OpenVMS x86-64 VSI C++ Release Notes
01-Mar-2023

1 VSI C++

This kit includes the field-test VSI C++ x86-64 compiler. This compiler runs on OpenVMS x86 and generates code for OpenVMS x86.

The compiler is based on the LLVM Clang compiler with additional OpenVMS features. You can learn more about Clang at https://clang.llvm.org .

The PCSI package provides two C++ compilers and CXX demangling tool:

1. CLANG - the compiler with only UNIX-style command line support.
   To use this compiler, you will need to call SYS$STARTUP:CXX$STARTUP.COM command file. It will set up only CLANG symbol.

2. CXX - the compiler with only DCL command line support.
   It does not support all qualifiers that the IA64 CXX does.
   All unsupported qualifiers are silently ignored. See section 1.5.

   The "$ HELP CXX" prints out general information about CXX compiler and lists all supported qualifiers. First listed are qualifiers that have actual effect, and then the qualifiers that are ignored for now.

   The demangling tool called CXX$DEMANGLE. It is as foreign command and sets up by calling SYS$STARTUP:CXX$STARTUP.COM script. It is taken from LLVM tool-set known as llvm-cxxfilt and is working in UNIX-style. For more information you can type:

   $ CXX$DEMANGLE --help

To see the changes for every version you can type

```
$ type SYS$HELP:CXX.CHANGELOG
```

The compilers behaves very much like the Linux version of the clang compiler in terms of command line options and language features. The primary differences are the support of the OpenVMS specific pragmas and predefined symbols.

VSI C++ predefines common OpenVMS symbols much like the Itanium C++ compiler (for example, VMS, __VMS, __vms, __INITIAL_POINTER_SIZE) as well as industry standard symbols such as __x86_64 and __x86_64__.

The compiler uses the headers extracted from SYS$STARLET.TLB and its own version of DECC$RTLDEF.TLB headers which are adapted for the C++ compiler. Currently those headers aren't part of the OS and come with the compiler, but in future releases those are going to be merged with DECC$RTLDEF. Also, clang brings its own cxx headers.

Unlike a traditional UNIX compiler which "compiles and links" with a single command, the compiler on OpenVMS only compiles (ie, the equivalent of the "-c" option on UNIX).

There are two RTLs to support C++ (LIBCXX and LIBCXXABI). C++ provides two versions of these libraries, static and shared. They are copied to the SYS$COMMON:[SYSLIB] directory during the installation. They are inserted into SYS$LIBRARY:IMAGELIB.OLB as well. So, if the program need to be linked against shared libraries, they will be found automatically. However, if the program needs to be linked against static ones, you need to provide them explicitly, or you can use SYS$COMMON:[SYSHLP.EXAMPLES]CXX.OPT (or SYS$EXAMPLES:CXX.OPT) file.

The OpenVMS x86 linker places code into 64-bit address by default (the default on Alpha and Itanium was to place code into 32-bit address space). While most programs will not notice the difference.

Example :

```
// HW.CXX
```

```
#include <iostream>

int main()
{
  std::cout << "Hello World!\n";
  return 0;
}

$ CXX HW.CXX
$ LINK HW,SYS$EXAMPLES:CXX/OPT
$ RUN HW
Hello World!
```

1.1 OpenVMS Specific Pragmas

Both CXX and Clang are extended to support OpenVMS specific pragmas which are supported include:

```
  #pragma pointer_size [options]
  #pragma required_pointer_size [options]
  #pragma [no]member_alignment [options]
  #pragma extern_model [options]
  #pragma extern_prefix [options]
```

Other OpenVMS pragmas are processed but only have partial/limited support at present. The #pragma message pragma is supported but the names of the error messages are different between Itanium and x86 so they need to be modified.

Notes:

* The #pragma pointer_size pragma is active only when the command line option -pointer-size is given. This matches the behavior of the VSI C++ Itanium compiler.

* The option provided to #pragma nomember_alignment has a meaning of base alignment of the structure. If provided appropriate alignment attribute is set on the structure declaration.
When using base_alignment option, please declare structures using typedef format, like

```
   typedef struct S {};
```

and not the free-standing form, like

```
   struct S {};
```

This form will produce warning message and will actually ignore the alignment attribute.

* At this moment we have partially support for #pragma extern_model.
Currently the only strict_refdef model is supported.

