Hercules (Hyperion)

The emulator needed to run IBM mainframe bits is called “Hercules” and has gone through a LOT of evolution through the many years of its existence.

The primary maintainer of the current version is David Trout (a.k.a. “Fish”) of SoftDev Labs.

His code is excellent and he maintains the current version of the emulator on Github under the SDL-Hercules-390 project.

David describes the build process on the SDL web page Hercules Build Instructions but as easy as it looks, it's not for all the reasons described here. I've annotated his work for clarity (and reality) of what it takes to get this built properly under Windows - and to be clear, the complexity is in getting the LIBRARIES working properly since his instructions point to builds of utility libraries that are over 10 years old.

Because we're not concerned with the older Visual Studio implementations, I'm only concerned with my build environment VS2022.

Also - because the use of the IDE actually gets in the way of a fast build - I am limiting my remediation instructions to the command prompt **`makefile.bat`** build.

> **SPECIAL NOTE**: The build of Hercules requires **WIN32.MAK** which is a deprecated, but available feature in VS2022

(for the original Visual Studio 2008 instructions, click here which will take you to the SoftDevLabs web site.)

Hercules (Hyperion) Windows Build Instructions

(Visual Studio 2022)

Introduction

> EVERYTHING in this document assumes you are running Windows 10 or Windows 11

This document provides instructions on how to build the Windows “MSVC” version of SDL Hercules 4.x Hyperion. It parallels David Trout's version of the document but points out the “reality” of what is needed to setup for an updated build of Hercules.

SDL did a great job of constructing the build process and fully utilizing the MSBUILD machinery - it's *textbook* … The challenges were in doing a proper **complete** build without the use of libraries which he provided.

Those libraries were built in 2008 and because I don't want and don't have the sources for any libraries that old (I use the same library builds across all projects), the fun starts when it comes time to update the dependent libraries (as of this writing):

Dependencies

Significant functionality is provided by these three *optional* libraries. But the majority of the features I will require depend upon them… so for this process, they are *required*.

- ZLIB (version 1.2.11) is a very lightweight and full-featured compression library created by Greg Roelofs and maintained by Mark Adler
- PCRE FTP ***P**erl **C**ompatible **R**egular **E**xpressions* is a very sophisticated regular expression library. **PCRE support is required in order to build the *Hercules Automated Operator* functionality.** The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5.

  There are two major versions of the PCRE library:

  - The current version, PCRE2, first released in 2015, is now at version 10.35
  - The older, but still widely deployed PCRE library, originally released in 1997, is at version 8.44. Its API and feature set are stable—future releases will be for bugfixes only. Any new features will be added to PCRE2, and not to the PCRE 8.x series. The current version is 8.44 This build example uses PCRE 8.44.
- BZIP2 The bzip2 file compression program was developed by Julian Seward and launched on the 18th of July in 1996. It has remained an open source program, available to all for free, for over twenty two years now. The last stable release was seven years ago. The version 1.0.6 was released on the 20th of September in 2010. bzip2 compression program is based on Burrows–Wheeler algorithm. The current version is 1.0.6.

> THE PCRE MODULE: creates complexity in revising the build because SDL's instruction conveniently include links to binaries which are very old and refer to the *UNIX* names of `pcre3.(lib|dll)` which is confusing to anyone trying to build the libraries today. More on this later …

To build the Windows MSVC version of Hercules you need to first download and install Microsoft's free (Free, fully-featured IDE for students, open-source and individual developers) Visual Studio Community 2019 product.

To build the DEPENDENCIES you will need to Download and Install CMAKE.

Summary of Steps Involved

The overall setup of the build environment indicated above is straightforward but incomlete. SDL indicates the 5 steps necessary for Hercules and assumes that you use their pre-built libraries. Before doing anything - it's important to have a proper build structure ready.

Choose your build location and GIT/Clone the working directories:

```
git clone https://github.com/SDL-Hercules-390/hyperion.git
```

The resulting development tree (after following all of the subsequent steps) will look like this:

User Terminal

```
+---autoconf +---crypto | +---include | \---lib +---decNumber | +---include | \---lib +---html | +---images | \---include +---m4 +---man +---msvc.* [SDL Old Library Build Outputs] +---msvc.makefile.includes [SDL Original Makefile Includes] +---msvc.makefile.sz.includes ["Current" Library Makefile Includes] +---msvc.sz.* ["Current" Library build outputs] +---readme | \---images +---scripts +---SoftFloat | +---doc | +---include | \---lib +---telnet | +---include | \---lib +---tests +---util \---winbuild <-- It is highly recommended that you use this. | It is the default, and makes the build process | less complex to diagnose and maintain. | +---bzip2 [SDL Libraries] | +---Debug | \---x64 | \---Debug +---bzip2.sz ["Current" Libraries] | +---Debug | +---include | \---x64 | \---Debug +---pcre [SDL Libraries] | +---bin | +---include | +---lib | \---x64 | +---bin | +---include | \---lib +---pcre.sz ["Current" Libraries] | +---bin | +---include | +---lib | \---x64 | +---bin | +---include | \---lib +---zlib [SDL Libraries] | +---Debug | | +---include | | \---lib | +---include | +---lib | \---x64 | +---Debug | | +---include | | \---lib | +---include | \---lib \---zlib.sz ["Current" Libraries] +---Debug | +---include | \---lib +---include +---lib \---x64 +---Debug | +---include | \---lib +---include \---lib
```

Because we are building not only Hercules, but the dependencies as well the more precise required steps are:

1. Download and install **Visual Studio 2022**.
2. Define the `INCLUDE` and `VS170COMNTOOLS` environment variables, and fix Visual Studio's “**Default** Property Sheets”.
3. Download/UnZip/Position the following pre-built libraries from SDL so the original build functionality can be adequately tested using the defaults:
   1. VC2008/SDL ZLIB
   2. VC2008/SDL BZIP2
   3. VC2008/SDL PCRE

If you intend to build the “Current Libraries” - these additional steps will be required:

- Copy the folder structure for all three libraries *(we assume the defaults of `winbuild` noted above)*
- Copy the makefile structure and give it a proper identifier *(we are using `sz` in this context)*

User Terminal

```
:: Change to the directory root of the GIT cloned repository CD [hyperion_root] :: Copy the STRUCTURE of the three libraries XCOPY ".\winbuild\bzip2" ".\winbuild\bzip2.sz" /T /E XCOPY ".\winbuild\pcre" ".\winbuild\pcre.sz" /T /E XCOPY ".\winbuild\zlib" ".\winbuild\zlib.sz" /T /E :: Clone the MAKEFILE structures XCOPY ".\makefile.msvc" ".\makefile.sz.msvc" XCOPY ".\makefile-dllmod.msvc" ".\makefile-dllmod.sz.msvc" :: EDIT the Cloned MSVC makefiles to reflect the parallel structure NOTEPAD ".\makefile.sz.msvc" > Replace this line: !include makefile-dllmod.msvc > With this line: !include makefile-dllmod.sz.msvc > Save NOTEPAD ".\makefile-dllmod.sz.msvc" > Replace this line: INCDIR = msvc.makefile.includes > With this line: INCDIR = msvc.makefile.sz.includes > Save :: CLONE the original makefiles XCOPY ".\msvc.makefile.includes\*.*" ".\msvc.makefile.sz.includes\*.*"
```

If you decide to locate those packages OUTSIDE of the build tree, you may indicate their proper location through the use of environment variables:

- `SET ZLIB_DIR=<UnQuotedDirectoryLocation>`
- `SET BZIP2_DIR=<UnQuotedDirectoryLocation>`
- `SET PCRE_DIR=<UnQuotedDirectoryLocation>`

> It is **important** to note that the subordinate structures must match the hierarchy indicated above.

