option then debugging is enabled and if an 'X' is displayed then debugging is disabled. Any time a Script or Template is run (see Running Templates and Scripts) and debugging is enabled, the debugger automatically monitors for breakpoints. If a breakpoint is hit then the program execution pauses at the breakpoint and the debugger is started. Scripts or Templates can also be run by selecting a Script or Template in the editor and clicking the '*Debug > Start Debugging*' menu option. Note this is equivalent to using '*Scripts > Run Script*' when a Script is selected or '*Templates > Run Template*' when a Template is selected. Debugging can only occur on one Script or Template at a time and the current Script or Template must finish before another Script or Template can begin.

Another method of starting the debugger is to select a Script or Template and then click the '*Debug > Step Into*' menu option. This option starts execution of the program but pauses at the first executable line of the program and starts the debugger. Also the debugger can be started by right-clicking on a Script or Template in the Text Editor and choosing *Run to Cursor* from the right-click menu. This option attempts to run the Script or Template until the selected line is reached at which point the execution is paused and the debugger is started.



```
    // in the help file).
    if( tag == 0x04034b50 )
    {
        SetBackColor( cLtGray );
        ZIPFILERECORD record;
    }
    else if( tag == 0x08074b50 )
```

When program execution is paused the current line is marked with a yellow arrow as displayed in the figure above. To continue execution of the program click '*Debug > Continue*', or use '*Scripts > Continue Script or Template*' or '*Templates > Continue Script or Template*'. The debugger can also be stepped to another line as discussed in the Stepping Through Scripts or Templates section below. Program execution can be paused by clicking the '*Debug > Pause*' menu option while a Script or Template is running. Pausing a Script or Template pauses the program, places the cursor at the next line to be executed and starts the debugger.

To stop a Script or Template which is running or paused click the '*Debug > Stop Script/Template*' menu option or press the keyboard shortcut Shift+Esc. Scripts or Templates can also be stopped using '*Scripts > Stop Script*' or '*Templates > Stop Template*' which only appear when a Script or Template is running. Stopping a Script or

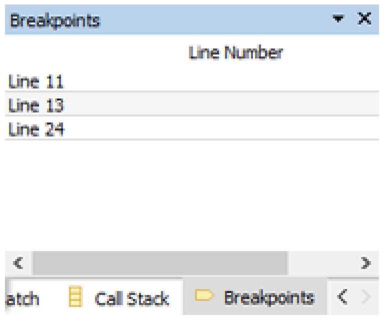
Template which is currently paused at a breakpoint resets the debugger.

Breakpoints

A *breakpoint* is a line in a Script or Template the debugger will pause at when execution reaches that line. Breakpoints can be set by moving the cursor to the requested line in a Script or Template and clicking '*Debug > Toggle Breakpoint*'. Click *Toggle Breakpoint* on a line which already contains a breakpoint to remove that breakpoint. Alternately breakpoints can be set or removed by single-clicking with the mouse in the address column on the left side of each Text Editor. In the address column a number of symbols can appear:

- 🟥- Indicates a breakpoint is set at this line.
- 🟨- Indicates the debugger is currently at this line.
- 🟨- Indicates the debugger is currently at this line and the line also contains a breakpoint.
- ☐- Marks a breakpoint which will not be hit because debugging is turned off. Debugging can be turned on by clicking '*Debug > Debugging Enabled*'.
- ☐- Marks a breakpoint which will not be hit because the Script or Template has been modified since execution began. Use '*Debug > Stop Script*' or '*Debug > Stop Template*' and then rerun the Script or Template to hit the breakpoint.

Hovering the mouse cursor over one of the above symbols in the Text Editor will also give information about the symbol in a hint popup. A list of all set breakpoints for the current Script or Template is available in the *Breakpoints* tab. The *Breakpoints* tab is located in a tab group with the Inspector and the '<' or '>' arrows may be used to locate the tab, or click '*Debug > View Breakpoints*'.



Inside the *Breakpoints* tab is a list of the line numbers for each breakpoint in the file. Right-click on the Breakpoints tab and choose *Add Breakpoint* to set a breakpoint by line number. Select a breakpoint and choose *Remove Breakpoint* to delete the breakpoint from the file. Double-clicking on a breakpoint in the list will jump the mouse cursor to that line in the editor. Note that breakpoints are persistent, meaning they are saved to disk and reloaded when 010 Editor is shut down and restarted (this behaviour can be turned off using the Compiling Options dialog). To delete all breakpoints in all files use the '*Debug > Delete All Breakpoints*' menu option.

Note that breakpoints are currently not hit when 010 Editor is starting up and reloading files that were previously opened. To hit a breakpoint in a Script or Template rerun the Script or Template after 010 Editor has finished starting up. If a breakpoint is set on a line which cannot be executed (e.g. a comment) then the breakpoint will be moved to the next line that can be executed when the Script or Template is run.

Stepping Through Scripts or Templates

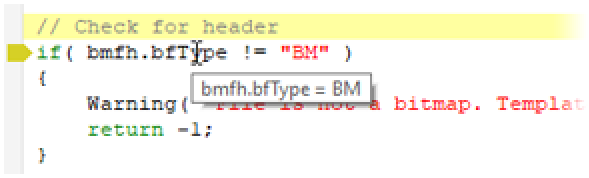
Once the debugger has paused at a line in a Script or Template there are three ways to step to the next line, all of which can be access on the *Debug Menu*:

- **Step Over** - Advances to the next line of the file. If the current line contains a function or struct then

all the statements inside the function or struct are executed without stopping.
- **Step Into** - Advances to the next line of the file. If the current line contains a function or struct the debugger pauses at the first line of the function or struct. This menu option can also be used before a Script or Template has started to start the debugger and pause at the first line that can be executed.
- **Step Out** - If program execution is paused at a line inside a function or struct, all the rest of the lines within the function or struct are executed and the debugger pauses at the first line outside of the function or struct.

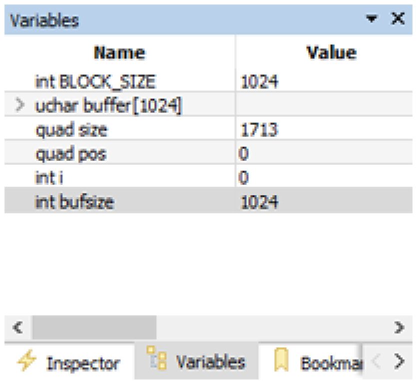
If stepping to a line which is inside an included file, the included file will be opened automatically.

Investigating Variables



```
// Check for header
if( bmfh.bfType != "BM" )
{
    Warning( "File is not a bitmap. Templat
    return -1;
}
```

When program execution has paused at a line a number of ways exist to the check the value of different variables. The first is to place the mouse cursor over a variable name in the Text Editor and the value of the variable will be displayed in a hint popup as shown above. This is called a *Variable Hint* and can be turned off using the Compiling Options dialog. Currently the variables inside a struct cannot be viewed using this technique and to view the contents of a struct use the Quick Watch dialog. Simple expressions can also be evaluated by using a mouse-over. Select the expression to evaluate in the Script or Template and then place the mouse cursor over the selection. If the expression can be evaluated the result is displayed in a hint popup above the selection. Only simple functions such as sizeof, startof, exists, etc. can be evaluated using a hint but if the expression contains complex functions use the Quick Watch dialog.

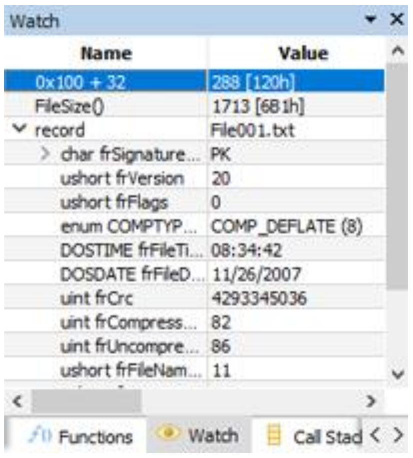


Name	Value
int BLOCK_SIZE	1024
uchar buffer[1024]	
quad size	1713
quad pos	0
int i	0
int bufsize	1024

When debugging a Template the list of created variables can be viewed in either the Template Results or the *Variables* tab in the Inspector as shown in the above figure. When debugging a Script the list of variables can only be viewed in the *Variables* tab. Note that when viewing the created variables for a Template sometimes the local variables are hidden and to display them right-click on the *Template Results* or *Variables* tab and choose *Show Local Variables*.

The final way to view the values of variables is to use watches as detailed in the following sections.

Watches

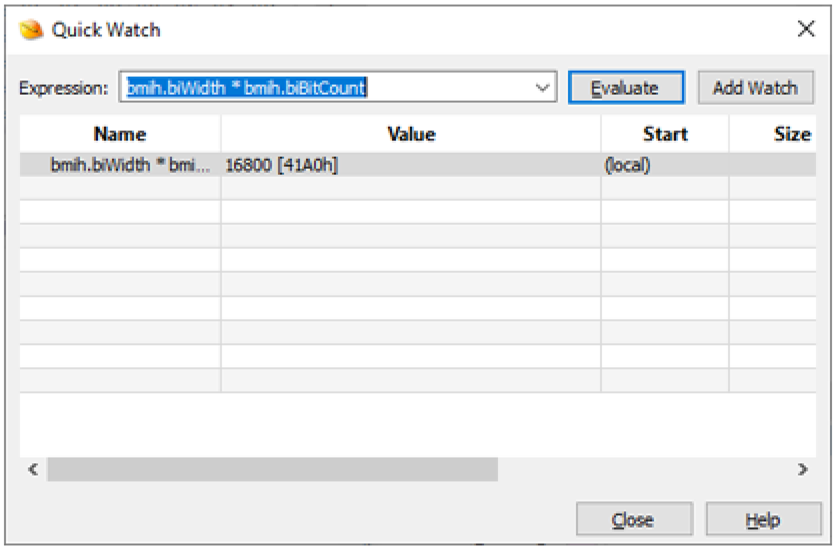


Name	Value
0x100 + 32	288 [120h]
FileSize()	1713 [6B1h]
record	File001.txt
char frSignature...	PK
ushort frVersion	20
ushort frFlags	0
enum COMPTYP...	COMP_DEFLATE (8)
DOSTIME frFileTi...	08:34:42
DOSDATE frFileD...	11/26/2007
uint frCrc	4293345036
uint frCompress...	82
uint frUncompre...	86
ushort frFileNam...	11

A *watch* is an expression which is evaluated every time program execution pauses at a line in a Script or Template. The list of watches is located in the *Watch* tab of the *Inspector* as shown in the above figure. Locate the *Watch* tab by pressing the '<' or '>' arrows in the Inspector or by clicking '*Debug > View Watches*'. New watches can be added to the list by double-clicking on the first empty cell in the *Name* column or by right-clicking on the *Watch* tab and selected *Add Watch*. Existing watches can be deleted by right-clicking on them and selecting *Remove Watch* or by using the Delete key. Note that all Scripts and Templates share a single list of watches.

Watches can contain almost any supported expression including arithmetic operations +,-,*,/, etc., built-in functions, and user-defined functions. If a watch evaluates to a single variable then the value of the variable is displayed in the *Value* column. If a watch evaluates to a struct then the whole struct is displayed and can be browsed similar to the Template Results or the *Variables* tab. Another way to view the value of an expression is to use the Quick Watch dialog.

Quick Watch

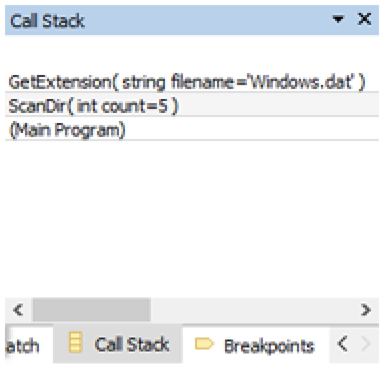


Expression: bmih.biWidth * bmih.biBitCount

Name	Value	Start	Size
bmih.biWidth * bmi...	16800 [41A0h]	(local)	

The *Quick Watch* dialog is used to view the value of almost any expression. When program execution has pause at a line in a Script or Template, the Quick Watch dialog can be opened by clicking the '*Debug > Quick Watch*' menu option. Enter the expression to evaluate in the *Expression* field and click the *Evaluate* button or press the Enter key to evaluate the expression. The result of the expression is displayed in the *Value* column. Note that if a selection is made when the *Quick Watch* dialog is opened then the selection will be copied to the *Expression* field and evaluated immediately. If the result of the expression is a struct then the members of the struct can be browsed as using the Template Results or *Variables* tab. A list of previous expressions can be recalled by clicking the down arrow. Click the *Add Watch* button to add the expression in the *Expression* field to the *Watch* tab as described above. Click the *Close* button to exit the dialog.

Using the Call Stack

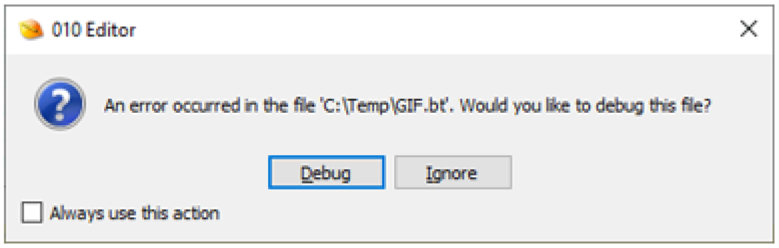


GetExtension(string filename='Windows.dat')
ScanDir(int count=5)
(Main Program)

The *Call Stack* displays the list of functions or structs that have been executed in order to reach the current line in the debugger. The *Call Stack* tab only displays information when program execution has paused at a line in a Script or Template. Access the *Call Stack* tab using the '<' or '>' arrows in the Inspector or by clicking '*Debug > View Call Stack*'. The current function or struct being executed is listed at the top of the Call Stack or '(Main Program)' is listed if no function or struct is being executed. The function or struct which called the function or struct listed on the top line is listed on the next line and so forth. For example in the above figure the main program called the function 'ScanDir' which then called the function 'GetExtension'. The arguments to the functions and their values when called are listed beside the function or struct name. Double-clicking on a function or struct name moves the cursor in the Text Editor to the last line in that function or struct that was executed.

When viewing a variable value using watches or by placing the mouse over variables names in a Script or Template, the results are calculated using the local variables from the function or struct listed at the top of the call stack. To instead use the local variables from a different function or struct, double-click on the function or struct name in the call stack. Double-clicking on a call stack item changes how watches, quick watches, and Variable Hints locate variables. Double-click the top item in the call stack to return the default behaviour.

Debugging Runtime Errors



An error occurred in the file 'C:\Temp\GIF.bt'. Would you like to debug this file?

Always use this action

When an error occurs in a Script or Template that is running, a dialog is displayed asking to start the debugger. Clicking the *Debug* button starts the debugger and places the cursor on the line that caused the error. Note that sometimes errors in Templates can occur because the data file was in a different format than expected or the data file contained invalid data. When the debugger is active, the values of variables can be queried, watches can be evaluated and the call stack checked. Stepping to the next line or continuing execution causes the Script or Template to stop. Clicking *Ignore* causes the Script to Template to stop without starting the debugger. If the error occurred in a Script or Template that is not loaded in the editor, the dialog will also have an option *Load* to open the file and view the line that caused the error. Enable the *Always use this action* toggle before clicking *Debug* or *Ignore* to always do the requested action when an error occurs. The action to perform can also be controlled using the Compiling Options dialog.

Debugging Read/Write Functions

Some Templates contain custom variables which can have special read, write, name or comment functions. These functions are called whenever data needs to be displayed in the Template Results panel or the Variables tab. When the Template Results or Variables tab call these functions, they are run in a separate thread meaning that any breakpoints inside the read/write/name/comment functions will not be hit; however, if these functions are called directly inside the Script or Template then breakpoints will be hit. If debugging needs to be done on these custom functions then call them directly at the end of the Script or Template.

Debugging On-Demand Structures

On-Demand Structures are special struct which are not created until they are opened in order to save memory and time. An On-Demand Structure which is opened by using the Template Results or Variables tab is created in a separate thread, meaning that any breakpoints inside the structure will not be hit; however, if a Template accesses a variable inside an On-Demand Structure then it is created directly in the Template and breakpoints will be hit inside the structure. To debug On-Demand Structures access a variable inside the On-Demand Structure at the end of the Template.

Debugging Highlighting Functions

Some Templates contain Syntax Highlighters which are special functions used to apply coloring to text or hex files. Currently the HighlightLineRealtime and HighlightBytesRealtime functions are called in a separate thread, meaning that breakpoints inside those functions will not be hit. To debug the HighlightLineRealtime function, call the function directly inside the Template. For example to highlight the first 5 lines the following code could be used:

```
#define MAX_LEN 1000
local char    str[MAX_LEN];
local int     line, count, foreColors[MAX_LEN], backColors[MAX_LEN];
local ushort  flags;
local wchar_t text[MAX_LEN];
for( line = 0; line < 5; line++ )
 {
     count = TextReadLine( str, line, MAX_LEN, false );
     text = StringToWString( str );
     HighlightLineRealtime( line, text, foreColors, backColors,
        count, flags );
 }
```

Related Topics:
Compiling Options
Custom Variables
Debug Menu
On-Demand Structures
Running Templates and Scripts
Using Syntax Highlighting

Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Script Basics

010 Editor has a powerful scripting engine that allows many tasks to be automated. Script files have the extension '.1sc' and are very similar in syntax to C. All scripts are executed similar to how an interpreter would run the file, starting at the first line of the program and progressing downwards (there is no need to write a 'main' function as in ANSI C). Scripts can be used to perform editing operations on files, manipulate files on disk, or even perform complex operations like file comparisons, checksums, and find in files. To open or run a script see the Scripts Menu or Running Templates and Scripts. A repository of scripts can be accessed by clicking the '*Scripts > Script Repository*' menu option (see the Repository Dialog). Scripts can be configured to run on startup, shutdown, or when files are opened, and scripts can also be added to the '*Scripts*' menu (see Script Options).

View the following topics for more information on the syntax used when writing Scripts:

- Expressions
- Declaring Script Variables
- Data Types, Typedefs, and Enums
- Arrays and Strings
- Control Statements
- Functions
- Special Keywords
- Preprocessor
- Includes
- Script Limitations

A large number of functions are available when writing Scripts. The available functions are described in:

- Interface Functions
- I/O Functions
- String Functions
- Math Functions
- Tool Functions

Binary Templates have a syntax similar to scripts, but allow a file to be parsed into a number of variables. Scripts can be used to modify the variables that are defined in Templates. See Template Basics for an introduction to using Templates.

Related Topics:
Arrays and Strings
Control Statements
Data Types, Typedefs, and Enums
Declaring Script Variables
Expressions
Functions
Includes
Interface Functions
I/O Functions
Math Functions
Preprocessor
Script Limitations
Special Keywords
String Functions
Template Basics
Tool Functions

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Expressions

Expressions in Scripts or Templates can contain any of the standard C operators:

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- ~ (binary invert)
- ^ (binary xor)
- & (binary and)
- | (binary or)
- % (modulus)
- ++ (increment)
- -- (decrement)
- ?: (ternary)
- << (shift left)
- >> (shift right)

Brackets '(' or ')' can be used to group expressions. For example:

```
(45 + 123) * (456 ^ 16)
```

is a valid expression. The following comparison operators can be used:

- < (less than)
- > (greater than)
- <= (less than or equal)
- >= (greater than or equal)
- == (equal)
- != (not equal)
- ! (not)

For example,

```
(45 > 32)
```

would return the value 1. Any of the assignment operators +=, -=, *=, /=, &=, ^=, %=, |=, <<=, or >>= can also be used. A number of addition Special Keywords including *sizeof* can be used in expressions.

Boolean Operators

The following boolean operators can be used in expressions:

- && (AND)
- || (OR)

- ! (NOT - takes one operator only)

For example, to perform an operation if A and B are true or if C is not true, use:

```
    if( (A && B) || !C ) ...
```

Brackets can be used to indicate which order the operations should be performed.

Numbers

Numbers may be entered in a number of different formats (see Introduction to Number Systems):

- **Decimal -** 456
- **Hexadecimal -** 0xff, 25h, 0EFh
- **Octal -** 013 (with a zero before any numbers)
- **Binary -** 0b011

The 'u' character can be used after a number to indicate an unsigned value (e.g. '12u'), or 'L' can be used to indicate an 8-byte int64 value (e.g. '-1L'). Floating-point numbers may contain 'e' for an exponent (e.g. 1e10). A floating-point number is automatically assumed to be an 8-byte double unless an 'f' character is located after the name (e.g. 2.0f), in which case the number is assumed to be a 4-byte float.

Related Topics:
Introduction to Number Systems
Script Basics
Special Keywords

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Declaring Script Variables

A variable can be declared using the syntax '<data type> <variable name>;' Use an '=' to assign a value to the variable. Multiple variables can also be declared using the ',' operator. For example:

```
int x;
float a = 3.5f;
unsigned int myVar1, myVar2;
```

For a complete list of types allowed, see Data Types. Any variables declared in a Script will be displayed in the *Variables* tab of the Inspector. Arrays of variables can also be declared (see Arrays and Strings). Declaring variables is the main way of building Templates (see Declaring Template Variables for more information).

Constants

Constants can be declared using the keyword '*const*' before a variable declaration. For example:

```
const int TAG_EOF = 0x3545;
```

This syntax is generally better for defining constants than using the '*#define*' preprocessor directive. A number of constants are built into 010 Editor, including *true*, *false*, *TRUE*, *FALSE*, *M_PI*, and *PI*. See Interface Functions for a list of other constants used for coloring or Tool Functions for a list of constants used when calling tool functions.

Related Topics:
Arrays and Strings
Data Types, Typedefs, and Enums
Declaring Template Variables
Interface Functions
Script Basics
Script Limitations
Tool Functions
Using the Inspector

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Data Types, Typedefs, and Enums

Support for a number of different data types is built into 010 Editor. These data types are used when writing a Template (see Declaring Template Variables) or when declaring variables in a Script (see Declaring Script Variables). Commonly, a number of different names refer to the same data type (for example, '*ushort*' and '*WORD*' usually refer to a 16-bit unsigned integer). The following lists each of the data types and all of the names currently supported for that type:

- **8-Bit Signed Integer -** char, byte, CHAR, BYTE
- **8-Bit Unsigned Integer -** uchar, ubyte, UCHAR, UBYTE
- **16-Bit Signed Integer -** short, int16, SHORT, INT16
- **16-Bit Unsigned Integer -** ushort, uint16, USHORT, UINT16, WORD
- **32-Bit Signed Integer -** int, int32, long, INT, INT32, LONG
- **32-Bit Unsigned Integer -** uint, uint32, ulong, UINT, UINT32, ULONG, DWORD
- **64-Bit Signed Integer -** int64, quad, QUAD, INT64, __int64
- **64-Bit Unsigned Integer -** uint64, uquad, UQUAD, UINT64, QWORD, __uint64
- **32-Bit Floating Point Number -** float, FLOAT
- **64-Bit Floating Point Number -** double, DOUBLE
- **16-Bit Floating Point Number -** hfloat, HFLOAT
- **Date Types -** DOSDATE, DOSTIME, FILETIME, OLETIME, time_t, time64_t (for more information on date types see Using the Inspector)

Note that date types can be used in Templates, but they must be cast to an int or float before any operations can be performed on them. Default date and time formats can be set using the Inspector Options dialog. 010 Editor also has support for a special string type.

Typedefs

Other data types can be created using the '*typedef*' keyword. The syntax for creating new types is '*typedef* <data_type> <new_type_name>'. For example,

```
typedef unsigned int myInt;
```

would generate a new data type *myInt* for unsigned integers. Typedefs can also be used with arrays (see Arrays and Strings) using the syntax 'typedef <data_type> <new_type_name> [<array_size>]'. Note that the array size must be a constant in this case. For example, to generate a new string type with 15 characters use:

```
typedef char myString[15];
myString s = "Test";
```

Note that typedefs cannot be used to create multi-dimensional arrays (see Template Limitations). Typedefs can also be used with structs (see Structs and Unions).

Enums

Use the enum keyword to specify a number of constants for a variable. An enum data type can be created using the C syntax '*enum* <type_name> { <constant_name> [= expression], ... } <variable_list>'. If no expression is given for the first constant, it is assumed to be zero. If no expression is given for any other constant, its value is assumed to be the previous constant plus one. For example,

```
    enum MYENUM { COMP_1, COMP_2 = 5, COMP_3 } var1;
```

would declare the constants COMP_1 equal to 0, COMP_2 equal to 5, and COMP_3 equal to 6. By default, enums are the same type as an integer, but the type can be changed by placing '<' type_name '>' after the enum keyword. For example,

```
    enum <ushort> MYENUM { COMP_1, COMP_2 = 5, COMP_3 } var1;
```

would declare the same variable but as an unsigned short. Enums can be useful when writing Templates (see Declaring Template Variables).

Related Topics:
Arrays and Strings
Declaring Script Variables
Declaring Template Variables
Structs and Unions
Using the Inspector
Template Limitations

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Arrays and Strings

Arrays

An array of variables can be defined using the syntax '<data type> <variable name> [<expression>]'. Any of the types in the Data Types list can be used. For example:

```
int myArray[15];
```

Note that unlike ANSI C, the size of the array can be *any* expression including variables, functions, or operators. For example

```
int myArray[ FileSize() - myInt * 0x10 + (17 << 5) ];
```

The individual elements of the array can be accessed using the '[ ]' operator. For example:

```
for( i = 0; i < 15; i++ )
    myArray[i] = i;
```

If an array is declared with size zero a warning will be printed out and no variable will be created, but no error will be generated.

Strings

An array of characters is treated as a special string type. The keyword '*string*' can also be used to declare a string. The operators '=', '+', '+=', and comparison operators can be used on strings as if they were a separate data type. For example:

```
char str[15] = "First";
string s = "Second";
string r1 = str + s;
string r2 = str;
r2 += s;
return (r1 == r2);
```

Strings will automatically resize if assigned too many characters, and a warning will be displayed in the Output text area. All strings are assumed to be null-terminated. For a list of functions that can be used when working with strings, see String Functions.

Wide Strings (Unicode Strings)

Regular strings above assume each character can be stored in 8-bits; however this character size is not appropriate for many languages so 010 Editor also supports wide strings (also called Unicode strings) where each character is a 16-bit unsigned short. Use the special '*wstring*' type to define a wide string and each character of a wstring is assumed to be of type '*wchar_t*' (a wchar_t is equivalent to an unsigned short).

The same operators '=', '+' and '+=' are supported for wstrings as for strings and a wide string constant can be declared by placing a 'L' character before a string or character constant. For example:

```
    wchar_t str1[15] = L"How now";
    wstring str2 = "brown cow";
    wstring str3 = str1 + L' ' + str2 + L'?';
```

Extended characters can be placed in string constants using the UTF-8 character encoding. Wide strings are assumed to be null-terminated and a list of functions available for working with wide strings is available in the String Functions help topic. Wide strings can be converted to regular strings using the WStringToString or StringToWString functions, or by casting.

Related Topics:
Data Types, Typedefs, and Enums
String Functions

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Control Statements

if Statements

Statements may be grouped by '{' or '}' and may contain the regular C *if* statement, or the *if-else* statement in the form '*if*(<condition>) *then* <statement> [*else* <statement>]'. For example:

```
if( x < 5 )
    x = 0;
```

or

```
if( y > x )
    max = y;
else
{
    max = x;
    y = 0;
}
```

for Statements

The standard C *for* statement can be used in any Script or Template in the form '*for*(<initialization>; <condition>; <increment>) <statement>'. For example:

```
for( i = 0, x = 0; i < 15; i++ )
{
    x += i;
}
```

while Statements

The *while* and *do-while* statements can also be used in the form '*while*(<condition>) <statement>' or '*do* <statement> *while*(<condition>)'. For example:

```
while( myVar < 15 )
{
    x *= myVar;
    myVar += 2;
}
```

or

```
do
{
    x *= myVar;
    myVar += 2;
}
while( myVar < 23 );
```

switch statements

A *switch* statement can be used to compare a variable with a number of values and perform different operations depending on the result. *switch* statements are of the form:

```
    switch( <variable> )
    {
      case <expression>: <statement>; [break;]
        .
        .
        .
      default : <statement>;
    }
```

For example:

```
    switch( value )
    {
        case 2  : result = 1; break;
        case 4  : result = 2; break;
        case 8  : result = 3; break;
        case 16 : result = 4; break;
        default : result = -1;
    }
```

break and continue

break or *continue* can be used in a Script or Template using the syntax '*break*;' or '*continue*;'. Use *break* to exit out of the current *for*, *switch*, *while*, or *do* block and transfer program control to the first statement after the block. *break* can also be used to break out of structs when writing Templates. Use *continue* to jump to the end brace of any *for* or *while* loop and continue execution of the loop.

return

At any point during program execution the statement '*return* <expression>;' can be used to stop execution. The returned value will be displayed in the Output tab of the Output window which can be displayed by pressing *Alt+3*. For example:

```
    return 45 + 0x10;
```

would display 61 in the Output area. *return* is also used to return a value when defining custom functions (see Functions).

Related Topics:
Script Basics

Functions

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Functions

A large number of functions are built into 010 Editor. Many of the standard C functions are available, but have a uppercase first letter to differentiate them. Functions are called using the typical C syntax '<function name> (<argument_list>)'. For example:

```
string str = "Apple";
return Strlen( str );
```

would return 5. Some functions can have a variable number of arguments. For example:

```
Printf( "string='%s' length='%d'\n", str, Strlen( str ) );
```

would display "string='Apple' length='5'" in the Output tab of the Output Window. See Interface Functions, I/O Functions, String Functions, Math Functions, Tool Functions for a list of all functions.

Custom Functions

Custom functions can be defined with the regular C syntax '<return type> <function name> (<argument_list>) { <statements> }'. The return type can be *void* or any of the supported data types. For example:

```
void OutputInt( int d )
{
    Printf( "%d\n", d );
}
OutputInt( 5 );
```

Arguments are usually passed by value, but can be passed by reference using the '&' character before the argument name. Array arguments can be indicated using the characters '[]' after the argument name. Array arguments are passed by reference if possible, or by value if not. For example:

```
char[] GetExtension( char filename[], int &extLength )
{
    int pos = Strchr( filename, '.' );
    if( pos == -1 )
    {
        extLength = 0;
        return "";
    }
    else
    {
        extLength = Strlen( filename ) - pos - 1;
        return SubStr( filename, pos + 1 );
    }
}
```

Prototypes can also be used on functions by replacing the statements with a semi-colon ';' after the argument list. Full recursion is supported on custom functions. Note that unlike regular C, the function 'main' is not required and execution begins from the first line of the Script or Template. Note that if a Script is run on a file after a Template has been run on that same file, the functions in the Template can be called from the Script.

Calling External Functions

Functions can also be called which reside in an external library (for example, a Windows DLL). For more information see the External (DLL) Functions in Scripts help topic.

Related Topics:
Data Types, Typedefs, and Enums
External (DLL) Functions in Scripts
Interface Functions
I/O Functions
Math Functions
String Functions
Tool Functions

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Special Keywords

sizeof

The *sizeof* operator can be used to calculate the size in bytes of one of the basic types or a variable that has been declared. For example,

```
    sizeof(double)
```

would return 8. The *sizeof* operator can also compute the size of simple structs or unions (see Structs and Unions). A simple struct is one that does not contain *if* statements or other statements that may change its size when declared. Attempting to compute the size of a non-simple struct or union will generate an error.

startof

The special keyword *startof* can be used on a variable that has been declared in a Template to calculate the starting address of the bytes the variable is mapped to in the file. For example, after opening a BMP file, use the following command in a script to position the cursor at the beginning of the first line of bitmap data:

```
    SetCursorPos( startof( lines[0] ) );
```

exists

The special *exists* operator can be used to determine if a variable has been declared. The operator will return 1 if the given variable exists, or 0 if it does not. For example, after opening a ZIP file, the following command in a script will output all file names:

```
    int i;
    string s;
    while( exists( file[i] ) )
    {
        s = file[i].frFileName;
        Printf( "%s\n", s );
        i++;
    }
```

function_exists

The *function_exists* operator can be used to test if a particular function is defined either as a user-defined function or a built-in function (see Functions for more information). The operator returns 1 if the function exists or 0 if it does not. For example:

```
    if( function_exists(CopyStringToClipboard) )
    {
```

Copyright © 2003-2019 SweetScape Software

```
        ...
    }
```

Requires 010 Editor v3.1 or higher.

this

The *this* keyword can be used to access the variable representing the current structure being defined (see Structs and Unions). For example:

```
    void PrintHeader( struct HEADER &h )
    {
        Printf( "ID1 = %d\n", h.ID1 );
        Printf( "ID2 = %d\n", h.ID2 );
    }

    struct HEADER
    {
        int ID1;
        int ID2;
        PrintHeader( this );
    } h1;
```

If no current structure is currently being defined, *this* is NULL.

Requires 010 Editor v3.1 or higher.

parentof

The *parentof* keyword can be used to access the struct or union that contains a given variable. For example:

```
    void PrintHeader( struct HEADER &h )
    {
        Printf( "ID1 = %d\n", h.ID1 );
        Printf( "ID2 = %d\n", h.ID2 );
    }

    struct HEADER
    {
        int ID1;
        int ID2;

        struct SUBITEM
        {
            int data1;
            int data2;
            PrintHeader( parentof(this) );
        } item1;

        PrintHeader( parentof(item1) );
    } h1;
```

If the given variable is not inside a struct or union an error will be generated.

Requires 010 Editor v3.1 or higher.

Related Topics:
Data Types, Typedefs, and Enums
Functions
Structs and Unions

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Preprocessor

When a Script or Template is executed, a special *preprocessor* stage is run before the main compilation of the file begins. In the preprocessor stage, the software finds any preprocessor directives (e.g. #include, #ifdef, #define, etc.) and uses them to modify the text of the original source code. Each preprocessor directive must start with a '#' character and the '#' character must be the first non-whitespace character on a line.

Defines

Special preprocessor constants can be defined using the syntax '*#define* <constant_name> [<text_value>]'. For example:

```
#define PI 3.14159265
```

Note that a semi-colon is not required after the statement. When any occurrence of the defined constant name is encountered in the rest of the source code (except inside of a string), it will be textually replaced with the defined value of the constant. For example:

```
Printf( "ToRadians=%lf\n", 90.0 * (PI/180.0) );
```

When defining a constant, a value is not required after the constant name in which case a constant is still defined but it will have an empty value. The constant value can be any text string and multi-line values can be defined by placing a '\' character as the last character of a value line. For example:

```
#define CHECK_VALUE if( value > 5) { \
    Printf( "Invalid value %d\n", value ); \
    Exit(-1); }

int value = 4;
CHECK_VALUE;
value = 10;
CHECK_VALUE;
```

Constants created with *#define* can also include other constants that have previously been created with *#define*. For example:

```
#define FILE_ICON   12
#define FOLDER_ICON (FILE_ICON+100)
```

Any constants that are defined can be undefined later using the syntax '*#undef* <constant_name>'. Note that some preprocessors support macros with #define statements but this is not currently supported in 010 Editor.

Built-in Constants

The following constants are defined automatically in 010 Editor depending upon which version of 010 Editor is being run:

- **_010EDITOR** - always defined when running 010 Editor.
- **_010_WIN** - defined if running the Windows version of 010 Editor.
- **_010_MAC** - defined if running the Macintosh version of 010 Editor.
- **_010_LINUX** - defined if running a Linux version of 010 Editor.
- **_010_64BIT** - defined if 010 Editor is being run in 64-bit mode.

Conditional Compilation

The preprocessor directives *#ifdef* and *#ifndef* can be used to compile or ignore whole sections of source code depending on if certain constants were defined with the *#define* directive above. The syntax for these commands is:

```
#ifdef | #ifndef <constant_name>
   (...)
[ #else ]
   (...)
#endif
```

A common usage of this syntax is to place code such as:

```
#ifndef CONSTANTS_H
#define CONSTANTS_H
```

at the beginning of a header file (e.g. constants.h) and then an '#endif' statement at the end of the header file. Then, if this header file is included twice into the source code with *#include* (see below), the code inside the header file will only be compiled once (the second time the file is included the constant CONSTANTS_H is already defined so the #ifndef statement skips the rest of the code). Note that multiple #ifdef or #ifndef statements can be nested inside of each other, but make sure the #endif statements properly line up with the #ifdef/#ifndef statements.

Warnings and Errors

The preprocessor directive *#warning* can be used to output a message to the Output tab of the Output Window during compilation with the syntax '*#warning* "<message>"'. For example:

```
#ifdef NUMBITS
    value = value + NUMBITS;
#else
#warning "NUMBITS not defined!"
#endif
```

The *#error* directive is similar to the *#warning* directive except compilation will stop once an *#error* directive is reached. For example:

```
#ifndef CURRENT_OS
#error  "CURRENT_OS must be defined. Compilation stopped."
#endif
```

Includes

The *#include* directive is supported to insert additional text files into the current file. This directive is discussed in the separate Includes help topic.

External Functions

The *#link/#endlink* directive can be used to call functions inside an external DLL. See the External (DLL) Functions in Scripts help topic for more information.

Related Topics:
Compiling Options
External (DLL) Functions in Scripts
Includes
Script Basics

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Includes

Additional source code files can be inserted into the current script or template using the syntax '#include "filename"'. For example:

```
#include "Vector.bt"
```

Note that a semi-colon is not required after the statement. The angle brackets '<' and '>' can be used around the file name instead of quotes (as in regular C), but the semantics are the same. The indicated file will be inserted into the script or template and the code will be executed as if it were one long file.

When a file is included, it will be searched for in the following directories in the following order:

- **Current Working Directory** - If 010 Editor was started from the command line, the current working directory would be the directory where 010 Editor was started from, else the current working directory is usually the directory where 010 Editor was installed ('C:\Program Files\010 Editor\' by default).
- **File Directory** - The directory where the file that contains the include statement currently resides.
- **Template Directory** - The default Template directory specified using the Directory Options.
- **Script Directory** - The default Script directory specified using the Directory Options.
- **Template Repository Directory** - The directory where Templates from the Repository are installed (see Directory Options).
- **Script Repository Directory** - The directory where Scripts from the Repository are installed (see Directory Options).
- **Additional Include Directories** - Any additional directories specified using the Compiling Options dialog.

Include files can be nested as in regular C. Note that during execution, if an error occurs inside an include file, you will be asked if you want to load the include file in the interface.

Related Topics:
Compiling Options
Directory Options
Preprocessor
Script Basics

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

External (DLL) Functions in Scripts

A Script or Template can execute a function that is located inside an external dynamic linked library. Supported libraries include DLLs on Windows (*.dll), Shared Objects on Linux (*.so), and DYLIB libraries on macOS (*.dylib). Note that the 32-bit version of 010 Editor must be used when calling functions in a 32-bit library and the 64-bit version of 010 Editor but be used when calling functions in an 64-bit library.

Linking to Functions

In a Script or Template, to declare a function that exists inside an external DLL enclose the function prototypes inside a '*#link "<filename>"*' statement and an '*#endlink*' statement. For example:

```
#link "TestDLL.dll"
int  MyDLLFunc1( int a, int b );
int  MyDLLFunc2();
#endlink
```

The declared functions should not have a body (i.e. only a semicolon should be placed after the function prototype). Any other code can be placed inside a #link, #endlink section but limiting the code in this section to function prototypes is recommended. When a library file name is specified with a #link statement, 010 Editor looks for the library in a number of different directories in the same order that is used for Includes. Library file names can be given with no extension, in which case they will automatically be given the extension .dll on Windows, .so on Linux, or .dylib on macOS. If the library could not be found or the function could not be found inside the library then an error is generated and execution of the Script or Template is stopped.

Writing External Functions

When creating a dynamic link library using C++, note that *extern "C"* should be used before the function definition to prevent name mangling. Also certain Window compilers may require *__declspec(dllexport)* before a function to allow that function to be called by external programs. For example to define a simple function in a C++ file the following could be used:

```
#define DECL_EXPORT extern "C" __declspec(dllexport)

DECL_EXPORT int MyDLLTotalFunc( int count, int *values )
{
    int i, total = 0;
    for( i = 0; i < count; i++ )
        total += values[i];
    return total;
}
```

Functions written using 32-bit compilers must use the cdecl calling convention for parameter passing to work properly. Functions written using 64-bit compilers must use the Microsoft x64 calling conversion on Windows or the System V AMD64 ABI calling convention on Linux and macOS.

Passing Parameters to External Functions

The following data types can be passed to external functions:

- Basic integer types char, short, int, int64 (signed and unsigned).
- Floating point types float and double.
- References to the above basic types using the & character. Note that pointers are currently not allowed in Scripts or Templates.
- Arrays of the above basic types using '[]' or '[<size>]' to specify an array. Arrays are passed by reference if possible.
- *string* or *wstring*. Note that 010 Editor assumes that wstring and wchar_t are unsigned shorts but some compilers assume wchar_t is a 32-bit integer in which case the strings will need to be converted before being used.

Note that passing structs is currently not supported with external DLLs although this may be supported for certain compilers in the future. As an example, the following code could be used in a Binary Template to call the C++ function defined above:

```
#link "TotalDLL.dll"
int MyDLLTotalFunc( int count, int values[] );
#endlink

local int data[3] = { 30, 45, 2 };
local int total = MyDLLTotalFunc( 3, data );
Printf( "Total = %d\n", total );
```

Returning Values from External Functions

The following data types can be returned from external functions:

- Basic integer types char, short, int, int64 (signed and unsigned).
- Floating point types float and double.
- *string* or *wstring*. The string or wstring is assumed to be null-terminated and is copied. Note that char[] or wchar_t[] can also be used to specify a string or wstring.

External Functions in Templates

Calling external DLL functions in Templates is the same as Scripts except the Template has to be granted permission to execute external DLLs. See the External (DLL) Functions in Templates help topic for more information.

Related Topics:
External (DLL) Functions in Templates
Functions
Includes

010 Editor v10.0 Manual - Windows Edition

Copyright © 2003-2019 SweetScape Software

Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Script Limitations

Scripts have a syntax similar to C; however, the software was designed for parsing binary files and is not meant to be fully ANSI compliant. This section lists most of the important differences between ANSI C and 010 Editor when writing code:

- **Pointers -** No pointers are currently allowed using '*'. References using '&' are only allowed when passing arguments to custom functions (see Functions).
- **Preprocessor -** Most preprocessor directives are supported including #define, #ifdef, #ifndef, etc. However, the #if directive is not currently supported and defining macros using the #define directive is not currently supported.
- **Multi-dimensional Arrays -** Multi-dimensional arrays are currently not supported. See Template Limitations for an alternate way of declaring multi-dimensional arrays in Templates.
- **Control statements -** The *goto* statement is not supported.
- **Local structs -** Currently structs can only be defined in Templates and local structs are not supported in Scripts.

Related Topics:
Declaring Script Variables
Functions
Includes
Template Limitations

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Template Basics

Binary Templates, one of the most powerful features of 010 Editor, allow virtually any binary file to be parsed into a series of variables. Templates allow binary files to be understood and edited in a much easier fashion than typical hex editors. Each Template is stored as a text file with the extension ".bt" and can be edited directly in 010 Editor (see the Templates Menu). Templates are executed as an interpreter would run, starting from the first line in the file and progressing downwards. When a Template is executed the file is parsed into a number of variables and the variables are displayed in the *Template Results* panel (see Template Results for more information). Templates can be configured to automatically load and execute each time a file is opened (see Template Options). For an example of how Templates work open any ZIP, BMP, or WAV file on your computer and see the Repository Dialog for information on installing Templates from the Repository.

The syntax of Binary Templates is similar to that of scripts. The following topics describe the syntax specifically used when writing Binary Templates:

- Declaring Template Variables
- Data Types, Typedefs, and Enums
- Structs and Unions
- Arrays, Duplicates, and Optimizing
- Bitfields
- Expressions
- Control Statements
- Functions
- Special Keywords
- Preprocessor
- Includes
- Editing with Scripts
- Custom Variables
- On-Demand Structures
- Template Limitations

A large number of functions are available when writing Templates. The following topics list all available functions:

- Interface Functions
- I/O Functions
- String Functions
- Math Functions
- Tool Functions

Scripts can be used to modify variables defined in a Template. See Script Basics for more information on Scripts and Editing with Scripts for information on using Scripts and Templates together.

Related Topics:
Arrays, Duplicates, and Optimizing
Bitfields
Custom Variables
Declaring Template Variables
Editing with Scripts
Interface Functions
Introduction to Templates and Scripts
I/O Functions
Math Functions
On-Demand Structures
Preprocessor
Script Basics
String Functions
Structs and Unions

Template Limitations
Tool Functions
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Declaring Template Variables

Declaring variables in templates is performed similar to ANSI C and Scripts, but with an important difference: every time a variable is declared in the Template, that variable is mapped to a set of bytes in a file. For example, running the template:

```
    char       header[4];
    int        numRecords;
```

Would create the character array *header*, which is mapped to the first 4 bytes of the current file, and the integer *numRecords*, which is mapped to the next 4 bytes of the file. Both variables will be displayed in the Template Results panel and can be used for editing the file. Variables can also be edited using Scripts (see Editing with Scripts). The main way of grouping Template variables together is to declare structs or unions. See the Structs and Unions help topic for more information.

Special Attributes

One or more special attributes can be specified after a variable inside '<' and '>' brackets. The following attributes are supported:

```
   < format=hex|decimal|octal|binary,
     fgcolor=<color>,
     bgcolor=<color>,
     comment="<string>"|<function_name>,
     name="<string>"|<function_name>,
     open=true|false|suppress,
     hidden=true|false,
     read=<function_name>,
     write=<function_name>
     size=<number>|<function_name> >
```

All attributes are discussed below except for the *read* and *write* attributes which can be used to create special Custom Variables (see Custom Variables) and the *size* attribute which can be used to create on-demand structures (see On-Demand Structures).

Display Format

By default, all variables declared will be displayed in the Template Results panel in decimal format. To switch between hexadecimal, decimal, octal, or binary display formats, see the functions DisplayFormatDecimal, DisplayFormatHex, DisplayFormatBinary, and DisplayFormatOctal in Interface Functions.

An alternate way of specifying the format for a variable is to use the syntax '<*format*=hex|decimal|octal|binary>' after a variable declaration or a typedef. For example:

```
    int id;
    int crc <format=hex>;
    int flags <format=binary>;
```

Colors

When parsing a file, different colors can be applied to variables by using a Template. For example, the header bytes of a file could be colored differently than the rest of the file. There are two ways to control the color of variables. If you just wish to set the color of a single variable, the syntax '<*fgcolor*=???>' or '<*bgcolor*=???>' can be used after a variable to set the foreground or background color respectively. Here '???' can indicate either a built-in color constant (see SetBackColor for a list) or a number constant in the format '0xBBGGRR' (e.g. 0xFF0000 is blue). For example:

```
     int id <fgcolor=cBlack, bgcolor=0x0000FF>;
```

The second way of coloring variables is to use the SetForeColor, SetBackColor, or SetColor functions to set the default color. Every variable defined after a call to one of these functions will be assigned the default color. The special color constant 'cNone' can be used to turn off coloring. For example:

```
    SetForeColor( cRed );
    int first;  // will be colored red
    int second; // will be colored red
    SetForeColor( cNone );
    int third;  // will not be colored
```

See the SetBackColor function for more information. When using a dark Theme, background colors are automatically darkened to fit in better with the theme and this can be controlled using the ThemeAutoScaleColors function. Note that the *fgcolor* and *bgcolor* syntax requires 010 Editor version 3.1 or higher.

Endian

Any data written or read from a file depends upon the endian of the file (see Introduction to Byte Ordering). By default, all variables declared will have the same endian as the file, but the endian can be modified by using the functions BigEndian or LittleEndian (see I/O Functions). Using this technique, the same file can contain both little and big endian data.

Comments

A comment can be attached to a variable using the syntax '<*comment*="<string>">'. For example:

```
    int machineStatus <comment="This should be greater than 15.">;
```

This comment will be displayed in the *Comment* column of the Template Results. Alternately, a comment can be provided using a custom function using the syntax '<*comment*=<function_name>>'. The comment function takes as arguments a variable and returns a string to be displayed in the *Comment* column. For example:

```
     int machineStatus <comment=MachineStatusComment>;

     string MachineStatusComment( int status )
     {
         if( status <= 15 )
```

Copyright © 2003-2019 SweetScape Software

```
            return "*** Invalid machine status";
        else
            return "Valid machine status";
    }
```

Note that extended characters can be included in string constants when a Template file uses the UTF-8 character set. Comments with strings are only available in version 3.1 of 010 Editor or higher and comments with custom functions are only available in version 4.0 of 010 Editor or higher.

Names

The *name* attribute can be used to override the text displayed in the *Name* column of the Template Results. Similar to the *comment* attribute above, the *name* attribute can be given a string with the syntax '<*name*="<string>">' or can be given a function with the syntax '<*name*=<function_name>>'. A *name* function is similar to a comment function and takes as arguments a variable and returns a string. The following is an example of using the name attribute with a string:

```
    byte _si8 <name="Signed Byte">;
```

The above statement would display "Signed Byte" in the *Name* column of the Template Results instead of the string "byte _si8". Note that if the variable is part of an array, the array indices will be automatically appended to the name. The name attribute is only available in version 4.0 of 010 Editor or higher.

Order

After each Template variable is declared, the current file position is moved forward. The current file position can be examined using the function FTell. By using the functions FSeek or FSkip, the current position can be moved around the file. This technique allows a file to be parsed out of order. Note that to read from a file without defining a variable, the functions ReadByte, ReadShort, ReadInt, etc. can be used (see I/O Functions).

Local Variables

In some instances, a variable may be required which is not mapped to a file and not displayed in the Template Results (i.e. a regular C/C++ variable). In this case, the special keyword '*local*' can be used before the declaration. For example:

```
    local int i, total = 0;

    int    recordCounts[5];
    for( i = 0; i < 5; i++ )
        total += recordCounts[i];
    double records[ total ];
```

In this example, *i* and *total* are not added to the Template Results panel by default; however, the display of local variables in the Template Results panel can be enabled by right-clicking on the Template Results panel and clicking *Show Local Variables*.

Open Status of Variables

When a Template is run, all the created variables are displayed in a tree format in the Template Results. By default all arrays and structs will be closed in the tree and can be opened by clicking the small '+' or arrow beside each item; however, sometimes it is useful to have an array or struct open by default which makes viewing important data easier. To open an array or struct by default use the syntax '<*open*=true>' after a variable. The syntax '<*open*=false>' can also be used to set an array or struct closed after the template is run (this is the default behavior).

When the *Expand All Children of Node* operation is run on the Template Results tree (this is performed by right-clicking on the Template Results), all arrays or structs under the selected variable are opened. Alternately, all nodes in the Template Results can be recursively opened by right-clicking on the tree and selecting *Expand All Nodes*. To prevent an array or struct from being opened during an Expand All operation, use the syntax '<*open*=suppress>' after the variable. Controlling the open status of variables is only available in version 3.1 of 010 Editor or higher.

Strings

Null-terminated strings are commonly defined in binary files. 010 Editor allows the special syntax in a template to read a null-terminated string:

```
    char str[];
```

or

```
    string str;
```

will both read a string until a 0 byte is encountered. Unicode strings (wide strings) can be read using:

```
    wchar_t str[];
```

or

```
    wstring str;
```

See Arrays and Strings for more information on strings.

Enums

Enums are useful when writing Templates (see Data Types, Typedefs, and Enums for information on declaring enum types). When an enum variable is declared and the variable is selected in the Template Results, a down arrow will appear to the right of the text field. Clicking on the down arrow displays a drop-down list of all constants defined for the enum. Selecting an item from the drop-down list, or entering a constant in the edit field and pressing Enter will assign the variable to a new value. Enums may also be used with bitfields.

Hidden Variables

The syntax '<*hidden*=true>' can be used to hide the display of variables in the Template Results. This syntax can also be used with typedefs and '<*hidden*=false>' can be used to re-enable the display of a variable. Hidden variables are only available in version 3.1 of 010 Editor or higher.

Related Topics:
Arrays and Strings
Bitfields
Custom Variables
Data Types, Typedefs, and Enums
Declaring Script Variables
Editing with Scripts
Interface Functions
Introduction to Byte Ordering
I/O Functions
On-Demand Structures
Structs and Unions
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Structs and Unions

The C keyword '*struct*' can be used to define a hierarchical data structure when parsing a file. Structures can be defined using C/C++ syntax. For example:

```
struct myStruct {
    int a;
    int b;
    int c;
};
```

See Declaring Template Variables for information on declaring the variables within a struct. Note the semi-colon after the *struct* definition is required. This syntax would actually generate a new type *myStruct*, but would not declare any variables until an instance of type *myStruct* is declared:

```
myStruct s;
```

After this declaration, the Template Results would have an entry 'myStruct s' with a '+' beside it. Clicking the '+' would display the variables *a*, *b*, and *c* beneath.

Instances of structures can be declared at the same time the structure is defined. For example:

```
struct myStruct {
    int a;
    int b;
    int c;
} s1, s2;
```

would generate two instances of myStruct. *s1* would cover the first 12 bytes of the file (4 bytes for each of the 3 integers) and *s2* would cover the next 12 bytes of the file.

These structs are more powerful than typical C structs since they may contain control statements such as *if*, *for*, or *while*. For example:

```
struct myIfStruct {
    int a;
    if( a > 5 )
        int b;
    else
        int c;
} s;
```

In this example, when *s* is declared only two variables are generated: *a*, and one of *b* or *c*. Remember that templates are executed as an interpreter would, evaluating each line before stepping to the next. The value of *a* is read directly from the current file.

Structures can be nested and array of structures can also be declared. For example:

```
struct {
    int width;
```

Copyright © 2003-2019 SweetScape Software

```
        struct COLOR {
            uchar r, g, b;
        } colors[width];

    } line1;
```

Note that forward-declared structs are supported and structs can even be nested recursively. Typedefs can be used with structs as an alternate way to define a structure. For example:

```
    typedef struct {
        ushort id;
        int    size;
    }
    myData;
```

Unions

A union can be declared using the same syntax as structs except that the keyword '*union*' is used instead of '*struct*'. In a union, all the variables start at the same position and the size of the union is just large enough to contain the largest variable. For example, for the union:

```
    union myUnion {
        ushort s;
        double d;
        int    i;
    } u;
```

all three variables would be read starting from the same position and the size of the union would be 8 bytes to contain the double.

Structs with Arguments

A list of arguments can be specified after the '*struct*' or '*union*' keyword when defining a structure or union. This is a powerful way of working with complex structures and the argument list is defined in a similar manner to Functions. For example:

```
    struct VarSizeStruct (int arraySize)
    {
        int id;
        int array[arraySize];
    };
```

Then, when instances of this struct are declared the proper parameters must be passed to the struct in brackets. For example:

```
    VarSizeStruct s1(5);
    VarSizeStruct s2(7);
```

Arguments can also be used with structs or unions defined using typedefs. For example:

```
    typedef struct (int arraySize)
    {
        int id;
        int array[arraySize];
    } VarSizeStruct;
```

Related Topics:
Data Types, Typedefs, and Enums
Declaring Template Variables
On-Demand Structures

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Arrays, Duplicates, and Optimizing

Arrays and Duplicates

When writing a template, regular arrays can be declaring using the same syntax as scripts (see Arrays and Strings). However, 010 Editor has a syntax that allows arrays to be built in a special way. When declaring template variables, multiple copies of the same variable can be declared. For example:

```
int x;
int y;
int x;
```

010 Editor allows you to treat the multiple declarations of the variable as an array (this is called a *Duplicate Array*). In this example, *x[0]* could be used to reference the first occurrence of *x* and *x[1]* could be used to reference the second occurrence of *x*. Duplicate arrays can even be defined with *for* or *while* loops. For example:

```
local int i;
for( i = 0; i < 5; i++ )
    int x;
```

For another example, see the 'ZIPTemplate.bt' file that uses duplicate arrays to parse ZIP files. Duplicate arrays are also useful because of how 010 Editor deals with arrays of structs.

Optimizing Structs

010 Editor contains an optimization that allows it to generate arrays of structures with millions and millions of elements. The optimization calculates the size of the first element of the array and assumes all elements of the array are the same size. This optimization will cause incorrect results if the elements can be variable size (010 Editor will display a warning in the Output text area if it detects this may be happening). To turn the optimization off, use the syntax '*<optimize=false>*' after the array declaration. For example:

```
typedef struct {
    int   id;
    int   length;
    uchar data[ length ];
} RECORD;

RECORD record[5] <optimize=false>;
```

If 010 Editor displays a warning in the Output text area, but you would still like to use the optimization, use the syntax '*<optimize=true>*' to turn off display of the warning. The '*<optimize=false>*' syntax can also be used with struct typedefs to indicate that optimization should not be used on any arrays using that structure. Unoptimized arrays can also be rewritten as duplicate arrays (see above) since the elements of duplicate arrays can have different sizes.

Related Topics:
Arrays and Strings
Structs and Unions

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Bitfields

A bitfield allows structures to be subdivided into groups of bits. This process allows multiple variables to be packed into a single block of memory. The syntax for defining a bitfield is 'type_name <variable_name> : number_of_bits'. The type can be char, short, int, or int64 (unsigned or signed) or any of the equivalent types. If the variable name is omitted, the given number of bits will be skipped. For example,

```
    int alpha : 5;
    int       : 12;
    int beta  : 15;
```

would pack alpha and beta into one 32-bit value, but skip 12 bits in the middle. 010 Editor has two special bitfield modes that determine how the bits are packed into variables: *padded* bitfields and *unpadded* bitfields.

Padded Bitfields

With padded bitfields (the default), how the bits are packed into a variable depends upon the current endianness. By default, bits are packed right-to-left for little endian files, and left-to-right for big endian files. For example, for the bitfields:

```
    ushort a : 4;
    ushort b : 7;
    ushort c : 5;
```

In little endian mode, this structure would be stored as the bits:

```
    cccccbbb bbbbaaaa
```

(and stored on disk as bbbbaaaa cccccbbb). In big endian mode, this structure would be stored as the bits:

```
    aaaabbbb bbbccccc
```

(and stored on disk as aaaabbbb bbbccccc). Whether the bits are packed left-to-right or right-to-left can be controlled by the functions BitfieldLeftToRight, and BitfieldRightToLeft (see I/O Functions).

In this mode, the program will automatically add padding bits when needed. If the size of the type being defined changes, padding will be added so the bitfield will be defined on the next variable boundary. Also, if the specified bitfield would step across the boundary of a variable, padding is added so the bitfield starts at the next variable. For example:

```
    int   apple  : 10;
    int   orange : 20;
    int   banana : 10;
    int   peach  : 12;
    short grape  : 4;
```

The bitfields apple and orange would be packed into one 32 bit value. However, banana steps over the variable boundary, so 2 bits of padding are added so that it starts at the next 32 bit value. Banana and peach are packed into another 32-bit value, but because the size of the type changes with grape, an extra 10 bits of padding is added before grape is defined.

Unpadded Bitfields

010 Editor includes a special unpadded bitfield mode that treats the file as one long bit stream. No padding bits are added if the variable type changes or if the bits cannot be packed into a single variable. The unpadded mode can be entered by calling the function BitfieldDisablePadding (padding can be turned back on by calling BitfieldEnablePadding).

In unpadded bitfield mode, each variable defined reads some bits from the bitstream. For example:

```
BitfieldDisablePadding();
short a : 10;
int   b : 20;
short c : 10;
```

Here *a* reads the first 10 bits from the file, *b* reads the next 20 bits from the file, and so on. If the bitfields are defined as reading from right to left (this is the default for little endian data and can enabled using the function BitfieldRightToLeft), the variables would be stored as the bits:

```
aaaaaaaa bbbbbbaa bbbbbbbb ccbbbbbb cccccccc
```

If the bitfields are defined as reading from left to right (this is the default for big endian data and can enabled using the function BitfieldLeftToRight), the variables would be stored as the bits:

```
aaaaaaaa aabbbbbb bbbbbbbb bbbbbbcc cccccccc
```

When declaring structures containing unpadded bitfields, no extra padding bits are added between the structures. (Note that internally, any unpadded right-to-left bitfield is forced to be declared in little endian mode and any unpadded left-to-right bitfield is forced to be declared in big endian mode.)

Bitfields and Enums

Bitfields may be combined with enums by placing a colon and a number of bits after the enum definition. For example, to pack two enums into a single ushort, the following may be used:

```
enum <ushort> ENUM1 { VAL1_1=25, VAL1_2=29, VAL1_3=7 } var1 : 12;
enum <ushort> ENUM2 { VAL2_1=5,  VAL2_2=6 }            var2 : 4;
```

Related Topics:
Declaring Template Variables
I/O Functions

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Editing with Scripts

Templates are meant only to parse a binary file and cannot modify the data file they are run on (however, Templates can be allowed to read from or write to other files using Permissions). To edit the variables defined from the template, use either the Template Results panel or a script. When a variable is declared in a Template, it is mapped to a set of bytes in the file. Reading the variable causes bytes in the file to be read, and assigning to the variable causes the bytes of the file to be modified.

Scripts have access to any of the variables declared in the Template and can use '.' to access data in structures. For example, using the template:

```
struct myStruct {
    int a;
    int b;
    int c;
} s1, s2;
```

a script could be written to swap the *a* variables:

```
int temp;
temp = s1.a;
s1.a = s2.a;
s2.a = temp;
```

Note that there is no special syntax to use; a Script can automatically access any Template variable if the Script is run after the Template is run. For a list of other special keywords that can be used while editing Template variables with a Script, see Special Keywords. Using a combination of Templates and Scripts provides a powerful method of editing binary files.

Related Topics:
Permission Options
Special Keywords
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Custom Variables

Some binary file formats use variable types that are different from the regular Data Types. 010 Editor has a powerful syntax for letting you define custom variables in practically any format. To build a custom variable, write a custom function that converts a variable into a string. This string will be displayed in the Template Results panel as the value for the variable (this custom function is called a *read* function). Optionally, a *write* function can be written that converts a string in the Template Results panel back to the variable when the variable is edited and Enter is pressed. To assign a *read* and *write* function to a variable, use the syntax '<*read*= <function_name> [, *write*= <function_name>] >'after a typedef. A *read* function takes as arguments a variable and returns a string. A *write* function takes as arguments a reference to a variable and a string. For example, to define a fixed point data type that uses 16 bits (the 8 high bits define the whole part and the 8 low bits define the fractional part), use:

```
    typedef ushort FIXEDPT <read=FIXEDPTRead, write=FIXEDPTWrite>;

    string FIXEDPTRead( FIXEDPT f )
    {
        string s;
        SPrintf( s, "%lg", f / 256.0 );
        return s;
    }

    void FIXEDPTWrite( FIXEDPT &f, string s )
    {
        f = (FIXEDPT)( Atof( s ) * 256 );
    }
```

In this example, the FIXEDPT variable could be defined without a write function using '<read=FIXEDPTRead>', but the variable will be read-only in the Template Results panel. Note that the typedef must be defined before the functions in the source file. If an error can occur when a *write* function is called, change the return value to int and return 0 on success or -1 on failure.

If a run-time error occurs inside a read function, 010 Editor will display "(error)" in the Template Results panel and then display "(error)" every time the read function was to be called (010 Editor will not repeatedly call functions which cause run-time errors because of performance issues). Fix the error within the read function and then re-run the template to use the read function again.

When displaying arrays in the Template Results panel, usually no value is displayed for an array until the array is opened up. To specify a value beside the array, a custom variable can be defined. For example:

```
    typedef float VEC3F[3] <read=Vec3FRead, write=Vec3FWrite>;

    string Vec3FRead( VEC3F v )
    {
        string s;
        SPrintf( s, "(%f %f %f)", v[0], v[1], v[2] );
        return s;
    }

    void Vec3FWrite( VEC3F &v, string s )
    {
        SScanf( s, "(%f %f %f)", v[0], v[1], v[2] );
    }
```

Note that currently, read/write functions cannot be declared simultaneously for an array of elements and for each element of the array. The same process can be used to display a value for a struct in the Template Results panel. For an example, see the 'ZIPTemplate.bt' file that uses a read function to display the file name beside each structure.

Related Topics:
Arrays and Strings
Data Types, Typedefs, and Enums
Functions
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

On-Demand Structures

Some 010 Editor templates that define a large number of variables may use a significant amount of system memory. To get around this issue, templates may use *On-Demand Structures*. With *On-Demand Structures* the code which defines the variables inside a struct or union is not executed until it is needed (either the structure is opened in the Template Results or a Script or Template accesses a member variable of the struct or union). In order to use on-demand structures, 010 Editor must know the size of the struct or union so that it knows where the next variable exists within the file. To enable on-demand parsing for a struct or union, use the syntax '<*size*=<number>|<function name>' after a typedef.

If the size of a struct or union is always fixed, a number can be passed to the *size* attribute (note that sizes are always in number of bytes). For example:

```
typedef struct
{
    int header;
    int value1;
    int value2;
} MyStruct <size=12>;
```

In this case, anytime a struct of type MyStruct is defined the variables *header*, *value1*, and *value2* are not defined until they are needed.

If the size of struct or union is not fixed, a custom *size* function can be specified. The *size* function takes as arguments a variable and returns an int or int64. If data needs to be read from the disk inside a size function the Read<data_type> functions must be used (see I/O Functions). The member variables of the struct or union cannot be accessed inside the *size* function because they do not exist until after the *size* function is called. For an example of a *size* function, in the ZIPTemplate.bt file the ZIPFILERECORD struct could be converted to on-demand using this syntax:

```
typedef struct {
    <...>
    uint      frCompressedSize;
    uint      frUncompressedSize;
    ushort    frFileNameLength;
    ushort    frExtraFieldLength;
    if( frFileNameLength > 0 )
        char      frFileName[ frFileNameLength ];
    if( frExtraFieldLength > 0 )
        uchar     frExtraField[ frExtraFieldLength ];
    if( frCompressedSize > 0 )
        uchar     frData[ frCompressedSize ];
} ZIPFILERECORD <size=SizeZIPFILERECORD>;

int SizeZIPFILERECORD( ZIPFILERECORD &r )
{
    return 30 +                          // base size of the struct
        ReadUInt(startof(r)+18) +        // size of the compressed data
        ReadUShort(startof(r)+26) +      // size of the file name
        ReadUShort(startof(r)+28);       // size of the extra field
        }
```

Note that the *size* function should always return the size of a single element of the struct or union even if the struct or union is defined as part of an array. Also note that applying colors to variables within on-demand structures may not work as expected because the colors are not defined until the variables are created. Using on-demand structs combined with arguments is only supported in 010 Editor v10 or higher versions.

Related Topics:
I/O Functions
Structs and Unions
Working with Template Results

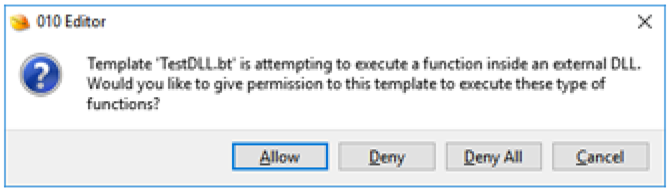
010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

External (DLL) Functions in Templates

Scripts and Templates can execute functions that reside in an external dynamic link library. The method of declaring and writing these functions is described in the External (DLL) Functions in Scripts help topic. The main difference when executing external functions in Templates is that the Template has to be granted permission to access the dynamic link library.

Granting Permission to Templates for using DLLs

When attempting to use the #link statement inside a Binary Template the following message is displayed:


010 Editor

Template 'TestDLL.bt' is attempting to execute a function inside an external DLL. Would you like to give permission to this template to execute these type of functions?

The *Allow* button must be clicked to let the Template continue execution otherwise the Template will be stopped with an error. The permissions for executing functions inside DLLs can also be granted or revoked using the Permission Options dialog.