And the following options are available:

```
  gbl lcl
  shr noshr
  wrt nowrt
  ovr con
  exe noexe
  vec novec
```

and alignments:

```
  byte
  word
  long
  quad
  octa
```

1.2 TLB And Headers

VSI C++ currently doesn't support TLB format and instead it extracts

SYS$STARTLET.TLB headers into the SYS$COMMON:[VSICXX$LIB.INCLUDE.SYS$STARLET_C]
during the installation.

In addition, DECC$RTLDEF.TLB headers adapted for C++ are copied to the
SYS$COMMON:[VSICXX$LIB.INCLUDE.DECC$RTLDEF] from the kit during the
installation and Clang's cxx headers copied into the
SYS$COMMON:[VSICXX$LIB.INCLUDE.LIB_CXX] respectively. TLB support is planned for
the future releases and the RTL headers will include Clang C++ conditionals.

The DCL symbols for CXX and CLANG include options to point at these header
reference copies.

1.3 Differences Between C++ on OpenVMS Itanium and OpenVMS x86

The datatypes 'long', 'size_t', 'nullptr_t', 'ptrdiff_t' are 64-bits wide
on OpenVMS x86 but only 32-bits wide on OpenVMS Itanium.

The default size for pointers is 64-bits on OpenVMS x86 but only 32-bits on
OpenVMS Itanium. You can use the command line option '-pointer-size' for
CLANG and /POINTER_SIZE qualifier for CXX to change the default size of pointers.
Below there is a detailed description.

The compiler does not automatically upcase external names like all
other OpenVMS compilers. There is a '-names' command line option for CLANG
and /NAMES qualifier for CXX and their default is as_is.

1.4 Known Issues

- No support for TLB files and /LIBRARY qualifier

- long double
  The long double data type is not yet fully supported.

- No listing file generation

- No machine code listing file generation. You can create something
  similar with ANALYZE/OBJECT/DISASSEMBLE

- Still have some problems with 32-bit pointers. In some test-cases it causes
  an LLVM back-end fatal error. We are now working on this problem.

- The TRY/CATCH statement may not work correctly on V9.2 systems.

- The builtin function lib$establish() is not implemented in clang. It is in list.

- Calling 'std::cout.operator<<' within global objects constructors brings
  to “%SYSTEM-F-ACCVIO” when linked with static libraries cxx_static.olb
  and cxxabi_static.olb

- Don’t support the global new/delete operators overrides at this moment.
  To get this working, symbol preemption is needed. VMS does not have symbol preemption.

- Comma-separated list of source files is not supported by CXX, meaning you can't compile
  multiple files at once.

1.4.1 Using the Clang command line

- Clang is sensitive to input file extensions. Clang is both a C and a C++ compiler.
  If you compile a file with an “.c” extension, clang enters “C mode”.
  If you compile with a “.cpp” (or “.cxx”) it will enter “C++ mode”.
  In C-mode C++ features and libs (even STL) are not available.
  But there is "-x" CL option which can force clang to swith to specified
  language mode.

  $ clang -x c++ a.c
  This will compile a.c as C++ code.

- The command line options are also case-sensitive. But you can meet this kind of
  problem:

```
$ clang except.cpp -Wall -I disk2:[000000]
clang: error: unknown argument '-wall'; did you mean '-Wall'?
clang: error: unknown argument: '-i'
```

The issue here is that DCL with traditional parsing will upcase all the arguments and then the CRTL in an attempt to be "nice" will downcase all the arguments. You can prevent DCL from upcasing by using double-quotes on the "-Wall" and "-I" command line options.
Alternatively, you can prevent DCL from upcasing and the CRTL from downcasing with these definitions (you can put them in your login.com):

```
$ set process /parse=extended
$ define/nolog DECC$ARGV_PARSE_STYLE ENABLE
$ define/nolog DECC$EFS_CASE_PRESERVE ENABLE
$ define/nolog DECC$EFS_CHARSET ENABLE
```

- Clang accepts multiple source files in one command line. This line is ok:

```
$ clang a.cpp b.cpp
```

1.5 Supported and Not Supported DCL Qualifiers

The following table shows the current set of supported DCL qualifiers.
Support for the remaining qualifiers will be added in a future release.