Detailed Activity Steps

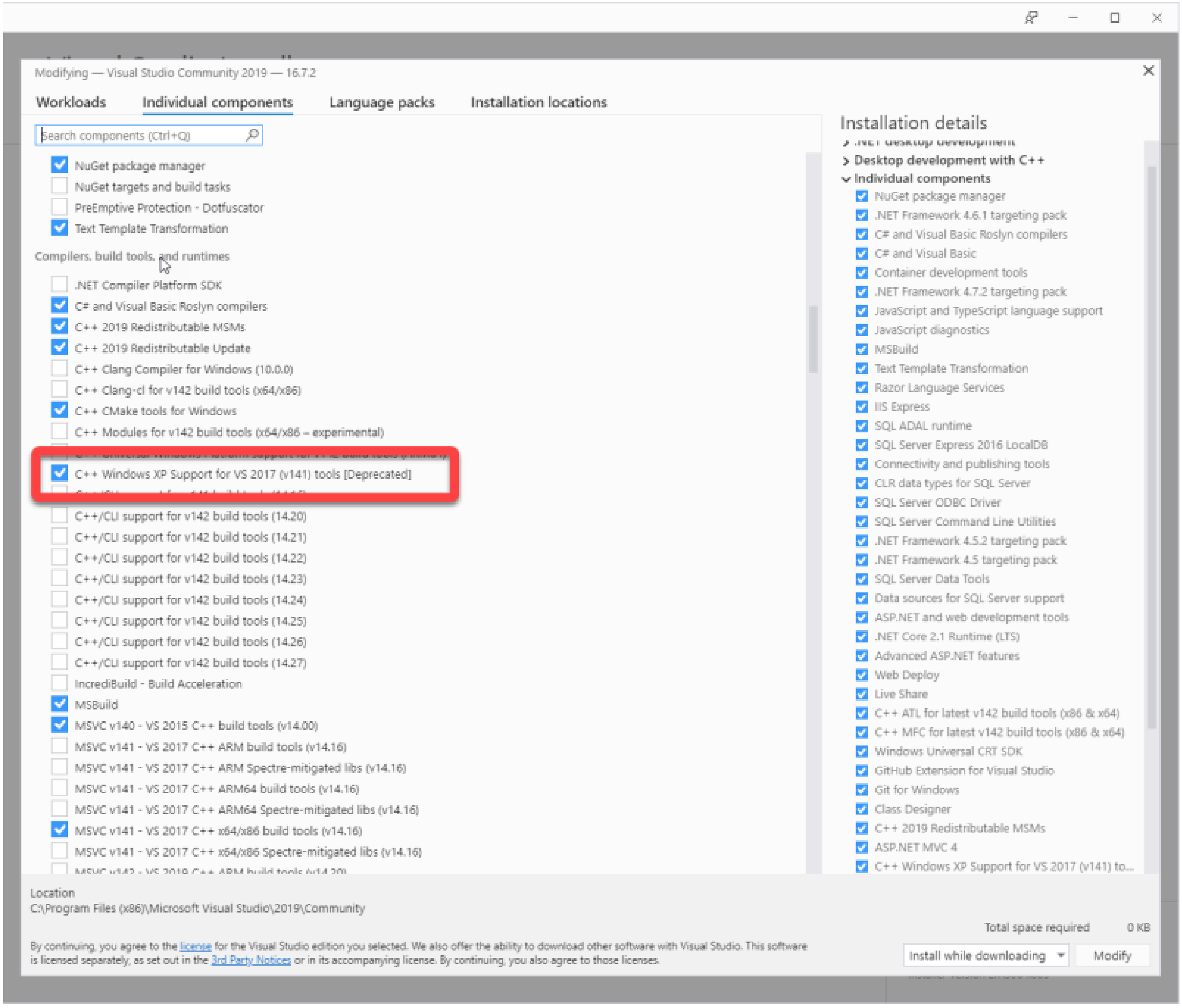
1. Download and Install Visual Studio

Click the download button for the “Community” edition from the Visual Studio download web page using the above link to download a small installer stub. Run the installer and select which components you wish to install, and then let the installer install your selected components.

The install takes quite a while to finish, so get yourself a cup of coffee while you wait.

IMPORTANT!

> You must select the “C++ Windows XP Support for VS 2017 (v141) tools [Deprecated]” option!


Modifying — Visual Studio Community 2019 — 16.7.2

- [✓] NuGet package manager
- [ ] NuGet targets and build tasks
- [ ] PreEmptive Protection - Dotfuscator
- [✓] Text Template Transformation

Compilers, build tools, and runtimes

- [ ] .NET Compiler Platform SDK
- [✓] C# and Visual Basic Roslyn compilers
- [✓] C++ 2019 Redistributable MSMs
- [✓] C++ 2019 Redistributable Update
- [ ] C++ Clang Compiler for Windows (10.0.0)
- [ ] C++ Clang-cl for v142 build tools (x64/x86)
- [✓] C++ CMake tools for Windows
- [ ] C++ Modules for v142 build tools (x64/x86 – experimental)
- [✓] C++ Windows XP Support for VS 2017 (v141) tools [Deprecated]
- [ ] C++/CLI support for v142 build tools (14.20)
- [ ] C++/CLI support for v142 build tools (14.21)
- [ ] C++/CLI support for v142 build tools (14.22)
- [ ] C++/CLI support for v142 build tools (14.23)
- [ ] C++/CLI support for v142 build tools (14.24)
- [ ] C++/CLI support for v142 build tools (14.25)
- [ ] C++/CLI support for v142 build tools (14.26)
- [ ] C++/CLI support for v142 build tools (14.27)
- [ ] IncrediBuild - Build Acceleration
- [✓] MSBuild
- [✓] MSVC v140 - VS 2015 C++ build tools (v14.00)
- [ ] MSVC v141 - VS 2017 C++ ARM build tools (v14.16)
- [ ] MSVC v141 - VS 2017 C++ ARM Spectre-mitigated libs (v14.16)
- [ ] MSVC v141 - VS 2017 C++ ARM64 build tools (v14.16)
- [ ] MSVC v141 - VS 2017 C++ ARM64 Spectre-mitigated libs (v14.16)
- [✓] MSVC v141 - VS 2017 C++ x64/x86 build tools (v14.16)
- [ ] MSVC v141 - VS 2017 C++ x64/x86 Spectre-mitigated libs (v14.16)

Installation details

- .NET desktop development
- Desktop development with C++
- Individual components
  - [✓] NuGet package manager
  - [✓] .NET Framework 4.6.1 targeting pack
  - [✓] C# and Visual Basic Roslyn compilers
  - [✓] C# and Visual Basic
  - [✓] Container development tools
  - [✓] .NET Framework 4.7.2 targeting pack
  - [✓] JavaScript and TypeScript language support
  - [✓] JavaScript diagnostics
  - [✓] MSBuild
  - [✓] Text Template Transformation
  - [✓] Razor Language Services
  - [✓] IIS Express
  - [✓] SQL ADAL runtime
  - [✓] SQL Server Express 2016 LocalDB
  - [✓] Connectivity and publishing tools
  - [✓] CLR data types for SQL Server
  - [✓] SQL Server ODBC Driver
  - [✓] SQL Server Command Line Utilities
  - [✓] .NET Framework 4.5.2 targeting pack
  - [✓] .NET Framework 4.5 targeting pack
  - [✓] SQL Server Data Tools
  - [✓] Data sources for SQL Server support
  - [✓] ASP.NET and web development tools
  - [✓] .NET Core 2.1 Runtime (LTS)
  - [✓] Advanced ASP.NET features
  - [✓] Web Deploy
  - [✓] Live Share
  - [✓] C++ ATL for latest v142 build tools (x86 & x64)
  - [✓] C++ MFC for latest v142 build tools (x86 & x64)
  - [✓] Windows Universal CRT SDK
  - [✓] GitHub Extension for Visual Studio
  - [✓] Git for Windows
  - [✓] Class Designer
  - [✓] C++ 2019 Redistributable MSMs
  - [✓] ASP.NET MVC 4
  - [✓] C++ Windows XP Support for VS 2017 (v141) to...

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 0 KB

Once Visual Studio is installed, you will need to do make important configuration changes:

1. You need to manually define very important environment variables
2. Fix Visual Studio's “**Default** Property Sheets” to add the INCLUDE directories specifying the location of the `win32.mak` file installed by the above installation option. This will be more clearly explained in the next step.

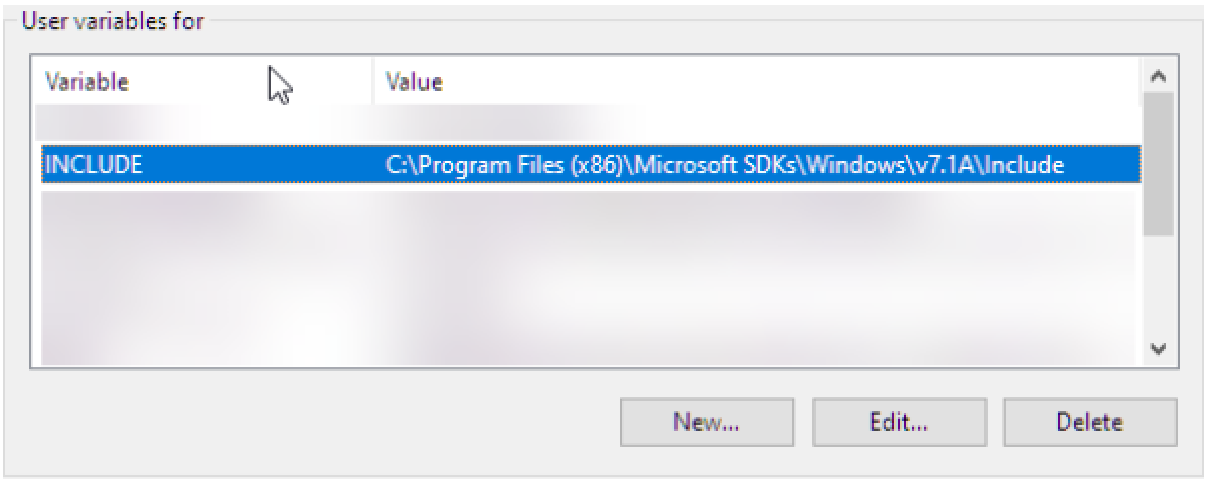
---

2. Define Environment Variables and Fix Property Sheets

INCLUDE

The **`INCLUDE`** environment variable must be defined because it identifies the location of an additional list of compiler search directories. This *must* indicate the directory where the “win32.mak” file was installed. (The “Windows XP support” install option is what provided the “win32.mak”.) This directory varies depending on the version of Visual Studio installed and where it was installed, but for most people it will be:

> **C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1A\Include**



INCLUDE C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1A\Include

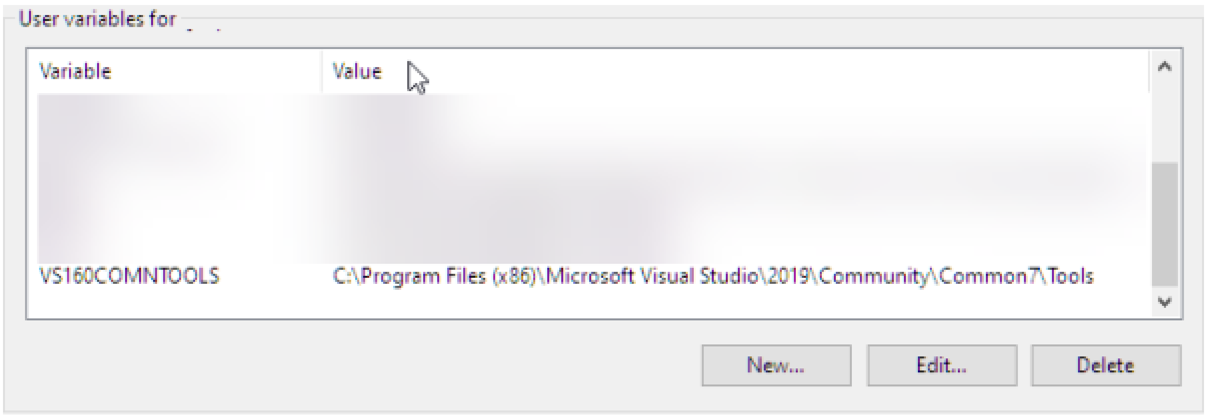
---

VSnnnCOMNTOOLS

The **`VSnnnCOMNTOOLS`** environment variable must also be defined *(where 'nnn' is the internal version number of Visual Studio)*

- '140' for Visual Studio 2015
- '150' for Visual Studio 2017
- '160' for Visual Studio 2019
- **'170'** for Visual Studio 2022

In earlier versions of Visual Studio the installer automatically defined this environment variable for you, but in later versions of Visual Studio the installer no longer provides this. Because we require this variable for the automatic/batch build process, it will need to be defined.



VS160COMNTOOLS C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\Tools

This value identifies the location of the Visual Studio “Common Tools” directory. These tools enable the build routine `makefile.bat` to locate the necessary configuration files which initialize Visual Studio's build environment.

For most people this directory will be:

> **C:\Program Files (x86)\Microsoft Visual Studio\yyyy\Community\Common7\Tools**

For Visual Studio 2022, the directory is most likely to be:

> **C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\Tools**

where '*yyyy*' is of course the version of Visual Studio (“2015”, “2017”, “2019”, “2022”).

Default Property Sheets

> NOTE: This task must be done in ELEVATED mode because it modifies installed Program files.

Finally, we must modify Visual Studio's “Default Property Sheets”; one for 32-bit and another for 64-bit.

These property sheets contain the defaults which are used by MSBUILD and tell Visual Studio how to initialize its various configuration values. Visual Studio does not (by default) include directories identified by the `INCLUDE` environment variable. These modification enable the capability.

The Default Installation contains the following `toolset.props` property files:

- C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Microsoft\VC\v150\Platforms\Win32\PlatformToolsets\v141\Toolset.props
- C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Microsoft\VC\v150\Platforms\Win32\PlatformToolsets\v141_xp\Toolset.props
- C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Microsoft\VC\v150\Platforms\x64\PlatformToolsets\v141\Toolset.props
- C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Microsoft\VC\v150\Platforms\x64\PlatformToolsets\v141_xp\Toolset.props
- C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Microsoft\VC\v160\Platforms\ARM\PlatformToolsets\v142\Toolset.props
- C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Microsoft\VC\v160\Platforms\Win32\PlatformToolsets\v142\Toolset.props
- CC:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Microsoft\VC\v160\Platforms\x64\PlatformToolsets\v142\Toolset.props

If you have other toolchains installed - you may need to locate them:

- C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\ARM\PlatformToolsets\v140\Toolset.props
- C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\Win32\PlatformToolsets\v140\Toolset.props
- C:\Program Files

(x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\Win32\PlatformToolsets\v140_xp\Toolset.props
- C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\x64\PlatformToolsets\v140\Toolset.props
- C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\x64\PlatformToolsets\v140_xp\Toolset.props

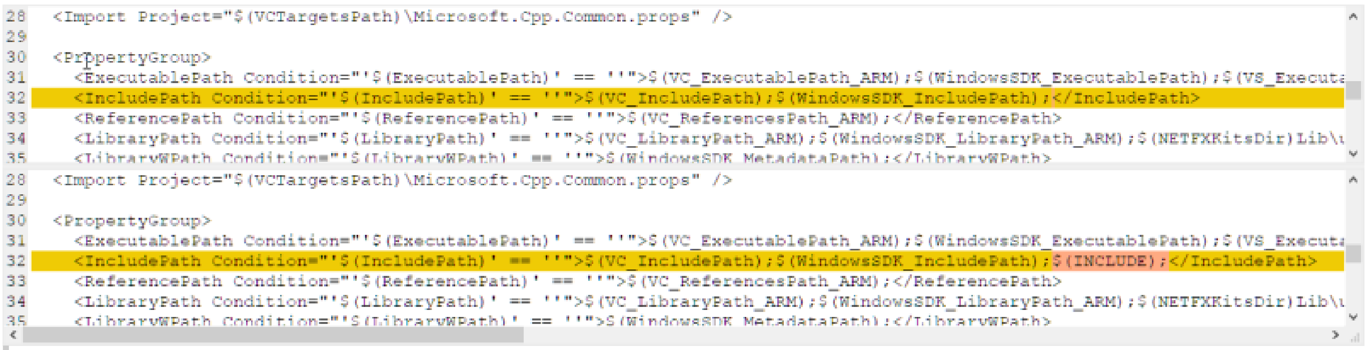
We are only concerned with these three:

- C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\ARM\PlatformToolsets\v140\Toolset.props
- C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\Win32\PlatformToolsets\v140\Toolset.props
- C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\x64\PlatformToolsets\v140\Toolset.props

Backup/rename the originals to:

- C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\ARM\PlatformToolsets\v140\Toolset.props.original
- C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\Win32\PlatformToolsets\v140\Toolset.props.original
- C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\v140\Platforms\x64\PlatformToolsets\v140\Toolset.props.original

Then modify them as illustrated here:



```
<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Common.props" />

<PropertyGroup>
  <ExecutablePath Condition="'$(ExecutablePath)' == ''">$(VC_ExecutablePath_ARM);$(WindowsSDK_ExecutablePath);$(VS_Executa
  <IncludePath Condition="'$(IncludePath)' == ''">$(VC_IncludePath);$(WindowsSDK_IncludePath);</IncludePath>
  <ReferencePath Condition="'$(ReferencePath)' == ''">$(VC_ReferencesPath_ARM);</ReferencePath>
  <LibraryPath Condition="'$(LibraryPath)' == ''">$(VC_LibraryPath_ARM);$(WindowsSDK_LibraryPath_ARM);$(NETFXKitsDir)Lib\u
  <LibraryWPath Condition="'$(LibraryWPath)' == ''">$(WindowsSDK_MetadataPath);</LibraryWPath>
```

```
<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Common.props" />

<PropertyGroup>
  <ExecutablePath Condition="'$(ExecutablePath)' == ''">$(VC_ExecutablePath_ARM);$(WindowsSDK_ExecutablePath);$(VS_Executa
  <IncludePath Condition="'$(IncludePath)' == ''">$(VC_IncludePath);$(WindowsSDK_IncludePath);$(INCLUDE);</IncludePath>
  <ReferencePath Condition="'$(ReferencePath)' == ''">$(VC_ReferencesPath_ARM);</ReferencePath>
  <LibraryPath Condition="'$(LibraryPath)' == ''">$(VC_LibraryPath_ARM);$(WindowsSDK_LibraryPath_ARM);$(NETFXKitsDir)Lib\u
  <LibraryWPath Condition="'$(LibraryWPath)' == ''">$(WindowsSDK_MetadataPath);</LibraryWPath>
```

> NOTE: The "INCLUDE" environment variable we defined earlier above is very important in that it defines the location where the "win32.mak" file lives which Hercules needs to initialize its build settings. We place it in its own environment variable because the "INCLUDE" environment variable is defined as a semicolon-delimited list of directories to be searched, and thus can contain other additional directories to be searched. Each development toolchain behaves differently so we have modified Visual Studio's default behavior in such a way that our build work can be more flexible.

As you can see, all we are doing is appending the "**$(INCLUDE)**" directories to the end of Visual Studio's default search directories. This enables Visual Studio to locate the critical "*win32.mak*" file when the build procedure asks for it.

Once you have Visual Studio installed and have defined the two environment variables and fixed the

Property Sheets, then you are finished with the Visual Studio installation portion of the setup.

SPECIAL INSTRUCTIONS FOR WINDOWS 10 and 11:

> Before your newly defined environment variables can take effect, you will need to first logoff and then log back on! Earlier versions of Windows are smart enough to dynamically update the environment immediately after being modified, but Windows 10 is different! You have to logoff and logon again (or reboot) before the new environment variables will take effect!

---

3. Setting up ZLIB Support

ZLIB is a compression algorithm written by Jean Loup Gailly and Mark Adler and *may* be used in the Hercules project pursuant to the ZLIB License (a copy of which may be seen at http://www.zlib.net/zlib_license.html).

