MPLAB® XC8 C Compiler Version 2.32 Release Notes for PIC® MCU

Includes the MPLAB XC8 PIC Assembler

THIS DOCUMENT CONTAINS IMPORTANT INFORMATION RELATING TO THE MPLAB XC8 C COMPILER WHEN TARGETING MICROCHIP PIC DEVICES. PLEASE READ IT BEFORE RUNNING THIS SOFTWARE.

SEE THE MPLAB XC8 C COMPILER RELEASE NOTES FOR AVR DOCUMENT IF YOU ARE USING THE COMPILER FOR 8-BIT AVR DEVICES

1. Overview

1.1. Introduction

This release of the Microchip MPLAB® XC8 C compiler contains new features, bug fixes, and new device support.

1.2. Release Date

The official release date of this compiler version is the 1 February 2021.

1.3. Previous Version

The previous MPLAB XC8 C compiler version was 2.31, released 12 October 2020.

1.4. Functional Safety Manual

A Functional Safety Manual for the MPLAB XC compilers is available in the documentation package when you purchase a functional safety license.

1.5. Component Licenses and Versions

Components of the MPLAB® XC8 C Compiler for PIC MCUs tools are written and distributed under the LLVM Release License, detailed in the file named `LLVM_LICENSE.txt`, located the `docs` subdirectory of your install directory.

This compiler uses an implementation of Clang version 4.0.1 based upon LLVM 4.0.1.

1.6. System Requirements

The MPLAB XC8 C compiler and the licensing software it utilizes are available for a variety of operating systems, including 64-bit versions of the following: Professional editions of Microsoft Windows 7, and Windows 10; Ubuntu 16.04; and Mac OS X 10.15.1. Binaries for Windows have been code-signed. Binaries for Mac OS X have been code-signed and notarized.

If you are running a network license server, only computers with operating systems supported by the compilers may be used to host the license server. As of xclm version 2.0, the network license server can be installed on a Microsoft Windows Server platform, but the license server does not need to run on a server version of the operating system.

1.7. Devices Supported

This compiler supports all known 8-bit PIC® MCU devices at the time of release. See `pic_chipinfo.html` (in the compiler's `doc` directory) for a list of all supported baseline and mid-range devices and `pic18_chipinfo.html` for a list of all supported PIC18 devices. These files also list configuration bit settings for each device.

1.8. Editions and License Upgrades

The MPLAB XC8 compiler can be activated as a licensed (PRO) or unlicensed (Free) product. You need to purchase an activation key to license your compiler. A license allows for a higher level of optimization compared to the Free product. An unlicensed compiler can be operated indefinitely without a license.

An MPLAB XC8 Functional Safety compiler must be activated with a functional safety license purchased from Microchip. The compiler will not operate without this license. Once activated, you can select any optimization level and use all the compiler features.

1.9. Installation and Activation

See also the Migration Issues and Limitations sections for important information about the latest license manager included with this compiler.

If using MPLAB IDE, be sure to install the latest MPLAB X IDE before installing this tool. Quit the IDE before installing the compiler. Run the `.exe` (Windows), `.run` (Linux) or `.app` (OS X) compiler installer application, e.g. `XC8-1.00.11403-windows.exe` and follow the directions on the screen. The default installation directory is recommended. If you are using Linux, you must install the compiler using a terminal and from a root account. Install using a Mac OS X account with administrator privileges.

Activation is now carried out separately to installation. See the document *License Manager for MPLAB® XC C Compilers* (DS52059) for more information.

If you choose to run the compiler under the evaluation license, you will now get a warning during compilation when you are within 14 days of the end of your evaluation period. The same warning is issued if you are within 14 days of the end of your HPA subscription.

Note that as of MPLAB XC8 version 1.34, the XC Network License Server is a separate installer and is not included in the single-user compiler installer.

Note also that the use of MPLAB XC8 version 1.34 with MPLAB IDE v8 is now deprecated. DLL files needed by this IDE are no longer installed with the compiler.

The XC License Manager now supports roaming of floating network licenses. Aimed at mobile users, this feature allows a floating license to go off network for a short period of time. Using this feature, you can disconnect from the network and still use your MPLAB XC compiler. See the doc folder of the XCLM install for more on this feature.
MPLAB X IDE v1.40 includes a Licenses window (Tools > Licenses) to visually manage roaming.

1.9.1. Resolving Installation Issues

If you experience difficulties installing the compiler under any of the Windows operating systems, try the following suggestions.

- Run the install as an administrator.
- Set the permissions of the installer application to 'Full control'. (Right-click the file, select Properties, Security tab, select user, edit.)
- Set permissions of the temp folder to ‘Full Control’.

To determine the location of the temp folder, type `%temp%` into the Run command (Windows logo key + R). This will open a file explorer dialog showing that directory and will allow you to determine the path of that folder.

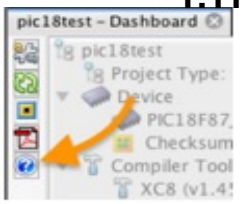
1.10. Compiler Documentation

There are several versions of the compiler user's guide, each customised for a particular use of the compiler. The guides described below can be opened from the HTML page that opens in your browser when clicking the blue help button in MPLAB X IDE dashboard, as indicated in the screenshot.

If you are building for 8-bit PIC targets and you are using the new `xc8-cc` driver, the MPLAB® XC8 C Compiler User's Guide for PIC® MCU contains information on the new-format compiler options and C99 features that are applicable to this architecture.

If you are building legacy projects for 8-bit PIC targets or you using the original `xc8` driver, the legacy MPLAB® XC8 C Legacy Compiler User's Guide contains information on the old-style compiler options and C90 features that are applicable to this architecture.

1.11. Customer Support



Common problems are explained in the FAQ list. You can also ask questions of other users of this product in the XC8 Forum.

Microchip welcomes bug reports, suggestions or comments regarding this compiler version. Please direct any bug reports or feature requests via the Support System.

At times, advisory message 1395 may be issued by the compiler. This message is part of a new testing process. The compiler will display this message if it encounters a specific code sequence that results in internal compiler templates being used in a unique way. This message does not imply a bug in the generated code; however, the code sequence encountered could be used to further improve the compiler's performance. If you wish to participate by contributing the code that generated this message, you are welcome to send the project to Support; otherwise, you may ignore this message.

2. Documentation Updates

For on-line and up-to-date versions of MPLAB XC8 documentation, please visit Microchip's Online Technical Documentation website.

The *MPLAB® XC8 C Compiler User's Guide for PIC® MCU* (DS50002737), which describes the operation and options associated with the new `xc8-cc` driver and the C99 compliant compiler features, is currently at revision C. This document has been migrated to a new authoring and publication system; you may see differences in the formatting compared to previous revisions.

If you need assistance with the options for the previous `xc8` driver or with C90-specific information, the *MPLAB® XC8 C Legacy Compiler User's Guide* (DS50002053) is available.

The *MPLAB XC8 Getting Started Guide* (DS50002173) has been pulled from this distribution. If you are just starting out with 8-bit PIC devices and the MPLAB XC8 C Compiler, the more up-to-date *MPLAB® XC8 User's Guide for Embedded Engineers - PIC MCUs* (DS50002400) has information on setting up projects in the MPLAB X IDE and writing code for your first MPLAB XC8 C project. This guide is available on Microchip's website.

The *MPLAB® XC8 PIC Assembler User's Guide for Embedded Engineers* guide has been updated to revision B in this compiler release.

The following sections provide corrections and additional information to that found in the user's guide shipped with the compiler.

2.1. Interrupts and Bits Example For PIC18 Devices

The example code shown shown in Section 8, *Interrupts and Bits Example For PIC18 Devices*, of the *MPLAB® XC8 User's Guide for Embedded Engineers - PIC MCUs* uses the wrong psect for the interrupt function. The address of interrupt functions when using the interrupt vector table must be a multiple of 4, since this address is shifted by 2 bits in the vector table.

In the example, the interrupt function begins:

```
PSECT code
tmr0Isr:
```

The `code` psect is a predefined psect with a `reloc` (relocation) value of 2. The `reloc` value should be 4, so a user-defined psect must instead be used. For example:

```
PSECT textISR,class=CODE,reloc=4
tmr0Isr:
```

2.2. Functions

The *MPLAB® XC8 C Compiler User's Guide for PIC® MCU* does not indicate that an empty implementation of the `getch()` function has been provided that can be customised for use in the scanf family of functions. Customize this function as required, then include the source file into your project. This function returns an `int` when used in C99 projects, but a `char` when used in C90 projects.

2.3. Hexmate Hash Calculations

The following description contains new features in Hexmate, that are not mentioned in the *MPLAB® XC8 C Compiler User's Guide for PIC® MCU*.

The `-ck` Hexmate option is used to calculate a hash value. The usage of this option is:

`-ck=`*`start-end`*`@`*`dest`*`[+`*`offset`*`][w`*`Width`*`][t`*`Code`*`[.`*`Base`*`]][g`*`Algorithm`*`][p`*`Polynomial`*`][r`*`revWidth`*`][s`*`skipWidth`*`[.`*`skipBytes`*`]]`

where:

`start` and *`end`* specify the hexadecimal address range over which the hash will be calculated. If these addresses are not a multiple of the data width for checksum and Fletcher algorithms, the value zero will be padded into the relevant input word locations that are missing.

`dest` is the hexadecimal address where the hash result will be stored. This address cannot be within the range of addresses over which the hash is calculated.

`offset` is an optional initial hexadecimal value to be used in the hash calculations. It is not used with SHA algorithms.

`Width` is optional and specifies the decimal width of the result. Results can be calculated for byte-widths of 1 to 4 bytes for most algorithms, but it represents the bit width for SHA algorithms. If a positive width is requested, the result will be stored in big-endian byte order. A negative width will cause the result to be stored in little-endian byte order. If the width is left unspecified, the result will be 2 bytes wide and stored in little-endian byte order. This width argument is not required with any Fletcher algorithm, as they have fixed widths, but it may be used to alter the default endianism of the result.

`Code` is a hexadecimal code that will trail each byte in the result. This can allow each byte of the hash result to be embedded within an instruction, for example `t34` will embed each byte of the result in a `retlw` instruction on Mid-range devices, or `t0000` will append two 0x00 bytes to each byte of the hash. This code sequence can be optionally followed by `.`*`Base`*, where *`Base`* is the number of bytes of the hash to be output before the trailing code sequence is appended. A specification of `t1122.2`, for example, will output the bytes 0x11 and 0x22 after each two bytes of the hash result.

`Algorithm` is a decimal integer to select which Hexmate hash algorithm to use to calculate the result. If unspecified, the default algorithm used is 8-bit checksum addition (algorithm 1).

`Polynomial` is a hexadecimal value which is the polynomial to be used if you have selected a CRC algorithm.

`revWidth` is a decimal word width. If this is non-zero, then bytes within each word are read in reverse order when calculating a hash value. Words are aligned to the addresses in the HEX file. At present, the width must be 0 or 2. A zero width disables the reverse-byte feature, as if the `r` suboption was not present. This suboption should be used when using Hexmate to match a CRC produced by a PIC hardware CRC

module that use the Scanner module to stream data to it.

skipWidth is a decimal word width. If this is non-zero, then the MSB within each word is skipped for the purposes of calculating a hash value. Words are aligned to the addresses in the HEX file. At present, the width must be 0 (which disables the skip feature, as if the `s` suboption was not present) or greater than 1. This value can be optionally followed by `.`*skipBytes*, where *skipBytes* is a number representing the number of bytes to skip in each word, for example `s4.2` will skip the two most significant bytes in each 4-byte word. To avoid including the 'phantom' 0x00 byte added to HEX files by the MPLAB XC16 C Compiler in hash calculations, use `s4`.

A typical example of the use of the checksum option is:

```
-ck=0-1FFF@2FFE+2100w-2g2
```

This will calculate a checksum over the range 0 to 0x1FFF and program the checksum result at address 0x2FFE. The checksum value will be offset by 0x2100. The result will be two bytes wide and stored in little-endian format.

Note that the reverse and skip features act on words that are aligned to the HEX file addresses, not to the position of the data in the sequence being processed. In other words, they are not affected by the start and end addresses specified in the `-ck` option. Consider this option:

```
-ck=0-5@100w2g5p1021s2
```

which specifies that every second byte be skipped over HEX addresses 0 thru 5. If it is acting on the HEX record (data underlined):

```
:10000000 64002500030A7500760077007800 64001C
```

the hash will be calculated from the (hexadecimal) bytes 64, 25, and 03. Processing the same HEX record with an option that uses a different start and end address range (1 thru 6):

```
-ck=1-6@100w2g5p1021s2
```

the hash will be calculated from the (hexadecimal) bytes 25, 03, and 75. These features attempt to mimic data read limitations of code running on the device, and thus the words they use are aligned with device addresses, which are in turn aligned to HEX file addresses.