Related Topics:
External (DLL) Functions in Scripts
Functions
Includes
Permission Options

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Template Limitations

Binary Templates have a slightly different syntax than regular ANSI C files and are not meant to be fully ANSI compliant. The following is a list of important differences between ANSI C and Binary Templates:

- **Pointers -** See Script Limitations.
- **Preprocessor -** See Script Limitations.
- **Multi-dimensional Arrays -** When declaring arrays (see Arrays and Strings), multi-dimensional arrays are not supported (this includes arrays of strings). When writing a Template, a combination of structs and arrays can be used to simulate multi-dimensional arrays. For example, the array 'float matrix[4][4]' could be defined as

```
    typedef struct
    {
        float row[4];
    }
    MATRIX[4];

    MATRIX m;
```

- **Control statements -** See Script Limitations.

Related Topics:
Arrays and Strings
Script Limitations
Structs and Unions

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Interface Functions

The following is a list of functions for communicating with the 010 Editor program when writing a Template or Script.

void AddBookmark(
int64 pos,
string name,
string typename,
int arraySize=-1,
int forecolor=cNone,
int backcolor=0xffffc4,
int moveWithCursor=false)

Creates a bookmark in the file on which the current Script or Template is being executed (not the actual Script or Template). See Using Bookmarks for information on bookmarks and see Running Templates and Scripts for information on controlling on which file a Script or Template is run.

pos indicates the file address where the bookmark will be generated and *name* gives the name of the bookmark (this name can be empty). *typename* is a string giving the type of the bookmark to generate and this must be one of the built-in types or a user-defined type. If the type is user-defined and the AddBookmark function is run from a Template, that type must be defined in the current Template. If the type is user-defined and the AddBookmark function is run from a Script, that type must be defined in a Template that has already been run on the file (not in the current Script). Specify an array of variables using the *arraySize* argument or set *arraySize* to -1 to just generate a single variable. The text color of the bookmark can be indicated using the *forecolor* argument and the background color can be specified with the *backcolor* argument. If the *moveWithCursor* argument is true, the bookmark will reposition itself to the cursor as the cursor moves around the file.

For example, after loading a ZIP file in 010 Editor (which will cause the 'ZIPTemplate.bt' Template to be run on the file), create a new Script and run the following command on the file:

```
AddBookmark( GetCursorPos(), "endmarker",
    "ZIPENDLOCATOR", -1, cRed );
```

Here the type "ZIPENDLOCATOR" type is defined in the 'ZIPTemplate.bt' file, not in the Script. The functions GetNumBookmarks, GetBookmarkPos and GetBookmarkName can be used to query existing bookmarks and the RemoveBookmark function can be used to erase bookmarks.

Requires 010 Editor v3.1 or higher.

void Assert(int value, const char msg[] = "")

Stops execution of the script or template if the *value* is equal to zero. If execution is stopped, the text message *msg* will be displayed in the Output tab of the Output Window (note that this is an optional argument). Because conditional statements evaluate to 1 or 0 in C/C++, comparisons can be used in asserts. For example:

```
Assert( numRecords > 10,
  "numRecords should be more than 10." );
```

Requires 010 Editor v3.1 or higher.

void ClearClipboard()

Removes any data that is on the currently active clipboard. See the SetClipboardIndex function to control which clipboard is active.

Requires 010 Editor v3.1 or higher.

---

void CopyBytesToClipboard(uchar buffer[], int size, int charset=CHARSET_ANSI, int bigendian=false)
Copies *size* bytes from the array *buffer* and places them on the currently active clipboard. If the data being copied represents a string, use the *charset* parameter to indicate which character set the string uses (see the ConvertString function for a list of character sets). If copying Unicode data the *bigendian* parameter indicates if the data is in big or little endian format, otherwise this parameter is ignored. See the SetClipboardIndex function to control which clipboard is currently active.

Requires 010 Editor v5.0 or higher.

---

void CopyStringToClipboard(const char str[], int charset=CHARSET_ANSI)
Sets the currently active clipboard to contain the string passed in the *str* argument. The *charset* parameter tells the clipboard what character encoding is used for the string (e.g. ANSI, UTF-8, etc.) and the list of possible character sets is available in the ConvertString function. If copying hex bytes to the clipboard see the CopyBytesToClipboard function and the active clipboard can be set using the SetClipboardIndex function.

Requires 010 Editor v3.1 or higher.

---

void CopyToClipboard()
Copies the currently selected bytes to the active clipboard. See SetSelection to change the selection or SetClipboardIndex to set the active clipboard.

---

void CutToClipboard()
Copies the currently selected bytes to the active clipboard and deletes them from the file. See SetSelection to change the selection or SetClipboardIndex to set the active clipboard.

---

int DeleteFile(char filename[])
Deletes the file given by *filename* from disk. Note that the file should not be open in the editor or the delete will fail. Returns a value less than zero on failure.

---

void DisableUndo()
Turns off undo support for a file. This function will speed up a script when writing a large number of changes to a file. Undo is automatically turned on after the script is finished. Note that undo is automatically disabled for files created with FileNew. See EnableUndo below.

---

void DisplayFormatBinary()
void DisplayFormatDecimal()
void DisplayFormatHex()
void DisplayFormatOctal()
Sets the display format of variables in the Inspector to binary, decimal, hexadecimal, or octal. Any variables declared after this function is called will be displayed in the selected format. Note that the format can also be set for one variable using the syntax <format=hex|decimal|octal|binary> after a declaration or typedef (see Declaring Template Variables for more information).

---

void EnableUndo()
Turns back on Undo support for a file after calling DisableUndo. Undo is automatically turned on after the script is finished.

---

int Exec(const char program[], const char arguments[], int wait=false)
int Exec(const char program[], const char arguments[], int wait, int &errorCode)
Executes an external application using the given *program* and *arguments*. Returns 0 if successful or a negative number if an error occurred. If the *wait* argument is true, this function waits for the external application to finish processing before it returns. If *wait* is true and the *errorCode* parameter is given, the error code from the executed program will be stored in the *errorCode* variable. Note that starting in 010 Editor version 3.1, this function can no longer be called in a Template because of security issues.

Requires 010 Editor v3.1 or higher for the wait parameter.
Requires 010 Editor v6.0 or higher for the errorCode parameter.

---

void Exit(int errorcode)
Ends execution of the current script or template and displays the *errorcode* in the Output tab of the Output panel. Note that usually the keyword *return* can be used to accomplish the same task unless execution is inside of a custom function.

This function is special in that it can be used to return an ERRORLEVEL code to a batch file (see Command Line Parameters for more information). The last *errorcode* that was passed to an Exit function, either from a script or a template, will be returned as the global ERRORLEVEL when 010 Editor exits.

---

void ExpandAll()
Recursively opens all tree nodes in the Template Results panel. Variables that have the attribute '*<open=suppress>*' set will not be opened. See Template Results for more information.

Requires 010 Editor v3.2 or higher.

---

void ExportCSV(const char filename[])
Writes the contents of the Template Results panel to a file in a comma-delimited format which can be opened in other programs such as Excel. The file is saved to the given *filename* and the ExpandAll function can be called first to ensure all variables are included in the export. Alternately, the FPrintf function can be used to output individual variables.

Requires 010 Editor v3.2 or higher.

---

void ExportXML(const char filename[])
Similar to ExportCSV except the contents of the Template Results panel are written to a file in XML format. The file to write is given by the *filename* parameter. Call the ExpandAll function first to ensure all variables are included in the XML file.

Requires 010 Editor v6.0 or higher.

---

void FileClose()
Closes the current file. No file will be active after this function is called and use the FileSelect function to activate another file. This function can also be used to close an open process or drive. Note that the user will not be asked to save changes if the file is modified.

---

int FileCount()
Returns the number of existing file handles. See FileSelect to set the current file.

---

int FileExists(const char filename[])
Returns true if the given file name exists on disk, or false if it does not.

---

int FileNew(char interface[]="", int makeActive=true)
Creates a new file in the editor and returns the index of the created file (see GetFileNum). If the *interface* parameter is not empty, the created file will be assigned that File Interface the same as using the SetFileInterface function (for example, "Hex", "Text", or "Unicode"). If no *interface* is given, the default File Interface is used from the Editor Options dialog.

If this function is run from a script and the *makeActive* parameter is true, the created file becomes the current file. If this function is run from a template or the *makeActive* parameter is false, the current file does not change (use the FPrintf function to write data to the returned file handle). Note that when new files are created, the undo buffer is turned off by default, but will be turned on as soon as the script finishes.

Requires 010 Editor v4.0 or higher for the interface parameter.
Requires 010 Editor v5.0 or higher for the makeActive parameter.

---

int FileOpen(const char filename[], int runTemplate=false, char interface[]="", int openDuplicate=false)
Opens the file specified by the UTF-8 encoded string *filename* into the editor. If *runTemplate* is true and a Template is associated with that file (see Template Options), the Template will be run on the file. The opened file will be assigned the File Interface *interface* (for example, "Hex", "Text", or "Unicode") and if *interface* is an empty string, FileOpen will automatically choose an interface. If *openDuplicate* is true and the file to be opened is already open in the editor, a duplicate copy of the file will be created but if *openDuplicate* is false, no action will be taken. This function returns the file index of the opened file if the file could be loaded (see GetFileNum), the file index of the already opened file if *openDuplicate* is false and the file is already open, or a negative number if the file could not be opened (no error message is displayed if the file could not be found). FileOpen is usually called in a Script but can be called in a Template if that Template is given special permission to read other files (see Permissions).

Requires 010 Editor v3.1 or higher for the runTemplate parameter.
Requires 010 Editor v4.0 or higher for the interface or openDuplicate parameters.

---

int FileSave()
int FileSave(const char filename[])
int FileSave(const wchar_t filename[])
Saves the current file to the given file name. The file name can be either a UTF-8 string or a wide string. If FileSave is called with no parameters, the file is saved to the current file name. See GetFileName or GetFileNameW to retrieve the name of the current file. Returns a value less than zero on error. Note that when using backslashes in Windows path names, two backslashes must be used in string constants. For example:

```
    FileSave( "C:\\temp\\data.log" );
```

---

int FileSaveRange(const char filename[], int64 start, int64 size)
int FileSaveRange(const wchar_t filename[], int64 start, int64 size)
Similar to the FileSave function except that only part of the file is written to disk. *size* bytes starting from the address *start* are written to the given *filename* and *filename* can be given in UTF-8 or Unicode format. Note that *filename* must specify a different file than the current file name. Returns a negative value on error.

Requires 010 Editor v5.0 or higher.

---

void FileSelect(int index)
Only one file can be active at a time and all of the Read/Write operations occur on that file. Use FileSelect to select a different file to be the current file. The files are numbered from 0 up to FileCount()-1. See GetFileNum to get the index of the current file.

---

int FindOpenFile(const char path[])
int FindOpenFileW(const wchar_t path[])
Searches through the list of all open files to see if the file indicated by *path* is currently open. If the file is open, the index of the file is returned which can be used with the FileSelect function. If the file cannot be found, -1 is returned. The FindOpenFileW operates similarly to the FindOpenFile function but takes as input a Unicode file name.

Requires 010 Editor v4.0 or higher.

---

char[] GetArg(int index)
wchar_t[] GetArgW(int index)
Returns an argument passed to a script or a template from the command line. The GetArg function returns an ASCII string and the GetArgW function returns a wide string. The *index* parameter must be a number between 0 and GetNumArgs()-1. If *index* is an invalid number an empty string is returned. See the GetNumArgs function or the -script or -template parameter for more information on passing command line arguments.

Requires 010 Editor v3.2 or higher.

---

int GetBackColor()
Returns the currently active background color for template variables. See SetBackColor for more information.

Requires 010 Editor v6.0 or higher.

---

int GetBookmarkArraySize(int index)
If the bookmark specified by *index* is an array this function returns the number of elements in the array, otherwise this function returns -1. *index* should be greater or equal to zero and less than the value returned by GetNumBookmarks. See AddBookmark for more information.

Requires 010 Editor v5.0 or higher.

---

int GetBookmarkBackColor(int index)
This function returns an integer representing the background color of the bookmark specified by *index*. *index* should be zero or greater and less than the number of bookmarks (see GetNumBookmarks). More information about creating bookmarks can be found in the AddBookmark help topic.

Requires 010 Editor v5.0 or higher.

---

int GetBookmarkForeColor(int index)
Returns an integer representing the foreground (text) color of the bookmark indicated by the *index* parameter. A return value of *cNone* means that the foreground color is not changed. *index* should be greater or equal to zero and less that the total number of bookmarks. See GetNumBookmarks to retrieve the number of bookmarks and AddBookmark for more information on adding bookmarks.

Requires 010 Editor v5.0 or higher.

---

int GetBookmarkMoveWithCursor(int index)
This function returns true if the bookmark specified by *index* is set to move around the file as the cursor moves

(a floating bookmark). False is returned if this is not a floating bookmark. Bookmarks can be created to move with the cursor using the AddBookmark function. Note that *index* should be greater or equal to zero and less that the value returned by GetNumBookmarks.

Requires 010 Editor v5.0 or higher.

---

string GetBookmarkName(int index)
Returns a string which contains the name of a bookmark. *index* controls which bookmark name is returned and the value of *index* should be greater or equal to zero, and less than the number of bookmarks (see GetNumBookmarks). See AddBookmark for information on creating bookmarks.

Requires 010 Editor v3.1 or higher.

---

int64 GetBookmarkPos(int index)
Returns the starting address of a bookmark and the bookmark returned is specified using the *index* argument. *index* should be greater or equal to zero, and less than the number of bookmarks (see GetNumBookmarks). See AddBookmark for information on creating bookmarks.

Requires 010 Editor v3.1 or higher.

---

string GetBookmarkType(int index)
This function returns a string representing the type of data stored by the given bookmark (for example, "char" or "int"). The *index* parameter must be greater or equal to zero and less than the value returned by GetNumBookmarks. See the AddBookmark function for information about creating bookmarks.

Requires 010 Editor v5.0 or higher.

---

int GetBytesPerLine()
Returns the number of bytes displayed per line in the current Hex Editor Window. This value is by default 16, but may change depending upon the current settings in the View Menu.

---

int GetClipboardBytes(uchar buffer[], int maxBytes)
Reads data from the currently active clipboard into the given *buffer*. At most *maxBytes* bytes will be read from the clipboard and the *buffer* array must be large enough to hold *maxBytes* bytes. The return value is the number of bytes written into the buffer, or 0 if no bytes could be read. Also see the GetClipboardString for reading strings from the clipboard and the active clipboard can be set using the SetClipboardIndex function.

Requires 010 Editor v5.0 or higher.

---

int GetClipboardIndex()
Returns the index of the currently active clipboard. A return value of 0 means the standard system clipboard is chosen and a value of 1 to 9 inclusive means a custom clipboard is chosen. All clipboard operations including CopyToClipboard, CutToClipboard, PasteFromClipboard, CopyBytesToClipboard, CopyStringToClipboard, GetClipboardBytes, GetClipboardString, and ClearClipboard operate on the currently selected clipboard. Use the SetClipboardIndex function to select a different clipboard and see Using the Clipboard for more information on using multiple clipboards. The currently active clipboard is indicated in the Status Bar.

Requires 010 Editor v5.0 or higher.

---

string GetClipboardString()
If the active clipboard currently contains a string, this string will be returned by the GetClipboardString function. If the data on the clipboard cannot be converted to a string, an empty string will be returned. If the clipboard contains hex data that may include null characters, use the GetClipboardBytes function instead. The active

clipboard can be set using the SetClipboardIndex function.

Requires 010 Editor v3.1 or higher.

---

string GetCurrentTime(char format[] = "hh:mm:ss")
Returns a string representing the current time in the format "hh:mm:ss" by default (note this is using the 24-hour clock). For information on different formats that can be used see the GetCurrentDateTime function and to use the current application time format see the GetDefaultTimeFormat function.

Requires 010 Editor v3.1 or higher.
Requires 010 Editor v4.0 or higher for the format parameter.

---

string GetCurrentDate(char format[] = "MM/dd/yyyy")
Returns a string representing the current date in the format "MM/dd/yyyy" by default. For information on different formats that can be used see the GetCurrentDateTime function and to use the current application date format see GetDefaultDateFormat.

Requires 010 Editor v3.1 or higher.
Requires 010 Editor v4.0 or higher for the format parameter.

---

string GetCurrentDateTime(char format[] = "MM/dd/yyyy hh:mm:ss")
Returns a string representing the current date and time in the format "MM/dd/yyyy hh:mm:ss" by default (note this is using the 24-hour clock). When specifying a custom *format*, the following characters can be used:

- h - hour without leading zero
- hh - hour with leading zero
- m - minute without leading zero
- mm - minute with leading zero
- s - second without leading zero
- ss - second with leading zero
- z - millisecond without leading zero
- zzz - millisecond with leading zero
- AP - either AM or PM
- ap - either am or pm
- d - day without leading zero
- dd - day with leading zero
- ddd - short day (e.g. 'Mon')
- dddd - long day (e.g. 'Monday')
- M - month without leading zero
- MM - month with leading zero
- MMM - short month (e.g. 'Jan')
- MMMM - long month (e.g. 'January')
- yy - 2-digit year
- yyyy - 4-digit year

The function GetDefaultDateTimeFormat can also be used to get the current application date/time format. Functions such as StringToTimeT or StringToOleTime can be used to convert the resulting string to a date format.

Requires 010 Editor v3.1 or higher.
Requires 010 Editor v4.0 or higher for the format parameter.

---

int64 GetCursorPos()
Returns the address of the cursor in the file.

---

string GetDefaultDateFormat()
Returns the default date format for the application as set in the Inspector Options dialog. This date format is used in the Inspector and the Template Results.

Requires 010 Editor v8.0 or higher.

---

string GetDefaultDateTimeFormat()
Returns a string containing the default date format followed a single space followed by the default time format. Both date and time formats can be set using the Inspector Options dialog.

Requires 010 Editor v8.0 or higher.

---

string GetDefaultTimeFormat()
Returns the default time format set in the Inspector Options dialog. This time format is used in the Inspector and the Template Results.

Requires 010 Editor v8.0 or higher.

---

char[] GetEnv(const char str[])
Attempts to locate the system environment variable indicated by *str*. If the environment variable is found, the value of the environment variable is returned as a UTF-8 string and if it could not be found an empty string is returned.

Requires 010 Editor v4.0 or higher.

---

int GetFileAttributesUnix()
Returns the file attributes of a file on a Unix or Macintosh operating system as a bit flag. The resulting value has nine different flags: read, write and execute for each of the areas user, group, and other. The following constants can be used for testing the bits:

- FILEATTR_READ_USER
- FILEATTR_WRITE_USER
- FILEATTR_EXE_USER
- FILEATTR_READ_GROUP
- FILEATTR_WRITE_GROUP
- FILEATTR_EXE_GROUP
- FILEATTR_READ_OTHER
- FILEATTR_WRITE_OTHER
- FILEATTR_EXE_OTHER

If the file is not a valid file or the file is not in a Unix or Macintosh operating system, the constant FILEATTR_INVALID is returned. For example:

```
int flags = GetFileAttributesUnix();
if( (flags != FILEATTR_INVALID) && !(flags & FILEATTR_READ_USER) )
    Printf( "File is read only.\n" );
```

See the SetFileAttributesUnix function to modify the attributes.

Requires 010 Editor v4.0 or higher.

---

int GetFileAttributesWin()
Returns the file attributes of a file on a Windows operating system as a bit flag. The resulting constants can be

used for testing the bits:

- FILEATTR_ARCHIVE
- FILEATTR_HIDDEN
- FILEATTR_READONLY
- FILEATTR_SYSTEM

If the file is not a valid file or the file is not in a Windows operating system, the constant FILEATTR_INVALID is returned. For example:

```
int flags = GetFileAttributesWin();
if( (flags != FILEATTR_INVALID) && (flags & FILEATTR_READONLY) )
    Printf( "File is read only.\n" );
```

See the SetFileAttributesWin function to modify the attributes.

Requires 010 Editor v4.0 or higher.

---

int GetFileCharSet()
Returns an integer representing the character set of the current file. The list of possible character sets is available in the ConvertString function.

Requires 010 Editor v4.0 or higher.

---

char[] GetFileInterface()
Returns a string representing the File Interface of the current file. The File Interface name is listed in the *Edit As* section above each editor (for example: "Hex", "Text", or "Unicode"). See the SetFileInterface function to change the interface of the current file.

Requires 010 Editor v4.0 or higher.

---

char[] GetFileName()
Returns a string representing the file name of the current file including the path. The string is returned in UTF-8 format.

---

wchar_t[] GetFileNameW()
Returns a wide string which contains the file name of the current file including the path.

Requires 010 Editor v3.2 or higher.

---

int GetFileNum()
Each open file is assigned a index from 0 up to FileCount()-1. This function returns the index of the current file. Use the FileSelect function to make another file the active file.

---

int GetForeColor()
Returns the currently active foreground (text) color for template variables. See SetForeColor for more information.

Requires 010 Editor v6.0 or higher.

Copyright © 2003-2019 SweetScape Software

int GetMouseWheelScrollSpeed()
Returns the number of lines that are scrolled every time a mouse scroll wheel is clicked ahead or back. The scroll speed can be controlled with the Editor Options dialog or with the SetMouseWheelScrollSpeed function.

Requires 010 Editor v6.0 or higher.

---

int GetNumArgs()
Returns the number of arguments that were passed to this script or template from the command line. The individual arguments can be accessed using the GetArg and GetArgW functions. For more information on how to pass arguments see the -script and -template command line parameters.

Requires 010 Editor v3.2 or higher.

---

int GetNumBookmarks()
Returns the number of bookmarks set for the file that the current Script or Template is being run on (see Running Templates and Scripts for more information on choosing which file to run a Template or Script on). See the AddBookmark function for information on creating bookmarks.

Requires 010 Editor v3.1 or higher.

---

int GetReadOnly()
Returns true if the file is marked as read-only, or false if the file can be modified.

---

char[] GetScriptName()
wchar_t[] GetScriptNameW()
These functions return the name of the script that is currently being executed. This name is calculated by taking the full file name of the script and removing the path information. For example, if the script being run was 'H:\Scripts\Test.1sc', these functions would return 'Test.1sc'. GetScriptName returns a UTF-8 string and GetScriptNameW returns a Unicode string.

Requires 010 Editor v4.0 or higher.

---

char[] GetScriptFileName()
wchar_t[] GetScriptFileNameW()
When a script is being executed, these functions return the full file name of the script being run. GetScriptFileName returns a UTF-8 string and GetScriptFileNameW returns a Unicode string.

Requires 010 Editor v4.0 or higher.

---

int64 GetSelSize()
Returns the number of bytes that have been selected. Returns 0 if no selection is made.

---

int64 GetSelStart()
Returns the start address of the selection. Use GetSelSize to determine if a selection has been made.

---

string GetTempDirectory()
Returns a string representing the current temporary directory set using the Directory Options dialog. The temp

directory will contain a slash as the last character (e.g. "C:\temp\").

Requires 010 Editor v3.1 or higher.

---

char[] GetTempFileName()
Returns the full path of a file that can be used as a temporary file. The file will be in the current temporary directory and will be guaranteed not to exist. The returned file name is in UTF-8 format.

Requires 010 Editor v4.0 or higher.

---

char[] GetTemplateName()
wchar_t[] GetTemplateNameW()
These functions operate by taking the full template file name as returned from the functions GetTemplateFileName or GetTemplateFileNameW, removing the path information and returning the result. For example, if the current Template being run was 'H:\Templates\Test.bt', these functions would return 'Test.bt'. GetTemplateName returns a UTF-8 string and GetTemplateNameW returns a Unicode string.

Requires 010 Editor v4.0 or higher.

---

char[] GetTemplateFileName()
wchar_t[] GetTemplateFileNameW()
When run in a Template, these functions return the full file name of the Template that is being run. When run in a Script, these functions return the full file name of the Template that has been associated with the target file, or an empty string if there is no associated Template. GetTemplateFileName returns a UTF-8 string and GetTemplateFileNameW returns a Unicode string.

Requires 010 Editor v4.0 or higher.

---

char[] GetWorkingDirectory()
wchar_t[] GetWorkingDirectoryW()
Returns the full path of the current working directory for the application. The last character of the directory will be a slash and the current working directory can be set using the SetWorkingDirectory and SetWorkingDirectoryW functions. GetWorkingDirectory returns UTF-8 string and GetWorkingDirectoryW returns a Unicode string.

Requires 010 Editor v4.0 or higher.

---

void HighlightAllowInstanceSharing(int allowSharing)
Instance sharing is used for Syntax Highlighters to make loading of syntax highlighters faster and use less memory. When *allowSharing* is true then all files in 010 Editor that use this syntax highlighter share a single copy of the compiled template in memory. For example, if instance sharing is turned on for the CPP.bt template, then all files that use that template share a single copy of the template so it does not have to be recompiled every time a new C++ file is opened. Only enable instance sharing if a template calls the HighlightLineRealtime function and only creates local variables.

Requires 010 Editor v9.0 or higher.

---

void HighlightApplyColor(
    int foreColors[],
    int backColors[],
    int start,
    int count,
    int foreColor,
    int backColor)
Given an array of foreground colors *foreColors*, this function sets *count* colors starting at position *start* to the color *foreColor*. *count* colors in the *backColors* array beginning at the *start* index are also set to the color *backColor*. If *foreColor* or *backColor* is -1 then those colors are not applied. This function is similar to

HighlightApplyStyle which uses styles instead of colors.

Requires 010 Editor v9.0 or higher.

---

void HighlightApplyStyle(
int foreColors[],
int backColors[],
int start,
int count,
int styleIndex)
Sets *count* colors in the arrays *foreColors* and *backColors* to the style *styleIndex*, starting at index *start*. If the style has a foreground or background color of -1 (none) then those colors are not applied. See the HighlightFindStyle function to retrive a styleIndex. This function can be used in the HighlightLineRealtime function to apply styles.

Requires 010 Editor v9.0 or higher.

TKeywordList HighlightBuildKeywordList(
int options,
string str1 [, string str2, ...])
This function builds a TKeywordList structure out of the *options* paramater and a list of one or more strings. Once the TKeywordList structure is built it can be used in the HighlightCheckKeywordRule or HighlightMatchKeyword functions to quickly check if text matches a number of different keywords. *options* is a bitmask of HIGHLIGHT_WHOLEWORD and/or HIGHLIGHT_IGNORECASE. If HIGHLIGHT_WHOLEWORD is enabled then partial words cannot match the keyword list and if HIGHLIGHT_IGNORECASE is enabled then the keywords are compared case insensitive.

Requires 010 Editor v9.0 or higher.

***virtual* void HighlightBytesRealtime(**
int64 pos,
uchar bytes[],
int foreColors[],
int backColors[],
int count)
This is a special virtual function. When this function is implemented inside a Binary Template, the function will be called for hex files every time a block of data is to be displayed in the editor. The block of data starts at address *pos* and has *count* bytes which are stored in the array *bytes*. Any colors applied to the *foreColors* or *backColors* arrays will be used to color text in the hex editor. See the HighlightColorPattern function for a method of applying a repeating block of colors to an array of colors. Note that this function can cause performance issues in 010 Editor if the function is very slow and cancelling of execution while this function is running is disabled.

Requires 010 Editor v9.0 or higher.

---

int HighlightCheckCommentRule(
wchar_t text[],
int count,
wstring target,
int &pos,
int foreColors[],
int backColors[],
int styleIndex,
int options=0)
Applies a single-line comment rule to a line of *text* of size *count*. If the *target* string is found at position *pos* then the color *styleIndex* is applied to *foreColors* and *backColors* from *pos* to the end of the line. If the *target* matches then *pos* is set to *count*. See HighlightFindString for a list of possible values for the *options* parameter. Use the HighlightFindStyle function to locate a syntax style. If a match is found this function returns true, otherwise this function returns false.

Requires 010 Editor v9.0 or higher.

int HighlightCheckKeywordRule(
    wchar_t text[],
    int count,
    TKeywordList &keywords,
    int &pos,
    int foreColors[],
    int backColors[],
    int styleIndex,
    int options=0)

This function checks if a set of keywords matches the *text* array of size *count* starting at position *pos*. The *keywords* structure is created with the HighlightBuildKeywordList function. If any of the keywords match then the *foreColors* and *backColors* arrays are colored using the *styleIndex* color. See the HighlightFindStyle function to locate a style. If a match is found this function returns true and *pos* is updated to the next character past the found keyword, otherwise false is returned. See HighlightFindString for a list of possible values for the *options* parameter and note that to perform case-insenstive keyword matches the list must be built with the HIGHLIGHT_IGNORECASE option in HighlightBuildKeywordList.

Requires 010 Editor v9.0 or higher.

int HighlightCheckMultiLineRule(
    wchar_t text[],
    int count,
    wstring startKeyword,
    wstring endKeyword,
    int &pos,
    int &rule,
    int ruleStart,
    int ruleTarget,
    int foreColors[],
    int backColors[],
    int styleIndex,
    int options=0)

Similar to HighlightCheckSingleLineRule except that this function can be used to apply coloring that spans several lines such as a multi-line comment. Multi-line coloring is achieved through the *rule* parameter which is assumed to be saved in the *flags* paramater of the HighlightLineRealtime function. The *text* line of size *count* is investigated and the *styleIndex* color is applied to *foreColors* and *backColors* from the *startKeyword* to the *endKeyword* inclusive. This function has two different modes. If *rule* has the value *ruleStart* then we are not yet coloring according to this rule and *startKeyword* is checked at position *pos*. If *startKeyword* is found the coloring rule becomes active, *rule* is set to *ruleTarget* and coloring is applied until *endKeyword* is found or the end of the line is reached. If *endKeyword* is found then *rule* is set back to *ruleStart* and *pos* is updated to the first character past *endKeyword* otherwise *pos* is set to the end of the line. If HighlightCheckMultiLineRule is called and *rule* is equal to *ruleTarget* then we are currently coloring using this rule. Coloring is applied starting from *pos* until *endKeyword* is found or the end of the line is found similar to the above case. Using the *rule* parameter coloring can be applied to multiple lines but note multi-line functions should be called first in a loop to check if they are active before checking other rules. This function returns true if coloring was applied. See HighlightFindStyle function to locate style indices and see HighlightFindString for a list of possible values for the *options* parameter.

Requires 010 Editor v9.0 or higher.

int HighlightCheckSingleLineRule(
    wchar_t text[],
    int count,
    wstring startKeyword,
    wstring endKeyword,
    int &pos,
    int foreColors[],
    int backColors[],
    int styleIndex,
    int options=0)

Applies syntax highlighting to a single line of *text* of size *count*. The *styleIndex* color is applied to the array *foreColors* and *backColors* if the *startKeyword* string is found at position *pos*. If *startKeyword* is found, coloring is applied from *pos* to the end of the *endKeyword* if *endKeyword* is found or to the end of the line if *endKeyword* is not found. If *startKeyword* is found this function returns true and *pos* is updated to the next character past what was colored, otherwise false is returned. This function can be used to color a single line string if *startKeyword* is '"' and *endKeyword* is '"'. See the HighlightFindStyle function to locate style indices and see HighlightFindString for a list of possible values for the *options* parameter.

Requires 010 Editor v9.0 or higher.

---

int HighlightCheckTagRule(
wchar_t text[],
int count,
wstring startKeyword,
wstring endKeyword,
int &pos,
int &rule,
int ruleStart,
int ruleTarget,
int foreColors[],
int backColors[],
int styleIndex,
int &foundName,
int options=0)

Used to apply coloring to tags in an XML or HTML file but could be used for a variety of other coloring as well. This function applies coloring to just *startKeyword* and *endKeyword* whereas HighlightCheckMultiLineRule colors everything in between as well. Multi-line coloring can be achieved through the *rule* parameter which is assumed to be saved in the *flags* parameter of the HighlightLineRealtime function. If *rule* has the value *ruleStart* the keyword *startKeyword* is checked at the *pos* location. If it is found then it is colored by applying *styleIndex* to the arrays *foreColors* and *backColors*, *rule* is set to *ruleTarget*, *pos* is moved to the first character after the keyword, and *foundName* is set to false. If *rule* has the value *ruleTarget* then the *endKeyword* is checked at the *pos* location. If it is found then coloring is applied using *styleIndex*, *rule* is set to *ruleStart* and *pos* is updated to the first character after the keyword. See the HighlightFindStyle function to locate style indices and see HighlightFindString for a list of possible values for the *options* parameter.

Requires 010 Editor v9.0 or higher.

---

int HighlightCheckTagTokenRule(
wchar_t text[],
int count,
int &pos,
int foreColors[],
int backColors[],
int tagStyle,
int nameStyle,
int attributeStyle,
int &foundName)

Used as a convenience function for coloring text inside a tag, such as used in XML or HTML. The next token is extracted from the *text* of size *count* similar to the HighlightGetNextToken function. The colors corresonding to the token in the *foreColors* and *backColors* arrays are then colored according to one of three possible styles. If the token starts with a letter and *foundName* is false then the token is colored according to the *nameStyle* and *foundName* is set to true. This allows the first name found inside a tag to be colored differently than the other names. If *foundName* is true and the token starts with a letter then the *attributeStyle* is used to color the next, otherwise the *tagStyle* is used to color the text. Use the HighlightFindStyle function to retrieve style indices. This function returns true if a token was found and *pos* is updated to the next character after the end of the token.

Requires 010 Editor v9.0 or higher.

---

void HighlightColorPattern(
int start,
int end,
int foreColors[],
int backColors[],
int count,
int patternSize1, int foreColor1, int backColor1
[, int patternSize2, int foreColor2, int backColor2, ...])

Applies a repeating color pattern to a block of bytes. This function is useful to apply coloring to a hex file when using the HighlightBytesRealtime function. Given an array of colors *foreColors* and *backColors* of size *count*, a pattern is applied starting at the index *start* and ending at *end*. The pattern consists of *patternSize1* bytes of color *foreColor1* and *backColor1*, followed by *patternSize2* bytes of color *foreColor2* and *backColor2*, etc., and repeats until the end is reached. For example:

```
void HighlightBytesRealtime( int64 pos, uchar bytes[],
        int foreColors[], int backColors[], int count )
{
    int start = 18, end = 1278;
    if( (pos < end) && (pos+count >= start) )
        HighlightColorPattern( start-pos, end-pos,
            foreColors, backColors, count,
            6,  -1, cDkBlue,
            12, -1, cDkAqua,
            64, -1, -1 );
}
```

Requires 010 Editor v9.0 or higher.

int HighlightFindString(
wchar_t text[],
int count,
wstring target,
int start,
int &matchlen,
int options=0)

Attempts to locate the string *target* in the array *text* of size *count* characters. The search starts at index *start*. If a find occurrence is found the starting index is the return value of the function and the number of matching characters is returned in the *matchlen* variable. *options* may be a bitmask of the following constants:

- HIGHLIGHT_REGEX - searches with regular expressions
- HIGHLIGHT_IGNORECASE - perform a case-insensitive search
- HIGHLIGHT_CSTRING - when searching for quotes, do not allow \" or \' to end a string
- HIGHLIGHT_XMLSTRING - when searching for quotes, allow '>' to end a string (malformed XML)

Note that HIGHLIGHT_REGEX can currently only be combined with HIGHLIGHT_IGNORECASE.

Requires 010 Editor v9.0 or higher.

int HighlightFindStyle(
string styleName,
int lightForeColor=-1,
int lightBackColor=-1,
int darkForeColor=-1,
int darkBackColor=-1)

Returns the index of the style with name *styleName* in the list of Syntax Styles as located in the Theme/Color Options dialog. This style index can be used with other functions such as HighlightGetStyleForeColor, HighlightGetStyleBackColor, HighlightApplyStyle, HighlightCheckCommentRule, etc. If the *styleName* is not found then a new style is created with the colors *lightForeColor* and *lightBackColor* indicating the color when using a light theme, and *darkForeColor* and *darkBackColor* indicating the color for a dark theme. If *darkForeColor* and *darkBackColor* are both -1 then *lightForeColor* and *lightBackColor* are used for both light and dark colors. Colors should be given in the format 0xGGBBRR or -1 for no color.

Requires 010 Editor v9.0 or higher.

int HighlightGetNextToken(wchar_t text[], int count, int start, int splitAlphaNum=1)

This function is used to divide up a line of text into a series of tokens to check for syntax highlighting. Given a *text* array of size *count*, this function starts scanning at index *start* and returns the index of the first character that has a different character class than the character at *start*. The different character classes are letter, number, symbol, or whitespace. If *splitAlphaNum* is false then letters and numbers are considered the same character class. If the character class never changes then *count* is returned. Use this function to speed up the search for keywords in a line of text since keywords only need to be checked every time the character class changes.

Requires 010 Editor v9.0 or higher.

---

int HighlightGetStyleForeColor(int styleIndex)
Returns the foreground (text) color for a *styleIndex*. The *styleIndex* parameter can be obtained using the HighlightFindStyle function. Colors are in the format 0xGGBBRR and the color -1 indicates no color (clear).

Requires 010 Editor v9.0 or higher.

---

int HighlightGetStyleBackColor(int styleIndex)
Returns the background color for a *styleIndex*. The *styleIndex* parameter can be obtained using the HighlightFindStyle function. Colors are in the format 0xGGBBRR and the color -1 indicates no color (clear).

Requires 010 Editor v9.0 or higher.

***virtual* void HighlightLineRealtime(**
    int64 line,
    wchar_t text[],
    int foreColors[],
    int backColors[],
    int count,
    ushort &flags)
This is a special virtual function. When this function is implemented in a Binary Template then that Template is used for Syntax Highlighting a text file. This function is called every time a line of text needs to be colored in the editor. The line number (starting at 0) is given with the *line* parameter. The text of the line is converted to Unicode characters (wstring) and passed in the *text* parameter as *count* characters. The function is responsible for scanning the text and placing colors into the *foreColors* and *backColors* arrays. *foreColors* holds the foreground (text) color of each character and *backColors* holds the background color of each character.

Any method can be used to apply colors to the arrays but a number of functions are provided to make applying colors easier and these functions all start with 'Highlight'. Syntax Highlighters can use Syntax Styles as described in the Theme/Color Options dialog to apply colors. See the HighlightFindStyle function to locate styles and HighlightApplyStyle to apply the styles. Styles can also be applied using the HighlightCheckCommentRule, HighlightCheckKeywordRule, HighlightCheckMultiLineRule, HighlightCheckSingleLineRule, or HighlightCheckTagRule functions. These functions provide an easier way to apply different types of syntax highlighting rules to text.

The *flags* parameter is a special parameter that allows multi-line syntax highlighting to work. For the first line in the file the value of *flags* is zero. *flags* may be modified by the HighlightLineRealtime function and the value of *flags* at the end of the function is passed to the next line when HighlightLineRealtime is called. For example, using this variable it is possible to keep track if the highlighter is inside a multi-line comment. *flags* is a 16-bit value and a combination of numbers or flags can be stored in this variable using bitmasks.

Note that this function can cause performance issues in 010 Editor if the function is very slow and cancelling of execution while this function is running is disabled.

Requires 010 Editor v9.0 or higher.

---

int HighlightMatchKeyword(
    TKeywordList &keywords,
    wchar_t text[],
    int count,
    int start,
    int options=0)

Given a *keywords* structure as created by HighlightBuildKeywordList, this function returns true if any of the keywords match the *text* array of size *count* starting at position *start*. If a match is found the number of matching characters is returned, otherwise the function returns 0. *options* may be 0 or HIGHLIGHT_WHOLEWORD, in which case partial matches are not returned. Note that to perform case-insenstive keyword matches the list must be built with the HIGHLIGHT_IGNORECASE option in HighlightBuildKeywordList.

Requires 010 Editor v9.0 or higher.

---

void HighlightMatchString(
wchar_t text[],
int count,
wstring target,
int start,
int options=0)

Given a line of text *text* of size *count*, this function checks if the string *target* exists at position *start*. If a match is found the number of matching characters is returned otherwise zero is returned. *options* can be a bitmask of either HIGHLIGHT_REGEX to perform regular expression searches and/or HIGHLIGHT_IGNORECASE to perform case-insensitive searches.

Requires 010 Editor v9.0 or higher.

---

char[] InputDirectory(const char title[], const char defaultDir[]="")

Allows the user to choose a directory using the standard system directory select dialog. The *title* string is displayed in the title bar of the dialog. The *defaultDir* parameter allows setting the initial directory when the dialog is first displayed and if this parameter is null, the last chosen directory will be used. The return value is the directory name in UTF-8 format including a trailing slash, or an empty string if the dialog was cancelled.

Requires 010 Editor v5.0 or higher.

---

double InputFloat(
const char title[],
const char caption[],
const char defaultValue[])

Opens up a dialog with a single edit box. The *title* displays in the title bar of the dialog and the *caption* displays above the edit box. The *defaultValue* specifies the starting value in the edit box. This function returns the floating-point value of the number entered in the edit box. If an invalid number is entered or *Cancel* is pressed, the constant *BAD_VALUE* is returned.

---

int InputNumber(
const char title[],
const char caption[],
const char defaultValue[])

Similar to InputFloat except an integer is returned instead of a float value. If an invalid number is entered or *Cancel* is pressed, the constant *BAD_VALUE* is returned.

---

char[] InputOpenFileName(
char title[],
char filter[]="All files (*)",
char filename[]="")

Shows a standard file open dialog box with the caption *title*. The *filter* controls which file masks are available in the File Type drop-down list. Specify filters as an array of strings separated by the '|' character (the file masks should be inside brackets). The *filename* gives the default file name of the file to open and may just contain a directory to start the dialog in that directory. Only a single file may be selected with the file dialog box. The returned value is the full chosen file name in UTF-8 format, or an empty string if cancel was pressed.

TOpenFileNames InputOpenFileNames(
char title[],
char filter[]="All files (*)",
char filename[]="")

Similar to InputOpenFileName except that multiple files may be selected. The results are returned in a structure TOpenFileNames that contains a *count* variable indicating the number of opened files (zero if cancel was pressed), and an array of *file* variables which each have a UTF-8 *filename* variable indicating the selected file. For example:

```
int i;
TOpenFileNames f = InputOpenFileNames(
    "Open File Test",
    "C Files (*.c *.cpp)|All Files (*)" );
for( i = 0; i < f.count; i++ )
    Printf( "%s\n", f.file[i].filename );
```

will print out all file names selected.

---

int InputRadioButtonBox(
const char title[],
const char caption[],
int defaultIndex,
const char str1[], const char str2[], const char str3[]="",
const char str4[]="", const char str5[]="", const char str6[]="",
const char str7[]="", const char str8[]="", const char str9[]="",
const char str10[]="", const char str11[]="", const char str12[]="",
const char str13[]="", const char str14[]="", const char str15[]="")

Displays a dialog box containing a number of radio buttons which allows a user to pick one choice from a list of options. The *title* string is displayed in the title bar and the *caption* string is displayed above the first radio button. The list of options to be displayed is controlled using the *str1* .. *str15* parameters. Which radio button is chosen by default is controlled with the *defaultIndex* parameter (use a value of -1 to not use a default and note that 0 represents the first item). The return value of this function is the index of the chosen radio button (0 is the first item, 1 the next, etc), or -1 is returned if the dialog was cancelled.

Requires 010 Editor v5.0 or higher.

---

char[] InputSaveFileName(
char title[],
char filter[]="All files (*)",
char filename[]="",
char extension[]="")

Uses a standard file save dialog box to select a file name suitable to use when saving a file. The user will be asked to overwrite a file if it already exists on disk. The *title*, *filter* and *filename* arguments are similar to the InputOpenFileName function. If no extension is given for the file name, a period and the *extension* argument will automatically be appended to the file name. The return value is the full chosen file name in UTF-8 format, or an empty string if cancel was pressed.

---

char[] InputString(
const char title[],
const char caption[],
const char defaultValue[])

Similar to InputFloat except that the string value of the edit box is returned instead of a float value. If *Cancel* is pressed, an empty string is returned. The string is returned in UTF-8 format.

---

wstring InputWString(
const char title[],
const char caption[],
const wstring defaultValue)
Displays a dialog for the user to enter a wide string (unicode string). See the InputFloat function for an explanation of the different arguments. If *Cancel* is pressed, an empty string is returned.

Requires 010 Editor v3.1 or higher.

---

int InsertFile(const char filename[], int64 position)
Inserts all of the bytes in the file given by *filename* into the current file starting at *position*. A negative number is returned if the file cannot be inserted.

---

int IsEditorFocused()
Returns true if a Editor Window is currently focused. This function is useful if you want to build a script that controls the cursor of the Editor Window and only want the cursor to move when the window is focused.

---

int IsModified()
Returns true if any changes have been made to the file, or false otherwise.

---

int IsNoUIMode()
Returns true if 010 Editor is currently in -noui mode, or false otherwise. See Command Line Parameters for more information on -noui mode.

Requires 010 Editor v3.2 or higher.

---

int MessageBox(int mask, const char title[], const char format[] [, argument, ...])
Displays a message box to the user with a number of buttons to press. The buttons displayed depend upon an OR mask of four values. The *Ok*, *Cancel*, *Yes*, or *No* buttons can be displayed by using the constants *idOk*, *idCancel*, *idYes*, or *idNo* respectively (note that not all possible combinations are supported). The *title* is display in the title bar of the message box. The message to display is obtained using a syntax similar to printf (see Printf below). The return value is one of the constants *idOk*, *idCancel*, *idYes*, or *idNo* depending upon which button was pressed. For example, to display a message box with Yes and No buttons use:

```
if( MessageBox( idYes | idNo,
    "Script",
    "Save changes to '%s'?",
    GetFileName() )
      == idYes )
  . . .
```

Note that extended characters can be displayed in the message box by passing strings in UTF-8 format.

---

void OutputPaneClear()
The *Output Pane* refers to the area where text from the Printf function is displayed (located in the Output tab of the Output Window). When this function is called, all previous output in the Output Pane is cleared.

Requires 010 Editor v3.1 or higher.

int OutputPaneSave(const char filename[])
Saves all text in the Output tab of the Output Window to a file on disk. The *filename* argument gives the name of the target output file. This function returns 0 if the file was successfully written or a negative number if the file could not be written.

Requires 010 Editor v3.1 or higher.

void OutputPaneCopy()
Copies the text in the Output tab of the Output Window to the operating system clipboard.

Requires 010 Editor v3.1 or higher.

void PasteFromClipboard()
Inserts any bytes in the currently active clipboard into the file starting at the current cursor position. If a selection has been made, the selected bytes will be deleted before the bytes are inserted. See the SetClipboardIndex function to control which clipboard is active.

int Printf(const char format[] [, argument, ...])
Similar to the standard C printf function. Accepts a format specifier and a series of arguments. The results of the Printf are displayed in the Output tab of the Output Window and the following codes can be used to specify different data types:

- **%d, %i** - signed integer
- **%u** - unsigned integer
- **%x, %X** - hex integer
- **%o** - octal integer
- **%c** - character
- **%s** - string
- **%f, %e, %g** - float
- **%lf** - double
- **%Ld** - signed int64
- **%Lu** - unsigned int64
- **%Lx, %LX** - hex int64

Width, precision, and justification characters are also supported (e.g. "%5.2lf", or "%-15s") and the newline character can be specified with '\n'. This function is different than the standard C printf function because type checking and casting are done on each of the arguments of the function. For example, in this function call the number '5' is automatically cast to a double:

```
Printf( "Num = %d, Float = %lf, Str = '%s'\n", 15, 5, "Test" );
```

The above statement would display "Num = 15, Float = 5.000000, Str = 'Test'". Extended characters can be printed in the Output Window by passing a string in UTF-8 format. Wide strings can be printed with Printf using the regular '%s' specifier because they will automatically be cast to a UTF-8 string. Previous to version 4.0 of 010 Editor, the Printf function did some interpretation of basic HTML tags but this has been removed in version 4.0. Consult documentation on the standard C printf function for more information on different specifiers that can be used.

The Output Window is automatically shown when Printf is called in a Script but is not automatically shown when Printf is called in a Template. This functionality can be controlled via the Compiling Options dialog.

int64 ProcessGetHeapLocalAddress(int index)
Each memory heap in a process is assigned a position in the current file. This function returns the starting local file address for the heap given by *index*. *index* must be between 0 and ProcessGetNumHeaps()-1. If the current file is not a process or if *index* does not refer to a valid heap, -1 is returned. See Editing Processes for more information on processes and heaps.

Requires 010 Editor v3.2 or higher.

---

wchar_t[] ProcessGetHeapModule(int index)
Returns the name of the module to which the given memory heap belongs. The module name is returned as a wide string and *index* must be between 0 and ProcessGetNumHeaps()-1. If the current file is not a process or if *index* does not refer to a valid heap or if the given heap is not associated with a module, an empty string is returned. For more information on heaps and modules see Editing Processes.

Requires 010 Editor v3.2 or higher.

---

int ProcessGetHeapSize(int index)
Returns the size in number of bytes for the heap given by *index*. *index* must be between 0 and ProcessGetNumHeaps()-1. If the current file is not a process or if *index* does not refer to a valid heap, -1 is returned.

Requires 010 Editor v3.2 or higher.

---

int64 ProcessGetHeapStartAddress(int index)
Returns the starting address in memory for the heap given by *index*. *index* must be between 0 and ProcessGetNumHeaps()-1. If the current file is not a process or if *index* does not refer to a valid heap, -1 is returned.

Requires 010 Editor v3.2 or higher.

---

int ProcessGetNumHeaps()
Returns the number of memory heaps for the current process. If the current file is not a process, 0 is returned. See Editing Processes for more information on processes and heaps.

Requires 010 Editor v3.2 or higher.

---

int64 ProcessHeapToLocalAddress(int64 memoryAddress)
Each heap in a process has two addresses: a memory address where the data actually exists in computer memory and a local file address where the data is located for editing in 010 Editor. Given an address *memoryAddress* in system memory, this function returns the equivalent address in the local file. See Editing Processes for more information on processes and heaps.

Requires 010 Editor v3.2 or higher.

---

int64 ProcessLocalToHeapAddress(int64 localAddress)
Each heap in a process has two addresses: a memory address where the data actually exists in computer memory and a local file address where the data is located for editing in 010 Editor. Given an address *localAddress* in the local file, this function returns the equivalent address in system memory. See Editing Processes for more information on processes and heaps.

Requires 010 Editor v3.2 or higher.

---

void RemoveBookmark(int index)

Copyright © 2003-2019 SweetScape Software

Removes a bookmark from the current file. The *index* argument specifies which bookmark to remove and its value should be greater or equal to zero, and less than the number of bookmarks (see GetNumBookmarks). For information on creating bookmarks see AddBookmark.

Requires 010 Editor v3.1 or higher.

---

int RenameFile(const char originalname[], const char newname[])
Renames a file on disk from *originalname* to *newname*. Note that the file should not be open in the editor when it is renamed. A negative number if returned if the rename fails.

---

void RequiresFile()
Scripts can either be run with a target file or without a target file (select "(none)" in the *Run on File* section when editing a Script to run it without a target file). If this function is called and the current script is being run without a target file, a runtime error will be displayed and the script will be stopped. This function should not be called in a Template because Templates must always have a target file.

Requires 010 Editor v4.0 or higher.

---

void RequiresVersion(int majorVer, int minorVer=0, int revision=0)
Indicates what version of 010 Editor is required to execute the current script or template. The execution of the script or template will stop if the current version of 010 Editor is less than the version number given by the *majorVer*, *minorVer*, and *reversion* parameters. For example, with 010 Editor version '3.0.2', the '3' is the major version, '0' is the minor version, and '2' is the revision number. Use this function to ensure that other people that are running your scripts or templates are using the proper version of 010 Editor.

---

void RunTemplate(const char filename[]="", int clearOutput=false)
This function can be called in a Script to execute a Template on the current file. The *filename* argument indicates which Template file to run and the *filename* can either be a full file path, or can be just the name of the file and 010 Editor will attempt to locate the Template using the same rules as locating Include files. For example, 'RunTemplate("ZIPTemplate.bt");' will attempt to locate the "ZIPTemplate.bt" file and execute it on the current file. *filename* can also be an empty string in which case the Template which is associated with the current file will be run (usually this is the last Template that was run on the file). If the Template cannot be located or if there is an error executing the Template, program execution will be stopped. If *clearOutput* is true, the Output tab of the Output Window will be cleared before the Template is started.

Requires 010 Editor v3.1 or higher.
Requires 010 Editor v4.0 or higher for the clearOutput parameter or passing an empty filename.

---

void SetBackColor(int color)
void SetColor(int forecolor, int backcolor)
void SetForeColor(int color)
These functions are used when writing a Template to apply color to different variables. All variables defined after calling one of these functions will be displayed in the given color. SetForeColor sets the foreground (text) color, and SetBackColor sets the background color. Use SetColor to set both the foreground and background color at the same time. A color is an integer made up of 3 hex bytes specifying the blue, green, and red components of the color. The following constants are defined and may be used when setting colors:

- cBlack - 0x000000
- cRed - 0x0000ff
- cDkRed - 0x000080
- cLtRed - 0x8080ff
- cGreen - 0x00ff00
- cDkGreen - 0x008000
- cLtGreen - 0x80ff80

- cBlue - 0xff0000
- cDkBlue - 0x800000
- cLtBlue - 0xff8080
- cPurple - 0xff00ff
- cDkPurple - 0x800080
- cLtPurple - 0xffe0ff
- cAqua - 0xffff00
- cDkAqua - 0x808000
- cLtAqua - 0xffffe0
- cYellow - 0x00ffff
- cDkYellow - 0x008080
- cLtYellow - 0x80ffff
- cDkGray - 0x404040
- cGray - 0x808080,
- cSilver - 0xc0c0c0,
- cLtGray - 0xe0e0e0
- cWhite - 0xffffff
- cNone - 0xffffffff

The cNone constant indicates that no color should be applied. When a dark Theme is used for the editor, any background colors are automatically darkened and see the ThemeAutoScaleColors function for more information. See GetBackColor and GetForeColor to get the value of the currently active color.

void SetBytesPerLine(int bytesPerLine)
When displaying a file in the Hex Editor, calling this function allows overriding the number of bytes per line of the editor. Usually the number of bytes per line comes from the current File Interface and is set using the '*View > Line Width*' menu. Setting a bytesPerLine value of 0 uses the default value from the File Interface. If a value is set with this function and then a new value is chosen with the '*View > Line Width*' menu, the value set with this function will be discarded. Currently the maximum bytes per line accepted is 1024.

Requires 010 Editor v8.0 or higher.

int SetClipboardIndex(int index)
Sets which clipboard is the currently active clipboard. An *index* value of 0 indicates the normal system clipboard and a value of 1 through 9 indicates one of the custom clipboards. All clipboard functions including CopyToClipboard, PasteFromClipboard, etc. operate on the currently active clipboard. See the GetClipboardIndex function to retrieve which clipboard is active and see Using the Clipboard for more information on using multiple clipboards. Note that setting the clipboard index effects the whole application and the selected clipboard will remain active after the script has finished executing. See the Status Bar for a visual indication of which clipboard is active.

Requires 010 Editor v5.0 or higher.

void SetCursorPos(int64 pos)
Sets the cursor position in the current file to *pos*. A flashing caret will visually indicate the cursor position in the file.

int SetEnv(const char str[], const char value[])
Attempts to set the system environment variable indicated by *str* to the given *value* (both *str* and *value* are UTF-8 strings). Note that the environment variable changes are only local to the 010 Editor process, so when 010 Editor is closed the changes will be lost. Returns 0 on success or a negative number on failure.

Requires 010 Editor v5.0 or higher.

int SetFileAttributesUnix(int attributes)
Sets the file attributes of the current Unix or Macintosh file to *attributes*. The *attributes* are a bit flag composed of the constants listed in the GetFileAttributesUnix function. This function returns true if the file is on a Unix or Macintosh operating system and the attributes were set properly, or false otherwise. For example:

```
int flags = GetFileAttributesUnix();
if( flags != FILEATTR_INVALID )
    SetFileAttributesUnix( flags & ~(FILEATTR_READ_USER) );
```

Requires 010 Editor v4.0 or higher.

---

int SetFileAttributesWin(int attributes)
Sets the file attributes of the current Windows file to the given *attributes*. The *attributes* are a bit flag made up of the constants in the GetFileAttributesWin function. This function returns true if the file is a valid Windows file and the attributes were set properly, or returns false otherwise. For example:

```
int flags = GetFileAttributesWin();
if( flags != FILEATTR_INVALID )
    SetFileAttributesWin( flags | FILEATTR_READONLY );
```

Requires 010 Editor v4.0 or higher.

---

int SetFileInterface(const char name[])
Sets the File Interface of the current file to the interface with the given *name*. The current interface of the file can be returned using the GetFileInterface function. The list of valid File Interfaces is found in the Edit As section of the File Bar above each editor (for example: "Hex", "Text", or "Unicode"). This functions returns 0 if the change was successful or a negative number if the Interface could not be found.

Requires 010 Editor v4.0 or higher.

---

void SetMouseWheelScrollSpeed(int speed)
Sets the number of lines that are scrolled each time a scroll wheel on a mouse is clicked forward or backward. The scroll speed is also listed in the Editor Options dialog and the scroll speed can be investigated with the GetMouseWheelScrollSpeed function.

Requires 010 Editor v6.0 or higher.

---

int SetReadOnly(int readonly)
Sets the read-only status of the current file to true or false. A negative number is returned if the read-only status could not be changed.

---

void SetSelection(int64 start, int64 size)
Selects *size* bytes from the file starting at the address *start*. The selected bytes will appear blue in the main window.

---

int SetWorkingDirectory(const char dir[])
int SetWorkingDirectoryW(const wchar_t dir[])
Sets the current working directory of the application to the directory *dir*. The current working directory can be retrieved using the GetWorkingDirectory or GetWorkingDirectoryW functions. This functions returns 0 if the directory is valid and could be set, or a negative number if the directory is not valid.

Requires 010 Editor v4.0 or higher.

---

void Sleep(int milliseconds)
Halts program execution for the given number of *milliseconds*. For example, 'Sleep(2000);' would cause a pause of two seconds. Note that for sleeps of more than 1000 milliseconds the screen is automatically repainted before the sleep occurs.

Requires 010 Editor v3.1 or higher.

---

void StatusMessage(const char format[] [, argument, ...])
Similar to the Printf function except the resulting string is displayed in the Status Bar of the application as a normal status message. See also the Warning function.

Requires 010 Editor v4.0 or higher.

---

void ThemeAutoScaleColors(int autoScale, float scaleFactor=0.5f)
Background colors in a Template can be specified with either the bgcolor special attribute or with the SetBackColor function. When the editor is using a dark Theme the background colors are multiplied by a scale factor which effectively darkens the colors to fit in with the dark theme. To turn off this scaling call the ThemeAutoScaleColors function with *autoScale* set to false. To adjust the scale factor set *autoScale* to true and set the scale factor using *scaleFactor*. Scale factors between 0 and 1.0 are accepted and 0.5 is the default. To check if the current theme is dark see the ThemeIsDark function.

Requires 010 Editor v8.0 or higher.

---

int ThemeIsDark()
Returns true if the current Theme uses a dark color for the Editor background or returns false if the current Theme uses a light color for the Editor background. Note that colors applied using a Template automatically adjust their colors when being used on a dark theme and see the ThemeAutoScaleColors function to control this.

Requires 010 Editor v8.0 or higher.

---

void Terminate(int force=true)
Exits out of the script and then shuts down 010 Editor. If force is true, all open files will be closed and any unsaved modifications will be lost. If force is false and files are modified, the user will be asked if they want to save the files and optionally given a chance to cancel the shut down procedure.

---

void Warning(const char format[] [, argument, ...])
Similar to the Printf function except the resulting string is displayed in the Status Bar of the application and is highlighted orange. This is useful to display an error that occurs in a Template. See also the StatusMessage function.

Related Topics:
Declaring Template Variables
I/O Functions
Inspector Options
Math Functions
Running Templates and Scripts
Status Bar
String Functions
Tool Functions
Using Bookmarks
Using Syntax Highlighting

Copyright © 2003-2013 SweetScape Software - www.sweetscape.com

I/O Functions

The following is a list of input/output functions that can be used when writing Templates or Scripts.

---

void BigEndian()
Indicates that all subsequent reads and writes from the file should use big-endian byte order. This function can be used in a Template to specify the byte order of variables.

---

void BitfieldDisablePadding()
void BitfieldEnablePadding()
These functions control how multiple bitfields are packed into a set of bits. See Bitfields more information on bitfields. Padding is enabled by default.

---

void BitfieldLeftToRight()
void BitfieldRightToLeft()
These functions control how bitfields are packed into a variable. See Bitfields for an introduction to using bitfields. The packing is different depending on if the Template is in big or little endian mode. In little endian mode the default is right-to-left and in big endian mode the default is left-to-right.

---

double ConvertBytesToDouble(uchar byteArray[])
float ConvertBytesToFloat(uchar byteArray[])
hfloat ConvertBytesToHFloat(uchar byteArray[])
These functions take as input an array of hex bytes *byteArray* and returns either the double, float, or hfloat that is represented by those bytes. The *byteArray* parameter must contain at least 8 bytes for the *ConvertBytesToDouble* function, 4 bytes for the *ConvertBytesToFloat* function, or 2 bytes for the *ConvertBytesToHFloat* function. The conversion is performed using the endian for the current file, which can be controlled using the BigEndian or LittleEndian functions. See the ConvertDataToBytes function to convert a double, float, or hfloat to a set of bytes.

Requires 010 Editor v5.0 or higher.

---

int ConvertDataToBytes(*data_type* value, uchar byteArray[])
Given a variable *value* that can be of any of the main data types (e.g. float, short, int, etc.), this function converts the variable to a set of bytes as it would be written to a file and stores the results in the *byteArray* variable. The return value is the number of bytes written to the array. Note that *byteArray* must be large enough to hold the bytes from the conversion. The endian used for the conversion is taken from the current file endian, which can be set using the BigEndian or LittleEndian functions.

Requires 010 Editor v3.1 or higher.

---

void DeleteBytes(int64 start, int64 size)
Deletes *size* bytes from the file, starting at address *start*.

Copyright © 2003-2019 SweetScape Software

int DirectoryExists(string dir)
Returns true if the given directory exists on disk or false if it does not. *dir* should be the full path for a directory.

---

int FEof()
Returns true if the current read position is at the end of the file.

---

int64 FileSize()
Returns the size of the current file in bytes.

TFileList FindFiles(string dir, string filter)
This function scans the given directory *dir* and returns all files that match the *filter*. The filter can contain the wildcard characters * and ? and can contain multiple filters separated by semi-colons (for example, "*.cpp;*.c;*.h"). The results are returned in a TFileList structure which has a *filecount* variable indicating the number of files that match the filter, and an array of *file* variables which each contain a string *filename*. The TFileList also contains a list of sub-directories in the given directory. The *dircount* variable indicates how many sub-directories exist and an array of *dir* variables contains a *dirname* string for each directory. For example:

```
    TFileList fl = FindFiles( "C:\\temp\\", "*.zip" );
    int i;
    Printf( "Num files = %d\n", fl.filecount );
    for( i = 0; i < fl.filecount; i++ )
    {
        Printf( " %s\n", fl.file[i].filename );
    }
    Printf( "\n" );
    Printf( "Num dirs = %d\n", fl.dircount );
    for( i = 0; i < fl.dircount; i++ )
    {
        Printf( " %s\n", fl.dir[i].dirname );
    }
```

---

int FPrintf(int fileNum, char format[], ...)
Performs a Printf starting from *format* and writes the resulting string to the file with index *fileNum*. Use the function GetFileNum to get the index of a file. The string is written at the current read/write position as given by FSeek and then the read/write position is moved forward. Use this function to read data from one file and write the results to another file. This function can also be used in a Template to write data to a different file as the Template is being run (see Permissions). See Printf for more information on format specifiers.

---

int FSeek(int64 pos)
Sets the current read position to the address *pos*. The read position is used when defining variables in a Template. Using this function, bytes can be processed in any order. Returns 0 if successful or -1 if the address was out of range. Use the FTell function to query the current read position.

---

int FSkip(int64 offset)
Moves the current read position ahead by *offset* bytes. *offset* can also be negative to move the read position backwards. The read position is used when defining variables in a template. Using this function, bytes can be

processed in any order. Returns 0 if successful, or -1 if the address was out of range.

---

int64 FTell()
Returns the current read position of the file. This read position is used when defining variables in a Template. Every time a variable is defined in a template, the read position moves ahead the number of bytes used by the variable. See the FSeek and FSkip functions for another way to change the read position, and note that functions like ReadByte do not affect the read position.

---

void InsertBytes(int64 start, int64 size, uchar value=0)
Inserts *size* bytes into the file starting at address *start*. If *start* is at the end of the file, the file will be lengthened by *size* bytes. The value of each inserted byte can be controlled using the *value* parameter. Note that this function or any of the 'Write' functions can be used to lengthen a file. See also the OverwriteBytes function to overwrite data in a file.

Requires 010 Editor v5.0 or higher for the value parameter.

---

int IsBigEndian()
Returns true if the file is being read in big-endian byte order, or false otherwise.

---

int IsLittleEndian()
Returns true if the file is being read in little-endian byte order, or false otherwise.

---

void LittleEndian()
Indicates that all subsequent reads and writes from the file should use little-endian byte order. This function can be used in a Template to specify the byte order of variables.

---

int MakeDir(string dir)
Attempts to create the directory given by *dir*. If any of the parent directories of the given directory do not exist, they will be created too. For example, *MakeDir("C:\\app\\data\\backup\\")* will create the directories "C:\app\" and "C:\app\data\" if necessary. Returns true if the functions succeeds or false otherwise.

---

void OverwriteBytes(int64 start, int64 size, uchar value=0)
Overwrites *size* bytes in the file starting at address *start*. The value of each byte written is controlled by the *value* parameter. If the overwrite goes past the end of the file, the file will automatically be lengthened. See also the InsertBytes function to insert data into a file.

Requires 010 Editor v5.0 or higher.

---

char ReadByte(int64 pos=FTell())
double ReadDouble(int64 pos=FTell())
float ReadFloat(int64 pos=FTell())
hfloat ReadHFloat(int64 pos=FTell())
int ReadInt(int64 pos=FTell())
int64 ReadInt64(int64 pos=FTell())
int64 ReadQuad(int64 pos=FTell())
short ReadShort(int64 pos=FTell())

uchar ReadUByte(int64 pos=FTell())
uint ReadUInt(int64 pos=FTell())
uint64 ReadUInt64(int64 pos=FTell())
uint64 ReadUQuad(int64 pos=FTell())
ushort ReadUShort(int64 pos=FTell())
Returns data read from the file at address *pos*. If no *pos* is given, *pos* defaults to the current read position as reported by FTell. These functions can be used in a Template to read data from a file without declaring a variable and note that these functions do not affect the current read position.

Requires 010 Editor v5.0 or higher for the ReadHFloat function.
Requires 010 Editor v6.0 or higher for the default parameter.

---

char[] ReadLine(int64 pos, int maxLen=-1, int includeLinefeeds=true)
Reads a string from the file starting at address *pos*. Reads characters until a null-character or end-of-line sequence is found. If *maxLen* is greater than zero, the returned string will have at most *maxLen* characters but if *maxLen* is -1 the parameter will be ignored. If *includeLinefeeds* is true the linefeeds will be returned as part of the string, otherwise the linefeed characters will not be included.

Requires 010 Editor v5.0 or higher for the maxLen parameter.
Requires 010 Editor v6.0 or higher for the includeLinefeeds parameter.

---

void ReadBytes(uchar buffer[], int64 pos, int n)
Reads *n* bytes starting from the address *pos* into the character array *buffer*. Note that char[] and uchar[] can be used interchangeably.

