OpenVMS x86-64 VSI C++ Release Notes
Version: 230731

*** IMPORTANT ***
Starting from 230718 version all upcoming kits will require OpenVMS V9.2-1.
Please upgrade your systems.

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 the CXX demangling tool:

1. CLANG - this compiler has a UNIX-style command line and is invoked as a OpenVMS foreign-command. To use this compiler, you will need to call the SYS$STARTUP:CXX$STARTUP.COM command file to set up the CLANG DCL symbol.

2. CXX - this compiler has a DCL command line that is compatible with the C++ compiler on OpenVMS IA64. It currently does not support all possible qualifiers. Unsupported qualifiers are silently ignored. See section 1.5 for a summary of the qualifiers.

   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.

3. CXX$DEMANGLE - the demangling tool called CXX$DEMANGLE. It is invoked as a OpenVMS foreign-command. The SYS$STARTUP:CXX$STARTUP.COM command file defines the CXX$DEMANGLE DCL symbol. The tool is taken from LLVM tool-set known as llvm-cxxfilt and uses a UNIX-style command line. For more information:

   ```
   $ CXX$DEMANGLE --help
   ```

The kit contains a log file of changes for this release and for prior releases.

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

The compiler behaves very much like the Linux version of the clang compiler in terms of language features. The primary differences is the support of the OpenVMS-specific features, 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__.

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.

For example,

```
// HW.CXX
#include <iostream>

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

$ CXX HW.CXX
$ LINK HW ! Link against installed shared libraries
$ RUN HW
Hello World!
$ LINK HW,SYS$EXAMPLES:CXX/OPT ! Link against static libraries
$ RUN HW
Hello World!
```

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). Programs should not notice the difference. The linker creates small trampolines in 32-bit address space so the address of a routine will still fit into a 32-bit variable.

1.1 OpenVMS Specific Pragmas

The compilers have been extended to support OpenVMS specific pragmas which include:

```
  #pragma pointer_size [options]
  #pragma required_pointer_size [options]
  #pragma [no]member_alignment [options]
  #pragma extern_model [options]
  #pragma extern_prefix [options]
  #pragma include_directory <string-literal>
```

Other OpenVMS pragmas are processed but only have partial/limited support at present. The #pragma message 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 (or -pointer-size with the Linux command line) is given. This matches the behavior of the VSI C++ Itanium compiler.

  Neither /POINTER_SIZE (-pointer-size) nor #pragma [required_]pointer_size has any effect on vptr. The size of virtual pointer is always 64 bit.

* The option provided to #pragma nomember_alignment [option] has a meaning of base alignment of the structure. If provided appropriate alignment attribute is set on the structure declaration.

  There are some differences from IA64 CXX member alignment:

  * In IA64 CXX:

```
    struct
    /*
     ** This directive does not change the alignment of s1 because we've
     ** already started the struct.
     */
    #pragma nomember_alignment
```

```
     {
     ...
     } s1;

    But on OpenVMS x86, it does change the alignment.  It is best to use
    the pragma prior to the struct definition.

  * In IA64 CXX the #pragma pack cancels the base alignment defined by

    #pragma nomember_alignment <base_alignment>

    But on OpenVMS x86, #pragma pack has no effect on base alignment.


* We have support for #pragma extern_model, that includes
 strict_refdef, common_block, relaxed_refdef, globalvalue, save and restore options support.

  The following attributes are available:

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

 and alignments:

  byte
  word
  long
  quad
  octa

* OpenVMS style #pragma message is also available. Currently, we support only

#pragma message [option1] ("message-list")

format.

Parameter [option1] must be one of the following keywords:
  enable - Enables issuance of the messages specified in the message-list
  disable - Disables issuance of the messages specified int the message-list
  error - Sets the severity of each message in the message list to Error.
  fatal - Sets the severity of each message in the message list to Fatal.
  informational/warning - Sets the severity of each message in the message list to Warning.
  save - Saves the current state of the compiler messages.
  restore - Restores the saved state of the compiler messages.

Example:
    // example.cpp
    #pragma message enable ("unused-variable")
    #pragma message disable ("return-type implicitly-unsigned-literal")
    int foo() {
      long long a = 12341234123412341234;
      int b = a;
    }

So with this example you should get something like this warning
  "warning: unused variable 'a' [-Wunused-variable]"

Values for (message-list) is the name following the message severity code letter.
```


In this example "unused-variable" message is enabled and
"return-type" and "implicitly-unsigned-literal" are disabled. The values in
message-list should be space separated.

NOTE: Itanium CXX compiler's message identifiers not supported (silently ignored),
you should use clang's respective message ids. Also, it's impossible
to reduce message severity to warning after it was set to error or fatal,
but it's available to reduce from fatal to error.

1.2 TLB And Headers

The C++ compiler now contains support to search and use text-libraries (TLB)
with the /LIBRARY qualifier and with automatic searches for headers performed
by the compiler. Prior C++ kits on OpenVMS V9.2 systems contained private
copies of headers with C++ specific changes. All of these changes are now
in the V9.2-1 system-supplied TLB files and the C++ kit no longer provides
these special headers. With this change, the C++ kit isn't supported on V9.2
systems since it would use system-supplied headers without the proper C++ support.
This would result in compilation failures and incorrect run-time behavior.