In source form, the Hercules project *does* not contain any ZLIB source code.

In *binary* form however, the Hercules project *may* include an *unmodified* version of the ZLIB runtime DLL in addition to its own distribution binaries.

The '**ZLIB_DIR**' environment variable defines the location of where the required files are for building a version of Hercules that supports ZLIB compression. The `makefile.bat` and related MSBuild files used by the Hercules build process test whether this environment variable is defined as an indicator to build ZLIB compression support into Hercules.

If 'ZLIB_DIR' is undefined when you invoke the `makefile` the MSBuild functionality attempts to find the required files in a predefined default directory **`winbuild`**. If the MSBuild functionality cannot find them, then ZLIB support will not be generated. Otherwise 'ZLIB_DIR' must point to a valid directory where the ZLIB package is installed and that directory MUST have the following structure:

General User
HYPERION\WINBUILD\ZLIB +---Debug | +---include | \---lib +---include +---lib \---x64 +---Debug | +---include | \---lib +---include \---lib

ZLIB_DIR must contain the top path of the ZLIB directory.

When building a 64-bit (x64) version of Hercules the above 'x64' subdirectories are automatically searched. As long as the above directory structure is observed then Hercules ZLIB functionality will be included.

The SDL-provided Library distribution expected within this structure is:

General User
HYPERION\WINBUILD\ZLIB | zlib1.dll | zlib1.pdb | +---Debug | | zlib1.dll | | zlib1.pdb | | | +---include | | zconf.h | | zlib.h | | | \---lib | zdll.lib | +---include | zconf.h | zlib.h | +---lib | zdll.lib | \---x64 | zlib1.dll | zlib1.pdb | +---Debug | | zlib1.dll | | zlib1.pdb | | | +---include | | zconf.h | | zlib.h | | | \---lib | zdll.lib | +---include | zconf.h | zlib.h | \---lib zdll.lib

3.A Setting up ZLIB "Current Library" Support

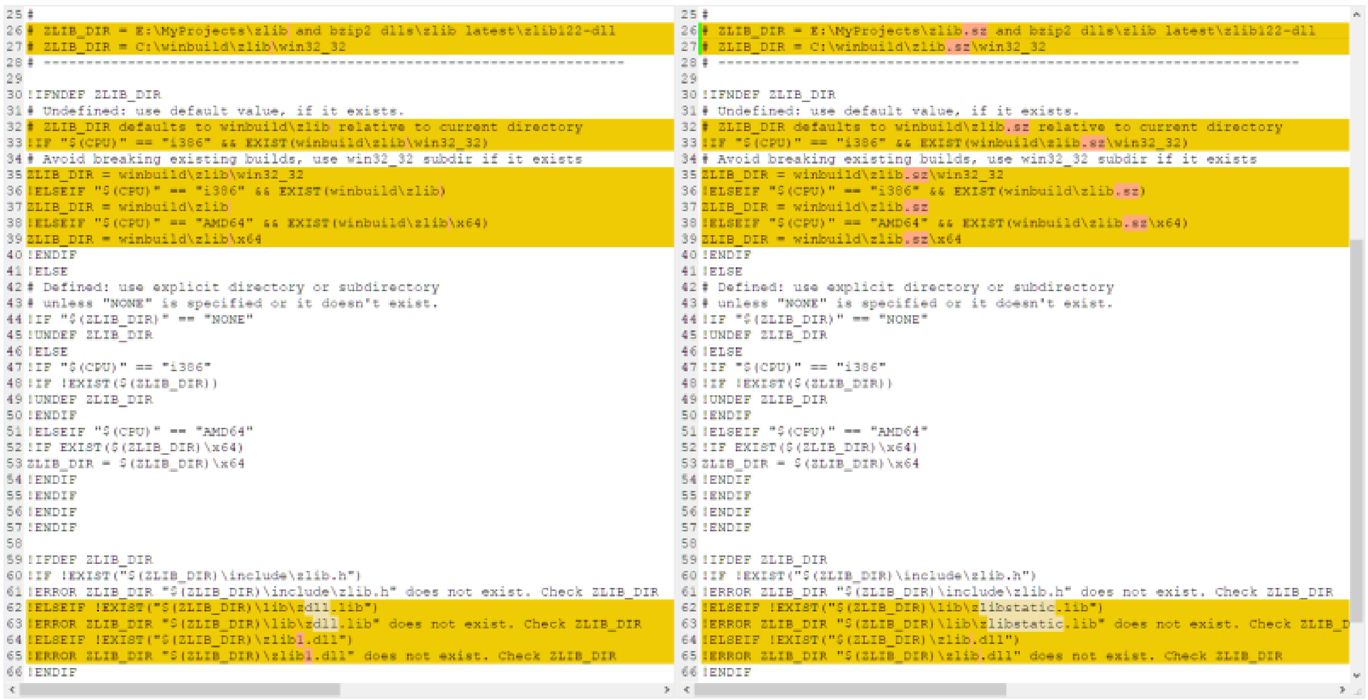
If your build requires the “Current Version” libraries, the resulting hierarchy will be substantially different than the one required for Hyperion/Hercules. It is for this reason that you should BUILD the libraries separately, and then move them into place using the established hierarchy indicated above.

For ZLIB, the required hierarchy using ZLIB's present-day build mechanism on windows requires that you copy the files from their build directory structure into the following hierarchy:

General User
HYPERION\WINBUILD\ZLIB.SZ | zlib.dll | zlib.pdb | zlibstatic.lib | +---Debug | | zlibd.dll | | zlibd.pdb | | | +---include | | zconf.h | | zlib.h | | | \---lib | zlibd.lib | zlibstaticd.lib | +---include | zconf.h | zlib.h | +---lib | zlib.lib | zlibstatic.lib | \---x64 | zlib.dll | zlib.pdb | +---Debug | | zlibd.dll | | zlibd.pdb | | | +---include | | zconf.h | | zlib.h | | | \---lib | zlibd.lib | zlibstaticd.lib | +---include | zconf.h | zlib.h | \---lib zlib.dll zlib.exp zlib.ilk zlib.lib zlib.pdb zlibstatic.lib

and you will then need to modify the MSVC files accordingly:

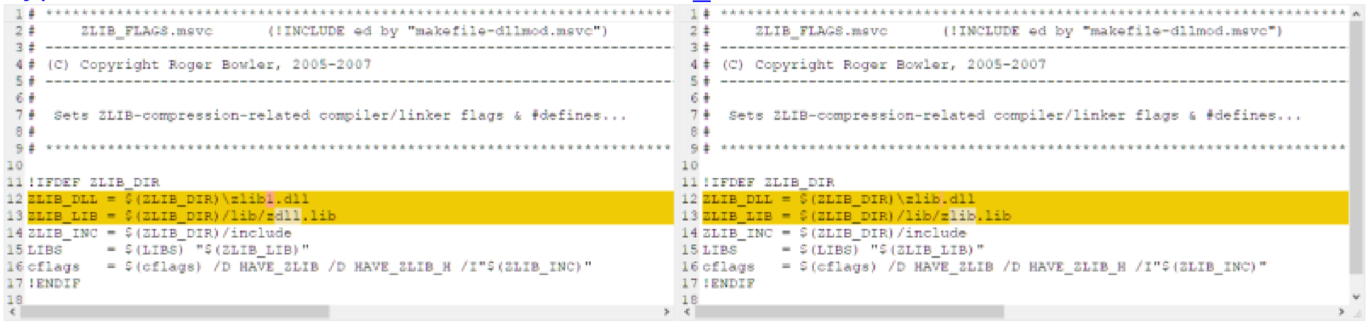
hyperion\msvc.makefile.sz.includes\ZLIB_DIR.msvc



```
#
# ZLIB_DIR = E:\MyProjects\zlib and bzip2 dlls\zlib latest\zlib122-dll
# ZLIB_DIR = C:\winbuild\zlib\win32_32
# ---------------------------------------------------------------------

!IFNDEF ZLIB_DIR
# Undefined: use default value, if it exists.
# ZLIB_DIR defaults to winbuild\zlib relative to current directory
!IF "$(CPU)" == "i386" && EXIST(winbuild\zlib\win32_32)
# Avoid breaking existing builds, use win32_32 subdir if it exists
ZLIB_DIR = winbuild\zlib\win32_32
!ELSEIF "$(CPU)" == "i386" && EXIST(winbuild\zlib)
ZLIB_DIR = winbuild\zlib
!ELSEIF "$(CPU)" == "AMD64" && EXIST(winbuild\zlib\x64)
ZLIB_DIR = winbuild\zlib\x64
!ENDIF
!ELSE
# Defined: use explicit directory or subdirectory
# unless "NONE" is specified or it doesn't exist.
!IF "$(ZLIB_DIR)" == "NONE"
!UNDEF ZLIB_DIR
!ELSE
!IF "$(CPU)" == "i386"
!IF !EXIST($(ZLIB_DIR))
!UNDEF ZLIB_DIR
!ENDIF
!ELSEIF "$(CPU)" == "AMD64"
!IF EXIST($(ZLIB_DIR)\x64)
ZLIB_DIR = $(ZLIB_DIR)\x64
!ENDIF
!ENDIF
!ENDIF
!ENDIF

!IFDEF ZLIB_DIR
!IF !EXIST("$(ZLIB_DIR)\include\zlib.h")
!ERROR ZLIB_DIR "$(ZLIB_DIR)\include\zlib.h" does not exist. Check ZLIB_DIR
!ELSEIF !EXIST("$(ZLIB_DIR)\lib\zdll.lib")
!ERROR ZLIB_DIR "$(ZLIB_DIR)\lib\zdll.lib" does not exist. Check ZLIB_DIR
!ELSEIF !EXIST("$(ZLIB_DIR)\zlib1.dll")
!ERROR ZLIB_DIR "$(ZLIB_DIR)\zlib1.dll" does not exist. Check ZLIB_DIR
!ENDIF
```

```
#
# ZLIB_DIR = E:\MyProjects\zlib.sz and bzip2 dlls\zlib latest\zlib122-dll
# ZLIB_DIR = C:\winbuild\zlib.sz\win32_32
# ---------------------------------------------------------------------

!IFNDEF ZLIB_DIR
# Undefined: use default value, if it exists.
# ZLIB_DIR defaults to winbuild\zlib.sz relative to current directory
!IF "$(CPU)" == "i386" && EXIST(winbuild\zlib.sz\win32_32)
# Avoid breaking existing builds, use win32_32 subdir if it exists
ZLIB_DIR = winbuild\zlib.sz\win32_32
!ELSEIF "$(CPU)" == "i386" && EXIST(winbuild\zlib.sz)
ZLIB_DIR = winbuild\zlib.sz
!ELSEIF "$(CPU)" == "AMD64" && EXIST(winbuild\zlib.sz\x64)
ZLIB_DIR = winbuild\zlib.sz\x64
!ENDIF
!ELSE
# Defined: use explicit directory or subdirectory
# unless "NONE" is specified or it doesn't exist.
!IF "$(ZLIB_DIR)" == "NONE"
!UNDEF ZLIB_DIR
!ELSE
!IF "$(CPU)" == "i386"
!IF !EXIST($(ZLIB_DIR))
!UNDEF ZLIB_DIR
!ENDIF
!ELSEIF "$(CPU)" == "AMD64"
!IF EXIST($(ZLIB_DIR)\x64)
ZLIB_DIR = $(ZLIB_DIR)\x64
!ENDIF
!ENDIF
!ENDIF
!ENDIF

!IFDEF ZLIB_DIR
!IF !EXIST("$(ZLIB_DIR)\include\zlib.h")
!ERROR ZLIB_DIR "$(ZLIB_DIR)\include\zlib.h" does not exist. Check ZLIB_DIR
!ELSEIF !EXIST("$(ZLIB_DIR)\lib\zlibstatic.lib")
!ERROR ZLIB_DIR "$(ZLIB_DIR)\lib\zlibstatic.lib" does not exist. Check ZLIB_D
!ELSEIF !EXIST("$(ZLIB_DIR)\zlib.dll")
!ERROR ZLIB_DIR "$(ZLIB_DIR)\zlib.dll" does not exist. Check ZLIB_DIR
!ENDIF
```