---

char[] ReadString(int64 pos, int maxLen=-1)
Reads a string from the file starting at address *pos*. Reads characters until a null-character is found or *maxLen* characters are read. If *maxLen* equals -1, the *maxLen* parameter is ignored.

Requires 010 Editor v5.0 or higher for the maxLen parameter.

---

int ReadStringLength(int64 pos, int maxLen=-1)
Returns the length of a null-terminated string if it were read at the address *pos* in the target file. In other words, this function counts the number of bytes until a null-byte is encountered, starting from address *pos*. The returned length includes space for the null-character. If *maxLen* is greater than zero, the returned string will have at most *maxLen* bytes but if *maxLen* is -1, the file will be scanned until the end-of-file is reached.

Requires 010 Editor v4.0 or higher.

---

wstring ReadWLine(int64 pos, int maxLen=-1)
Reads a wide (unicode) string from the file starting at address *pos*. The string is read until a null-character or end-of-line sequence is encountered. If *maxLen* is greater than zero, the returned string will have at most *maxLen* characters but if *maxLen* is -1 the parameter will be ignored. Note that the endian used for reading is taken from the current file endian, which can be set using the BigEndian or LittleEndian functions.

Requires 010 Editor v3.1 or higher.
Requires 010 Editor v5.0 or higher for the maxLen parameter.

---

wstring ReadWString(int64 pos, int maxLen=-1)
Reads a wide (unicode) string from the file starting at the address *pos*. The string is read until a null-character is encountered or *maxLen* characters are read. If *maxLen* equals -1, the *maxLen* parameter is ignored. Note that the endian used for reading is taken from the current file endian, which can be set using the BigEndian or LittleEndian functions.

Requires 010 Editor v3.1 or higher.

Requires 010 Editor v5.0 or higher for the maxLen parameter.

---

int ReadWStringLength(int64 pos, int maxLen=-1)
Calculates the number of characters in a null-terminated Unicode string if it were read at byte position *pos* in the target file. This is equivalent to counting the number of words (a word is a group of two hex bytes) in the file until a word with zero value is encountered. The returned count includes the null-terminating word. By default this function searches until it reaches the end of the file but can be limited to a set number of characters by setting *maxLen* to a value greater than zero.

Requires 010 Editor v4.0 or higher.

---

int64 TextAddressToLine(int64 address)
Given *address*, the position of a byte within a text file, this function returns the number of the line that contains that byte. Note that lines are numbered starting from 0. If the file is not a text file or the *address* is invalid, -1 is returned. See TextLineToAddress for the inverse operation.

Requires 010 Editor v3.2 or higher.

---

int TextAddressToColumn(int64 address)
Given an *address* of a byte in a file, this function returns the text column where that byte is located. Note that a column number is returned only when using a fixed-width font, otherwise -1 is returned. If the address does not reference a valid byte or the file is not a text file, -1 is also returned.

Requires 010 Editor v3.2 or higher.

---

int64 TextColumnToAddress(int64 line, int column)
Given a line number *line* (note that line numbers start from 0) and a *column*, this function returns the byte address of the character in that column. If the address cannot be determined -1 is returned or if the line contains less than *column* number of columns, the address of the last character on the line is returned.

Requires 010 Editor v4.0 or higher.

---

int64 TextGetNumLines()
Returns the number of lines in the current text file. If the current file is a hex file, -1 is returned.

Requires 010 Editor v3.2 or higher.

---

int TextGetLineSize(int64 line, int includeLinefeeds=true)
Returns the number of bytes that the given *line* contains. The returned size includes the size of the linefeeds if *includeLinefeeds* is true but does not include the size of the linefeeds if *includeLinefeeds* is false. Note that line numbers start at 0 and if an invalid *line* is passed to this function, -1 is returned. If the current file is not a text file, -1 is also returned.

Requires 010 Editor v3.2 or higher.
Requires 010 Editor v6.0 or higher for the includeLinefeeds parameter.

---

int64 TextLineToAddress(int64 line)
Given a *line* number, this function returns the address of the first byte of that line. Note that line numbers start with 0. See TextAddressToLine for the inverse operation.

Requires 010 Editor v3.2 or higher.

---

int TextReadLine(char buffer[], int64 line, int maxsize, int includeLinefeeds=true)

This function reads the byte data from line number *line* and places it into the string *buffer*. Up to *maxsize* bytes will be read from the file and the number of bytes read is returned. If *line* is an invalid line an empty string will be placed into the *buffer*. Note that the returned string will not include linefeed characters if *includeLinefeeds* is false.

Requires 010 Editor v3.2 or higher.
Requires 010 Editor v6.0 or higher for the includeLinefeeds parameter.

---

int TextReadLineW(wchar_t buffer[], int64 line, int maxsize, int includeLinefeeds=true)
Reads bytes from the given *line* and places them into the wide string *buffer*. The full line, up to *maxsize* characters will be read from the file and the number of characters read is returned. Use this function if the current file is a Unicode file, otherwise use the TextReadLine function. The returned buffer will include linefeed characters if *includeLinefeeds* is true and will not include linefeeds if *includeLinefeeds* is false.

Requires 010 Editor v3.2 or higher.
Requires 010 Editor v6.0 or higher for the includeLinefeeds parameter.

---

void TextWriteLine(const char buffer[], int64 line, int includeLinefeeds=true)
Writes the data from the *buffer* to the given *line*, replacing the existing data of that line. Note that line numbers start at 0. If the line to write is longer or shorter than the existing line, the file will be automatically expanded or contracted. If *includeLinefeeds* is true the data to write should contain a linefeed at the end of the line, and if *includeLinefeeds* is false the data should not contain a linefeed.

Requires 010 Editor v3.2 or higher.
Requires 010 Editor v6.0 or higher for the includeLinefeeds parameter.

---

void TextWriteLineW(const wchar_t buffer[], int64 line, int includeLinefeeds=true)
This function writes the wide string from *buffer* to the given text file at line number *line*. Use this function if the current file is a Unicode file, otherwise use the TextWriteLine function. Line numbers start at 0 and if the data to write is longer or shorter than the existing line, the file will be automatically expanded or contracted. *buffer* should contain a linefeed if *includeLinefeeds* is true and *buffer* should not contain linefeeds if *includeLinefeeds* is false.

Requires 010 Editor v3.2 or higher.
Requires 010 Editor v6.0 or higher for the includeLinefeeds parameter.

---

void WriteByte(int64 pos, char value)
void WriteDouble(int64 pos, double value)
void WriteFloat(int64 pos, float value)
void WriteHFloat(int64 pos, float value)
void WriteInt(int64 pos, int value)
void WriteInt64(int64 pos, int64 value)
void WriteQuad(int64 pos, int64 value)
void WriteShort(int64 pos, short value)
void WriteUByte(int64 pos, uchar value)
void WriteUInt(int64 pos, uint value)
void WriteUInt64(int64 pos, uint64 value)
void WriteUQuad(int64 pos, uint64 value)
void WriteUShort(int64 pos, ushort value)
Writes the *value* to the current file at the address *pos*. Note that if bytes are written past the end of the file, the file will automatically be expanded.

Requires 010 Editor v5.0 or higher for the WriteHFloat function.

---

void WriteBytes(const uchar buffer[], int64 pos, int n)
Writes *n* bytes from the array *buffer* to the file at the address *pos*. Note that char[] and uchar[] can be used interchangeably. If bytes are written past the end of the file, the file will automatically be expanded.

---

void WriteString(int64 pos, const char value[])
Writes the string *value* to the current file at address *pos*. Stops when the null-character is reached.

---

void WriteWString(int64 pos, const wstring value)
Writes the wide string *value* to the current file at address *pos* and stops when the null-character is reached. Note that the endian used for writing is taken from the current file endian, which can be set using the BigEndian or LittleEndian functions.

Requires 010 Editor v3.1 or higher.

Related Topics:
Bitfields
Interface Functions
Math Functions
String Functions
Tool Functions

Copyright © 2003-2013 SweetScape Software - www.sweetscape.com

String Functions

The following is a list of string functions that can be used when writing Templates or Scripts.

double Atof(const char s[])
Converts a string to a floating-point number. Returns zero on error.

int Atoi(const char s[])
Converts a string to an integer. Returns zero on error.

int64 BinaryStrToInt(const char s[])
Converts a string containing a binary number *s* to an integer and returns the result. For example:

```
    return BinaryStrToInt( "01001101" );
```

would return the number 77. If the string is not a valid binary string, zero is returned.

Requires 010 Editor v4.0 or higher.

char[] ConvertString(const char src[], int srcCharSet, int destCharSet)
Given a string *src* that uses the character set encoding *srcCharSet*, the string is converted to use the character set encoding *destCharSet* and returned as a string. The following character set constants exist:

- CHARSET_ASCII
- CHARSET_ANSI
- CHARSET_OEM
- CHARSET_EBCDIC
- CHARSET_UNICODE
- CHARSET_MAC
- CHARSET_ARABIC
- CHARSET_BALTIC
- CHARSET_CHINESE_S
- CHARSET_CHINESE_T
- CHARSET_CYRILLIC
- CHARSET_EASTEUROPE
- CHARSET_GREEK
- CHARSET_HEBREW
- CHARSET_JAPANESE
- CHARSET_KOREAN_J
- CHARSET_KOREAN_W
- CHARSET_THAI
- CHARSET_TURKISH
- CHARSET_VIETNAMESE
- CHARSET_UTF8

- CHARSET_ARABIC_ISO
- CHARSET_BALTIC_ISO
- CHARSET_CYRILLIC_KOI8R
- CHARSET_CYRILLIC_KOI8U
- CHARSET_CYRILLIC_ISO
- CHARSET_EASTEUROPE_ISO
- CHARSET_GREEK_ISO
- CHARSET_HEBREW_ISO
- CHARSET_JAPANESE_EUCJP
- CHARSET_TURKISH_ISO

Custom character sets can also be specified using the *ID Number* specified in the Character Set Options dialog. This function should not be used with Unicode character sets (CHARSET_UNICODE). To perform conversions with Unicode strings see the StringToWString and WStringToString functions.

Requires 010 Editor v4.0 or higher.
Requires 010 Editor v9.0 or higher for CHARSET_ARABIC_ISO or greater.

---

string DosDateToString(DOSDATE d, char format[] = "MM/dd/yyyy")
Converts the given DOSDATE into a string and returns the results. By default the date will be in the *format* 'MM/dd/yyyy' but other formats can be used as described in the GetCurrentDateTime function. Click here for more information on the DOSDATE type and see the FileTimeToString function for an example of using SScanf to parse the resulting string.

Requires 010 Editor v4.0 or higher for the format parameter.

---

string DosTimeToString(DOSTIME t, char format[] = "hh:mm:ss")
Converts the given DOSTIME into a string and returns the results. By default the time will be in the *format* 'hh:mm:ss' but other formats can be used as described in the GetCurrentDateTime function. Click here for more information on the DOSTIME type and see the FileTimeToString function for an example of using SScanf to parse the resulting string.

Requires 010 Editor v4.0 or higher for the format parameter.

---

string EnumToString(enum e)
If the given variable *e* is an enum, the value is converted into the string which represents that enum value and returned. The enum may be a constant or an enum variable. For example:

```
    enum { FIRST, SECOND, THIRD } value;
    string s1, s2;
    value = SECOND;
    s1 = EnumToString( THIRD ); //s1 = "THIRD"
    s2 = EnumToString( value ); //s2 = "SECOND"
```

Note that if *e* is a valid enum but no string corresponds to that enum value, an empty string is returned.

---

char[] FileNameGetBase(const char path[], int includeExtension=true)
wchar_t[] FileNameGetBaseW(const wchar_t path[], int includeExtension=true)
When called with a full path name for a file in *path*, this function removes the path name and returns the resulting string. If *includeExtension* is true, the file path will still contain any file extension if it exists, or if false the file extension is removed. For example:

```
     return FileNameGetBase( "C:\\temp\\file.dat" );
```

would return "file.dat", and

```
    return FileNameGetBase( "C:\\temp\\file.dat", false );
```

would return "file". FileNameGetBaseW works in the same way except this function accepts a Unicode string and returns a Unicode string.

Requires 010 Editor v4.0 or higher.

---

char[] FileNameGetExtension(const char path[])
wchar_t[] FileNameGetExtensionW(const wchar_t path[])
Given a file name *path*, this function returns the extension for the file name including the '.'. For example:

```
    return FileNameGetExtension( "C:\\temp\\file.dat" );
```

would return ".dat". FileNameGetExtensionW works the same way except the path is a Unicode string and a Unicode string is returned.

Requires 010 Editor v4.0 or higher.

---

char[] FileNameGetPath(const char path[], int includeSlash=true)
wchar_t[] FileNameGetPathW(const wchar_t path[], int includeSlash=true)
Given a full file name *path*, this function returns just the path portion of the file name. If *includeSlash* is true, the last character in the returned path will be a slash. For example:.

```
    return FileNameGetPath( "C:\\temp\\file.dat" );
```

would return "C:\temp\", and

```
    return FileNameGetBase( "C:\\temp\\file.dat", false );
```

would return "C:\temp". FileNameGetBaseW operates the same way but accepts a Unicode path and returns a Unicode string.

Requires 010 Editor v4.0 or higher.

---

char[] FileNameSetExtension(const char path[], const char extension[])
wchar_t[] FileNameSetExtensionW(const wchar_t path[], const wchar_t extension[])
This function takes as input a file name *path* and an *extension*. The function then removes any existing extension in *path*, appends the new *extension* and then returns the resulting string. Note that *extension* may or may not start with a '.' character and the original *path* argument is not modified. For example:

```
    return FileNameSetExtension( "C:\\temp\\file.dat", "bmp" );
```

would return "C:\temp\file.bmp". Similarly, FileNameSetExtensionW can be used on Unicode strings and a Unicode string is returned.

Requires 010 Editor v4.0 or higher.

---

string FileTimeToString(FILETIME ft, char format[] = "MM/dd/yyyy hh:mm:ss")
Converts the given FILETIME into a string and returns the results. By default the time will be in the *format* 'MM/dd/yyyy hh:mm:ss' but other formats can be used as described in the GetCurrentDateTime function. Click here for more information on the FILETIME type. The resulting string can be separated into parts using the SScanf function. For example:

```
   int hour, minute, second, day, month, year;
```

```
    string s = FileTimeToString( ft );
    SScanf( s, "%02d/%02d/%04d %02d:%02d:%02d",
        month, day, year, hour, minute, second );
    year++;
    SPrintf( s, "%02d/%02d/%04d %02d:%02d:%02d",
        month, day, year, hour, minute, second );
```

Requires 010 Editor v4.0 or higher for the format parameter.

---

char[] IntToBinaryStr(int64 num, int numGroups=0, int includeSpaces=true)
Takes an input an integer *num* and returns that number converted to a binary string. The returned string will contain as many groups of 8 binary digits as are necessary to represent the number, and the minimum number of groups returned can be controlled with the *numGroups* parameter. If *includeSpaces* is true, a space will be included between each group. For example:

```
    return IntToBinaryStr( 1132 );
```

would return "00000100 01101100" and

```
    return IntToBinaryStr( 62, 2, false );
```

would return "0000000000111110".

Requires 010 Editor v4.0 or higher.

---

int IsCharAlpha(char c)
int IsCharAlphaW(wchar_t c)
Returns true if the given character *c* is a letter or false otherwise.

Requires 010 Editor v9.0 or higher.

---

int IsCharNum(char c)
int IsCharNumW(wchar_t c)
Returns true if the given character *c* is a number or false otherwise.

Requires 010 Editor v9.0 or higher.

---

int IsCharAlphaNum(char c)
int IsCharAlphaNumW(wchar_t c)
Returns true if the given character *c* is a letter or number, or false otherwise.

Requires 010 Editor v9.0 or higher.

---

int IsCharSymbol(char c)
int IsCharSymbolW(wchar_t c)
Returns true if the given character *c* is a symbol, or false otherwise.

Requires 010 Editor v9.0 or higher.

---

int IsCharWhitespace(char c)
int IsCharWhitespaceW(wchar_t c)
Returns true if the given character *c* is a space, tab, or linefeed character.

Requires 010 Editor v9.0 or higher.

---

int Memcmp(const uchar s1[], const uchar s2[], int n)
Compares the first *n* bytes of *s1* and *s2*. Returns a value less than zero if *s1* is less than *s2*, zero if they are equal, or a value greater than zero if *s1* is greater than *s2*.

---

void Memcpy(uchar dest[], const uchar src[], int n, int destOffset=0, int srcOffset=0)
Copies a block of *n* bytes from *src* to *dest*. If *srcOffset* is not zero, the bytes are copied starting from the *srcOffset* byte in *src*. If *destOffset* is not zero, the bytes are copied to *dest* starting at the byte *destOffset*. See the WMemcpy function for copying wchar_t data.

Requires 010 Editor v6.0 or higher for the destOffset and srcOffset parameters.

---

void Memset(uchar s[], int c, int n)
Sets the first *n* bytes of *s* to the byte *c*.

---

string OleTimeToString(OLETIME ot, char format[] = "MM/dd/yyyy hh:mm:ss")
Converts the given OLETIME into a string and returns the results. By default the time will be in the *format* 'MM/dd/yyyy hh:mm:ss' but other formats can be used as described in the GetCurrentDateTime function. Click here for more information on the OLETIME type and see the FileTimeToString function for an example of using SScanf to parse the resulting string.

Requires 010 Editor v4.0 or higher for the format parameter.

---

int RegExMatch(string str, string regex);
int RegExMatchW(wstring str, wstring regex);
Attempts to match the Regular Expression *regex* with the string *str*. For the RegExMatch function both strings are assumed to be in ASCII+ANSI format and for the RegExMatchW function both strings are in Unicode format. These functions return 1 if the regular expression completely matches the string, 0 if they do not match, or -1 if *regex* is an invalid regular expression. To match just part of a string see the RegExSearch function. For example, to test if an email address is valid use:

```
if( RegExMatch( "test@test.ca",
    "\\b[A-Za-z0-9.%_+\\-]+@[A-Za-z0-9.\\-]+\\.[A-Za-z]{2,4}\\b" )
    == false )
{
    Warning( "Invalid email address" );
    return -1;
}
```

Requires 010 Editor v6.0 or higher.

---

int RegExSearch(string str, string regex, int &matchSize, int startPos=0);
int RegExSearchW(wstring str, wstring regex, int &matchSize, int startPos=0);
Searches for an occurrence of the Regular Expression *regex* within the string *str*. Use RegExSearch to search ASCII+ANSI strings or the RegExSearchW function to search Unicode strings. These functions return the index of the first matching character in *str* if a match is found, -1 if no match is found, or -2 if the regular expression is invalid. The number of characters in the match will be stored in the *matchSize* parameter. By default the search starts from the first character of *str* but to specify a different starting character use the *startPos* parameter. For example, to search for an IP address within a string use:

```
int result, size;
result = RegExSearch(
    "12:03:23 AM - 192.168.0.10 : www.sweetscape.com/",
    "\\d{1,3}\\.\\d{1,3}.\\d{1,3}.\\d{1,3}", size );
Printf( "Match at pos %d of size %d\n", result, size );
```

This code would display: 'Match at pos 14 of size 12'.

Requires 010 Editor v6.0 or higher.

---

int SPrintf(char buffer[], const char format[] [, argument, ...])
Performs a Printf starting from *format* and places the result into *buffer*. See Printf for more information.

---

int SScanf(char str[], char format[], ...)
This function parses the *str* parameter into a number of variables according to the *format* string. The *format* string uses the same specifiers as the Printf function. Following the format must be a list of arguments, one for each format specifier in the *format* string. Note that unlike the regular C function, do not use '&' for each argument. For example:

```
    int a, b;
    SScanf( "34, 62", "%d, %d", a, b );
```

would read the value 34 and 62 into a and b. The return value will be the number of successfully read arguments (in this example the return value would be 2).

---

void Strcat(char dest[], const char src[])
Appends the characters from *src* to the end of the string *dest*. The string may be resized if necessary. The += operator can also be used for a similar result.

---

int Strchr(const char s[], char c)
Scans the string *s* for the first occurrence of the character *c*. Returns the index of the match, or -1 if no characters match.

---

int Strcmp(const char s1[], const char s2[])
Compares the one string to another. Returns a value less than zero if *s1* is less than *s2*, zero if they are equal, or a value greater than zero if *s1* is greater than *s2*.

---

void Strcpy(char dest[], const char src[])
Copies string *src* to string *dest*, stopping when the null-character has been copied.

---

char[] StrDel(const char str[], int start, int count)
Removes *count* characters from *str* starting at the index *start* and returns the resulting string.

---

int Stricmp(const char s1[], const char s2[])

Identical to Strcmp except the strings are compared without case sensitivity.

---

int StringToDosDate(string s, DOSDATE &d, char format[] = "MM/dd/yyyy")
Converts the given string into a DOSDATE and stores the results in *d*. The format of the date string is given with the *format* parameter and is by default 'MM/dd/yyyy' but other formats can be used as described in the GetCurrentDateTime function. This function returns 0 if it succeeds or a negative number on failure. More information on date types is available here.

Requires 010 Editor v4.0 or higher for the format parameter.

---

int StringToDosTime(string s, DOSTIME &t, char format[] = "hh:mm:ss")
Converts the given string into a DOSTIME and stores the results in *t*. The format of the time string is given with the *format* parameter and is by default 'hh:mm:ss' but other formats can be used as described in the GetCurrentDateTime function. This function returns 0 if it succeeds or a negative number on failure. More information on date types is available here.

Requires 010 Editor v4.0 or higher for the format parameter.

---

int StringToFileTime(string s, FILETIME &ft, char format[] = "MM/dd/yyyy hh:mm:ss")
Converts the given string into a FILETIME and stores the results in *ft*. The format of the time string is given with the *format* parameter and is by default 'MM/dd/yyyy hh:mm:ss' but other formats can be used as described in the GetCurrentDateTime function. This function returns 0 if it succeeds or a negative number on failure. More information on date types is available here.

Requires 010 Editor v4.0 or higher for the format parameter.

---

int StringToOleTime(string s, OLETIME &ot, char format[] = "MM/dd/yyyy hh:mm:ss")
Converts the given string into an OLETIME and stores the results in *ot*. The format of the string is given with the *format* parameter and is by default 'MM/dd/yyyy hh:mm:ss' but other formats can be used as described in the GetCurrentDateTime function. This function returns 0 if it succeeds or a negative number on failure. More information on date types is available here.

Requires 010 Editor v4.0 or higher for the format parameter.

---

int StringToTimeT(string s, time_t &t, char format[] = "MM/dd/yyyy hh:mm:ss")
Converts the given string into a time_t and stores the results in *t*. The format of the string is given with the *format* parameter and is by default 'MM/dd/yyyy hh:mm:ss' but other formats can be used as described in the GetCurrentDateTime function. This function returns 0 if it succeeds or a negative number on failure. More information on date types is available here.

Requires 010 Editor v4.0 or higher for the format parameter.

---

int StringToTime64T(string s, time64_t &t, char format[] = "MM/dd/yyyy hh:mm:ss")
Converts the given string into a time64_t and stores the results in *t*. The format of the string is given with the *format* parameter and is by default 'MM/dd/yyyy hh:mm:ss' but other formats can be used as described in the GetCurrentDateTime function. This function returns 0 if it succeeds or a negative number on failure. More information on date types is available here.

Requires 010 Editor v9.0 or higher.

---

char[] StringToUTF8(const char src[], int srcCharSet=CHARSET_ANSI)
Takes as input a string *src* which uses the character set encoding *srcCharSet*. The string is converted to the UTF-8 character set and returned. The list of character set constants is available in the ConvertString function and this

function is equivalent to 'ConvertString(src, srcCharSet, CHARSET_UTF8);'.

Requires 010 Editor v4.0 or higher.

---

wstring StringToWString(const char str[], int srcCharSet=CHARSET_ANSI)
Converts the given string *str* into a wide (unicode) string. *str* is assumed to be an ANSI string but other character sets can be specified using the *srcCharSet* parameter (see the ConvertString function for a list of character set constants). See Arrays and Strings for information on wide strings and note that *wstring* and *wchar_t[]* are equivalent.

Requires 010 Editor v3.1 or higher.
Requires 010 Editor v4.0 or higher for the srcCharSet parameter.

---

int Strlen(const char s[])
Returns the number of bytes in *s* before the null-character.

---

int Strncmp(const char s1[], const char s2[], int n)
Similar to Strcmp, except that no more than *n* characters are compared.

---

void Strncpy(char dest[], const char src[], int n)
Similar to Strcpy, except that at most *n* characters will be copied.

---

int Strnicmp(const char s1[], const char s2[], int n)
Similar to Strcmp except that at most *n* characters are compared and the characters are compared without case sensitivity.

---

int Strstr(const char s1[], const char s2[])
Scans the string *s1* for the first occurrence of *s2*. Returns the index of the first matching character, or -1 if no match is found.

---

char[] SubStr(const char str[], int start, int count=-1)
Returns a string containing *count* characters from *str* starting at the index *start*. If *count* is -1, all the characters from the *start* index to the end of the string are returned.

---

string TimeTToString(time_t t, char format[] = "MM/dd/yyyy hh:mm:ss")
Converts the given time_t into a string and returns the results. By default the time will be in the *format* 'MM/dd/yyyy hh:mm:ss' but other formats can be used as described in the GetCurrentDateTime function. Click here for more information on the time_t type and see the FileTimeToString function for an example of using SScanf to parse the resulting string.

Requires 010 Editor v4.0 or higher for the format parameter.

---

string Time64TToString(time64_t t, char format[] = "MM/dd/yyyy hh:mm:ss")
Converts the given time64_t into a string and returns the results. By default the time will be in the *format*

'MM/dd/yyyy hh:mm:ss' but other formats can be used as described in the GetCurrentDateTime function. Click here for more information on the time64_t type and see the FileTimeToString function for an example of using SScanf to parse the resulting string.

Requires 010 Editor v9.0 or higher.

---

char ToLower(char c)
wchar_t ToLowerW(wchar_t c)
Takes as input a character *c*, converts the character to lowercase and then returns the result. If the character cannot be converted to lowercase, the unmodified character is returned. For example:

```
return ToLower( 'A' );
```

would return 'a'. The ToLowerW function operates on wide characters and handles converting Unicode characters to lowercase.

Requires 010 Editor v4.0 or higher.

---

char ToUpper(char c)
wchar_t ToUpperW(wchar_t c)
Returns the given character *c* converted to an uppercase character. If the character cannot be converted to uppercase the same character is returned. For example:

```
return ToUpper( 'c' );
```

would return 'C'. Use the ToUpperW function to convert Unicode characters to uppercase.

Requires 010 Editor v4.0 or higher.

---

void WMemcmp(const wchar_t s1[], const wchar_t s2[], int n)
Compares the first *n* wchar_t items of the arrays *s1* and *s2*. This function returns a value less than zero if *s1* is less than *s2*, zero if they are equal, or a value greater than zero if *s1* is greater than *s2*.

Requires 010 Editor v3.1 or higher.

---

void WMemcpy(wchar_t dest[], const wchar_t src[], int n, int destOffset=0, int srcOffset=0)
Copies *n* wchar_t items from the array *src* to the array *dest*. If *srcOffset* is not zero, the bytes are copied starting from the *srcOffset* index in *src*. If *destOffset* is not zero, the bytes are copied to *dest* starting at the index *destOffset*. See the Memcpy function for copying byte data.

Requires 010 Editor v3.1 or higher.
Requires 010 Editor v6.0 or higher for the destOffset and srcOffset parameters.

---

void WMemset(wchar_t s[], int c, int n)
Sets the first *n* wchar_t items of the array *s* to the value *c*.

Requires 010 Editor v3.1 or higher.

---

void WStrcat(wchar_t dest[], const wchar_t src[])
Appends all characters from the *src* string to the end of the *dest* string. Note that the string may be resized if required and the += operator can also be used for a similar result.

Requires 010 Editor v3.1 or higher.

---

int WStrchr(const wchar_t s[], wchar_t c)
Searchs through the string *s* for the first occurrence of the character *c*. If the character is found, this function returns the index of the match, otherwise -1 is returned.

Requires 010 Editor v3.1 or higher.

---

int WStrcmp(const wchar_t s1[], const wchar_t s2[])
Use this function to compare one wide string to another. Returns a value less than zero if *s1* is less than *s2*, zero if they are equal, or a value greater than zero if *s1* is greater than *s2*.

Requires 010 Editor v3.1 or higher.

---

void WStrcpy(wchar_t dest[], const wchar_t src[])
Copies the string *src* to the string *dest*, stopping when the null-character has been copied.

Requires 010 Editor v3.1 or higher.

---

wchar_t[] WStrDel(const whar_t str[], int start, int count)
Returns a string where *count* characters have been removed from the string *str* starting at the index *start*. Note that the *str* argument is not modified.

Requires 010 Editor v3.1 or higher.

---

int WStricmp(const wchar_t s1[], const wchar_t s2[])
Identical to WStrcmp except the strings are compared without case sensitivity.

Requires 010 Editor v3.1 or higher.

---

char[] WStringToString(const wchar_t str[], int destCharSet=CHARSET_ANSI)
Converts the given wide string *str* by default into an ANSI string and returns it. The string can be converted to other character sets using the *destCharSet* parameter (see the ConvertString function for a list of character set constants). Note that not all characters can be successfully converted from wide characters to other character sets and any characters that cannot be converted will be replaced with the '?' character. See Arrays and Strings for information on wide strings and note that *wstring* and *wchar_t[]* are equivalent.

Requires 010 Editor v3.1 or higher.
Requires 010 Editor v4.0 or higher for the destCharSet parameter.

---

char[] WStringToUTF8(const wchar_t str[])
Takes as input a Unicode string *str* which is then converted to the UTF-8 character set and returned as a string.

Requires 010 Editor v4.0 or higher.

---

int WStrlen(const wchar_t s[])
Counts the number of characters in *s* before the null-character is encountered and returns the result.

Requires 010 Editor v3.1 or higher.

---

int WStrncmp(const wchar_t s1[], const wchar_t s2[], int n)

Similar to WStrcmp, except that at most *n* characters are compared between the two strings.

Requires 010 Editor v3.1 or higher.

void WStrncpy(wchar_t dest[], const wchar_t src[], int n)
Similar to WStrcpy, except that at most *n* characters will be copied.

Requires 010 Editor v3.1 or higher.

int WStrnicmp(const wchar_t s1[], const wchar_t s2[], int n)
Similar to WStrcmp except that at most *n* characters are compared and the characters are compared without case sensitivity.

Requires 010 Editor v3.1 or higher.

int WStrstr(const wchar_t s1[], const wchar_t s2[])
Searches through the wide string *s1* for the first occurrence of the string *s2*. If the string is found, the index of the first matching character is returned, otherwise -1 is returned.

Requires 010 Editor v3.1 or higher.

wchar_t[] WSubStr(const wchar_t str[], int start, int count=-1)
Returns a wide string containing *count* characters from *str* starting at the index *start*. If *count* is -1, all the characters from the *start* index to the end of the string are returned.

Requires 010 Editor v3.1 or higher.

Related Topics:
Character Set Options
Interface Functions
I/O Functions
Math Functions
Using Regular Expressions
Tool Functions

Copyright © 2003-2013 SweetScape Software - www.sweetscape.com

Math Functions

The following is a list of math functions that can be used when writing Templates or Scripts.

double Abs(double x)
Returns the absolute value of the float pointer number *x*.

double Ceil(double x)
Returns the smallest integer not less than *x*.

double Cos(double a)
Returns the cosine of the given angle. The angle is given in *degrees*.

double Exp(double x)
Calculates the exponential *e* to the power of *x*.

double Floor(double x)
Returns the highest integer less than or equal to *x*.

double Log(double x)
Calculates the natural logarithm of x. This value is also known as *ln(x)*.

double Max(double a, double b)
Returns the larger of the numbers *a* and *b*.

double Min(double a, double b)
Returns the smaller of the numbers *a* and *b*.

double Pow(double x, double y)
Returns *x* to the power of *y*.

Copyright © 2003-2019 SweetScape Software

int Random(int maximum)
Returns a random integer between 0 and *maximum*-1 inclusive.

---

double Sin(double a)
Computes the sine of the given angle. The angle is given in *degrees*.

---

double Sqrt(double x)
Calculates the positive square-root of the number *x*.

***data_type* SwapBytes(*data_type* x)**
Swaps the bytes of the variable and returns the result. Any of the basic data types can be specified (*byte*, *short*, *int*, *int64*, *float*, or *double*).

---

double Tan(double a)
Calculates the tangent of the given angle. The angle is given in *degrees*.

Related Topics:
Interface Functions
I/O Functions
String Functions
Tool Functions

Copyright © 2003-2013 SweetScape Software - www.sweetscape.com

Tool Functions

The following is a list of functions that allow running many of the tools in the Tools Menu or Search Menu, as well as functions for working with drives and processes.

---

int64 Checksum(
 int algorithm,
 int64 start=0,
 int64 size=0,
 int64 crcPolynomial=-1,
 int64 crcInitValue=-1)
Runs a simple checksum on a file and returns the result as a int64. The *algorithm* can be one of the following constants:

- CHECKSUM_BYTE - Treats the file as a set of unsigned bytes
- CHECKSUM_SHORT_LE - Treats the file as a set of unsigned little-endian shorts
- CHECKSUM_SHORT_BE - Treats the file as a set of unsigned big-endian shorts
- CHECKSUM_INT_LE - Treats the file as a set of unsigned little-endian ints
- CHECKSUM_INT_BE - Treats the file as a set of unsigned big-endian ints
- CHECKSUM_INT64_LE - Treats the file as a set of unsigned little-endian int64s
- CHECKSUM_INT64_BE - Treats the file as a set of unsigned big-endian int64s
- CHECKSUM_SUM8 - Same as CHECKSUM_BYTE except result output as 8-bits
- CHECKSUM_SUM16 - Same as CHECKSUM_BYTE except result output as 16-bits
- CHECKSUM_SUM32 - Same as CHECKSUM_BYTE except result output as 32-bits
- CHECKSUM_SUM64 - Same as CHECKSUM_BYTE
- CHECKSUM_CRC16
- CHECKSUM_CRCCCITT
- CHECKSUM_CRC32
- CHECKSUM_ADLER32

If *start* and *size* are zero, the algorithm is run on the whole file. If they are not zero then the algorithm is run on *size* bytes starting at address *start*. See the ChecksumAlgBytes and ChecksumAlgStr functions to run more complex algorithms. *crcPolynomial* and *crcInitValue* can be used to set a custom polynomial and initial value for the CRC functions. A value of -1 for these parameters uses the default values as described in the Check Sum/Hash Algorithms topic. A negative number is returned on error.

---

int ChecksumAlgArrayStr(
 int algorithm,
 char result[],
 uchar *buffer,
 int64 size,
 char ignore[]="",
 int64 crcPolynomial=-1,
 int64 crcInitValue=-1)
Similar to the ChecksumAlgStr function except that the checksum is run on data stored in an array instead of in a file. The data for the checksum should be passed in the *buffer* array and the *size* parameter lists the number of bytes in the array. The result from the checksum will be stored in the *result* string and the number of characters in the string will be returned, or -1 if an error occurred. See the ChecksumAlgStr function for a list of available algorithms.

Requires 010 Editor v4.0 or higher.

Copyright © 2003-2019 SweetScape Software

---

int ChecksumAlgArrayBytes(
int algorithm,
uchar result[],
uchar *buffer,
int64 size,
char ignore[]="",
int64 crcPolynomial=-1,
int64 crcInitValue=-1)

Similar to the ChecksumAlgStr function except that the checksum is run on data in an array instead of in a file and the results are stored in an array of bytes instead of a string. The data for the checksum should be passed in the *buffer* array and the *size* parameter lists the number of bytes in the array. The result of the checksum operation will be stored as a set of hex bytes in the parameter *result*. The function will return the number of bytes placed in the *result* array or -1 if an error occurred. See the ChecksumAlgStr function for a list of available algorithms.

Requires 010 Editor v4.0 or higher.

---

int ChecksumAlgStr(
int algorithm,
char result[],
int64 start=0,
int64 size=0,
char ignore[]="",
int64 crcPolynomial=-1,
int64 crcInitValue=-1)

Similar to the Checksum algorithm except the following algorithm constants are supported:

- CHECKSUM_BYTE
- CHECKSUM_SHORT_LE
- CHECKSUM_SHORT_BE
- CHECKSUM_INT_LE
- CHECKSUM_INT_BE
- CHECKSUM_INT64_LE
- CHECKSUM_INT64_BE
- CHECKSUM_SUM8
- CHECKSUM_SUM16
- CHECKSUM_SUM32
- CHECKSUM_SUM64
- CHECKSUM_CRC16
- CHECKSUM_CRCCCITT
- CHECKSUM_CRC32
- CHECKSUM_ADLER32
- CHECKSUM_MD2
- CHECKSUM_MD4
- CHECKSUM_MD5
- CHECKSUM_RIPEMD160
- CHECKSUM_SHA1
- CHECKSUM_SHA256
- CHECKSUM_SHA512
- CHECKSUM_TIGER

The *result* argument specifies a string which will hold the result of the checksum. The return value indicates the number of characters in the string, or is negative if an error occurred. Any ranges to ignore can be specified in string format with the *ignore* argument (see Check Sum/Hash Algorithms). The *crcPolynomial* and *crcInitValue* parameters are used to set a custom polynomial and initial value for the CRC algorithms. Specifying -1 for these parameters uses the default values as indicated in the Check Sum/Hash Algorithms help topic. See the Checksum function above for an explanation of the different checksum constants.

---

int ChecksumAlgBytes(
int algorithm,
uchar result[],
int64 start=0,
int64 size=0,
char ignore[]="",
int64 crcPolynomial=-1,
int64 crcInitValue=-1)

This function is identical to the ChecksumAlgStr function except that the checksum is returned as a byte array in the *result* argument. The return value is the number of bytes returned in the array.

TCompareResults Compare(
int type,
int fileNumA,
int fileNumB,
int64 startA=0,
int64 sizeA=0,
int64 startB=0,
int64 sizeB=0,
int matchcase=true,
int64 maxlookahead=10000,
int64 minmatchlength=8,
int64 quickmatch=512)

Runs a comparison between two files or between two blocks of data. The *type* argument indicates the type of comparison that should be run and can be either:

- COMPARE_SYNCHRONIZE (a binary comparison)
- COMPARE_SIMPLE (a byte-by-byte comparison)

fileNumA and *fileNumB* indicate the numbers of the file to compare (see GetFileNum). The file numbers may be the same to compare two blocks in the same file. The *startA*, *sizeA*, *startB*, and *sizeB* arguments indicate the size of the blocks to compare in the two files. If the start and size are both zero, the whole file is used. If *matchcase* is false, then letters of mixed upper and lower cases will match. See Comparing Files for details on the *maxlookahead*, *minmatchlength* and *quickmatch* arguments. The return value is TCompareResults structure with contains a *count* variable indicating the number of resulting ranges, and an array of *record*. Each record contains the variables *type*, *startA*, *sizeA*, *startB*, and *sizeB* to indicate the range. The *type* variable will be one of:

- COMPARE_MATCH=0
- COMPARE_DIFFERENCE=1
- COMPARE_ONLY_IN_A=2
- COMPARE_ONLY_IN_B=3

For example:

```
int i, f1, f2;
FileOpen( "C:\\temp\\test1" );
f1 = GetFileNum();
FileOpen( "C:\\temp\\test2" );
f2 = GetFileNum();
TCompareResults r = Compare( COMPARE_SYNCHRONIZE, f1, f2 );
for( i = 0; i < r.count; i++ )
{
     Printf( "%d %Ld %Ld %Ld %Ld\n",
         r.record[i].type,
         r.record[i].startA,
         r.record[i].sizeA,
         r.record[i].startB,
            r.record[i].sizeB );
    }
```

char ConvertASCIIToEBCDIC(char ascii)
Converts the given ASCII character into an EBCDIC character and returns the result.

void ConvertASCIIToUNICODE(
int len,
const char ascii[],
ubyte unicode[],
int bigendian=false)
Converts an ASCII string into an array of bytes and stores them in the *unicode* argument. *len* indicates the number of characters to convert and the *unicode* array must be of size at least 2*len. If *bigendian* is true, the bytes are stored in big-endian mode, otherwise the bytes are stored in little-endian mode.

void ConvertASCIIToUNICODEW(
int len,
const char ascii[],
ushort unicode[])
Converts an ASCII string into an array of words and stores the array in the *unicode* argument. The number of characters to convert is given by the *len* argument and the *unicode* argument must have size at least len.

char ConvertEBCDICToASCII(char ebcdic)
Converts the given EBCDIC character into an ASCII character and returns the result.

void ConvertUNICODEToASCII(
int len,
const ubyte unicode[],
char ascii[],
int bigendian=false)
Converts an array of UNICODE characters in the *unicode* argument into ASCII bytes and stores them in the *ascii* array. *len* indicates the number of characters to convert. *unicode* must be of size at least size 2*len and *ascii* must be of size at least len. If *bigendian* is true, the bytes are stored in big-endian mode, otherwise the bytes are stored in little-endian mode.

void ConvertUNICODEToASCIIW(
int len,
const ushort unicode[],
char ascii[])
Converts the array of words in the *unicode* argument to ASCII bytes and saves them to the *ascii* argument. The number of characters to convert is given by *len*. *unicode* and *ascii* must be of size at least size len.

int ExportFile(
int type,
char filename[],
int64 start=0,
int64 size=0,
int64 startaddress=0,

int bytesperrow=16,
int wordaddresses=0)
Exports the currently open file to a file on disk given by *filename* using one of the following *type* formats:

- EXPORT_HEXTEXT
- EXPORT_DECTEXT
- EXPORT_BINARYTEXT
- EXPORT_CCODE
- EXPORT_JAVACODE
- EXPORT_INTEL8
- EXPORT_INTEL16
- EXPORT_INTEL32
- EXPORT_S19
- EXPORT_S28
- EXPORT_S37
- EXPORT_TEXT_AREA
- EXPORT_HTML
- EXPORT_RTF
- EXPORT_BASE64
- EXPORT_UUENCODE

The *start* and *size* arguments indicate what portion of the file to export. If they are both zero then the whole file is exported. *startaddress* indicates the starting address that is written to the file for Intel Hex or Motorola formats. *bytesperrow* indicates the number of bytes written on each line of the output file. If *wordaddresses* is true and the export format is Intel Hex, the file will be written using word-based addresses. See Importing/Exporting Files for more information on exporting.

TFindResults FindAll(
<datatype> data,
int matchcase=true,
int wholeword=false,
int method=0,
double tolerance=0.0,
int dir=1,
int64 start=0,
int64 size=0,
int wildcardMatchLength=24)
This function converts the argument *data* into a set of hex bytes and then searches the current file for all occurrences of those bytes. *data* may be any of the basic types or an array of one of the types. If *data* is an array of signed bytes, it is assumed to be a null-terminated string. To search for an array of hex bytes, create an unsigned char array and fill it with the target value. If the type being search for is a string, the *matchcase* and *wholeworld* arguments can be used to control the search (see Using Find for more information). *method* controls which search method is used from the following options:

- FINDMETHOD_NORMAL=0 - a normal search
- FINDMETHOD_WILDCARDS=1 - when searching for strings use wildcards '*' or '?'
- FINDMETHOD_REGEX=2 - when searching for strings use Regular Expressions

wildcardMatchLength indicates the maximum number of characters a '*' can match when searching using wildcards. If the target is a float or double, the *tolerance* argument indicates that values that are only off by the tolerance value still match. If *dir* is 1 the find direction is down and if *dir* is 0 the find direction is up. *start* and *size* can be used to limit the area of the file that is searched. *start* is the starting byte address in the file where the search will begin and *size* is the number of bytes after *start* that will be searched. If *size* is zero, the file will be searched from *start* to the end of the file.

The return value is a TFindResults structure. This structure contains a *count* variable indicating the number of matches, and a *start* array holding an array of starting positions, plus a *size* array which holds an array of target lengths. For example, use the following code to find all occurrences of the ASCII string "Test" in a file:

```
int i;
TFindResults r = FindAll( "Test" );
Printf( "%d\n", r.count );
for( i = 0; i < r.count; i++ )
    Printf( "%Ld %Ld\n", r.start[i], r.size[i] );
```

To search for the floating point value 4.25 with tolerance 0.01, use the following code:

```
float value = 4.25f;
TFindResults r = FindAll( value, true, false, 0, 0.01 );
```

Type Specifiers can be used in a string as an alternate way to specify the type of data to find. For example, the string "15.5,lf" can be used to search for the double 15.5. Using type specifiers is currently the only way to search for hex bytes with wildcards. For example:

```
TFindResults r = FindAll( "FF*6A,h", true, false,
    FINDMETHOD_WILDCARDS );
```

When searching for regular expressions with FindAll, remember to use '\\' to denote a backslash character. For example, to find all numbers with 8 digits use:

```
TFindResults r = FindAll( "\\b\\d{8}\\b", true, false,
    FINDMETHOD_REGEX );
```

Requires 010 Editor v4.0 or higher for the wildcardMatchLength parameter.
Requires 010 Editor v6.0 or higher for method=FINDMETHOD_REGEX.

---

int64 FindFirst(
<datatype> data,
int matchcase=true,
int wholeword=false,
int method=0,
double tolerance=0.0,
int dir=1,
int64 start=0,
int64 size=0,
int wildcardMatchLength=24)
This function is identical to the FindAll function except that the return value is the position of the first occurrence of the target found. A negative number is returned if the value could not be found.

Requires 010 Editor v4.0 or higher for the wildcardMatchLength parameter.
Requires 010 Editor v6.0 or higher for method=FINDMETHOD_REGEX.

TFindInFilesResults FindInFiles(
<datatype> data,
char dir[],
char mask[],
int subdirs=true,
int openfiles=false,
int matchcase=true,
int wholeword=false,
int method=0,
double tolerance=0.0,
int wildcardMatchLength=24)
Searches for a given set of data across multiple files. See the FindAll function for information on the *data*, *matchcase*, *wholeword*, *method*, *wildcardMatchLength* and *tolerance* arguments. The *dir* argument indicates the starting directory where the search will take place. *mask* indicates which file types to search and may contain the characters '*' and '?'. If *subdirs* is true, all subdirectories are recursively searched for the value as well. If *openfiles* is true, only the currently open files are searched. The return value is the TFindInFilesResults structure which contains a *count* variable indicate the number of files found plus an array of *file* variables. Each *file* variable

contains a *count* variable indicating the number of matches, plus an array of *start* and *size* variables indicating the match position. For example:

```
int i, j;
TFindInFilesResults r = FindInFiles( "PK",
    "C:\\temp", "*.zip" );
Printf( "%d\n", r.count );
for( i = 0; i < r.count; i++ )
{
    Printf( "   %s\n", r.file[i].filename );
    Printf( "   %d\n", r.file[i].count );
    for( j = 0; j < r.file[i].count; j++ )
        Printf( "       %Ld %Ld\n",
            r.file[i].start[j],
            r.file[i].size[j] );
}
```

See Using Find In Files for more information.

Requires 010 Editor v4.0 or higher for the wildcardMatchLength parameter.
Requires 010 Editor v6.0 or higher for method=FINDMETHOD_REGEX.

---

int64 FindNext(int dir=1)
This function returns the position of the next occurrence of the target value specified with the FindFirst function. If *dir* is 1, the find direction is down. If *dir* is 0, the find direction is up. The return value is the address of the found data, or -1 if the target is not found.

TFindStringsResults FindStrings(
int minStringLength,
int type,
int matchingCharTypes,
wstring customChars="",
int64 start=0,
int64 size=0,
int requireNull=false)
Attempts to locate any strings within a binary file similar to the Find Strings dialog which is accessed by clicking '*Search > Find Strings*' on the main menu. Specify the minimum length of each string in number of characters with the *minStringLength* parameter. The *type* option tells the algorithm to look for ASCII strings, UNICODE strings or both by using one of the following constants:

- FINDSTRING_ASCII
- FINDSTRING_UNICODE
- FINDSTRING_BOTH

To specify which characters are considered as part of a string, use an OR bitmask ('|') of one or more of the following constants:

- FINDSTRING_LETTERS - the letters A..Z and a..z
- FINDSTRING_LETTERS_ALL - all international numbers including FINDSTRING_LETTERS
- FINDSTRING_NUMBERS - the numbers 0..9
- FINDSTRING_NUMBERS_ALL - all international numbers including FINDSTRING_NUMBERS
- FINDSTRING_SYMBOLS - symbols such as '#', '@', '!', etc. except for '_'
- FINDSTRING_UNDERSCORE - the character '_'
- FINDSTRING_SPACES - spaces or whitespace
- FINDSTRING_LINEFEEDS - line feed characters 0x0a, 0x0d

- FINDSTRING_CUSTOM - include any custom characters in the *customChars* string

Note if the FINDSTRING_CUSTOM constant is included, any characters from *customChars* are considered as part of the string otherwise the *customChars* string is ignored. The *start* and *size* parameters indicate the range of the file to search and if *size* is zero, the file is searched starting from *start* to the end of the file. If *requireNull* is true, the strings must have a null (0) character after each string.

The return value is a *TFindStringsResults* structure which contains a *count* variable with the number of strings found, a *start* array holding the starting position of each string, a *size* array holding the size in bytes of each string, and a *type* array which indicates FINDSTRING_ASCII if the string is an ASCII string or FINDSTRING_UNICODE if the string is a Unicode string. For example, the following code finds all ASCII strings of length at least 5 containing the characters "A..Za..z$&":

```
TFindStringsResults r = FindStrings( 5, FINDSTRING_ASCII,
    FINDSTRING_LETTERS | FINDSTRING_CUSTOM, "$&" );
Printf( "%d\n", r.count );
for( i = 0; i < r.count; i++ )
    Printf( "%Ld %Ld %d\n", r.start[i], r.size[i], r.type[i] );
```

Requires 010 Editor v6.0 or higher.

---

int GetSectorSize()
Returns the size in bytes of the sectors for this drive. If this file is not a drive, the current sector size is defined using the '*View > Division Lines > Set Sector Size*' menu option.

---

int HexOperation(
int operation,
int64 start,
int64 size,
operand,
step=0,
int64 skip=0)
Perform any of the operations on hex data as available in the Hex Operations dialog. The *operation* parameter chooses which operation to perform and these operations are described in the Hex Operations dialog documentation. *start* and *size* indicate which range of bytes to operate on and if *size* is 0, the whole file is used. The *operand* indicates what value to use during the operation and the result is different depending upon which operation is used (see the Hex Operations dialog). *operand* can be any of the basic numeric or floating point types and the type of this parameter tells the function how to interpret the data. For example, if a 'ushort' is passed as an *operand*, the block of data is considered as an array of 'ushort' using the current endian. If *step* is non-zero, the *operand* is incremented by *step* after each operation and if *skip* is non-zero, *skip* number of bytes are skipped after each operation. This function returns the number of bytes modified if successful, or a negative number on error. The following constants can be used for the *operation* parameter:

- HEXOP_ASSIGN
- HEXOP_ADD
- HEXOP_SUBTRACT
- HEXOP_MULTIPLY
- HEXOP_DIVIDE
- HEXOP_NEGATE
- HEXOP_MODULUS
- HEXOP_SET_MINIMUM
- HEXOP_SET_MAXIMUM
- HEXOP_SWAP_BYTES
- HEXOP_BINARY_AND
- HEXOP_BINARY_OR
- HEXOP_BINARY_XOR
- HEXOP_BINARY_INVERT
- HEXOP_SHIFT_LEFT
- HEXOP_SHIFT_RIGHT

- HEXOP_SHIFT_BLOCK_LEFT
- HEXOP_SHIFT_BLOCK_RIGHT
- HEXOP_ROTATE_LEFT
- HEXOP_ROTATE_RIGHT

For example, the following code would treat the bytes from address 16 to 48 as an array of floats and add the value 3.0 to each float in the array:

```
HexOperation( HEXOP_ADD, 16, 32, (float)3.0f );
```

Alternately, the following code would swap all groups of 2 bytes in a file:

```
HexOperation( HEXOP_SWAP_BYTES, 0, 0, (ushort)0 );
```

Requires 010 Editor v4.0 or higher.

---

int64 Histogram(int64 start, int64 size, int64 result[256])
Counts the number of bytes of each value in the file from 0 up to 255. The bytes are counting starting from address *start* and continuing for *size* bytes. The resulting counts are stored in the int64 array *results*. For example, result[0] would indicate the number of 0 bytes values found in the given range of data. The return value is the total number of bytes read.

---

int ImportFile(int type, char filename[], int wordaddresses=false, int defaultByteValue=-1)
Attempts to import the file specified by *filename* in one of the supported import formats. The format is given by the *type* argument and may be:

- IMPORT_HEXTEXT
- IMPORT_DECTEXT
- IMPORT_BINARYTEXT
- IMPORT_SOURCECODE
- IMPORT_INTEL
- IMPORT_MOTOROLA
- IMPORT_BASE64
- IMPORT_UUENCODE

If successful, the file is opened as a new file in the editor. If the function fails, a negative number is returned. If *wordaddresses* is true and the file is an Intel Hex or Motorola file, the file is imported using word-based addressing. When importing some data formats (such as Intel Hex or S-Records) these formats may skip over certain bytes. The value to assign these bytes can be controlled with the *defaultByteValue* parameter and if the parameter is -1, the value from the Importing Options dialog is used. See Importing/Exporting Files for more information on importing.

---

int IsDrive()
Returns true if the current file is a physical or logical drive, or false otherwise (see Editing Drives).

---

int IsLogicalDrive()
Returns true if the current file is a logical drive, or false otherwise (see Editing Drives).

---

int IsPhysicalDrive()

Returns true if the current file is a physical drive, or false otherwise (see Editing Drives).

---

int IsProcess()
Returns true if the current file is a process, or false otherwise (see Editing Processes).

---

int OpenLogicalDrive(char driveletter)
Opens the drive with the given *driveLetter* as a new file in the editor. For example, 'OpenLogicalDrive('c');'. This function returns a negative number on failure. See Editing Drives for more information on drive editing.

---

int OpenPhysicalDrive(int physicalID)
Opens the physical drive *physicalID* as a new file in the editor (see Editing Drives). For example, 'OpenPhysicalDrive(0);'. This function returns a negative number on failure.

---

int OpenProcessById(int processID, int openwriteable=true)
Opens a process identified by the *processID* number (see Editing Processes). If *openwriteable* is true, only bytes that can be modified are opened, otherwise all readable bytes are opened. A negative number if returned if this function fails.

---

int OpenProcessByName(char processname[], int openwriteable=true)
Attempts to open a process given by the name *processname* as a new file in the editor. For example: 'OpenProcessByName("cmd.exe");' If *openwriteable* is true, only bytes that can be modified are opened, otherwise all readable bytes are opened. A negative number if returned if this function fails. See Editing Processes for more information.

---

int ReplaceAll(
<datatype> finddata,
<datatype> replacedata,
int matchcase=true,
int wholeword=false,
int method=0,
double tolerance=0.0,
int dir=1,
int64 start=0,
int64 size=0,
int padwithzeros=false,
int wildcardMatchLength=24)
This function converts the arguments *finddata* and *replacedata* into a set of bytes, and then finds all occurrences of the find bytes in the file and replaces them with the replace bytes. The arguments *matchcase*, *wholeword*, *method*, *wildcardMatchLength*, *tolerance*, *dir*, *start*, and *size* are all used when finding a value and are discussed in the FindAll function above. If *padwithzeros* is true, a set of zero bytes are added to the end of the replace data until it is the same length as the find data. The return value is the number of replacements made.

Requires 010 Editor v4.0 or higher for the wildcardMatchLength parameter.
Requires 010 Editor v6.0 or higher for method=FINDMETHOD_REGEX.

Related Topics:
Check Sum/Hash Algorithms
Comparing Files
Editing Drives

Editing Processes
Importing/Exporting Files
Interface Functions
I/O Functions
Math Functions
Regular Expressions
String Functions
Using Find
Using Find Strings
Using Find in Files

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Introduction to the Repository

The *Repository* is an online collection of Binary Templates and Scripts that have been created by either SweetScape Software or other users of the software. 010 Editor can connect to the Repository, download and install files, and upload files to the Repository which can then be shared to other users. The Repository contains Binary Templates for a variety of formats including AVI, BMP, EXE, ICO, JPG, MP3, OGG, PNG, RAR, TIF, WAV, ZIP, etc. To install Templates or Scripts click either '*Templates > Template Repository*' or '*Scripts > Script Repository*' and see:

- Using the Repository Dialog

When a file is opened in 010 Editor, if a Binary Template is found in the Repository that can help understand data in the file then a dialog is displayed asking to install or ignore the template. For more information see:

- Installing Files on Open from the Repository

Installing a Template or Script from the Repository copies the file to the Repository directory (see the Directory Options dialog) and also adds a record to the list of Installed Templates or Installed Scripts, as stored in the Template Options and Script Options dialogs. For more information see:

- Template Options
- Script Options

Once Templates or Scripts are installed they can be managed from either the Repository Dialog or from the Repository Menu in the File Bar above each Template and Script in the editor. For more information see:

- Using the Repository Menu

The Repository can even store multiple versions of the same file and allows viewing, updating, and merging between versions. For more information see:

- Updating and Merging Files

To submit new files to the Repository or submit an update to an existing file see:

- Submitting Files to the Repository

010 Editor periodically downloads updates from the online Repository. For more information see:

- Updating the Repository

010 Editor even contains a copy of files from the Repository in a Local Repository, meaning some files can be installed from the Repository even when 010 Editor is used on a computer not connected to the internet. The Repository can also be viewed on the SweetScape Software website at 'http://www.sweetscape.com/010editor/repository/'.

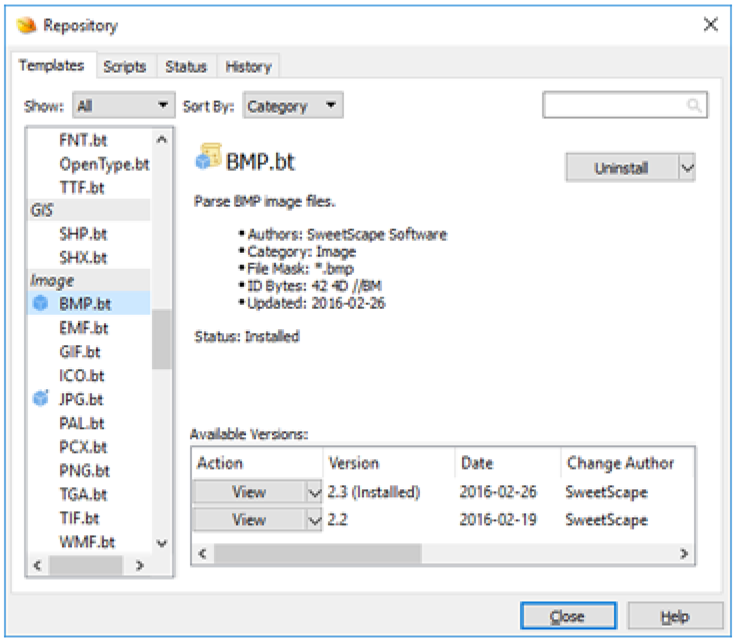
Related Topics:
Directory Options
Installing Files on Open from the Repository
Introduction to Templates and Scripts
Submitting Files to the Repository
Template Options

Updating and Merging Files
Updating the Repository
Using the Repository Dialog
Using the Repository Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using the Repository Dialog

The Repository Dialog is the main way to install and uninstall Templates or Scripts in the Repository. Access the Repository Dialog by clicking either '*Templates > Template Repository*' or '*Scripts > Script Repository*' on the main menu. Once the dialog is displayed, click the *Templates* or *Scripts* tabs at the top of the screen to toggle between the repositories.



Show: All
Sort By: Category

FNT.bt
OpenType.bt
TTF.bt
GIS
SHP.bt
SHX.bt
Image
BMP.bt
EMF.bt
GIF.bt
ICO.bt
JPG.bt
PAL.bt
PCX.bt
PNG.bt
TGA.bt
TIF.bt
WMF.bt

BMP.bt

Parse BMP image files.

- Authors: SweetScape Software
- Category: Image
- File Mask: *.bmp
- ID Bytes: 42 4D //BM
- Updated: 2016-02-26

Status: Installed

Available Versions:

Action	Version	Date	Change Author
View	2.3 (Installed)	2016-02-26	SweetScape
View	2.2	2016-02-19	SweetScape

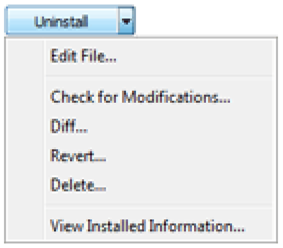
Templates Tab

The Templates tab displays all Binary Templates in the repository in a list along the left side of the dialog. The icon next to a Template name indicates that Template is currently installed. By default all Templates are displayed sorted by *Category*. To display only installed Templates click the *Show* drop-down list and select *Installed* or select *Uninstall* to show those templates that have not been installed. Selecting *Update* from the list shows only those templates that have updates available to install. To display all Templates sorted by file name but without categories click the *Sort By* drop-down list and select *Alphabetic* from the list. Selecting *Newest* from the *Sort By* menu shows the most recently modified Templates at the top of the list.

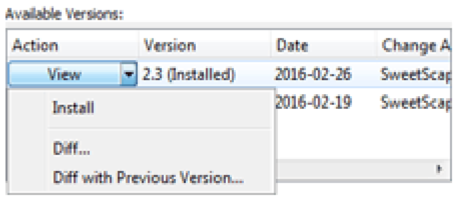
At the top-right of the dialog is a search field marked with the icon . To search for a value within the Repository click this field and type the value. Only those Templates that contain the value, either in the file name, description or file mask, will be displayed in the list of Templates. To cancel the search either delete the find value or click the X icon beside the search field.

Once a Template is selected from the list the information for that Template is displayed in the main part of the dialog. Click the *Install* button or right-click on the Template name in the list of Templates and choose *Install* to install the Template. When a Template is installed it is either downloaded from the online repository or copied from the local repository storage (see Updating the Repository), and then copied into the Template Repository Directory as set in the Directory Options dialog. When a Template is installed a record is added to the list of installed Templates as stored in the Template Options dialog and the Template will appear on the main application *Templates* menu listed under its category.

To uninstall a Template that has been installed either click the *Uninstall* button, right-click on the Template name in the list of Templates and choose *Uninstall*, or delete the Template from the list of installed Templates in the [Template Options]() dialog. When a Template is uninstalled using this dialog the installed file is automatically deleted from the Template Repository Directory unless the Template has been modified, in which case a dialog is displayed giving the option to either Delete or Keep the Template file.



Click the small down arrow to the right of the *Uninstall* button to access the *Repository Menu*. This menu has many of the same options as the [Repository Menu]() located in the File Bar above a Script or Template. Click the *Edit File* menu option to close the Repository dialog and load the installed Template in the editor. Clicking the *View Installed Information* option closes the dialog and displays the information for this Template in the [Template Options]() dialog. The other menu options on this menu are discussed in the [Repository Menu]() help topic.



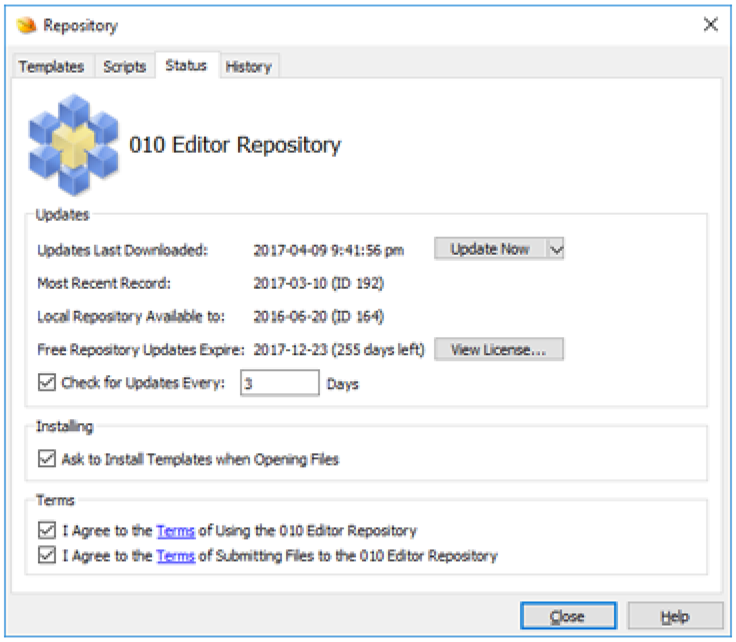
At the bottom of the dialog is the *Available Versions* table which lists all versions of the Template that exist in the Repository. Scroll to the right to view the comments for each version. Some Templates may require a newer version of 010 Editor to be installed as listed in the *Requires Version* column. Clicking the *View* button beside each version loads that version in the editor as a read-only file with the version included in the file name (for example, version 2.3 of ZIP.bt would be loaded as 'ZIP.v2.3.bt'). The file is not installed. To install a specific version click the down arrow to the right of the *View* button and choose *Install* from the popup menu. If modifications have been made to a file then installing a version will display a dialog box which asks to either *Overwrite* the modified file or *Keep* the modified file. Each version can be compared with the currently installed version by clicking the down arrow to the right of *View* and select *Diff* from the menu. To compare a version with the previous (older) version in the table select *Diff with Previous* from the menu.

If a Template is not installed the *Ask to install this template when opening files* toggle is displayed under the *Available Versions* table. If this toggle is enabled and a data file is opened in 010 Editor that matches the *File Mask* field (and the *ID Bytes* field if non-empty), then a dialog is displayed asking to install this Template (see [Installing Files on Open from the Repository]()). If this toggle is not checked then the dialog box will not be displayed.

Scripts Tab

The *Scripts* tab shows all 010 Editor [Scripts]() that can be installed from the Repository. The functionality of this tab is identical to the *Templates* tab as described above except that the *Ask to install this template when opening files* toggle is not shown and 010 Editor will not ask to install Scripts when opening files in the main editor.

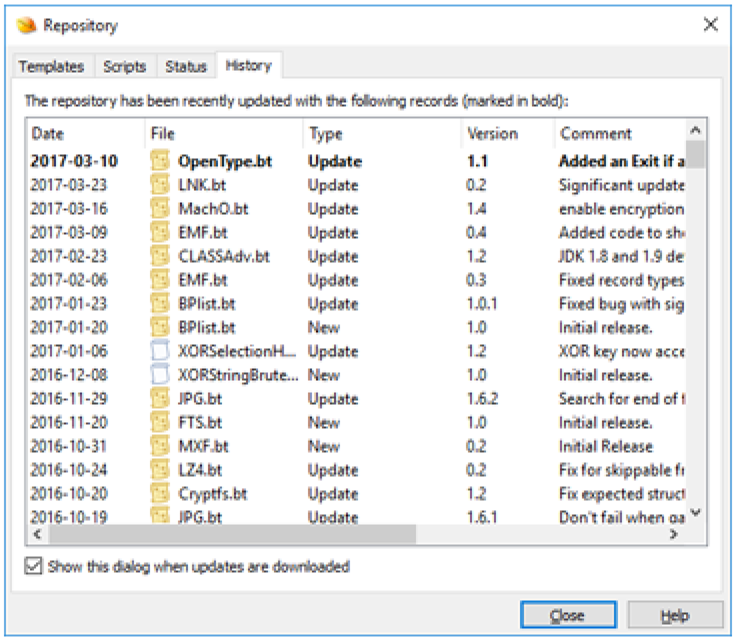
Status Tab



Updates Last Downloaded: 2017-04-09 9:41:56 pm
Most Recent Record: 2017-03-10 (ID 192)
Local Repository Available to: 2016-06-20 (ID 164)
Free Repository Updates Expire: 2017-12-23 (255 days left)
☑ Check for Updates Every: 3 Days
☑ Ask to Install Templates when Opening Files
☑ I Agree to the Terms of Using the 010 Editor Repository
☑ I Agree to the Terms of Submitting Files to the 010 Editor Repository

The current status of the repository is listed in the *Status* tab of the Repository dialog. The *Updates* area contains information about when the Repository was last updated and is discussed in the help topic Updating the Repository. The *Installing* and *Terms* areas are identical to the options available in the Repository Options and are discussed in that help topic.