2.4. Assembler-provided Psects

Table 5.2 in the *MPLAB® XC8 PIC® Assembler User's Guide*, which provides the names and classes of psect that are provided by the PIC assembler, incorrectly states that the name of the psect that should be used for EEPROM data is `eedata`. The provided psect is called `edata`, but as stated, it is associated with the class `EEDATA`.

3. What's New

The following are new PIC-target features the compiler now supports. The version number in the subheadings indicates the first compiler version to support the features that follow.

3.1. Version 2.32

Stack Guidance Available with a PRO compiler license, the compiler's new stack guidance feature can be used to estimate the maximum depth of the stacks used by a program. It constructs and analyses the program's call graph, determines the stack usage of each function, and produces a report, from which the depth of the program's stacks can be inferred.

This feature is enabled through the `-mchp-stack-usage` command-line option. A summary of stack

usage is printed after execution. A detailed stack report is available in the map file, which can be requested in the usual way.

New device support Support is available for the following PIC parts: 16F15245, 16F15225, 18F04Q41, 18F04Q40, 18F15Q41, 18F15Q40, 18F05Q40, 18F05Q41, 18F14Q40, and 18F14Q41.

3.2. Version 2.31

New device support Support is available for the following PIC parts: 18F16Q40, 18F06Q40, 18F16Q41, and 18F06Q41.

3.3. Version 2.30

Hexmate hash calculations Hexmate has two new hash-calculation features. Bytes in the HEX file can be skipped for the purposes of calculating a hash value by using a new 's' argument to Hexmate's `-CK` option. This would be useful when there is data in the Hex file that is not present on the device, such as padding bytes added by the compiler. Another extension to the `-CK` option's 't' argument allows the trailing code sequence to be appended to a specified number of bytes in the hash, not just to each byte. This would be useful where the target device cannot read every byte of program memory and the hash value has to be padded.

3.4. Version 2.29 (Functional Safety Release)

Header file for compiler built-ins To ensure that the compiler can conform to language specifications such as MISRA, a new `<builtins.h>` header file, which is automatically included by `<xc.h>`, has been added. This header contains the prototypes for all in-built functions, such as `__nop()` and `_delay()`. Some built-ins may not be MISRA compliant; these can be omitted by adding the define `__XC_STRICT_MISRA` to the compiler command line. The built-ins and their declarations have been updated to use fixed-width types.

3.5. Version 2.20

New device support Support is available for the following PIC parts: 16F15213, 16F15214, 16F15223, 16F15224, 16F15243,16F15244, 18F25Q43, 18F45Q43, 18F55Q43, 18F26Q43, 18F46Q43, and 18F56Q43. The following device names may now be additionally used for existing devices: RFPIC12C509AF, RFPIC12C509AG, RFPIC12F675F, RFPIC12F675H, RFPIC12F675K.

Complete 64-bit application set All applications for all platforms are now 64-bit applications. This covers all compiler and utility applications and on the Windows and Linux platforms.

Updated XCLM The license manage utilities have been updated to version 2.28. This version fixes bugs and is a 64-bit build.

In-built messages The compiler warnings and error messages, which are contained in a separate file, have now also been built into most compiler applications and will be used when the message description file cannot be found. This will mostly benefit the use of applications like Hexmate, which are often run independently to the compiler driver.

New byte ordering option for Hexmate hash calculations A new `revword=`*`n`* suboption to the compiler driver's `-mchecksum` option requests Hexmate to read bytes in reverse order within *`n`*-byte words in the hex file for the purposes of calculating a hash, such as a checksum or CRC. This allows Hexmate to produce a result that will match that produced by devices that use the Scanner module to stream data to the CRC module in order to produce a CRC at runtime. Currently only word widths of 2 are supported.

(Using the suboption but specifying *n* as 0 will also disable this reverse feature). If you are using Hexmate directly, use the corresponding `rn` suboption in Hexmate's `-CK` option.

SHA-ZAM The SHA256 algorithm has been added to Hexmate's suite of hash algorithms. It is accessible as algorithm #10 from Hexmate's `-CK` option, or from the compiler's `-mchecksum` option.

Type 3 HEX records Hexmate now processes type 3 records in HEX files, which might be produced when writing bootloaders for 8-bit AVR devices. These records are output verbatim in the final HEX file. A warning will be produced if more than one type 3 record is encountered and the record data is not consistent.

Replacing compiler-generated printf routines Code for the printf family of functions is generated with each build by the compiler, based on how those functions were used in your source code. Previously, it has not been possible to use your own versions of these functions, but the compiler now places the generated printf code into a temporary library, so that the usual library override features of the linker can be used. Any function defined by your code and whose name matches that of a printf-family function will be used in preference to the compiler-generated library routine.

SHA summary The memory summary option `sha256` is now available. It works in the same way as the existing `sha1` option, except it uses the SHA256 algorithm when creating a hash of the HEX file.

Movff errata workaround The PIC18(L)F27/47/57K42 family of devices in silicon revisions A1, A3 suffer from a fault triggered when the `movff` instruction is used while BSR is set to 63. The compiler can now employ work-arounds (via the `-merrata` option) to avoid this issue when compiling C code. It will also attempt to detect potential issues in hand-written assembly and issue a warning.

Multi-content library archives Library archive files (`.a` extension) may now contain any mix of p-code (`.p1`) or assembler object (`.o`) modules. Previously, an archive could contain modules of only one type.

Exclusion of AVR ASM The AVR ASM assembler for 8-bit AVR devices is no longer shipped as part of this compiler distribution.

New stand-alone assembler The MPASM assembler for 8-bit PIC devices is no longer supported and is not shipped as part of this compiler distribution. In its place is the new MPLAB XC8 PIC Assembler (PIC Assembler), which is based on the assembler application used by the XC8 compiler package and which allows assembly-only projects to be built. The PIC Assembler is not code compatible with MPASM, although some MPASM features have been added to it to ease migration of MPASM source code, should that be necessary. It is recommended that you continue to use MPASM for legacy assembly projects, but that new projects be written for the PIC Assembler. A new user's guide and example guide for embedded engineers are now available and are shipped with this distribution. A migration guide is available to assist with migration to the PIC Assembler for projects that are still being actively developed and might need to use future devices or assembler features.

The following changes are related to this new assembler.

New assembler command-line driver A new assembler driver, `pic-as`, is bundled with this distribution in the `pic-as/bin` directory and can be invoked to perform all aspects of the build, including preprocessing, assembly, and link steps. Its use is the recommended way to invoke the assembler, as it hides the complexity of all the internal applications and provides an interface consistent with the `xc8-cc`, the XC8 C compiler command-line driver. Assembly-only projects can be created in the MPLAB X IDE, which will then us the `pic-as` driver when building.

PIC18 extended instruction set support The assembler supports the extended PIC18 instruction set.

Assembly-only projects can enable the extended instruction set (using the XINST configuration bit) and be written for this mode. Note that although the assembler application invoked by PIC-AS is the same as that invoked by the MPLAB XC8 C compiler, the code generators in the compiler still do not support the extended instruction set and produce code that will only execute correctly on devices running the standard instruction set.

Specifying configuration bits in assembler code It is now possible to specify a device's configuration bit settings in assembly code using a new CONFIG assembler directive. The arguments to this directive are the same as those to the #pragma config directive, which is still usable in C source code, and consist of setting/value pairs. The MPLAB X IDE can assist with the generation of code that can be copied into your assembly source.

Expanded instruction syntax The use of the ,0 and ,1 operands with many PIC instructions is now permitted. These can be used to indicate the destination with file register instructions, the use of the access bank with PIC18 file register instructions, and the call/return mode with call, return, and retfie PIC18 instructions. The previously-used and more readable operands are still available, but you should not mix the style of operand in a single instruction.

New assembler directives The MESSG, ERROR, and RADIX directives have been added to the assembler. Their use and function are equivalent to their namesakes in the MPASM assembler.

Provided psect and class definitions As well as SFR and other device-specific information, the assembler will provide psect (section) definitions as well as linker classes when <xc.inc> has been included. The names of the psect are related to the directives used by MPASM that place content into similar sections. For example the PSECT code directive would act in a similar way to the CODE directive used with MPASM.

Call graph options The pic-as driver option -mcallgraph=*style* has been implemented to allow the selection of the style of call graph printed in the map file, which might be useful if you use a compiled stack in assembly projects. The allowable styles are: none, crit (critical paths only), std, or full.

3.6. Version 2.19 (Functional Safety Release)

None.

3.7. Version 2.10

Code Coverage This release includes a code coverage feature that facilitates analysis of the extent to which a project's source code has been executed. Use the option -mcodecov=ram to enable it. After execution of the program on your hardware, code coverage information will be collated in the device, and this can be transferred to and displayed by the MPLAB X IDE via a code coverage plugin. See the IDE documentation for information on this plugin can be obtained.

Expanded interrupt arguments The following keywords may now be used in the __interrupt() specifier: __irq, __base, __default, __low_priority and __high_priority. The non-underscored versions of these keywords are still valid.

New device support Support is available for the following PIC parts: 18F57Q43, 18F47Q43, 18F27Q43, 18F57Q83, 18F56Q83, 18F47Q83, 18F46Q83, 18F27Q83, 18F26Q83, 18F57Q84, 18F56Q84, 18F47Q84, 18F46Q84, 18F27Q84, and 18F26Q84.

3.8. Version 2.05

More bits for your buck The OS X version of this compiler and license manager is now a 64-bit application. This will ensure that the compiler will install and run without warnings on recent versions of Mac OS X.

Less code for no bucks Unlicensed (Free) versions of this compiler now allow optimizations up to and including level 2. This will permit a similar, although not identical, output to what was previously possible using a Standard license. Virtually all code generation optimizations are now enabled regardless of the license type, but most assembler optimizations still require a PRO license for them to be enabled. The `--mode` option to the legacy driver, `xc8`, no longer has any effect.

Expanded long long support Support for 64-bit `long long` types has been expanded to include Enhanced Mid-range devices. These devices, as well as PIC18 devices, can use these types in expressions, but note that their use will greatly increase the amount of code and data memory required by the project.

Wider C99 support You can now use the C99 library with Enhanced Mid-range devices that use the reentrant stack model. Previously with these devices, you were limited to using C99 with the compiled (non-reentrant) stack model.

Larger stack allocation Functions that use the reentrant stack model in Enhanced Mid-range projects were limited to a total of 31 bytes of stack for local objects. This limitation has been lifted and there is now no theoretical limit to how much data a function can define on the stack. Note, however, that exceeding 31 bytes of stack usage will increase the size of generated code for each access of these stack objects by few instructions.

int24_t types added to C99 The `int24_t` and `uint24_t` types (along with the existing `__int24` and `__uint24` types) are now available when using the C99 library and when CCI is not active.

Welcome MPASM The MPASM assembler for 8-bit devices is now included in the XC8 compiler installer, rather than being distributed with the MPLAB X IDE. This assembler is not used by the XC8 compiler, but is available for projects based on hand-written assembly source.

3.9. Version 2.00

Top-level driver A new driver, called `xc8-cc`, is positioned above the previous `xc8` driver, and it can call the appropriate applications based on the selection of the target device. This driver accepts GCC-style options, although the PIC implementation uses the same back end as the previous compiler version. The new driver allows a similar set of options with similar semantics to be used with any PIC target and is thus the recommended way to invoke the compiler. Note that the options used by the new `xc8-cc` driver, unlike those used by the previous `xc8` driver, are case sensitive. If required for legacy projects, the previous driver can be called directly using the old-style options it accepted in earlier compiler versions.

New file extensions When you are using the new driver, the extensions of input and output files are different to those used with the previous driver. The table below shows the new extensions that are used.

File type	Previous extension	New extension
Preprocessed C source	`.pre`	`.i`
P-code library	`.lpp`	`.a`
Object-code library	`.lib`	`.a`
Object	`.obj`	`.o`

Assembly source	.as	.s (.S, .sx)

New librarian driver A new librarian driver is positioned above the previous PIC `libr` librarian. This driver accepts GCC-archiver-style options, which are either translated for or passed through to the librarian being executed. The new driver allows a similar set of options with similar semantics to be used to create or manipulate any PIC library file and is thus the recommended way to invoke the librarian. If required for legacy projects, the previous librarian can be called directly using the old-style options it accepted in earlier compiler versions.

Clang front end The compiler's front end, responsible for preprocessing and parsing the C source code, is now implemented using Clang. This frontend is used when compiling for the C99 standard, regardless of whether you are using the new (`xc8-cc`) or previous (`xc8`) drivers. When using Clang, there might be differences in how source code is preprocessed, and different warning or error messages might be produced during the parsing stage.