Installation places C++ standard library headers in the
SYS$COMMON:[VSICXX$LIB.INCLUDE.LIB_CXX] directory.

The DCL CXX command knows the location of these headers. The CLANG DCL symbol
defined by the SYS$STARTUP:CXX$STARTUP.COM command file also includes
the location of these headers.

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.

The compiler does not automatically truncate external names longer than 31
characters. External names can be of any length. There are symbol length
limits in the LIBRARIAN and in LINKER options files but in general the LINKER
doesn't care about symbol lengths. The /NAMES=TRUNCATED option can be used
to revert to the IA64 C++ behavior.

1.4 Known Issues

- 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. We strongly encourage that you install "V9.2 update 2".

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

- 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.

- 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	-	+
/LIBRARY	-	+
/L_DOUBLE_SIZE	-	+
/MEMBER_ALIGNMENT	-	+
/NAMES	-	+
/OBJECT	-	+
/OPTIMIZE	-	+
/POINTER_SIZE	-	+
/PREPROCESS_ONLY	-	+
/RTTI	-	+
/STANDARD	-	+
/UNDEFINE	-	+
/WARNINGS	-	+
/NESTED_INCLUDE_DIRECTORY	-	+
/VERBOSE	-	+
/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	+	-
/LIST	+	-
/LINE_DIRECTIVE	+	-
/LOOKUP	+	-
/MACHINE_CODE	+	-
/MAIN	+	-
/MMS_DEPENDENCIES	+	-
/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 Using Clang Command Options from DCL Command Line

We have addded a new qualifier that allows you to pass clang
command line options when there is no equivalent DCL qualifier.

```
  /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
```

1.7 Mapping Clang Command Options to DCL Command Options

This sections shows Clang options that are similar to CXX DCL qualifiers.

```
  /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

  Also supported the CXX$SYSTEM_INCLUDE, CXX$LIBRARY_INCLUDE and CXX$USER_INCLUDE logical names.
  If some of them are defined then the compiler will do header search also in the directories
  mentioned by the logicals in the same order as placed above.
```

```
Please note that that CXX$USER_INCLUDE will also impact angle-bracket include directives,
that is NOT the same behavior as in IA64 CXX. Also the /ASSUME qualifier is not supported yet.

/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|argv64}
  * "argv64" means:
    - The main argument argv will be an array of 64-bit pointers. The default
      is an array of 32-bit pointers
    - 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
  * "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=(UPPERCASE,AS_IS)
-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.

/NAMES=(TRUNCATED,SHORTENDED)
-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 "$".

  By default, external names can be of any length.  Itanium C++ defaults
  to /NAMES=TRUNCATED.

/[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.

/LIBRARY
-text-library=<TEXT_LIBRARY_PATH>
Clang's option to add text library to include search path.

/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 OpenVMS does 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 processor instruction set to be
    used by the compiler.

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

    X86=option   Similar to Clang's -march option.
                        For example
                        $ cxx /arch=x86="skylake" hw.cxx

    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.

  /NESTED_INCLUDE_DIRECTORY

        /NESTED_INCLUDE_DIRECTORY
        /NESTED_INCLUDE_DIRECTORY[=option]
        /NESTED_INCLUDE_DIRECTORY=INCLUDE_FILE (D)

     Controls the first step in the search algorithm the compiler
     uses when looking for files included using the quoted form of
     the #include preprocessing directive: #include "file-spec" The
     /NESTED_INCLUDE_DIRECTORY qualifier has the following options:

     Option     Usage

     PRIMARY_   Directs the compiler to search the default file type
     FILE       for headers using the context of the primary source
                file. This means that only the file type (".H" or ".")
                is used for the default file-spec but, in addition,
                the chain of "related file-specs" used to maintain
                the sticky defaults for processing the next top-level
                source file is applied when searching for the include
                file. This is not supported for X86.
     INCLUDE_   Directs the compiler to search the directory
     FILE       containing the file in which the #include directive
                itself occurred. The meaning of "directory containing"
                is: the RMS "resultant string" obtained when the file
                in which the #include occurred was opened, except that
                the filename and subsequent components are replaced
                by the default file type for headers (".H", or just
                "." if /ASSUME=NOHEADER_TYPE_DEFAULT is in effect,
                /ASSUME is not supported for X86).
                The "resultant string" will not have translated any
                concealed device logical.
     NONE       Directs the compiler to skip the first step of
                processing #include "file.h" directives. The compiler
                starts its search for the include file in the
                /INCLUDE_DIRECTORY directories.

     For more information on the search order for included files, see
     the /INCLUDE_DIRECTORY qualifier.

  /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:

 o  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-list	Has no effect.
WARNINGS=message-list	Don't error out on the specified warnings. 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.

There is a list of warnings in clang that are turned off by default and are enabled with their specific ID or with the keyword "all".
The VMS-specific warning "cast from long pointer to short pointer will lose data" is off by default. It can be enabled with the ID "may-lose-data" or with "all".

/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`

  /VERBOSE
    With this qualifier the driver first prints out the fully generated
    command line which is going to Clang, and then runs the compiler

1.8 Threads and variable sharing
  OpenVMS does not currently support thread_local/__thread/_Thread_local
  declaration specifiers. As alternative use POSIX API, functions
  pthread_key_create, pthread_setspecific, pthread_getspecific,
  pthread_key_delete might be a good place to start.

1.9 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
```