History Tab


Repository

Templates | Scripts | Status | History

The repository has been recently updated with the following records (marked in bold):

Date	File	Type	Version	Comment
2017-03-10	**OpenType.bt**	**Update**	**1.1**	**Added an Exit if a**
2017-03-23	LNK.bt	Update	0.2	Significant update
2017-03-16	MachO.bt	Update	1.4	enable encryption
2017-03-09	EMF.bt	Update	0.4	Added code to sh
2017-02-23	CLASSAdv.bt	Update	1.2	JDK 1.8 and 1.9 de
2017-02-06	EMF.bt	Update	0.3	Fixed record types
2017-01-23	BPlist.bt	Update	1.0.1	Fixed bug with sig
2017-01-20	BPlist.bt	New	1.0	Initial release.
2017-01-06	XORSelectionH...	Update	1.2	XOR key now acce
2016-12-08	XORStringBrute...	New	1.0	Initial release.
2016-11-29	JPG.bt	Update	1.6.2	Search for end of f
2016-11-20	FTS.bt	New	1.0	Initial release.
2016-10-31	MXF.bt	New	0.2	Initial Release
2016-10-24	LZ4.bt	Update	0.2	Fix for skippable fr
2016-10-20	Cryptfs.bt	Update	1.2	Fix expected struct
2016-10-19	JPG.bt	Update	1.6.1	Don't fail when ga

☑ Show this dialog when updates are downloaded

All Repository updates are listed in a table on the *History* tab of the Repository dialog with the newest updates at the top of the dialog. Scroll to the right to view additional information about each update. Some Templates or Scripts may require a newer version of 010 Editor to be installed as listed in the *Requires Version* column. Double-clicking on a Template or Script will display all the information for that file in either the *Templates* or the *Scripts* tab.

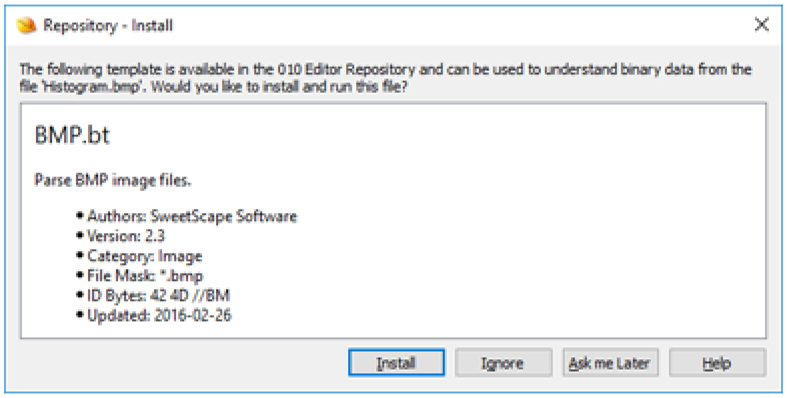
This dialog is displayed by default when new records are downloaded from the online repository (see Updating the Repository). To no longer display this dialog when records are downloaded uncheck the *Show this dialog when updates are downloaded* toggle at the bottom of the dialog.

Related Topics:
Directory Options
Installing Files on Open from the Repository
Introduction to the Repository
Introduction to Templates and Scripts
Repository Options
Template Options
Updating the Repository
Using the Repository Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Installing Files on Open from the Repository

When any file is opened in 010 Editor, 010 Editor checks the file name and the ID bytes at the beginning of the file to see if any Templates exist in the Repository that can parse data in the file. If a Template is found and no Template is already installed to parse the data file (see the Template Options dialog for a list of installed Templates) then the following dialog is displayed:


Repository - Install

The following template is available in the 010 Editor Repository and can be used to understand binary data from the file 'Histogram.bmp'. Would you like to install and run this file?

BMP.bt

Parse BMP image files.

- Authors: SweetScape Software
- Version: 2.3
- Category: Image
- File Mask: *.bmp
- ID Bytes: 42 4D //BM
- Updated: 2016-02-26

Clicking the *Install* button will automatically install and run the displayed Template on the current data file. Sometimes multiple Templates exist in the Repository which can parse the same file. In this case all the Templates will be displayed in the above dialog and select the Template to install by clicking on it before pressing the *Install* button. Alternately, double-clicking on a Template will install and run that file.

Note that some Templates exist in the repository with the File Mask '*' meaning files of this type do not have a specific file extension (an example of this is Linux or macOS executable files). 010 Editor will not automatically display this dialog for Templates that use the '*' mask and these type of Templates must be installed directly using the Repository Dialog.

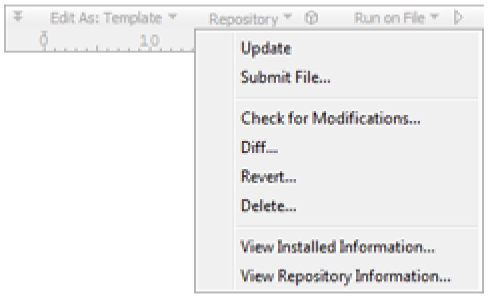
Clicking the *Ignore* button will close the dialog without installing the Template and the dialog will not be displayed again asking to install this Template. In the Repository Dialog the toggle *Ask to install this template when opening files* is displayed at the bottom of each Template page. When the *Ignore* button is clicked this toggle is unchecked for that Template. To no longer ignore a template when opening files, recheck the *Ask to install this template when opening files* box. If multiple Templates are displayed in the above dialog box the *Ignore* button instead displays *Ignore All* and clicking *Ignore All* will ignore all Templates in the list.

Click the *Ask me Later* button to close the dialog without installing any Templates but this dialog will be displayed again the next time a data file is opened that this Template can parse. See the Template Options dialog for more information on the meaning of the File Mask and ID Bytes fields. Other Templates can be installed or uninstalled using the Repository Dialog.

Related Topics:
Introduction to the Repository
Template Options
Using the Repository Dialog

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Using the Repository Menu


Edit As: Template
Repository
Run on File
Update
Submit File...
Check for Modifications...
Diff...
Revert...
Delete...
View Installed Information...
View Repository Information...

The *Repository* Menu appears in the File Bar above the Text Editor for each Template or Script that is opened in 010 Editor (see Working with File Interfaces and make sure *Edit As* is set to either *Template* or *Script*). The Repository Menu changes if the current file has been installed from the Repository. If the current file has not been installed from the Repository then no icon is displayed beside the Repository Menu and the menu only contains two options: *Submit File* which is discussed below, and *View Repository* which opens the Repository Dialog.

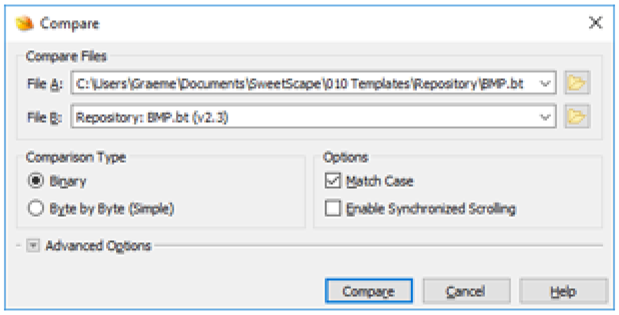
If the current Template or Script has been installed from the Repository then an icon beside the Repository Menu shows its current state:

- - This file has been installed from the Repository and is up to date.
- - This file has been installed from the Repository and has been locally modified.
- - A new version of the file is ready for install from the Repository.
- - A new version of the file is ready for install from the Repository and the file has been locally modified.
- - The file contains conflicts as the result of an Update operation.

Clicking on the icon displays the Repository Dialog. The following menu options are available in the *Repository* menu:

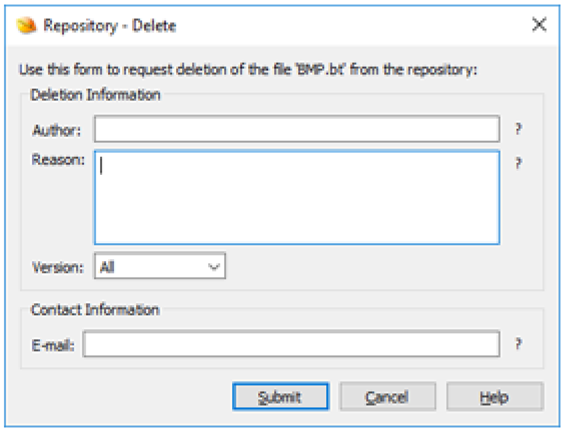
- **Update** - Attempts to install a newer version of the current file from the Repository. Allows a merge to be performed if a new version has been found and the file is locally modified (see Updating and Merging Files).
- **Submit File** - Opens the Submit Dialog to allow uploading either a new Template or Script to the Repository or an update to an existing Template or Script.
- **Check for Modifications** - Either displays *MODIFIED* if the Template or Script does not exactly match the file installed from the Repository, or *Unmodified* if the Template or Script has not been changed. Usually the modification status can be seen from the icon shown beside the Repository menu.
- **Diff** - Opens the Compare Dialog to allow viewing the differences between the current Template or Script and the file that was installed from the Repository (see the dialog below). The down arrows beside the *File A* and *File B* fields can be used to select special entries starting with *Repository:* that indicate the file is read from the Repository. A variety of different comparisons can be made by selecting the different repository versions.

Copyright © 2003-2019 SweetScape Software



File A: C:\Users\Graeme\Documents\SweetScape\010 Templates\Repository\BMP.bt
File B: Repository: BMP.bt (v2.3)
Comparison Type: [✓ Binary] Byte by Byte (Simple)
Options: [✓ Match Case] Enable Synchronized Scrolling

- **Revert** - If the current file has been locally modified then this option discards all changes and returns the file to the version that was installed from the Repository. If no changes have been made this option has no effect.
- **Delete** - Opens a dialog which can be used to request deletion of a file from the Repository. Enter your name or initials in the *Author* field and enter the reason for the deletion in the *Reason* field. This deletion request can either apply to all existing versions for a file by selecting *All* in the *Version* drop-down list or to request just a specific version be deleted select that version in the *Version* list. Enter your e-mail in the *E-mail* field and this information will not be made public but will only be used to contact you if there is an issue with your request.



Use this form to request deletion of the file 'BMP.bt' from the repository:
Version: All

- **View Installed Information** - Views the active settings for the current file as listed in the [Template Options]() dialog for Templates or the [Script Options]() dialog for Scripts. These dialogs display the active *Category*, *File Mask*, *ID Bytes* etc. Note that the Category, File Mask and ID Bytes listed in the comments at the beginning of the file only specify the *default* values when the Template or Script is first installed and to edit the current settings click the *View Installed Information* menu option.
- **View Repository Information** - Views the current Template or Script information in the [Repository Dialog]().

Note that the Repository Menu is also available in the [Repository Dialog]() in a slightly different form by clicking the down-arrow to the right of the *Uninstall* button for an installed Template or Script.

Related Topics:
Comparing Files
Introduction to the Repository
Script Options
Submitting Files to the Repository
Template Options
Updating and Merging Files
Using the Repository Dialog
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Updating and Merging Files

When the Repository has been updated (see Updating the Repository) and a new version of an already installed Template or Script has been found, the special icon appears beside the file in the Templates or Script list of the Repository Dialog. Also, if the Template or Script is open in the editor then the icon appears in the File Bar by the Repository Menu. To install the new version either click the *Update* button in the Repository Dialog or the *Update* menu option in the Repository Menu of the File Bar above each editor. The Repository Dialog can display a list of all avaiable versions of a file in the Repository and indicates which version is currently installed.

Merging and Conflicts

If the local Template or Script has not been modified then installing the update only involves downloading and coping the new file over the old file; however, if the local Template or Script has been modified then a dialog is displayed which gives the option to either *Merge* or *Overwrite* the file. Clicking *Overwrite* discards all the local changes and copies in the new file. Clicking *Merge* attempts to insert all the changes from the Repository into the current working file using a 3-way merge algorithm.

Often merges can be done with no issues; however, problems can occur when a change from the Repository occurs in the same place that a local edit has been made and this is called a *conflict*. For example if the new version in the Repository added the line:

```
struct HDRINFO hdr;
```

to the end of the file but the local Template was already modified to add the line:

```
struct DATARECORD data;
```

to the end of the file. This is a conflict and 010 Editor places both lines into the merged Template marked with the '<<<<<<<', '=======', and '>>>>>>>' tags. For example:

```
<<<<<<<
struct HDRINFO hdr;
=======
struct DATARECORD data;
>>>>>>>
```

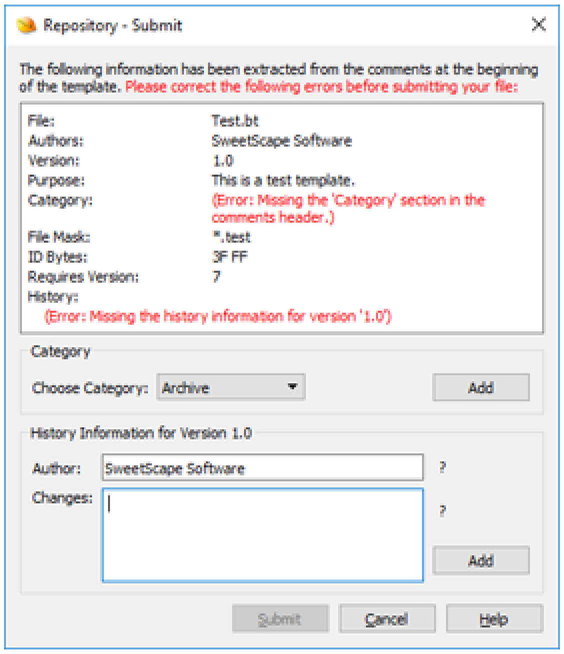
When a conflict occurs the icon appears in the File Bar beside the Repository Menu. To resolve the conflict the user must edit the file and remove the '<<<<<<<', '=======', and '>>>>>>>' tags. Then either delete the first option, delete the second option, or keep both options in some combination. Once the tags have been deleted, saving the file will automatically remove the icon. Before the merge takes place the local Template or Script is copied to a file with the extension *.premerge* and this file can be copied back if problems occur during the merge.

Related Topics:
Introduction to the Repository
Updating the Repository
Using the Repository Dialog
Using the Repository Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Submitting Files to the Repository

New Templates or Scripts or updates to existing Templates or Scripts can be submitted to the Repository by clicking the Repository Menu in the File Bar above each Template or Script and selecting *Submit* from the drop-down menu. Files can also be submitted by e-mailing them to 'support@sweetscape.com' but please try to ensure that the header at the beginning of the file contains the proper information as described below.



The following information has been extracted from the comments at the beginning of the template. Please correct the following errors before submitting your file:

File:	Test.bt
Authors:	SweetScape Software
Version:	1.0
Purpose:	This is a test template.
Category:	(Error: Missing the 'Category' section in the comments header.)
File Mask:	*.test
ID Bytes:	3F FF
Requires Version:	7
History:	

(Error: Missing the history information for version '1.0')

Category
Choose Category: Archive
Add
History Information for Version 1.0
Author: SweetScape Software
Changes:
Add
Submit
Cancel
Help

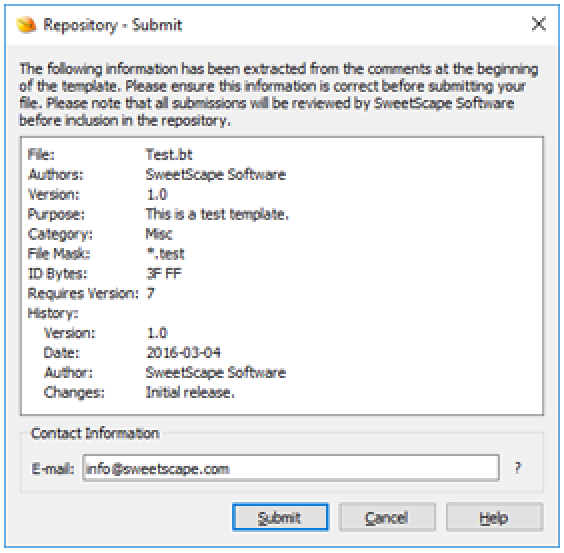
Submissions to the Repository require a special header at the beginning of the file that includes some special information about the file. The *Submit* dialog reads information from the comments to ensure they are in the correct format and any problems must be corrected before the *Submit* button can be clicked. An empty comment header is created every time a new Template or Script is created with the '*Templates* > *New Template*' or '*Scripts* > *New Script*' menu options. An example of the header for a Template is listed below:

```
//------------------------------------------------
//--- 010 Editor v7.0 Binary Template
//
//      File: Test.bt
//   Authors: SweetScape Software
//   Version: 1.0
//   Purpose: This is a test template.
//  Category: Misc
// File Mask: *.test
//  ID Bytes: 3F FF
//   History:
//   1.0   2016-03-04 SweetScape Software: Initial release.
//------------------------------------------------
RequiresVersion(7);
```

Most of this information must be entered using the editor; however, the Submit dialog contains some fields which make adding a *Category* or *History* item easier. The different sections of the header are described below:

- **File** - Must contain the name of the file that is being submitted (e.g. 'Test.bt').
- **Authors** - List the name or the initials of the author of the file. To list an e-mail or website add a separate line to the header marked *E-mail:* or *Website:*.
- **Version** - Lists the current version of the file and versions must contain only numbers or periods '.'. If submitting an update to an existing file, the highest version number is listed in the Submit dialog as *Updates Version* and the current Version must be greater than that version (for example, version 1.5.1 is greater than version 1.5 and version 2 is greater than version 1.5.1).
- **Purpose** - Describe the Template or Script.
- **Category** - Templates and Scripts are listed by category in the Repository. Also when Templates are Scripts are installed they are listed in categories in either the *Templates* or the *Scripts* main menu. If no category is found in the header a *Category* box will appear in the submit dialog. Choose one of the existing categories from the drop-down list and click the *Add* button to insert the category into the header. To use a category that does not yet exist enter the category directly in the editor.
- **File Mask** - *(not required)* For a Template this lists the file names of the data files that this Template can parse. The wildcards '*' (matches 0 to many characters) and '?' (matches exactly 1 character) can be used and multiple masks can be listed separated by commas. For example "*.test,test.A??" would match all files with extension .test or any 3-digit extension starting with A.
- **ID Bytes** - *(not required)* For data files which match the *File Mask* above, the Template will only be run if the hex bytes listed in this field match the bytes at the beginning of the file. This provides an extra check that the data is in a format that can be read and allows multiple Templates with the same File Mask to exist that parse different types of data. Some formats such as Linux or macOS executables do not have an extension and in this case enter the File Mask '*'. ID Bytes uses hex notation for bytes and to skip bytes use the special syntax [+DDD] or [+0xHHH] where DDD is a decimal number or HHH is a hex number. Comments may also be listed in the ID Bytes using '//'. For example the ID Bytes '00 [+4] FF' would match a '00' byte at position 0 in the file and a 'FF' byte at position 5. Currently only the first 2048 bytes of the file are checked for matches. This field can be left blank in which case the Template will be run on all files that match the *File Mask*.
- **History** - Provides a list of changes in each version of the file. On the next line after the *History:* line in the comments should be a line with the current version number, the date in YYYY-MM-DD format, the Author (either name or initials), a colon ':', and then a description of the changes in this version. The Submit dialog provides an easy way to add this information to the Template or Script. Enter information in the *Author* and *Changes* fields in the dialog and then click the *Add* button. If the submission is for a new file please enter "Initial release." in the *Changes* field.
- **Requires Version** - *(not required)* This is a special field which reads the first non-comment line in the Template or Script. If the line calls the RequiresVersion function the values are extracted from that function call. For example "RequiresVersion(3,1);" means that 010 Editor v3.1 or higher is needed to execute this Template. 010 Editor will warn the user when trying to install a file when the RequiresVersion is greater than the current application version. Use the RequiresVersion function when using functions or syntax in Templates or Scripts that is only available in particular versions of 010 Editor.

Once all the information in the header has been verified the *Submit* button will be enabled as shown in the figure below. Enter your e-mail address in the *E-mail* field and this address will not be published and will only be used to contact you if there is an issue with your submission. Click *Submit* to upload your file.



The following information has been extracted from the comments at the beginning of the template. Please ensure this information is correct before submitting your file. Please note that all submissions will be reviewed by SweetScape Software before inclusion in the repository.

```
File:              Test.bt
Authors:           SweetScape Software
Version:           1.0
Purpose:           This is a test template.
Category:          Misc
File Mask:         *.test
ID Bytes:          3F FF
Requires Version:  7
History:
   Version:        1.0
   Date:           2016-03-04
   Author:         SweetScape Software
   Changes:        Initial release.
```

Contact Information

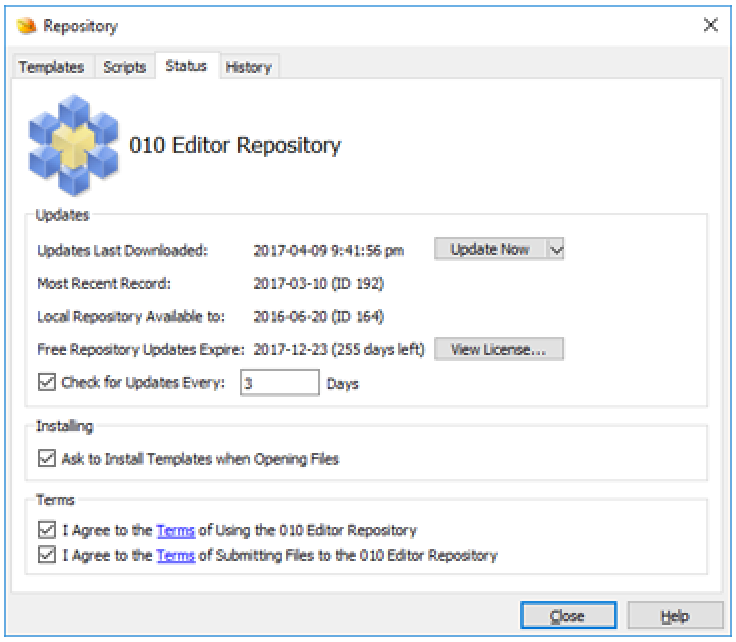
E-mail: info@sweetscape.com

Once your file is uploaded please allow up to 48 hours for the file to be reviewed and added to the Repository. See Updating the Repository to check if your submission has been accepted. If adding a new file to the Repository, install the file from the Repository when it becomes available to ensure that any future update submissions are processed correctly. If adding an update to a file then see Updating and Merging Files to update your file to the new file in Repository.

Related Topics:
Introduction to the Repository
Updating and Merging Files
Updating the Repository
Using the Repository Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Updating the Repository



Updates Last Downloaded: 2017-04-09 9:41:56 pm

Most Recent Record: 2017-03-10 (ID 192)

Local Repository Available to: 2016-06-20 (ID 164)

Free Repository Updates Expire: 2017-12-23 (255 days left)

☑ Check for Updates Every: 3 Days

☑ Ask to Install Templates when Opening Files

☑ I Agree to the Terms of Using the 010 Editor Repository

☑ I Agree to the Terms of Submitting Files to the 010 Editor Repository

By default 010 Editor automatically downloads updates from the online Repository every 3 days. To modify the length of time either click '*Templates > Template Repository*' on the main menu and then select the *Status* tab, or click '*Tools > Options*' on the main menu and locate the *Repository* section. Then enter the number of days between updates in the *Check for Updates Every* field or to disable updates uncheck the *Check for Updates Every* toggle. Note that the minimum time between updates is currently 1 day.

To check if updates are being downloaded successfully locate the *Status* tab of the Repository dialog as shown above and then click the *Update Now* button. If updates fail please check that your internet connection is working and that a firewall or anti-virus program is not blocking access to the internet for 010 Editor. Updates may also fail if the support/maintenance period of your license has expired (usually 1 year from the date of purchase) and see the Repository Licensing Issues section below for more information.

If updates have been downloaded successfully in the past the last update time is listed in the *Updates Last Downloaded* field. The *Most Recent Record* field displays the date of the most recent record downloaded from the repository and the ID is the sequential ID number of the update, starting from 1. The most recent record is also displayed at the top of the History tab.

When updates are successfully downloaded the History tab of the Repository dialog is automatically displayed. To disable this uncheck the *Show this dialog when updates are downloaded* toggle at the bottom of the History tab. Any recent updates to the repository are also displayed on the Startup Page.

Local Repository

The 010 Editor install package currently includes part of the repository as part of the download. This is called the *Local Repository* and means that files from the repository can be installed on computers that are not internet connected (although to download updates an internet connection is required). The *Local Repository Available to* field lists the date and ID of the last record in the Local Repository and any records before that date are available in the Local Repository. Records can be viewed sorted by date on the History tab.

Resetting the Repository

If a problem occurs with the repository then the repository can be reset by clicking the down arrow to the right of the *Update Now* button and choosing *Reset Repository* from the menu. Performing a reset returns the repository to the state when 010 Editor was first installed and only those records from the Local Repository will be listed. After resetting, updates can be downloaded again by clicking the *Update Now* button.

Repository Licensing Issues

When a new license or upgrade license of 010 Editor is purchased the license typically includes free Support, Upgrades and Repository Updates for 1 year from the date of purchase. The date that Support/Upgrades expires is listed in the *Free Repository Updates Expire* field and the date can also be viewed in the Register Dialog. Note that the repository can be updated after the expiry date is past, but the update will only include those records that were uploaded on or before the expiry date. If 010 Editor is being used as a 30-day trial updates can be downloaded until the end of the 30-day trial. If your Support/Updates period is expired see How to Buy 010 Editor for information on purchasing an upgrade.

Related Topics:
How to Buy 010 Editor
Introduction to the Repository
Using the Repository Dialog
Using the Startup Page

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

File Menu

The section lists all available menu options in the File menu:

- **New -** Displays a list of file types that can be created. Click on a file type to create that file in the editor. To control which file types are displayed see the *Manage New File Types* button in the Editor Options dialog.
- **Open File -** Opens an existing file using a standard file dialog box. See Opening Files for more information. The Directory Options dialog allows some control over the initial directory when the file dialog box is opened.
- **Open Drive -** Allows opening of logical or physical drives on your computer for editing. See Editing Drives for details.
- **Open Process -** Allows opening of the memory of a currently running process for editing. See Editing Processes for more information. *(Windows only)*
- **Open Recent -** Displays a list of files which have been recently loaded in the editor.
- **Save -** Saves the current file to disk. If the file was created using New, the file must be given a file name using the standard file dialog box. See Saving Files for more information.
- **Save As -** Allows saving the current file to a different file name on disk. The name of the file will change to the new file name after saving. Using *Save As* on a drive or process will allow saving an image of the drive or process to a file. Use the Directory Options dialog to control the initial directory displayed in the file dialog box.
- **Save a Copy -** Saves a copy of the current file to disk. The current file name will not be modified.
- **Save Selection -** Saves the selected bytes in the file to a new file. The start address and size of the selection will be added to the file name to save in either hex or decimal format, depending upon which format is choosen in the Select Bar.
- **Save All -** Saves all modified files to disk and files marked as read-only will be skipped. Any options specified with the '*Tools > Options...*' dialog box will also be saved to disk.
- **Close -** Closes the current editor window.
- **Close All -** Closes all open editor windows.
- **Revert/Refresh -** Discards all changes for the file and reloads the last copy of the file from disk. If editing a process or a drive, data will be re-read from the process or drive.
- **Special > Rename File -** Renames the current file to a different file name. The file is changed on disk only. Note that the file must be saved before it can be renamed.
- **Special > Delete File -** Closes the current file and removes the file from the disk. Use with caution, as the file will be permanently deleted.
- **Special > E-mail File -** Opens the default e-mail program (if available) to send the current file as an attachment. *(Windows only)*
- **Import Hex -** Converts data from one of the supported data formats into a binary file for editing (see Importing/Exporting Files for more information).
- **Export Hex -** Converts the current file to a different format and saves it to disk (see Importing/Exporting Files for more information).
- **Print -** Opens the Print Dialog for sending the current document to the printer (see Printing).
- **Print Preview -** Generates a preview of what the current document would look like if sent to the printer (see Print Preview).
- **Page Setup -** Sets up various options for printing the document including margins, headers, footers, orientation, and font (see Page Setup for more information).
- **Exit -** Closes all files and exits the program.

Related Topics:
Directory Options
Editing Processes
Editing Drives
Editor Options
File Interface Options
General Options

Copyright © 2003-2019 SweetScape Software

Importing/Exporting Files
Opening Files
Page Setup
Print Preview
Printing
Saving Files

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Edit Menu

The section lists all available menu options in the Edit menu:

- **Undo -** Reverts the current file to the state before the last editing operation. The menu caption will change depending upon which operation was last performed. For example, if a block was pasted into the file, the menu will read '*Undo Paste*'. All commands are stored on a stack so clicking Undo multiple times will undo the last editing operations in reverse order.
- **Redo -** If an Undo operation was performed, the Redo menu option will perform the edit again (Redo is the opposite of Undo). The title of this menu option will change to indicate which editing operation will be redone (for example: '*Redo Edit*', or '*Redo Script*'). Click Redo multiple times will apply the last editing operations on the Undo stack in order.
- **Cut -** If a set of bytes is selected in the editor, this menu option will remove the bytes from the file and place them on a clipboard (see Using the Clipboard for more information). Note that when editing drives or processes, the file size cannot be changed so *Copy* must be used instead of *Cut*.
- **Copy -** The Copy menu option is similar to the Cut command except the selected bytes are copied to the clipboard but not removed from the file.
- **Copy As > Copy as Hex Text -** Copies the current selection to the clipboard but converts the data from hex bytes to characters. For example, if the bytes 00 and FF were selected, '*Copy As Hex Text*' would copy the characters '0', '0', 'F', and 'F' to the clipboard. This command is useful for copying binary data between 010 Editor and a text-only editor.
- **Copy As > Copy As (*export_type*) -** This set of options allows copying the selection to the clipboard in any of the available export types (e.g HTML, RTF, Intel-Hex, etc.). Selecting one of these options exports the selection and then copies the results to the clipboard for easy transfer to another application. For example, select a set of bytes, click the '*Copy as Web Page (HTML)*' menu option, switch to Microsoft Word or another HTML editor and paste the HTML directly into the application. Note that when the data is exported, all of the options are used from the last time that type of data was exported using the '*File > Export Hex...*' menu option. See Importing/Exporting Files for an explanation of all export types.
- **Paste -** If data has been copied onto the clipboard, the Paste command has two possible effects: When editing a text file or when in *Insert* mode (*INS* will appear in the Status Bar) Paste will insert the bytes on the clipboard into the file at the cursor position. When editing hex data in *Overwrite* mode (*OVR* will appear in the Status Bar) the bytes in the clipboard are pasted over the bytes in the file. If any bytes are currently selected in the file, those bytes will be deleted before the paste operation. To switch between *Insert* and *Overwrite* mode, use the *Insert* key. Note that the functionality of Paste can be controlled through the Hex Editor Options dialog (see Hex Editor Options for more information). When working with drives or processes, the file size cannot be changed so *Paste* will only work if the file size remains unchanged.
- **Paste Special -** Some applications paste data to the clipboard in a number of different formats. The Paste Special command allows inserting of data to the current document in any of the available formats. See Using Paste Special for more information.
- **Paste From > Paste from Hex Text -** This command is similar to the Paste command except that data is automatically converted from characters to hex bytes. For example, if the string "3f45" was copied onto the clipboard, the bytes 3f and 45 would be pasted into the file (note that extra characters included spaces are ignored). *Paste From Hex Text* is useful when copying binary data from other applications, such as a text editor.
- **Paste From > Paste from (*import_type*) -** This set of commands provides an easy way to import information from data that is currently on the clipboard. Using one of these commands is the same as clicking the '*File > Import Hex...*' menu option to import data, except that the data is read from the clipboard instead of a file. See Importing/Exporting Files for a list of all import types.
- **Delete -** If a selection is made in the current file, the *Delete* menu option will remove the selected bytes from the file. When working with a drive or process, the file size is fixed so bytes cannot be deleted from the file.
- **Clipboard > (*Clipboard List*) -** 010 Editor has a total of 10 possible clipboards, which includes the standard Windows clipboard plus 9 custom clipboards. All *Cut*, *Copy*, and *Paste* commands operate on the active clipboard (see Using the Clipboard for more information). The Clipboard menu indicates which clipboard is currently active by placing a check mark beside the clipboard name. Other clipboards can be selected by clicking on the list. The current clipboard is also displayed in the Status Bar.

Copyright © 2003-2019 SweetScape Software

- **Clipboard > Clear All Clipboards -** Clears the data on all of the available clipboards. This command is useful to remove large blocks of memory on the clipboard that are no longer needed.
- **Select All -** Selects all bytes in the current file.
- **Select Range -** Shows the Select Bar at the bottom of the current editor that can be used to select a set of bytes in a file (see Selecting a Range for more information). If a selection is already made, the Select Bar displays the start address and size of the selection.
- **Insert/Overwrite > Insert File -** Opens a file dialog box that can be used to insert a file into the editor at the current cursor position. See Inserting or Overwriting Files for more information. Files cannot be inserted into a drive or process since the file size is fixed.
- **Insert/Overwrite > Insert Bytes -** Displays the Insert Bytes dialog that is used to insert a set of bytes into the current file. See Inserting or Overwriting Bytes for more information. Bytes cannot be inserted into a drive or process since the file size is fixed.
- **Insert/Overwrite > Overwrite File -** Similar to the '*Insert Files*' command except that once a file is chosen using the standard file dialog box, data from the selected file is written over the current file starting from the cursor position. See Inserting or Overwriting Files for more information. This operation can be used to write data to a drive or a process.
- **Insert/Overwrite > Overwrite Bytes -** Sets all selected bytes to a single byte value. See Inserting or Overwriting Bytes for more information.
- **Insert Color -** Opens a standard color selection dialog to select a color. After selecting a color and clicking *OK* the color is converted to a text string and inserted into the file at the current cursor position. By default the HTML color text format '#RRGGBB' is used but other formats can be specified with the Text Editor Options dialog. Note that 'RR', 'GG', and 'BB' represent the red, green, and blue color components in hex notation respectively.
- **Insert Date/Time -** Converts the current date and time into a string and inserts the string at the current cursor position. The format of the date string can be controlled via the Text Editor Options dialog.
- **Set File Size -** Allows setting the exact size of the current file through the Set File Size dialog (see Setting the File Size). The file size cannot be modified for drives or processes.
- **Read Only -** Marks that no edits should be made to the current file. When set, '*Read Only*' will appear after the file name in the title bar and a lock icon will appear beside the file name in the File Tabs.
- **Keep File Time -** Marks that the file timestamp should not be changed when the file is saved to disk. '*Keep Time*' will appear in the title bar when and a clock icon will appear on the File Tab beside the file name. *Keep File Time* only works for regular files and not for drives or processes.
- **Properties -** Shows the Properties dialog, displaying information about the current file, drive, or process.

Related Topics:
File Properties
Hex Editor Options
Importing/Exporting Files
Inserting or Overwriting Bytes
Inserting or Overwriting Files
Introduction to the Data Engine
Selecting a Range
Setting the File Size
Status Bar
Text Editor Options
Using the Clipboard
Using Paste Special

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Search Menu

The section lists all available menu options in the Search menu:

- **Find -** Opens the Find Bar below the current editor to search a file for a string or data type. See Using Find for more information.
- **Find Next -** Searches for the next occurrence of a target value in the current file using the last performed Find operation from the Find Bar. Note that if no find operation was performed, the Find Bar will be displayed.
- **Find Previous -** Searches for the previous occurrence of a target value in the current file using the last performed Find operation. If no find operation has been performed, the Find Bar will be displayed.
- **Replace -** Opens the Replace Bar below the current editor that can be used to find and replace strings or data types in the current file. See Using Replace for more information.
- **Replace Next -** Repeats the last Replace operation. Replaces a match of the target value and then searches the file for the next occurrence. If no Replace operation has been performed, the Replace Bar will be shown. A combination of '*Find Next*' and '*Replace Next*' can be used to step through the file, making replacements as necessary.
- **Replace Previous -** Performs the last Replace operation again. Once the replacement has been made the file is then searched for the previous occurrence of the target value. The Replace Bar will be shown if no replacement operation has been performed. Use '*Find Previous*' and '*Replace Previous*' to skip through a file going up, making replacements as necessary.
- **Find Strings -** Attempts to locate strings within a binary files using the Find Strings dialog.
- **Find in Files -** Opens the Find in Files Bar below the editor that can be used to search for a string or data type across multiple files. See Using Find in Files for more information.
- **Replace in Files -** Displays the Replace in Files Bar below the editor that can be used to search and replace a string or data type across multiple files. See Using Replace in Files for more information.
- **Goto -** Shows the Goto Bar below the editor that can be used to set the cursor to a specific address within the current file. See Using Goto for more information.
- **Goto Again -** Repeats the last Goto operation. If the last Goto operation was from the current position (using either '+' or '-'), this command can be used to step through the file. If no Goto operations have been performed, the Goto Bar will be displayed.
- **Add/Edit Bookmark -** Shows the Add/Edit Bookmark dialog that is used to add a bookmark to a file at the current location. If a bookmark already exists at the current cursor location, the bookmark is displayed in the dialog for editing. See Using Bookmarks for more information. Created bookmarks are shown in the *Bookmarks* tab of the Inspector.
- **Toggle Bookmark -** If the cursor in the Editor Window is positioned over an existing bookmark, clicking '*Toggle Bookmark*' will delete the bookmark from the file. If no bookmark exists at the cursor, a quick bookmark will be created at that position. More advanced bookmarks can be created using the '*Add/Edit Bookmark*' option.
- **Next Bookmark -** Moves the cursor to the next bookmark in the file.
- **Previous Bookmark -** Moves the cursor to the previous bookmark in the file.
- **Clear All Bookmarks -** Removes all bookmarks from the current file.
- **Jump to Template Variable -** If a template has been run on the current file, clicking '*Jump to Template Variable*' will try to locate a template variable at the current cursor position. If a template variable can be found, it will be shown either in the *Template Results* panel, or the *Variables* tab of the Inspector, which ever panel was last used. See Working with Template Results for more information.
- **Previous Sector -** When editing a drive, this menu option allows moving the cursor to the previous drive sector. See Editing Drives for more information on sectors.
- **Next Sector -** Moves the cursor to the next drive sector when editing a logical or physical drive. See Editing Drives for more information on sectors.

Related Topics:
Editing Drives
Using Bookmarks
Using Find

Copyright © 2003-2019 SweetScape Software

Using Find in Files
Using Goto
Using Replace
Using Replace in Files
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

View Menu

The View Menu is used for controlling the current File Interface. The File Interface (which includes the Font, Character Set, Linefeeds/Line Width, Tabs, Addresses, Group By, Division Lines, Areas, Highlighting, Ruler, and Status Bar) tells the program how each Editor Window should display data (see Working with File Interfaces for more information). When one of the interface options is modified, this affects all files that use that interface (except for the Endian and Word Wrap setting as discussed below). The View menu will display different options when a text-based File Interface is active as opposed to a hex-based File Interface because some options are only applicable to one type of interface. By default, 010 Editor comes installed with a number of File Interfaces as displayed on the '*View > Edit As*' menu; however, different interfaces can be created for different files based on the file name or extension. Note that all File Interfaces are saved automatically upon exit. The View Menu is also used to control which panels are displayed and controls options for the Tool Bars. This section lists all available menu options in the View menu:

- **Edit As > (File Interface List) -** Shows a list of possible File Interfaces (see Working with File Interfaces) that can be used to edit files. A check mark is placed beside the interface that is applied to the current file. To create more interfaces, see the '*Create New File Interface*' or '*Edit File Interfaces*' menu option.
- **Edit As > Toggle Hex Interface -** If editing a file using a text-based File Interface clicking this option chooses a hex-based File Interface to edit the file. Similarly, if the current interface is a hex-based File Interface the File Interface is switched to a text-based interface.
- **Edit As > Create New File Interface -** Creates a new interface for all files with the same extension as the current file. For example, if the current file has a '.bmp' extension then all files that are opened with a '.bmp' extension will use the created interface and the interface will be named 'bmp' on the Edit As menu.
- **Edit As > Edit File Interfaces -** Displays the File Interface Options dialog that is used to modify, create, or delete File Interfaces.
- **Font -** Sets the font for the current interface. This menu will display either *Use Default Text Editor Font* or *Use Default Hex Editor Font* depending upon if this Interface is for text or hex files. If this toggle is enabled, the default font is used from the Font Options dialog, otherwise this interface uses its own custom font. Clicking *Change Font* displays the standard Font dialog that allows setting of the font type, size, and style. Use the *Enlarge Font* (*Ctrl++*) or *Shrink Font* (*Ctrl+-*) menu options to increase or decrease the size setting for the font. If this Interface is using one of the default fonts, the *Change Font*, *Enlarge Font* and *Shrink Font* menu options affect the default font, otherwise these options affect the custom font for this Interface.
- **Character Set -** This option controls which character set is used when displaying characters in the Text Editor or the character area of the Hex Editor. The list of character sets can be viewed and modified using the Character Set Options dialog. The Character Set menu lists the most commonly used character sets at the top of the menu (this can be controlled using the *Show at Top Level* toggle in the Character Set Options dialog), followed by a list of recently used character sets which are not on the common list. Next the menu lists the rest of the available character sets sorted into *Standard*, *International* and *Custom* categories. At the bottom of the Character Set menu is the *Use Default* toggle. When this toggle is turned on the character set is used from the current File Interface and changing to a new character set modifies all files that use that File Interface; however, when the *Use Default* toggle is off this file uses a per-file character set and changing the character set only affects the selected file. The *Use Default* toggle may be turned off automatically when opening a file if a different character set is detected than is set in the File Interface. See the Character Set Options dialog for more information about character sets. Note that the Endian setting can affect certain character sets such as the Unicode character set.
- **Linefeeds *(text only)* -** Controls how a text file is divided into lines. Currently this menu is used to control how *Word Wrap* is applied to files. When *Word Wrap* is enabled, any lines that go beyond the edge of the Text Editor Window are wrapped onto a new line. Click the *Word Wrap* option to turn wrapping on or off and the text **/wrap** will appear beside the File Interface name in the File Bar when enabled (see Word Wrap for more information). Note that whether Word Wrap is turned on is stored separate to the File Interface and the current Word Wrap state is remembered for files if the *Remember Last Used Interface* toggle is set in the Editor Options dialog. If a file has not been opened before (or the *Remember Last Used Interface* toggle is turned off), the wrap state can be set with the *Initial Wrap State* menu option to *On*, *Off* or *Auto-detect*. For *Auto-detect*, 010 Editor will automatically turn on Word Wrap if it finds the file contains long text lines. Wrapping is usually done at the edge of the Text

Copyright © 2003-2019 SweetScape Software

Editor Window but can be set to occur at a specific column with the *Wrap Width* menu option. Wrapping can be set to *Column 80* or any other column number by choosing *Custom Column*. When wrapping occurs at a set column, a line will be drawn in the Text Editor Window to indicate the wrap width. When applying Word Wrap, by default whole words are always kept together but wrapping can be done on any letter by choosing *Wrap on Letters* on the *Wrap Method* menu. Select *Wrap on Words* to return to keeping whole words together. The *Column Mode* menu item switches the editor to or from Column Mode and see the separate Column Mode help topic for more information.

- **Line Width *(hex only)* -** Specifies how many bytes are displayed on each line of the Hex Editor. For example, if *Fixed 16 Bytes* is chosen the file is displayed with 16 bytes per line. When the linefeeds are set to *Auto Width* the number of bytes to display is automatically chosen to fill the width of the Hex Editor Window. The width can be set to 4, 8, 12, 16, 20, or *Custom Width*. When setting to a *Custom Width* a dialog will be displayed for entering any width (values between 1 and 1024 are allowed). Note that the number of bytes per line must be divisible by the number of bytes in the *Group By* option.
- **Tabs/Whitespace *(text only)* -** When editing a text file the columns of the file are divided into a number of tab stops. Enable the *Show Whitespace* menu option to display symbols indicating where all spaces, tabs and linefeeds exist in the file (the color of the symbols can be controlled using the Theme/Color Options dialog and the symbols displayed can be controlled using the Text Editor Options dialog). Choose the number of characters between each tab stop using the *Tab Size:* menu options (click the *Tab Size: Custom* menu option to set a specific tab size using an input dialog). When a tab character is encountered in a file the next character is drawn at the next tab stop. The number of characters to insert when the Tab key is pressed can be controlled through the *Indent Size:* menu options. Note that the *Indent Size* may be different than the *Tab Size*. If the *Insert Spaces* toggle is enabled (the default) spaces are inserted into a file to simulate the tab positions. To insert actual Tab characters disable the *Insert Spaces* option and make sure the *Tab Size* and *Indent Size* are set to the same amount. An easy way to access this menu is by clicking the *Tab:* section of the status bar.
- **Addresses -** Controls the addresses for the current File Interface as displayed in the left-hand column of the Editor Window. If *Show Addresses* is off then no labels are displayed but the addresses will be shown in a hint popup when the mouse is placed over the address column for a second. The lower section of the Addresses menu controls the format of the addresses and can display either the *Byte Number*, *Line Number*, *Sector Number* or *Short Number* (note that a Short is a group of two bytes in a hex file). Some addresses can be displayed in either *Hex*, *Decimal*, or *Octal* format as indicated in brackets to the right of the address type. Note that when editing a hex file the line number displayed depends upon the number of bytes per row in the editor (set the Line Width option above). When *Sector Number* is chosen the address is displayed in the format '< *sector number* >|< *sector offset* >' (see Editing Drives for more information on sectors). If *None* is chosen no addresses are displayed for the file.
- **Group By *(hex only)* -** When using a hex-based File Interface this option sets how many bytes are grouped together in the display. Bytes that are grouped together are displayed without spaces. The default value is *Byte*, meaning that each byte is displayed with spaces around it. A custom *Group By* value can be set by clicking the *Custom* option and entering a value. Note that the number of bytes per line (see Line Width above) must be divisible by the number of bytes per group. The Hex Editor Window has a special mode that allows groups of bytes to be visually swapped without modifying the underlying data. This mode can be enabled by clicking the *Swap Little-Endian Bytes by Group* toggle and choosing a Group By other than *Byte*. See Swapping Bytes for more information.
- **Division Lines *(hex only)* -** When using a hex-based File Interface the Division lines allow drawing lines on the Hex Editor Window to visualize how data is grouped into sections. Currently there are two types of lines that can be displayed: *Division Lines* and *Sector Lines*. By default Division lines are displayed every 4 bytes of the file and are drawn in a light gray color (use the Color Options dialog to modify the color). The top portion of the '*View > Division Lines*' menu is used to control Division Lines. Choose 1, 2, 4, 8, or *Custom* to set the division spacing, or select *None* to hide the lines. The '*Set Starting Division Offset*' allows starting the division lines on an address other than the beginning of the file. This feature is useful if your file contains a header at the beginning of the file and then a number of fixed size records (the starting offset can be used to skip over the header).

  Sector Lines are meant to visualize the sectors of a hard drive (usually 512, 1024, or 2048 bytes in size), but can be modified by the user to visualize other types of data when not editing a hard drive. By default, Sector Lines are displayed as dark gray lines but this can be modified in the Theme/Color Options dialog. Use the bottom portion of the '*View > Division Lines*' menu to control the Sector Lines. Show or hide the Sector Lines by clicking the '*Show Sector Lines*' menu option. When editing a hard drive, the sector size is determined from the physical device (see Editing Drives); however, when editing a regular file, the sector size can be user defined by clicking the '*Set Sector Size*' menu option (this option is useful if you are editing a file which is an image of a drive). When a regular file is being editing a starting address for the sector lines can be specified using the '*Set Starting Sector Offset*', similar to the Division Lines.

- **Left Area *(hex only)* -** If the current File Interface is for hex-based files, this option controls which numeric format is used to display the bytes in the left side of the Hex Editor Window. The options are

Hex, *Char*, *Octal*, *Binary*, and *Decimal*. See Introduction to Number Systems for more information.

- **Right Area *(hex only)* -** When using a hex-based interface, this option controls which numeric format is used to display the bytes in the right side of the Hex Editor Window. The options are *Hex*, *Char*, *Octal*, *Binary*, and *Decimal*. As well, the *Hide* option can be selected to only display the left area.
- **Highlighting -** Controls which bytes are highlighted in the current interface using a color scheme. When a color scheme is enabled by clicking on an item in the *Highlighting* menu, the background color of all bytes that match that scheme are modified. A check mark will appear beside an active highlight, and the highlight can be turned off by clicking on the highlight name again. Note that multiple highlights can be turned on at the same time (highlights at the top of the list take precedence over highlights at the bottom). The default Highlight options are '*Linefeed Characters*' (0x0d and 0x0a), '*Alphanumeric Characters*' (all letters and numbers), '*Control Characters*' (any of the bytes from 0 to 31), and '*Non-ASCII Characters*' (any of the bytes from 128 to 255). Custom highlights can be generated by clicking the '*Edit Highlights...*' menu option (see Highlight Options for more information) and highlights can also be created for highlighting Shorts (a Short is a group of two hex bytes). Note that Syntax Highlighting is now applied using the File Bar.
- **Ruler -** The *Show Ruler* option controls whether the Ruler is displayed above each file. The ruler is a band with tick marks, indicating byte offsets from the first byte in a line. If *Show Labels* is turned off then no text labels are displayed on the Ruler but the text labels will be shown in a hint popup when the mouse is placed over the ruler for a second (both the mouse location and the current cursor position are shown). If *Show Arrows* is on then small arrows indicating the current mouse or cursor position are displayed in the ruler. The units used for the ruler labels can be set to hex or decimal using *Hex Units* or *Decimal Units*.
- **Status Bar > File Position -** Controls the format of the current cursor position as displayed in the Status Bar as either *Byte Number*, *Line Number*, *Sector Number*, or *Short Number* (a Short is a group of two bytes within a hex file). See Status Bar for more information.
- **Status Bar > File Size -** Sets the format of the file size displayed in the Status Bar as either *Byte Count*, *Line Count*, *Sector Count*, or *Short Count*.
- **Status Bar > Selection Size -** When a selection is made, the number of selected bytes is displayed in the status bar with the label *Sel:*. This menu option controls the format used to display the selection size.
- **Endian -** Controls which byte-ordering is used for the current file (see Introduction to Byte Ordering). The ordering can either be *Little Endian* (Intel machines), or *Big Endian* (Motorola Machines). When the current file is in little endian mode, the Status Bar will contain the letters *LIT* and when the file is in big endian mode, *BIG* will be displayed (see Status Bar). Use the *Toggle Endian* menu option to switch between the two endians. The Endian setting is different than the other options in the File Interface since this option contains only the default Endian setting when a file is opened. Changing the Endian does not modify other files that are using the same File Interface (for example, you can have a big endian file and a little endian file open at the same time both using the Unicode File Interface).
- **Workspace Windows -** Allows control over which tabs in the Workspace panel are visible. Click *Show/Hide All Workspace Windows* to toggle visibility of the Workspace tab and set all other tabs to the same visibility.
- **Inspector Windows -** Hides or shows the tabs of the Inspector panel in the main window. To show or hide all of the tabs at the same time click the *Show/Hide All Inspector Windows* menu option.
- **Output Windows -** Used to show or hide the different Output tabs including the Find Results, Find in Files, Compare, Histogram, Checksum, and Process tabs. Click *Show/Hide All Output Windows* to toggle the visibility of all Output tabs at once (the Esc key can also be pressed to hide the Output tabs).
- **Template Results -** Toggles whether the Template Results panel is displayed at the bottom of the current Editor Window. See Working with Template Results for more information.
- **Floating Tab Group -** Toggles the display of the Floating Tab Group as discussed in Using File Tabs.
- **Tool Bars -** Allows control of which Tool Bars are displayed in the program.
- **Other Windows -** Allows display of other miscellaneous windows for 010 Editor: the Startup Page and the initial *Welcome* dialog.

Related Topics:
Character Set Options
Column Mode
Editing Drives
Editor Options
File Interface Options
Font Options
Highlight Options
Introduction to Byte Ordering
Introduction to Number Systems
Status Bar
Using Syntax Highlighting
Text Editor Options
Theme/Color Options

Using the Inspector
Using the Workspace
Working with File Interfaces
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Format Menu

The section lists all available menu options in the Format menu:

- **Uppercase -** Converts all letters in the current selection to their uppercase form (e.g. 'a' -> 'A'). All other characters will not be modified.
- **Lowercase -** Modifies each letter in the current selection to their lowercase form (e.g. 'A' -> 'a'). All other characters will not be modified.
- **Capitalize -** The command will capitalize the first letter of each word in the current selection and make all other letters lowercase (e.g. 'the title' -> 'The Title'). All non-letter characters will not be changed.
- **Tabify -** When working with text files, this command converts any set of two or more spaces into tab characters if the conversion can be done to preserve the spacing of each line. If no selection is made this operation converts the whole file, and if a selection is made the operation converts all lines that contain all or part of the selection. To view the current tabs and spaces in a file see the Show Whitespace menu option.
- **Untabify -** The Untabify command converts all tab characters into their equivalent in spaces. If no selection is made the whole file is converted, and if a selection is made only those lines that intersect the selection are modified. Note this command is only available when working with text files. All tabs and spaces in the file can be shown using the Show Whitespace menu option and the number of spaces per tab can be controlled using the View > Tabs/Whitespace menu.
- **Comment Selection -** Adds characters to the file so that the currently selected bytes are considered as comments in a file. If no selection is made the line the cursor is on will be commented. The Comment Selection command supports two different types of comments: line comments (for example '//' in C++) and multi-line comments (for example '/*' and '*/' in C++ or '<!--' and '-->' in HTML). The commenting used is derived from the current syntax highlighting. If no commenting characters can be determined from the current syntax highlighting, C/C++ comments are used. Line comments will be inserted on each line if possible, but if not possible then multi-line comments will be inserted. Comments can be deleted using the *Uncomment Selection* command.
- **Uncomment Selection -** This command attempts to remove commenting from the current selection and if no selection is made comments are removed from the current line. Both line comments and multi-line comments are supported as described in the *Comment Selection* command above. If the bytes to be uncommented contain line comments, those comments are removed first. If no line comments exist then multi-line comments are removed instead.
- **Increase Line Indent -** Adds tab or space characters to the start of each line that lies all or partially within the selected bytes. If no selection is made, tabs or spaces are added to the current line. Whether tabs or spaces are inserted is controlled using the '*View > Tabs/Whitespace > Insert Spaces*' menu option. The number of spaces or tabs inserted is controlled using '*View > Tabs/Whitespace > Indent Size*' (see the View Menu for more information). This command has the same effect as pressing the Tab key in the editor while bytes are selected (see Using the Text Editor).
- **Decrease Line Indent -** Removes tabs or spaces from the beginning of each line that intersects the current selection. If no selection is made, tabs or spaces are removed from the current line. The number of tabs or spaces deleted is controlled using '*View > Tabs/Whitespace > Indent Size*' (see the View Menu. This command is equivalent to pressing the Shift+Tab key in the Text Editor when a selection is made (see Using the Text Editor).
- **Delete Line -** If no bytes are selected then the line the cursor is on is deleted. If bytes are selected then all lines that contain the selection are deleted.
- **Delete Blank Lines -** When bytes are selected, all lines in the selection that are empty or just contain whitespace are deleted from the file. If no bytes are selected then all lines in the file that are empty or just contain whitespace are deleted.
- **Delete Left Word -** Deletes the first word to the left of the cursor. If the cursor is part way through a word, the word is deleted starting from the cursor position to the left until the start of the word is found. If a selection is active just the selected bytes are deleted.
- **Delete Right Word -** Deletes the first word to the right of the cursor. When the cursor is part way through a word, the word is deleted starting from the cursor position to the right until the end of the word is found. If a selection is active just the selected bytes are deleted.
- **Trim Trailing Whitespace -** This command deletes all spaces and tabs that occur after the last visible character on each line. If no selection is made the whitespace will be trimmed from every line in the file,

but if a selection is made each line that lies completely or partially within the selection will be trimmed.

Related Topics:
Using Syntax Highlighting
Using the Text Editor
View Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Scripts Menu

Scripts are short programs similar to C that can be used to edit text or binary files (see Introduction to Templates and Scripts). This section lists all available menu options in the Scripts menu:

- **New Script -** Creates a new script file and opens it for editing in the interface.
- **Open Script -** Displays a file dialog box for loading an 010 Editor Script file (usually with file extension ".1sc").
- **Edit Script -** If a script has been associated with the current file, the script is opened in the application for editing. If no script is associated with the current file, the last accessed script will be opened and if no script has been opened you will be asked to create a new script or open an existing script. Scripts can be associated with the current file by using the *Run Script* or the *Run on File* section of the File Bar above each editor. See Running Templates and Scripts for more information.
- **Run Script -** If the currently selected file is *not* a script, clicking the '*Run Script*' menu option will run any script that is associated with the current file. If the current file is a script then clicking this option will execute the script on the selected file. See Running Templates and Scripts for more information. If a script modifies a file, the '*Edit > Undo*' menu option will undo any changes. Note that other editing operations can be performed while a long Script is running.
- **Continue Script or Template -** This menu option only appears when a Script or Template is paused at a line using the debugger. Clicking continue will resume execution of the Script or Template from the current line. Note that only one Script or Template can be run at a time and the current Script or Template must be stopped to run another.
- **Stop Script -** This menu option only appears when a Script is executing or is paused at a line using the debugger. Click *Stop Script* to stop the Script and reset the debugger without deleting any variables that were created.
- **(Installed Script List) -** Displays a list of installed scripts sorted by category such as '*Randomize*', '*IsASCII*', '*MultiplePaste*', '*SplitFile*', and '*JoinFile*'. Clicking on an item in this menu usually executes the script but can also be configured to load the script into the interface for editing. See Script Options for more information on adding scripts, changing script options, or using the default custom scripts. Scripts can also be installed using the Repository Dialog.
- **View Installed Scripts -** Opens the Script Options dialog that can be used to add or customize scripts on the Installed Script List.
- **Script Repository -** Displays the Repository Dialog which can be used to install Scripts other people have uploaded. Please consider uploading to the Repository any useful scripts you have developed. The Repository can also be accessed online at '*http://www.sweetscape.com/010editor/repository/*'.

Related Topics:
Introduction to Templates and Scripts
Running Templates and Scripts
Script Options
Using the Debugger
Using the Repository Dialog

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Templates Menu

The Templates menu allows loading and running Binary Templates (see Introduction to Templates and Scripts). This section lists all available menu options in the Templates menu:

- **New Template -** Creates a new Binary Template and opens it for editing in the interface.
- **Open Template -** Opens a file dialog box to load a Binary Template (usually with file extension ".bt").
- **Edit Template -** If the current file has a template associated with it, that template is displayed for editing in the interface. If there is no template associated with the current file, the last accessed template is opened and if no template has been opened you will be asked to create a new template or open an existing template. Templates can be associated with the current file by using the *Run Template* or the *Run on File* section of the File Bar above each editor. See Running Templates and Scripts for more information.
- **Run Template -** If the currently selected file is *not* a template, clicking the '*Run Template*' menu option will run any template that is associated with the current file. If the current file is a template, clicking the option will execute the template on the selected file. See Running Templates and Scripts for more information. The results are displayed either in the *Template Results* panel below the Editor Window or in the *Variables* tab of the Inspector (see Working with Template Results for more information). Note that other editing operations can be done while a Template is running.
- **Continue Script or Template -** This menu option only appears when a Script or Template is paused at a line using the debugger. Clicking continue will resume execution of the Script or Template from the current line. Note that only one Script or Template can be run at a time and the current Script or Template must be stopped to run another.
- **Stop Template -** This menu option only appears when a Template is currently running or is paused at a line using the debugger. Clicking *Stop Template* stops the Template from executing and resets the Template but does not delete any variables that were created.
- **(Installed Template List) -** Displays the list of installed templates, such as '*BMP*', '*ZIP*', and '*WAV*', sorted by category. Clicking on an item on the list usually executes the template but can also be configured to open the template for editing in the interface. Templates can be added to the list by clicking the '*View Installed Templates*' menu option or by using the Repository Dialog. Note that templates can be configured to load automatically based on the opened file type (see Template Options for more information).
- **View Installed Templates -** Displays the Template Options dialog that can be used to add templates to the Installed Template List for easy access.
- **Template Repository -** Displays the Repository Dialog which can be used to install Templates other people have uploaded to the Repository. Please consider uploading any templates you have created that may be useful to other people. The Repository can also be accessed online at '*http://www.sweetscape.com/010editor/repository/*'.

Related Topics:
Introduction to Templates and Scripts
Running Templates and Scripts
Template Options
Using the Inspector
Using the Debugger
Using the Repository Dialog
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Debug Menu

The Debug menu controls functions of the Debugger which is used to find and fix issues with 010 Editor Templates and Scripts. This section lists the menu options for the Debug menu:

- **Debugging Enabled -** Indicates whether debugging is turned on for the application. If debugging is on a checkmark will appear beside this menu option and if off an 'X' will appear. Toggle whether debugging is turned on by clicking the menu option. When debugging is off, execution will not pause at any breakpoints and many of the options in the Debug menu will be disabled. Also, when turned off the application will not ask to start the debugger if any errors occur in a Script or Template.
- **Start Debugging -** Starts execution of the currently selected 010 Editor Script or Template. This option is only available when a Script or Template is selected and not when a data file is selected. Execution will pause at any breakpoints that have been set in the file. Note this option has the same effect as using '*Scripts > Run Script*' or '*Templates > Run Template*' when debugging is turned on.
- **Continue -** When the debugger has paused at a line in a Script or Template, clicking this option continues execution from the current line. Note this option is only visible when execution is paused and is identical to using '*Scripts > Continue Script or Template*' or '*Templates > Continue Script or Template*'.
- **Pause -** Pause execution of a currently running 010 Editor Script or Template and starts the debugger, placing the cursor at the next line of the program to be run. The current line is marked with a yellow arrow in the Text Editor.
- **Stop Script/Template -** If a Script or Template is currently being run or is paused, this menu option stops execution of the Script or Template. Scripts or Templates which are taking too long to execute can be cancelled with this option.
- **Step Over -** Executes any statements on the current debugger line and advances to the next line in the Script or Template. If the current line contains a function or struct, the whole function or struct will be run without stopping the debugger.
- **Step Into -** This menu option has two uses. If the current Script or Template has not been started then clicking this option will start the Script or Template and pause execution at the first line of the program that can be run. If program execution is paused then this option will execute any statements on the current debugger line and advance to the next line in the Script or Template. If the current line contains a function or struct, the debugger will pause at the first line of the function or struct. Note the debugger cannot currently step into functions located inside a DLL.
- **Step Out -** If program execution is paused at a line which is inside a function or struct, this menu option executes the rest of the lines inside the function or struct and stops at the next line which is outside the function or struct. If the program is paused at a line which is not inside a function or struct then the rest of the Script or Template is executed.
- **Toggle Breakpoint -** Marks that the debugger should stop at the current line of the selected Script or Template by creating a breakpoint. If the current line already has a breakpoint then the breakpoint is deleted. Breakpoints are indicated by a red arrow along the left-hand column of a Script or Template and breakpoints can also be toggled by clicking the left-hand column of a Script or Template with the mouse.
- **View Breakpoints -** Displays the *Breakpoints* tab which contains a list of all the breakpoints in the current Script or Template. The *Breakpoints* tab is located in a tab group with the *Inspector* tabs and may occasionally be hidden underneath the Floating Tab Group. Breakpoints may be viewed, added or deleted using the Breakpoints tab.
- **Delete All Breakpoints -** Removes all breakpoints in all files. Note that breakpoints are stored to disk and reloaded when 010 Editor is restarted.
- **Quick Watch -** Displays the Quick Watch dialog which can be used to evaluate expressions or view the value of variables in the current Script or Template. The Quick Watch dialog can only be opened when execution is paused at a line in a Script or Template.
- **View Watches -** Shows the *Watch* tab which can be used to evaluate a set of expressions every time program execution pauses at a line in the Script or Template. The *Watch* tab exists in the *Inspector* tab group which sometimes may be hidden by the Floating Tab Group. See Watches for more information.
- **View Call Stack -** Displays the *Call Stack* tab which exists in a tab group with the other *Inspector* tabs. When a Script or Template is paused at a line which is inside a function or struct, the Call Stack shows which functions or structs were called to reach the current line. See Using the Call Stack for more

Copyright © 2003-2019 SweetScape Software

information.

Related Topics:
Using the Debugger
Running Templates and Scripts
Scripts Menu
Templates Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Tools Menu

The Tools menu contains a number of powerful tools for editing, plus the Options and Register dialogs. This section lists all available menu options in the Tools menu:

- **Calculator -** Opens the 010 Editor Calculator that uses a syntax similar to C (see the Calculator for more information).
- **Compare Files -** Opens the Compare tool that is used to compare two files (see Comparing Files for more information).
- **Hex Operations -** Performs a mathematic operation, such as *Add* or *Multiply* on a set of bytes in the file (see Hex Operations for more information).
- **Convert -** Used to convert the current file into other formats (see Converting Files for more information).
- **Histogram -** Performs a histogram (byte count) operation on the current file (see Histograms).
- **Check Sum -** Opens the Check Sum dialog to perform a set of check sum or hash algorithms on the file (see Check Sum/Hash Algorithms for more information).
- **Base Converter -** Displays a tool used to convert between Hex, Decimal, Octal, and Binary numeric formats (see Base Converter for more information).
- **(Custom Tool List) -** Displays a list of custom external programs that can be run. Clicking *Windows Notepad* loads the currently open file in that program. Other tools can be added to the list and see Program Options for the list of standard tools.
- **Register -** Displays the Register Dialog that is used to enter your Name and Password after purchasing 010 Editor (see How to Buy 010 Editor for more information).
- **Options -** Displays the options for 010 Editor (see General Options more information).