C99 support By default, the `xc8-cc` driver will compile for C99 conformance. You can use the `-std` option with this driver to explicitly specify the standard, choosing either `c90` or `c99`. The previous `xc8` driver builds for the C90 standard by default, although you can request the C99 language standard using the `--std` option (note the double dash), in which case the compiler will swap to using Clang as the front end. New types available with the C99 standard include a 64-bit `long long` integer types (currently implemented only for PIC18 devices) and boolean type, but not all other C99 features are yet implemented. Note also that 24-bit floating-point types are not permitted when building for C99.
If you would like to move towards the C99 standard for existing projects but want to minimise any changes to your source code, use the `xc8-cc` driver, set the language standard using `-std=c99`, and use the `-mc90lib` option (if you prefer to use the previous `xc8` driver, use the equivalents: `--std=c99` and `--runtime=+c90lib`). This will select the Clang front end, but use C90-compliant libraries and keep many of the code features (such as 24-bit floating-point types) as they were for the previous compiler. It is recommended that you use the new `xc8-cc` driver for new projects.

C99 library support This initial release has limited C99-compliant libraries; furthermore, these libraries are only available for PIC18 devices and also for Enhanced Mid-range devices that are using the compiled (non-reentrant) stack model. More complete versions of these libraries will be made available in a future release. Note also that some routines in the C99 library will be much larger than their C90 counterparts, and that you might see an increase in data memory, program memory, or stack usage. These routines will be optimized in future releases.
See the user's guide for the available functions, but note that the following functions are only available for PIC18 devices:

```
lldiv_t                 exp2
atoll                   exp2f
strtoll                 exp2l
strtoull                fma
llabs                   fmaf
lldiv                   fmal
time_t                  llrint
difftime                llrintf
mktime                  llrintl
time                    llround
ctime                   llroundf
```

```
gmtime                          llroundl
localtime
```

The following functions are not included for any device:

```
tgamma                          strftime
tgammaf                         tgamma
tgammal
```

Better interrupt response in Free mode The list of registers saved when using the lower optimization levels now more closely matches that with level s/3 (formerly PRO mode).

New device support Support is available for the following PIC parts (and their corresponding LF variants): 16F18455 and 16F18456.

3.10. Version 1.45

New device support Support is available for the following parts: 16LF73, 18F27Q10, 18F47Q10, MCP19128, MCP19132, and MCP19133; and the following parts (and their corresponding LF variants): 16F18424, 16F18425, 16F18426, 16F18444, 16F18445, 16F18446.

3.11. Version 1.44

New device support Support is also available for the following parts: 18F26Q10, 18F45Q10, and 18F46Q10.

3.12. Version 1.43

New device support Support is also available for the following parts (and their corresponding LF variants): 18F25K83, 18F26K83, 18F27K42, 18F47K42, 18F57K42, and 18LF27K42.

3.13. Version 1.42

New register macros (XC8E-142) Preprocessor macros are now provided for all special function registers so that code can be conditionally compiled based on their presence. The macros are defined with the same name as the register and look similar to the following example.

```
#define PORTA PORTA
```

Macros are only provided for entire registers, not bits within those registers.

New device support Support is also available for the following part: MCP19215; and the following parts (and their corresponding LF variants): 16F19155, 16F19156, 16F19175, 16F19176, 16F19185, 16F19186, 18F26K42, 18F45K42, 18F55K42, 18F46K42, and 18F56K42.

3.14. Version 1.41

New device support Support is now available for the following parts: 18F24Q10 and 18F25Q10. Support is also available for the following parts (and their corresponding LF variants): 16F15313, 16F15323, 16F15324, and 16F15344.

MPLAB X IDE library builds Recent versions of the IDE allow you to build library projects when using MPLAB XC8. Information concerning such builds can be found in this wiki article. Note that to perform source-level debugging of code inside a library, you must either copy the library source code to the project using the library (as described in the wiki article) or enable absolute paths in the in the IDE preferences when the library is built. This setting can be found under Embedded > Project Options >

File Path Mode in the IDE's preferences dialog.

New Errata Workaround A new device errata workaround has been added to circumvent issues that relate to the memory regions accessed by table read instructions immediately after reset. This issue might affect the runtime startup code of programs that need to access program memory.

A new `--ERRATA` suboption, `nvmreg`, has been added. The `<errata.h>` header file defines the macro `EERATA_NVMREG`, which can be used in programs to compare against the compiler defined macro `_ERRATA_TYPES`.

New local optimizations setting A new optimization setting has been added to the compiler to allow you to use omniscient code generation (OCG) optimizations with libraries or individual program modules and have the scope of those optimizations restricted to code within those libraries or modules. See the Documentation Updates section for more information.

New pointer target specifiers Two new specifiers, `__ram` and `__rom`, can be used to indicate the acceptable memory space of pointer targets. If a pointer is assigned the address of an object that is not in the memory space specified by these qualifiers, an error will be generated. See the Documentation Update section for more information.

3.15. Version 1.40

New device support Support is now available for the following parts (and their corresponding LF variants): 16F15356, 16F15375, 16F15376, 16F15385, 16F15386, 16F19195, 16F19196, 16F19197, 16F15325, 16F15345, 18F24K42, and 18F25K42.

Overhauled Interrupt Handing As part of the support for the new PIC18Fxxk42 devices, which employ a new Interrupt Controller Module and interrupt vector tables, the way that interrupt functions can be defined has been expanded. The `__interrupt()` specifier, previously part of the Common C Interface (CCI), can now be used with all devices that have interrupts. For those devices using the vector tables, interrupt sources and vector table base addresses can be specified with the `irq()` and `base()` arguments, respectively, and as fully described in *Section 5.9 Interrupts* in the new User's Guide. The interrupt functions for these devices can be define with a function parameter to determine the interrupt source. A new driver option, `--UNDEFINTS`, allows otherwise unimplemented interrupts sources to trigger certain behaviors, and the `default` interrupt source can be used to provide a default interrupt function when using `__interrupt()`.

Improved/expanded in-built delay functions (XC8E-106, XC8E-16) The in-built delay routines (`_delay()`, `_delaywdt()`, `__delay_us()`, and `__delay_ms()`, etc.) have been improved. All devices can now generate a three-deep loop, allowing a maximum delay of 50,463,240 instruction cycles. The watchdog variant of this delay is now available for all devices. Several inaccuracies in the generated delay have been corrected. Delays for enhanced mid-range devices are more efficient and use less temporary variables. The delay times are now no longer affected by the location of temporary variables. The `_delay3()` in-built function is now implemented for enhanced mid-range devices.

New Assembler Controls New `ASMOPT_PUSH` and `ASMOPT_POP` controls allow the state of the assembler optimizer to be saved on to a stack of states and then restored at a later time. They work with the existing `ASMOPT_ON/OFF` controls and are useful when you need to ensure the optimizers are disabled for a small section of code but you are not aware if the optimizers have previously been disabled. For example:

```
OPT ASMOPT_PUSH
OPT ASMOPT_OFF
```

```
;protected code
OPT ASMOPT_POP
```

3.16. Version 1.38

New device support Support is now available for the following parts: MCP19126, 16F15354, 16F15355, 16LF15354, and 16LF15355.

Defines for DCI and DIA data (XC8E-105) Macros are now supplied in the relevant device-specific header files for device information that is contained in the published DCI and DIA tables.

3.17. Version 1.37

New device support Support is now available for the following parts (and their corresponding LF variants): 18F27K40, 18F47K40, and 18F67K40.

CRC hash algorithms available in Hexmate Hexmate can now perform CRC hash calculations on data in the HEX file. See the User's Guide for more information.

3.18. Version 1.36

New device support Support is now available for the following parts (and their corresponding LF variants): 16F18326, 16F18346, 18F46K40, 18F26K40, 18F65K40, and 18F66K40.

New assertion A new assertion macro has been added. It implements a light-weight embedded version of the standard C `assert()` macro, and is used in the same way. The macro is called `__conditional_software_breakpoint()` and, if possible, it attempts to halt program execution via a software breakpoint if its argument is false.

Coverity support files Support files for Coverity are now provided. They can be found in the compiler's `etc/coverity` directory.

New EEPROM access The compiler now has the ability to access EEPROM for those devices that use the NVMREG register set. Access is transparent when using any provided EEPROM routines. A new preprocessor macro `_EEPROM_INT` is defined to indicate the access type available with the selected device. Note that those devices which use the new NVMREG set have no flash library support.

Assembly routines with parameters New assembler macros, accessible after you include `<xc.inc>`, have been added to assist you write assembly reentrant routines which can take parameters and return a value. These routines use the compiler's software stack, and thus can only be written for PIC18 or enhanced mid-range devices. See the section Writing Reentrant Assembly Routines with Parameters in the *MPLAB XC8 Compiler User's Guide*.

3.19. Version 1.35

New device support The 16LF1566 and 16LF1567 parts are now fully supported by this release. Support is also available for the following parts (and their corresponding LF variants): 16F18854, 16F18856, 16F18857, 16F18876, and 16F18877.

New floating-point libraries Replacement libraries have been supplied that contain updated routines to perform 32-bit floating-point operations. These routines produce more accurate results than the previous 32-bit library routines. See the Migration Issues section for important information on how this change might affect existing projects.

New errata workaround The compiler performs a new hardware errata workaround, called `BRKNOP2`, and

which can be controlled using the existing --ERRATA option. The hardware issue affects the ability of some devices to halt execution at a hardware breakpoint that is placed immediately following a branch instruction that branches to that breakpoint location. Enabling this workaround presents the compiler from generating such instructions.

New active-optimization macros The compiler now defines a number of macros that indicate the types of optimization currently employed. These can be used to generate errors, if the optimization level is not that required, or can be used to conditionally control what source code is compiled when using different optimizations.

3.20. Version 1.34

New device support The following parts (and their corresponding LF variants) are now fully supported by this release: 18F24K40, 18F25K40, and 18F45K40.

New license manager and installers XCLM 2.0 has been updated to the latest version of the Reprise License software (version 11.2). The status display option, xclm -status, has been updated to display additional information. During compilation you will now receive a warning when you are within 14 days of the end of your HPA subscription or demo period. The XC Network License Server is now a separate installer and is no longer included in the single-user compiler installer.

'MAXIPIC 'hypothetical device The compiler will terminate compilation if the selected device runs out of program memory, data memory, or EEPROM. A new --MAXIPIC option tells the compiler to generate code for a hypothetical device with the same physical core and peripherals as the selected device, but with the maximum allowable memory resources permitted by the device family. You might choose to use this option if your code does not fit in your intended target device. This option will allow you to see the total memory requirements of your program and give an indication of the code or data size reductions that need to be made to fit the program to the device.

Operating mode fall back A new option, --NOFALLBACK, has been implemented to ensure that the compiler is not executed with an operating mode below that specified by the --MODE option. If the compiler has not been activated to run in the requested mode, an error will be produced and compilation terminate when this option is used. Without this option, the compiler will fall back to either the Standard or Free operating mode if it is not activated to run in the requested mode.

Comparison optimisations Code associated with equality and relational comparisons has been improved in many situations, especially for software stack variables on enhanced mid-range and PIC18 devices.

Automatic resetbits enabling The compiler can now detect if code has accessed the variables; __resetbits, __powerdown and/or __timeout associated with the STATUS register reset bits and will enable the –runtime=+resetbits option automatically. The option can still be enabled manually, if desired.

3.21. Version 1.33

None.

3.22. Version 1.32

New device support The following parts (and their corresponding LF variants) are now fully supported by this release: PIC16F1614, and PIC16F1618. Support is also present for the following devices: MCP19118, MCP19119, MCP19124, and MCP19125. See also renamed devices.

Runtime speed improvements For enhanced mid-range devices, a faster inlined version of memcpy() will

be used when compiling with speed optimizations enabled. For PIC18 devices, some integer math routines now use a faster inlined version when compiling with speed optimizations enabled. More use is made of the PIC18 hardware multiply instruction, including floating point multiply operations. Many general optimizations also have been applied to PIC18 output.

New Free-mode optimization An additional optimization has been added to improve removal of redundant bank selection instructions when using Free mode. The effect of this optimization will only be observable when the assembler optimizers are enabled.

Part support version When compiling, the compiler will indicate a part support version identification banner, for example:

```
Microchip MPLAB XC8 C Compiler (PRO Mode) V1.32
Part Support Version: 1.32 (A)
```

Part-support patches will become available as separate installer and will allow you update this compiler's device-specific information, such as header files and supported devices.

Instruction-invariant optimization setting For PIC18 and enhanced mid-range devices only, a new optimization (`--OPT=invariant`) can be specified. Typically, this optimization might be used when building libraries. Its use ensures that the sequence of instructions generated by the compiler will not change from build to build. This optimization does not affect instruction operands, so the binary image will still vary as the program is modified and objects and code are linked in different locations. When employed, functions are forced to be reentrant and all pointer sizes are made a fixed width. (The `--CP=size` option can be used to change code and function pointer sizes from the default size. Option arguments are: `16`, `24`, and `auto` (default). The compiler will normally use 16-bit wide pointers, but may change it to 24, where appropriate, e.g. invariant optimizations are enabled, the device is a PIC18, and there is more than 0xFFFF words of program memory. The `--CP` option has no effect when this optimization is not used.) Many regular code-reduction optimizations are disabled when the invariant optimization is enabled such that the generated code is typically larger.