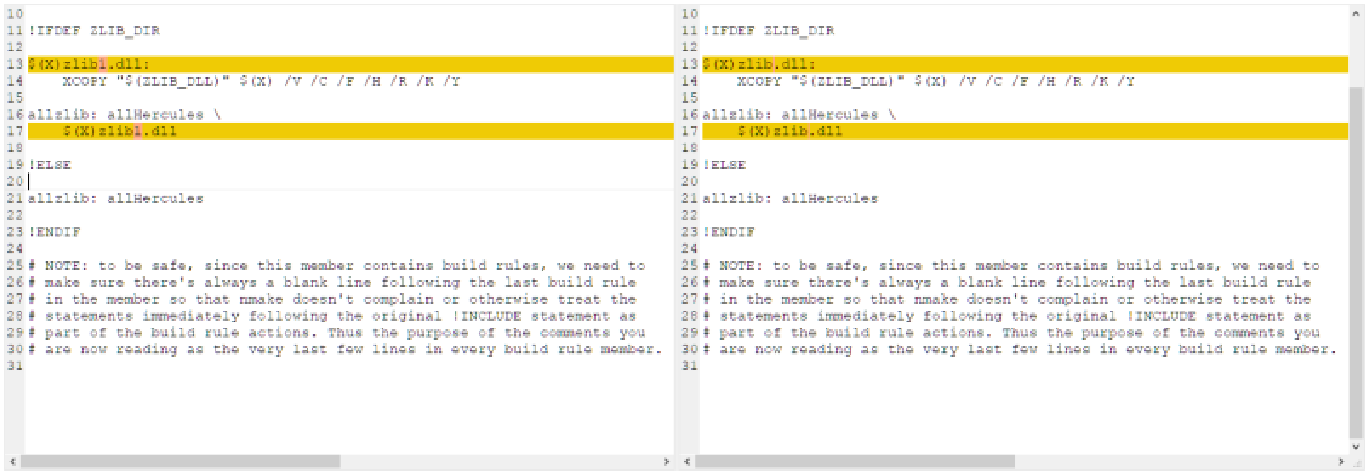
hyperion\msvc.makefile.sz.includes\ZLIB_FLAGS.msvc



```
# ********************************************************************
#      ZLIB_FLAGS.msvc       (!INCLUDE ed by "makefile-dllmod.msvc")
# --------------------------------------------------------------------
# (C) Copyright Roger Bowler, 2005-2007
# --------------------------------------------------------------------
#
#  Sets ZLIB-compression-related compiler/linker flags & #defines...
#
# ********************************************************************

!IFDEF ZLIB_DIR
ZLIB_DLL = $(ZLIB_DIR)\zlib1.dll
ZLIB_LIB = $(ZLIB_DIR)/lib/zdll.lib
ZLIB_INC = $(ZLIB_DIR)/include
LIBS     = $(LIBS)  "$(ZLIB_LIB)"
cflags   = $(cflags)  /D HAVE_ZLIB /D HAVE_ZLIB_H /I"$(ZLIB_INC)"
!ENDIF

```

```
# ********************************************************************
#      ZLIB_FLAGS.msvc       (!INCLUDE ed by "makefile-dllmod.msvc")
# --------------------------------------------------------------------
# (C) Copyright Roger Bowler, 2005-2007
# --------------------------------------------------------------------
#
#  Sets ZLIB-compression-related compiler/linker flags & #defines...
#
# ********************************************************************

!IFDEF ZLIB_DIR
ZLIB_DLL = $(ZLIB_DIR)\zlib.dll
ZLIB_LIB = $(ZLIB_DIR)/lib/zlib.lib
ZLIB_INC = $(ZLIB_DIR)/include
LIBS     = $(LIBS)  "$(ZLIB_LIB)"
cflags   = $(cflags)  /D HAVE_ZLIB /D HAVE_ZLIB_H /I"$(ZLIB_INC)"
!ENDIF

```

hyperion\msvc.makefile.sz.includes\ZLIB_RULES.msvc



```

!IFDEF ZLIB_DIR

$(X)zlib1.dll:
    XCOPY "$(ZLIB_DLL)" $(X) /V /C /F /H /R /K /Y

allzlib: allHercules \
    $(X)zlib1.dll

!ELSE

allzlib: allHercules

!ENDIF

# NOTE: to be safe, since this member contains build rules, we need to
# make sure there's always a blank line following the last build rule
# in the member so that nmake doesn't complain or otherwise treat the
# statements immediately following the original !INCLUDE statement as
# part of the build rule actions. Thus the purpose of the comments you
# are now reading as the very last few lines in every build rule member.

```

```

!IFDEF ZLIB_DIR

$(X)zlib.dll:
    XCOPY "$(ZLIB_DLL)" $(X) /V /C /F /H /R /K /Y

allzlib: allHercules \
    $(X)zlib.dll

!ELSE

allzlib: allHercules

!ENDIF

# NOTE: to be safe, since this member contains build rules, we need to
# make sure there's always a blank line following the last build rule
# in the member so that nmake doesn't complain or otherwise treat the
# statements immediately following the original !INCLUDE statement as
# part of the build rule actions. Thus the purpose of the comments you
# are now reading as the very last few lines in every build rule member.

```

4. Setting up BZIP2 Support

BZIP2 is a freely available (open-source (BSD-style) license), patent free (as far as the author knows), high-quality data compressor written by Julian R Seward. It typically compresses files to within 10% to 15% of the best available techniques (the PPM family of statistical compressors), whilst being around twice as fast at compression and six times faster at decompression.

In source form, the Hercules project *does* not contain any BZIP2 source code.

In *binary* form however, the Hercules project *may* include an *unmodified* version of the BZIP2 runtime DLL in addition to its own distribution binaries.

The '**BZIP2_DIR**' environment variable defines the location of where the required files are for building a version of Hercules that supports BZIP2 compression. The `makefile.bat` and related MSBuild files used by the Hercules build process test whether this environment variable is defined as an indicator to build BZIP2 compression support into Hercules.

If 'BZIP2_DIR' is undefined when you invoke the `makefile` the MSBuild functionality attempts to find the required files in a predefined default directory **`winbuild`**. If the MSBuild functionality cannot find them, then BZIP2 support will not be generated. Otherwise 'BZIP2_DIR' must point to a valid directory where the BZIP2 package is installed and that directory MUST have the following structure:

General User
HYPERION\WINBUILD\BZIP2 +---Debug \---x64 \---Debug

ZLIB_DIR must contain the top path of the ZLIB directory.