Related Topics:
Base Converter
Calculator
Check Sum/Hash Algorithms
Comparing Files
Program Options
General Options
Histograms
How to Buy 010 Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Window Menu

The Window menu is used to manage all open files. This section lists all available menu options in the Window menu:

- **Duplicate Window -** Opens another Editor Window that can be used to view the current file. The title of the window will change to include a ':' followed by a number to indicate that there are multiple windows open for the same file. Note that if a change is made in one file, the change will also appear in the additional view. See Opening Files for more information.
- **Split Window -** Divides the current Editor Window into two areas, a top and a bottom area. These two areas can be used to edit two different portions of a file at the same time. The window can also be split by grabbing the small button above the vertical scroll bar in the Editor Window (see Using the Text Editor or Using the Hex Editor for more information). If the Editor Window has already been split into two areas, clicking this menu option will remove the split and display only one area.
- **Synchronize Scrolling -** This menu option is a toggle that can be enabled or disabled by clicking on it. When the toggle is enabled and one Editor Window is scrolled up or down, then all other Editor Windows that are currently visible will scroll by the same amount. This option is useful to visually compare large sections of two different files. When using the Compare Files tool, an option exists in the Compare dialog to automatically enable *Synchronize Scrolling* after a comparison is run.
- **Synchronize Template Results Scrolling -** Similar to *Synchronize Scrolling* except when this option is enabled and one Template Results panel is scrolled up or down, then all other Template Results panels that are visible will scroll by the same amount. One way to view multiple Template Results panels at the same time is to use multiple Tab Groups and see Using File Tabs for more information. Use this option when comparing the Template Results between two files and an option exists in the Compare dialog to enable this option after a comparison is done.
- **Next Window -** Selects the next Editor Window in the interface as the active Window.
- **Previous Window -** Selects the previous Editor Window in the interface as the active Window.
- **Move to New Horizontal Tab Group -** If multiple Editor Windows are open, they may be separated into multiple horizontal Tab Groups. Selecting this option creates a new horizontal Tab Group and places the current Editor Window into that group. See Using File Tabs for more information on Tab Groups. If any Tab Groups are laid out vertically, they will be switched to horizontal.
- **Move to New Vertical Tab Group -** Multiple Editor Windows may be separated into different vertical Tab Groups. This option creates a new vertical Tab Group containing the current Editor Window. See Using File Tabs for more information on Tab Groups. Clicking this option will change any Tab Groups laid out horizontally to be laid out vertically.
- **Move to Floating Tab Group -** An additional Tab Group in a moveable window is available by clicking the '*View > Floating Tab Group*' menu option. The current Editor Window can be moved to this Tab Group by clicking on this option (see Using File Tabs).
- **Move to Next Tab Group -** If multiple Tab Groups have been created by clicking the '*Move to New Horizontal Tab Group*' or '*Move to New Vertical Tab Group*' menu options or by dragging the File Tab to a new position, this option can be used to move the current Editor Window to the next Tab Group.
- **Move to Previous Tab Group -** This command is similar to the '*Move to Next Tab Group*' option except the current Editor Window is moved to the previous Tab Group.
- **Merge All Tab Groups -** Clicking the '*Merge All Tab Groups*' option takes all open Editor Windows (including any Editor Windows in the Floating Tab Group) and places them into a single Tab Group in the main interface.
- **Window List -** Displays a dialog containing a list of all open windows. Double-click on a window, or select a window from the list and click the *Activate* button to focus that window. Click the *Cancel* button to close the dialog.

At the bottom of the Windows menu a list of all open windows is displayed with a check mark beside the active window. Select a file from the list to activate that file.

Related Topics:
Comparing Files
Opening Files
Using File Tabs

Using the Hex Editor
Using the Text Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Help Menu

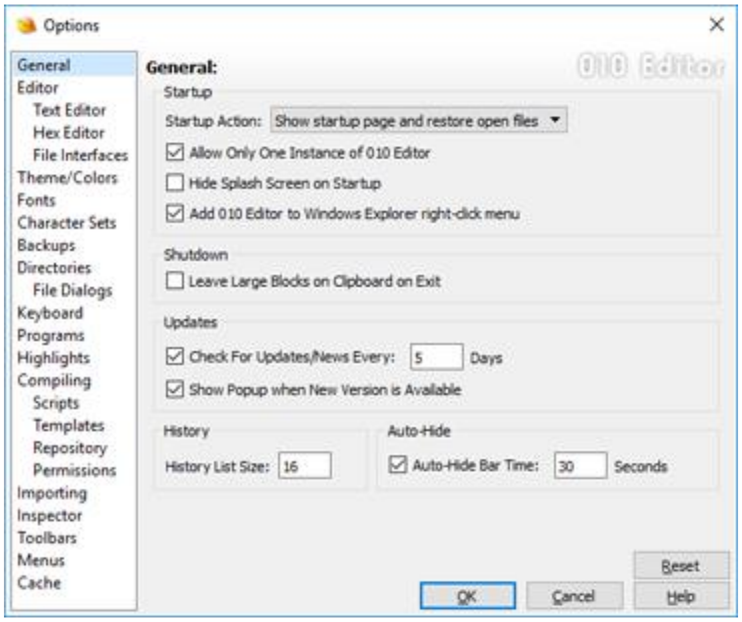
The Help menu gives access to the documentation, website, and support for 010 Editor. This section lists all available menu options in the Help menu:

- **Help Topics -** Displays this help file.
- **SweetScape Homepage -** Loads an Internet browser (if available) and visits '*http://www.sweetscape.com/*'.
- **010 Editor Homepage -** Loads an Internet browser (if available) and visits '*http://www.sweetscape.com/010editor/*'.
- **Buy Now -** Visits the website '*http://www.sweetscape.com/store/*' which allows you to purchase a copy of 010 Editor. See How to Buy 010 Editor for more information.
- **Tutorials > Introduction to Binary Templates -** Displays a short tutorial on how Binary Templates work, including how to run templates and how to use the Template Results panel.
- **Tutorials > Writing Binary Templates -** Shows a tutorial on how to create your own Binary Templates.
- **Tutorials > Using the Repository -** Displays a tutorial on the basics of the 010 Editor Repository.
- **Check for Updates -** Checks if a new version of 010 Editor is available if the computer is internet connected. Note that a popup window will automatically be displayed if 010 Editor detects a new version is available and this can be turned off using the General Options dialog.
- **View Release Notes -** Displays the list of changes for the current and previous versions of 010 Editor (see Release Notes).
- **View Shortcut List -** Displays a list of shortcut keys in the application sorted by shortcut name. See Keyboard Options for more information.
- **Support on the Web -** Obtain support by visiting the website '*http://www.sweetscape.com/support/*'.
- **Support by E-mail -** Opens a web form which can be used to send an e-mail message to '*support@sweetscape.com*'.
- **About -** Displays the About dialog containing version information for 010 Editor.

Related Topics:
General Options
How to Buy 010 Editor
Introduction to the Repository
Keyboard Options
How to Get Support
Release Notes

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

General Options



Options

General:

Startup

Startup Action: Show startup page and restore open files

☑ Allow Only One Instance of 010 Editor

☐ Hide Splash Screen on Startup

☑ Add 010 Editor to Windows Explorer right-click menu

Shutdown

☐ Leave Large Blocks on Clipboard on Exit

Updates

☑ Check For Updates/News Every: 5 Days

☑ Show Popup when New Version is Available

History

History List Size: 16

Auto-Hide

☑ Auto-Hide Bar Time: 30 Seconds

The General Options dialog contains options for the whole application. Access this dialog by clicking the '*Tools > Options...*' menu option and selecting *General* from the list.

Startup

- **Startup Action -** This drop-down list indicates what 010 Editor should display when first started. The default is *Show startup page and restore open files* meaning that all open files are reloaded from when 010 Editor was last run but the Startup Page is focused. If *Show startup page* is selected just the Startup Page is shown and no files are reloaded. If *Restore all open files* is selected, all files will be reloaded and the last file being edited will be focused. Alternately, 010 Editor can create a new file or display no tabs at all by selecting the *Create new file* or *Display empty interface* options respectively.
- **Allow Only One Instance of 010 Editor -** If this flag is set, only one copy of 010 Editor is allowed to run at a time. If new files are opened from the Windows Explorer or elsewhere, they are loaded into the already-running program.
- **Hide Splash Screen on Startup -** If enabled, this toggle will prevent the splash screen from displaying while 010 Editor is starting up. This option is only available if you have purchased a license of 010 Editor and entered your license information in the Register dialog.
- **Add 010 Editor to Windows Explorer right-click menu -** If this toggle is enabled, an entry that says *010 Editor* will be added to the Windows Explorer popup menu when you right-click on a file. Clicking the *010 Editor* option will cause the file to be opened in 010 Editor.

Clipboard

- **Leave Large Blocks on Clipboard on Exit -** When this toggle is set and the clipboard contains a large block of data on exit (over 4 megabytes), 010 Editor will copy this block onto the system clipboard. When not set, large blocks are discarded and will be unavailable to other programs after 010 Editor exits. Note that small blocks (under 4 megabytes) are automatically copied. 010 Editor supports copying large blocks (gigabytes in some cases) that the Windows clipboard cannot handle (the limit is about 16 megabytes on some systems). Therefore some blocks may not copy onto the clipboard properly if too much data is copied.

Updates

- **Check For Updates/News Every -** If this toggle is enabled 010 Editor will download any news, including notification of new releases, from our website and display it on the Startup Page. Enter the number of days between downloads in the *Days* field and note that new updates are not installed automatically. See the Startup Page for more information.
- **Show Popup when New Version is Available -** When this toggle is enabled and 010 Editor discovers a new version is available while checking for updates, a popup window will be displayed. If the *Check for Updates* toggle is turned off a popup will not be displayed. Updates can also be checked manually by clicking the '*Help > Check for Updates*' menu option on the main menu.

History

- **History List Size -** Specifies the number of items to appear in the recent file history list. This affects the number of items on the '*File > Open Recent*' list and the number of *Recent Files* in the Workspace (see Using the Workspace for more information).

Auto-Hide

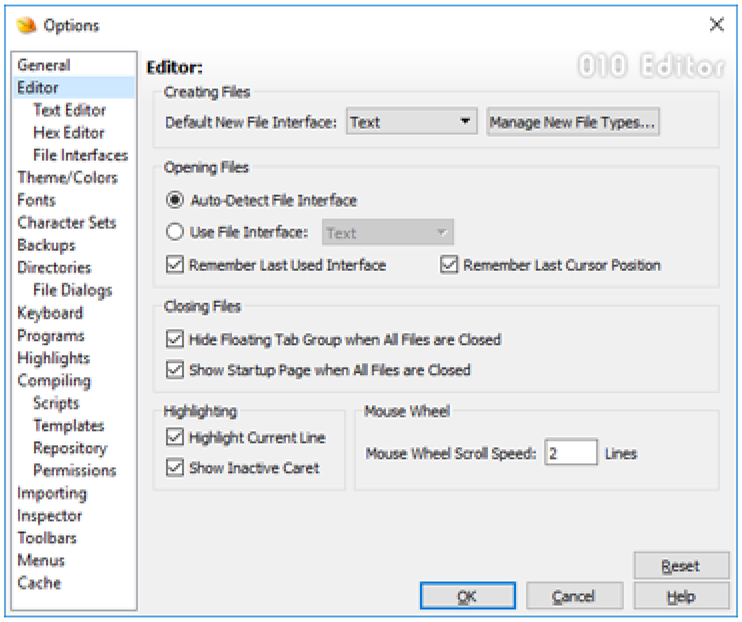
- **Auto-Hide Bar Time -** Some tools in 010 Editor are displayed as a bar at the bottom of the editor including the Find Bar, Replace Bar, Goto Bar, Select Bar, etc. When any of these bars are not used for a period of time, they will be automatically hidden. Enter the number of seconds of inactivity in the *Auto-Hide Bar Time* field before a bar is hidden. If the toggle to the left of *Auto-Hide Bar Time* is disabled, the auto-hide time will be ignored and bars will never be automatically hidden.

Click the *Reset* button to return all of the General Options to their default values.

Related Topics:
Introduction to the Data Engine
Using the Workspace

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Editor Options



Editor:

Creating Files
Default New File Interface: Text

Opening Files
[● Auto-Detect File Interface]
○ Use File Interface: Text
☑ Remember Last Used Interface
☑ Remember Last Cursor Position

Closing Files
☑ Hide Floating Tab Group when All Files are Closed
☑ Show Startup Page when All Files are Closed

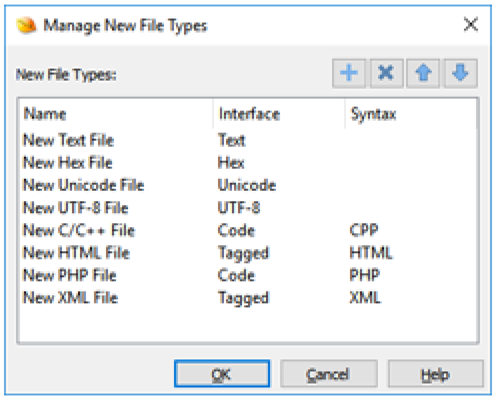
Highlighting
☑ Highlight Current Line
☑ Show Inactive Caret

Mouse Wheel
Mouse Wheel Scroll Speed: 2 Lines

The Editor Options dialog is used to modify options that affect both the Hex Editor Window and the Text Editor Window. Open the Editor Options dialog by clicking '*Tools > Options...*' and selecting *Editor* from the list.

Creating Files

- **Default New File Interface -** When a file is created by clicking the New icon in the Tool Bar, the created file is assigned a File Interface (see Working with File Interfaces for more information). Select which File Interface the created file should have from the drop-down list.



Name	Interface	Syntax
New Text File	Text	
New Hex File	Hex	
New Unicode File	Unicode	
New UTF-8 File	UTF-8	
New C/C++ File	Code	CPP
New HTML File	Tagged	HTML
New PHP File	Code	PHP
New XML File	Tagged	XML

Copyright © 2003-2019 SweetScape Software

- **Manage New File Types -** Clicking this button displays the *Manage New File Types* dialog as shown above. This dialog controls which entries appear in the '*File > New*' menu or the menu accessed by clicking the down arrow to the right of the New icon in the tool bar. The *Name* column controls which text appears on the menus and the *Interface* column controls which File Interface is assigned to the file after it is created. A Syntax Highlighter can also be assigned to the file when it is created using the *Syntax* column. Click the '+' icon to create a New File Type, the 'x' button to delete a New File Type, or the arrow keys to move a New File Type up or down in the list. Shortcut keys can be assigned to the different New File Types using the Keyboard Options dialog.

Opening Files

When a file is opened in 010 Editor, the file is assigned a File Interface based on the masks in the list of interfaces (see File Interface Options). If no mask matches the file name, 010 Editor can automatically try to detect the correct File Interface (select *Auto-Detect File Interface*) or 010 Editor can assign a specific interface (select *Use File Interface* and choose the interface from the drop-down list). When the *Remember Last Used Interface* toggle is set, 010 Editor will remember the last used File Interface, Endian, Character Set and Word Wrap setting for a file last time it was closed and will restore those settings when the file is opened again. If the *Remember Last Cursor Position* toggle is enabled then the last cursor position and scroll bar position will be restored to their previous values when 010 Editor is closed and reopened.

Closing Files

- **Hide Floating Tab Group when All Files are Closed -** When all tabs are closed in the Floating Tab Group, it will automatically be hidden if this toggle is enabled. If this toggle is turned off, an empty Floating Tab Group will be displayed when all tabs are closed. The Floating Tab Group can be displayed by clicking '*View > Floating Tab Group*' on the main menu.
- **Show Startup Page when All Files are Closed -** When this toggle is enabled and all files are closed in the editor, the Startup Page will automatically be shown. If this toggle is disabled then a blank interface will be shown when all files are closed and right-click on the blank interface and select '*Startup Page*' to display the Startup Page.

Mouse Wheel

- **Mouse Wheel Scroll Speed -** If your mouse is equipped with a scroll wheel, the Editor Window can be scrolled by rolling the wheel forward or backward. By default, for every click of the wheel the window scrolls two lines; however, to change the scroll speed enter a value in the *Mouse Wheel Scroll Speed* field. Higher values scroll faster and lower values scroll slower.

Highlighting

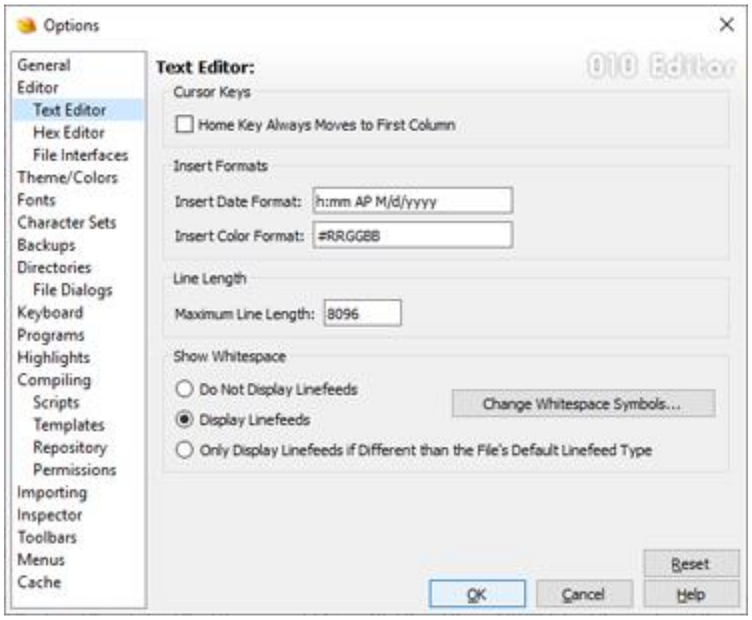
- **Highlight Current Line -** By default, the line the cursor is on in the Editor Window will be highlighted a yellow color. When this option is turned off, the highlight will no longer be displayed. Note that the color of the line can be controlled from the Color Options dialog.
- **Show Inactive Caret -** When an editor is not currently focused, a vertical gray line will appear where the cursor was. This line is called the *Inactive Caret* and can be disabled by turning the *Show Inactive Caret* toggle off.

The *Reset* button can be used to return all of the Editor Options to their default values.

Related Topics:
Color Options
File Interface Options
Introduction to Templates and Scripts
Using Syntax Highlighting
Using the Clipboard
Using the Hex Editor
Using the Startup Page
Working with File Interfaces
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Text Editor Options



Text Editor:

Cursor Keys

☐ Home Key Always Moves to First Column

Insert Formats

Insert Date Format: h:mm AP M/d/yyyy

Insert Color Format: #RRGGBB

Line Length

Maximum Line Length: 8096

Show Whitespace

○ Do Not Display Linefeeds

● Display Linefeeds

○ Only Display Linefeeds if Different than the File's Default Linefeed Type

Use the Text Editor Options dialog to control how the Text Editor Window operates (see Using the Text Editor for more information). Options for the Text Editor Window are also controlled by the Editor Options and the File Interface Options dialogs. To display the Text Editor Options window, click '*Tools > Options...*' and select *Text Editor* from the list.

Cursor Keys

- **Home Key Always Moves to First Column -** By default, when the Home key is pressed in the Text Editor Window the cursor moves to the first non-space character on the current line. If the cursor is already at the first non-space character, pressing the Home key moves the cursor to the first column. If the *Home Key Always Moves to First Column* toggle is enabled the cursor will always move to the beginning of the line when Home is pressed.

Insert Formats

- **Insert Date Format -** This option controls the format of the date when inserted with the '*Edit > Insert Date/Time*' menu option. The following character codes can be used when setting the date format:
  - h - hour without leading zero
  - hh - hour with leading zero
  - m - minute without leading zero
  - mm - minute with leading zero
  - s - second without leading zero

- ss - second with leading zero
- z - millisecond without leading zero
- zzz - millisecond with leading zero
- AP - either AM or PM
- ap - either am or pm
- d - day without leading zero
- dd - day with leading zero
- ddd - short day (e.g. 'Mon')
- dddd - long day (e.g. 'Monday')
- M - month without leading zero
- MM - month with leading zero
- MMM - short month (e.g. 'Jan')
- MMMM - long month (e.g. 'January')
- yy - 2-digit year
- yyyy - 4-digit year

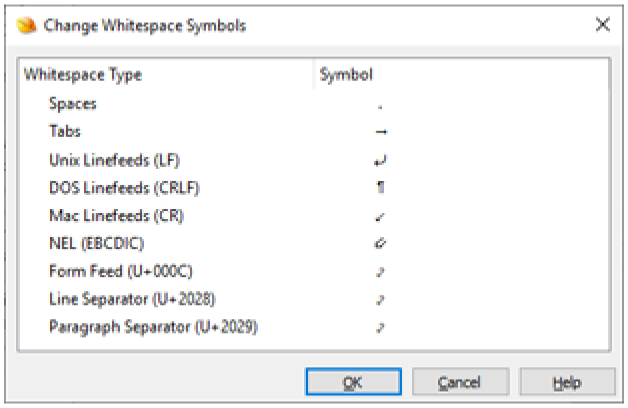
- **Insert Color Format -** This option determines the format of a color when inserted using the '*Edit > Insert Color*' menu option. The following special codes are available for use within the color format:
    - RR - insert the red component of the color in hex notation
    - GG - insert the green component of the color in hex notation
    - BB - insert the blue component of the color in hex notation

Line Length

- **Maximum Line Length -** When using the Text Editor with Word Wrap turned off, any long lines in the file will be split into multiple lines in the editor. Any line that is generated by splitting a long line is indicated by a '-' mark in the Address area on the left side of the editor. The maximum length of each line before it is split can be specified using the *Maximum Line Length* field. Any line length can be used but note that using very long lines may cause performance issues on some computers.

Show Whitespace

The *Show Whitespace* group controls options when whitespace visualization is turned on using the '*View > Tabs/Whitespace > Show Whitespace*' menu option on the View Menu. When the *Display Linefeeds* toggle is enabled a symbol is drawn at the end of each line in the Text Editor to indicate the type of linefeed the line uses. If *Do Not Display Linefeeds* is chosen then no symbol is displayed and if *Only Display Linefeeds if Different than the File's Default Linefeed Type* is chosen then a symbol is only drawn at the end of the line if the line contains a different type of linefeed (for example if editing a file containing DOS linefeeds and the file also contained a few lines with Unix linefeeds, only the Unix linefeed symbols would be shown).



Whitespace Type	Symbol
Spaces	·
Tabs	→
Unix Linefeeds (LF)	↵
DOS Linefeeds (CRLF)	¶
Mac Linefeeds (CR)	↙
NEL (EBCDIC)	[illegible]
Form Feed (U+000C)	↗
Line Separator (U+2028)	↗
Paragraph Separator (U+2029)	↗

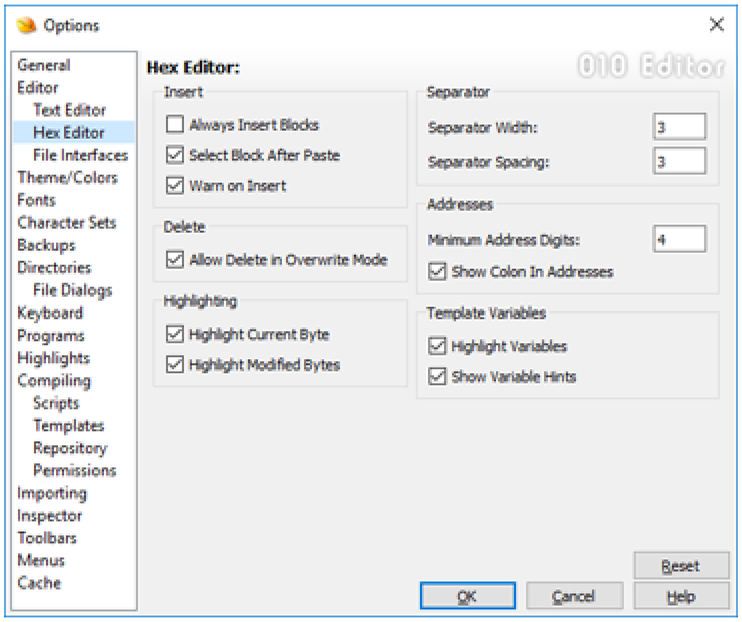
To control which symbol is displayed in the Text Editor for each type of whitespace type click the *Change Whitespace Symbols* button. Click on a symbol in the table and select a new symbol using the Change Symbol dialog. Note the color of the whitespace symbols can be controlled using the *Show Whitespace* color in the Theme/Color Options dialog.

Press the *Reset* button to return all Text Editor Options and whitespace symbols to their default values.

Related Topics:
Editor Options
File Interface Options
Theme/Color Options
Using the Text Editor
View Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Hex Editor Options



Hex Editor:

Insert
- ☐ Always Insert Blocks
- ☑ Select Block After Paste
- ☑ Warn on Insert

Delete
- ☑ Allow Delete in Overwrite Mode

Highlighting
- ☑ Highlight Current Byte
- ☑ Highlight Modified Bytes

Separator
- Separator Width: 3
- Separator Spacing: 3

Addresses
- Minimum Address Digits: 4
- ☑ Show Colon In Addresses

Template Variables
- ☑ Highlight Variables
- ☑ Show Variable Hints

The Hex Editor Options dialog controls options when using the Hex Editor Window (see Using the Hex Editor for more information). Open the Hex Editor Options window by clicking '*Tools > Options...*' and choosing *Hex Editor* from the list.

Insert

- **Always Insert Blocks -** By default, when a block is pasted into the current editor using '*Edit > Paste*' or *Ctrl+V*, two results are possible: The block will be inserted when in *Insert* mode, or written over the current bytes when in *Overwrite* mode (see Using the Clipboard for details). When this toggle is enabled, the block will always be inserted, regardless of the current mode.
- **Select Block After Paste -** When this toggle is enabled and a block is pasted using '*Edit > Paste*' or *Ctrl+V*, the inserted bytes will be selected. If the toggle is disabled, no bytes will be selected.
- **Warn on Insert -** Every time that data is inserted into a file (either using '*Edit > Paste*' or typing in the editor while in *Insert* mode), the Status Bar will show a warning message in orange that bytes were inserted. If this toggle is turned off, just a regular message will be displayed in the Status Bar.

Delete

- **Allow Delete in Overwrite Mode** - When this toggle is turned on and the Delete key is pressed while the hex editor is in *Overwrite* mode, the current selection will be deleted from the file. If no selection is made, the byte the cursor is on will be deleted. When this toggle is disabled no deletions will be allowed

Copyright © 2003-2019 SweetScape Software

in *Overwrite* mode and switch to *Insert* mode to delete bytes.

Highlighting

- **Highlight Current Byte -** There are two possible areas in the Hex Editor Window: the left area, and the right area. When the cursor is in one area, the current byte in the other area will be highlighted gray by default. If the toggle is turned off, the byte will no longer be highlighted.
- **Highlight Modified Bytes -** As changes are made to a file, the text in the Hex Editor will change to orange to indicate where modifications were made. By turning this toggle off, modified bytes will be displayed the same as unmodified bytes.

Separator

- **Separator Width -** The Separator is a vertical line that separates the left and right editing areas in a Hex Editor Window. Enter the width in pixels of the Separator in this field.
- **Separator Spacing -** A small space is drawn immediately to the left and to the right of the Separator in the Hex Editor. This field controls the width in pixels of that space.

Addresses

- **Minimum Address Digits -** Controls the minimum number of digits displayed for addresses on the left side of the Hex Editor Window (see Using the Hex Editor). Usually the number of digits is 4 and expands if the address is too large. Enter a value between 4 and 16 inclusive.
- **Show Colon In Addresses -** If this toggle is enabled, colons are inserted into hexadecimal addresses to divide the addresses into groups of 4 digits. If the toggle is disabled, no colons are displayed.

Template Variables

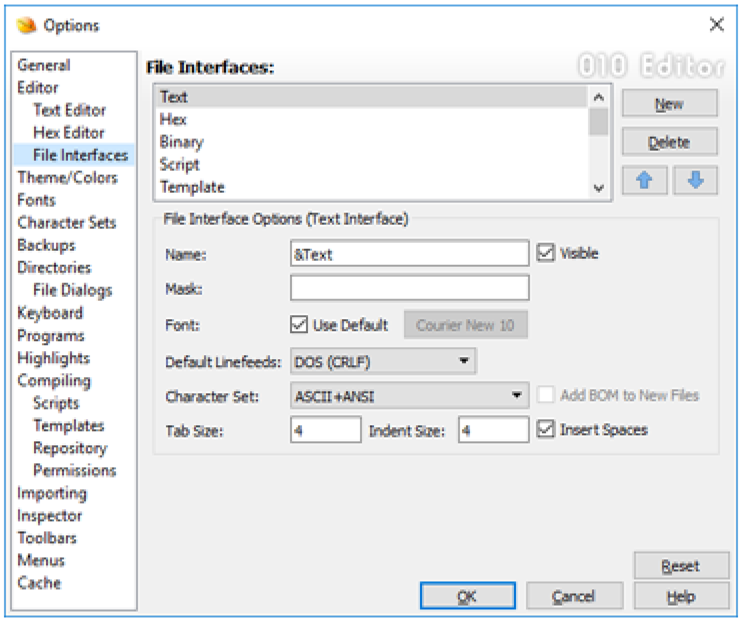
- **Highlight Variables -** After a Binary Template has been run on a file (see Introduction to Templates and Scripts and Working with Template Results), moving the mouse over the Hex Editor Window will display brackets to indicate where the declared template variables exist. If this toggle is disabled, the brackets will not be shown. The color of the brackets can be controlled from the Color Options dialog.
- **Show Variable Hints -** After a Binary Template has been run on a file, positioning the mouse over the Hex Editor Window where a template variable is defined will cause a hint to popup showing the value of the template variable. Turn this toggle off to prevent the hint from displaying. See Working with Template Results for more information.

The *Reset* button can be used to return all of the Hex Editor Options to their default values.

Related Topics:
Color Options
Introduction to Templates and Scripts
Using the Clipboard
Using the Hex Editor
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

File Interface Options



File Interfaces:

Text
Hex
Binary
Script
Template

File Interface Options (Text Interface)

Name: &Text ☑ Visible
Mask:
Font: ☑ Use Default Courier New 10
Default Linefeeds: DOS (CRLF)
Character Set: ASCII+ANSI ☐ Add BOM to New Files
Tab Size: 4 Indent Size: 4 ☑ Insert Spaces

A File Interface includes all of the following options: Font, Addresses, Character Set, Linefeeds/Line Width, Tabs/Whitespace, Addresses, Group By, Areas, Highlight, Division Lines, Ruler, and Endian (basically all options in the top part of the View menu). Consult Working with File Interfaces for more information. Each file that is loaded is assigned a File Interface from the *File Interfaces* list; however, different interfaces can be generated and applied automatically to different files by file type. The File Interface Options dialog allows creating or deleting interfaces for editing. Access this dialog by clicking the '*Tools > Options...*' menu option and selecting *File Interfaces* from the list, or by clicking '*View > Edit As > Edit File Interfaces...*'.

A list of all available file interfaces is displayed at the top of the dialog. Clicking the *New* button will generate a new file interface and you must specify whether you would like to create a text-based file interface or a hex-based file interface. Select an item from the list and click *Delete* to remove an interface. Clicking the up or down arrows will change the position of the interface in the list.

After a file interface is selected from the list, options for that interface will be displayed in the *File Interface Options* box. Enter a name for the interface in the *Name* field. This name will appear under the '*View > Edit As*' menu and the *Edit As* drop-down list in the File Bar above each editor. Clicking the *Visible* toggle provides any easy way to show or hide the interface on the menu.

When a file is opened, it is automatically assigned an interface based on the *Mask* field. The file mask may contain the characters '*' or '?' to indicate wildcards and is not case sensitive. For example, use '*.bmp' to match all BMP files, or 'C:\temp\*.0??' to match all files under the temporary directory with an extension starting with '0'. If multiple masks match a file, the last matching interface in the list will be applied to the file. Note that if no file mask matches the file, 010 Editor will automatically try to detect the correct file interface to use (see *Opening Files* in the Editor Options dialog for more options). 010 Editor can remember the last used File Interface for a file so this method of applying File Interfaces only applies to files that have not been opened before in 010 Editor.

If the *Use Default* toggle is enabled, the file interface uses either the default text editor font or the default hex editor font from the Font Options dialog. If the *Use Default* toggle is disabled, a custom font can be set for the interface by clicking the *Font* button and using the standard font dialog to select a font. If the file interface is for a text-based file the *Default Linefeeds* drop-down list is used to choose which linefeeds to insert when a new file is created using this File Interface. The linefeeds to choose depend upon which type of character set is chosen with the *Character Set* option:

- **Unicode -** Linefeed choices include *DOS* (CR+LF - 0x000D000A), *UNIX* (LF - 0x000A), *Mac* (CR - 0x000D), *NEL* (0x0085), *FF* (0x000C), *LS* (0x2028), or *PS* (0x2029).
- **EBCDIC -** Linefeed choices include *DOS* (CR+LF - 0x0D25), *UNIX* (LF - 0x25), or *NEL* (0x15).
- **All other Character Sets -** Linefeed choices include *DOS* (CR+LF - 0x0D0A), *UNIX* (LF - 0x0A), or *Mac* (CR - 0x0D).

If the file interface is for a hex-based file the *Line Width* drop-down list can be used to indicate *Fixed Width* or *Auto Width*. When using *Fixed Width* mode the field to the right of the *Line Width* drop-down list is used to enter the number of bytes per line. Choose the character set using the *Character Set* drop-down list and if the *Add BOM to New Files* toggle is enabled, a Byte-Order Mark will be added to files when they are created (note that Byte-Order Marks are only used for Unicode or UTF-8 files and see Byte-Order Marks for more information). Options for the tabs can be controlled through the *Tab Size*, *Indent Size* and *Insert Spaces* fields. See Working with File Interfaces for more information on tabs. If the *Add To New Drop-Down Menu* toggle is enabled, an entry will be added to the '*File > New*' menu and to the New drop-down menu which is accessed by clicking the small arrow to the right of the New icon in the Tool Bar.

All created file interfaces can be accessed by clicking the '*View > Edit As*' menu option or the *Edit As* drop-down list in the File Bar above each editor. The '*View > Edit As > Create New File Interface*' menu option provides any easy way to generate a new file interface for a file type (see View Menu for more information).

The following File Interfaces are available by default:

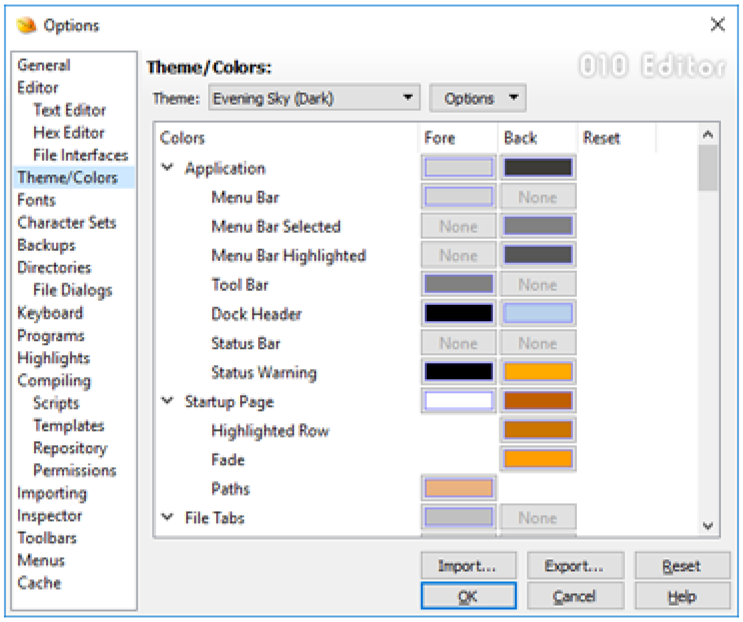
- **Text -** used for editing text (ASCII) files.
- **Hex -** used for editing hex (binary) files in hexadecimal notation (e.g. FF 3A).
- **Binary -** used for editing hex (binary) files in binary notation (e.g. 00101001).
- **Script -** used for editing an 010 Editor script file.
- **Template -** used for editing an 010 Editor Binary Template.
- **EBCDIC -** used for editing EBCDIC text files.
- **Unicode -** used for editing Unicode text files.
- **UTF-8 -** used for editing UTF-8 text files.
- **Drive -** used for editing physical and logical drives.
- **Process -** used for editing processes.
- **Code -** used for editing source code such as C/C++ or PHP.
- **Tagged -** used for tagged files such as HTML or XML.

The *Reset* button will restore all file interfaces to their original values.

Related Topics:
Editor Options
Font Options
View Menu
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Theme/Color Options



Theme/Colors:

Theme: Evening Sky (Dark)

Colors	Fore	Back	Reset
Application	⬜	⬛	
Menu Bar	⬜	None	
Menu Bar Selected	None	⬛	
Menu Bar Highlighted	None	⬛	
Tool Bar	⬛	None	
Dock Header	⬛	🟦	
Status Bar	None	None	
Status Warning	⬛	🟧	
Startup Page	⬜	🟫	
Highlighted Row		🟧	
Fade		🟧	
Paths	🟧		
File Tabs	⬜	None	

010 Editor contains a number of *themes* which are a set of colors for all the various user interface elements in the program. A number of different themes exist including themes with dark backgrounds or light backgrounds and the Theme/Colors Options dialog allows choosing the theme (the theme can also be chosen in the Welcome dialog displayed when 010 Editor is run for the first time). Open the Theme/Color Options dialog by clicking '*Tools > Options...*' and selecting *Theme/Colors* from the list. Styles for all the Syntax Highlighting rules are also controlled by this dialog. New themes can be created and the individual colors in the themes can be customized. The following themes are available:

- **Evening Sky -** The default theme with a darker background which some users may find gives less eye strain when used in low-light conditions (nighttime).
- **Blue Sky -** A theme with a white background and blue tabs.
- **Rain Cloud -** Identical to Blue Sky except the tabs are displayed as gray.
- **Day & Night -** Uses a light theme for the editor and a dark theme for the application background.
- **Midnight -** Similar to Evening Sky except the background is black.
- **Classic -** A theme similar to older versions of 010 Editor.

If a theme has been imported using the *Import...* button at the bottom of the dialog then a new theme named *Custom* will also be available in the list followed by the file name of the theme in brackets. Clicking the *Export...* button will save the current theme to a file including any modified colors.

Some user interface elements are drawn using a *native* style, meaning the operating system draws those elements using its standard drawing procedures (for example the Mac menu bar and the Mac status bar are always drawn by the operating system). 010 Editor overrides some of this drawing with its own user interface styles but the native style can be turned on for some elements by clicking the *Options* button and turning on native drawing for either the Menu Bar, Tool Bars, Dock Headers (the title above each dock window), the Dock Window tabs or File tabs. Also when switching themes when individual colors have been modified by the user, 010 Editor will ask whether to *Reset* the modified colors or to *Keep* the modified colors. If the *Reset Colors when Switching Themes* option is set to *Ask* then a dialog is displayed asking for the user choice but if the choice has already been set the *Reset Colors when Switching Themes* option will be set to *Yes* or *No*.

The *Colors* table allows customizing individual colors in each theme. Click the color box in the *Fore* column to set the foreground (text) color of the item or click the color box in the *Back* column to set the background color of the item. If a color has been modified the color name is drawn in Bold and a *Reset* button appears in the *Reset* column. Clicking the *Reset* button returns just that color to its original value. Some colors may be set to *None*, which means that the foreground or background color will be inherited from what is drawn behind the element (or in some cases setting to *None* means the default operating system color will be used instead).

Click on a color box to open up the Color Selector Box. Select a color from the box, or click the *More Colors...* button to open the standard Color Selector Dialog. When using the Color Selector Dialog, select a color from either the color boxes on the left or the color area on the right and click the *OK* button. Clicking the *Add to Custom Colors* button adds the current color to the list of 16 colors on the bottom left and this list is also displayed at the bottom of the color selector. The *Cancel* button will close the dialog without selecting a color.

The following lists each of the color options and what they control. Note that when each color is changed the results usually are updated in the application immediately, making it easier to see how color changes will affect the application.

Application Colors

- **Application -** The color of the main 010 Editor application window.
- **Menu Bar -** Controls the color of the menu bar at the top of the application window. A color of None means the default operating system colors are used. Menus can also been drawn using native drawing by using the *Options* drop-down menu.
- **Menu Bar Selected -** When the contents of a Menu Bar item are being displayed (for example, the *File* menu) the Menu Bar item is displayed using this color.
- **Menu Bar Highlighted -** When the mouse is hovered over a Menu Bar item that item is displayed using this color.
- **Tool Bar -** Controls the color of the Tool Bars. The foreground color sets the color of the handle and arrows in each Tool Bar. A color of None means the Application color is used instead. Tool Bars can also be drawn using native drawing with the *Options* drop-down menu.
- **Dock Header -** Sets the color of the header of the top of each Dock Window. Dock headers can be drawn using the system (native) style by using the *Options* drop-down menu.
- **Status Bar -** The color of the status bar at the bottom of the Application. Note that on macOS the status bar background is always drawn using the native style.
- **Status Warning -** If an error occurs when performing an operation, an error message is sometimes displayed in the Status Bar. The color of the error message will be displayed as this color (by default orange).

Startup Page Colors

- **Startup Page -** The foreground and background color of the Startup Page.
- **Highlighted Row -** When the mouse hovers over an item in the *Recent Files* list that item will be displayed with this background color.
- **Fade -** Controls the fade at the very top of the Startup Page. If this color is set to None then no fade is drawn.
- **Paths -** Sets the color of the paths displayed at the right-hand side of the *Recent Files* list.

File Tab Colors

- **File Tabs -** Controls the foreground and background color of the unselected tabs in the list of tabs used to choose the active file above each editor. A background color of None means the Application background color is used.
- **Selected Tabs -** The selected (active) tab in the list of documents is drawn using this color.
- **Highlighted Tabs -** When the mouse is hovered over a tab in the list of files that tab is drawn using this color.

- **Division Line -** Sets the color of the line underneath the File Tabs.

Dock Window Tab Colors

- **Dock Window Tabs -** This option controls the color of all unselected tabs in the main application when Dock Windows are docked together to create a set of tabs.
- **Selected Tabs -** The color of the selected tab in the list of Dock Windows in the main application.
- **Highlighted Tabs -** Controls the color of the tab in the Dock Windows which is displayed as highlighted when the mouse moves over the tab.
- **Division Line -** The color of the separator line displayed immediately above the Dock Window Tabs.

Dock Window Colors

- **Dock Windows -** Sets the default text and background color for the Dock Windows, which include the Workspace, Inspector, Output Windows, etc. A color of None means the Application color is used.
- **Workspace -** Controls the text and background color of the Workspace. A color of None means the Dock Windows color is used.
- **Explorer -** The text and background color of the Explorer. Setting the color to None uses the Dock Windows color.
- **Functions -** Sets the color of the Functions tab and a color of None uses the Dock Windows Color.
- **Output -** The text and background color of the Output Windows. A value of None means the Dock Windows color is used. Note that changing this color will not affect any text already present in the Output Window.
- **Output Warning -** Controls the color of warnings in the Output Window as a result of running a Script or Template. Note that changing this color will not affect any warnings already present in the Output Window.
- **Output Error -** The color of any error messages display in the Output Window. Note that changing this color will not affect any errors already present in the Output Window.

Table Colors

- **Tables -** Controls the foreground and background color of tables in the application, including the Inspector, Template Results, Bookmarks, Find Results, etc. A color of None means the Application color is used.
- **Alternating Rows -** Controls the color of every second row in any of the Tables.
- **Selected Row -** The color of the currently selected row in a table when the table has input focus.
- **Inactive Selected Row -** Controls the color of the selected row in a table when the table does not have input focus.
- **Header Row -** The header row is a special row of data in a table to show the start of a block of information. For example in the Find In Files results the results for each file begin with a header row. This color gives the foreground and background color of the row.
- **Header Outline -** Controls the box around the Header Row in a table and setting to None means that no box is drawn.
- **Edit -** This color is used when a cell of the table is being edited, for example in the Inspector or the Template Results.
- **Paths -** The text color used when displayed paths on the right side of the Workspace.
- **Highlighted Row -** Some tables highlight the row the mouse is currently over (for example in the Workspace). The highlighted row is drawn in this color.

Graph Colors

- **Graphs -** Controls the color of various graphs displayed in the Output Windows, such as the Find Results, Histogram Results, etc. The foreground color controls the box around the graph.
- **Find Item -** The color a line in the Find Results graph which shows where a find occurrence is in a file.
- **Find Selected -** Show which Find occurrence is currently selected in the Find table.
- **Find Marker -** Beside the currently selected Find occurrence in the Find graph, two small arrows are drawn. This color controls the color of those arrows.

- **Compare Difference -** Controls the color of a difference in the Compare results graph.
- **Compare Only In -** The color of bytes that are only in one file in the Compare results graph.
- **Compare Selected -** The outline of the range that is currently selected in the Compare results graph.
- **Histogram Item -** Color of the bars in the Histogram results graph.
- **Histogram Alt Item -** Every second bar in the Histogram results graph is drawn in an alternate color. This option controls the alternate color.
- **Histogram Selected -** The color of the select bar in the Histogram results graph.
- **Process Read -** Controls the color of heaps in the Process graph that are marked as read-only.
- **Process Read/Write -** The color of heaps in the Process graph that are readable and writable.
- **Process Selected -** Indicates the color of the currently selected heap in the Process graph.
- **Labels -** Sets the color of text labels in the different output graphs.
- **Empty Graph -** The color of a graph which currently contains no data.

App Bar Colors

- **App Bars -** App Bars are thin bars displayed below an editor which are currently used for Find, Replace, Goto and Selecting a Range. This option controls the color of the App Bars and a color of None means the Application color is used.
- **App Bar Line -** Controls the color of the line displayed above each App Bar. A color of None means no line is displayed.
- **App Bar Arrows -** Sets the color of a number of upward facing arrows on the App Bar that popup additional panels.
- **App Bar Info Text -** The color of an special information text on the App Bars (currently used for the hex-bytes display on the Find bars).

Editor Colors

- **Editor -** Indicates the foreground and background colors of the main Editor Window (see Using the Text Editor and Using the Hex Editor for more information).
- **Selected -** After selecting a set of bytes (see Selecting Bytes), those bytes are drawn in a different color. By default, the text color is changed to white and the background to blue.
- **Highlight Line -** By default, the line the cursor is located on will be drawn a different color which can be controlled with this option. The *Highlight Line* option can be turned on or off (see Editor Options).
- **Addresses -** Specifies the color of the addresses along the left side of the Text Editor or Hex Editor Window. If set to None the Application color is used instead.
- **Addresses Line -** The color of the line immediately to the right of the addresses. If this color is set to None then no line is drawn.
- **Addresses Hover Marker-** When '*View > Addresses > Show Addresses*' is off, a small vertical line is drawn at the left side of the address column when the mouse is placed over the address column. This option specifies the color of the line and to turn off the line display set the color to None.
- **Addresses End Marker-** When '*View > Addresses > Show Addresses*' is off, a triangle is displayed in the address column to mark the last line in the file. Set the color of the triangle using this option or set to None to hide the triangle.
- **Ruler/File Bar -** Indicates the color of the ruler and the File Bar along the top of each Editor Window (when enabled). If set to None the Application color is used instead.
- **Ruler Line -** Controls the color of the line underneath the Ruler. If set to None then no line is drawn.
- **Ruler Marker -** The color of the small arrow which indicates the current column in the Text Editor Window ruler can be controlled with this option. The default color is gray.
- **Input Method Editor -** Some languages use an Input Method Editor (IME) to insert characters into the Editor Window. When the IME is displayed, its color will be controlled by this option.
- **Caret -** This option controls the color of the caret (the blinking cursor) in either the Hex Editor or Text Editor Window.
- **Inactive Caret -** Sets the color of a line indicating the current insertion point even when an editor does not have any input focus. This is called the *Inactive Caret* and can be turned off using the Editor Options dialog.
- **Highlighting -** This highlighting color is used when highlighting a set of bytes using '*View > Highlighting*' (see Working with File Interfaces for more information).
- **Bookmarks -** Controls the color of Quick Bookmarks as displayed in the Text or Hex Editor, and any other bookmarks created when the *Use Custom Color* toggle is turned off in the Add Bookmark dialog.
- **Show Whitespace -** When Show Whitespace is enabled for a text file, special symbols are drawn in the

editor to indicate where space and tab characters exist. This option controls the color of the symbols that are drawn.
- **Breakpoint -** Specifies the color of breakpoints in the debugger. Note that the foreground color is only used when '*View > Addresses > Show Addresses*' is enabled.
- **Debug Active Line -** Controls the color of the active line marker in the debugger. The foreground color is only used when '*View > Addresses > Show Addresses*' is turned on.

Hex Editor Colors

- **Modified -** When bytes are modified in the Hex Editor, the colors of those bytes are changed. By default, the text color is changed to orange and the background color remains unchanged.
- **Alternating Hex Lines -** When using the Hex Editor, alternating lines are displayed in a different color. Change this color to match the *Editor* background color to obtain a single color in the background.
- **Highlight Byte -** When the cursor is in the left or the right editing areas, the current byte will be highlighted in the other editing area. The byte is colored light gray by default, but can be changed by clicking the color box.
- **Highlight Variable -** After a Template has been run on a file, moving the mouse over the Hex Editor Window will cause brackets to display indicating where the template variables were declared (see Working with Template Results). The color of the brackets can be modified by clicking this color box. The *Highlight Variable* option can be turned on or off using the Hex Editor Options dialog.
- **Empty Area -** The color to the far right of both editing areas in the Hex Editor Window can be controlled by clicking this color box. The default color is the color of the window.
- **Area Separator -** The Separator is the line that separates the left area from the right area in a Hex Editor Window. Click this color to change the Separator color.
- **Division Lines -** Division Lines are lines that are drawn on the Hex Editor Window that indicate groups of bytes. By default, Division Lines are drawn every 4 bytes and they are colored light gray. Use the '*View > Division Lines*' menu to adjust the division lines (see Working with File Interfaces). Clicking this color box allows changing the color of the Division Lines.
- **Sector Lines -** Sector Lines are similar to Division Lines except they are usually used to visualize where sectors are located on a drive. Sector Lines can be controlled on the same menu as Division Lines '*View > Division Lines*' (see Working with File Interfaces). The color of the sector lines can be changed from the default dark gray using this color box.
- **Template Results Header -** Controls the color of the header for the Template Results panel.
- **Template Results Line -** The color of the line immediately above the Template Results header. Note that setting this color to None will remove the line.

Find

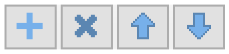
- **Find Results -** After clicking the *Find All* button in the Find Dialog (see Using Find), all occurrences in the file that match the target are colored according to this rule.
- **Find Selection Lock -** When using the Find Bar or Replace Bar, it is possible to limit the Find or Replace to a selected range of bytes. When a Find or Replace is locked to a selection by clicking the *Lock to Selection* button in the *Options* dialog, the selection will then be drawn in this color until the range is unlocked.

Compare

- **Difference -** After performing a comparison between two files (see Comparing Files), those bytes that are different between the files will be colored. This option controls which color will be applied to the bytes in the Hex Editor Window.
- **Only In -** After a comparison between two files (see Comparing Files), bytes that are only in one file and not the other will be colored according to this rule.

Syntax Styles

Use the Syntax Style section to control the color scheme for all of the Syntax Highlighting rules. The use of styles allows multiple Syntax Highlighting rules to share a single color scheme (see Using Syntax Highlighting for more information). For example, both commenting in C++ and PHP share the 'code-comment' style. Changing the 'code-comment' style affects the colors of all rules that use that style. All styles that begin with 'code-' are typically used in programming languages while all styles that being with 'tag-' are used in tag-based (e.g. XML, HTML) languages. Syntax Styles can be assigned a different color depending on if they are being used for a dark theme or a light theme. When a Syntax Style is selected in the list a special group of icons appears at the top-right corner of the dialog:



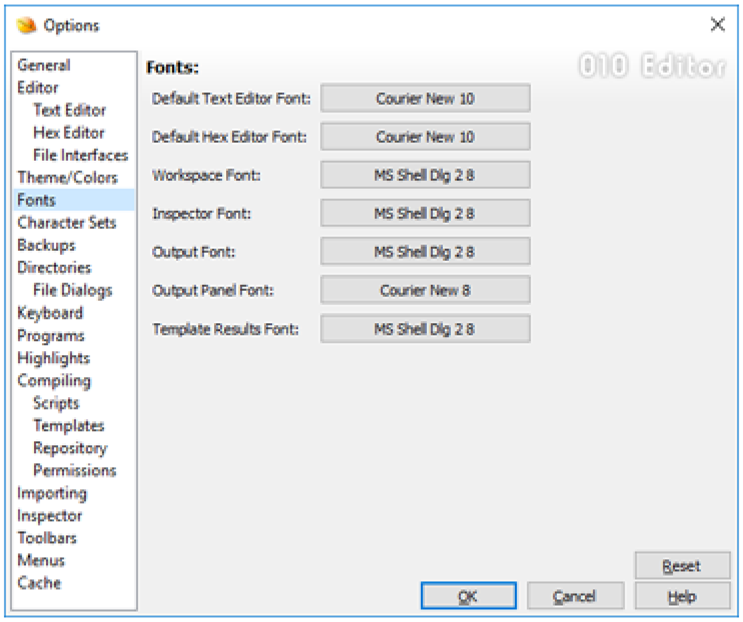
Clicking the Plus icon creates a new style and clicking the X icon deletes the selected style. Note that some Syntax Highlighters automatically create syntax styles if they do not already exist. The Up and Down arrows can be used to move styles to a new position in the list. To rename a style double-click the style name in the list. Note that any created styles are marked as bold in the list and the Reset button will not appear beside the color when it is modified. Styles are exported automatically when the current Theme is exported using the *Export...* button. Styles can also be created in Binary Templates that do Syntax Highlighting if they call the HighlightFindStyle function.

Clicking the *Reset* button will return all of the colors to their default values but note that any created Syntax Styles will be kept but moved to the bottom of the Syntax Styles list.

Related Topics:
Comparing Files
Editor Options
Hex Editor Options
Selecting Bytes
Using Find
Using the Hex Editor
Using the Startup Page
Using Syntax Highlighting
Using the Debugger
Using the Text Editor
Using Tool Bars
Working with File Interfaces
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Font Options



Fonts:

Default Text Editor Font: Courier New 10
Default Hex Editor Font: Courier New 10
Workspace Font: MS Shell Dlg 2 8
Inspector Font: MS Shell Dlg 2 8
Output Font: MS Shell Dlg 2 8
Output Panel Font: Courier New 8
Template Results Font: MS Shell Dlg 2 8

The Font Options dialog allows setting a number of the different fonts used within 010 Editor. Open the Font Options dialog by clicking '*Tools > Options...*' on the main menu and selecting *Fonts* from the list. Click the button to the right of each font description to set the font for that item using the standard font dialog. The following fonts may be set using this dialog:

- **Default Text Editor Font -** By default, all text files open in 010 Editor will use this font for editing; however, different fonts can be assigned for editors by using File Interfaces and turning off the *Use Default* toggle for the File Interface.
- **Default Hex Editor Font -** By default, all hex files open in 010 Editor will use this font for editing. By using File Interfaces, other fonts can be assigned to Editor Windows by turning off the *Use Default* toggle.
- **Workspace Font -** Allows changing the font for the Workspace.
- **Inspector Font -** Sets the font for the Inspector.
- **Output Font -** Sets the font for all tabs of the Output Window except the *Output* tab (see Output Panel Font below). The Output Window displays the results from operations such as Find, Find in Files, Compare, Histograms, Checksums, etc.
- **Output Panel Font -** Controls the font used for the *Output* tab of the Output Window. This *Output* tab displays results from running scripts or templates as well as any output from the Printf function.
- **Template Results Font -** Sets which font to use in the Template Results panel below each editor.

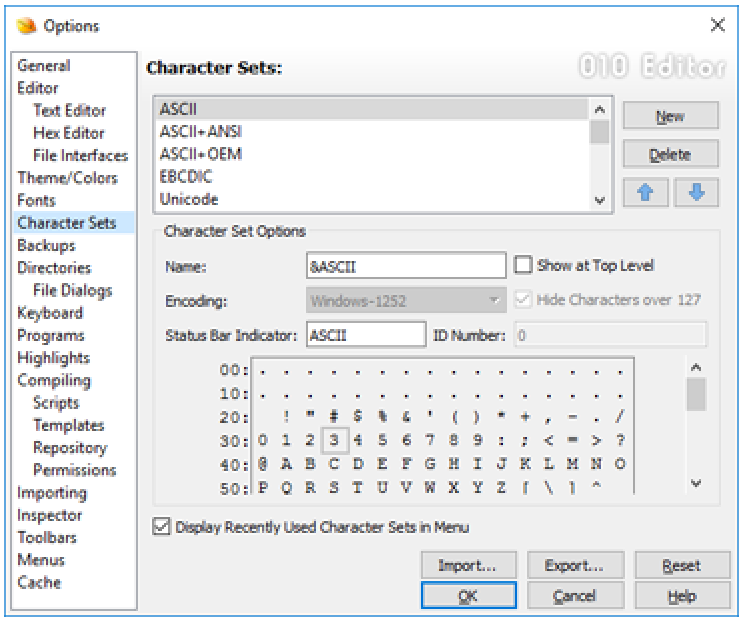
Clicking the *Reset* button will restore all fonts to their default values.

Related Topics:
File Interface Options
Interface Functions
Using the Inspector
Using the Workspace
Working with File Interfaces
Working with Template Results

Copyright © 2003-2019 SweetScape Software

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Character Set Options



Options

General
Editor
Text Editor
Hex Editor
File Interfaces
Theme/Colors
Fonts
Character Sets
Backups
Directories
File Dialogs
Keyboard
Programs
Highlights
Compiling
Scripts
Templates
Repository
Permissions
Importing
Inspector
Toolbars
Menus
Cache

Character Sets:

ASCII
ASCII+ANSI
ASCII+OEM
EBCDIC
Unicode

New | Delete

Character Set Options

Name: &ASCII
☐ Show at Top Level
Encoding: Windows-1252
✓ Hide Characters over 127
Status Bar Indicator: ASCII
ID Number: 0

```
00: . . . . . . . . . . . . . . . .
10: . . . . . . . . . . . . . . . .
20:   ! " # $ % & ' ( ) * + , - . /
30: 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
40: @ A B C D E F G H I J K L M N O
50: P Q R S T U V W X Y Z [ \ ] ^
```

✓ Display Recently Used Character Sets in Menu

Import... | Export... | Reset
OK | Cancel | Help

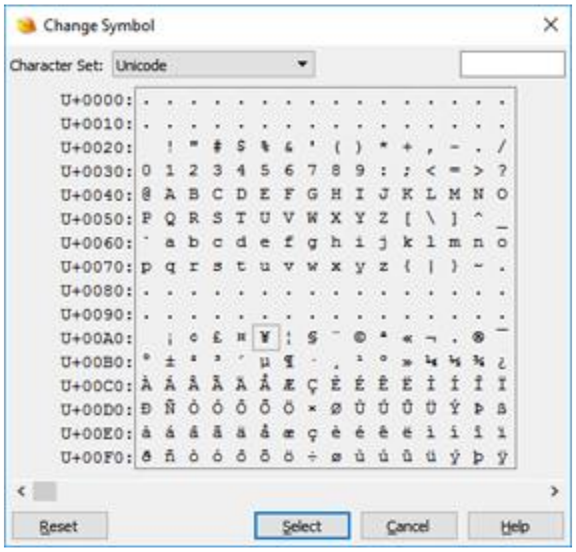
A *character set* is a mapping from a set of raw hex bytes into a set of characters that can be displayed on the screen. The list of all available character sets can be accessed in the *Character Sets* dialog, accessed by clicking the '*Tools > Options...*' menu option and selecting *Character Sets* from the list. To assign a character set to a file use the '*View > Character Set*' menu. Typically all files that use a particular File Interface have the same character set, but a character set can be assigned on a per-file basis by unchecking the '*View > Character Set > Use Default*' toggle.

010 Editor has two main types of character sets: *simple* and *complex* (also called *multi-byte*). Simple character sets (for example ASCII+ANSI) have only 256 different characters and each byte represents a different character. A *complex* or *multi-byte* character set (for example Chinese Simplified) has more than 256 characters and multiple bytes are sometimes needed to represent a single character. For the *Unicode* character set every two bytes indicate one character and Unicode also uses the endian setting from the '*View > Endian*' menu when converting from bytes to characters. Some character sets have a variable number of bytes per character and UTF-8 can have between 1 and 4 bytes per character. Characters are sometimes listed in *Unicode code-point* notation U+XXXX where XXXX is a hexadecimal number. For example the Unicode code-point U+007B is the character '{'. Characters that have no representation in the editor (e.g. control characters) are displayed as '.' in hex-mode or a square in text-mode.

The list of all available character sets is available at the top of this dialog and can be reordered with the Up and Down arrows. Clicking *New* creates a copy of the currently selected character set. The default character sets cannot be modified so a copy of the character set must be created before edits can be made. Clicking *Delete* removes a character set from the list and the '*View > Character Set*' menu but note that any of the standard character sets cannot be deleted.

The name of the character set as appears in the '*View > Character Set*' menu can be modified using the *Name* field. If the *Show at Top Level* toggle is enabled then the character set will be listed at the top of the Character Set menu. The character set will also be listed in either the *Standard*, *International* or *Custom* area of the '*View > Character Set*' menu. The *Encoding* drop-down list can be used to select the underlying character encoding used from the user interface library and note that the encoding cannot be changed for the default character sets (create a custom character set with the *New* button to modify the encoding). The *Hide Characters over 127* toggle is used for the ASCII character set to hide all characters greater than 127. The *Status Bar Indicator* field controls which text is displayed in the Status Bar when the character set is active. The *ID Number* field displays the internal number of this character set that can be used with a number of Template and Script functions. Usually one of the character constants listed in the ConvertString function is used instead of the ID number, but having an ID number for a custom character set allows that character set to be used in Scripts or Templates. Note that ID numbers for custom character sets should be 1000 or greater.

At the bottom of the Character Sets dialog is a table displaying the chosen character set. For simple character sets all 256 characters are displayed in the table and the starting byte value of each line is displayed along the left-hand side of the table. Moving the mouse over the character set table displays a tool tip containing the character, the byte value, and the character number in Unicode code-point notation (U+XXXX). For complex character sets (for example Chinese Simplified) a scroll bar will appear at the bottom of the table to scroll through all possible values. The values on the left side of the table represent two hex bytes that are converted into a character. When viewing the *Unicode* character set, 'U+' is displayed at the left side to indicate that Unicode code-points are being viewed. To change the font of the table right-click on the table and choose *Change Font* or return to the original font by right-clicking on the table and selecting *Reset Font*.



```
U+0000: . . . . . . . . . . . . . . . .
U+0010: . . . . . . . . . . . . . . . .
U+0020:   ! " # $ % & ' ( ) * + , - . /
U+0030: 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
U+0040: @ A B C D E F G H I J K L M N O
U+0050: P Q R S T U V W X Y Z [ \ ] ^ _
U+0060: ` a b c d e f g h i j k l m n o
U+0070: p q r s t u v w x y z { | } ~ .
U+0080: . . . . . . . . . . . . . . . .
U+0090: . . . . . . . . . . . . . . . .
U+00A0:   ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ . ® ¯
U+00B0: ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿
U+00C0: À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
U+00D0: Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
U+00E0: à á â ã ä å æ ç è é ê ë ì í î ï
U+00F0: ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ
```

When viewing a simple custom character set, clicking on a character in the character set table opens up the *Change Symbol* dialog. If clicking on a character of one of the default character sets, you will be asked to copy the character set to a new custom character set before the character set can be modified. Clicking on a symbol in the *Change Symbol* dialog will switch to using that symbol in the character set and the character will be colored orange in the table. By default the dialog will display characters in the *Unicode* character set but other character sets can be chosen with the *Character Set* drop-down list. Click the *Reset* button in the *Change Symbol* dialog to return to using the default character. Alternately a symbol can be selected by entering the Unicode code point in hex notation in the field at the top-right of the dialog and then clicking the *Select* button. Decimal notation can also be used in the field by entering ',d' after the number.

Simple (single-byte) character sets can be exported to a CSV file by clicking the *Export...* button. The exported file will contain 256 values in hex notation separated by commas and each value represents the Unicode code-point of that character. For example:

```
...
0x20AC,0xFFFD,0x201A,0x0192,0x201E,0x2026,0x2020,0x2021,
0x02C6,0x2030,0xFFFD,0x2039,0x0152,0xFFFD,0xFFFD,0xFFFD,
...
```

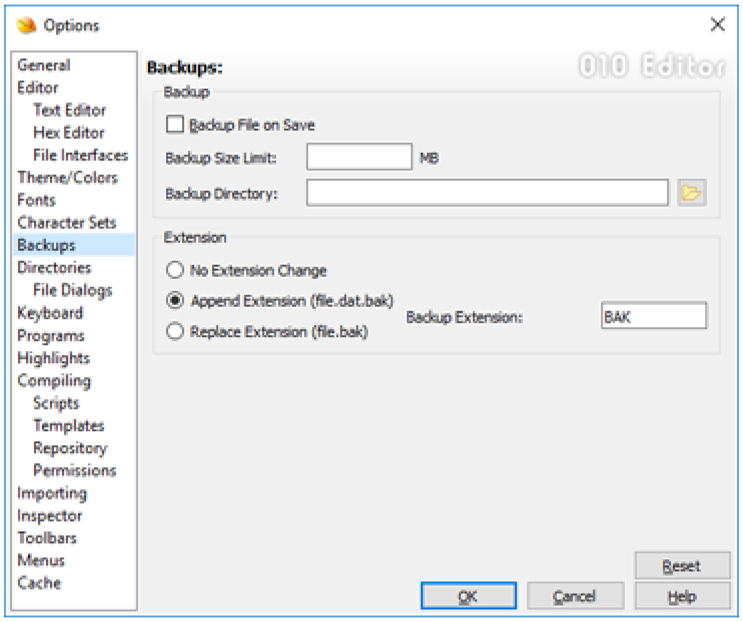
Clicking the *Import...* button will allow importing a simple character set into the program. The imported file should contain 256 numbers in decimal notation or 0xXXXX hex notation separated by commas, spaces or tabs. In an imported file the special number -1 can be used to indicate no change for a character in the table.

By default, a list of up to four recently used character sets which do not have the *Show at Top Level* toggle enabled are shown near the top of the '*View > Character Set*' menu. To disable the display of the recently used character sets turn off the *Display Recently Used Character Sets in Menu* toggle. Click the *Reset* button to delete all custom character sets and restore all default character sets to their original values.

Related Topics:
Working with File Interfaces
Status Bar
Using the Hex Editor
Using the Text Editor
View Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Backup Options



Backups:

Backup

☐ Backup File on Save

Backup Size Limit: MB

Backup Directory:

Extension

○ No Extension Change

● Append Extension (file.dat.bak)

○ Replace Extension (file.bak)

Backup Extension: BAK

The Backup Options controls the creation of backup files when a file is saved in the editor. Access the Backup Options by clicking '*Tools > Options...*' and selecting *Backups* from the list.

To enable the creation of backups, click the *Backup File on Save* toggle box. If the Backup toggle is set, the *Backup Size Limit* lists the size limit in megabytes for automatically making backup files. For example, if the size limit is 10 megabytes, then a backup will not be made when saving a file that is over 10 megabytes. The *Backup Directory* field lists the directory where backups will be saved. If this field is blank, the backups will be saved in the same directory where the file is located. Click the folder button beside the field to use a browse dialog to select a directory.

A number of options exist for controlling the extension of the backup file. If the *No Extension Change* toggle is selected, the backup file name will be the same as the original file name (this is only valid when a directory is specified in the *Backup Directory* field). If the *Append Extension* toggle is set, an extension will be added to the end of the file name when the backup is written. For example, if the original file was '*file.dat*', the backup file will be '*file.dat.bak*'. To replace the file's extension with the backup extension, select the *Replace Extension* toggle. For example, if the original file was '*file.dat*', the backup file would be '*file.bak*'. To change the backup extension, enter a new extension into the *Backup Extension* field (without the period).

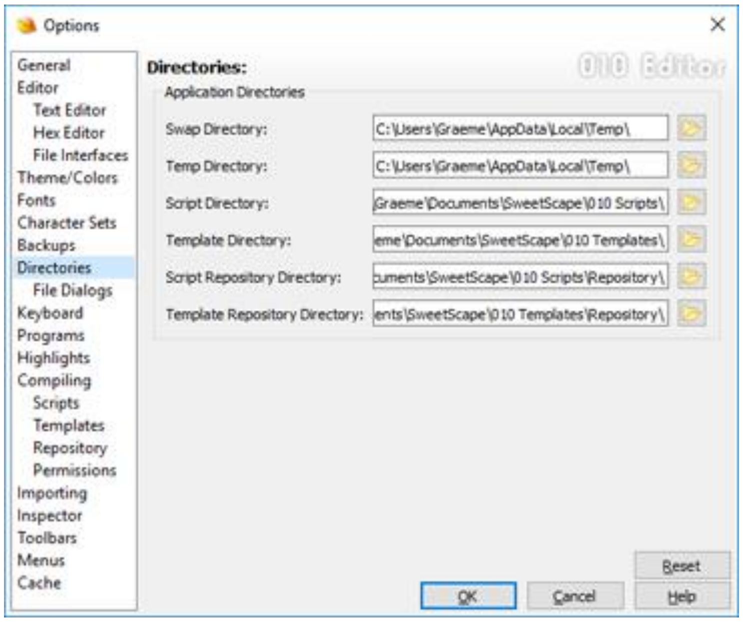
Note that backups will not be made when saving drives or processes.