Compilation speed improvements Refactoring of commonly used utility routines have seen improvements in speed for all compiler applications. All projects should build faster using this compiler version.

Function pointer simplification For enhanced mid-range devices only, function pointers are now always 16-bits wide and represent the full address of the target function. The compiler no longer uses a jump table when making indirect calls for these devices. The code that uses these pointers has been improved and in many situations is smaller and faster than code generated by the previous compiler version. The limit on the number of functions which can be called indirectly (approximately 120), therefore, is lifted for enhanced mid-range devices.

Memory space usage symbols Symbols are now defined by the linker that represent the lower and upper addresses of the used part of each memory space, e.g. `__Lspace_0`, `__Hspace_1` etc. The numbers represent the memory space number, which are listed in the User's Guide table `SPACE FLAG NUMBERS`. These symbols might be useful when calculating checksums.

Additional checks for initialized definitions If there is more than one external definition for an object and these definitions disagree or more than one is initialized, the behavior is undefined. The compiler now makes an additional check to ensure that the number of and numerical value of initial values are consistent across all definitions. A warning is produced if this is not the case. Note that uninitialized external definitions are implicitly initialized with the value zero (for all object elements or members, where appropriate) and that these values will be considered in this comparison.

Using external memory for program code and const data The compiler's `--RAM` option has been able to

allocate external memory (for those PIC18 devices that support this memory interface) to `far` variables. The `--ROM` option can now be used in a similar way to allocate external memory for use by code and `const` data. Make sure you use the compiler's `--EMI` option to configure the external memory interface to suit your application.

3.23. Version 1.31

New device support The following parts are now fully supported by this release: PIC16LF1554, PIC16LF1559, PIC16F1615, PIC16F1619, PIC16LF1615 and PIC16LF1619.

New speed-enhanced C libraries The standard C libraries now come in space and speed variants, and the corresponding library filenames include 'sp 'or 'sz 'strings, respectively, in their name to differentiate them. Space-optimized libraries are used by default, unless the `--OPT=+speed` option (or MPLAB X IDE project properties equivalent) is used. Multiplication and some string routines have been optimized for speed.

Optimization macros Dependent on the state of the selected optimization (`--OPT` option), the macros `__OPTIMIZE_SPEED__` and `__OPTIMIZE_SIZE__` are now defined by the compiler driver to indicate optimization for speed or space (size), respectively.

3.24. Version 1.30

Software stack The compiler can now allocate stack-based (auto and parameter) variables to a software stack (dynamic allocation to a memory area accessed via a stack pointer register) in addition to the compiled stack (static memory allocation). The compiled stack was used exclusively in prior compiler versions and is still the stack used by the default function model. Allocation to the software stack allows for function reentrancy but is less efficient. Function specifiers (`reentrant`/`software`) or the `--STACK` option can be used to control the stack used for the whole program or for individual functions, if required.

Function profiling support The compiler can now generate function registration code that can be used by a MPLAB® REAL ICE™ In-circuit emulator to provide function profiling. *To see function profiling results, you must use a version of the MPLAB X IDE that supports MPLAB XC8 code profiling*. When function profiling is enabled, the compiler inserts assembly code into the prolog and epilog of each C function that is defined. This code communicates runtime information to the REAL ICE to signal when a function is being entered and when it exits. This information, along with further measurements made by the Power Monitor Board, can provide a record of the energy being used by each function.

Better Dwarves The compiler now uses the DWARF 3.0 Debugging Format Standard in ELF files. Provided you are using a compatible version of MPLAB X IDE and generating an ELF output, this will provide more accurate debugging information in some situations.

New device support The following parts (and their corresponding LF variants) are now fully supported by this release: PIC16F1613, PIC12F1612, PIC16F1717, PIC16F1718, PIC16F1719, PIC16LF1718, PIC16F18323, PIC16F18313, PIC16F18345, and PIC16F18325.

Software Breakpoints Two new builtins have been added to allow the insertion of opcodes that can trigger software breakpoints. These are `__BUILTIN_SOFTWARE_BREAKPOINT()` (unconditional for all builds) and `__debug_break()` (debug builds only). These macros are only available for PIC18 and mid-range devices and are ignored for baseline devices.

Better detection of function parameter mismatch (XC8-776) A warning is now issued for calls to a function which is passed arguments, but whose corresponding function definition (as opposed to

declaration) has an empty parameter list. The warning is issued for direct and indirect calls.

Memory range addresses (XC8-339) A leading `0x` may now be used with any value in a memory range associated with the `--ROM` or `--RAM` option (or IDE equivalent). Note that values that do not use this prefix are still assumed to be in hex format.

External memory preprocessor symbol (XC8-836) The preprocessor symbol `__EXTMEM` will be set to the size of the external memory implemented by the target device for those PIC18 devices that provide an external memory interface.

New assembly operator form (XC8-42) The use of the `BANKMASK` preprocessor macro in assembly macros (`MACRO..ENDM`) was not permitted as the assembler would see the `&` operator in its expansion as the assembly macro concatenation operator, not bitwise AND. The assembler now allows the use of `and` as well as `&` for bitwise AND operations. The `BANKMASK` preprocessor macro in assembly header files has been updated to use this new operator and so may now also be used in assembly macros.

Warning on possible write to read-only object (XC8-67) The compiler may now warn if using a pointer to write values and that pointer has targets that are qualified `const`. This warning may still be accompanied by other error messages. Writing values to `const`-qualified objects has undefined behavior.

Optimizations (XC886, XC8-79) The compiler now optimizes some code better, in particular interrupt code. Code which uses 24-bit floating point addition (`ftadd` routine) is now much smaller and faster.

3.25. Version 1.21

New device support The following parts (and their corresponding LF variants) are now fully supported by this release: PIC16F1713, PIC16F1716, PIC16F1703, PIC16F1707, PIC12F1572, PIC12F1571, PIC16F1705, and PIC16F1709.

New Installer Features The Installer application has new options for setting up a network client and displaying the Licensing Information page.

3.26. Version 1.20

New device support The following parts are now fully supported by this release: PIC16F1704, PIC16LF1704, PIC16F1708, and PIC16LF1708.

ELF/DWARF debugging Previously, COFF debug files were produced to allow source-level debugging in MPLAB IDE. The compiler can now produce ELF/DWARF files, although the default output format is still COFF. ELF/DWARF files are not compatible with MPLAB IDE v8 but will be the preferred format if you are using MPLAB X IDE as they have fewer limitations. You must be using a version of MPLAB X IDE that supports this file format for the MPLAB XC8 compiler. Not all aspects of DWARF are implemented in this compiler release. See the Known Issues section for more details.

New Free mode optimizations The assembler's jump-to-jump optimizations, which previously was only available with a licensed compiler operating mode, is now available in Free mode. By default, this optimization is disabled, but it can be enabled from the `--OPT` option or the Optimization category in MPLAB X IDE in the usual way. If enabled, this optimization reduced the size of code output by the compiler.

New language extension option A new driver option `--EXT` can be set to `cci`, `iar` or `xc8` to indicate that the C language extensions accepted by the compiler are those belonging to the Common C Interface, IAR compatibility mode, or the native XC8 syntax, respectively. All of these extensions are discussed

in the user's guide.

Expanded hardware multiply usage The PIC18 hardware multiply instructions are now used for 16x16 bit integer multiplication. The library routine that implements this feature breaks the multiplication into several operations that can use the 8-bit hardware multiply instruction. The 32-bit integer multiplication routines continue to use an iterative solution and do not use these instructions.

New listing option Previously a C list file was produced for each C source file being compiled. If an assembly list file was request (which is the default in MPLAB IDE) then these listing files were overwritten with the assembly listing content. The C listing files are no longer produced by default. If you would like C listing files to be generated, a new option `--CLIST` has been added to request this action.

3.27. Version 1.12

New device support The following parts are now fully supported by this release: MCP19114, MCP19115, PIC16LF1824T39A, PIC16F570, PIC16F753 and PIC16HV753.

3.28. Version 1.11

New device support The following parts are now fully supported by this release: PIC16F1788, PIC16F1789, PIC16LF1788 and PIC16LF1789.

New peripheral libraries The following devices (and their LF counterparts) now have peripheral library support: PIC18F45K50, PIC18F24K50 and PIC18F25K50. The peripheral libraries for all supported devices have been updated to the latest code base.

New parsing option A new driver option has been added that alters the generation of intermediate files produced by the parser. The option is `--PARSER` and can be set to `lean` or `rich`. The `lean` suboption (the default) will not include unused symbols in intermediate files. These are included when selecting the rich suboption, but not that this setting will generate larger intermediate (p-code) files and will slow down the compilation considerably. The operation of prior versions of the compiler was equivalent to the `rich` setting. Use the `rich` setting if you need unused symbols to be included in the link stage.

New enhanced mid-range errata workaround The compiler now has the ability to employ an errata workaround for some enhanced mid-range devices. This is controlled by the `--ERRATA` option (`CLOCKSW`), which is used to control other PIC18 errata workarounds. The workaround will affect the device startup code, but is not enabled by default. Check your device literature to see if this workaround should be enabled.

3.29. Version 1.10

New device support The following parts are now fully supported by this release: MCP19110, MCP19111, RF675F, RF675H, RF675K, PIC12F529T39A, PIC12F529T48A, PIC12LF1840T39A, PIC12LF1552, PIC16F527, PIC16F1454, PIC16LF1454, PIC16F1455, PIC16LF1455, PIC16F1459, PIC16LF1459, PIC16F1784, PIC16LF1784, PIC16F1786, PIC16LF1786, PIC16F1787, PIC16LF1787, PIC18F45K50, PIC18F24K50, PIC18F25K50, PIC18LF45K50, PIC18LF24K50, PIC18LF25K50, PIC18F97J94, PIC18F87J94, PIC18F67J94, PIC18F96J94, PIC18F86J94, PIC18F66J94, PIC18F95J94, PIC18F65J94, PIC18F85J94, PIC18F96J99, PIC18F86J99, PIC18F66J99.

The Common C Interface (CCI) The Common C Interface is a documented set of ANSI standard refinements, non-standard extensions and guidelines to help you write code which is more portable across all MPLAB XC C compilers. A new chapter has been added to the XC8 User's Guide describing

these features.

User's guide A new compiler user's guide has been included with this release. See the Documentation Updates section, above, for more information.

Roam-out Licensing A new “roam out” feature allows a network license to be checked out for an extended period for use on a particular computer. While the license is checked out, the computer has licensed use to an XC compiler, and need not be in contact with the network license server. When the license is returned to the network license server, it is once more available to be used as a floating license, or to be roamed out to other computers.

Psect allocation The CCI `__section()` specifier can also be used in non-CCI mode. Refer to the CCI chapter in the user's guide. It can be used in place of the `#pragma psect` directive.

Function and module information Information about each function, which appears in the assembly list file, is now also displayed in the map file. In addition, a summary of program memory usage on a module-by-module basis is shown in the map file. This allows an estimate of the size of the code (excluding data) being generated by each module.

Bank specification with PIC18 devices The qualifiers `bank0` through `bank3` may now be used with PIC18 devices to allocate variables to a specific memory bank. These qualifiers must be used with the `--ADDRQUAL` option.

Implementation of strftime function The `strftime()` function has been implemented and is available in the standard libraries.

Qualifier synonyms A side effect of the CCI features is that when *not* in CCI or strict ANSI mode, most of the non-standard specifiers, e.g. `bit`, `near` and `far`, can also be represented with two leading underscores, as in `__bit`, `__near` and `__far` etc.

SFR structure types The structures that are defined to represent SFRs are now a typedef type that is available for general use.

Function in-lining A new qualifier, `inline`, can be applied to some C functions. A call to any function, thus qualified, will not generate a call-return sequence, but will be replaced by the assembly code associated with the function body. This expansion will save stack usage and may reduce code size if the function body is small. The assembler has always had the ability to inline small assembly sequences, so code size reductions may not be large. The operation of this qualifier is similar to use of the `new #pragma inline` directive. The previous (version 1.00 and earlier) `inline` pragma implementation has been renamed to `instrinsic`. (See Migration Issues below.)

Assembly psect flags To support function inlining, two new psect flags have been added: `inline` and `keep`. These indicate to the assembler, respectively, that the contents of a psect may be inlined (and then removed), and that the contents of a psect may be inlined but must never be removed.

3.30. Version 1.01

Enhanced PIC optimizations New optimizations, specifically aimed at the enhanced mid-range PIC devices, have been added. These optimizations reduce code size and target any code that indirectly accesses locations via the FSR registers.

3.31. Version 1.00

Psect merging and splitting Two new `PSECT` directive flags have been added to allow splitting or merging of psects by the assembler optimizer. Now, by default, no splitting or merging takes place, but the use

of the `split=allow` and `merge=allow` flags can indicate that these optimizations can take place. See the assembly language chapter in the user's guide.