When building a 64-bit (x64) version of Hercules the above 'x64' subdirectories are automatically searched. As long as the above directory structure is observed then Hercules ZLIB functionality will be included.

The SDL-provided Library distribution expected within this structure is:

General User
HYPERION\WINBUILD\BZIP2 | bzip2.org.url | bzlib.h | libbz2.dll | libbz2.lib | libbz2.pdb | +---Debug | libbz2.dll | libbz2.lib | libbz2.pdb | \---x64 | bzlib.h | libbz2.dll | libbz2.lib | libbz2.pdb | \---Debug

libbz2.dll libbz2.lib libbz2.pdb

4.A Setting up BZIP2 "Current Library" Support

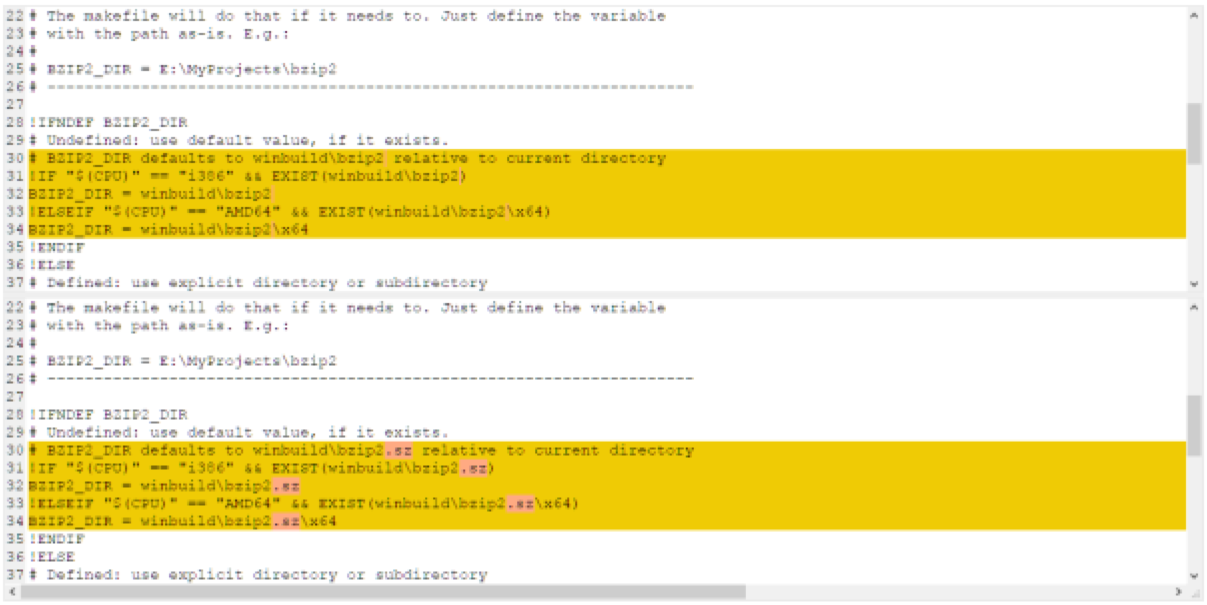
If your build requires the “Current Version” libraries, the resulting hierarchy will be substantially different than the one required for Hyperion/Hercules. It is for this reason that you should BUILD the libraries separately, and then move them into place using the established hierarchy indicated above.

For ZLIB, the required hierarchy using ZLIB's present-day build mechanism on windows requires that you copy the files from their build directory structure into the following hierarchy:

General User
HYPERION\WINBUILD\BZIP2.SZ | bzlib.h | libbz2.dll | libbz2.lib | libbz2.pdb | +---Debug | libbz2.dll | libbz2.lib | libbz2.pdb | +---include | bzlib.h | \---x64 | bzlib.h | libbz2.dll | libbz2.lib | libbz2.pdb | \---Debug libbz2.dll libbz2.lib libbz2.pdb

and you will then need to modify the MSVC files accordingly:

`hyperion\msvc.makefile.sz.includes\BZIP2_DIR.msvc`



```
# The makefile will do that if it needs to. Just define the variable
# with the path as-is. E.g.:
#
# BZIP2_DIR = E:\MyProjects\bzip2
# ------------------------------------------------------------------------

!IFNDEF BZIP2_DIR
# Undefined: use default value, if it exists.
# BZIP2_DIR defaults to winbuild\bzip2 relative to current directory
!IF "$(CPU)" == "i386" && EXIST(winbuild\bzip2)
BZIP2_DIR = winbuild\bzip2
!ELSEIF "$(CPU)" == "AMD64" && EXIST(winbuild\bzip2\x64)
BZIP2_DIR = winbuild\bzip2\x64
!ENDIF
!ELSE
# Defined: use explicit directory or subdirectory
```

```
# The makefile will do that if it needs to. Just define the variable
# with the path as-is. E.g.:
#
# BZIP2_DIR = E:\MyProjects\bzip2
# ------------------------------------------------------------------------

!IFNDEF BZIP2_DIR
# Undefined: use default value, if it exists.
# BZIP2_DIR defaults to winbuild\bzip2.sz relative to current directory
!IF "$(CPU)" == "i386" && EXIST(winbuild\bzip2.sz)
BZIP2_DIR = winbuild\bzip2.sz
!ELSEIF "$(CPU)" == "AMD64" && EXIST(winbuild\bzip2.sz\x64)
BZIP2_DIR = winbuild\bzip2.sz\x64
!ENDIF
!ELSE
# Defined: use explicit directory or subdirectory
```

5. Setting up PCRE Support

> NOTE: the Perl-Compatible Regular Expressions library is needed only to support the *Hercules Automatic Operator* (HAO) Facility. If you do not plan to ever use the Hercules Automatic Operator facility, then you do not need to install PCRE support and may safely skip this step.

PCRE (Perl-Compatible Regular Expressions) is: “a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5. PCRE has its own native API, as well as a set of wrapper functions that correspond to the POSIX regular expression API. The PCRE library is free, even for building commercial software.”

In source form, the Hercules project *does* not contain any PCRE source code.

In *binary* form however, the Hercules project *may* include an *unmodified* version of the PCRE runtime DLL in addition to its own distribution binaries.

The '**PCRE_DIR**' environment variable defines the location of where the required files are for building a version of Hercules that supports PCRE compression. The `makefile.bat` and related MSBuild files used by the Hercules build process test whether this environment variable is defined as an indicator to build PCRE compression support into Hercules.

If 'PCRE_DIR' is undefined when you invoke the `makefile` the MSBuild functionality attempts to find the required files in a predefined default directory **`winbuild`**. If the MSBuild functionality cannot find them, then PCRE support will not be generated. Otherwise 'PCRE_DIR' must point to a valid directory where the PCRE package is installed and that directory MUST have the following structure:

General User
HYPERION\WINBUILD\PCRE +---bin +---include +---lib \---x64 +---bin +---include \---lib

PCRE_DIR must contain the top path of the PCRE directory.

When building a 64-bit (x64) version of Hercules the above 'x64' subdirectories are automatically searched. As long as the above directory structure is observed then Hercules PCRE functionality will be included.

The SDL-provided Library distribution expected within this structure is:

General User
HYPERION\WINBUILD\PCRE +---bin | pcre3.dll | pcre3.pdb | pcreposix3.dll | pcreposix3.pdb | +---include | pcre.h | pcreposix.h | +---lib | pcre.lib | pcreposix.lib | \---x64 +---bin | pcre3.dll | pcre3.pdb | pcreposix3.dll | pcreposix3.pdb | +---include | pcre.h | pcreposix.h | \---lib pcre.lib pcreposix.lib