Clicking *Reset* will return the backup options to their original values.

Related Topics:

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Directory Options



Directories:

Application Directories

Swap Directory: C:\Users\Graeme\AppData\Local\Temp\
Temp Directory: C:\Users\Graeme\AppData\Local\Temp\
Script Directory: Graeme\Documents\SweetScape\010 Scripts\
Template Directory: eme\Documents\SweetScape\010 Templates\
Script Repository Directory: cuments\SweetScape\010 Scripts\Repository\
Template Repository Directory: ents\SweetScape\010 Templates\Repository\

The Directory Options dialog controls the location of various directories used in 010 Editor. To open the Directory Options dialog click the '*Tools > Options...*' menu option and select '*Directories*' from the list.

010 Editor stores file data in a cache in memory which can be controlled using the Cache Options dialog. Once too much data is loaded into the cache, data will be written to a swap file on disk. Enter the directory where the swap file should be stored in the *Swap Directory* field. Click the folder button to use a dialog to select the swap directory from a list. 010 Editor occasionally uses temporary files. Enter the directory where these files should be stored in the *Temp Directory* field or click the folder button to select a directory with a dialog box.

The *Script Directory* field lists the default location where local scripts are assumed to exist on disk (this is used for the ($SCRIPTDIR) variable in the Script Options). Similarly, the *Template Directory* field lists the default directory where local templates are stored (this is equivalent to the ($TEMPLATEDIR) variable in the Template Options).

The *Script Repository Directory* and *Template Repository Directory* fields control where Scripts and Templates are placed when they are installed from the Repository (see Using the Repository Dialog). These directories can be accessed using the constants ($SCRIPT_REPOS_DIR) or ($TEMPLATE_REPOS_DIR) in the Script Options or Template Options dialogs.

Note that if the *Script Directory*, *Template Directory*, *Script Repository Directory*, or *Template Repository Directory* fields are modified, you must physically move the script or template files to the new location otherwise you may get a 'file not found' error when attempting to run a file.

If using the Portable version of 010 Editor all directories will begin with the ($BASEDIR) constant, which is the root directory where the portable version was installed. See Using the Portable Version for more information on the directory structure with the portable version.

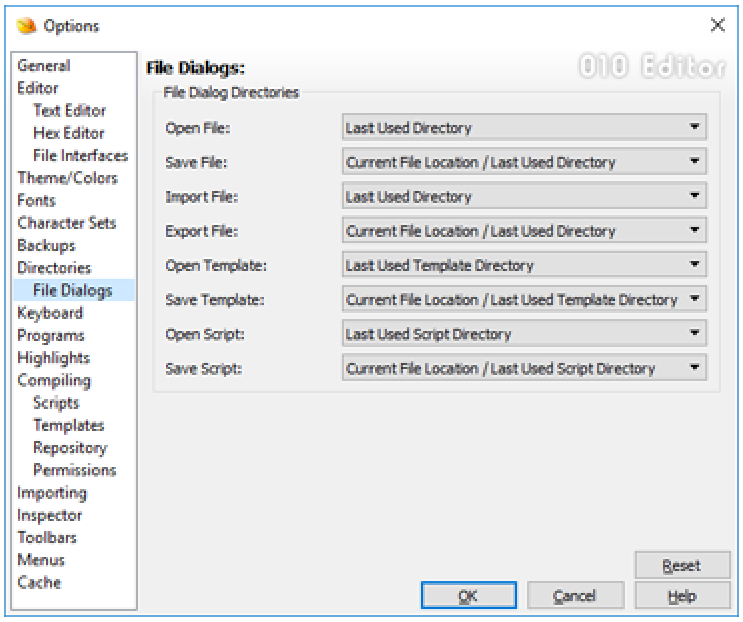
Clicking *Reset* will return all directory options to the default values.

Copyright © 2003-2019 SweetScape Software

Related Topics:
Cache Options
Script Options
Template Options
Using the Portable Version
Using the Repository Dialog

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

File Dialog Options



File Dialogs:

File Dialog Directories

Open File:	Last Used Directory
Save File:	Current File Location / Last Used Directory
Import File:	Last Used Directory
Export File:	Current File Location / Last Used Directory
Open Template:	Last Used Template Directory
Save Template:	Current File Location / Last Used Template Directory
Open Script:	Last Used Script Directory
Save Script:	Current File Location / Last Used Script Directory

Use the File Dialog Options dialog to set the initial directory when opening a file dialog box. Click the '*Tools > Options...*' menu option and select '*File Dialogs*' from the list to view this dialog.

The *File Dialog Directories* section of the dialog controls the initial directory when various file dialogs are opened in the application (for example, when using '*File > Save As*'). The *Open File* and *Save File* options control the directory when opening a file or saving a file. Note that when saving a template or script, the initial directory is controlled by the *Save Template* or *Save Script* options respectively. The *Import File* and *Export File* options control the initial directories when importing/exporting hex data. When opening templates or scripts (for example, by using '*Templates > Open Template*' or '*Scripts > Open Script*'), the initial file dialog directories are controlled with the *Open Template* and *Open Script* options. For file dialogs used for opening files there are two options:

- **Last Used Directory -** The file dialog is opened in the application global last-used directory. This directory is remembered from the last time a dialog was used to open or save a file which is set to use the *Last Used Directory* option.
- **Last Used *???* Directory -** where *???* is *Open*, *Save*, *Import*, *Export*, *Template* or *Script*. This option allows the file dialog to use a last-used directory specific to that option and the application global last-used directory is not touched. For example, by default when opening or saving templates a special last-used templates directory is used instead of the global last-used directory.

When saving files two additional options are available:

- **Current File Location / Last Used Directory -** If the file being saved contains a valid directory, the file dialog is opened in that directory. If the file being saved does not contain a valid directory (for example, when a file is first created), the file dialog is opened in the application global last-used directory. To ignore the current file location when saving, click the drop-down list and select *Last Used Directory*.
- **Current File Location / Last Used *???* Directory -** Similar to the *Current File Location / Last Used Directory* option, if the file being saved contains a valid directory the file dialog will be opened in that

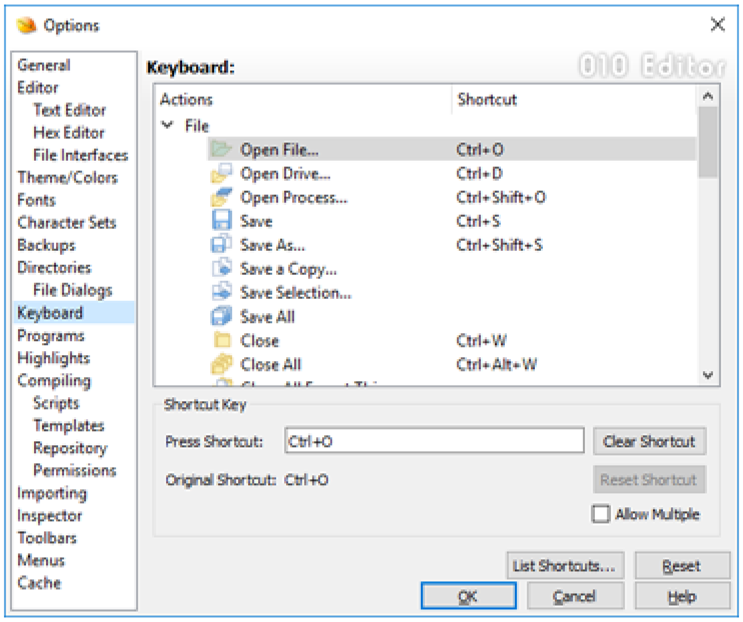
directory. If the file does not contain a directory it will be opened in a directory as described in the *Last Used ??? Directory* option above. To ignore the current file location, select the *Last Used ??? Directory* option from the list.

Clicking *Reset* will set all File Dialog options to their default values.

Related Topics:
Importing/Exporting Files
Introduction to the Data Engine
Opening Files
Saving Files
Script Basics
Template Basics

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Keyboard Options



Actions	Shortcut
File	
Open File...	Ctrl+O
Open Drive...	Ctrl+D
Open Process...	Ctrl+Shift+O
Save	Ctrl+S
Save As...	Ctrl+Shift+S
Save a Copy...	
Save Selection...	
Save All	
Close	Ctrl+W
Close All	Ctrl+Alt+W

Shortcut Key

Press Shortcut: Ctrl+O [Clear Shortcut]

Original Shortcut: Ctrl+O [Reset Shortcut]

☐ Allow Multiple

Keyboard Options allows customization of shortcut keys (also called hotkeys) for many of the operations in 010 Editor. Access the Keyboard Options by clicking the '*Tools > Options...*' menu option and selecting *Keyboard* from the list.

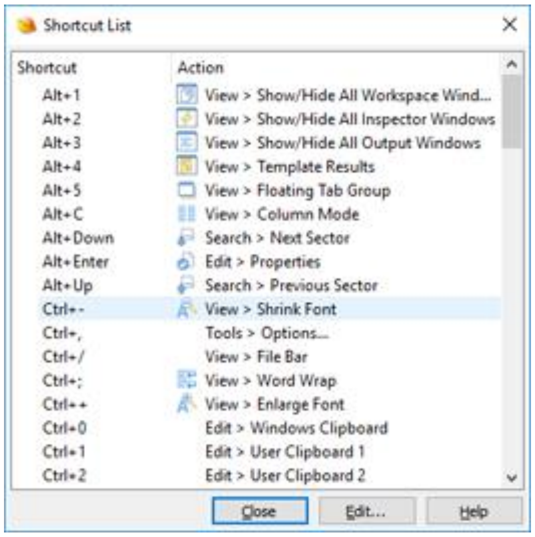
Select an action from the *Actions* list to edit the shortcut for that action. All actions are organized into categories by which menu they exist under in the main application window. Double-click on the category name (e.g. *File*, *Edit*, etc.) to show or hide all the actions in that category. The current shortcut key for each action is listed in the *Shortcut* column.

After an action is selected from the list, click the *Press Shortcut* field and type the shortcut key on your keyboard you would like to use for this action (for example, to use 'Ctrl+L' for a shortcut hold down the 'Ctrl' key and press the 'L' key). Any action that has been modified will appear bold in the *Actions* list. Clicking the *Clear Shortcut* button will remove any shortcut key from the selected action, and clicking the *Reset Shortcut* button will return the action to its original shortcut as indicated by the *Original Shortcut* field. If the shortcut for the current action is already in use by another action, the conflicting action name will be displayed below the *Original Shortcut* field.

Usually actions only have a single shortcut key but this dialog can be used to assign muliple shortcuts to the same action by enabling the *Allow Multiple* toggle. For example, on Windows both 'Ctrl+C' and 'Ctrl+Ins' can be used for the Copy command. When the *Allow Multiple* toggle is on, pressing a shortcut key in the *Press Shortcut* field will add the shortcut to the field separated by a comma. When editing using this method, click the *Clear Shortcut* button to remove the shortcuts from the list.

Clicking the *Reset* button will reset all shortcut keys to their original values. Click the *List Shortcuts* button to display a dialog showing all shortcuts in the application sorted by the shortcut name. Any actions that have been modified will be displayed as bold in the list. This dialog is also accessible by clicking the '*Help > View Shortcut List*' menu item. When the dialog is displayed from the Help menu an *Edit* button will be displayed at the bottom of the dialog and clicking this button will display the Keyboard options dialog.

Copyright © 2003-2019 SweetScape Software

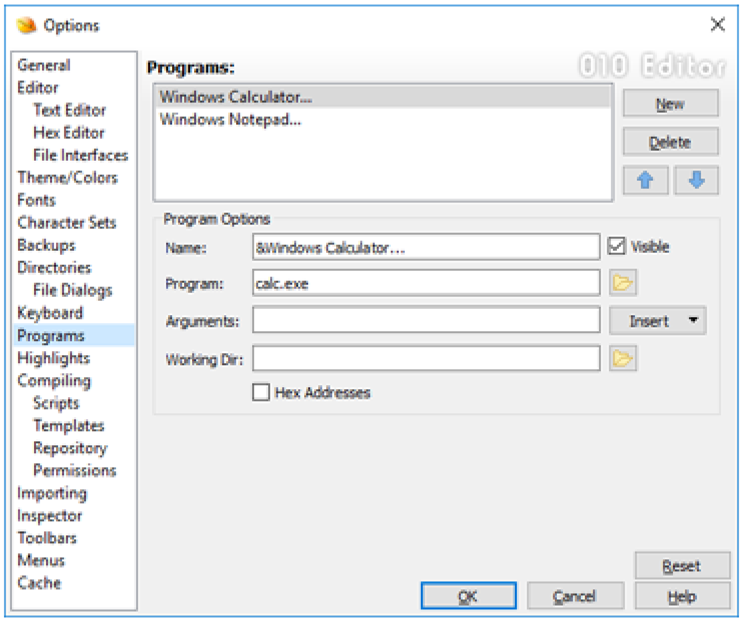


Shortcut	Action
Alt+1	View > Show/Hide All Workspace Wind...
Alt+2	View > Show/Hide All Inspector Windows
Alt+3	View > Show/Hide All Output Windows
Alt+4	View > Template Results
Alt+5	View > Floating Tab Group
Alt+C	View > Column Mode
Alt+Down	Search > Next Sector
Alt+Enter	Edit > Properties
Alt+Up	Search > Previous Sector
Ctrl+-	View > Shrink Font
Ctrl+,	Tools > Options...
Ctrl+/	View > File Bar
Ctrl+;	View > Word Wrap
Ctrl++	View > Enlarge Font
Ctrl+0	Edit > Windows Clipboard
Ctrl+1	Edit > User Clipboard 1
Ctrl+2	Edit > User Clipboard 2

Related Topics:

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Program Options



Programs:

Windows Calculator...
Windows Notepad...

Program Options

Name: &Windows Calculator... ☑ Visible

Program: calc.exe

Arguments:

Working Dir:

☐ Hex Addresses

The Program Options dialog allows adding custom programs on the *Tools* menu (see the Tools Menu for more information). Click the '*Tools > Options...*' menu option and select *Programs* from the list to view this dialog.

The upper area of the dialog contains a list of all custom programs available. Click the *New* button to add a new program to the list, or select an item and click *Delete* to remove a program from the list. The up and down arrows will change the order of items in the list.

After selecting a program from the list, the program's details will be shown in the *Program Options* box. The *Name* field indicates the caption that will appear on the Tools menu and may contains an '&' to indicate the default character. The *Program* field should contain the file name of the program to be run. Click the folder button beside the *Program* field to use a file dialog box to select a file. The arguments for the program can be specified in the *Arguments* field. The following special codes can be used in the argument list and will automatically be replaced by the correct values:

- **File Name (%f) -** Indicates the full path to the currently selected file. This is empty if no file is currently open. If the file name contains a space, quotes will automatically be added around the file name if necessary.
- **Cursor Position (%c) -** Gives the address of the cursor in the file.
- **Selection Start (%s) -** If a selection exists, gives the address of the first byte of the selection.
- **Selection Size (%z) -** If a selection exists, gives the size of the selection.

Note that all positions and sizes are specified in decimal format but can be specified in hex format by enabling the *Hex Addresses* toggle. Clicking the *Insert* button provides an easy way to insert different codes into the argument list. A working directory can be specified for the program by entering a directory into the *Working Dir* field. Click the folder button beside the field to use a directory browser to select the directory. Note that any environment variables can be included in the fields by using the '$' symbol (for example: '($WINDIR)'). The special symbol ($PROGDIR) can be used to indicate the path where 010 Editor exists.

Copyright © 2003-2019 SweetScape Software

By default, the following custom programs are installed:

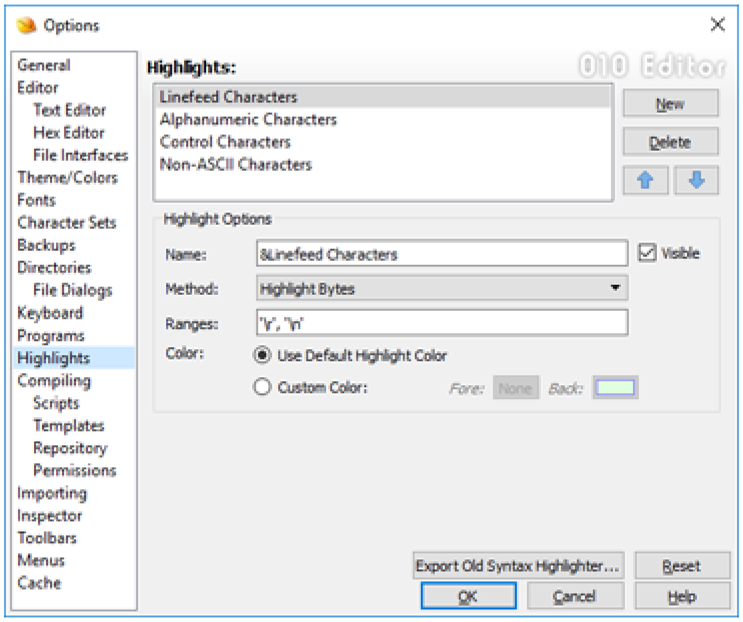
- **Windows Calculator... -** Displays the standard Windows Calculator.
- **Windows Notepad... -** Runs the standard Windows Notepad program with the currently loaded file.

Click the *Reset* button to restore all Program Options to the default values.

Related Topics:
Tools Menu

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Highlight Options



Highlighting provide a way of coloring certain bytes or shorts in a file for identification (a short is a group of two hex bytes). For example, with highlighting all of the bytes with values 128 to 255 could be colored to identify non-ASCII bytes. The Highlight Options dialog can be accessed by clicking the '*Tools* > *Options...*' menu option and selecting *Highlights* from the list, or clicking '*View* > *Highlighting* > *Edit Highlights...*'. Highlights can be turned on or off by clicking on them in the '*View* > *Highlighting*' menu. The enabled highlights are associated with a File Interface (see Working with File Interfaces for more information).

All available highlights are displayed in a list at the top of the dialog. Click *New* to generate a new highlight, or select an item in the list and click *Delete* to remove the highlight. Clicking the up or down arrows will change the selected item's position in the list.

When a highlight is selected from the list, its details will be displayed in the *Highlight Options* box. The name entered in the *Name* field will appear on any menu where the highlight will be accessed. If the *Visible* toggle is turned off, the highlight rule will not appear on any menus. If the *Method* drop-down list indicates *Highlight Bytes*, the *Ranges* field contains which byte values will be highlighted by this rule (values range from 0 to 255). Enter a value in any of the available numeric formats (see Introduction to Number Systems). Multiple values can be separated by the ',' character, or a range of values can be specified using '..'. For example, '0..31,127' will highlight bytes 0 to 31 inclusive, and 127. If the *Method* drop-down list specifies *Highlight Shorts*, enter the range of Shorts to highlight in the *Ranges* field (values range from 0 to 65535). Note that a Short is a group of two hex bytes in a file and how the Shorts are interpreted depends upon the current Endian of the file. Use the *Highlight Shorts* method when applying coloring rules to Unicode files.

The bytes in the current file that match this highlight rule will be colored. If the *Use Default Highlight Color* toggle is enabled, the bytes are colored according to the *Highlighting* color in the Theme/Color Options dialog. If the *Custom Color* toggle is enabled, the bytes are colored according to the *Fore* and *Back* color boxes. Using custom colors, multiple highlights can be turned on at the same time using different colors.

The following Highlight rules are available by default:

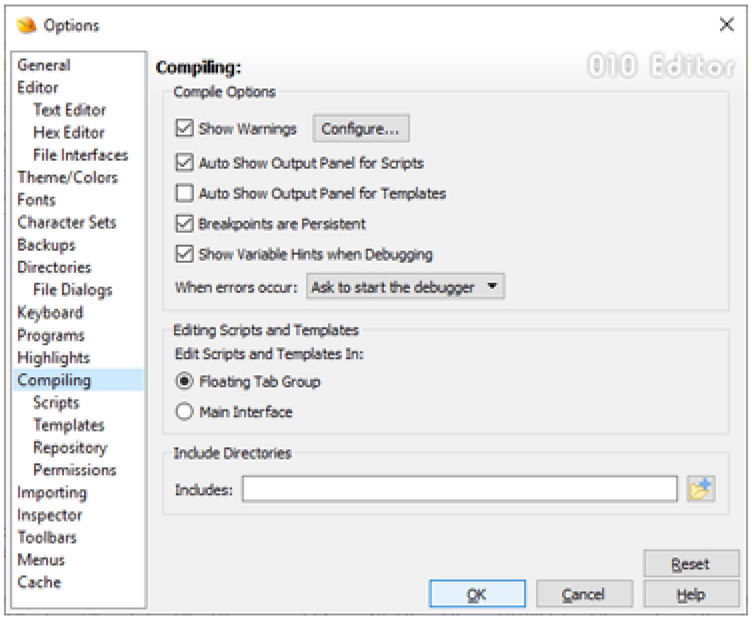
- **Linefeed Characters -** 0x0d and 0x0a
- **Alphanumeric Characters -** all letters and numbers
- **Control Characters -** any of the bytes from 0 to 31 and byte 127
- **Non-ASCII Characters -** any of the bytes from 128 to 255

In older versions of 010 Editor, Syntax Highlighting was performed through the Options dialog. If any old custom Syntax Highlighters are found, the *Export Old Syntax Highlighter* button will appear and clicking on this button allows exporting an old syntax highlighter to an XML file. See the Using Syntax Highlighting help topic for more information about syntax highlighters. Click the *Reset* button to restore all Highlight rules to their original values.

Related Topics:
Introduction to Number Systems
Theme/Color Options
Using Syntax Highlighting
Working with File Interfaces

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Compiling Options



Compiling:

Compile Options

- [✓] Show Warnings
- [✓] Auto Show Output Panel for Scripts
- [ ] Auto Show Output Panel for Templates
- [✓] Breakpoints are Persistent
- [✓] Show Variable Hints when Debugging

When errors occur: Ask to start the debugger

Editing Scripts and Templates

Edit Scripts and Templates In:

- [●] Floating Tab Group
- [○] Main Interface

Include Directories

Includes:

The Compiling Options dialog is used to control some options for running scripts and templates. For more information about compiling see Running Templates and Scripts. The Compiling Options dialog can be accessed by clicking the '*Tools > Options...*' menu item and selecting '*Compiling*'.

The *Compile Options* group controls how scripts and templates are run and how the debugger operates. If the *Show Warnings* toggle is enabled, any warnings generated during compilation will be displayed in the Output tab of the Output Window. Click the *Configure* button to control exactly which warnings are displayed. If the toggle is disabled no warnings will be displayed. If the *Auto Show Output Panel for Scripts* toggle is enabled and a script calls the Printf function to display data, the Output tab of the Output Window will automatically be displayed. If this toggle is disabled, the Output Window will not automatically be shown but can be accessed by the '*View > Output*' menu option. Similarly the *Auto Show Output Panel for Templates* controls whether the Output Window is automatically shown when Printf is called in a template.

The next options are for the debugger. If the *Breakpoints are Persistent* toggle is on then any breakpoints will automatically be saved to disk and reloaded when 010 Editor is shut down and restarted. When the debugger is paused at a line in a Script or Template then placing the mouse cursor over the name of a variable in the Text Editor will attempt to display the value of the variable in a hint popup. If the *Show Variable Hints when Debugging* toggle is turned off then no hints are displayed. When a runtime error occurs while executing a Script or Template, a message box will popup asking to start the debugger if the *Ask to start the debugger* option is chosen. If the *Start the debugger* option is chosen then the cursor is placed at the line where the error occurred and the debugger is started. If *Do not start debugging* is chosen then the Script or Template is stopped as usual. The last two options can also be selected by choosing the *Always use this action* toggle in the message box asking to start the debugger. Note the debugger will not be started if debugging is turned off using '*Debug > Debugging Enabled*'.

By default, when a script or template is loaded in 010 Editor it is displayed in the *Floating Tab Group* (see Using File Tabs) for more information. Scripts and templates can also be loaded in the main tab group by enabling the *Main Interface* toggle.

The *Includes* field contains a list of all directories to be searched when a file is included using the *#include*

Copyright © 2003-2019 SweetScape Software

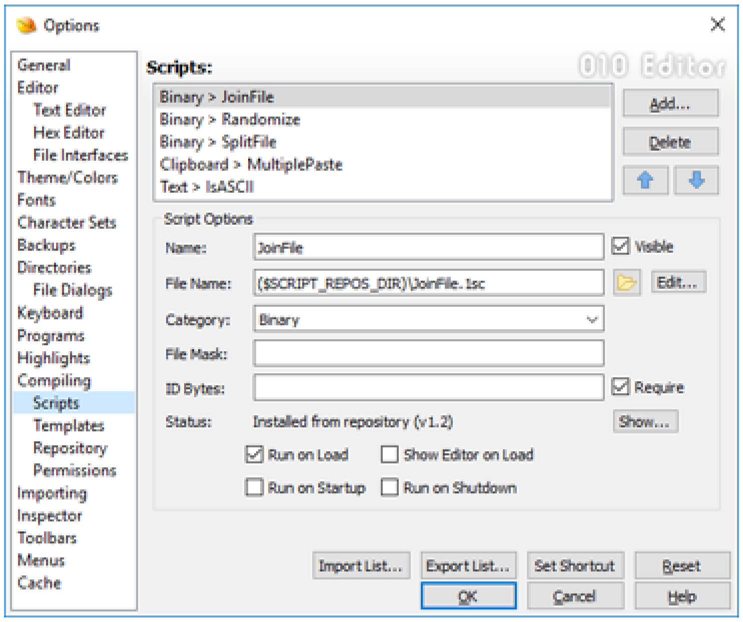
command in a template or script (see the Includes help topic for more information). Enter all directories to be searched separated by spaces in the *Includes* field. If a directory contains spaces, place double quotes around the directory in the field. Clicking the Browse button with the '+' symbol will allow adding a directory to the list using a standard directory select dialog. To control the directories where Scripts or Templates are stored see the Directory Options dialog.

Click the *Reset* button to reset all options to their original values.

Related Topics:
Directory Options
Includes
Running Templates and Scripts
Script Options
Template Options
Using File Tabs
Using the Debugger

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Script Options



Name: JoinFile [✓ Visible]
File Name: ($SCRIPT_REPOS_DIR)\JoinFile.1sc
Category: Binary
File Mask:
ID Bytes: [✓ Require]
Status: Installed from repository (v1.2)
[✓ Run on Load]

The Script Options dialog lists all Scripts that have been installed, either from the local drive or from the 010 Editor Repository. Installed Scripts are listed on the main application *Scripts* menu and can be set to run on startup or shutdown or when certain files are loaded. Click '*Tools > Options...*' and select *Scripts* from the list, or click the '*Scripts > View Installed Scripts...*' menu option to view the Script Options dialog.

A list of scripts is contained in the upper portion of the dialog displayed in the format '*Category > Script Name*'. Click the *Add* button to add a Script from the local drive into the list (note that multiple scripts can be added at the same time using the file dialog that is displayed). Select a script and click *Delete* to remove the script from the list (the file is not deleted from disk). The up or down arrows can be used to change the order of the scripts in the list.

Selecting a script will show its attributes in the *Script Options* box. Enter a name for the script in the *Name* field. This name will appear on the main Scripts menu (see the Scripts Menu) listed under the given *Category*. If the *Category* is empty the Script will be listed near the top of the *Scripts* menu. Turning off the *Visible* toggle provides an easy way to hide a script from the menu without having to delete it. Enter the script file name in the *File Name* field. Scripts usually have the extension '.1sc' and are very similar to C files. Click the folder button beside the field to select a script using a file dialog box or press the *Edit* button to close the Options dialog and view the view in the editor.

If a mask is entered in the *File Mask* field the script will automatically be loaded when a file is opened that matches this mask. File masks may contain the characters '*' and '?' to specify wildcards and multiple masks may be separated by commas. If a value is given in the *ID Bytes* field then the Script will not be loaded unless the *File Mask* matches the file name and the *ID Bytes* match the bytes at the beginning of the file (see Template Options for more information on the ID Bytes). If the selected Script has been installed from the Repository then the version number will be listed in the *Status* field. Clicking the *Show* button will hide this dialog and display information about the script in the Repository Dialog.

If the *Run on Load* toggle is set, the script will automatically be run when it is loaded. Note that if a template is set to load automatically for the same file, the template will be loaded and run before the script. If the *Show Editor on Load* toggle is set, the script will be opened for editing in the interface. Enabling the *Run on Startup* toggle causes the script to be run when 010 Editor is started. The script can also be run while 010 Editor is closing by enabling the *Run on Shutdown* toggle.

The following scripts are available by default:

- **JoinFile -** Combines a number of smaller files created with the *SplitFile* script into one large file. The files to join must contain an order number in their filename (for example, file0000.dat, file0001.dat, etc).
- **Randomize -** Randomizes the current selection. Allows setting a minimum and maximum byte value.
- **SplitFile -** Splits a large file into a number of smaller files (for example, file.dat could be split into file.dat.001, file.dat.002, etc). The size of each file to create can be specified when running the script.
- **MultiplePaste -** Allows data on the clipboard to be pasted multiple times.
- **IsASCII -** Reports whether the current file contains only ASCII characters.

The *Set Shortcut* button can be used to set a shortcut key for the selected script using the Keyboard Options dialog. Click the *Reset* button to restore all scripts to their default values.

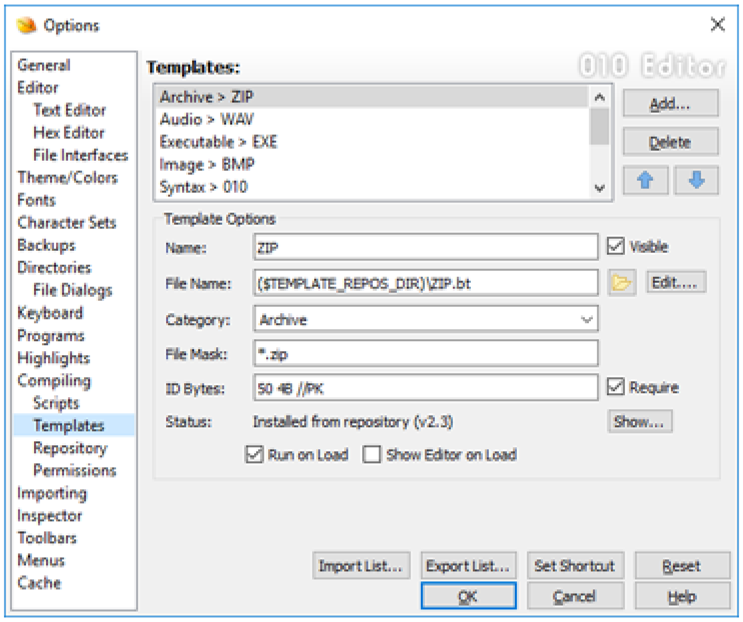
Importing and Exporting Lists of Scripts

The Script Options dialog can export a list of scripts to a file and this list can then be imported into another copy of 010 Editor. The list of scripts can be exporting by clicking the *Export List...* button. Select the file to write using the standard file dialog and the exported script list will have the extension ".1sl". Lists can also be imported by clicking the *Import List...* button. Exporting and importing scripts works the same way as exporting or importing templates. See the Template Options dialog for more information about exporting and importing using and the *Import Script List* dialog.

Related Topics:
Introduction to Templates and Scripts
Introduction to the Repository
Template Options
Scripts Menu
Using the Repository Dialog

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Template Options



Options

Templates:

Archive > ZIP
Audio > WAV
Executable > EXE
Image > BMP
Syntax > 010

Template Options

Name: ZIP ☑ Visible
File Name: ($TEMPLATE_REPOS_DIR)\ZIP.bt
Category: Archive
File Mask: *.zip
ID Bytes: 50 4B //PK ☑ Require
Status: Installed from repository (v2.3)
☑ Run on Load ☐ Show Editor on Load

The Template Options dialog lists all Binary Templates that have been installed, either from the local drive or from the 010 Editor Repository. When a Template is installed it is listed on the main *Templates* menu and can be set to automatically run when certain files are opened. Access the Template Options dialog by clicking the '*Tools > Options...*' menu item and selecting *Templates*, or by clicking '*Templates > View Installed Templates...*'.

The upper part of the dialog contains a list of templates displayed in the format '*Category > Template Name*'. New templates can be added from the local drive by clicking *Add* and selecting a template file (note that multiple templates can be added at the same time using the file dialog that is displayed and information such as the Category, File Mask and ID Bytes will be extracted from the file comments). Remove a template from the list by selecting an item and clicking *Delete* (the file will not be deleted on disk). Click the up or down arrows to arrange the templates in the list.

When a template is selected from the list its attributes are displayed in the *Template Options* box. Enter a name for the template in the *Name* field. This name will appear on the Templates Menu listed by *Category*. If the *Category* is empty the template will be displayed near the top of the *Templates* menu. Disabling the *Visible* toggle allows the template to be hidden from the menu without deleting it. Enter the file name for the template in the *File Name* field and Binary Templates usually have the extension '.bt'. The folder button beside the *File Name* field can be used to select a template using a file dialog box. Press the *Edit* button to close the Options dialog and view the file in the editor.

The *File Mask* and *ID Bytes* fields indicate which data files this template can parse. The *File Mask* field matches the file name of a file and can use the wildcard characters '*' (zero or more matches) or '?' (exactly one match) and can use the comma character to separate multiple masks. For example the File Mask "*.o,?.dylib" would match either "test.o" or "a.dylib". The *ID Bytes* indicates a set of bytes at the beginning of the data file that the file must contain before the Template is loaded. The bytes are listed in hex notation and the '//' characters indicate the start of a comment which is ignored when matching bytes. The special notation [+DDD] or [+0xHHH] can be used where DDD is a decimal number or HHH is a hex number to skip over bytes in the file. For example the ID Bytes '00 [+4] FF' means the data file must have a '00' byte at the beginning of the file and a 'FF' byte at position 5. Currently only the first 2048 bytes in the file are matched against the *ID Bytes*. If *ID Bytes* is

Copyright © 2003-2019 SweetScape Software

empty or the *Require* toggle is unchecked then only the *File Mask* will be used.

If the template has been installed from the 010 Editor Repository then the *Status* field lists the version that was installed. Clicking the *Show* button will hide this dialog and display the Template information in the Repository Dialog. If the *Run on Load* toggle is enabled, this template will be run automatically when it is loaded. When the *Show Editor on Load* toggle is set, the template will be opened for editing in the interface.

The following templates are installed by default:

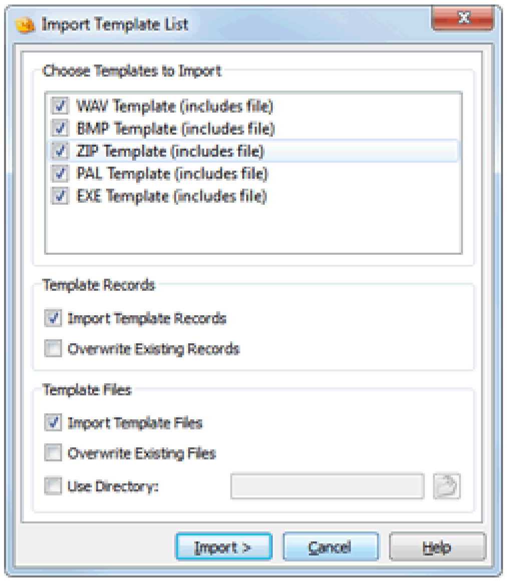
- **ZIP -** Template used to parse ZIP archives. Loads the '*ZIP.bt*' file.
- **WAV -** Template used to parse a WAV sound file. Loads the '*WAV.bt*' file.
- **BMP -** Template used to parse bitmap files. Loads the '*BMP.bt*' file.

Click the *Set Shortcut* button to jump to the Keyboard Options dialog to set a shortcut key for the selected template. The *Reset* button can be used to reset all templates to their original values.

Importing and Exporting Lists of Templates

The Template Options dialog can be used to export the current list of templates so that they may be importing into another copy of 010 Editor. To export the current list of templates click the *Export List...* button at the bottom right corner of the dialog. Choose the location to save the template list using the standard file dialog. Exported template lists contain two things: the list of Template Records (all the information displayed in the *Template Options* group) plus the actual template files. Exported template lists have the extension ".1tl".

To import an existing template list click the *Import List...* button. Choose a template list to import and the *Import Template List* dialog will be displayed.



☑ WAV Template (includes file)
☑ BMP Template (includes file)
☑ ZIP Template (includes file)
☑ PAL Template (includes file)
☑ EXE Template (includes file)

☑ Import Template Records
☐ Overwrite Existing Records

☑ Import Template Files
☐ Overwrite Existing Files
☐ Use Directory:

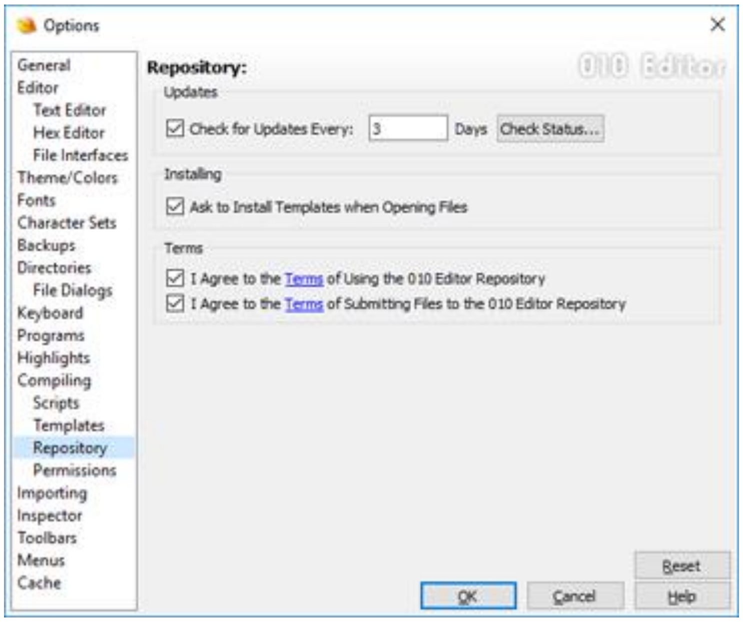
Choose which templates to import from the list at the top of the dialog. If the *Import Template Records* toggle is selected, all the template records are read (a template record includes all information in the *Template Options* area of the main Options dialog). By default any existing template records are not modified but they may be overwritten by enabling the *Overwrite Existing Records* toggle. If the file to import contains the actual template files (the text '*includes files*' should appear beside the template name in the list), then the actual template files can be written to disk. Enable *Import Template Files* to write the files to disk and enable the *Overwrite Existing Files* option to overwrite template files on disk. By default the template will be written to the same directory where it was exported from but if that directory does not exist on this machine, enable the *Use Directory* toggle and choose which directory to place all the template files.

Clicking the *Import* button will perform the import and will display a Results text area with information about the import. If any errors occurred while writing files, the errors will be displayed in the Results. Click the *Close* or *Cancel* button to dismiss the dialog. This dialog works the same way as the *Import Script List* dialog except the dialog operates on templates instead of scripts.

Related Topics:
Introduction to Templates and Scripts
Introduction to the Repository
Templates Menu
Using the Repository Dialog

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Repository Options



Repository:

Updates

☑ Check for Updates Every: 3 Days

Installing

☑ Ask to Install Templates when Opening Files

Terms

☑ I Agree to the Terms of Using the 010 Editor Repository

☑ I Agree to the Terms of Submitting Files to the 010 Editor Repository

Use the Repository Options dialog to control some settings of the 010 Editor Repository. Access the Repository Options dialog by clicking the '*Tools > Options...*' menu item and selecting *Repository* from the list. Many of these options are also available in the *Status* tab of the Repository Dialog.

010 Editor periodically downloads updates from the online Template and Script Repository. Set the number of days between updates using the *Check for Updates Every* field or turn off updates by unchecked the *Check for Updates Every* toggle. See Updating the Repository for more information about when updates are downloaded. Clicking the *Check Status* button hides this dialog and displays the *Status* tab of the Repository Dialog.

When opening a file in 010 Editor and a template is found in the Repository which can parse the file, a dialog asking to install the template is displayed. To disable checking files when they are opened uncheck the *Ask to Install Templates when Opening Files* toggle and see Installing Files on Open from the Repository for more information.

The 010 Editor Repository contains Templates and Scripts that have been uploaded by 3rd parties. To install files from the Repository you must agree to the *Terms of Using the 010 Editor Repository* and to upload files to the Repository you must agree to the *Terms of Submitting Files to the 010 Editor Repository*. These terms can be viewed by clicking the *Terms* labels and check the *I Agree* toggles to signify that you agree to these terms. If you do not agree to these terms you will not be able to download or upload files to the Repository.

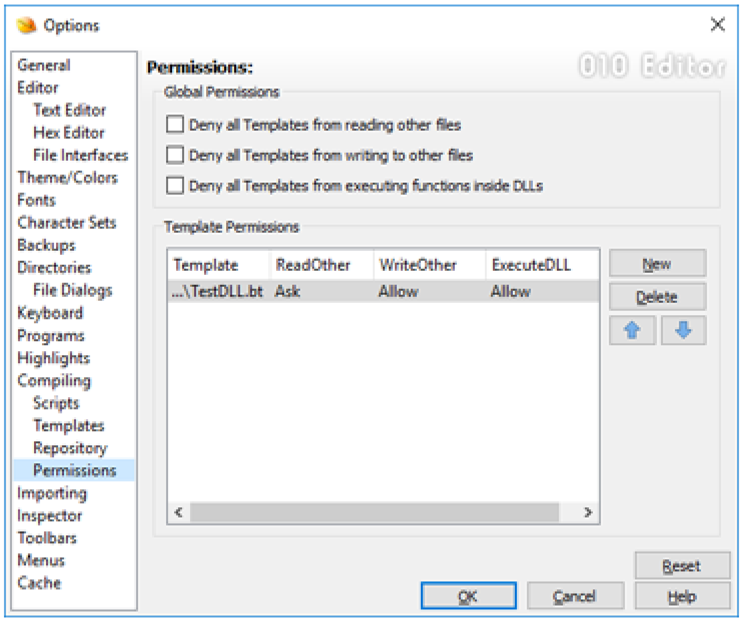
Click the *Reset* button to set all Repository options to their original values.

Related Topics:
Installing Files on Open from the Repository
Introduction to the Repository
Updating the Repository
Using the Repository Dialog

010 Editor v10.0 Manual - Windows Edition

Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Permission Options



Permissions:

Global Permissions

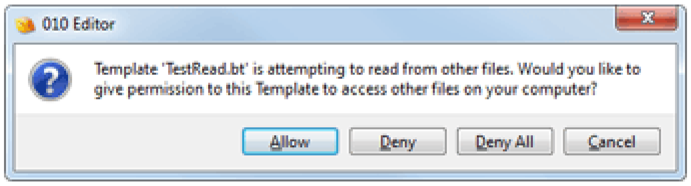
- [ ] Deny all Templates from reading other files
- [ ] Deny all Templates from writing to other files
- [ ] Deny all Templates from executing functions inside DLLs

Template Permissions

Template	ReadOther	WriteOther	ExecuteDLL
...\TestDLL.bt	Ask	Allow	Allow

A Binary Template usually only reads data from one file at a time and cannot modify any files; however, it is sometimes useful to have a Binary Template read from or write to another file (for example, if a dataset is split across multiple files or to write log files). Templates must be granted permission to read from or write to other files and these permissions can be controlled with the Permission Options dialog, as accessed by clicking '*Tools > Options...*' on the menu and selecting *Permissions* from the list. As one exception, a Template is allowed to create a new file with the FileNew function and write to that file without special permissions (using FPrintf for example), but to save the file to disk the Template must have write permissions. As well, Templates must be granted permission to call functions inside external dynamic link libraries as described in the External (DLL) Functions in Scripts help topic.

The *Global Permissions* toggles can be used to deny all Templates from reading other files, writing to other files, or executing functions in external DLLs. If the deny read, write or execute Global Permission is turned off and a Template attempts to perform one of these operations, a dialog similar to below will be displayed to ask the user for permission. Any permissions granted are displayed in the *Template Permissions* table below. Read requests can be generated by using the FileOpen function and write requests can be generated by using any of the write functions or the FileSave function. If the *Global Permissions* are set to deny for read, write or execute, then the read, write or execute columns in the *Template Permissions* table are ignored.



Template 'TestRead.bt' is attempting to read from other files. Would you like to give permission to this Template to access other files on your computer?

When no permissions exist for a Template or the permissions are set to *Ask* and a Template attempts to read from or write to another file or execute a DLL function, a dialog box such as in the figure above will be displayed. Clicking *Allow* or *Deny* will create an entry in the *Template Permissions* table as explained below. If the operation is denied the Template will be stopped. Clicking *Deny All* when asking for permissions will turn on the corresponding *Deny...* toggle in the *Global Permissions* section. Clicking the *Cancel* button will stop Template execution.

The *Template Permissions* table at the bottom of the Permission Options dialog displays any permissions that were allowed or denied for individual Templates. Additional permissions can be set by clicking the *New* button and choosing a Template in the standard file dialog box that is displayed. By default, the permissions are set to *Ask* for *ReadOther*, *WriteOther* and *ExecuteDLL* but can be changed to *Allow* or *Deny* by clicking on a permission and using the drop-down list to select a new value. Click the *Delete* button to remove a permission or the up and down arrows to reorder permissions.

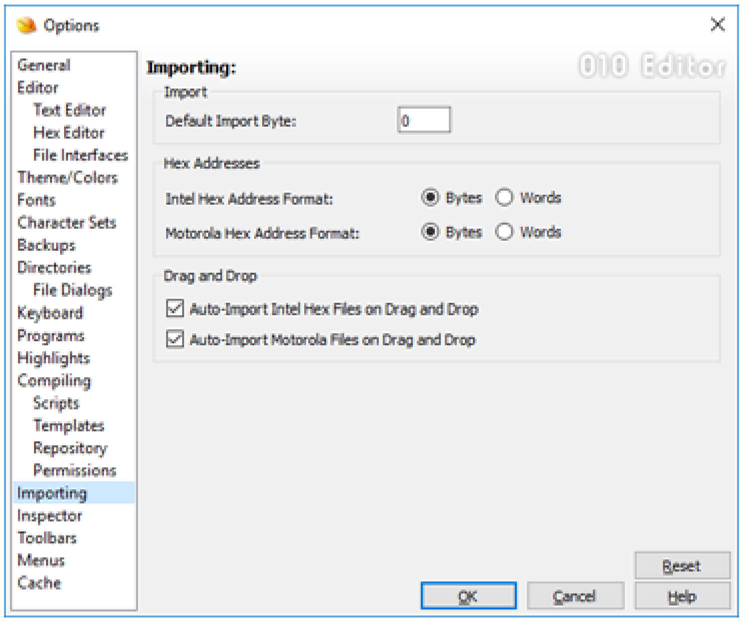
A special warning is displayed when a Template that is installed from the public Repository asks for permission to read, write or execute other files. Typically these Templates should be denied but only grant permissions to read, write or execute if you understand what the Template is doing and why it needs special access.

Clicking the *Reset* button removes all *Template Permissions* and resets the Global Permissions to allow access.

Related Topics:
External (DLL) Functions in Scripts
External (DLL) Functions in Templates
Interface Functions
Introduction to Templates and Scripts
Introduction to the Repository
I/O Functions

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Importing Options


Options

Importing:

Import

Default Import Byte: 0

Hex Addresses

Intel Hex Address Format: [✓ Bytes] Words

Motorola Hex Address Format: [✓ Bytes] Words

Drag and Drop

☑ Auto-Import Intel Hex Files on Drag and Drop

☑ Auto-Import Motorola Files on Drag and Drop

The Importing Options dialog controls various options when importing files using the '*File > Import Hex...*' menu option (see Importing/Exporting Files for more information). Open the Importing Options window by clicking '*Tools > Options...*' and selecting *Importing* from the list.

When importing data, some formats (such as Intel Hex or Motorola S-Records) may skip over some bytes. By default, the skipped bytes are assigned the byte value zero, but a different value can be specified by entering a number between 0 and 256 in the *Default Import Byte* field.

In some Intel Hex or Motorola files, the addresses are given in terms of Words instead of Bytes. 010 Editor handles these files by converting the Word-based addresses to Byte-based addresses by multiplying them by two. Click the *Words* toggle in the *Intel Hex Address Format* area or *Motorola Hex Address Format* area to perform the conversion on Import, or click the *Bytes* toggle to leave the addresses unmodified. Addresses can be converted to Word-based when exporting via the Export Options dialog (see Importing/Exporting Files).

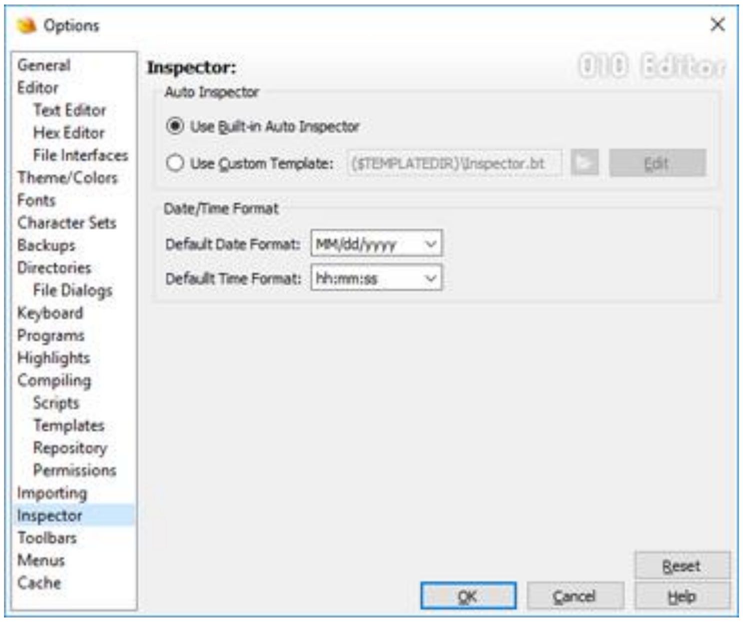
When dragging and dropping Intel Hex or Motorola S-Record files from the Windows Explorer onto 010 Editor, the files are automatically imported. To disable auto-importing and just display the original files, uncheck either the *Auto-Import Intel Hex Files on Drag and Drop* option or the *Auto-Import Motorola Files on Drag and Drop* option.

Clicking the *Reset* button restores all Importing options to their default values.

Related Topics:
Importing/Exporting Files

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Inspector Options



Inspector:

Auto Inspector

(●) Use Built-in Auto Inspector

( ) Use Custom Template: ($TEMPLATEDIR)\Inspector.bt

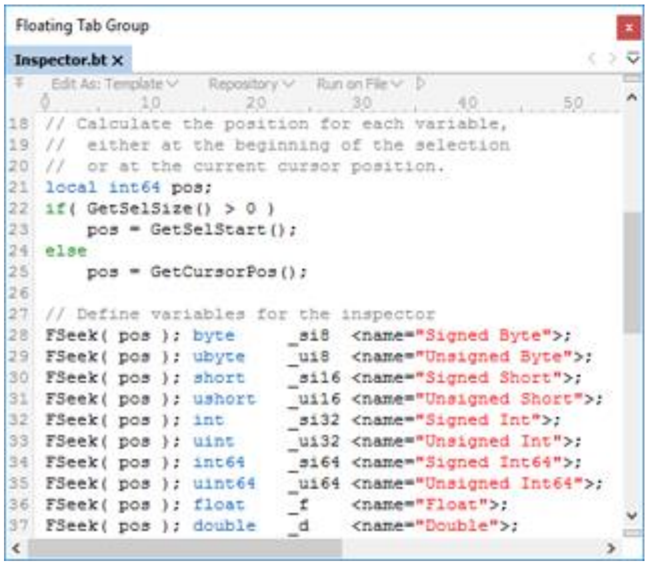
Date/Time Format

Default Date Format: MM/dd/yyyy

Default Time Format: hh:mm:ss

Use the Inspector Options dialog to control how the Inspector operates. View the Inspector Options window using '*Tools > Options...*' and selecting *Inspector* from the list or by right-clicking on the Inspector and choosing *Customize...*.

The Auto tab of the Inspector allows interpreting binary data in a file in a number of different formats. If the *Use Built-in Auto Inspector* toggle is enabled then the normal list of data types will appear in the Inspector; however, to modify which data types are displayed enable the *Use Custom Template* option. The custom Inspector usually uses the file '*Inspector.bt*' which comes included with 010 Editor (if this file does not exist it will be created when clicking *OK* or *Edit*). A different Inspector template can be chosen by clicking the Browse button to the right of the text file and selecting a template. Clicking the *Edit* button will close the Options dialog and open the Template for editing.



```
// Calculate the position for each variable,
//  either at the beginning of the selection
//  or at the current cursor position.
local int64 pos;
if( GetSelSize() > 0 )
    pos = GetSelStart();
else
    pos = GetCursorPos();

// Define variables for the inspector
FSeek( pos ); byte     _si8  <name="Signed Byte">;
FSeek( pos ); ubyte    _ui8  <name="Unsigned Byte">;
FSeek( pos ); short    _si16 <name="Signed Short">;
FSeek( pos ); ushort   _ui16 <name="Unsigned Short">;
FSeek( pos ); int      _si32 <name="Signed Int">;
FSeek( pos ); uint     _ui32 <name="Unsigned Int">;
FSeek( pos ); int64    _si64 <name="Signed Int64">;
FSeek( pos ); uint64   _ui64 <name="Unsigned Int64">;
FSeek( pos ); float    _f    <name="Float">;
FSeek( pos ); double   _d    <name="Double">;
```

When editing the '*Inspector.bt*' file, each variable is defined to start at the current cursor position or at the start of the current selection, if a selection exists. Variables are defined this way by including the statement 'FSeek(pos);' before each variable definition. The different data types of the Inspector may be modifying by reordering or deleting the lines in the '*Inspector.bt*' file. Additionally, other data types including structs or unions may be added using any of the regular techniques for writing templates (see Introduction to Templates and Scripts) but ensure the statement 'FSeek(pos);' exists before each variable. Even custom data types can be added to the Inspector file using Custom Variables. Every time the cursor position is moved in a file, the '*Inspector.bt*' file will be rerun to move the variables to their new position.

Date/Time Format

To control the formats used for date and time variable types in the Inspector use the *Date/Time Format* area. These formats are also used in the Template Results panel. Specify the default date format in the *Default Date Format* field and the default time format in the *Default Time Field*. A few common formats can be chosen by clicking the down arrow to the right of each field. The following characters can be used when writing formats:

- h - hour without leading zero
- hh - hour with leading zero
- m - minute without leading zero
- mm - minute with leading zero
- s - second without leading zero
- ss - second with leading zero
- z - millisecond without leading zero
- zzz - millisecond with leading zero
- AP - either AM or PM
- ap - either am or pm
- d - day without leading zero
- dd - day with leading zero
- ddd - short day (e.g. 'Mon')
- dddd - long day (e.g. 'Monday')
- M - month without leading zero
- MM - month with leading zero
- MMM - short month (e.g. 'Jan')
- MMMM - long month (e.g. 'January')
- yy - 2-digit year

- yyyy - 4-digit year

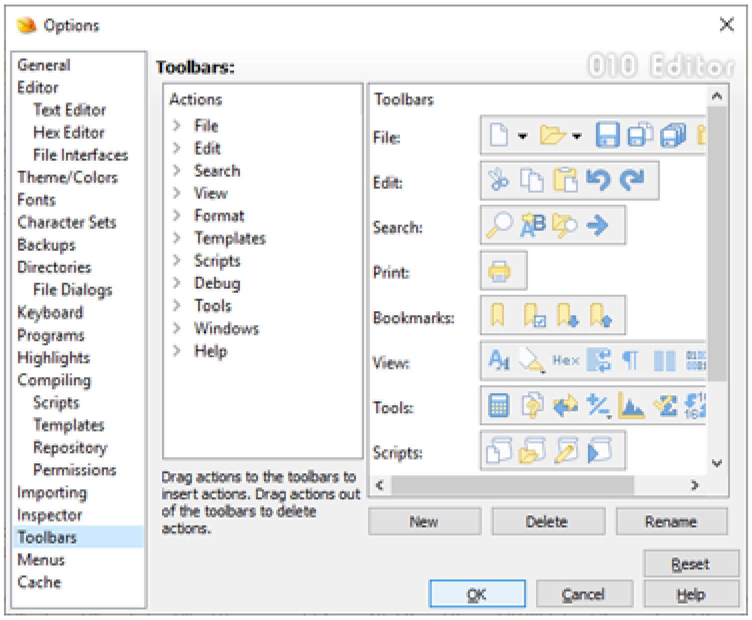
The chosen date and time formats can be accessed in Scripts and Templates using the GetDefaultDateFormat, GetDefaultDateTimeFormat and GetDefaultTimeFormat functions.

Click the *Reset* button to restore the Inspector options to their default values. If the '*Inspector.bt*' file has been modified, when the *Reset* button is clicked you will be asked if you wish to delete all modifications and return to the original '*Inspector.bt*' file.

Related Topics:
Custom Variables
Interface Functions
Introduction to Templates and Scripts
Using the Inspector
Working with Template Results

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

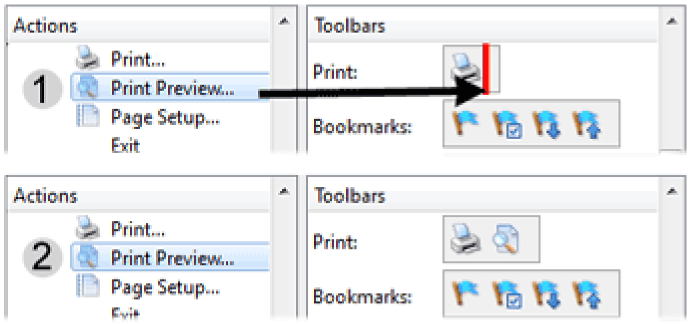
Toolbar Options



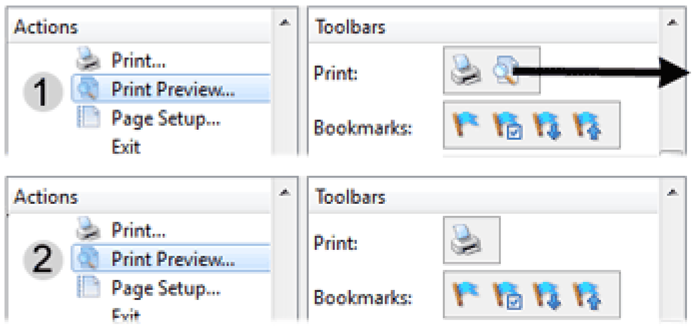
Drag actions to the toolbars to insert actions. Drag actions out of the toolbars to delete actions.

The Toolbar Options dialog is used to to control which operations appear in the application Toolbars (see Using Tool Bars for more information). Open the Toolbar Options dialog by clicking '*Tools > Options...*' and selecting *Toolbars* from the list or by right-clicking on a Toolbar and selecting *Customize...*.

All available actions that can be added to Toolbars are located in the *Actions* list. The actions are sorted into categories and double-click a category name (e.g. *File*, *Edit*, etc.) to view all actions in that category. To add an action to a toolbar click the action in the list and while holding the mouse button down, move the mouse over the *Toolbars* list until a red line indicates the position to insert the action (see 1 below). Releasing the mouse will insert the action at the chosen location (see 2 below). Note that if no icon is associated with an action, a text label will be inserted into the Toolbar. Actions can also be moved within Toolbars by clicking an action in a Toolbar and dragging the action to a new position while the mouse button is pressed.



To remove an action from a Toolbar, click the action in the Toolbar and with the mouse button pressed down, drag the mouse out of the window (see 1 below) until the mouse cursor changes to a circle with a slash or an X. Then release the mouse button to delete the action (see 2 below).



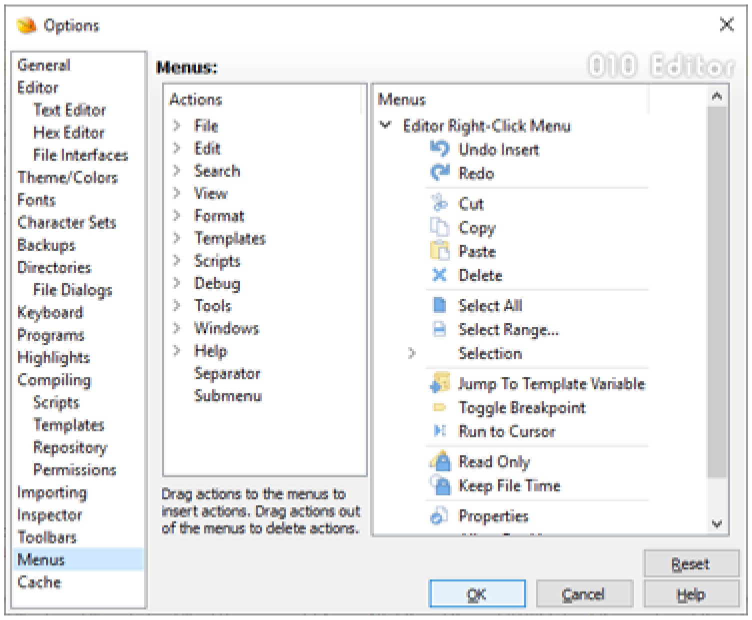
To create a new Toolbar in the list, click the *New* button and enter the name of the Toolbar in the dialog that is displayed. The name of each Toolbar is displayed in the *Toolbars* list and also in the '*View > Tool Bars*' menu. A Toolbar name can be changed by clicking a Toolbar in the *Toolbars* list and then clicking the *Rename* button. Toolbars can also be deleted by selecting a Toolbar and then clicking the *Delete* button. Note that the original list of Toolbars cannot be deleted or renamed, only Toolbars that have been created using the *New* button.

Click the *Reset* button to restore all Toolbars to their original list of actions and delete any custom created Toolbars.

Related Topics:
Using Tool Bars

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

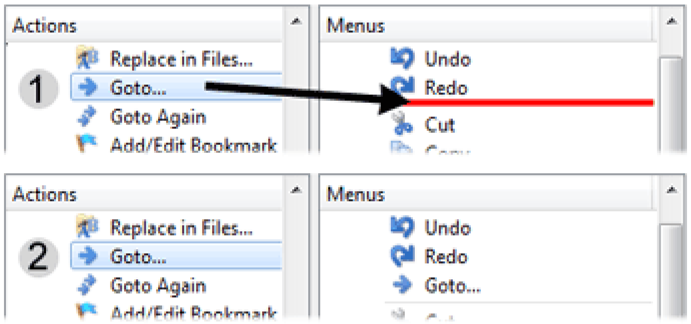
Menu Options



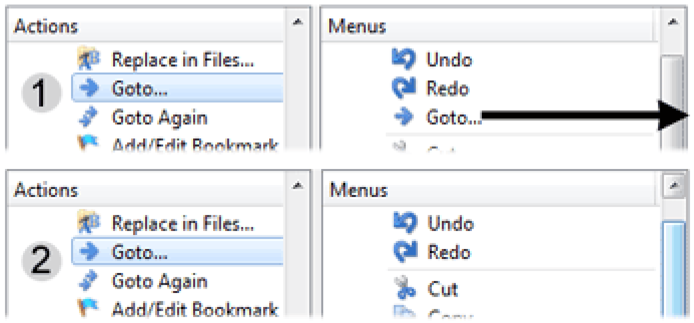
Drag actions to the menus to insert actions. Drag actions out of the menus to delete actions.

Menus in 010 Editor can be customized using the Menu Options dialog. Access the Menu Options by clicking '*Tools > Options...*' and choosing *Menus* from the list or by right-clicking on a Text Editor or Hex Editor window and choosing *Customize...* from the right-click menu.

All actions which can be placed on menus are located in the *Actions* tree. The actions are sorted into categories and double-clicking on a category name will show all actions in that category (e.g. *File*, *Edit* etc). Currently only the *Editor Right-Click Menu* can be customized (see Using the Text Editor or Using the Hex Editor) but other menus will be added in the future. To add actions to the right-click menu first locate an action to add in the *Actions* tree. Next click the action and with the mouse button pressed, drag the action onto the *Editor Right-Click Menu* and a red line will indicate the insertion point (see 1 below). Releasing the mouse button will add the action to the menu (see 2 below).



To delete an action from the menu, click the action name in the *Menus* area and then with the mouse button pressed down, drag the mouse out of the window (see 1 below) until the mouse cursor turns into a circle with a slash or an X. Release the mouse button to delete the action (see 2 below). Note that actions can also be dragged within a menu by clicking and dragging an action name in the *Menus* tree.



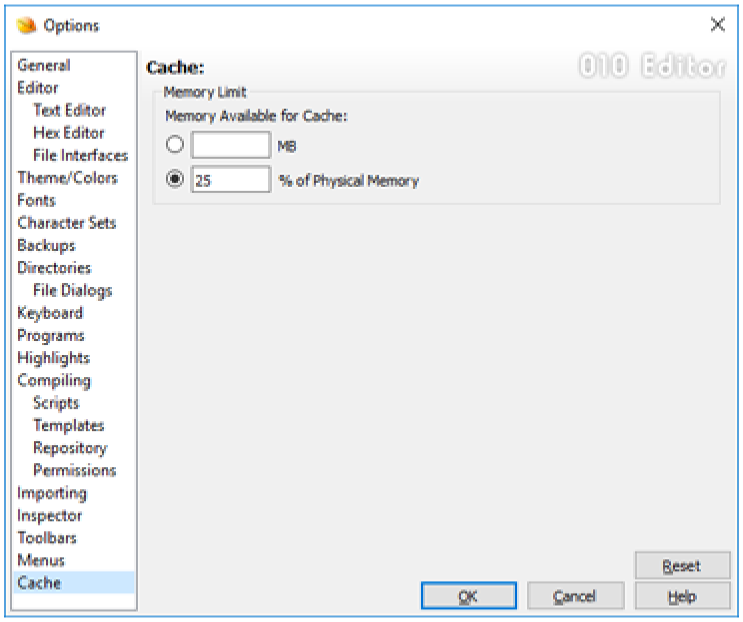
Two special items exist at the bottom of the *Actions* tree: *Separator* and *Submenu*. Dragging and dropping a *Separator* onto a menu will insert a horizontal line into the menu. Dragging and dropping a *Submenu* will add a new menu item that can function as a container for other actions. Submenu items have no action of their own but actions can be dragged into a submenu. When dragging and dropping an action onto a submenu, the regular red insertion line will turn into a red box around the submenu. Submenu items that contain children can be opened by clicking the icon to the left of the submenu name. To change the name of submenu double-click the submenu name, enter a new name, and then press the Enter key. When dragging a submenu item to a new location, all the children of the submenu will be moved as well.

Click the *Reset* button to restore the menu to its original state.

Related Topics:
Using the Hex Editor
Using the Text Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Cache Options



Cache:

Memory Limit

Memory Available for Cache:

[ ] MB

[●] 25 % of Physical Memory

010 Editor features a powerful data engine that enables loading huge files and cutting and pasting large blocks of memory very quickly (see Introduction to the Data Engine for more information). The Cache Options dialog controls various options of the data engine. Open the Cache Options dialog by clicking the '*Tools > Options...*' menu option and selecting '*Cache*' from the list.