Unified 8-bit device support This compiler unifies the two former HI-TECH C compilers which previously supported Microchip 8-bit PIC devices: HI-TECH C Compiler for PIC10/12/16 MCUs and HI-TECH C Compiler for PIC18 MCUs. This MPLAB XC8 compiler supports compilation for any supported 8-bit PIC device from the one application. A single device driver, xc8, is used to invoke the compiler regardless of the target device. This driver will invoke the appropriate internal applications based on your device selection. The `picc` and `picc18` drivers which controlled the former compilers are currently still employed, and the baseline/mid-range and PIC18 code generator and assembler applications are still separate. Only one copy of the generic applications, such as the preprocessor (`cpp`), the parser (`p1`), the linker (`hlink`), and utilities like `hexmate`, `objtohex` and `cromwell` are included with the compiler, and these are shared by all device targets.

Operating modes The former HI-TECH C Compiler for PIC18 MCUs only operated in a Lite or PRO mode. With XC8, a Free (previously called Lite), Standard and PRO mode are available for all target devices.

4. Migration Issues

The following are features that are now handled differently by the compiler. These changes may require modification to your source code if porting code to this compiler version. The version number in the subheadings indicates the first compiler version to support the changes that follow.

4.1. Version 2.32

4.2. Version 2.31

None.

4.3. Version 2.30

Math changes (XC8-2017) There are several things changed in `<math.h>` relevant for C99 builds.

- The math_errhandling macro value has changed from 2 to 1 (`MATH_ERRNO`) meaning that all errors are expressed through `errno`.
- The `clock_t` typedef has changed from `long` to `unsigned long`, to be consistent with the other XC compilers.

4.4. Version 2.29 (Functional Safety Release)

None.

4.5. Version 2.20

Hexmate's search specification (XC8-1883) The order of the bytes in the search value used by the `-FIND` command have been reversed so as to match the ordering used by the replace value and to make it easier to search for an opcode, for example.

Warning on missing operand (XC8E-607) A warning is now issued if a PIC18 file register instruction does not specify the RAM access bit operand, which indicates a banked or Access bank location, e.g. `,c`. The assembler will attempt to determine the destination if possible, but it is recommended that this operand always be specified with these assembly instructions.

Conversion of assembler controls to directives (XC8E-580) Most of the assembler controls (`OPT` *`control`*) have been changed to assembler directives, to be more compatible with their counterparts in MPASM. The following directives are now supported.

- `[no]list`
- `[no]cond`
- `title`
- `subtitle`
- `[no]expand`
- `callstack` (previously the `OPT stack` control)
- `pagelen`
- `pagewidth`
- `include`
- `asmopt`

These will work as they did before, but will no longer require the use of the `OPT` keyword. So, where previously you might have used `OPT TITLE "my Title"` for example, you should now use `TITLE "My Title"`. The four seperate `ASMOPT_*` controls are now a single `ASMOPT` directive that takes a parameter: `ON`, `OFF`, `PUSH`, or `POP`, so replace `OPT ASMOPT_PUSH`, for example, with `ASMOPT PUSH`.

The previous `OPT` controls will continue to work as they did before, but will trigger a warning message.

Conversion of branch instructions (XC8E-144) The PIC18 assembler will no longer convert a conditional branch instruction to the opposite conditional branch over a branch or a skip instruction over a jump where these occur in hand-written assembly modules.

4.6. Version 2.19 (Functional Safety Release)

None.

4.7. Version 2.10

Individual device INI files Previously, the compiler had two device INI files (one for PIC18; one for the other devices) that defined the architecture for all supported devices. These INI files have been split up so that there is one INI file for each device. This change should not require projects to be modified.

Legacy config/idloc macros The configuration bit macro, `__CONFIG()`, and ID location macro, `__IDLOC()`, are no longer supported when building for C99. A warning is issued if you attempt to use them in this way. They are still accepted when building for C90; however, it is recommended that you move to the newer-stye `#pragma config`, if possible.

4.8. Version 2.05

Removal of macros The `_HTC_EDITION_` and `_XC8_MODE_` macros are no longer defined. These macros were defined based on the operating mode of the compiler; however, these modes are no longer

recognised. As an alternative, use the macros that define the optimization level, such as __OPTIM_LEVEL.

4.9. Version 2.00

PIC18 peripheral library support The PIC18 peripheral library was removed from the compiler distribution in a previous release, and it had to be downloaded separately if you needed to use it for legacy projects. The library is now truely defunct if you are using C99 and the Clang front end, and the compiler will reject the option to use this library in that case. To continue to use the library with the C90 compiler front end, the library file must be downloaded separately. If any of the device SFR definitions have changed since the library was built and a redefinition error is produced, the library source files should be added to and compiled with your project.

EEPROM routines Previously, declarations were provided for the memcpyee() and eecpymem() functions when building for some Baseline and Mid-range devices. These routines were only intended for internal compiler use, however, it was possible to call these routines from your source code. These routines should no longer be used from your source code and declarations for them are no longer provided. These routines are automatically called when you access objects qualified with __eeprom and the routines to read and write EEPROM are still provided.

Mode warning messaging (XC8-1745) When the compiler built in Free mode, an advisory was always printed, indicating that compilation took place in this mode. This is no longer printed; however, a new message (2051) is issued whenever the compiler has been asked to run with a higher optimization level than that permitted by the compiler licence.

Non-standard types (XC8-1588) The 24-bit, non-standard integer short long int types must now be defined using the types __int24 and __uint24 when building for C99. The bit type must now be specified as __bit.

Change in keywords Many tokens that were mandatory only when using the CCI have been standardised when building for C99, even if you are not using the CCI. These include the keywords: near, far, bank*x*, eeprom, and persistent, which should be changed to __near, __far, __bank*x*, __eeprom, and __persistent, respectively, if you are using C99. The use of @ *address* to specify an absolute variable or function must now be performed using the __at(*address*) syntax. Interrupt functions that used the interrupt keyword must now be defined using __interrupt() and the appropriate arguments.

In-line assembly The #asm ... #endasm form of inline assembly is no longer accepted when building for C99. The asm() form of inline assembly is now the only way to embed assembly instructions inline with C code.

Small floats If you are building for C99, support for 24-bit float and double floating-point types is no longer provided, and these types will be forced to 32-bits wide. If you need the smaller version of these types with C99, use the C90 libraries (-mc90lib option).

Default optimization level If you build on the command line, the default optimization level has changed from the highest level to none. Ensure you explicitly state the optimization level you need using the -O option. New projects in the MPLAB X IDE always specify a level and will default to level 0 (no optimizations).

Predefined macros Some macros have been deprecated, such as the _HTC_VER_*xxxx*_ macros, which define the current version. Note also that there are several new macros defined by the compiler, such as __CLANG__, which can be used to conditionally compile code.

Unused function warning (XC8E-50) A warning with a unique message number is now issued for unused functions that are specified as `inline`. This allows these warnings to be suppressed but allowing warnings for regular unused functions to be emitted as usual.

4.10. Version 1.45

None.

4.11. Version 1.44

Bogus warnings in header file (XC8-1581) Warning (2028), relating to external declarations, was unnecessarily displayed when the status register preservation objects (e.g., `__resetbits`) were used and the project was being built for non-PIC18 devices.

Hexmate serial option change (XC8-1425) The Hexmate `-SERIAL` option now allows the explicit use of 0x before the Code argument.

Truncated hexmate fill value warning (XC8-1420) A warning is now printed if the Hemxate fill value does not wholly fit in the specified address range.

Better error reporting in Hexmate (XC8E-172) Processing of hexmate's `--EDF` option now occurs sooner, so that descriptive warning and errors can be produced when processing other Hexmate options.

4.12. Version 1.43

Local optimization support (XC8E-169) The compiler now only supports local optimizations for enhanced mid-range and PIC18 devices. An error message will be displayed if this optimization is selected with a device that does not support this feature.

Revival of types in <stdint.h> (XC8-1551) Non-standard 24-bit types, such as `int24_t` have been restored to `<stdint.h>`.

4.13. Version 1.42

No license check during preprocessing (XC8-141) A license is no longer checked out when the `--pre` option is used.

Warning for incompatible pointer to void assignment (XC8E-140) The compiler was silently allowing the address of a function to be assigned to a pointer to void. This assignment conversion is illegal. The compiler will now warn the user if any such conversions are detected in a program.

Removal of types from <stdint.h> (XC8-1551) Non-standard 24-bit types featured in `<stdint.h>` are no longer available.

Bogus aliases (XC8-1539) Aliases for SFR bits `GODONE` (in `ADCON0`), `ADCAL` (in `ADCON0`), and `VCFG11` (in `ADCON1`) have been removed from the PIC18 K40 and K42 device header files.

Non-standard floating-point macros (XC8-1528) The compiler was defining extra macros for `DLB_RADIX` and `DBL_ROUNDS` in `<float.h>` and these have been removed. The values for these characteristics are fixed for all floating-point types and the `FLT_RADIX` and `FLT_ROUNDS` macros should be used instead.

Warning for external objects (XC8-1521) The code generator has no knowledge of the location of objects defined in assembly code. The compiler will now use any bank specifiers used in declarations for such objects to produce valid and optimal code for the access of these objects. If no specifiers are present, a warning is now generated to indicate that the generated code is suboptimal. It is highly recommended that variables be defined in C code whenever possible.

Removal of timer macros (XC8-1394) The macros `T1RD16ON` and `T3RD16ON`, defined in `<pic18.h>`, are no longer supported. Their usage will now result in an "unsupported" warning. To enable 16-bit timer read/write operations, please refer to your device data sheet.

Inclusion of interrupts in library files (XC8-123) In previous versions of the compiler, interrupt functions that were contained in libraries were not included into projects that used the library. The compiler will now use all interrupt functions contained in libraries.

Change in warning level for implicit conversion (XC8E-109) The warning level for warning 373 (implicit signed to unsigned conversion) has been raised from -4 to -3.

4.14. Version 1.41

XCLM expiration messaging (XCLME-16) Some messaging issued by the license manager regarding the expiration of HPA and the compiler's licence have been suppressed.

Retraction of invariant/stable optimization mode The `invariant/stable` optimization setting is no longer available and a warning is issued (subject to your set warning level threshold) if this is enabled. This mode has been replaced with a new `ocg` optimization setting, detailed here.

Advisory on function duplication An advisory is now issued if a non-reentrant function called from multiple call graphs is duplicated by the compiler. There is no change in the generated code for such situations, but you are now alerted to the compiler's response.

Response to PIC18 eeprom-qualified variables (XC8-1498) Previously the compiler would issue a warning if a PIC18 project variable was qualified with `eeprom`. This is now an error.

4.15. Version 1.40

Error on overlapping segments (XC8-1499) Previously a warning was issued if two segments (psect collections) overlapped each other in memory. An error is now issued in such situations.

Removal of legacy EEPROM function calls (XC8E-118) Calls to the legacy EEPROM functions, `eeprom_read()` and `eeprom_write()` were previously mapped to the equivalent PIC18 peripheral library functions if EEPROM was supported by the device and that library was installed. If the library was not installed, theses calls were removed. The compiler now always maps these functions to the peripheral library. If the library is not installed when you build, an error will be triggered.

End of the end_init psect The `end_init` psect, which was used only for PIC18 projects to hold code which jumped to the main section of the runtime startup code, is no longer used. Code previously contained in this psect is now located in the `init` psect.

4.16. Version 1.38

Error on extended instruction set (XC8E-97) Attempting to enable the extended PIC18 instruction set will now result in error message (1504). This instruction set is not supported by the compiler.

4.17. Version 1.37

Unhelpful warning regarding empty structure definition (XC8-1377) Defining a structure with empty braces leads to undefined behavior, but the warning produced by XC8 indicated a misuse of the `__section` specifier and so was of little use.

Unhelpful error with condition inclusion (XC8E-74) In some instances, a malformed preprocessor `#if` directive might have caused the compiler to emit a confusing error message indicating illegal characters

in the directive.

4.18. Version 1.36

Debug optimizations setting inconsistent (XC8E-71) The operation of the debug optimization setting is now consistent for all device families. It disables all forms of inlining and procedural abstraction (reverse inlining), which can all negatively affect source-level debugging of projects. Enabling this optimization will increase code size, but will enhance the debuggability of projects.

Was invariant; now stable object The invariant optimization setting referred to in previous compiler versions has been renamed in this compiler version to stable-object mode, to better reflect its purpose. You can now use the `stable` suboption to the `--OPT` option instead of `invariant`, and the `__stable` function specifier instead of `__invariant`.

Void issues (XC8E-87) Using the `sizeof()` operator on an object of type `void` will now return the value 1. Code which performs arithmetic operations on `void *` objects will now issue a warning.