5.A Setting up PCRE "Current Library" Support

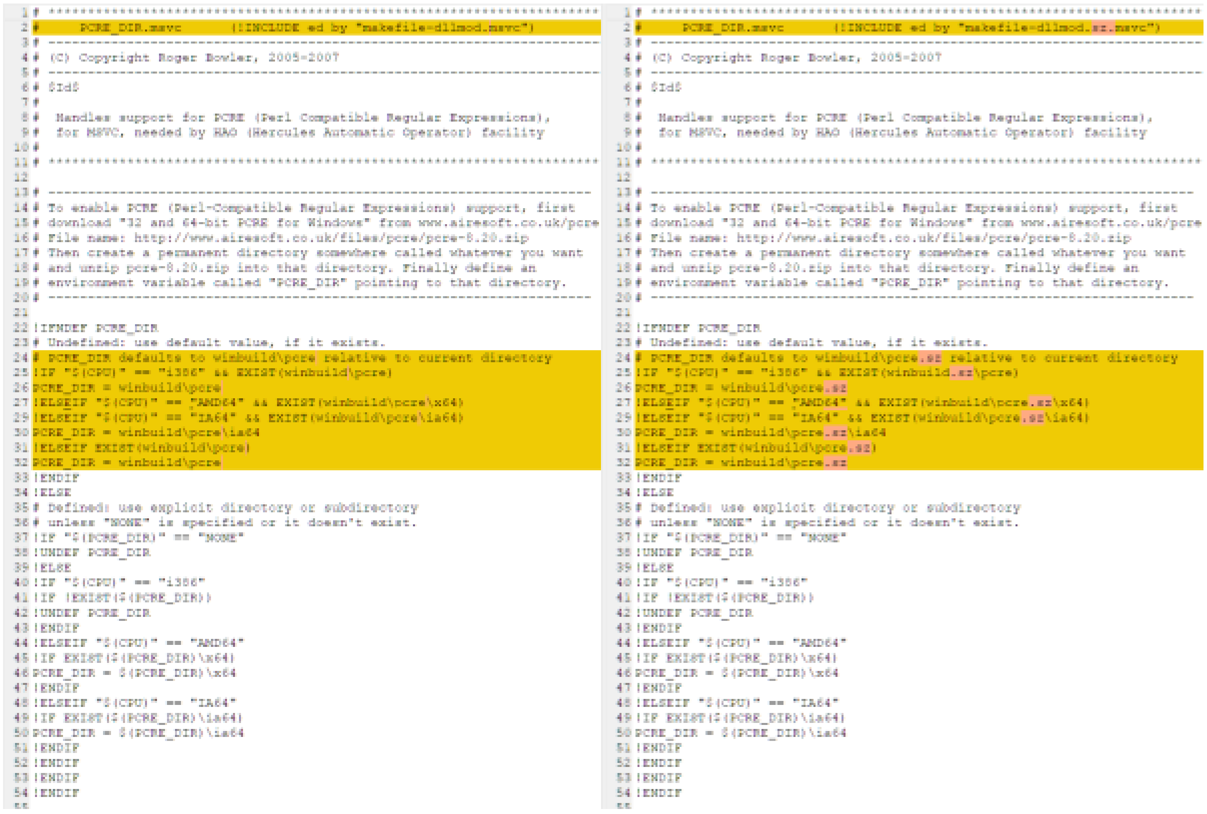
If your build requires the “Current Version” libraries, the resulting hierarchy will be substantially different than the one required for Hyperion/Hercules. It is for this reason that you should BUILD the libraries separately, and then move them into place using the established hierarchy indicated above.

For PCRE, the required hierarchy using PCRE's present-day build mechanism on windows requires that you copy the files from their build directory structure into the following hierarchy:

General User
HYPERION\WINBUILD\PCRE.SZ +---bin | pcre.dll | pcre.pdb | pcre16.dll | pcre16.pdb | pcre32.dll | pcre32.pdb | pcrecpp.dll | pcrecpp.pdb | pcreposix.dll | pcreposix.pdb | +---include | pcre.h | pcrecpp.h | pcreposix.h | +---lib | pcre.lib | pcre16.lib | pcre32.lib | pcrecpp.lib | pcreposix.lib | \---x64 +---bin | pcre.dll | pcre.pdb | pcre16.dll | pcre16.pdb | pcre32.dll | pcre32.pdb | pcrecpp.dll | pcrecpp.pdb | pcreposix.dll | pcreposix.pdb | +---include | pcre.h | pcrecpp.h | pcreposix.h | \---lib pcre.lib pcre16.lib pcre32.lib pcrecpp.lib pcreposix.lib

and you will then need to modify the MSVC files accordingly:

`hyperion\msvc.makefile.sz.includes\PCRE_DIR.msvc`

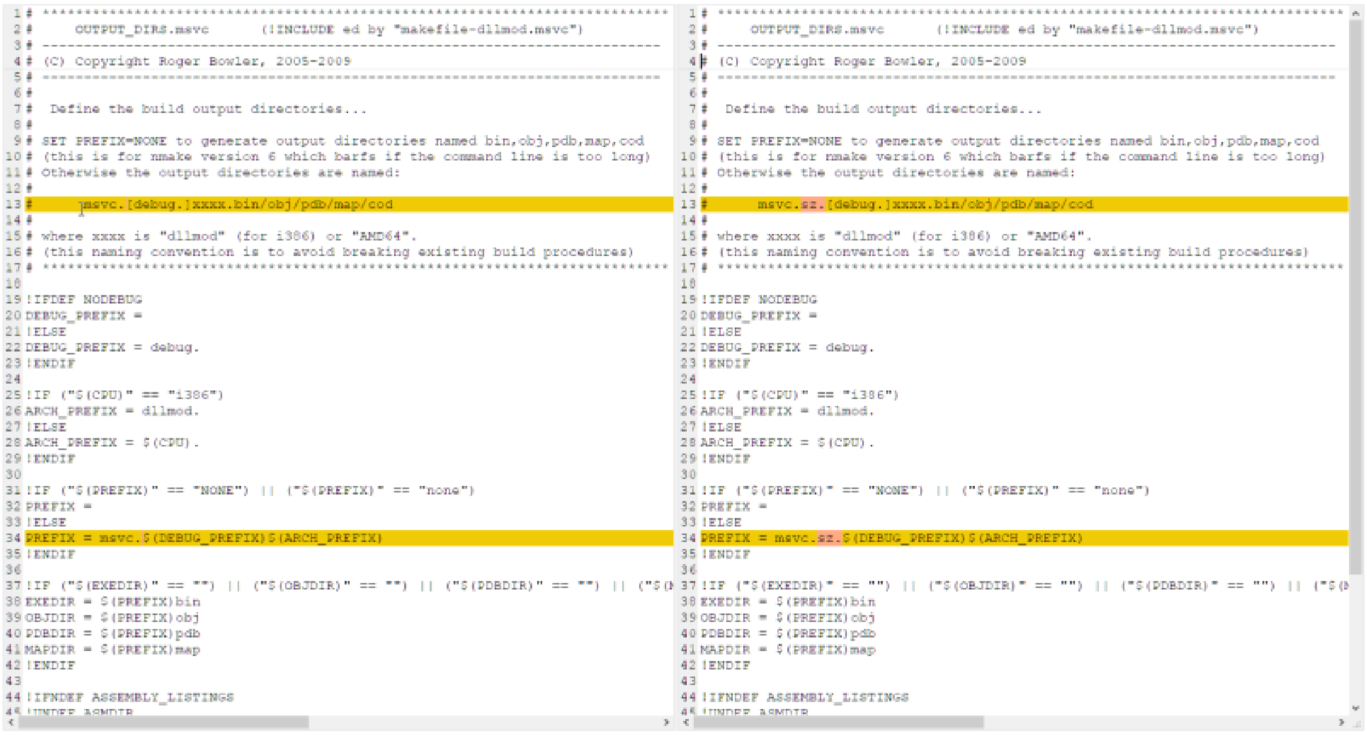


```
# *************************************************************************
#       PCRE_DIR.msvc       (!INCLUDE ed by "makefile-dllmod.msvc")
# -------------------------------------------------------------------------
# (C) Copyright Roger Bowler, 2005-2007
# -------------------------------------------------------------------------
# $Id$
#
#  Handles support for PCRE (Perl Compatible Regular Expressions),
#  for MSVC, needed by HAO (Hercules Automatic Operator) facility
#
# *************************************************************************

# -----------------------------------------------------------------------
# To enable PCRE (Perl-Compatible Regular Expressions) support, first
# download "32 and 64-bit PCRE for Windows" from www.airesoft.co.uk/pcre
# File name: http://www.airesoft.co.uk/files/pcre/pcre-8.20.zip
# Then create a permanent directory somewhere called whatever you want
# and unzip pcre-8.20.zip into that directory. Finally define an
# environment variable called "PCRE_DIR" pointing to that directory.
# -----------------------------------------------------------------------

!IFNDEF PCRE_DIR
# Undefined: use default value, if it exists.
# PCRE_DIR defaults to winbuild\pcre relative to current directory
!IF "$(CPU)" == "i386" && EXIST(winbuild\pcre)
PCRE_DIR = winbuild\pcre
!ELSEIF "$(CPU)" == "AMD64" && EXIST(winbuild\pcre\x64)
!ELSEIF "$(CPU)" == "IA64" && EXIST(winbuild\pcre\ia64)
PCRE_DIR = winbuild\pcre\ia64
!ELSEIF EXIST(winbuild\pcre)
PCRE_DIR = winbuild\pcre
!ENDIF
!ELSE
# Defined: use explicit directory or subdirectory
# unless "NONE" is specified or it doesn't exist.
!IF "$(PCRE_DIR)" == "NONE"
!UNDEF PCRE_DIR
!ELSE
!IF "$(CPU)" == "i386"
!IF !EXIST($(PCRE_DIR))
!UNDEF PCRE_DIR
!ENDIF
!ELSEIF "$(CPU)" == "AMD64"
!IF EXIST($(PCRE_DIR)\x64)
PCRE_DIR = $(PCRE_DIR)\x64
!ENDIF
!ELSEIF "$(CPU)" == "IA64"
!IF EXIST($(PCRE_DIR)\ia64)
PCRE_DIR = $(PCRE_DIR)\ia64
!ENDIF
!ENDIF
!ENDIF
!ENDIF
```

```
# *************************************************************************
#       PCRE_DIR.msvc       (!INCLUDE ed by "makefile-dllmod.sz.msvc")
# -------------------------------------------------------------------------
# (C) Copyright Roger Bowler, 2005-2007
# -------------------------------------------------------------------------
# $Id$
#
#  Handles support for PCRE (Perl Compatible Regular Expressions),
#  for MSVC, needed by HAO (Hercules Automatic Operator) facility
#
# *************************************************************************

# -----------------------------------------------------------------------
# To enable PCRE (Perl-Compatible Regular Expressions) support, first
# download "32 and 64-bit PCRE for Windows" from www.airesoft.co.uk/pcre
# File name: http://www.airesoft.co.uk/files/pcre/pcre-8.20.zip
# Then create a permanent directory somewhere called whatever you want
# and unzip pcre-8.20.zip into that directory. Finally define an
# environment variable called "PCRE_DIR" pointing to that directory.
# -----------------------------------------------------------------------

!IFNDEF PCRE_DIR
# Undefined: use default value, if it exists.
# PCRE_DIR defaults to winbuild\pcre.sz relative to current directory
!IF "$(CPU)" == "i386" && EXIST(winbuild.sz\pcre)
PCRE_DIR = winbuild\pcre.sz
!ELSEIF "$(CPU)" == "AMD64" && EXIST(winbuild\pcre.sz\x64)
!ELSEIF "$(CPU)" == "IA64" && EXIST(winbuild\pcre.sz\ia64)
PCRE_DIR = winbuild\pcre.sz\ia64
!ELSEIF EXIST(winbuild\pcre.sz)
PCRE_DIR = winbuild\pcre.sz
!ENDIF
!ELSE
# Defined: use explicit directory or subdirectory
# unless "NONE" is specified or it doesn't exist.
!IF "$(PCRE_DIR)" == "NONE"
!UNDEF PCRE_DIR
!ELSE
!IF "$(CPU)" == "i386"
!IF !EXIST($(PCRE_DIR))
!UNDEF PCRE_DIR
!ENDIF
!ELSEIF "$(CPU)" == "AMD64"
!IF EXIST($(PCRE_DIR)\x64)
PCRE_DIR = $(PCRE_DIR)\x64
!ENDIF
!ELSEIF "$(CPU)" == "IA64"
!IF EXIST($(PCRE_DIR)\ia64)
PCRE_DIR = $(PCRE_DIR)\ia64
!ENDIF
!ENDIF
!ENDIF
!ENDIF
```