When accessing files, data that is loaded into memory is stored in a cache. The size of the cache can be specified using the *Memory Limit* options. Select the *MB* toggle and enter the cache limit in number of megabytes. Alternately, select the *% of Physical Memory* toggle and enter the percentage of physical memory that should be used for the cache (between 1 and 100). Note that physical memory does not include any system virtual memory (using the Windows swap file). Once too much data is loaded into the cache, data will be written to a swap file on disk and see the Directory Options dialog to control where the swap file is written.

Clicking *Reset* will return all cache options to their default values.

Related Topics:
Directory Options
Introduction to the Data Engine

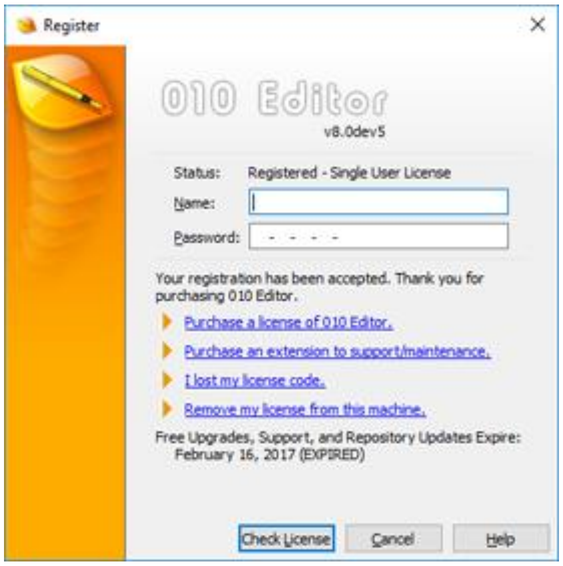
010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

How to Buy 010 Editor

Thank you for considering purchasing 010 Editor. We are committed to providing a top-quality, professional text/hex/disk editor with a full set of powerful editing tools.

After using this program for 30 days, you must purchase a license to continue using the software. Licenses can be purchased using all major credit cards, cheque, PayPal, fax, or phone. To purchase, click '*Tools > Register...*' and click the *Purchase a license of 010 Editor* link in the Register dialog, or click the '*Help > Buy Now...*' menu option. You may also directly visit '*http://www.sweetscape.com/store/*' with any web browser. Once at the website, follow the on screen instructions to complete your purchase. All purchases are secure and backed by a 60-day money-back guarantee.

Register Dialog


Register
010 Editor
v8.0dev5

Status: Registered - Single User License
Name:
Password: - - - -

Your registration has been accepted. Thank you for purchasing 010 Editor.

Purchase a license of 010 Editor.
Purchase an extension to support/maintenance.
I lost my license code.
Remove my license from this machine.

Free Upgrades, Support, and Repository Updates Expire: February 16, 2017 (EXPIRED)

Check License
Cancel
Help

After your purchase you will receive a license password that must be entered in the Register dialog. This dialog automatically displays if your evaluation period has ended, or can be accessed on the menu from '*Tools > Register...*' or from the Welcome dialog. The following is a list of the fields in the Register dialog:

- **Status -** Lists the current registration status of the program. This will say '*Registered*' if the program has been successfully registered or '*Evaluation Period Expired*' if your 30 day trial has ended. If still within the evaluation period, this field will display the number of days left. The status may also display '*Upgrade Required*' if the license you entered is for an earlier version of 010 Editor (see below).
- **Name -** Enter your name exactly as you entered it when purchasing the product. Your password will only work for your name.
- **Password -** Enter the password that was emailed to you when you purchased the product.

Clicking the *Check License* button will test if your entered name and password are valid. If you experience any problems, please visit '*http://www.sweetscape.com/support/*'. A number of links are also available at the bottom

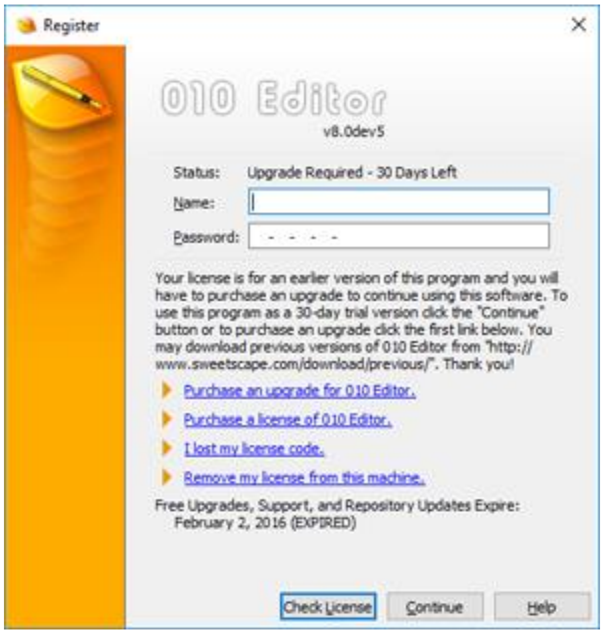
Copyright © 2003-2019 SweetScape Software

of the dialog:

- **Purchase a license of 010 Editor -** Visits the online store where you can purchase a license of 010 Editor.
- **Purchase an extension to support/maintenance -** If your support/maintenance period has expired (see the *Support/Maintenance* information below), clicking this link allows you to purchase an extension to your maintenance so you can continue to have access to support and free upgrades for 010 Editor.
- **I lost my license code -** Retrieves a lost license using our webpage '*http://www.sweetscape.com/010editor/retrieve_password.html*'. You will be required to enter the e-mail address used to purchase 010 Editor.
- **Remove my license from this machine -** Removes the current license from this computer. Warning: If you remove your license you will no longer to be able to use this software until you enter another valid license.

Below the list of links is the *Support/Maintenance* expiration date (this is displayed only after a valid license is entered). After purchasing 010 Editor, you are entitled to free support and maintenance (maintenance includes access to all patches and new versions of the software) and free Repository updates for a certain amount of time after your purchase, usually one year. This section of the dialog lists the date that support and maintenance expires and will display the text *EXPIRED* if the date has already passed. If a new version of 010 Editor is released and your support/maintenance is not expired, you may upgrade to the new version for free. See Updating the Repository for more information about your license and Repository updates.

Upgrading a License



Status: Upgrade Required - 30 Days Left
Name:
Password: - - - -

Your license is for an earlier version of this program and you will have to purchase an upgrade to continue using this software. To use this program as a 30-day trial version click the "Continue" button or to purchase an upgrade click the first link below. You may download previous versions of 010 Editor from "http://www.sweetscape.com/download/previous/". Thank you!

- Purchase an upgrade for 010 Editor.
- Purchase a license of 010 Editor.
- I lost my license code.
- Remove my license from this machine.

Free Upgrades, Support, and Repository Updates Expire:
February 2, 2016 (EXPIRED)

If the license you have entered is for a previous version of 010 Editor, you will be presented with the upgrade dialog as shown above (some variations may exist in the dialog depending upon what type of license is entered). You may use the software free for 30 days but after that time you will need to enter an upgraded license to continue using the software. In addition to the usual links at the bottom of the dialog the following links may be displayed:

- **Purchase an upgrade for 010 Editor -** If you are not eligible for a free upgrade you will need to purchase an upgrade to continue using the software. Click the *Purchase an upgrade for 010 Editor* link to visit our online store or visit the webpage '*http://www.sweetscape.com/store/*'.

If you have any questions please contact us by visiting '*http://www.sweetscape.com/support/*'. Thank you!

Related Topics:
How to Get Support
Updating the Repository

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

How To Get Support

At SweetScape, we are committed to providing quality, bug-free software. If you experience any problems with our software or find any bugs, please don't hesitate to email '*support@sweetscape.com*' with a detailed description of your problem. Be sure to include which operating system you are running and the current version of 010 Editor (found in the '*Help > About*' dialog box). Alternately, an email can be sent by clicking the '*Help > Support by E-mail...*' menu option.

Support can also be obtained by visiting our website at '*http://www.sweetscape.com/support/*'. This page contains a list of frequently asked questions and articles stored in a searchable knowledgebase that might solve your problem. The support website can be accessed by clicking the '*Help > Support on the Web...*' menu option.

If you have any other inquires, comments, ideas, or suggestions on things you would like 010 Editor to do, you can email 'feedback@sweetscape.com'. We want to hear from you!

Related Topics:
How to Buy 010 Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

License Agreement

The following is the end-user license agreement (EULA) to which you agreed while installing 010 Editor:

SWEETSCAPE SOFTWARE
SOFTWARE LICENSE AND LIMITED WARRANTY AGREEMENT

Notice: You should carefully read this Software License and Limited Warranty Agreement before continuing to install this program. By installing, downloading, copying, or otherwise using this software package ("Software"), you agree to all the terms and conditions of the Software License and Limited Warranty Agreement. You must indicate your agreement to be bound by the terms of this agreement by clicking the 'I accept the agreement' button on the Software License Agreement page during the installation process. If you do not agree with the terms of this license, do not continue this installation or otherwise use the Software. You may delete the downloaded Software or return it within 60 days, with any accompanying documentation or other components and a copy of your invoice to the place of purchase. You will be reimbursed or credited in accordance with the current policy of SweetScape Software or the representative from whom you acquired the Software.

COPYRIGHT: This software product is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. All title and copyrights in and to the software product (including but not limited to any programs, images, text, or documentation incorporated into the Software) are owned by SweetScape Software. Purchase of this license does not transfer any right, title or interest in the software to you except as specifically set forth in this agreement. Except as permitted by such license, no part of the accompanying documentation may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of SweetScape Software.

GRANT OF LICENSE: Subject to the condition that you have paid the license fee and are in compliance with the terms of this agreement, SweetScape Software grants you and only you a non-exclusive license to install and use this Software simultaneously on multiple computers running any of the supported operating systems. No other use, copying, or distribution of the Software product is allowed. If more than one user wishes to operate this Software, a separate License and fee is required for each user. You may not rent the Software nor may you offer use of it to others through any service provider. If you have purchased a Home/Academic License you may not, either directly or indirectly, use this Software for commercial purposes (commercial purposes may include but are not limited to any money-making activity) unless you are a member of a registered non-profit organization. Also, a Home/Academic license may not be used at any government or military organization. If you have purchased a Commercial License, this Software may be used for any commercial purposes. Alternately, a Site License may be purchased that allows any employee of the licensed business to use the Software on any machine. If you are installing this Software package as an upgrade or enhancement of a previous release of the same Software that was installed on the same computer, your rights under the prior license agreement are terminated and you are now bound solely under the terms of this license agreement.

LIMITATIONS: You may not reverse engineer, disassemble, or decompile this Software product. This Software is licensed as a single product and its components may not be separated for use on more than one computer. You may not modify, amend, adapt, translate, or create any derivative works based on, the Software or the provided documentation.

TERM: If the Software was distributed to you as an EVALUATION VERSION, the license granted under this agreement commences upon the installation of the Software and is effective for 30 days following the date you install the Software. Evaluation Version Software may include program code to disable their functionality after the expiration of the evaluation term. You may take no actions to circumvent the operation of such code, and you accept all risks that might arise from such disabling code. If you do attempt to disable this code, through backwards engineering or changing the system date, SweetScape Software reserves the right to immediately end your evaluation period. If the Software was not distributed as an Evaluation Version, or you converted an Evaluation Version to the Full-License Version by registering the Software, this agreement is effective until terminated by the terms of this agreement.

TERMINATION: Upon the expiration of the Evaluation Period (if any), your rights under this agreement will terminate automatically without notice from SweetScape Software. This agreement may also be terminated (a) by you, by returning the original Software and accompanying documentation, or (b) by SweetScape Software upon your breach of any of the provisions of this agreement. Upon termination of your rights under this agreement for any reason, you must destroy all copies of the Software and all components (including documentation) in your possession.

TRANSFER: You may permanently transfer all of your rights under this agreement to another individual, provided you retain no copies, you transfer all copies of the Software (including all component parts, and documentation), and the recipient agrees to be subject to the terms of this agreement. Upon the occurrence of such a transfer, your rights under this agreement terminate immediately.

LIMITED WARRANTY: SweetScape Software warrants to you (and only you) that the Software will perform substantially in accordance with the accompanying documentation (if any) for a period of (90) days from the date of original purchase from an authorized retailer or directly from SweetScape Software (or the date you obtained authorization from SweetScape to convert an Evaluation version product to a Full-License product). Implied warranties on the Software, to the extent required by applicable law, are limited to ninety (90) days from the purchase date. The Limited Warranty is void if failure of the Software has resulted from accident, abuse, misapplication, use of the Software other than described in the documentation, use of the Software in combination with other products that are not described as compatible in the documentation, or your breach of the terms of this agreement. No individual and no reseller or retailer has any authority to amend or add to any of the above representations and disclaimers.

CUSTOMER REMEDY: Your exclusive remedy for any breach of the Limited Warranty is for you to send notice of the breach by returning to SweetScape Software a copy of your purchase receipt for your copy of the Software and a description of the alleged breach. Then, at SweetScape's option, SweetScape shall either: (a) terminate the license agreement and refund the license fee upon return of the Software and accompanying documentation or (b) repair or replacement of the Software that does not meet SweetScape's Limited warranty and which is returned. The Limited Warranty period for any replacement product will be extended for the remainder of the original warranty period or thirty (30) days after the replacement is delivered to you, whichever is longer. If your license is for an Evaluation version, your exclusive remedy for any breach of this agreement, including a breach of the Limited Warranty, shall be to terminate your rights under this agreement.

THIRD PARTY WORKS: This software includes the Onigurama Regular Expression Library, Copyright (c) 2013, K.Kosako. ("Library") THIS LIBRARY IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

LIMITATION OF LIABILITY: TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL SWEETSCAPE SOFTWARE OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, DAMAGE TO EQUIPMENT, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF SWEETSCAPE SOFTWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS.

IN ANY CASE, SWEETSCAPE SOFTWARE'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID FOR THE SOFTWARE.

NO OTHER WARRANTIES: TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, SWEETSCAPE SOFTWARE DISCLAIMS ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT AND/OR ACCURACY OF INFORMATION, WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES.

This agreement is governed by the laws of the province of Prince Edward Island, Canada.

Related Topics:
How to Buy 010 Editor

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com

Release Notes

Version 10.0 - December 6th, 2019

The following is an overview of the new functionality in version 10.0 of 010 Editor:

- A full debugger is now available for finding and fixing problems with Templates and Scripts.
- The debugger can be accessed using the *Debug* menu and includes stepping, breakpoints, watches and a call stack.
- Templates and Scripts are now threaded, meaning other editing operations can be done while a Template or Script is running.
- When using the Text Editor, line numbers and ruler labels are now hidden by default (they can be shown with '*View > Addresses > Show Addresses*' or '*View > Ruler > Show Labels*').
- When line numbers or ruler labels are hidden, hover the mouse over the address column or ruler for a second to display a hint popup with the hidden information.
- '*View > Tabs/Whitespace > Show Whitespace*' now can be used to visualize linefeed types for each line.

The following is a list of all new features in version 10.0 of 010 Editor:

- **Debugger**
  - A full debugger is now included for finding and fixing problems with 010 Editor Templates and Scripts.
  - Added a new *Debug* menu for controlling the debugger.
  - Debugging can be turned on or off using the '*Debug > Debugging Enabled*' menu option.
- **Debugger Program Flow**
  - Scripts or Templates can be run the usual way (for example with '*Scripts > Run Script*' or '*Templates > Run Template*') or by selecting a Script or Template and clicking '*Debug > Start Debugging*'.
  - If debugging is enabled and a breakpoint is hit in the Script or Template, program execution will pause (see the next section for information on breakpoints).
  - When paused a yellow arrow will indicate the current debug active line in the Text Editor.
  - Use '*Debug > Step Over*' to step to the next line of the file, jumping over any functions or structs that are called.
  - Use '*Debug > Step Into*' to step to the next line of the file and step into any functions or structs that are called.
  - Use '*Debug > Step Out*' to execute the rest of the current function or struct and stop at the first statement outside the function or struct.
  - To continue running a paused Script or Template click '*Debug > Continue*', '*Scripts > Continue Script or Template*' or '*Templates > Continue Script or Template*'.
  - To pause a running Script or Template click '*Debug > Pause*'.
  - To stop a running or paused Script or Template click '*Debug > Stop*' or press Shift+Esc (note this has changed from the Esc key in previous versions).
  - Scripts or Templates are now run threaded meaning other editing operations can take place when a Script or Template is running.
  - If stepping to a line in an include file, the include file is automatically opened in the editor.
  - Right-click on a Script or Template and choose *Run to Cursor* from the right-click menu. The Script or Template will run (or continue) and execution will stop at the chosen line or at the first breakpoint encountered.
  - When a Script or Template is stopped, click '*Debug > Step Into*' to start the program and stop at the first executable line.
  - When stepping through a Template and the last line of the Template Results or Variables tab is selected, if any new variables are appended to the table then the selection will be moved to the last created variable.
- **Breakpoints**

  - A breakpoint marks a line to stop in the Script or Template and is marked by a red arrow in the left-hand column of the Text Editor.
  - Set or clear a breakpoint for the current line using '*Debug > Toggle Breakpoint*' or by left-clicking the left-hand column in the Text Editor.
  - If a breakpoint is set on a non-executable line then the breakpoint will be moved to the next line that is executable when the Script or Template executes.
  - Breakpoints are persistent (saved to disk) but this can be changed using the Compiling page of the Options dialog.
  - If debugging is disabled then no breakpoints will be hit and the breakpoints are displayed as red outlines in the Text Editor.
  - If the Script or Template is modified when program execution is paused then breakpoints will be disabled. The breakpoints will be displayed as outlines with a solid arrow head.
  - A list of all breakpoints can be found in the Breakpoints tab, which is found in the Inspector tab group or by clicking '*Debug > View Breakpoints*'.
  - In the Breakpoints tab, right-click on the table and select *Add Breakpoint* to set a breakpoint by line number.
  - All breakpoints in all files can be deleted by clicking '*Debug > Delete All Breakpoints*'.
  - The color of breakpoints or the active line marker can be controlled using the Theme/Colors page of the Options dialog.
  - Note that breakpoints are not hit when the application is starting up and any files are being reloaded.
- **Variable Hints**
  - When program execution is paused and the mouse is placed over a variable name in the Script or Template, a hint popup will display the value of the variable.
  - When a selection is made in the Script or Template and the mouse is placed over the selection, the selection will be evaluated and the results displayed in a hint popup.
  - Currently only simple functions (sizeof, startof, exists, etc) can be evaluated in a selection and open the Quick Watch dialog to evaluate a selection which contains more complex functions.
  - Note that how variables are scoped can be affected by the Call Stack tab.
  - Variable hints can be turned off using the Compiling page of the Options dialog.
- **Watches**
  - Watches can be set in the *Watch* tab found in the Inspector tab group or by clicking '*Debug > View Watches*'.
  - Add a watch by double-clicking on the first empty line in the Name column or by right-clicking on the Watch tab and choosing *Add Watch*.
  - A watch can be almost any expression or variable name (for example, 'FileSize()-1000' or 'blocks[i].data[10]').
  - Watches are evaluated every time program execution is paused (e.g. a breakpoint is hit) or when the program is stepped to the next line.
  - If the result of a watch is a struct, the struct can be opened and explored similar to the Template Results.
  - To delete a watch use the Delete key or right-click on a watch and choose *Remove Watch*.
  - A single list of watches is kept for the entire application.
  - Note that how variables are scoped can be affected by the Call Stack tab.
- **Quick Watch**
  - Expressions can also be evaluated without creating a watch using the Quick Watch dialog ('*Debug > Quick Watch*').
  - Enter an expression in the *Expression* field and click *Evaluate*.
  - The result of the expression or variable is displayed in the *Value* column.
  - A list of recent expressions is available by clicking the Down arrow in the dialog.
  - Click the *Add Watch* button to add the current expression to the *Watch* tab.
  - If a selection is made in the Text Editor before the Quick Watch dialog is opened, the selection is copied to the Expression field and evaluated.
- **Debugging Runtime Errors**
  - If a runtime error occurs in a Template or Script a popup dialog box will be displayed asking to start the debugger.
  - When debugging errors the cursor is placed on the line that caused the error.
  - Variables can be investigated with Variable Hints in the Text Editor or with watches.
  - Clicking *Continue* or stepping to the next line will stop the Script or Template.
  - Select the *Always use this action* toggle in the popup dialog box to always start the debugger or never start the debugger.
  - Whether the debugger starts on an error can also be controlled with the Compiling page of the Options dialog.

- **Call Stack**
  - The Call Stack is available in the *Call Stack* tab which can be found in the Inspector tab group or by clicking '*Debug > View Call Stack*'.
  - When program execution is paused, the Call Stack lists the functions or structs that were called to reach the current execution point.
  - The current function or struct is listed at the top of the call stack and the function or struct which called that function or struct is listed below it.
  - If execution is not inside a function or struct then *(Main Program)* is listed in the call stack.
  - Double-clicking on a function or struct jumps to the last position inside that function or struct.
  - Double-clicking on a function or struct also makes any local variables inside the function or struct in local scope (this affects any watches or Variable hints in the Text Editor).
- **Debugger Limitations**
  - Currently breakpoints are not hit inside custom read/write/name/comment functions that are called from the Template Results or Variables tab. To debug these functions call them directly inside the Template.
  - Currently breakpoints in on-demand structures are not hit when the structure is created by opening it in the Template Results. To debug these functions trigger creation of the struct directly in the Template by accessing a variable inside the struct.
  - Currently breakpoints are not hit inside the HighlightLineRealtime or HighlightBytesRealtime functions. To debug these functions see the *Using the Debugger* help topic in the manual for sample code to call.
- **Templates and Scripts**
  - A full debugger including breakpoints, watches and call stack is now available for Templates and Scripts.
  - Templates and Scripts are now threaded, meaning other editing operations can be done while a Template or Script is running.
  - When a Template is running click '*Templates > Stop Template*' or press Shift+Esc to cancel the Template.
  - When a Script is running click '*Scripts > Stop Script*' or press Shift+Esc to cancel the Script.
  - On-demand Structures which have arguments are now supported.
  - Custom read functions can now be called on structs with zero size.
  - Custom name/comment functions now work for local variables.
  - After selecting a Script or a Template that has been run, the Variables tab now shows the list of variables created by that Script or Template.
  - If an included file is opened and modified in the editor, the modified version is used when compiling instead of the disk version.
  - Which warnings are displayed in the Output panel can be configured using the *Compiling* page of the Options dialog.
  - When the application is starting up and files are being reloaded, the Output panel shows the results from all Templates that were run.
  - The Template Results panel only shows the results from a syntax highlighting template if the Template was run directly (not as the result of opening a file).
  - Can right-click on the Variables tab and select *Clear* to clear the results from a Script or Template.
  - The InputString function now returns a UTF-8 string.
  - Jump to Template Variable is now only shown on the Editor right-click menu when editing a hex file.
- **Editor**
  - In the Text Editor, line numbers are now hidden by default and can be displayed by clicking '*View > Addresses > Show Addresses*'.
  - When addresses are hidden, place the mouse cursor over the address column for a second to see the line number in a hint popup.
  - When addresses are hidden, a triangle marker indicates the last line in a file (this can be turned off by setting the *Address End Marker* to None in the Theme/Colors Options dialog).
  - When addresses are hidden, a '-' marker indicates lines that are created by word-wrap.
  - In the Text Editor, ruler labels are now hidden by default and can be shown using '*View > Ruler > Show Labels*'.
  - When ruler labels are hidden, place the mouse cursor over the ruler for a second to view the mouse and cursor position in a hint popup.
  - In the Hex Editor, small arrows in the ruler show the current cursor position and can be turned off using '*View > Ruler > Show Arrows*'.
  - '*View > Tabs/Whitespace > Show Whitespace*' now can be used to visualize linefeed types for each line.

    - The different symbols drawn for *Show Whitespace* can be configured using the *Text Editor* page of the Options dialog.
    - Breakpoints can be toggled by clicking the left-most column when editing a Script or Template.
    - When right-clicking on the editor, the cursor is now moved before the right-click menu is shown.
- **General**
    - The shortcut for opening the Base Converter was changed to Ctrl+F11.
    - Updated the visual style of the Windows installer.
    - Using *Import Hex* with Hex Text or *Paste from Hex Text* now supports data with more types of formatting.
- **Options**
    - On the Text Editor page added the *Show Whitespace* section to control how linefeeds are drawn.
    - On the Text Editor page added the *Change Whitespace Symbols* button to control which symbols are drawn for the different types of whitespace.
    - On the Theme/Colors page added an option to control colors of breakpoints and the debug active line.
    - On the Theme/Colors page added an option to control colors of the Address Hover Marker and Address End Marker (a triangle marker displayed on the last line when Show Addresses is turned off).
    - On the Compiling page added the *Configure* button to control which warnings are displayed in the Output panel.
    - On the Compiling page added the *Breakpoints are Persistent* toggle to control whether breakpoints are automatically saved to disk.
    - On the Compiling page added the *Show Variable Hints when Debugging* option to display the value of variables when the mouse is placed over a variable name in the Text Editor.
    - On the Compiling page added the *When errors occur* drop-down menu to control what action is taken when an error occurs in a Script or Template.
    - On the Inspector page added the default date format 'dd/MM/yyyy'.
- **Bugs**
    - Fixed scripts were not given permission to execute functions in DLLs in some cases.
    - Fixed incorrect error message 'Incorrect function' when trying to load a file that does not exist on some machines.
    - Fixed a crash replacing certain empty regular expressions with nothing.
    - Fixed a possible crash editing a text file which contains a very long line.
    - Fixed incorrect size of tabs in the Preferences dialog of the Help application.
    - Fixed Save All does not try to save text in the Calculator to a file.
    - Fixed the Inspector would sometimes not update properly after clicking a Floating Tab Group file and then a Template Results panel in the main window.
    - Fixed possible crash with the Memset function.
    - Fixed possible crash with ReadWString/ReadString functions and very large files.
    - Fixed '*Format > Comment Selection*' now works with Python commenting.
    - Fixed permission issue with the FileSaveRange function.
    - Fixed up some inaccurate error messages when using invalid name/comment functions.
    - Fixed a text color issue with the Output pane after calling the OutputPaneClear function.
    - Fixed when replacing with nothing, sometimes not all replacements were listed in the Replace Results when 2 or more occurrences were found together.
    - Fixed an empty struct could be executed twice in some cases.

Version 9.0.2 - April 26th, 2019

- Fixed icon transparency for the Windows Explorer shell extension.
- Fixed syntax highlighting for keywords (HighlightMatchKeyword) now finds the longest matching keyword instead of the shortest (e.g. 'background-repeat' vs 'background').
- Fixed when 010 Editor is run as administrator on Windows, files could no longer be opened from the Windows Explorer shell extension. Now a new non-admin copy of 010 Editor is opened before the files are opened.
- Fixed on Linux the IBus IME can now be used.

- Fixed possible crash in Memcpy, StrCat and SubStr when using very large arrays.
- Fixed possible memory overrun when downloading news.
- Fixed when opening a text file from the command line containing a BOM, sometimes bytes were improperly selected at the beginning of the file.

Version 9.0.1 - December 5th, 2018

- Fixed a number of cases when a Syntax Highlighter was not properly applied when a file was opened.
- Fixed an error running a Syntax Highlighter when the first line of a file was blank.
- Fixed using 'return' inside a struct which was declared in a function would cause an error in the function.
- Fixed bookmarks created while the Bookmarks panel was hidden were not properly shown when the Boomkarks panel was displayed.
- Fixed on Linux when using a dark theme, the category headings in the Repository dialog could not be read.
- Fixed the 'Hide Characters over 127' toggle should always be turned off when a custom character set is created.
- Fixed passing a very long line to a Syntax Highlighter could cause the software to slow down. Syntax Highlighters now only color up to the Maximum Line Length as set in the Text Editor Options.
- Fixed a problem running a template on a template, or a script on a script using the File Bar (right-click on the File Bar to enable this).
- Fixed the Sleep function on Linux/macOS was using microseconds instead of milliseconds.
- Fixed the Variables tab was not always showing the results after running a script.
- Added a warning when a very long line was encountered (over 100,000 bytes) and word wrap was turned on. These lines cannot currently be fully wrapped and this will be improved in the future.

Version 9.0 - October 10th, 2018

The following is an overview of the new functionality in version 9.0 of 010 Editor:

- Syntax Highlighters can now be shared via our online template repository.
- Syntax Highlighters are implemented as a function inside a Binary Template .bt file.
- Syntax Highlighters can be run by clicking the *Syntax:* section in the File Bar just above each text editor or by clicking '*Templates > Syntax*' on the main menu.
- Added a number of new character sets and simple character sets can be customized, imported, or exported.
- Scripts and Templates can now call functions in an external library (*.dll on Windows, *.so on Linux, *.dylib on macOS) using #link.
- Now the last cursor position and scroll position for files are restored when 010 Editor is restarted.
- Added Delete Line (Ctrl+Shift+Backspace) and Delete Blank Lines commands to the Format menu.
- Triple-click to select by line and drag to select multiple lines.

The following is a list of all new features in version 9.0 of 010 Editor:

- **Syntax Highlighting**
  - Syntax Highlighters can now be shared via our online template .
  - Syntax Highlighters are written in a different format than before and are implemented as a function inside a Binary Template bt file.
  - New Binary Templates XML.bt, CPP.bt, PHP.bt and HTML.bt are automatically installed to perform syntax highlighting for those formats and other Binary Templates will be added to the online repository soon.
  - Highlighting for 010 Editor Templates and Scripts is now performed with the 010.bt Binary Template.
  - Removed the '*View > Highlighting > Syntax Highlighting*' menu.

    - Syntax Highlighters are now run by clicking the *Syntax:* section in the File Bar just above each text editor or by clicking '*Templates > Syntax*' on the main menu.
    - Clear Syntax Highlighters from a file by clicking on the *Syntax:* section in the File Bar above a text editor and select '*(none)*'.
    - If a Syntax Highlighter is found in the repository to highlight the current file, a dialog will popup asking to install or ignore the file, similar to installing regular Binary Templates.
    - The *Syntax* page has been removed from the Options dialog.
    - When 010 Editor v9 is first run, any old custom Syntax Highlighters created with the *Syntax* page of the Options dialog are exported to XML files in the 'Documents\SweetScape\Old Syntax Highlighters' directory.
    - Old Syntax Highlighters must be converted manually to the new format and there is not yet an automated tool to do the conversion.
    - The new Syntax Highlighting method can handle a huge range of other text formats but requires some programming to implement.
    - SweetScape Software will be available to help with conversion of Syntax Highlighters for common text formats time permitting.
    - If Old Syntax Highlighters were exported, they can be exported again later by clicking the *Export Old Syntax Highlighter* button on the *Highlights* page of the Options dialog. If this button is not displayed then there are no Syntax Highlighters to export.
    - Syntax Highlighters are written by implementing the function HighlightLineRealtime inside a Binary Template. This function applies colors to a single line of text.
    - Highlighting can also be applied to binary files instead of text files by implementing the function HighlightBytesRealtime in a Binary Template.
    - Added new functions to help in the creation of Syntax Highlighters: HighlightFindStyle, HighlightGetStyleForeColor, HighlightGetStyleBackColor, HighlightAllowInstanceSharing, HighlightApplyStyle, HighlightApplyColor, HighlightGetNextToken, HighlightBuildKeywordList, HighlightMatchKeyword, HighlightMatchString, HighlightFindKeyword, HighlightFindString, HighlightCheckMultiLineRule, HighlightCheckCommentRule, HighlightCheckSingleLineRule, HighlightCheckKeywordRule, HighlightCheckTagRule, HighlightCheckTagTokenRule, HighlightColorPattern.
    - Some highlighting functions can do case insensitive matching by using the HIGHLIGHT_IGNORECASE constant.
    - Some highlighting functions can do regular expression matching by using the HIGHLIGHT_REGEX constant.
    - The HighlightFindStyle function has an option to create custom Syntax Styles if they do not yet exist.
    - Custom Syntax Styles now store separate colors for light and dark themes.
    - If instance sharing is turned on using the HighlightAllowInstanceSharing function then only one copy of the Syntax Highlighter is kept in memory and used for all open files of the target text format.
    - By default the Template Results panel is not shown when running a Binary Template that does syntax highlighting.
- **File Interfaces**
    - Syntax Highlighting is no longer controlled via the current File Interface (*View > Edit As*).
    - Combined the old file interfaces *XML* and *HTML* into a single *Tagged* interface.
    - Combined the old file interfaces *C/C++* and *PHP* into a single *Code* interface.
    - '*File > New*' menu now controlled by clicking *Manage New File Types...* on the *Editor* page of the Options dialog.
    - Possible to show or hide different sections (e.g. Run Script, Run Template, Repository...) of the File Bar above each text or hex editor by right-clicking on the File Bar.
- **Character Sets**
    - Added support for more character sets as well as custom character sets.
    - The '*View > Character Set*' menu has been reorganized to show a list of common character sets, followed by a list of recently used characters, followed by the character sets sorted into Standard, International, and Custom categories.
    - Character sets can be viewed, customized, imported and exported using the new *Character Sets* page of the Options dialog.
    - Now per-file character sets can be set by turning off the '*View > Character Set > Use Default' toggle*. If the *Use Default* toggle is on then the character set comes from the current File Interface.
    - When opening a file and a different character set is detected than the current File Interface has, the *Use Default* toggle will be turned off automatically (an example of this is opening a Unicode XML file).
    - If a different character set is chosen when the *Use Default* toggle is off, that character set will

be remembered when the file is closed and opened again.
- Added the following character sets:
    - Arabic (ISO) - ISO 8859-6
    - Baltic (ISO) - ISO 8859-13
    - Cyrillic (KOI8-R) - KOI8-R
    - Cyrillic (KOI8-U) - KOI8-U
    - Cyrillic (ISO) - ISO 8859-5
    - Eastern Europe (ISO) - ISO 8859-2
    - Greek (ISO) - ISO 8859-7
    - Hebrew (ISO) - ISO 8859-8
    - Japanese (EUC-JP) - EUC-JP
    - Japanese (ISO-2022-JP) - ISO 2022-JP
    - Turkish (ISO) - ISO 8859-9
- Existing character sets Arabic, Baltic, Cyrillic, Eastern Europe, Greek, and Hebrew now have "(Windows)" appended to their name to show they are using the Windows code pages.
- Japanese character set is now named 'Japanese (Shift_JIS)'.
- Now just a single Korean character set EUC-KR is supported.
- The *Character Sets* page of the Options dialog allows viewing character sets and creating custom character sets.
- Character sets are displayed in a 16x16 table for easy visualization.
- Information about different symbols is displayed in a tool tip when moving the mouse cursor over the character set table.
- When viewing a complex (multi-byte) character set a horizontal scroll bar will appear to allow scrolling through the different pages.
- The font of the character set table can be controlled by right-clicking on the table.
- Simple character sets can be modified by clicking on a symbol in the character set table and selecting a different symbol using the Change Symbol dialog that pops up.
- If attempting to modify a built-in characters set, the character set will be copied to a new custom character set before the modification is made.
- Simple character sets can be exporting to a CSV file by clicking on the *Export...* button.
- A simple character set can be imported by clicking the *Import...* button. The import format should be 256 numbers separated by commas, spaces, or tabs and the number -1 designates no change.
- In the Change Symbol dialog click the *Reset* button to reset the character to the original value or enter a new Unicode code point U+XXXX in the edit field at the top right corner.
- In the *Character Sets* page of the Options dialog enable the *Show at Top Level* toggle to display the character set at the top of the '*View > Character Sets*' menu.
- For custom character sets, the *Encoding* can be chosen to select which internal encoding this character set uses to convert bytes to characters.
- Enable the *Hide Characters over 127* toggle to hide the display of any character with Unicode code points over 127 (for example, this is used for the ASCII character set).
- The text displayed in the status bar when a character set is active can be controlled using the *Status Bar Indicator* edit box.
- An integer *ID Number* can be specified for custom character sets for use in scripting functions such as ConvertString.
- New constants are available for the ConvertString function for the newly available build-in character sets.
- Created character sets are available for conversion when using the '*Tools > Convert...*' tool.
- The Convert tool shows the character sets marked as *Show at Top Level* and recently used character sets at the top of the *Target Character Set* list.
- After the Convert tool is run, the file may be assigned a per-file character set (i.e. the '*View > Character Set > Use Default*' toggle is turned off and a different character set is assigned).

- **External Functions in DLLs**
    - Templates and Scripts can now call functions in an external dynamic library.
    - Works with Windows DLLs (*.dll), Linux shared objects (*.so), or macOS DYLIB files (*.dylib).
    - All functions defined inside of a #link "<filename>" and #endlink directive are assumed to be located in an external library (note that no body can be defined for these functions).
    - The 32-bit version of 010 Editor should be used when linking to 32-bit external libraries and the 64-bit version of 010 Editor should be used when linking to 64-bit external libraries.
    - Supports passing regular integer variables, floats and doubles and arrays of these types to external functions.
    - Support passing strings or wstrings to external functions.

  - Supports return types of any regular integer variables, float, double, string or wstring.
  - Use & to pass variables as references (pointers are not currently allowed in scripts and templates).
  - Structs cannot currently be passed to external functions.
  - Templates must be granted *ExecuteDLL* permission before being allowed to call functions in an external library. See the *Permissions* page of the Options dialog.
- **Templates and Scripts**
  - Templates and Scripts can now use functions in an external library (see External Functions in DLLs above).
  - Added support for time64_t including new functions StringToTime64T and Time64TToString.
  - The ImportFile function can now import Motorola with word-based addressing.
  - The default character set for Templates and Scripts for new installs of 010 Editor is now UTF-8 (the character set of existing installations of 010 Editor is not changed).
  - Syntax Highlighting is now done through Binary Templates.
  - Implement the HighlightLineRealtime function in a Template to provide syntax highlighting for a text file, or HighlightBytesRealtime function to provide highlighting for a binary file.
  - Added a variety of HighlightXXX functions to help with writing syntax highlighters.
  - Added new functions for testing the type of different characters: IsCharAlpha, IsCharAlphaW, IsCharNum, IsCharNumW, IsCharAlphaNum, IsCharAlphaNumW, IsCharSymbol, IsCharSymbolW, IsCharWhitespace, IsCharWhitespaceW.
- **General**
  - Now the last cursor position and scroll position for files are restored when 010 Editor is restarted (this can be turned off using the Editor page of the Options dialog).
  - Added '*Format > Delete Line*' command (Ctrl+Shift+Backspace shortcut by default).
  - Added Ctrl+Shift+N shortcut key to create a Hex file.
  - Triple-click the mouse to select a whole line in the text or hex editor.
  - Triple-click the mouse and drag to select by lines.
  - Added the '*Format > Delete Blank Lines*' command to delete empty lines in a file or selection.
  - Added '*Format > Delete Left Word/Delete Right Word*' to the format menu with shortcuts Ctrl+Backspace and Ctrl+Del.
  - Can export Motorola S19/S28/S37 hex data using word-based addressing (added *Motorola Hex Address Format* toggle to the Export Hex dialog).
  - Can import Motorola S19/S28/S37 hex data using word-based addressing (added *Motorola Hex Address Format* toggle to the Importing section of the Options dialog).
  - Added new 64-bit type time64_t to the Inspector (the new type has also been added to the Inspector.bt file).
  - Hid Unicode line in the Inspector as it could cause rare crashes when browsing large binary files. This can be re-enabled by choosing *Use Custom Inspector* in the Inspector Options and then editing the Inspector.bt file.
- **Options**
  - Add new *Character Sets* page for organizing and customizing character sets.
  - Remove *Syntax* page as Syntax Highlighting is now done through Binary Templates.
  - Added *Export Old Syntax Highlighter* button on the *Highlights* page if old custom syntax highlighters are found.
  - On the Editor page added *Show Startup Page when All Files are Closed* toggle.
  - On the Editor page added *Remember Last Cursor Position* toggle.
  - '*File > New*' menu is now controlled by clicking *Manage New File Types...* on the *Editor* page of the Options dialog.
  - Added *ExecuteDLL* permission to the *Permissions* page to allow Templates to execute functions in an external library.
  - On the Importing page added *Motorola Hex Address Format* radio buttons.
- **macOS**
  - Fixed a problem on macOS opening files from the command line that contain spaces.
  - Fixed a problem using the Esc key to cancel certain operations on macOS.
  - Fixed a problem with the hex editor Auto-line width calculation on macOS.
  - Fixed an issue with the version number display in the macOS Finder on some versions of macOS.
  - Fixed on macOS the undo stack was being lost after saving a file in some cases.
- **Bugs**
  - Fixed preprocessor symbol expansion was improperly expanding constants inside strings.
  - Fixed the Input Method Editor (IME) was not being displayed properly after certain operations.
  - Fixed a problem using the color picker in the Custom Colors section of the Find Bar Options

dialog.

- Fixed opening a file when a different character set is detected than is assigned in the current File Interface, then that file is assigned a per-file character set (for example if opening a Unicode XML file).
- Fixed a crash with Copy As Text Area operation in certain cases.
- Fixed an issue with the Portable version loading files from the command line when the application was already open.
- Fixed an issue displaying non-printable characters in the text editor on some machines.
- Fixed newly created Tool Bars where not being themed properly.
- Fixed a crash assigning to structs in certain cases (should generate an error instead of crashing).
- Fixing pressing Backspace when a selection was made and the cursor was at the beginning of the file was not deleting the selection.
- Fixed clicking 'View > Template Results' may not properly show a hidden Template Results panel when moved to the right side.
- Fixed an issue using Ctrl+C to copy text from the Output panel when text was selected in the main editor window.
- Fixed name and password are now being encoded before being sent to our online server for license checks.
- Fixed a problem using the FindAll and ReplaceAll functions using UTF-8.
- Fixed in the Template Repository, clicking the View button does not properly focus the displayed file.
- Fixed the Inspector when using a custom Template was not updating properly after FileNew was called in a script.
- Fixed using unpadded bitfields could result in a zero sized variable in some cases.
- Fixed a problem loading a custom theme that was imported.
- Fixed a bug on Linux with unreadable white text in the help file when using a dark OS theme.
- Fixed Copy Table command in the Checksum Results panel was only copying one column.
- Fixed Copy as Rich Text Format now creates background colors in a format compatible with WordPad.

Version 8.0.1 - September 29th, 2017

- Fixed crash on macOS using Printf and %Lx or %Ld in certain cases.
- Fixed templates with a file mask but no ID bytes were not being executed when a data file was opened.
- Fixed possible crashes in Find and Replace with Regular Expressions that use ^ or $.
- Fixed when installing on Mac, path info is added to .profile if it exists and .bash_profile does not.
- Fixed crash passing variables of the wrong type to user functions in certain cases.
- Fixed crash using Ctrl+Enter when certain panels were displayed.
- Fixed crash using ++ or -- operators on a struct (should be a syntax error).
- Fixed bug with FileSave and FileSaveRange properly converting UTF8 file names.
- Fixed crash in FileNameSetExtension function when passing an empty extension.
- Find/Goto/Select Bar Options button hotkey changed to Alt+P (Alt+O was not working).

Version 8.0 - May 3rd, 2017

The following is an overview of the new functionality in version 8.0 of 010 Editor:

- Different application themes are now available including dark and light themes.
- Themes can be customized, created, exported and imported using the Theme/Colors page of the Options dialog.
- A Portable version of 010 Editor is now available on Windows for running 010 Editor from USB keys.
- New visual style for the Text Editor, Hex Editor and File/Docking Tabs.
- Better support for high-DPI displays including new higher-resolution icons.

- Tabs in the Workspace, Inspector and Output windows can now be rearranged and undocked separately.
- Workspace, Startup page, plus other dialogs now show a list of files split into Name/Path.

The following is a list of all new features in version 8.0 of 010 Editor:

- **Themes**
  - 010 Editor now has themes which control the colors used for the application and how certain elements are drawn.
  - The default theme is a dark theme (Evening Sky) but a light theme is available (Blue Sky) plus others.
  - Themes can be chosen in the Welcome dialog when the application is first run or on the Theme/Colors page of the Options dialog.
  - A 'Classic' theme is available which is similar to previous versions of 010 Editor.
  - The individual colors of the themes can all be customized using the Theme/Colors page of the Options dialog.
  - New themes can be created, exported and imported using the Theme/Colors page.
  - Themes now control the colors of the Menu Bar, Tool Bars, File Tabs, Dock Window Tabs, Dock Windows, Tables, Graphs, Editors, etc.
  - Syntax Styles (for example for C/C++ or XML highlighting) are now controlled with the Theme.
  - Syntax Styles can be created, modified or deleted using the Theme/Colors page.
  - Background colors from templates are automatically darkened when using dark themes (see the new ThemeAutoScaleColors function to control this).
  - Bookmark colors are now controlled using the theme unless the *Use Custom Color* toggle is enabled in the Add Bookmark dialog.
  - New style for Find/Replace/Goto/Select bars and bars are now themeable.
  - Some UI elements can either be drawn using the OS standard method (called *Native* drawing) or with a custom themed drawing.
  - Native drawing of elements can be turned on or off using the *Options* drop-down menu on the Theme/Colors page of the Options dialog.
  - On macOS some UI elements are always drawn using Native rendering (for example the Menu bar and the Status bar).
  - The Startup page colors are now controlled using Themes.
- **Portable Version**
  - A Portable version of 010 Editor is now available for Windows 32 and 64-bit as a separate installer.
  - The Portable version can be placed on a USB key and moved between different computers.
  - Run the '010EditorPortable.exe' program to start 010 Editor (or the '010Editor.exe' program in the 'AppData' directory).
  - Licensing information is stored in the portable directory structure under 'AppData\Config' instead of the registry.
  - The program asks to copy a license from the registry to the portable directory structure the first time the portable version is run.
  - Scripts and Templates are automatically installed to the '010 Scripts' and '010 Templates' directories in the portable directory structure.
  - All directories in the Directories panel of the Options dialog are listed as being offset the '($BASEDIR)' constant.
  - No desktop icon is created and no extension to the Windows Explorer is installed.
  - To uninstall just delete the installed directory structure (no uninstaller is necessary).
- **High-DPI Displays**
  - Better support for high-DPI displays on Windows and Linux and Mac.
  - Includes new higher-resolution icons.
- **Docking Windows**
  - Individual tabs of the Workspace, Inspector and Output Windows can now be rearranged and torn off.
  - Tabs for the Docking Windows are drawn in a new style which is themeable.
  - A new docking menu can be accessed by clicking the down arrow in the Dock Header when the window is docked (not floating).
  - The docking menu can be used to *Hide* or *Float* the currently displayed Dock Window.
  - When docked, clicking the X button in the Dock Header hides all Tabs in the group.
  - The View menu now allows showing or hiding the individual tabs of the Dock Windows.

    - Commands were added to the View menu to show or hide all tabs in a Dock Window at the same time (for example see '*View > Workspace Windows > Show/Hide All Workspace Windows*').
    - *Allow Docking* option on the right-click menu of a Dock Window now applies to that particular tab, not the whole tab group.
    - The position of Docking Windows can be reset by clicking the down arrow in a custom Dock Header and choosing '*Reset All Docking*' from the drop-down menu or by using the -resetdocks command line option.
- **File Tabs**
    - New animated reorder when dragging File Tabs to new positions.
    - Can use the mouse scroll wheel to scroll through tabs if many tabs are open.
    - If a tab is dragged far enough, it will be torn off and can be moved to other tab groups (an arrow shows the insert position).
    - New visual style that is themeable.
    - Better support for dragging tabs when many tabs are opened.
    - Hint text for the tab now shows the size of the file.
- **Editor**
    - Double-click and drag on a text file to select by words.
    - In a text file Ctrl+Backspace deletes the previous word and Ctrl+Del deletes the next word.
- **Explorer**
    - Can use the *Root* field to set the root directory for the Explorer (only files and directories below that directory are shown).
    - Hide files that do not match the filter instead of just disabling them.
    - Removed the *Refresh* button as refreshes are done automatically.
- **Command Line**
    - Can use *-safe* command line option to start 010 Editor without running any scripts or templates on startup.
    - Renamed *-reset* to *-resetdocks* for resetting the Dock Window positions.
    - Can use *-install* on macOS to perform installation checks (check if 010 Editor has been added to the system path).
- **Templates/Scripts**
    - Background colors from templates are automatically darkened when using dark themes (see the new ThemeAutoScaleColors function to control this).
    - New function ThemeIsDark returns true if a dark theme is currently being used.
    - Date formats in the Template Results can now be set to different formats (see the Inspector page of the Options dialog).
    - New functions GetDefaultDateFormat, GetDefaultTimeFormat, and GetDefaultDateTimeFormat can be used to query the chosen date/time formats in the Inspector page of the Options dialog.
    - Can enabled synchronized scrolling of the Template Results using '*Window > Synchronize Template Results Scrolling*' (can also be turned on in the Compare dialog).
    - Added function SetBytesPerLine to override the number of bytes per line of a hex editor.
    - Fixed when using on-demand structs the endian setting was not being properly read from the parent.
- **General**
    - Workspace, Startup page, Window List, plus other dialogs now show a list of files split into Name/Path.
    - Can resize the columns of the Workspace and the Startup page Recent Files list.
    - Windows Installer is smaller and faster and does not require restarting the computer.
    - The current Find Results scroll position is saved and restored when switching files.
    - Can set a different date or time format for the Inspector/Template Results on the Inspector page of the Options dialog.
    - Add Bookmark dialog has been rearranged and Bookmark colors are controlled by the theme unless the *Use Custom Color* toggle is enabled.
    - Can clear the search history for the Find Bar or Replace Bar by clicking '(clear find history)' from the Find Bar history list.
    - When performing a Compare, the Floating Tab Group is hidden if files are moved to the main window as a result of tiling.
    - Save All command now skips over read-only files instead of showing an error that they could not be saved.
- **Options**
    - Can set a different date or time format for the Inspector/Template Results on the Inspector page of the Options dialog.

    - Colors page of the Options dialog renamed to Themes/Colors and can be used to control the current theme.
    - Individual colors are marked as bold when modified and can be reset individually.
    - Color changes are applied immediately in most cases so changes can be seen without having to press the OK button.
    - Removed the Style Options page (styles are now controlled on the Theme/Colors page).
    - Renamed Shadow Cursor to Inactive Caret on the Editor page.
    - Added a Separator Spacing option on the Hex Editor page.
- **macOS**
    - Added Minimize command on the Window menu (Ctrl+M).
    - Shortcut key for Compare changed to Ctrl+Shift+M.
    - Changed shortcut keys for Show and Hide Workspace/Inspector/Output/Floating panels from Alt+(number) to Alt+Shift+(number).
    - Added -install command line option to perform installation tasks again (check if the program is added to the system path).
    - When installing and the program is added to the system path, a copy of .bash_profile is made before overwriting.
    - Fixed issue running 010 Editor from the command line on some macOS systems.
    - Fixed crash on macOS with QAccessibleEvent::uniqueId function.
- **Linux**
    - Removed dependency on libpng12.so library.
    - Fixed some application focus issues on Ubuntu 16 when using Alt+Tab.
- **Bugs**
    - Fixed issue with local variables sometimes being placed in the wrong scope when allocating structs inside functions.
    - Fixed underscores were not being printed correctly.
    - Fixed some typos with the Writing Binary Templates tutorial.
    - Fixed a problem using parentof in a custom read function.
    - Fixed Atoi/Atof would work incorrectly when using a non-null-terminated string.
    - Fixed the Floating Tab Group was sometimes not focused correctly after showing it using the View menu.
    - Fixed when reloading a file that is in the repository, the repository icons in the File Bar were not updated correctly.
    - Fixed issue with the dialog size being too small when using the InputRadioButtonBox or InputString functions.
    - Fixed a Base64 import which was not divisible by 4 bytes could sometimes result in extra 00 padding bytes being imported.
    - Fixed some files were being incorrectly identified as Intel-Hex or Motorola on drag-and-drop.
    - Fixed a crash deleting a script or template from the Options dialog when none were installed.
    - Fixed a problem accessing the characters of a non-local string when the string is passed to a function inside a template.
    - Fixed updating the repository on machines using a proxy.
    - Fixed updating the repository on some machines when a directory was not being created properly.

Version 7.0.2 - June 6th, 2016

- Fixed newly downloaded repository records could be incorrectly duplicated on some systems.
- Fixed problem on some Linux systems copying data from other applications to 010 Editor.
- Fixed problem with Mac OS X certificate on OS X 10.9+.
- Fixed problem where a long script which opens files could steal focus from other applications.
- ID Bytes can now match the first 2048 bytes of a file (was 1024 bytes).
- Synchronized Scrolling can now work between text and hex files.

Version 7.0.1 - May 13th, 2016

- Fixed Find in Files would crash on some Windows 8/10 64-bit machines.
- Fixed ID Bytes detection may fail on some files or drives over 2 GB in length.
- Mac application is now properly signed.
- Fixed help viewer crash on Windows XP.
- Fixed output from Printf inside an on-demand struct or read function was not being properly displayed in the Output area.
- Fixed problem displaying the Print dialog on some machines.
- Fixed problem saving files on Windows with no extension.
- Fixed issue loading bookmarks containing non-ASCII character names.
- Fixed crash on some Mac machines rendering certain fonts.
- Now show a warning that the Print dialog cannot be displayed if no printers are installed.

Version 7.0 - March 23rd, 2016

The following is an overview of the new functionality in version 7.0 of 010 Editor:

- New 010 Editor Repository holds an online collection of Binary Templates and Scripts that have been submitted by users of the software.
- Files from the Repository can be downloaded and installed or uninstalled in 010 Editor with the click of a button.
- Templates or Scripts can be submitted to the Repository directly from 010 Editor.
- Can handle multiple versions of files in the Repository including updates, diffs, merges, etc.
- Updated the style of a number of dialogs including the Startup page and Welcome page.
- Added 'Using the Repository' tutorial to teach the basics of the repository.
- Major upgrade to UI library resulting in a number of minor bug fixes and visual fixes.
- New webpage 'http://www.sweetscape.com/010editor/repository/' shows the Repository in real time.
- New 'Repository' menu displayed in the File Bar above a Template or Script.

The following is a list of all new features in version 7.0 of 010 Editor:

- **Repository**
  - New Repository Dialog accessed by clicking 'Templates > Template Repository' or 'Scripts > Script Repository'.
  - Repository dialog lists all files in the repository and can sort by Category, Alphabetic, or Newest.
  - Install or Uninstall files from the Repository Dialog with the click of a button.
  - Can search for Scripts or Templates in the Repository Dialog with the search field.
  - Licenses for 010 Editor now include free Support, Upgrades, and Repository Updates for 1 year from the date of purchase.
  - When opening a file a dialog pops up if a Template is found in the Repository that can parse the file (can install the Template or ignore).
  - Repository can check for 'ID Bytes' for data files before asking to install.
  - New webpage 'http://www.sweetscape.com/010editor/repository/' shows the Repository in real time.
  - Download package includes a number of files from the Repository (called the Local Repository) so files can be installed even on computers not connected to the internet.
  - Online Repository checked for new updates every 3 days (can be changed or turned off).
  - Renamed a number of files in the repository (e.g. "BMPTemplate.bt" => "BMP.bt").
  - Recent repository updates are listed on the Startup page (double-click on an item to view) or on the History tab of the Repository Dialog.
  - Can submit files to the Repository directly from 010 Editor (see 'Submit File' in the 'Repository' menu above a Script or Template).
  - Submission dialog checks the header for errors and provides an easier way to add Category or

    History information to the submission.
    - Note that all submissions are reviewed by SweetScape Software before being allowed in the repository.
    - New 'Repository' menu in the File Bar above each Template or Script in the editor.
    - Icon in the File Bar shows the file repository status (Installed from Repository, Modified, Update Available, Conflict).
    - Use the Repository Menu to Update, Check for Modifications, Diff, Revert, Delete, or View Repository.
    - Full update engine including powerful 3-way merge algorithm and showing conflicts.
    - 'Available Versions' table in the Repository Dialog can show multiple versions of files in the Repository. The different versions can be viewed, installed or compared.
    - Display a special warning if a template installed from the Repository asks for permissions to read or write to other files.
    - History tab of the Repository Dialog is displayed when new updates downloaded (can be turned off).
    - Included new tutorial 'Help > Tutorials > Using the Repository'.
    - Default Templates and Scripts (e.g. 'BMP.bt', 'IsASCII.1sc', etc) are now installed from the Repository.
- **Templates/Scripts**
    - New Template and Script repository allows installing and submitting Templates and Scripts easily.
    - Templates and Scripts can be listed by Category in the Templates and Scripts menu.
    - Standard templates have been renamed (e.g. "BMPTemplate.bt" => "BMP.bt").
    - Templates can now have 'ID Bytes' which provide an extra check that a template can parse a data file. For example, the 'ZIP.bt' file has ID Bytes '50 4B //PK'.
    - Renamed 'Templates > Online Template Repository' to 'Template Repository' which now shows the Repository Dialog.
    - Renamed 'Scripts > Online Script Repository' to 'Script Repository' which now shows the Repository Dialog.
    - Renamed 'Templates > Edit Template List' to 'View Installed Templates'.
    - Renamed 'Scripts > Edit Script List' to 'View Installed Scripts'.
    - The Output panel is no longer shown by default when Printf is called in a template (this can be controlled using the Compiling Options page).
    - New standard comment header at the beginning of Templates or Scripts created with 'Templates > New Template' or 'Scripts > New Script'. New header includes History, Category, File Mask, and ID Bytes.
    - Local strings are now displayed using the character set from the current file (used to always be ANSI).
    - #include now searches the Template and Script Repository Directories.
- **General**
    - Updated style for a number of dialog boxes.
    - Inspector and Workspace are now docked to the right side by default (just drag-and-drop to move to the left side).
    - Updated Welcome Page with information about the repository.
    - Changed default selection background color.
    - A dialog now pops up when Support, Free Upgrade, and Free Repository Updates are about to expire (about 30 days left) and when expired.
    - Updated Startup page with modified style and Repository news.
    - License information is now displayed on the lower-right corner of the Startup page (click to display the Register dialog).
    - Downloading application news/updates is now done in the background instead of when the application was shut down.
    - Small modifications to the default colors for the editor.
    - Read-only state of files is saved with the workspace.
    - When a new license is entered a dialog displays the end dates for Support, Free Upgrades and Free Repository Updates.
    - About page now displays current license info and has a button to show the Register dialog.
    - Major upgrade to UI library resulting in a number of minor bug fixes and visual fixes.
- **Options**
    - Template Options and Script Options now display the list of installed files, including files that have been installed from the Repository.
    - Include the category name in the list of installed templates and scripts.
    - Easier way to add local Templates and Scripts to the list. Clicking the 'Add...' button displays a

multi-select file dialog box.
    - When adding local Templates or Scripts information such as File Mask, ID Bytes, and Category can be extracted from the comments at the beginning of the file.
    - Templates and Scripts now have an 'Edit...' button to load the file in the editor.
    - Added 'ID Bytes' for Templates as an extra way to check that data files can be parsed by a Template (ID Bytes can be turned off by unchecking the 'Require' toggle).
    - Added 'Status' info for Templates and Scripts and click the 'Show...' button to display the Repository dialog if installed from the Repository.
    - Added 'Set Shortcut' button on Scripts and Templates Options for an easy way to set a shortcut for these items.
    - New Repository Options page displays options for the Repository (some options are duplicated in the Repository Dialog).
    - Directory Options now allows setting the default Template and Script directory (moved from the Compiling Options).
    - Directory Options can set the directory where Scripts and Templates are installed from the Repository.
    - Moved File Dialog settings to a separate File Dialog Options page.
    - Added an option on the Compiling page to control whether the Output panel is displayed automatically when Printf is called in a Script or Template.
    - List of options on the left side of the Options Dialog is now resizable (you may need to enlarge the dialog first).
    - Options Dialog remembers its size when 010 Editor is shutdown and restarted.
    - New constants $TEMPLATE_REPOS_DIR and $SCRIPT_REPOS_DIR can be used in file names in the Options dialog.
- **Mac**
    - New installation method. Just drag-and-drop to install.
    - Now support OS X 10.7 and higher (sorry 10.6 users, we tried).
    - Help viewer now renders fonts better on retina displays.
    - Fixed some font spacing issues on dialogs on the Mac.
    - For OS X 10.7 and 10.8 the default hex editor font is now Andale Mono (there is a rendering bug using Courier New on these systems).
- **Bugs**
    - Fixed a crash in the Help Viewer when doing a search for certain keywords.
    - Fixed problem on Linux using Save As to save a large file in certain cases.
    - Fixed issue with multiple warnings when running scripts or templates are split onto separate lines.
    - Fixed bug in dialog asking to save a file which is marked as read-only was not displaying the file name correctly.
    - Fixed issue using the regular expression anchor '$' on files with DOS linefeeds.
    - Fixed using increment or decrement on a variable inside a variable accessor could sometime be called multiple times.
    - Fixed issue where the Addresses for the last line of a large text file were not displayed correctly using 'Byte Number (Decimal)'.
    - Fixed issue with wrong the background color sometimes being set when defining structs within structs.
    - Fixed bug with the FindFirst function incorrectly wrapping when starting the search from the end of the file.
    - Fixed crash when using an invalid regular expression in the FindFirst function.
    - Fixed using ReadByte/ReadInt/etc with no arguments and positions greater than 0xFFFFFFFF.
    - Fixed possible crash in FindOpenFile function after using FileClose.
    - Fixed when a custom name function had an error an empty string was being displayed for the name.
    - Fixed problem allocating a local variable after a struct definition sometimes having the improper scope.
    - Fixed FindInFiles and FindFiles now return strings in UTF-8 format.
    - Fixed possible crash assessing types defining by a call to RunTemplate.
    - Fixed FindFirst and FindAll would incorrectly search the whole file when size was zero.
    - Fixed crash using more than 15 parameters to SScanf.
    - Increased the maximum parameters for SScanf from 15 to 30.
    - Fixed bug with WMemSet function when the value to set was more than 255.
    - Fixed a problem viewing process names that contained non-ASCII characters.
    - Fixed a memory leak in user functions in certain cases.

- Fixed a crash when a script or template ends with an unmatched '*/'.
- Fixed an error message when trying to allocate an array more than 4 GB.
- Fixed a bug using the '+' operator to add strings and single characters in certain cases.
- Fixed some focus issues on Windows when opening a file and 010 Editor was minimized to the task bar.
- Fixed restoring the application to the correct monitor when 010 Editor was maximized and then closed.
- Fixed typing a backslash character when using the Japanese character set on Mac OS X.
- Fixed issue when FileOpen failed and afterwards an incorrect file was selected.

Version 6.0.3 - Sept 25th, 2015

- Fixed a crash when a find occurrence was found on a line containing more than 65535 characters.
- Fixed the line number was not being properly displayed in the Find in Files results for text files.
- Fixed a problem opening file names on Mac OS X containing certain characters.
- Fixed a bug using the number constant 0xffffffffffffffff in a script or template;
- Fixed a crash when starting 010 Editor during the 30-day trial period and the system clock was set back (an error is now displayed).
- Fixed a crash when using an invalid regular expression in a syntax highlighting rule.
- Fixed the wrong number of pages being printing in certain cases when printing a hex file.
- Fixed Export CSV of the Find results for Hex Bytes sometimes improperly included an extra linefeed.
- Fixed the AddBookmark function was not properly using the UTF-8 character set for the 'name' parameter.

Version 6.0.2 - April 13th, 2015

- Fixed problem opening physical drives on Linux.
- Fixed error using 'Copy As > Copy as Base64' and 'Copy As > Copy as Uuencoding'.
- Now display an error when an invalid character constant is encountered in a script or template.
- Fixed FILETIME error in the Inspector on Linux/Mac 64-bit.
- Fixed error in the first result using Find Strings on a custom range.
- Updated color for the Welcome and New Version dialogs.
- Fixed error in the documentation for the Exec function about the function's return value.
- Fixed error message in the footer of the Manual on the Mac.

Version 6.0.1 - January 28th, 2015

- Fixed crash on exit on some Mac OS X machines.
- Fixed problem opening processes on 64-bit Windows.
- Made the color of the Startup page darker (was too bright on some monitors).
- Can now change or reset the color of the Startup page by clicking the 'Options' button on the Startup page.
- Fixed some issues using some IME's with the hex editor.
- Fixed bytes not being swapped correctly when printing and swap by group is enabled.

Version 6.0 - December 16th, 2014

The following is an overview of the new functionality in version 6.0 of 010 Editor:

- 010 Editor is now available as a 64-bit program on Windows, Mac and Linux.
- Searching with Regular Expressions.
- Improved Find speed, up to 10 times faster in some cases.
- Changed the style of the Startup page, Welcome page, and File Bar.
- Added new 'Writing Binary Templates' tutorial to teach the basics of creating Binary Templates.
- Can export the Template Results in XML format.
- Can import Find results.

The following is a list of all new features in version 6.0 of 010 Editor:

- **General**
  - 010 Editor is now available as a 64-bit program on Windows, Mac and Linux.
  - Redesigned Startup page which is automatically shown on startup (this can be changed with the *Startup Action* section on the Startup page).
  - The Startup page is automatically shown when all files are closed (this can be turned off with the Options button on the Startup page).
  - Redesigned Welcome page with a link to the new Writing Binary Templates tutorial.
  - When asking to save changes to multiple files (for example, when closing 010 Editor), all the files to save will be listed in a single dialog.
  - Redesigned style for the File Bar/Ruler at the top of each Editor Window.
  - When a file is modified by an external program, the dialog asking to reload changes has two new options: *Always Reload* and *Always Ignore*. *Always Reload* will continually reload the file if changes are detected and *Always Ignore* will not prompt for reloading any more.
  - The Floating Tab Group is automatically hidden when all tabs in it are closed (this can be turned off in the Editor Options dialog).
  - Moved *Save Selection* command to the File menu.
  - A popup message is shown when a new version of 010 Editor is available (this can be turned off with the General Options dialog).
  - Added '*Help > Check for Updates*' to check if a new version of 010 Editor is available.
  - When created new files, the default linefeeds to use are now specified in the File Interface (see the *File Interfaces* page of the Options dialog).
  - When editing a word-wrapped line in the Text Editor, pressing the Home key twice goes to the very beginning of the line and pressing the End key twice goes to the very end of the line.
  - Exporting CSV from tables now exports strings as UTF-8.
  - When using Import Hex, the directory where the file was imported is remembered when using Save As to save the file.
  - 'Highlighting > Control Characters' now includes ASCII 127 by default (will need to reset the Highlighting options to take effect).
  - Clicking '*Help > Support by Email*' now opens a web form which allows sending a message on all platforms instead of trying to use a local email client.
  - The current version number is now displayed in the Register dialog.
  - Increased maximum line width in the Hex Editor from 1024 to 16384 bytes.
  - Syntax Highlighting can now apply rules based on Regular Expressions.
  - Made text in the About dialog easier to read.
- **Mac OS X**
  - Better support for fonts on Mac Retina displays.
- **Linux**
  - Linux can now use Gtk styles if available.
- **Find**
  - Can search with Regular Expressions (click *Search with Regular Expressions* on the Options panel of the Find Bar).
  - Improved Find speed, up to 10 times faster in some cases.
  - Default Find type for text files is now *Text* which searches for strings in the current character set of the file, whether that be ASCII, Unicode, UTF-8, etc.
  - The Find type *ASCII* is now just used for finding ASCII+ANSI strings (use the Find type *Text* to search for other character sets).
  - Can *Import* find results that were exported from the *Find* or *Find in Files* tabs (Import and Export can be done from the right-click menu of the results table).
  - Added a toggle to the Find Bar Options to prevent the Find Bar from disappearing after search.