4.19. Version 1.35

Removal of PIC18 peripheral libraries The PIC18 peripheral libraries and their associated header files and documentation are no longer shipped with this compiler. The MPLAB Code configurator (MCC) can be used to generate routines that perform similar tasks to those supplied by the peripheral library, and can generate code for many devices, not just the PIC18 family. The MCC is available as an MPLAB X IDE plugin and can be downloaded from the IDE under the **Tools > Plugins** menu. If you need to use the peripheral libraries that were formerly shipped with the compiler, an archive of these are available as a separate download and will need to be installed over the top of your compiler installation. The archive was built for v1.34 of the compiler. If you encounter any inconsistencies between the archived libraries and projects compiled with a more recent compiler, you can copy the relevant library source modules into your project.

New floating-point libraries Replacement libraries have been supplied that contain updated routines to perform 32-bit floating-point operations. These routines produce more accurate results than the previous 32-bit library routines; however, they have a ‘relaxed compliance ’and obey the following guidelines.

- Tiny (sub-normal) arguments are interpreted as zeroes.
- NaN arguments are interpreted as infinities.
- NaN results are never created in addition, subtraction, multiplication, or division—where a NaN would be created, an infinity of the proper sign is created instead. Square root of a negative number will return the "distinguished" NaN (default NaN used for error return), as in the "non-relaxed" IEEE-754 libraries.
- Infinities are legal arguments for all operations and behave as the largest representable numbers; for example, +inf + -inf yeilds zero.
- Tiny (sub-normal) results are flushed to zero.

  The results obtained from these new libraries might differ to those produced from the 32-bit libraries supplied with previous compiler versions, especially for tiny (sub-normal) values.

  The function `__fpnormalize()` can be called to convert any literal, 32-bit, floating-point value so that it conforms to the relaxed guidelines above, that is is will remove negative zeros and sub-normal values. The assembler operator `NORMALIZE32()` can be used to perform the same task for literals in hand-written assembly code.

The (default) 24-bit implementation of the floating-point routines have not changed in this compiler version. These new routines might lead to an increase in code size, but they typically execute faster. Advisory messages are output when they are in use.

Device oscillator calibration Oscillator calibration using `--RUNTIME=+config` will not work with new devices that obtain their calibration constant from a calibration word stored in flash program memory, such as the MCP19114/5 devices. Disable this runtime sub-option if you plan to swap to one of these new devices. Refer to the device data sheet for instructions on reading this data.

Application command lines To better accommodate long command lines, the compiler driver now passes a command file to all compiler applications when building, rather than passing a list of options on the command line. The command file is a temporary file that holds the options for the relevant application. If you select a verbose build, you will see the command file name being passed to the applications after the `@` character, for example:

```
/Applications/dev/XC8/v1.35/bin/hexmate @/tmp/hexmate_xcwst2geN.cmd
```

If you wish to see the content of these files and the specific options passed to the applications, use the verbose options twice, e.g. `-v -v`, for example:

```
/Applications/dev/XC8/v1.35/bin/hexmate @/tmp/hexmate_xcwst2geN.cmd [ --
edf=/Applications/dev/XC8/v1.35/dat/en_msgs.txt main.hex -Omain.hex -
logfile=main.hxl -addressing=1 -break=300000 ]
```

Relaxed linking of stringtext psect (XC8E-76) For enhanced mid-range devices, the `stringtext` psect is no longer restricted to being linked at an address that is a multiple of 0x100. This will make it easier to allocate code to memory and might suppress some can't find space errors.

Conditional assembly directive (XC8-1266) An error is now issued if the number of `ENDIF` directives does not match the number of `IF`/`ELSIF` directives in each psect. Previously these situations were silently ignored.

4.20. Version 1.34

Use of XC8 in MPLAB IDE v8 is now deprecated (XC8-1228) As of MPLAB XC8 v1.34, the use of MPLAB 8 IDE is deprecated, and the installation of the DLL files that the IDE used to interface to XC8 are removed from the compiler installer.

Reentrant main (XC8-937) If the reentrant stack model is selected for a project, the `main()` function in that project will be compiled using the software stack and be reentrant, as per other functions. Previously `main()` was always compiled using the compiled stack regardless of project stack model. Interrupt functions are aways compiled in a non-reentrant manner.

Disabling C libraries It is no longer possible to request that linking of the standard C libraries is not performed. Previously attempts to do so often resulted in compilation errors. The `clib` suboption to the `--RUNTIME` option can be turned off, but it will result in a warning and have no other effect.

The exit and abort macros The `abort()` library routine is now a macro that expands to be `exit()`. The exact behaviour of `exit()` is dependent on the selected device and whether the executable is a debug or production build. For debug builds, `exit()` will consist of a software breakpoint instruction followed by a reset instruction, if possible. For production builds, `exit()` will consist of only a reset instruction, if possible. In both cases, if a reset instruction is not available, a goto instruction that jumps to itself in an endless loop is output.

User assertions The `ASSERT()` macro depends on the implementation of the function `_fassert()`. The default `_fassert()` function, built into the library files, first calls the `printf()` function, which

prints a message identifying the source file and line number of the assertion. Next, `_fassert()` attempts to terminate program execution by calling `abort()` (see above point).

New license installers and messages During compilation you will now receive a warning when your are within 14 days of the end of your HPA subscription or demo period. The XC Network License Server is now a separate installer and is no longer included in the single-user compiler installer.

Merged library source files (XC8-1210) The source code for the baseline/mid-range `ftdiv` and `fldiv` floating-point library routines, formerly located in the `sources/pic` directory of your compiler, has been merged with the equivalent source code for the PIC18 devices. The new merged source files are `ftdiv.c` and `fldiv.c`, located in the `sources/common` directory of your compiler. These source files and the `ftarith.h` and `flarith.h` header files can no longer be found in the `sources/pic` directory. There is no function change to the operation of these routines.

The sizeof operator with pointers (XC8-1019) Changes have been made to ensure that if the `sizeof()` operator with a pointer operand has to be evaluated by the preprocessor, a warning is given. If the `sizeof()` operator with a pointer operand has to be evaluated by the code generator and the code generator cannot determine the result, an error will be output.

Warning with packed auto (XC8-1111, XC8-1156) A new warning (level -5) is now produced for attempted use of the `__pack` specifier with `auto` structures. This specifier is ignored for `auto` structures.

4.21. Version 1.33

None.

4.22. Version 1.32

Renamed devices The following indicates devices that have had their names changed. This release uses the new device names.

16F18323 -> 16F18323A
16LF18323 -> 16LF18323A
16F18313 -> 16F18313A
16LF18313 -> 16LF18313A
MCV082A -> 16F18313
MCV142A -> 16F18323

Function pointers simplification (XC8-1002) For enhanced mid-range devices only, function pointers are now always 16-bits wide and represent the full address of the target function. The compiler no longer uses a jump table when making indirect calls for these devices. No change is required for C code that uses function pointers, but any assembly code that accesses these will need to be modified.

4.23. Version 1.31

None.

4.24. Version 1.30

Psect names (XC8-865) The names of psects used to hold absolute `const` objects have changed from `xxxx_text` to `xxxx_const`, where `xxxx` is the name of the object being defined.

Parameter passing (XC8-914) For PIC18 devices the first byte-sized argument to a function was not passed in WREG in Free or Standard modes. (It was passed in WREG in PRO modes.) This has been changed

so that WREG is used regardless of the operating mode. Note that in the new software stack, argument passing is performed in a different way.

New warning level (XC8-151) The warning level of message 368, `array dimension on "*" ignored` has been reduced from 0 to -1.

New library source file locations (XC8-939) Note that the location of some of the library source files have changed since the previous release. Library source files that are specified to a target device family are located in either the `pic` or `pic18` subdirectory in the `sources` directory of the compiler. Source files that are identical across all device families are located in the `common` subdirectory.

4.25. Version 1.21

None.

4.26. Version 1.20

-G option disable The option that controlled generation of symbol files has been disabled. This option was always selected in MPLAB IDE and there was never a need to disable this feature. Symbol files are now always produced. You can continue to specify this option in builds, but it will be silently ignored.

C listings disabled by default C listing files are no longer produced by default. See the New listing option (in New Features) entry for more information.

Space number for EEPROM changed The space number used by psects that hold EEPROM data has changed from 2 to 3. This only affects PIC10/12/16 devices. This will not affect most projects, but if you see an error indicating `memory space redefined: 03/02`, then look for hand-written assembly that defines a psect used to hold EEPROM data and change the space value.

Warnings for absolute local objects (XC8-652) Previously the compiler would silently ignore any local (defined inside a function) object which the user had attempted to make absolute. The compiler will now issue a warning to say that the address specification will be ignored.

Warnings for qualified local objects (XC8-670) Objects which are local to a function (including autos and static local objects) cannot be qualified as `far` or `near`. The compiler was not indicating that the specifier was being ignored. A warning is now issued for such definitions.

4.27. Version 1.12

Access of multi-byte SFRs When writing literal values to multibyte `volatile` SFRs, the compiler will write high order byte first, then the low order byte. This conforms to the requirements of some PIC SFRs, such as the NCO register, which must be written in a particular order. This write order is not guaranteed for other expressions.

Watchdog configuration bit Some devices have had their configuration bit setting change from `WDTEN` to `WDT`. For these devices you will need to update any `#pragma config` directives that specify this setting.

Implicit function in-lining The implicit in-lining of function performed as an assembler optimization is now disabled by default. The assembler optimizer has, in the past, in-lined small assembly routines as part of its optimizations. You can re-enable it using the driver option `-A-auto_inline`. This change does not affect in-lining of C functions, controlled using the `inline` specifier.

4.28. Version 1.11

Inline SFR names When referencing SFR names specified in <xc.h> from in-line assembly code, do not use a leading underscore character with the symbol. So, for example, use the symbols PORTA or RA0_bit in in-line assembly code. (The same is true for SFR symbols in assembly modules that include <xc.inc>; however, the bit symbols have different names, for example, PORTA and RA0.) See the user's guide for more information on assembly code and accessing SFRs.

Compiler banner information The information shown in the compiler banner printed with each compilation has changed. The compiler operating mode is no longer printed, but you can still find this information in the list file. The license manage will now return and print the license type.

Unused symbol inclusion Unused symbols are no longer included in intermediate p-code files. This should not affect most customers. See the What's New section for information on the new --PARSER option which allows unused symbols to be included in the final link step.

Pointer comparison warning Warning number 1413 "*" is positioned at address 0x0 and has had its address taken; pointer comparisons may be invalid should now only be issued for symbols that are user-defined and not for temporary variables, such as "ROM" or "RAM".

Specification of config pragma The arguments to the #pragma config can now be quoted. You may prefer to quote the setting-value pairs to ensure that the preprocessor does not perform substitution of these tokens, e.g., #pragma config "BOREN=OFF"

4.29. Version 1.10

Re-activation of the compiler As of version 1.10, the license files used by the XC compiler license manager are installed in a different location. This is automatically handled by the installer; however, if you plan to use an older license file with v1.10, you will need to re-activate the v1.10 XC8 compiler with your activation key before you can use it. Previously, the installers could find older license files and did not require re-activation. Re-activation will not be necessary in future when updating compiler versions.

PIC10/12/16 Assembly Header Files The names of some of the assembly header files have changed. Provided code included the generic and recommended <xc.inc> assembly header file, then this will be transparent. Specifically, changes relate to using the extension .inc instead of .h for assembly-domain header files; the names of device-specific assembly header files now match their C-domain counterparts.

Deprecation of in-line assembly header files The PIC10/12/16 header files that were usable from assembly that was in-line with C code have been remove. These header files were previously included via the generic file <caspic.h>. The content of these files will be included once you include <xc.h> in your C source code.

Removal of single-letter bit definitions The definitions for SFR bits that used a single letter identifier have been removed, for example the SSP1CON register bit S. These register bits are still available in the corresponding SFR structures, for example SSP1CONbits.S. The absence of these very poorly chosen identifiers should not be missed.

4.30. Version 1.01

Missing SFR definitions There have been changes to the header files to ensure correlation with the data sheet. This may resulted in some SFR definitions being no longer available. If you see undefined symbol errors or other build errors relating to SFR names, please refer to the appropriate device-specific header file and update your code.

The inline pragma The much misused `inline` pragma has been changed. What was the `inline` pragma is now known as the `intrinsic` pragma. Since the `inline` pragma was not intended to be used with user-defined routines, this should have no impact on existing projects. Use of the `inline` pragma in this compiler version will be ignored; however, future versions may implement a user-operable inlining feature.

Filling unused values The `--FILL` option, which is used to fill unused memory locations, now assumes the value specified is a 2 byte word. Previously the size of this value was the same as the target device program memory addressability i.e. 1 for PIC10/12/16 devices and 2 for PIC18 devices. The width of the value can always be specified with the option. See the user's guide section relating to Hexmate.

4.31. Version 1.00

Library functions removed The following library functions and macros are now longer supported and have been removed from the libraries and header files.

- `device_id_read`
- `idloc_read`
- `idloc_write`
- `config_read`
- `config_write`
- `checksum8` and the macro `CHECKSUM8`
- `checksum16` and the macro `CHECKSUM16`
- `checksum32` and the macro `CHECKSUM32`

Any attempt to use these will result in a compiler error.