(6.A) Modify OUTPUT_DIRS

To ensure that MSBuild generates outputs to the proper isolated directories - the following modifications MUST be made to the following file:



```
# *************************************************************************
#       OUTPUT_DIRS.msvc       (!INCLUDE ed by "makefile-dllmod.msvc")
# -------------------------------------------------------------------------
# (C) Copyright Roger Bowler, 2005-2009
# -------------------------------------------------------------------------
#
#  Define the build output directories...
#
# SET PREFIX=NONE to generate output directories named bin,obj,pdb,map,cod
# (this is for nmake version 6 which barfs if the command line is too long)
# Otherwise the output directories are named:
#
#       msvc.[debug.]xxxx.bin/obj/pdb/map/cod
#
# where xxxx is "dllmod" (for i386) or "AMD64".
# (this naming convention is to avoid breaking existing build procedures)
# *************************************************************************

!IFDEF NODEBUG
DEBUG_PREFIX =
!ELSE
DEBUG_PREFIX = debug.
!ENDIF

!IF ("$(CPU)" == "i386")
ARCH_PREFIX = dllmod.
!ELSE
ARCH_PREFIX = $(CPU).
!ENDIF

!IF ("$(PREFIX)" == "NONE") || ("$(PREFIX)" == "none")
PREFIX =
!ELSE
PREFIX = msvc.$(DEBUG_PREFIX)$(ARCH_PREFIX)
!ENDIF

!IF ("$(EXEDIR)" == "") || ("$(OBJDIR)" == "") || ("$(PDBDIR)" == "") || ("$(
EXEDIR = $(PREFIX)bin
OBJDIR = $(PREFIX)obj
PDBDIR = $(PREFIX)pdb
MAPDIR = $(PREFIX)map
!ENDIF

!IFNDEF ASSEMBLY_LISTINGS
!UNDEF ASMDIR
```

```
# *************************************************************************
#       OUTPUT_DIRS.msvc       (!INCLUDE ed by "makefile-dllmod.msvc")
# -------------------------------------------------------------------------
# (C) Copyright Roger Bowler, 2005-2009
# -------------------------------------------------------------------------
#
#  Define the build output directories...
#
# SET PREFIX=NONE to generate output directories named bin,obj,pdb,map,cod
# (this is for nmake version 6 which barfs if the command line is too long)
# Otherwise the output directories are named:
#
#       msvc.sz.[debug.]xxxx.bin/obj/pdb/map/cod
#
# where xxxx is "dllmod" (for i386) or "AMD64".
# (this naming convention is to avoid breaking existing build procedures)
# *************************************************************************

!IFDEF NODEBUG
DEBUG_PREFIX =
!ELSE
DEBUG_PREFIX = debug.
!ENDIF

!IF ("$(CPU)" == "i386")
ARCH_PREFIX = dllmod.
!ELSE
ARCH_PREFIX = $(CPU).
!ENDIF

!IF ("$(PREFIX)" == "NONE") || ("$(PREFIX)" == "none")
PREFIX =
!ELSE
PREFIX = msvc.sz.$(DEBUG_PREFIX)$(ARCH_PREFIX)
!ENDIF

!IF ("$(EXEDIR)" == "") || ("$(OBJDIR)" == "") || ("$(PDBDIR)" == "") || ("$(
EXEDIR = $(PREFIX)bin
OBJDIR = $(PREFIX)obj
PDBDIR = $(PREFIX)pdb
MAPDIR = $(PREFIX)map
!ENDIF

!IFNDEF ASSEMBLY_LISTINGS
!UNDEF ASMDIR
```

Building Hercules using the Visual Studio "makefile.bat"

Once you have installed Microsoft's Visual Studio 2019 Community Edition and have finished setting up the build environment, you can then easily build Hercules via the provided Visual Studio solution and project files included as part of the Hercules source-code distribution.

General User

```
CD [hercules build directory] makefile.bat retail makefile.msvc 8 CLEAN makefile.bat retail-x64 makefile.msvc 8 CLEAN makefile.bat debug makefile.msvc 8 CLEAN makefile.bat debug-x64 makefile.msvc 8 CLEAN makefile.bat retail makefile.msvc 8 makefile.bat retail-x64 makefile.msvc 8 makefile.bat debug makefile.msvc 8 makefile.bat debug-x64 makefile.msvc 8 if you built the "Current Libraries" and want to use those: makefile.bat retail makefile.sz.msvc 8 CLEAN makefile.bat retail-x64 makefile.sz.msvc 8 CLEAN makefile.bat debug makefile.sz.msvc 8 CLEAN makefile.bat debug-x64 makefile.sz.msvc 8 CLEAN makefile.bat retail makefile.sz.msvc 8 makefile.bat retail-x64 makefile.sz.msvc 8 makefile.bat debug makefile.sz.msvc 8 makefile.bat debug-x64 makefile.sz.msvc 8
```

That's it!

> NOTE: When using the IDE, clicking the “Rebuild Solution” button in Visual Studio simply invokes Hercules's “makefile.bat” script, which in turn invokes the `nmake` command for the make file called `makefile.msvc` or `makefile.sz.msvc` (after calling a few helper batch scripts to first define the Visual Studio build environment).
>
> All of the actual building (compiling and linking) is controlled by the “makefile.msvc” make file (which, as explained, is invoked automatically by Visual Studio when you click the “Rebuild Solution” button).

General User

```
hyperion>makefile /? makefile.bat begun on Fri 08/28/2020 at 13:10:07.47 cmdline: makefile /? makefile.bat(1) : error C9999 : Help information is as follows: makefile.bat Initializes the Windows software development build envionment and invokes nmake to build the desired 32 or 64-bit version of the Hercules emulator. Format: makefile.bat {build-type} {makefile-name} {num-cpu-engines} \ [-asm] \ [-title "custom build title"] \ [-hqa {directory}] \ [-extpkg {directory}] \ [-a|clean] \ [{nmake-option}] Where: {build-type} The desired build configuration. Valid values are DEBUG / RETAIL for building a 32-bit Hercules, or DEBUG-X64 / RETAIL-X64 to build a 64-bit version of Hercules targeting (favoring) AMD64 processors. DEBUG builds activate/enable UNOPTIMIZED debugging logic and are thus VERY slow and not recommended for normal use. RETAIL builds on the other hand are highly optimized and thus the recommended type for normal every day ("production") use. {makefile-name} The name of our makefile: 'makefile.msvc' (or some other makefile name if you have a customized one) {num-cpu-engines} The maximum number of emulated CPUs (NUMCPU=) you want this build of Hercules to support: 1 to 64. -asm To generate assembly (.cod) listings. -title "xxx..." To define a custom title for this build. -hqa "directory" To define the Hercules Quality Assurance directory containing your optional "hqa.h" and/or "HQA.msvc" build settings override files. -extpkg "directory" To define the base directory where the Hercules External Packages are installed. Hercules will use the 'include' and 'lib' subdirectories of this directory to locate External Package header files and lib files during the build process. If not specified the default is to use the header and lib files that come with the Hercules repository. [-a|clean] Use '-a' to perform a full rebuild of all Hercules binaries, or 'clean'
```

to delete all temporary work files from all work/output directories, including any/all previously built binaries. If not specified then only those modules that need to be rebuilt are actually rebuilt, usually resulting in much quicker build. However, when doing a 'RETAIL' build it is HIGHLY RECOMMENDED that you always specify the '-a' option to ensure that a complete rebuild is done. [{nmake-option}] Extra nmake option(s). (e.g. -k, -g, etc...) makefile.bat ended on Fri 08/28/2020 at 13:10:07.59, rc=1