  - Display a warning when more than 10000000 find occurrences are found (this limit can be changed or turned off using the Find Bar Options).
  - In the Find Bar Options, use the *Use Custom Color* toggle to apply a custom color for any search.
  - Fixed issues searching for very long arrays of hex bytes (more than 16384 bytes).
  - Now display an error when a read-error is encountered when searching on a Windows hard drive.
  - Increase the maximum line length in the Find results from 4096 to 65535.
- **Tutorials**
  - Added new 'Writing Binary Templates' tutorial to teach the basics of creating Binary Templates.
  - Moved Tutorials to the Help menu.
- **Templates/Scripts**
  - Can export the Template Results in XML format.
  - Can layout Template Results to the right of the Hex Editor Window by right-clicking the Template Results and choosing *Template Results Position*.
  - Exporting Template Results as CSV or XML exports strings in UTF-8 format.
  - Default file name for exporting Template Results now includes the file name.
  - Preprocessor #define now works with symbols that have previously been defined.
  - Added note to the manual about preprocessor constant *_010_LINUX* being defined on Linux.
  - **New Functions**
    - Added RegExMatch, RegExMatchW, RegExSearch, and RegExSearchW functions for searching within strings for regular expressions.
    - Added ExportXML function to export Template Results as XML.
    - Added FindStrings function to perform a string search similar to '*Search > Find Strings*'.
    - New functions GetBackColor and GetForeColor can be used to query the currently active color.
    - Added GetMouseWheelScrollSpeed and SetMouseWheelScrollSpeed to get or set the current mouse wheel speed.
  - **Function Updates**
    - Can use regular expressions in the FindAll, FindFirst, FindInFiles or ReplaceAll functions with the 'method=FINDMETHOD_REGEX' parameter.
    - Memcpy/WMemcpy functions now have extra parameters 'int destOffset=0, int srcOffset=0' that can be used to read or write from the middle of an array.
    - Functions ReadLine, TextReadLine, TextWriteLine, TextReadLineW, TextWriteLineW, TextGetLineSize all have an option 'int includeLinefeed=true'. When this parameter is false, linefeeds are not included in the read, write, or size operation.
    - ReadByte/ReadInt/ReadShort/etc. now default to the position 'FTell()'.
    - Can retrieve the error code from a process called using the Exec function with the *errorCode* parameter.
- **Command Line**
  - Use *-line:* to jump to a specific line in a text file.
  - Use *-goto:* to jump to a byte address, sector, line or short in a file.
  - *-goto:* supports the syntax from the Goto Bar (for example, use '-goto:+10,s' to skip ahead 10 sectors).
  - Set a selection from the command line using *-select:<start>:<size>* or *-select::<size>*.
  - Old syntax of using @ after a file name to set the cursor or selection is still supported but will be deprecated in the future.
- **Options**
  - New option on General page: *Show Popup when a New Version is Available*.
  - Can set default linefeeds when creating new files on the File Interface page instead of the Text Editor page.
  - New *Directories* page used to control the initial directory when opening a file dialog box (open file, save file, import, export, open template, save template, open script, save script).
  - New option on Editor page: *Auto-hide Floating Tab Group when All Files are Closed*.
  - Moved options for the cache into a separate *Cache* page.
  - Ruler Color now effects the File Bar color as well.
- **Bugs**
  - Fixed problem on Mac using the %Lx format specifier in Printfs.
  - Fixed some drives on Windows were not correctly displaying the last few sectors.
  - Fixed Find on some Windows drives would stop before reaching the end of the drive.

- Fixed crash exporting a very large Find results from a text file.
- Fixed Histogram percentages not always totaling 100% with some types of histograms.
- Fixed problem displaying histogram for int64's.
- Fixed crash calling a function in a script when the first character in the argument list was a comma.
- Fixed issue where focus could be incorrectly grabbed from other applications in some cases.
- Fixed cannot delete bookmarks that exist beyond the end of the file.
- On Windows, fixed using file masks '.*' and '.' should match files with no extension.
- Fixed crash using bookmarks on files with file names longer than 128 characters (no limit now).
- Fixed a case where it was possible to cancel a Find in Files directory scan and get an incorrect progress state.
- Fixed Replacing text with nothing did not show any text in the Find tab of the Output panel.
- Fixed replace two spaces with nothing can cause a crash in certain cases.
- Fixed right-click and Mark Selection End on an existing selection sometimes did not work as expected.
- Fixed highlighting issue using Lock to Selection on the Find Bar with selections greater than 2 GB.
- Fixed ensuring Inspector.bt is called when the selection size is changed.
- Fixed opening a single file from the Windows Explorer right-click menu would sometimes not correctly restore previously open files.
- Fixed crash with long Comparisons when sorting set to the *Result* column.
- Fixed some address issues using Copy as Text Area.
- Fixed bug writing 0x0d0a to a text file from a script when two linefeeds would be displayed instead of one.
- Fixed bug with the FindFirst function searching up when a match occurs at the very top of the file.
- Fixed some progress issues when doing a long Replace operation.

Version 5.0.2 - July 28th, 2014

- Fixed problem writing to drives on Windows 7 and later versions. Windows now requires that all open file handles be closed before changes can be saved.
- If all file handles are not closed when a drive is opened, a warning is displayed and the drive is marked as read-only (Windows only).
- In the Open Drive dialog each physical drive shows a list of logical drives it contains (Windows only).
- Added 010 Editor to Mac Finder 'Open With' right-click menu for most file types.
- Changed Chinese Simplified encoding from GB2312 to GB18030 (GB18030 is a superset of GB2312).
- Fixed using Printf/FPrintf and %s with strings more than 4096 characters long.
- Fixed error message on some machines about regsvr32 in the Windows installer.
- Fixed crash with sizeof operator on certain structs declared with a *size* attribute.
- Fixed not being able to cancel a script after the RunTemplate function was called.
- Fixed Find in Files was not properly releasing some file handles.
- Functions InputNumber/InputFloat/InputString properly resize the dialog before it is displayed.
- Fixed problem using Exec function on Linux and Mac with file names than contain spaces.
- Using %f parameter in External Programs now auto-adds quotes for file names containing spaces.
- Now show a warning when attempting to open Template Results hierarchies deeper than 31 levels.
- Fixed 010 Editor unable to run on some Linux machines due to a Qt library issue.
- Fixed crash on Mac with duplicate enums.

Version 5.0.1 - June 19th, 2014

- Fixed Copy As Hex Text crash when the hex editor Line Width is set to Auto Width.
- Fixed possible text corruption in Linux when deleting.

- Fixed crash using SHA-512 on certain large files.
- Fixed setting a shortcut key for a script starting with a lowercase letter.
- Fixed possible crash with Allow Multiple Find Ranges and large files.
- Fixed Hex Operation dialog becoming uneditable after first operation on the Mac.
- Fixed crash with certain non-terminated strings in scripts or templates.
- Fixed problems opening files larger than 4 GB on Linux.
- Fixed clicking the upgrade link in the Register dialog when the licensee name contains certain non-ASCII characters.
- Added instructions on running 010 Editor on Ubuntu 14.04 64-bit.

Version 5.0 - June 24th, 2013

The following is an overview of the new functionality in version 5.0 of 010 Editor:

- Syntax highlighting for HTML, XML and PHP.
- Customizable syntax highlighting.
- Column mode (hold Ctrl while dragging the mouse to select columns).
- Customizable toolbars and right-click menu.
- Available for Linux, Windows and Mac OS X.
- New format menu with options for Uppercase, Lowercase, Capitalize, Tabify, Untabify, Comment Selection, Uncomment Selection, Increase Line Indent, Decrease Line Indent, and Trim Trailing Whitespace.
- Show whitespace.
- Better interface for customizing shortcut keys.
- Added SHA-512 algorithm.
- Added 16-bit half-float data type (hfloat).
- Support for importing and exporting binary text.
- 19 new functions for Scripts and Templates.

The following is a list of all new features in version 5.0 of 010 Editor:

- **Syntax Highlighting**
    - Added syntax highlighting for HTML, XML and PHP.
    - New Syntax Highlights can be created or existing Highlights modified using the Syntax page of the Options dialog.
    - Support styles for Syntax Highlights so multiple rules can share a single color.
    - Edit styles using the Styles page of the Options dialog.
    - Easily modify the list of highlighted keywords for a Syntax Highlight.
    - Support multiple rule types including Multi-Line Block, Single-Line Comment, Keywords, Single-Line Block, C-Style String, Tag Name, and Tag Attribute.
    - Rules can be applied with Ignore Case for case insensitivity.
    - Support multi-line C-style strings with '\'.
    - Import or export the list of Syntax Highlights including styles using the Import List or Export List buttons.
    - Support sub-rules so that different types of syntax highlighting can be applied to the same file.
- **Column Mode**
    - Support Column Mode for editing the columns of text or hex data.
    - Easily make a column selection with the mouse by holding down Ctrl while dragging.
    - Enter column mode using Alt+C or click the Column Mode icon in the Toolbar.
    - Make column selections and copy or paste them using the clipboard.
    - Make a column selection and start typing to insert text on each line at the same time.
    - Column selections supported in Hex mode as well.
    - Click and drag straight down to create a special column insert line.
    - When the clipboard contains just a single line of data, pasting the data with a column selection pastes the data on each line of the selection.
- **Linux Version**

    - Added Linux version with installer.
    - Officially support Ubuntu 10.04 and later.
    - Available as a 32-bit program (can be installed on 64-bit OS as well).
    - Can load scripts and templates by double-clicking them in the Ubuntu File Manager.
    - Support Unix-style middle-click clipboard.
- **Toolbar Customization**
    - Can customize Toolbars by using the Toolbars page of the Options dialog.
    - Access the Toolbars page by right-clicking on a Toolbar and selecting 'Customize...'.
    - Drag-and-drop actions to the Toolbar list to insert items into the Toolbars.
    - Insertion point marked with red line.
    - Drag icons out of the Toolbar list to delete items from the Toolbars.
    - Create, delete and rename custom Toolbars.
- **Menu Customization**
    - Customize the Editor Right-Click menu using the Menus page of the Options dialog.
    - Can also access the Menus page by right-clicking on an Editor Window and choosing 'Customize...'.
    - Drag-and-drop actions from the Actions tree to the menu to insert actions.
    - A red line marks the action insertion point.
    - Drag items out of the menu to delete actions from the menu.
    - Add separators to the menu using the special Separator action.
    - Add sub-menus to the menu with the Submenu action.
    - Double-click on a sub-menu name to rename the sub-menu.
- **Format Menu**
    - Added new Format menu for working with text data.
    - Can convert text to *Uppercase*, *Lowercase*, or *Capitalize* (e.g. APPLE, apple, Apple).
    - Converts spaces to tabs using *Tabify* or tabs to spaces using *Untabify*.
    - Can add or remove comments from the selection using *Comment Selection* and *Uncomment Selection*.
    - Commenting supports line comments (e.g. '//') and multi-line comments (e.g. '/*' and '*/') based off the current Syntax Highlighting scheme.
    - Format menu supports *Increase Line Indent* to add tabs or *Decrease Line Indent* to remove tabs (similar to using Tab or Shift+Tab in the editor).
    - *Trim Trailing Whitespace* removes any spaces or tabs from the end of each line.
    - *Tabify*, *Untabify* and *Trim Trailing Whitespace* operate on the whole file if no selection is made.
- **Show Whitespace**
    - Use 'View > Tabs/Whitespace > Show Whitespace' or the icon in Toolbar to display whitespace.
    - Shows symbols in a text file where spaces and tabs exist.
    - The color of the symbols can be controlled in the *Colors* page of the Options dialog.
    - The Show Whitespace setting is remembered with the current File Interface.
- **Shortcut Keys**
    - New, easier-to-use interface for changing Shortcut keys in the Options dialog.
    - Better support for Shortcut keys on the Mac.
    - Enter a Shortcut key by pressing the key on your keyboard instead of having to type out 'Ctrl+M'.
    - Use '*Help > View Shortcut List*' or click the *List Shortcuts* button in the Options dialog to view a list of all shortcuts sorted by shortcut name.
    - The list of shortcuts shows any customized shortcuts as bold.
    - Allow multiple shortcut keys to be assigned to a single action.
    - Better interface for removing shortcut keys from actions.
    - Can see any conflicting actions when assigning shortcut keys.
- **New Functions**
    - Converts a set of hex bytes to a double, float, or hfloat with ConvertBytesToDouble, ConvertBytesToFloat, or ConvertBytesToHFloat.
    - Support copying an array of hex bytes to or from the clipboard with CopyBytesToClipboard and GetClipboardBytes.
    - Save a portion of a file with the FileSaveRange function.
    - Access the custom clipboards with the GetClipboardIndex and SetClipboardIndex functions.
    - Access other properties of bookmarks with the GetBookmarkArraySize, GetBookmarkBackColor, GetBookmarkForeColor, GetBookmarkMoveWithCursor, and GetBookmarkType functions.

    - Added a radio button box input dialog using the function InputRadioButtonBox.
    - Ask the user for a directory with the InputDirectory function.
    - Can overwrite a block of bytes using the OverwriteBytes function.
    - Added ReadHFloat and WriteHFloat for working with half-floats.
    - Set environment variables within the working process with the SetEnv function.
- **Functions Updates**
    - FileClose was improperly asking to save changes on a modified file.
    - GetClipboardString, CopyStringToClipboard, and ClearClipboard uses the current clipboard (see GetClipboardIndex and SetClipboardIndex).
    - CopyStringToClipboard has a charset parameter to inform the clipboard of the type of data being copied.
    - Can specify a custom polynomial or initial value with the Checksum functions.
    - Added an optional number of bytes to read to the ReadWString, ReadString, ReadLine, and ReadWLine functions.
    - FileNew has a new makeActive parameter to control if the created file is set as the active file.
    - Added an optional fill character to the InsertBytes function.
    - InputOpenFileName, InputOpenFileNames, and InputSaveFileName functions now use UTF-8 strings.
    - The Sleep function now refreshes the screen before long sleeps (more than 1000 milliseconds).
    - Added SHA512 algorithm to the checksum functions.
    - Added Binary Text format to the Import/Export functions.
    - TextGetNumLines now returns -1 if the current file is a hex file.
    - Update help that Strlen returns the number of bytes instead of the number of characters.
- **Templates/Scripts**
    - Added 16-bit Half-float (hfloat) data type.
    - Automatically cast between hfloat, float and double data types.
    - Tooltips now properly use any custom <name=> functions.
- **General**
    - Updated visual style of the interface and updated some icons.
    - Added new File Interfaces HTML, PHP, and XML.
    - When selecting more than one line in a text editor, the number of selected lines is now shown by default in the status bar.
    - Added SHA-512 hash algorithm.
    - Able to delete individual recent files or clear the list of recent files by right-clicking on the Recent Files list in the Workspace or the Startup page.
    - Added Goto button to the Goto Bar.
    - Added new File Interfaces to the New list.
    - Auto-detect XML files.
    - Added -readonlyall command line option to mark all files as read only.
    - Added 'Selection > Goto Selection Start' and 'Selection > Goto Selection End' for jumping to the beginning or the end of a selection from the right-click menu.
    - Synchronized scrolling now only works between all hex files or all text files.
    - Faster Undo/Redo for operations which contain a large number of small operations.
    - Renamed 'Toggle Word Wrap' action to 'Word Wrap'.
    - Renamed configuration file to '010Editor50.cfg'.
    - Removed 3-computer limit in the End-User License Agreement.
- **Import/Export**
    - Added Import/Export option for the Binary Text format (e.g. 10100010 11110000).
    - Added *Copy As Binary Text* and *Paste From Binary Text* to the Edit menu.
    - 'Copy as <format>' now uses the displayed number of bytes per line (it was using the number of bytes from the last export).
- **Options**
    - Can disable the auto-import of Intel Hex or Motorola files when dragging and dropping from the system File Manager (see the Importing page).
    - Added option to disable the Backspace/Delete key in Overwrite mode for hex files (see the Hex Editor page).
    - Can control the color of the Show Whitespace characters on the Colors page.
- **Bugs**
    - Fixed bug with Save Selection on a newly created file.
    - Fixed crash with -compare command line option when the file names were the same.

- Fixed issue with updating the Recent Files list when scripts or templates were opened.
- Deleting a file from the Workspace Recent Files list did not update the Startup Page Recent Files list.
- Fixed up *Insert Date*/*Insert Color* on big endian unicode files.
- Fixed up 'Move to New Vertical Tab Group' missing after a Compare.
- Fixed warning dialog for optimized structs cannot be dismissed on Mac.
- Fixed crash when an improper argument was passed to a function inside a script in certain cases.
- Fixed error message in the Windows installer about Regsvr32.
- Possible to delete part of a DOS linefeed in certain cases.

Version 4.0.4 - March 20th, 2013

- Fixed Find Strings finding unicode strings on odd byte boundaries.
- Fixed crash assigning to a non-existent variable from a script.
- Fixed crash in find hex bytes when using ',a'.
- Fixed crash when checksum functions are used inside custom read/comment functions.

Version 4.0.3 - November 8th, 2012

- Fixed crash with Printf when using certain combinations of parameters.
- Fixed problem using Goto on the Mac.
- Fixed using on-demand structs within custom read functions in certain cases.
- Fixed printing multiple copies of files.
- Disabled redirection of files within the main Windows directory.
- Fixed some progress issues when using Find functions inside of scripts.
- Show error instead of crashing when trying to copy huge blocks of data to other applications.
- Fixed order of some items in the View menu in 'Tools > Options > Keyboard'.

Version 4.0.2 - July 24th, 2012

- Fixed a crash using Find on certain long text files.
- Fixed a bug with using Export Hex and more than 256 bytes per line.
- Fixed issues using on-demand structs inside arrays.
- Fixed color picker issues on the Mac.
- Fixed last two lines of a customized Inspector disappearing in certain cases.
- Fixed background colors not being cleared properly when a template was cleared.
- Fixed a notification during installation about an expired license.
- Now signing the installer using SHA-1.

Version 4.0.1 - June 6th, 2012

- Fixed crash with a custom Auto-Inspector and certain long operations.
- Fixed crash with wordwrap on very long lines.
- Fixed Paste from Hex Text on Mac OS X.
- Fixed clicking on a bookmark, variable, or find result will always select bytes in the editor.

- Fixed editing bookmarks that contain arrays.
- Fixed issues with the Find results when searching for text containing linefeeds.
- Fixed enums and different numeric formats when displaying local variables.
- Fixed adding or removing 010 Editor from the Windows Explorer right-click menu using the Options dialog.
- Fixed the Output tab automatically scrolling down as text is added.
- Fixed the FindNext function should not be selecting data.
- Fixed the FindNext function when used with DeleteBytes in a loop.
- Fixed crash with the HexOperation function on a new file.
- Fixed the Short type specifier in the Goto Bar.
- Fixed allowing hotkeys to be set for Mark Selection Start/End.
- Fixed misplaced cursor when changing the endian of a Unicode file.
- The Goto and Select Range Bars now accept extra whitespace.

Version 4.0 - May 14th, 2012

The following is an overview of the new functionality in version 4.0 of 010 Editor:

- Streamlined interface with a simplified tool bar.
- Easier, more intuitive way to run Templates or Scripts using the File Bar above each editor.
- Added support for word wrap and text files with a huge number of lines.
- Tools such as Find, Replace, Find in Files, Goto, Select Range are now displayed in a bar below each editor for a better workflow.
- Can customize the Auto Inspector using a custom Template.
- Revamped Find tool allows searching for variables in the Template Results.
- When a Find All or Find in Files is done on a text file, all text lines that contain a match are listed.
- Added '*Search > Find Strings*' dialog for listing all strings in a hex file.
- Can visually swap bytes in the hex editor without modifying the underlying data.
- Can restore all open files when 010 Editor is restarted (select 'Restore all open files' on the Startup page).
- Comments can now specify a custom function using '<comment=<function_name>>'.
- Printf function now does type checking and automatically casts parameters.
- Templates can now read from or write to other files when granted special permission.
- Over 45 new functions for Scripts and Templates.

The following is a list of all new features in version 4.0 of 010 Editor:

- **General**
    - Simplified the tool bars and updated some icons.
    - File Bar above each editor provides a more intuitive way to run Templates or Scripts.
    - Can restore all open files when 010 Editor is restarted (select 'Restore all open files' on the Startup page).
    - View menu now shows different options when using a text-based interface or hex-based interface.
    - In the Workspace, can sort the list of recent files by Name or by Time using the right-click menu (sort by Time is the default now).
    - Startup page has been redesigned with resizable sections.
    - Can customize the polynomial and initial value for CRC checksums.
    - Added more options to the '*View > Addresses*' menu.
    - Use '*View > Addresses > Shorts*' for displaying addresses as words (all addresses are divided by 2).
    - Can switch File Interfaces using the Edit As section of the File Bar.
    - Right-click on the Output panel to Copy All, Export Text, or Clear.
    - Can apply Highlighting rules by Shorts.
    - The Welcome dialog allows inputting a license when 010 Editor is in trial mode.
    - Added '-saveall' command line option.

    - Added 'Copy File Path' command to the File Tab right-click menu.
    - Synchronize Scrolling now only scrolls files that are visible.
    - Cannot change a hex-based interface to a text-based interface or vice versa.
    - When opening a file that is already opened, the file is selected before being asked to create a duplicate.
    - Right-click on the application when all tabs are closed for an option to show the Startup Page.
    - Removed comment column for bookmarks (currently unused).
    - 010 Editor configuration file is saved more often.
    - Configuration file now stored in 'Users\<Username>\AppData\Roaming\SweetScape\010 Editor\010Editor.cfg' on Windows.
    - Windows Explorer right-click menu now lists '010 Editor' instead of '010 Editor v3'.
    - Updated help viewer.
- **Text Editor**
    - Added support for text files with huge numbers of lines.
    - Any long text lines over the maximum line length will be split into multiple lines and marked by '-' in the address column.
    - The maximum line length can be set using the Text Editor Options dialog.
    - Clicking after the last line always moves the cursor to the last byte.
    - Hide the Group By, Division Lines, Left Area, Right Area, Line Width options in the View menu for text files.
    - Word Wrap
        - Can automatically wrap lines that are longer than the text editor window with '*View* > *Linefeeds* > Word Wrap' or *Ctrl+;*.
        - Word Wrap can be turned on automatically using '*View* > *Linefeeds* > Initial Wrap State > Auto-detect'.
        - '/wrap' is displayed in the Edit As section of the File Bar above each editor when Word Wrap is turned on.
        - Can wrap on the window edge or a custom column using '*View* > *Linefeeds* > Wrap Width'.
        - Can wrap on letters instead of words using '*View* > *Linefeeds* > Wrap Method'.
        - Word Wrap is supported for files with a large number of lines.
    - Byte-Order Marks
        - Added better support for working with Byte-Order Marks (BOMs).
        - Auto-detect UTF-8 BOMs.
        - Status Bar displays '+B' when a BOM is present.
        - Add or remove BOMs using '*Tools > Convert*'.
        - Can auto-add BOMs to new files using the File Interface Options dialog.
- **Hex Editor**
    - Can visually swap bytes in the hex editor without modifying the underlying data.
    - Bytes can be swapped in groups of 2, 4, 8, etc. (use '*View > Group By*' menu to enable).
    - Display 'LIT<>' in the status bar when swapping bytes is enabled.
    - Set the number of bytes per line in a hex editor using '*View > Line Width*'.
    - Hide the Linefeeds and Tabs/Whitespace options in the View menu for hex files.
- **Find**
    - Find/Replace/Find in Files/Replace in Files now performed from a bar at the bottom of the editor.
    - When a Find All or Find in Files is done on a text file, all text lines that contain a match are listed.
    - Revamped Find/Replace tools with icons for Find/Replace Next/Previous.
    - Added '*Search > Find Strings*' dialog for listing all strings in a hex file.
    - Can search for Variables in the Template Results using the Find Bar.
    - Selecting a Find occurrence in the editor highlights that occurrence in the Find results.
    - Can list all replacements after doing a Replace All (see 'Show All Replacements' in the Find Bar Options).
    - Lock Find/Replace to a selection using 'Lock Selection' in the Find Bar Options.
    - The locked selection will be colored brown and used for Find/Replace until unlocked.
    - Can display the address in the Find results in more formats.
    - Show or hide the size column in the Find results (hidden by default).
    - Ignoring case and match whole word work better when finding extended unicode characters.
    - Find in Files list updates automatically when the files are edited.
    - Can turn off wrapping for find (see Find Bar Advanced Options).

    - Can list all files when doing a Find in Files operation.
    - Can turn off the use of Find Specifiers using the Find Bar Advanced Options.
    - When searching for Unicode, selected Unicode text is automatically copied to the Find Bar when it is opened.
    - Default to Unicode Find type when finding in a Unicode file.
- **Scripts/Templates**
    - Use 'Run Script', 'Run Template' or 'Run on File' sections above each editor for running scripts and templates.
    - Comments can now specify a custom function using '<comment=<function_name>>'.
    - Templates can now read from or write to other files when granted special permission.
    - Permissions can be controlled using the Permissions section of the Options dialog and can be turned off for all files.
    - Support searching in the Template Results (use 'Variable Name'/'Variable Value' in the Find Bar).
    - From the Template Results right-click menu, can either expand all nodes of the selected node (Expand all Children of Node) or expand all nodes in the tree (Expand All Nodes).
    - Right-click on a numeric template variable and choose '*Goto > Goto Address*' or '*Goto > Goto Sector*'.
    - Can override the string displayed in the Name column of the Template Results using '<name="<string>">' or '<name=<function_name>>'.
    - On Windows can right click on a script or template in the Scripts or Templates menu and select '*Edit Script*' or '*Edit Template*'.
    - Updated included scripts to using RequiresFile and GetScriptName functions.
    - Can run a script on no file by selecting '(none)' in the Run on File section of the File Bar.
    - When doing a CSV export or Copy of the Template Results, the Color field will be converted to a hex color instead of being left blank.
    - An integer with <format=binary> will now show all bytes if zero instead of just the first byte.
    - 'L' string constants now support UTF-8.
    - Prevent running a template on a template.
    - Generate a warning if defining a large number of template variables.
    - Removed variable mouse-over brackets on a text file (was being displayed incorrectly).
- **New Functions**
    - Can convert between a binary string and an integer (BinaryStrToInt, IntToBinaryStr).
    - Added functions to perform checksums on arrays (ChecksumAlgArrayStr, ChecksumAlgArrayBytes).
    - Can convert strings between various encodings (ConvertString).
    - Added functions to extract path, extension, or base name from a file name (FileNameGetBase, FileNameGetBaseW, FileNameGetExtension, FileNameGetExtensionW, FileNameGetPath, FileNameGetPathW, FileNameSetExtension, FileNameSetExtensionW).
    - Can search for an open file in the editor (FindOpenFile, FindOpenFileW).
    - Can retrieve environment variables (GetEnv).
    - File attributes for Windows or Unix files can be retrieved or changed (GetFileAttributesUnix, GetFileAttributesWin, SetFileAttributesUnix, SetFileAttributesWin).
    - Retrieve the character set of the current file (GetFileCharSet).
    - Get or set which file interface is being used for the current file (GetFileInterface, SetFileInterface).
    - Retrieve the name of the current script or template being run (GetScriptName, GetScriptNameW, GetScriptFileName, GetScriptFileNameW, GetTemplateName, GetTemplateNameW, GetTemplateFileName, GetTemplateFileNameW).
    - Provided function to get a temporary file name (GetTempFileName).
    - Can get or set the current working directory (GetWorkingDirectory, GetWorkingDirectoryW, SetWorkingDirectory, SetWorkingDirectoryW).
    - Provided function to perform any of the Hex Operations in the Tools menu (HexOperation).
    - Can calculate the length of a null-terminated string in a file (ReadStringLength, ReadWStringLength).
    - Provided function to ensure that a script is being run on a target file (RequiresFile).
    - Can write a message to the status bar (StatusMessage).
    - Convert strings easily to UTF-8 format (StringToUTF8, WStringToUTF8).
    - Can convert characters to upper or lower case (ToLower, ToLowerW, ToUpper, ToUpperW).
    - Can convert from a text column to an address (TextColumnToAddress).
- **Function Updates**
    - Printf function now does type checking and automatically casts parameters.

- Interface type can be specified for the FileOpen and FileNew functions.
- GetFileName returns a UTF-8 string.
- FileOpen will now accept a UTF-8 string.
- FileOpen no longer pops up an error message if the file could not be found.
- Better handling of opening duplicate files with the FileOpen function.
- Can specify a different character set in the StringToWString and WStringToString functions.
- MessageBox supports UTF-8 encoding.
- RunTemplate can be run with no arguments to rerun the Template associated with the current file.
- Added argument to the RunTemplate function to prevent clearing the Output panel.
- SubStr/WSubStr now default to -1 instead of 0.
- Can specify a custom date format in the functions GetCurrentTime, GetCurrentDate, GetCurrentDateTime, StringToDosDate, StringToDosTime, StringToFileTime, StringToOleTime, StringToTimeT, DosDateToString, DosTimeToString, FileTimeToString, OleTimeToString, and TimeTToString.
- Can set the maximum wildcard length in the FindAll, FindFirst, FindInFiles, and ReplaceAll functions.
- Removed HTML tag processing from Printf.

- **Selections**
  - '*Edit > Select Range*' tool now displays a Select Bar at the bottom of the editor.
  - When selecting bytes while the Select Bar is open, the Select Bar will update to display the selection.
  - Right click on an editor and choose '*Select > Mark Selection Start*' or '*Select > Mark Selection End*' to set the selection.
  - Added Save Selection to the editor right-click menu.
  - Save Selection automatically appends the start/size of the selection to the file name.
- **Inspector**
  - Can customize the Auto Inspector (right-click the Inspector and choose 'Customize...').
  - Auto Inspector can be set to use a custom template (by default the 'Inspector.bt' file is included with 010 Editor).
  - Data types can be reordered or deleted from the Auto Inspector using the custom template.
  - Any data type (including Custom Variables) can be added to the Auto Inspector using the custom template.
- **Goto**
  - '*Search > Goto*' tool now displays the Goto Bar below the editor.
  - Use the Goto Bar to jump to a byte, line, sector or short in a file.
- **Options**
  - Can set the maximum line length for text files.
  - Can set the auto-hide time for bars in the application (Find Bar/Replace Bar/Goto Bar/Select Bar/etc.).
  - Added option to remember the last used File Interface, Endian, and Word Wrap for a file.
  - Allow setting the color for the Find Selection Lock.
  - Must specify whether a File Interface is text-based or hex-based when the interface is created.
- **Mac OS X**
  - Now use *Ctrl+G* for Find Next and *Ctrl+Shift+G* for Find Previous on the Mac.
  - Now use *Ctrl+L* for Goto and *Ctrl+Shift+G* for Goto Again on the Mac.
  - Add File Proxy icons to the Application Window (drag to other Windows to open).
  - Removed icons on the menu for Mac.
  - Removed icons on sub-windows for Mac.
  - Fixed order of some OK/Cancel buttons on the Mac.
  - Fixed Windows Clipboard should be called Macintosh Clipboard on the Mac.
- **Hotkeys**
  - *Ctrl+E* now toggles between Big and Little Endian.
  - Word Wrap can be toggled on or off using *Ctrl+;*.
  - Can access Options dialog using Ctrl+, on Windows.
  - Changed close window hotkey to *Ctrl+W*.
  - Changed close all window hotkey to *Ctrl+Alt+W*.
  - Fixed issue with Previous Tab shortcut.
  - Can show or hide the File Bar above each editor using *Ctrl+/*.
- **Bugs**
  - Fixed bug copying large blocks of data to other applications.

- Fixed issues with editing UTF-8 characters.
- Fixed crash when a startup script was missing.
- Fixed constants in the form 0B?h.
- Fixed crash in certain invalid assignments to a struct.
- Fixed bug with extra character when casting from a string to a wstring.
- Fixed bug with switch statement when default was the first statement.
- Fixed a case where a non-terminated string could potentially cause a crash.
- Fixed case where the Close All icon enabled state was not correctly set.
- Fixed case when Close All could cause the Floating Tab Group to be displayed.
- Fixed case where Close All Except This could close the wrong files.
- Fixed bug with assigning to a negative float or double.
- Fixed issue with using return in a function with no return value.
- Fixed OutputPaneCopy/OutputPaneSave missing the last row if no linefeed.
- Fixed case where commented lines were colored incorrectly when jumping through a file.
- Fixed crash in the Inspector with certain rare Unicode strings.
- Fixed bug with FindFirst/FindAll when size=0 and start>0.
- Fixed bug with FindFirst/FindAll and finding Unicode values.
- Fixed bug with FindNext/FindAll using dir=-1.
- Fixed FPrintf(newFile,"%c",0) now works properly.
- Fixed issue with all red text in the Output panel.
- Fixed case when a Hex file is split and then scroll down, the ruler would not be displayed correctly.
- Fixed InputOpenFileName/InputOpenFileNames not using the default file name.
- Fixed AddBookmark can add bookmarks in any of the base types without first running a template.
- Fixed bug with '//' after #define.
- Fixed constants COMPARE_SYNCHRONIZE and COMPARE_SIMPLE were switched.
- Fixed rare crash when running templates and progress issues.
- Fixed crash calling TextReadLine with an empty string.
- Fixed possible crash with MessageBox function when using only two parameters.
- Fixed issues with Copy/Paste from Hex Bytes with Unicode files.
- Fixed DOS linefeeds being transformed to Unix linefeeds when copying to the Find Bar.
- Fixed running a comparison can cause empty an Floating Tab Group to be displayed.
- Fixed a possible crash with an infinite recursion script.
- Fixed issues with character sets in the Find results window.
- Fixed correctly detecting when a file read-only flag is changed outside 010 Editor.
- Fixed Export CSV automatically adds quotes around items containing commas.
- Fixed in split-view mode, the selection could move after editing.
- Increased amount of memory the undo stack is allowed to use.

Version 3.2.2 - August 8th, 2011

- Added support for Mac OS X Lion.
- Fixed losing registration problem on Mac OS X Lion.
- Fixed problem with loading the help file on the Mac.
- Now generate an error when accessing member variables inside a 'size' function.

Version 3.2.1 - July 13th, 2011

- Mac OS X version is now a 32-bit program instead of 64-bit.
- Changed close window hotkey on the Mac to Command+W.
- Fixed Windows context menu in 3rd party applications including Directory Opus.
- Fixed problem with Unicode searches and wildcards.

- Fixed showing the help file when minimized and then a topic is loaded.
- Fixed showing 010 Editor when minimized and the Windows context menu is clicked.
- Fixed inputting certain floats or doubles in the Inspector on Windows.
- Fixed Copy As and then paste to other applications with big-endian Unicode text.
- Fixed date in release notes.

Version 3.2 - June 17th, 2011

The following is an overview of the new functionality in version 3.2 of 010 Editor:

- 010 Editor is now available for Mac OS X.
- On-demand parsing of templates using <size=...> for lower memory usage.
- Can set fonts for the Template Results, Inspector, etc in the Options dialog.
- 'New' on the file menu now lists all types of files available for creation.
- Can pass arguments into scripts or templates from the command line.
- New functions for accessing process information in scripts or templates.
- New functions for easier reading and writing of text files.
- Can export or import a list of scripts or templates from the Options dialog.
- New help viewer with tabs.

The following is a list of all new features in version 3.2 of 010 Editor:

- **General**
  - 010 Editor is now available for Mac OS X.
  - Can set fonts for the Template Results, Inspector, etc in the Options dialog.
  - 'New' on the File menu now lists all types of files available for creation.
  - Each file interface can either use the default text or hex font or a custom font.
  - Separate insert/overwrite state for text and hex files.
  - Can click INS/OVR in the status bar to toggle insert state.
  - Histogram hotkey changed to *Ctrl+Shift+T*.
  - Added option on the Help Menu to view the Release Notes.
  - 010 Editor does a better job converting text data between formats when copying text between files with different character sets.
  - Can set a custom color format for 'Edit > Insert Color' in the Options dialog.
  - Non-ANSI characters now supported in Bookmark names.
  - Release notes added to the manual instead of separate text file.
- **Command Line**
  - Can pass arguments to a script or template using the form: -script:File.1sc:(arg1,arg2)
  - Added ',a' specifier to the '-replace' command to replace in all open files.
  - '-h' now displays the manual command line page instead of a separate dialog.
- **Templates and Scripts**
  - On-demand parsing of templates using <size=*number*> or <size=*function_name*> for lower memory usage.
  - Can export or import a list of scripts or templates from the Options dialog.
  - Added ifdef constant for Windows (_010_WIN) and Mac OS X (_010_MAC).
  - Added ifdef constant for 64-bit versions (_010_64BIT).
  - Non-ANSI characters now supported in comments (use UTF-8).
- **Functions**
  - Added GetFileNameW function to get the wide-string version of the current file name.
  - FileSave function can accept a wide-string or no string at all to save to the current file name.
  - Added ExpandAll function to open all nodes in the Template Results.
  - Added ExportCSV to save the Template Results to a comma-delimited file.
  - GetNumArgs, GetArg, and GetArgW functions can be used to retrieve special arguments passed from the command line.
  - IsNoUIMode function returns true if 010 Editor is being run in '-noui' mode.
  - RunTemplate function now returns the return value from the template that was executed.

    - FileOpen now returns the file index of the file that was opened.
    - Process information can be returned using the functions ProcessGetHeapLocalAddress, ProcessGetHeapModule, ProcessGetHeapSize, ProcessGetHeapStartAddress, ProcessGetNumHeaps, ProcessHeapToLocalAddress, and ProcessLocalToHeapAddress.
    - Added functions to read or write text data in the editor including TextAddressToLine, TextAddressToColumn, TextGetNumLines, TextGetLineSize, TextLineToAddress, TextReadLine, TextReadLineW, TextWriteLine, TextWriteLineW.
- **Bug Fixes**
    - Fixed some foreign characters in names not working in the Register dialog.
    - Fixed the temp directory displaying a short path name.
    - The documentation listed the incorrect return value for the StringToDosDate, StringToDosTime, StringToFileTime, StringToOleTime, and StringToTimeT functions.
    - Fixed bug with multi-line defines and dos-style linefeeds.
    - Fixed bug resetting shortcuts where the interface was not updating properly.
    - Insert Date/Color should work now for Unicode files.
    - Fixed issues with certain input method editors that caused them to not work properly.
    - Now display error message when trying to set linefeeds to 'Auto Detect' while in binary mode.
    - Fixed bug where the Close All icon should be available when the startup tab is selected.
    - Fixed InputSaveFileName function not properly using the default file name.
    - Fixed an undefined variable in a script could be displayed as an error in an associated template.
    - Fixed redraw bug in the Base Converter.
    - Fixed some code examples in the help file.
    - Fixed occasional problem drawing underscores.
    - Fixed template variables sometimes not accessible after calling RunTemplate.
    - Fixed open icon in the tutorial on Windows 7.
    - Fixed Histogram panel initial size on Windows 7.
    - Fixed the size of the Hex Operations dialog on Windows 7.
    - Fixed accessing new file created during a call to RunTemplate.

Version 3.1.3 - November 10th, 2010

- Fixed possible Windows DLL exploit (see https://www.microsoft.com/technet/security/advisory/2269637.mspx).
- Fixed crash using Export CSV with very large datasets.
- Fixed compile errors using DOS style linefeeds after preprocessor statements.
- Fixed crash on comparing large files.
- Fixed occasional problems with button labels in message boxes.
- Fixed some button sizes on Windows 7.
- Now opening .lnk and .url files will load those files directly, not the files to which they link.
- Updates for installing from a CD.

Version 3.1.2 - May 21st, 2010

- Fixed crash with 'Run as administrator' on Windows 7 x64 machines.
- Fixed some constants being incorrectly identified as 'int64' instead of 'int'.

Version 3.1.1 - May 5th, 2010

- Fixed Windows Explorer shell extension on Windows 7 x64 machines.

- Template results now display ASCII strings in the current file character set.
- Fixed error if a script or template included more than 16 files.
- Fixed error using sizeof operator to calculate the size of a struct.
- Fixed error message with -replace command line option when no values found.
- Fixed behavior comparing int -1 to 0xffffffff to be consistent with 3.0.

Version 3.1 - February 16th, 2010

The following is an overview of the new functionality in version 3.1 of 010 Editor:

- Support for the UTF-8 character set.
- Support preprocessor directives #define, #ifdef, #ifndef, #else, etc.
- Support for wstring and wchar_t in Scripts/Templates (unicode strings).
- Use -noui to run 010 Editor without a user interface for batch files.
- Template variables can now have comments (e.g. <comment=""> after a variable).
- New options when defining template variables: <fgcolor=???>, <bgcolor=???>, <open=true|false|suppress>, <hidden=true|false>.
- Can pass arguments into structures.

The following is a list of all new features in version 3.1 of 010 Editor:

- **Editor**
  - Support the UTF-8 character set.
  - Added 'UTF-8' and 'Binary' to the 'Edit As' drop-down list.
  - Added support for Ctrl+Ins(Copy), Shift+Ins(Paste), and Shift+Del(Cut).
  - Shows a current column position arrow in the text editor ruler.
  - Soft hyphen (0xAD) displayed as hyphen instead of space.
- **Command Line**
  - Use -noui to run 010 Editor without a user interface for batch files.
  - Added -nowarnings to disable display of message boxes when using -noui.
  - Can run comparisons from the command line with -compare.
  - Added -exitnoerrors to close 010 Editor only if there are no script or template errors.
- **Histogram**
  - Char column in histogram displays the current file character set.
- **Bookmarks**
  - Toggle Bookmark (Ctrl+F2) makes a bookmark for the whole selection, not just the selected byte.
- **Compare**
  - If exactly two files are open, they will automatically be filled in the Compare dialog when opened.
  - Can run comparisons from the command line.
  - Increased number of internal allowed differences in comparison algorithm.
- **Options**
  - Can hide the splash screen on startup (only when the software is registered).
  - Control the color of the caret.
  - Can change the input-method editor (IME) color.
  - Can control the ruler column marker arrow color.
- **Templates and Scripts**
  - Can display comments for variables using <comment=""> syntax after a variable.
  - Can set the color of a variable using <fgcolor=???> or <bgcolor=???> after a variable (for example, <fgcolor=cBlack, bgcolor=0x803020>).
  - Can use <open=true|false> after a variable to have a variable open by default.
  - Can use <hidden=true|false> after a variable to hide variables.
  - Can use <open=suppress> after a variable to prevent it opening during Expand All.
  - Can pass arguments into structures.
  - Support #ifdef, #ifndef, #define, #undef, #endif, #else, #warning, #error preprocessor

directives.
    - Added support for unicode strings (wstring/wchar_t).
    - Can use L to indicate wide-string constants (e.g. L"dog").
    - Bitfields and enums can now work together.
    - Can now call functions in templates from scripts.
    - Enum values can now be any expression.
    - Better handling of int64 constants (auto-detect).
    - Added current template directory to include path search.
    - Use Ctrl+Left/Right/Enter for better navigation in the template results tree.
    - Double-click on an error or warning takes you to the source code line.
    - Enum list items can be selected on single-click instead of double-click.
    - Support showing local variables inside a struct.
    - Correctly update focus highlight in Variables tab when switching files.
    - Can cast time types to ints or floats.
    - Updating variable coloring rule so colors are properly propagated from parents to children.
    - Exec function is no longer allowed in a template.
- **New Functions**
    - Can run a template from a script (RunTemplate).
    - Can set bookmarks from a script or template (AddBookmark, GetBookmarkName, GetBookmarkPos, GetNumBookmarks, RemoveBookmark).
    - Added new clipboard functions (ClearClipboard, CopyStringToClipboard, GetClipboardString).
    - Get the current time (GetCurrentTime, GetCurrentDate, GetCurrentDateTime).
    - Added an Assert function.
    - Convert a variable such as int or float to bytes (ConvertDataToBytes).
    - Retrieve the current temporary directory (GetTempDirectory).
    - Control the output panel where Printf data is displayed (OutputPaneClear, OutputPaneSave, OutputPaneCopy).
    - Can wait for a certain number of milliseconds (Sleep).
    - New wstring (unicode string) functions: InputWString, ReadWLine, ReadWString, StringToWString, WMemcmp, WMemcpy, WMemset, WriteWString, WStrcat, WStrchr, WStrcmp WStrcpy, WStrDel, WStricmp WStringToString, WStrlen, WStrncmp WStrncpy, WStrnicmp, WStrstr, WSubStr.
    - Can determine if a function exists with 'function_exists' keyword.
    - Exec function has a parameter to wait until execution is finished before returning.
    - FileOpen can execute the template associated with a file.
    - Can access the current structure variable with 'this' keyword.
    - Can access the parent of a structure variable with 'parentof' keyword.
- **General**
    - Installer can warn user if installing a version that will require an upgrade.
    - Register dialog button displays 'Cancel' instead of 'Continue' to prevent confusion.
    - Temp directory can pick up the TEMP system variable.
    - Add Ctrl+Enter shortcut in Find in File results (keeps focus on Output Window).
    - Better handling of multiple versions of the configuration file.
    - Configuration file should be less prone to corruption.
- **Bug Fixes**
    - Fixed templates variable arrays if the size of the array is greater than 2 GB.
    - Fixed bug where a file could display no data after canceling a script or template.
    - Fixed bug drawing the ruler when certain fonts are chosen.
    - Now correctly report physical disk size on Windows XP or higher.
    - Improved error message when using shift operators.
    - Improved error message when defining a structure twice with typedef.
    - Fixed empty structure warning when using bitfields and no padding.
    - Fixed bug with drawing Fg: in the template results.
    - Fixed bug selecting bytes in Unicode.
    - Fixed bug extending a selection with Shift+click.
    - Fixed reading bookmarks containing enums.
    - Fixed opening files containing '@'.
    - Fixed importing a hex text containing very long lines.
    - Fixed using bitfields mixing named and unnamed variable.
    - Fixed auto-detect of decimal import text.

- Fixed bug using post increment during array access.
- Fixed printfs sometimes being colored red after an error.
- Fixed syntax highlighting issue with quotes.
- Fixed possible crash with the FileClose function.
- Fixed error message when defining struct variables incorrectly.
- Fixed a typo in the welcome dialog.
- Fixed issue with the syntax highlighting menu.
- Fixed an issue with double-clicking on find-in-file results.
- Fixed Variables Tab not being updated correctly after running a script.
- Fixed issue working with files that are deleted.
- Fixed issue passing strings to functions which are then passed to other functions.
- Fixed selecting issue by dragging over the last line in the text editor.
- Fixed negative enums.
- Fix for optimizing warning on struct that just contains a 'string' variable.

Version 3.0.6 - October 26th, 2009

- Fixed importing hex text with very long lines.
- Fixed templates which mixed bitfields and regular variables.
- Fixed Printf with '%%'.
- Fixed issues on Windows 7.

Version 3.0.5 - April 13th, 2009

- Now support Input Method Editors (IMEs) for multi-byte character sets.
- Fixed bug with Find dialog and multi-byte character sets.
- Fixed problem with backspace and multi-byte character sets.
- Fixed possible buffer overruns running templates and scripts (as reported by Bach Khoa Internetwork Security (Bkis) - http://security.bkis.vn/).
- Fixed problems with the FindFirst and ReplaceAll functions.
- Scripts and templates now support '0h' as well as '00h'.
- Fixed bug in displaying warning message about optimized arrays.
- Fixed crash using Overwrite Bytes on a large drive.
- Fixed possible crash when closing a file after a comparison.
- Fixed not being able to open certain logical and physical drives.
- Fixed bookmarks being deleted when overwriting a block of data.
- Fixed being able to open Unicode filenames from the Windows Explorer right-click menu.

Version 3.0.4 - February 6th, 2009

- Fixed issues with displaying Japanese and Chinese character sets.
- Updated character handling engine for non-ASCII character sets.
- Fixed restoring the maximized state of 010 Editor on restart.
- Fixed possible crash when opening files from Windows Explorer right-click menu.
- Fixed issues with properly refreshing the Inspector values.
- Fixed crash in Replace dialog.
- Fixed crash when setting certain Group By values.
- Fixed issue with certain foreign characters in the Register dialog.

- Fixed typo in the calculator.

Version 3.0.3 - October 17th, 2008

- Improved memory handling for running very large templates.
- Fixed possible crash with syntax highlighting.
- Fixed possible crash running a script with a long line.
- Fixed some Unicode rendering issues in the hex editor.
- Fixed crash with exporting a text area from a Script in certain cases.
- Fixed the find in selection not working properly in certain cases.
- Fixed disabled/enabled problems with buttons in the find dialog.
- Fixed a problem with the ',' constant in scripts.
- Fixed error message for attempting incorrect operations on structs.
- Fixed possible crash using the StrDel function.
- Fixed typo in script cancelled error message.
- Can now copying Unicode data to the clipboard correctly.

Version 3.0.2 - August 28th, 2008

- Automatically hide empty floating tab group on startup.
- Fixed crash with inspector when displaying certain characters.
- Fixed focus issues when loading files via the Windows explorer right-click menu.
- Fixed potential crash with -exit command line option.
- Bug fix for Keep File Time functionality.
- Fixed crash closing deleted files in certain cases.
- Allow custom read/write functions to work with equivalent types.

Version 3.0.1 - July 16th, 2008

- **Bug Fixes**
  - Fixed periodic hangs when using the Explorer tab to browse a network drive.
  - Fixed international character set issues.
  - Fixed some keyboard keys not working on international keyboards.
  - Fixed possible crash with Copy As Hex Bytes.
  - Fixed replace up missing some occurrences in certain cases.
  - Fixed visual issues editing Unicode hex data.
  - Fixed using the clipboard when some specialized clipboard managers were installed.
  - Fixed progress update on running some scripts.
  - Fixed issues with the scope of local variables between functions and structs.
  - Fixed problem with empty statements inside a switch statement.
  - Fixed a reporting issue with Byte by Byte comparisons.
  - Fixed being able to run the Calculator on other files in the interface.
  - Fixed occasional crash when opening files when all windows were closed.
  - Fixed display glitch with Output tab showing red text.
  - Fixed tool bar undocking issues.
- **General**
  - Each Tool Bar can now be resized to be smaller than its contents.
  - Visual changes on Vista machines for more consistency.

Version 3.0 - May 19th, 2008

The following is an overview of the new functionality in version 3.0 of 010 Editor:

- Includes a text editor with syntax highlighting.
- Improved interface with new icons.
- Improved file tabs with close buttons that can be dragged to new positions.
- Can have multiple scripts/templates open at the same time.
- Find/replace, goto, etc. can be applied to templates or scripts.
- Full support for Unicode file names and strings in the application.
- Scripts and Templates now stored in "My Documents\SweetScape" directory.
- Official support for Windows 98/Me/NT has been dropped.
- A whole range of other improvements.

The following is a list of all new features in version 3.0 of 010 Editor:

- **Text Editor**
  - Can now edit text, Unicode, or EBCDIC files.
  - Basic support for syntax highlighting for C/C++ files.
  - Support indenting or unindenting text with Tab or Shift+Tab.
  - Added insert date/time and insert color commands.
- **Hex Editor**
  - Improved visual style for hex editing.
  - Can display addresses in sectors.
- **File Tabs**
  - Improved tab-based interface with close buttons.
  - Added a floating tab group.
  - Can drag and drop file tabs (even between the floating tab group).
  - Able to have multiple tab groups laid out horizontally or vertically.
- **Startup Page**
  - Optional startup page shows recent files, latest application news, and tips.
- **File Interfaces**
  - Added 'Edit As' combo box to switch between editing text, hex, C++, etc. files.
  - Added Ctrl+H shortcut to toggle between text and hex files.
  - Use Ctrl++ or Ctrl+- to enlarge or shrink fonts.
  - Can set status bar and ruler display formats for different file interfaces.
- **Find/Replace**
  - Added Replace in Files command.
  - Find dialog automatically copies selected bytes to the Value field.
  - Improved status bar for doing long replace operations.
  - Find results update when bytes inserted/deleted.
- **Goto**
  - Can use goto dialog to jump to an address, line or a sector.
  - Use ',a', ',l' or ',s' options in Goto combo box for address, line or sector.
- **Bookmarks**
  - Added a Toggle Bookmark command (can be used to add quick bookmarks).
  - Bookmark positions properly update when bytes inserted/deleted.
- **Printing**
  - Enhanced Print Preview dialog.
  - Printing now works with text files (including Templates and Scripts).
- **Tools**
  - Improved calculator with input buttons for performing quick calculations using the mouse.
  - Improved conversion utility can be used to convert character sets and/or linefeeds.
  - Improved checksum tool can treat data as ushorts, uints, or uint64s.
  - Improved histogram tool can treat data as other data types.
- **Workspace**

    - Enhanced the 'Explorer' tab of the Workspace.
- **Inspector**
    - Added Unicode string to the Inspector.
    - Moved list of available functions to the 'Functions' tab of the Inspector.
- **Scripts and Templates**
    - Can have more than one script or template open at a time.
    - Find, print, other operations can all be applied to a script or template.
    - Select which script or template to run using drop-down list in the Tool Bar.
    - Results from Printf now displayed in 'Output' tab in the Output panel.
    - Code Editor has been removed and replaced with a Floating Tab Group.
    - Can display local variables in the template results.
    - Add RequiresVersion, ReadInt64, ReadUInt64, WriteInt64, WriteUInt64 functions.
    - Variables defined in script are now displayed in the 'Variables' tab of the Inspector.
    - List of functions now displayed in the 'Functions' tab of the Inspector.
    - Removed size limitation for scripts and templates.
    - Scripts now stored in "My Documents\SweetScape\010 Scripts" directory.
    - Templates now stored in "My Documents\SweetScape\010 Templates" directory.
    - Added default import byte to 'ImportFile' function.
- **Register Dialog**
    - Improved register dialog lists when Support/Maintenance expires.
    - Can look up forgotten passwords.
    - Can remove license from the current machine.
- **General**
    - Added tutorial for using Binary Templates.
    - Updated some hotkeys for various tasks.
    - Show current character set, linefeeds, and tabs in the status bar.
    - Synchronized Scrolling now synchronizes scrolling horizontally and vertically.
    - Added Overwrite File and Overwrite Bytes commands.
    - Many dialogs have an expandable 'Options' section.
    - Improved New and Open tool buttons with drop-down list.
    - Full support for Unicode file names and strings in the application.
    - Can use 'File > Revert/Refresh' to update processes or drives.
    - Added -reset and -resetall command line options to reset the interface.
    - Official support for Windows 98/Me/NT has been dropped.
- **Options**
    - Can control the mouse wheel scroll rate.
    - Can control the directories where Scripts and Templates are stored.
- **Bug Fixes**
    - Fixed problems with different DPI settings.
    - Fixed problem imported certain base64 files.
    - Fixed occasional crash on shutdown.
    - Fixed disappearing columns on some multi-monitor systems.
    - Fixed crash when deleting multiple files that were open in 010 Editor.

Version 2.1.4 - February 22nd, 2008

- Bug fix with assigning to dates in a script.
- Bug fix with importing long lines.
- Bug fix with memory leak in FPrintf.
- Updated licensing to support version 3 licenses.

Version 2.1.3 - April 5th, 2007

- Workaround for startup problems caused by faulty Microsoft hotfix.

Version 2.1.2 - March 10th, 2007

- Bug fix for 'FindAll' function.

Version 2.1.1 - March 9th, 2007

- Added support for Windows Vista.
- Fixed bug with string compare in scripts and templates.

Version 2.1 - March 5th, 2007

- **General**
  - Improved right-click menu in Windows explorer.
  - Updated some icons.
- **Import/Export**
  - Added ability to export as HTML, RTF, or a text area.
  - Can import or export Base64 and uuencoded data.
  - Can use 'Copy As' or 'Paste from' for many different types of data.
- **Tab Menu**
  - Added right-click menu to file Tabs.
  - Can middle-click on a tab to close a file.
- **Command Line**
  - Can Replace a string or set of bytes from the command line.
  - Added ability to save or close a file from the command line.
- **Scripts and Templates**
  - Improved bitfield mode without padding (see BitfieldDisablePadding).
  - Added 'Always on Top' toggle for the Code Editor (from right-click menu).
  - Check to reload script/template if modified by an external program.
- **Editor**
  - Now able to input characters more than 0x80 into the character area.
  - Added option to remove ':' separators in file addresses.
- **New Functions**
  - BitfieldEnablePadding/BitfieldDisablePadding for bit streaming.
  - Function for locating files in a directory (FindFiles).
  - Added directory manipulation functions (DirectoryExists, MakeDir).
  - Added function to convert an enum to a string (EnumToString).
  - Functions to convert Dates to strings and vise versa.
  - Exit function can be used to return errorlevel to a batch file.
- **Bug Fixes**
  - Fixed problem exporting Intel-Hex 16-bit files with custom addresses.
  - Fixed printf functions occasionally detecting the wrong number of arguments.
  - Importing files now properly imports last line if it does not have a carriage return.
  - Fixed sign of variables when doing divisions.
  - Fixed template hints occasionally not displaying correctly.
  - Fixed bug in FileSave function saving to a different filename.

Version 2.0.3 - November 28th, 2006

- Fixed bug with FileSave function.
- Fixed bug with allowing more than one instance.
- Fixed problem with exporting hex text and long lines.
- Now use Exit function to set global errorlevel return value.
- Allow bookmarks on consecutive bytes.
- Fixed bug exporting data from disk or process.

Version 2.0.2 - June 9th, 2005

- Fixed problem in help index with function names not jumping directly to function help.
- Fixed using copy and paste when editing a template variable.
- Fixed problems using Printf on some strings.
- Fixed bugs with custom read functions and forward structs.
- Added uninstall survey.

Version 2.0.1 - April 28th, 2005

- Fixed issues with custom read/write functions.
- Fixed issues with minimize/maximize behaviour.
- Fixed issues with time data formats.
- Bug fixes with canceling operations, resizing, window order, and divisions.

Version 2.0 - March 30th, 2005

The following is an overview of the new functionality in version 2.0 of 010 Editor:

- Added hard drive editing for logical and physical drives.
- Added editing of system processes.
- More powerful interface for viewing template results (results displayed in panel below each hex editor, mouse-over hints, reverse lookup for template variables).
- More powerful scripts/templates syntax (custom functions, custom data types, more standard keywords supported, union support, include support, 44 new functions).
- New Windows XP style.
- Many improvements in tools (new Find in Files tool).
- A whole range of other improvements.

The following is a list of all new features in version 2.0 of 010 Editor:

- **Hard Drive Editing**
  - Open entire logical or physical hard drives using 'File > Open Drive'.
  - Make disk images for drive using 'File > Save As'.
  - Jump to next or previous drive sector with Alt+Down, Alt+Up.
  - Get properties of the drive using 'Edit > Properties'.
  - Open hard drives from the command line.
- **Process Editing**

    - Open processes using 'File > Open Process'.
    - Select which heaps or modules to open.
    - Make image of process using 'File > Save As'.
    - Get properties of the process using 'Edit > Properties'.
    - Open processes from the command line.
    - List of current heaps displayed in 'Process' tab of the Output Window.
- **Templates/Scripts**
    - Templates/Scripts Interface
        - Templates results can now be displayed in the 'Template Results' panel below each hex editor.
        - Application contains links to an online repository for scripts and templates.
        - Display hint for template variable when the cursor is over bytes in the hex editor.
        - New 'Jump to Template Variable' command to lookup a template variable from a file address.
        - Scripts and templates can be opened by double-clicking them in Windows Explorer.
        - Scripts and templates can be run from the command line.
        - Can mark scripts to be run on startup, shutdown, or when certain files are opened.
        - Speed improvements for scripts and templates.
        - Added default shortcut for saving a script or template.
        - Context sensitive help in the Code Editor.
    - Templates/Scripts Syntax
        - Define your own custom functions.
        - Use '#include' keyword to include files.
        - Support for switch, case, break, and continue keywords.
        - Support for unions.
        - Default custom variables using the syntax <read=[functionname], write=[functionname]>.
        - Control whether arrays of structures are optimized with <optimize=true|false>.
        - Specify display format for variable using the syntax <format=hex|decimal|octal|binary>.
        - sizeof now works properly on simple structures.
        - Special new keywords 'exists' and 'startof' for template variables.
        - Array initializers work properly (e.g. int a[3] = {1, 2, 3};).
        - No need to use 'return' when doing simple expressions in the calculator.
        - Changed how scoping works in structs - automatically look up a level.
        - 44 new functions: Checksum, ChecksumAlgStr, ChecksumAlgBytes, Compare, ConvertASCIIToUNICODE, ConvertASCIIToUNICODEW, ConvertUNICODEToASCII, ConvertUNICODEToASCIIW, DeleteFile, Exit, ExportFile, FindAll, FindFirst, FindInFiles, FindNext, FPrintf, GetBytesPerLine, GetReadOnly, GetSectorSize, Histogram, ImportFile, InputOpenFileName, InputOpenFileNames, InputSaveFileName, InsertFile, IsBigEndian, IsDrive, IsEditorFocused, IsLittleEndian, IsLogicalDrive, IsModified, IsPhysicalDrive, IsProcess, OpenLogicalDrive, OpenPhysicalDrive, OpenProcessById, OpenProcessByName, RenameFile, ReplaceAll, SetReadOnly, SScanf, StrDel, SubStr, Terminate.
    - New Templates/Scripts
        - New default scripts for splitting or joining binary files.
        - New default template for parsing a WAV sound file.
- **General**
    - New Windows XP Style.
    - New 'File > Special > Save Selection' command to save selected bytes.
    - Show selection start and size in the status bar.
    - Click on position or size in status bar brings up an edit dialog.
    - Many tables have an 'Export CSV' option on right-click menu to write a comma-delimited file.
    - Can change read-only plus other flags in file properties.
- **Hex Editor**
    - Cursor size changes in Insert/Overwrite mode.
    - UNICODE character set support.
    - International character sets support.
    - Can split the hex editor into two parts using 'Window > Split Window' or button above scroll bar.
    - Scrolling can be synchronized between windows with 'Window > Synchronize Scrolling'.

        - Division lines can be used to indicate blocks of data.
        - Sector lines are drawn to indicate sectors on a hard drive.
        - Addresses can be displayed as octal format or as a line number.
    - **Tools**
        - Find in Files tool
            - Can recursively search a directory or all open files for a set of bytes.
            - Display find in files results in 'Find in Files' tab of Output Window.
            - Can expand or hide results for each file.
        - Find
            - Can search with wildcards '*' and '?'.
            - Allow multiple find ranges to color the same file.
            - Find works with UNICODE.
        - Highlights
            - Allow multiple highlights to be applied at the same time.
            - Assign different colors to highlights.
        - Comparison
            - Can limit which bytes are compared in a file (use to compare two regions in the same file).
            - Comparison results are sortable by clicking on the table headings.
            - Add color indicator box to the type column.
            - Can enable synchronized scrolling after running a comparison.
        - Base Converter
            - Support for Float, Double, ASCII strings, EBCDIC strings, UNICODE strings in base converter.
        - Checksum
            - Can exclude a set of bytes in the file from the checksum.
            - Can display the checksum results in decimal format.
    - **Inspector**
        - String type in inspector.
        - Display start address as local (offset from parent) from right-click menu.
        - Template variable name can be 'Type + Name' or just 'Name' (use Column Display Format).
    - **Bookmarks**
        - Bookmarks can now use custom data types defined in Templates.
        - Bookmarks can be set to move when the cursor changes position.
    - **Importing**
        - Do checksum when importing an Intel-hex file.
        - In import file dialog box, can set file type as 'All Supported Import Types'.
        - Can import multiple files at the same time.
        - Support for reading and writing Intel Hex files that use word-based addresses.
    - **Options**
        - Specify color of right area, separator lines, sector lines, and variable highlight.
        - Specify additional include directories.
        - Specify minimum number of digits in address.
        - Option for turning on/off mouse over and hints.
        - Option for adding 010 Editor to the Windows Explorer right-click menu.
    - **Command line**
        - Can open a drive or process from the command line using '-drive:' or '-process:'.
        - Run a script or a template from the command line using '-script:' or '-template:'.
        - Mark a file as readonly using '-readonly'.
        - Exit the application using '-exit'.
        - Can use wildcards when opening files or importing files on the command line.
    - **Help**
        - New improved help file.
    - **Bugs**
        - Bug with window opening at zero height (in special cases).
        - Bug with a slow-down when defining large arrays in scripts.
        - Bug with divisions and the resulting types in scripts (in special cases).
        - Bug with replace all never finishing on certain replaces.

Copyright © 2003-2019 SweetScape Software

- Other minor bug fixes.

Version 1.3.2 - April 19th, 2004

- Added functions ConvertASCIIToEBCDIC, ConvertEBCDICToASCII
- Added functions BitfieldLeftToRight, BitfieldRightToLeft
- Bug fixes with Printf, Operations, and local arrays in Templates

Version 1.3.1 - April 6th, 2004

- Changed header for templates and scripts
- Bug fixes for the system path, and the Inspector

Version 1.3 - April 1st, 2004

- **Templates**
  - Added bitfield support to templates.
  - Added DisplayFormatBinary and DisplayFormatOctal functions.
  - Can disable warnings under 'Tools > Options > Code Editor'.
- **General**
  - Now use the new XP file dialog boxes.
- **Clipboard**
  - Can use 'Paste Special' command to paste in different formats.
- **Bug fixes**
  - Fixed problems with workspace, filling bytes, running templates, pasting large blocks, and selecting bytes on NT machines.

Version 1.2 - Jan 24th, 2004

- **Inspector and Output Windows**
  - Added 'Copy Column', 'Copy Row', and 'Copy Table' to right-click menu.
  - Added 'Column Display Format' to right-click menu - set the format to hex or decimal.
- **Scripts**
  - Fixed scripts to work better when writing out large files.
  - Any of the 'Write' functions now automatically expand the file size when writing past the end of the file .
  - A single variable (i.e int x) defined in a template can be accessed as x[0].
  - Added 'DisableUndo' and 'EnableUndo' to turn on or off undo - speeds up script when writing large files.
  - The 'FileNew' function now returns the file number of the created file.
- **Import/Export**
  - Added the ability to import or export 'Decimal Text'.
- **Bug Fixes**
  - Fixed 'Invalid Pointer Operation' bug in the Code Editor.
  - Fixed a bug with the 'GetFileNum' function.

    - A few other minor bug fixes with hotkeys and Templates.

Version 1.1.1 - Dec 19th, 2003

- Minor bug fixes with bookmarks and the inspector.
- Added Expand All option to template right-click menu.

Version 1.1 - Nov 1st, 2003

- **Templates**
    - Support for enums (e.g. 'enum <ushort> MYENUM { COMP_1, COMP_2=5, COMP_3 } var1;').
    - Enums variables are displayed with a drop-down list in the Inspector.
    - Forward declared and recursive structs work properly.
    - Defining local variables inside structs works better (proper scope).
    - Zero-length arrays generate no variable (but do generate a warning).
    - Can specify hex or decimal display in Inspector with DisplayFormatHex() or DisplayFormatDecimal() functions.
    - Zip template now contains an example of enums.
- **Installer**
    - 010 Editor can be added to the system path automatically.
    - Can automatically associate with Intel Hex or Motorola S-Record files.
- **Explorer**
    - Can drag-and-drop files from Windows Explorer to open them.
    - Drag-and-drop for Intel Hex or Motorola S-Record files will automatically import them.
- **Importing**
    - Added default import byte under General Options (used for Intel Hex files).
- **Comparison**
    - Improved comparison algorithm.
- **Bug fixes**
    - Minor bug fixes with the Code Editor.

Version 1.0.1 - Sept 26th, 2003

- Minor bug fixes with printing and inspector

Version 1.0 - Sept 16th, 2003

- Initial Release

Related Topics:

010 Editor v10.0 Manual - Windows Edition
Copyright © 2003-2019 SweetScape Software - www.sweetscape.com