Qualifier	Ignored	Supported
/VERSION	-	+
/CLANG	-	+
/COMMENTS	-	+
/DEFINE	-	+
/DEBUG	-	+
/ERROR_LIMIT	-	+
/EXCEPTIONS	-	+
/FIRST_INCLUDE	-	+
/INCLUDE_DIRECTORY	-	+
/L_DOUBLE_SIZE	-	+
/MEMBER_ALIGNMENT	-	+
/NAMES	-	+
/OBJECT	-	+
/OPTIMIZE	-	+
/POINTER_SIZE	-	+
/PREPROCESS_ONLY	-	+
/RTTI	-	+
/STANDARD	-	+
/UNDEFINE	-	+
/WARNINGS	-	+
/BREAKPOINTS	+	-

/GEMDEBUG	+	-
/DUMPS	+	-
/SWITCHES	+	-
/TRACEPOINTS	+	-
/ANSI_ALIAS	+	-
/ARCHITECHTURE	+	-
/ASSUME	+	-
/BE_DUMPS	+	-
/CHECK	+	-
/DIAGNOSTICS	+	-
/FLOAT	+	-
/G_FLOAT	+	-
/IMPLICIT_INCLUDE	+	-
/ENDIAN	+	-
/EXPORT_SYMBOLS	+	-
/EXTERN_MODEL	+	-
/GRANULARITY	+	-
/IEEE_MODE	+	-
/INSTRUCTION_SET	+	-
/LIBRARY	+	-
/LIST	+	-
/LINE_DIRECTIVE	+	-
/LOOKUP	+	-
/MACHINE_CODE	+	-
/MAIN	+	-
/MMS_DEPENDENCIES	+	-
/NESTED_INCLUDE_DIRECTORY	+	-
/OS_VERSION	+	-
/PENDING_INSTANTIATIONS	+	-
/PREFIX_LIBRARY_ENTRIES	+	-
/PSECT_MODEL	+	-
/PURE_CNAME	+	-
/USING_STD	+	-
/DISTINGUISH_NESTED_ENUMS	+	-
/ALTERNATIV_TOKENS	+	-

/QUIET	+	-
/REPOSITORY	+	-
/REENTRANCY	+	-
/ROUNDING_MODE	+	-
/SHARE_GLOBALS	+	-
/MODEL	+	-
/STACK_CHECK	+	-
/SHOW	+	-
/TEMPLATE_DEFINE	+	-
/UNSIGNED_CHAR	+	-
/XREF	+	-
/FE_DUMP	+	-
/BOTH_CASE	+	-

1.6 CXX Qualifiers To Clang CL Options

This section shows Clang options that are similar to CXX qualifiers.
Not all of these are perfect matches but are listed to provide a starting point.

/CLANG
This qualifier takes Clang options and passes them the Clang driver.
It's important to specify them in double quotes and separately.
So, instead of

```
CXX /CLANG=(-Wall, "-D__macro1 -D__MACRO2") foo.cxx
```

specify

```
CXX /CLANG=("-Wall", "-D__macro1", "-D__MACRO2") foo.cxx
```

/DEFINE
-D flag, so /DEFINE=TRUE becomes -DTRUE which results to "#define TRUE 1".

/INCLUDE_DIRECTORY
-I <directory> - add directory to include search path.
Also the compiler is extended to support files lookup at predefined logicals
locations.
Example:

```
// DISK:[SOURCES]INC.CPP
#include <SYS/INC.HPP>
int main()
{
  f();
}

// DISK:[HEADERS]INC.HPP
void f() {};
```

To compile INC.CPP, it's enough to define respective SYS logical as

```
$ DEFINE SYS DISK:[HEADERS]
```