Memory functions replaced by peripheral library equivalents Flash and EEPROM functions and macros have been removed from the compiler libraries and header files. They have been replaced with those in the peripheral library.

Compiler drivers While you may continue to call the `picc` or `picc18` command-line drivers, it is recommended that projects be swapped to use the unified `xc8` driver. Future compiler releases may discontinue the device-specific drivers or additional functionality may be added to `xc8` that will not be usable if you continue to use the device-specific drivers.

FN-type directives All FN-type directives are no longer supported and should not be used. Such directive include: `FNBREAK`, `FNSIZE`, `FNROOT` etc. The `FNCALL` and `FNROOT` directives are still issued by the compiler, but it is recommended that these not be used in handwritten assembly code.

5. Fixed Issues

The following are corrections that have been made to the compiler. These might fix bugs in the generated code or alter the operation of the compiler to that which was intended or specified by the user's guide. The version number in the subheadings indicates the first compiler version to contain fixes for the issues that follow. The bracketed label(s) in the title are that issue's identification in the tracking database. These may be useful if you need to contact support.

5.1. Version 2.32

Breaking builtins broken (XC8-2407) The `__builtin_software_breakpoint()`, `_debug_break()` and `__conditional_software_breakpoint()` builtins did not use the correct instruction coding for traps when used with K42 and Q PIC18 devices. A `nop` instruction has also been added after the trap to account for debugger skidding.

Second access of library fails (XC8-2381) Invoking the Windows version of the `xc8-ar.exe` library archiver a second time to access an existing library archive may have failed with an `unable to rename` error message.

Incomplete expansion of assembly macros (XC8-2334) Assembly macros (from either PIC-AS source or in assembly modules when using the C compiler) were incompletely rendered in MPLABX's disassembly view, showing on the first instruction in the sequence. This issue did not affect code operation, and all instructions in the macro should now be displayed in the IDE.

Consistency check for target device (XC8-2327) The linker will now check the extended ident record (where present) to ensure that all object files passed to it were built for the same device architecture. Extended ident records are produced by the Microchip PIC-AS assembler.

Assembly division by zero failure (XC8-1960) The assemblers were not detecting a division by zero in constant assembly expressions, which would result in a build failure but no reported error.

Long built times (XC8-1930) Programs containing recursion could have taken an inordinate amount of time to build. Better tracking of the call graph will now decrease the built times for such programs.

Bogus warning with absolute functions (XC8-1809) Absolute functions located above the highest RAM address on PIC18 devices that implement an interrupt controller module would have caused the compiler to erroneously warn that the function `lies outside available data space`. The warning did not affect the operation of the code.

5.2. Version 2.31

Unexplained compiler failures (XC8-2367) When running on Windows platforms that had the system temporary directory set to a path that included a dot '`.`' character, the compiler may have failed to execute.

5.3. Version 2.30

Missing linker options (XC8-2333) The PIC Assembler driver was not issuing several linker options. With these options missing, the delta value of classes contained in program memory on Baseline and Mid-range devices were not being correctly set, and for devices that implemented EEPROM the `EEDATA` class was missing entirely. This issue would not have affected the generated code, but not all of the available memory areas would have been usable, and premature `can't find space` or other errors might have resulted. The XC8 C compiler driver was not affected by this issue and C programs built as expected.

Debugging assembly source (XC8-2319) Stepping through assembly source code that included other files might not have worked correctly due to the incorrect interpretation of embedded debug information.

Debugging with structures (XC8-2303) The DWARF debugging information for programs that contained a mixture of complete and incomplete structure/union types might have been incorrect. When the MPLAB X IDE encountered such errors, it would have stopped reading the remaining debugging information, thus affecting the debugability of the program.

Missing stdint.h types and macros (XC8-2302) The following 24-bit integer types were unavailable in

<stdint.h>: int_least24_t, int_fast24_t, uint_least24_t, and uint_fast24_t.

The following macros related to 24-bit integer types were also unavailable: INT24_MIN, INT24_MAX, UINT24_MAX, INT_LEAST24_MIN, INT_LEAST24_MAX, UINT_LEAST24_MAX, INT_FAST24_MIN, INT_FAST24_MAX, UINT_FAST24_MAX, INT24_C(), and UINT24_C().

Wrong assembler device support list (XC8-2301) The pic-as driver option -mprint-devices incorrectly reported AVR devices as being supported, which was not the case.

No chip lists for old driver (XC8-2297) Using the --chipinfo option of the legacy xc8 driver might not have found the device information and issued an error.

Not so fast (XC8-2296) The fast 16-bit types and macros provided by the C99 version of <stdint.h> incorrectly used 32-bit types.

Target option not following GCC style (XC8-2291) The xc8-ar library archiver utility did not accept an equal sign after the --target option, as with the GCC implementation of this utility. If the option was used in this way, the archiver might not have functioned correctly. You may now use either --target=*chipname* or --target *chipname*.

Warning, warning (XC8-2289) When the generation of an assembler listing file was enabled for C99 builds, some warning messages might have appeared twice.

Nul characters not printed (XC8-2285) Formatted printing functions in the C99 standard library would fail to print characters using the %c format-specifier if that character was '\0'.

Wrong function sizes (XC8-2284) In the generated map file, under module information, the size indicated for functions might have been 1 greater than the function's actual size.

Missing SFR access bits (XC8-2282) Using the bit-access SFR macros (for example btfsc TMR0IE) supplied in the device-specific assembler include files resulted in the warning RAM access bit operand not specified, when the assembly instruction did not specify the access bit. The macros now define the access bit so this does not need to be specified in project assembly source code.

Crash with GOTO in inline assembly (XC8-2266) For PIC18 projects that enabled assembler optimizations and that used a goto instruction without a symbolic destination operand (i.e. it was a constant address) in inline assembly, the compiler might have unexpected terminated.

Incorrect library macro (XC8-2265) The definition of SIZE_MAX in the C99 <stdint.h> standard library header was incorrectly defined to be UINT32_MAX. This has been made coherent with size_t and is now defined to be UINT16_MAX.

Math changes (XC8-2017) The following fixes have been made to the C99 <math.h> functions.

- The log10 , log2, and log families of functions now generate a domain error should their argument be negative.
- The log1p family of functions now generate a domain error should their argument be less than -1.

Invalid library creation (XC8-2015) When building a PIC-device library using the -r option to xc8-ar, if the named library file did not exist, an invalid library was created that would produce errors when it was later used.

Bad address shifts (XC8-2013) For Baseline and Mid-range projects, expressions that took the address of an object, cast that to an integer, and right shifted the result, might have produced incorrect values.

Undetected fixup errors (XC8-2009) The linker was not detecting that fixup of some assembly operand expressions was overflowing and hence did not produce a fixup overflow error. Although this affected

all PIC devices and could lead to code failure, the expressions that triggered this situation were extremely rare.

Unable to build pointer conversion (XC8-2008) The compiler might not have been able to build expressions involving a conversion from a 1-byte wide `const` pointer to a 24-bit integer, in PIC18 projects, failing with the error: `registers unavailable for code generation of this expression`.

Fixup error with large-RAM devices (XC8-1996) For PIC18 devices with more than 4K of data memory, the compiler would in some instances use a `movff` instruction where a `movffl` instruction was needed. This would have resulted in a fixup error stopping the build.

Bad comparisons (XC8-1994) For Baseline and Mid-range devices, expressions that compared integers that were greater than 2 bytes in size, were comprised of objects allocated to different banks, and yielded certain values, the result might have been incorrect due to a missing banked selection instruction.

Incorrect abstraction of non-identical code (XC8-1978) In PIC18 projects, the assembler optimizer did not correctly recognise that instructions that accessed the same file register address but that specified different access bits were different instructions. As a result, instruction sequences were being factored out as common, even when there was this small discrepancy in their functions.

Floats printed badly (XC8-1972) Formatted printing functions in the C99 standard library failed to take into account that rounding a `float` to be printed may increase that value's power of 10 and hence the number of significant digits to print.

Crash when using hybrid stack model (XC8-1955) Programs for Enhanced Mid-range devices that used the hybrid stack model caused the compiler to crash if that program contained functions that used the compiled stack and accessed 64-bit integer objects.

Looping around allocGlobals error (XC8-1943) Building projects that used a compiled stack and had functions that were called reentrantly and that defined pointer parameters or pointer auto objects might have result in a "looping around allocGlobals()" error.

Unnecessary memory allocation for statics (XC8-1942) When optimizing PIC projects at level `3` or level `s`, in some instances, functions that were never called and that defined `static` local variables may have had memory allocated for those variables even though the function itself was removed.

Multi-line pragmas failing (XC8-1916) Any `#pragma` directives in C99 projects that were split across multiple lines using a line continuation character might have triggered an error.

Lack of no space error (XC8-1865) The compiler did not issue an error when a region of memory was entirely consumed by absolute objects and a memory-specified object (used in conjunction with `-maddrqual=require`) had to be located in that same region.

Crash with pointers to incomplete type (XC8-1863) In certain circumstances where a program used pointers to an incomplete structure type, the compiler attempted to generate code for expressions that accessed members from that type rather than from the complete type, which was properly defined and available, resulting in a compiler crash.

Bad call to subtype for indirect call (XC8-1859) Indirect function calls that appear to have been made using a NULL pointer trigger a warning and are replaced with a constant expression of zero. However, the compiler did not also convert that expression to the return type of the replaced function call and in some cases this resulted in the error message `bad call to typeSub`.

Can't find space error (XC8-1843) For projects targeting PIC10/12/14/16 devices that are using optimizations and have `const`-qualified absolute objects, the compiler may have issued a can't find

space error, even though there may have be sufficient program memory available.

Too large arrays undetected (XC8-1827) The Clang front end (used by C99 projects) was silently truncating the size of very large arrays. If the number of array elements now exceeds the maximum allowable, an error will be issued.

Bogus warning in library code (XC8-1814) Some C99 programs that used formatted printing elicited the warning `pointer in expression may have no targets` for files in the standard library. These warnings will no longer be issued.

Pointers truncated when printed (XC8-1241) Pointers printed using the `%p` `printf` format specifier were truncated to 2-bytes before printing, resulting in the upper byte being absent when the pointer was 3 bytes wide.

Arithmetic overflow (XC8-1429, XC8-1122) When converting a constant expression type to a smaller type as part of an assignment, the code generator would emit a warning, even when an appropriate cast was used.

5.4. Version 2.29 (Functional Safety Release)

Bad labels in powerup stub (XC8-2011) The powerup assembly source provided to customise project startup sequences was missing colons on labels, which is now mandatory and which would have led to syntax errors. This issue did not affect projects built for PIC18 devices. Note that the C99 powerup source files now use a `.S` extension, and the C90 source files are present with a `.as` and `.S` extension. Use the `.as` version of these files only if you are using the older compiler driver, `xc8`.

5.5. Version 2.20

Undependable dependency files (XC8-1991) When using the driver's `-Mx` options, a dependancy file was not being generated in all instances.

Preprocess-only builds not stopping (XC8-1989) If the `-E` option (preprocess only) is used with assembler source files, the compiler or assembler may exit with error 141, stating that it is unable to open an object file.

Wrong destination assumption (XC8-1986) If a `movwf` instruction did not specify an addressing mode (banked or the Access bank) the assembler assumed the Access bank was to be used if it could determine that the address of the file operand was in the SFR portion of the Access bank, or the file operand was prefixed with `c:`; otherwise, it assumed banked addressing. Now, if the assembler can detect that the file operand addresses the GPR portion of the Access bank, it will assume the Access bank. It is recommended that you always use the appropriate operand (e.g. `,a` or `,b`) to indicate the desired addressing mode in hand-written assembly code rather than reply on defaults.

File-creation errors (XC8-1985) When building MPLAB X IDE projects that invoke Hexmate to perform non-standard operations, such as merging for bootloaders, there might have been file creation errors relating to Hexmate temporary files.

Unknown rfPIC devices (XC8-1981) The compiler was not able to find the device INI file when building for any of the rfPIC12C509AF, rfPIC12C509AG, rfPIC12F675F, rfPIC12F675H, or rfPIC12F675K devices.

No output for -dM option (XC8-1974) When a source file was provided on the command line together with the `-dM` option, no output was produced. This has been corrected, and a list of defined macros is output as expected.

Wrong escaped character constants (XC8-1971) In C99 PIC projects built using the Mac OS X compiler, escaped hexadecimal character constants greater than 0x7F were being stored as 0xFF.

Extraneous call graph heading (XC8-1968) In some instance a call graph heading may have been shown in the map file even though no call graph should have been produced.

Bank selection with long objects (XC8-1967) In PIC18 projects, the bank of 32-bit wide integer objects that were a member/element of an aggregate type larger than a bank might not have been selected prior to that object being used in expressions involving basic math operations.