And then just compile INC.CPP

```
$ SET DEF DISK:[SOURCES]
$ CXX INC.CPP
```

/POINTER_SIZE, /NOPOINTER_SIZE
-no-pointer-size

```
    * Disables processing of '#pragma pointer_size'
    * Predefines the preprocessor macro __INITIAL_POINTER_SIZE to 0
    * THE INITIAL DEFAULT POINTER SIZE IS 64-bit
      (On Alpha and Itanium, /NOPOINTER_SIZE directs the compiler
       to assume that all pointers are 32-bit pointers)

  -pointer-size={long|short|64|32}
    * "long" or "64" means:
      - Enables processing of '#pragma pointer_size'
      - Sets the initial default pointer size to 64-bit for translation unit
      - Predefines the preprocessor macro __INITIAL_POINTER_SIZE to 64
    * "short" or "32" means:
      - Enables processing of '#pragma pointer_size'
      - Sets the initial default pointer size to 32-bit for translation unit
      - Predefines the preprocessor macro __INITIAL_POINTER_SIZE to 32

  The default is "-no-pointer-size" option.

  /NAMES=(option1,option2)
  -names={uppercase|as_is} option
    This option controls the conversion of external symbols to the case
    specified. Two possible options are "uppercase" and "as_is".

    * "uppercase" - uppercases all the external symbols in translation unit.
    * "as_is" - leaves them as they are. This is default.

  -names2={truncated|shortened} option
    This option controls whether or not external names greater than
    31 characers get truncated or shortened.
    Two possible options are "truncated" and "shortened".

    * "truncated" - Truncates long external names to first 31 characters.
    * "shortened" - shortens long external names

    A shortened name consists of the first 23 characters of the name
    followed by a 7-character Cyclic Redundancy Check (CRC) computed
    by looking at the full name, and then a "$".

  /[NO]MEMBER_ALIGNMENT
  -[no-]member-alignment
  Clang is extended to support the -[no-]member-alignment command-line option
  Directs the compiler to naturally align data structure members. This means that data structure
  members are aligned on the next boundary appropriate to the type of the member, rather than on
  the next byte. For instance, a long variable member is aligned on the next longword boundary;
a
  short variable member is aligned on the next word boundary.
  Any use of the #pragma member_alignment or #pragma nomember_alignment
  directives within the source code overrides the setting established by this qualifier.
  Specifying -no-member-alignment causes data structure members to be byte-aligned (with
  the exception of bit-field members)

  /PENDING_INSTANTIATIONS
  -ftemplate-depth=n, set the maximum instantiation depth for
  template classes to n.

  /EXTERN_MODEL
  This qualifier is not supported by the compiler but it is
  planned for future releases to support respective pragma.

  /L_DOUBLE_SIZE

      /L_DOUBLE_SIZE
      /L_DOUBLE_SIZE=option
      /L_DOUBLE_SIZE=128 (D)
   Determines how the compiler interprets the long double type. The qualifier options are 64 and
128.

  /L_DOUBLE_SIZE for the Clang is respective to:
  -mlong-double-[128,64,80] 80 is supported by Clang, but VMS is not supporting 80 bit long
double size.
```

/STANDARD
/STANDARD=(option)
/STANDARD=RELAXED (D)
Following new keywords are added to explicitly specify C++ standards.
  CXX98, GNU98, CXX03, GNU03, CXX11, GNU11, CXX14, GNU14, CXX17, GNU17, CXX20, GNU20

Example:
  /STANDARD=CXX03 is translated to --std=c++03
  /STANDARD=GNU03 is translated to --std=gnu++03

The value LATEST is equivalent to STRICT_ANSI which is equivalent to CXX98

The default standard for CXX is set to GNU98

The equivalent CLANG option is:
-std=<value> - language standard to compile for.

/ARCHITECTURE

    /ARCHITECTURE=option
    Determines the Alpha,X86 or Intel processor instruction set to be
    used by the compiler.
    All Alpha processors implement a core set of instructions.
    Certain Alpha processor versions include additional instruction
    extensions.

    Select one of the /ARCHITECTURE qualifier options shown in the
    following table.

    X86=option   Similar to Clang's -march option.

    GENERIC   Ignored on X86.

    HOST   Ignored on X86.

    ITANIUM2   Ignored on X86.

    EV4   Ignored on X86.

    EV5   Ignored on X86.

    EV56   Ignored on X86.

    PCA56   Ignored on X86.

    EV6   Ignored on X86.

    EV68   Ignored on X86.

    EV7   Ignored on X86.

/FIRST_INCLUDE
-include <file> - include file before parsing.

/UNDEFINE
-U <macro> - undefine macro <macro>.

/LINE_DIRECTIVES
-P - disable linemarker output in preprocessing mode.

/UNSIGNED_CHAR for the compiler is respective to:
-fno-signed-char - char is unsigned.
-fsigned-char - char is signed.

/COMMENTS
-C - include comments in preprocessed output.

/ERROR_LIMIT

    /ERROR_LIMIT
    /ERROR_LIMIT[=number]

/NOERROR_LIMIT Limits the number of error-level diagnostic messages that are acceptable during program compilation. Compilation terminates when the limit (number) is exceeded. /NOERROR_LIMIT specifies that there is no limit on error messages. The default is /ERROR_LIMIT=20, which specifies that compilation terminates after issuing 20 error messages.

-ferror-limit=n - stop emitting diagnostics after n errors have been produced.

/OBJECT
-o <file> - write output to the file. Default object file has .obj extension.

/PREPROCESS_ONLY
-E - only run the preprocessor.

/RTTI
/RTTI (D)
/NORTTI
Enables or disables support for RTTI (runtime type identification) features: dynamic_cast and typeid. Disabling runtime type identification can also save space in your object file because static information to describe polymorphic C++ types is not generated. The default is to enable runtime type information features and generate static information in the object file. The /RTTI qualifier defines the macro __RTTI. Note that specifying /NORTTI does not disable exception handling.

Clang has similar options
-fno-rtti - disable generation of rtti information.
-frtti - enable generation of rtti information(default).

Clang does not define the __RTTI macro.

/WARNINGS

/WARNINGS
/WARNINGS[=(option[,...])]
/WARNINGS (D)
/NOWARNINGS

Controls the issuance of compiler diagnostic messages and lets you modify the severity of messages.

The default qualifier, /WARNINGS, outputs all enabled warning.
The /NOWARNINGS qualifier suppresses warning messages.

The message-list in the following table of options can be any one of the following:

- A single message identifier in double quotes (within parentheses, or not).
- A comma-separated list of message identifiers in double quotes, enclosed in parentheses.
- The keyword "all".

The options are processed and take effect in the following order:

NOWARNINGS	Suppresses warnings. Similar to Clang's -w option.
NOINFORMATIONALS	Has no effect.
ENABLE=message-list	Enables issuance of the specified messages. Can be used to enable specific messages that normally would not be issued when messages disabled with /WARNINGS=DISABLE. Specify "all" to enable all warnings. Similar to Clang's -W<warning> option.
DISABLE=message-list	Disables issuance of the specified

```
                        messages. Specify "all" to suppress all
                        warnings.
                        Similar to Clang's -Wno-<warning> option.

 INFORMATIONALS=message-Has no effect.
 list

 WARNINGS=message-      Don't error out on the specified warnings.
 list
                        Similar to Clang's -Wno-error=<warning> option.

 [NO]ANSI_ERRORS        Has no effect.
 [NO]TAGS               Has no effect

 ERRORS=message-list    Sets the severity of the specified
                        messages to Error.
                        Similar to Clang's -Werror=<warning> option.
```

IMPORTANT NOTE
CXX uses Clang message IDs. For example, if Clang issues a warning like

"warning: non-void function does not return a value [-Wreturn-type]"

The ID of message is "return-type".

All message IDs should be given to CXX as they are in double quotes to preserve the case. This is true for the keyword "all" as well.

/EXCEPTIONS
-fexceptions - enable support for exception handling.
-fno-exceptions - disable support for exception handling.

/OPTIMIZE
-O0, -O1, -O2, -O3, -Ofast, -Os, -Oz, -Og, -O, -O4

Specify which optimization level to use:

`-O0` Means "no optimization": this level compiles the fastest and generates the most debuggable code.

`-O1` Somewhere between `-O0` and `-O2`.

`-O2` Moderate level of optimization which enables most optimizations.

`-O3` Like `-O2`, except that it enables optimizations that take longer to perform or that may generate larger code (in an attempt to make the program run faster).

`-Ofast` Enables all the optimizations from `-O3` along with other aggressive optimizations that may violate strict compliance with language standards.

`-Os` Like `-O2` with extra optimizations to reduce code size.

`-Oz` Like `-Os` (and thus `-O2`), but reduces code size further.

`-Og` Like `-O1`. In future versions, this option might disable different optimizations in order to improve debuggability.

`-O` Equivalent to `-O2`.

`-O4` and higher

Currently equivalent to `-O3`

1.7 See also:
In the following links is the full docuentation about Clang compiler:
https://releases.llvm.org/10.0.0/tools/clang/docs/index.html

https://releases.llvm.org/10.0.0/tools/clang/docs/ClangCommandLineReference.html