Broken assembler LIST control (XC8-1958) Some valid arguments to the `LIST` assembler control may have triggered a syntax error. Note that assembler controls have now been made directives, see Migration issues for more information.

Looping around allocGlobals (XC8-1851) PIC18 projects using the C90 standard and libraries might have generated a `looping around allocGlobals()` error message in rare instances.

Library search order (XC8-1936) When searching for a symbol that was present in more than one library, the linker might have linked in the module from the wrong library. The library search order is now followed under all circumstances.

Cromwell crashes with F* filenames (XC8-1933) When a project had multiple `static` functions with the same name and these functions were defined in a source file whose name begins with 'F', the cromwell utility might have crashed.

Missing keywords (XC8-1921) The compiler did not recognize the `__software` and `__compiled` keywords when building for C99 projects.

Code generator crash (XC8-1905, XC8-1806, XC8-1923, XC8-1954) When compiling for the C99 standard, the compiler's parser (Clang) would fail to ensure that the initializing expressions for `const`-qualifed local objects were a compile-time constant. The compiler's code-generators assumed that check has already taken place, and might have crashed when encountering erroneous intermediate code from the parser.

Wrong constant propagation in duplicated functions (XC8-1903) In some instances, constant propagation optimizations were being incorrectly applied to auto variables in duplicated functions.

Incorrect initialization (XC8-1887) For Enhanced Mid-range projects, the data used to initialize large objects whose address had been taken, might have been incorrect.

Looping error (XC8-1876) Programs containing calls to unprototyped functions that had not been defined resulted in a "looping around globalAllocs" error.

Bad rotate code (XC8-1868) At level 2 or higher optimization levels, the compiler would generate erroneous code or be unable to generate code for some expressions that implemented a left bit-rotate.

Unnecessary conversion of branch instruction (XC8-1866) The PIC18 assembler might have unnecessarily converted a conditional branch instruction to the opposite conditional branch over a branch or a skip instruction over a jump when the target of the branch was specified using the location counter, `$`.

Unnecessary use of movffl instruction (XC8-1856) For projects targeting PIC18 devices with more than 15 banks of data memory, the compiler used a `movffl` instruction to access some objects (e.g. objects with automatic storage) allocated to banks 0-15 whereas a smaller `movff` instruction would have sufficed.

Bad timer reads (XC8-1854) The device-specific headers of some PIC18 devices (e.g. PIC18F'K42) do not provide definitions of joined timer registers (e.g. TMR0, TMR1, etc.). In such cases, the

`READTIMERx()` macros expanded into invalid code and potentially with no indication that this was the case. In such circumstances, the compiler will now emit a warning message that the macro is not supported by the current device.

No sign extension of right shift (XC8-1833) The compiler generated incorrect code for some expressions involving a right-shift of `long` or `long long` integer object by 16 bits.

Bank assumption disregarded when using code offset (XC8-1533) For PIC10/12/16 devices, the compiler generated runtime startup code might not have functioned correctly when the code offset feature (`-mcodeoffset`) was used and code (typically a bootloader) executed prior to the startup code exited with a bank other than 0 selected.

Assembler memory leaks (XC8-1482) A number of substantial memory leaks were identified and closed in the compiler's assemblers, thereby reducing their memory footprint. These leaks might have triggered out of memory errors for large projects.

Can't find space error (XC8-1112) When building code for Enhanced Mid-range devices, the placement of large objects that were less than the size of a bank might have triggered can't find space errors, even though there appeared to be sufficient memory available.

5.6. Version 2.19 (Functional Safety Release)

None.

5.7. Version 2.10

Bogus warnings relating to structure sizes (XC8-1900) Warnings stating incorrect sizes of structures might have been issued when building C99 projects. This issue did not affect the generated code, which used the correct sizes and which would execute correctly.

Ineffective switch pragma (XC8-1893) `#pragma switch` directives placed inside a function body did not affect `switch` statements within that function.

Interrupts not linked from libraries (XC8-1888) In projects where an interrupt function (ISR) was provided in a library and other modules in that library located before the ISR module were required by the program, the ISR module was not correctly linked into the program.

Unwatchable reentrant autos (XC8-1878) The ELF/DWARF output for some PIC18 projects (e.g. those for PIC18 K42 devices) had the wrong address of the frame-pointer register encoded. This prevented auto variables within reentrant functions from being watched in the MPLAB X IDE.

Assertion failure for absolute objects (XC8-1870) When a region of PIC18 memory is entirely consumed by absolute addressed objects and a program requires that an object with the corresponding memory-specifier be located in that region, the code-generator reported an assertion failure.

Broken Mirrors (XC8-1862) In the device-specific C header files for 18(L)FxxK42 parts, many of the macros that describe the address of register-mirrors in the DMA space were incorrect.

Too much information (XC8-1861) Clang might have produced extraneous log output under Windows 7.

Bit-fields not promoted (XC8-1832) For non-PIC18 devices, operations on a structure bit-field may not have correctly promoted the bit-field value to a larger type, resulting in an incorrect result.

rand() out of range (XC8-1823) The C99 standard library function `rand()` could have returned a pseudo-random number outside of its stipulated range of 0 to `RAND_MAX`.

Wrong serial values in Hexmate (XC8-1820) Hexmate serial values that had leading zeros in the `-serial`

option might not have been processed correctly, resulting in the wrong value or wrong number of values being inserted into the hex file.base

32-bit installer component (XC8-1794) One component of the installer application was not a 64-bit application, resulting in a warning message being issued by Mac OS X.

config pragma error undetected (XC8-1747) The compiler failed to detect some syntax errors, such as missing assignment values, in the arguments provided to the `#pragma config` directive when more than one setting was specified in the one pragma.

Bogus pointer warning (XC8-1685) For PIC16(L)F devices that support `eeprom`-qualified objects, the compiler, in certain circumstances, may have incorrectly issued error 1402, `a pointer to eeprom cannot also point to other data types`.

Wrong file in error message (XC8-1603) If a syntax error was encountered on a preprocessed assembly source file, the compiler may have referenced an intermediate source file in its error message rather than the source file.

No error for bad codeoffset (XC8-1571) Using an invalid value for the codeoffset feature might have resulted in the compiler exiting without an appropriate error message.

Code size fluctuations with formatted printing (XC8-1556) Standard printing functions (e.g. `sprintf`) that were referenced but not ultimately used in the project might have influenced the encoding (hence the code size) of other printing functions that *were* linked into the final program output.

Can't find space (XC8-1553) For mid-range devices that have large amounts of common memory, the compiler may have produced a can''t find space error rather than allocating large objects to that space.

Bogus warnings from library code (XC8-1552) In some cases when compiling for C90, library code from formatted output functions, like `printf()`, would emit warning 373, `implicit signed to unsigned conversion`

Illegal instructions permitted (XC8-1489) The mid-range assembler failed to detect illegal instructions if the destination argument was supplied with the instruction.

Illegal initialization not detected (XC8-1457) The compiler did not detect when absolute-addressed variables located in RAM were initialized with zero. Initializing such objects is not permitted and an error message is now emitted should such an initilization be found.

5.8. Version 2.05

Two too small (XC8-1816) For PIC10/12/16 devices, the size of the used program memory displayed in the memory summary might have been 2 words less than the actual usage. This issue only affected the memory summary printed by the driver; the linker map file was accurate.

No abstraction of inlined (XC8-1813) When compiling for space, procedural abstraction optimizations might not have been applied to code inside functions that had been inlined.

Clang hanging (XC8-1798) The compiler hung when building for C99, it encountered code that declared an incomplete array and later redeclared a complete version of the array in the same translation unit.

Multiplication errors (XC8-1770) For PIC18 projects being built for speed, expressions involving multiple instances of 16-bit multiplication that used certain operands might have produced an incorrect result due to the `PROD` register being clobbered mid-calculation.

int$flags undefined (XC8-1768) The compiler might have issued an undefined symbol error for the symbol `int$flags` when building small PIC18 projects that used interrupts.

Redefinition errors with library interrupts (XC8-1766) When linking libraries that contained interrupt functions, a conflicting declaration or redefined symbol error might have occurred.

Errors with C99 interrupts (XC8-1764) Macros passed as arguments to the `__interrupt()` specifier might not have been expanded when building for C99 PIC18 projects, resulting in a compilation error.

EI-DI-O (XC8-1719) The `ei()` and `di()` macros, which enable and disable, respectively, the global interrupt enable bit, were not defined for some Enhanced Mid-range device. The affected devices have the identification `PIC14EX` in the compiler's `chipinfo.ini` file. In addition, these same devices were missing the prototype for the `_delay3()` builtin function, which would have resulted in a an error if that function had been used in a project. All these macros and functions are now available for these devices.

Return value clobbers other objects (XC8-1709) In some cases when a function returned a pointer to a structure type that contains a function-pointer member, the compiler may not have allocated space on a compiled stack for the return value, resulting in it clobbering other objects.

Can't generate code with Free license (XC8-1660) Case label values that used expressions involving the conditional (ternary) operator might have triggered can't generate code errors when using an unlicensed compiler or the optimizations were disabled.

Can't generate code with Free license (XC8-1600) In some instances, initialization of `const`-qualified objects with a complex constant expression might have issued a can't generate code error when using an unlicensed compiler or the optimizations were disabled.

Bad conversion of pointer return value (XC8-1590) In some Enhanced Mid-range projects, incorrect code was generated for the conversion of a 1-byte RAM pointer to a 2-byte pointer. Specifically, in the case where the 1-byte pointer contents was a return value that was used immediately by the `return` statement of another function which returned that pointer.

Can't generate code for pointe dereferences (XC8-1579) Can't generate code errors might have been produced for complex projects that defined structures that contained function pointers members.

Can't generate code in if() (XC8-1384) In instances where an `if()` statement in a Mid-range project had no body statements and the controlling expression had `volatile` identifiers, a Can't Generate Code error might have been produced.

Too positive (XC8-1244) Code in PIC18 projects that used a reentrant stack was printing negative floating-point values as positive rather than negative values. This only affected those projects using the C90 libraries.

Incorrect stack allocations (XC8-1105) Projects assigning incompatible function addresses to a function pointers might have experienced memory allocation issues with the indirectly referenced functions' stack-based objects.

5.9. Version 2.00

Timer values (XC8-1743) When comparing timer registers with a constant value, the compiler might have avoided reading the lower byte of the timer register, which would result in the entire timer register not being updated correctly. This optimization is no longer applied to objects specified as `volatile`.

Negative zeros (XC8-1694) The compiler had not been capable of generating a negative zero floating-point constant. This has been corrected, and the constant `-0.0` will be encoded with the sign bit set.

Unwelcome degenerates (XC8-1684) In expressions where a compiler optimization replaced the use of a variable with a literal constant zero, the compiler might have warned about the variable no longer being

used or being degenerate in comparisons/relational expressions.

Delay errors and inaccuracies (XC8-1677) If the parameter to the built-in delay routine was not a constant value (e.g. a constant expression), the compiler might have issued undefined symbol errors, or generated an inaccurate delay. An error will now be issued if the delay argument is not a constant value.

pic.h rejected (XC8-1676) When building for non-PIC18 devices, errors might have been produced for code in the `pic.h` header file when the `--ADDRQUAL=reject` option was specified.

Inaccurate device memory report (XC8-1675, XC8-1650) For some devices, the total available program or data memory reported in the memory summary after compilation might not have been accurate. This issue did not affect the reported amount of memory used.

Function pointer holding data address warning (XC8-1672) Assigning ((void *)0) to a function pointer might have incorrectly generated a warning sating that a function pointer cannot be used to hold the address of data. This issue will be corrected when building with the C99 standard and the Clang front end.

Bogus Arithmetic overflow warnings (XC8-1671) Certain complex constant expressions might have produced arithmetic overflow warnings for valid code when compiled with level 0 optimizations.

Can't generator code for case label (XC8-1660) When compiling with a non-zero level optimization, the compiler might have issued a Can't generate code error for `case` label expressions in `switch()` statements involve the `? :` operator. This has been corrected; however, the error still exists with level 0 optimizations.

Data corruption with free mode context save (XC8-1638) When operating in Free mode, the compiler might have saved context to `btemp` registers that were never defined. This issue will no longer occur, and additionally, the list of registers saved by Free mode more closely matches that of PRO mode.

Librarian crash (XC8-1634) When extracting modules from a library under Windows, the librarian might have crashed when creating a directory.

Can't find space (XC8-1609) For PIC10/12/14/16 projects using non-default linker options and where one or more regions of program memory were quite small, a can't find space error might have occurred, even though there was sufficient remaining space.

No type match error when shifting (XC8-1606) When building PIC18 projects with level 2 optimizations, expressions involving right shifts by 8 bits and a conversion of the result to a smaller type might have produced a no type match error.

Context save corruption with reentrant stack (XC8-1604) When building for enhanced mid-range devices in Free mode and using the reentrant stack, the interrupt context save code was not correctly mirrored by the context restoration code, resulting in data corruption. This issue will no longer occur, and additionally, the list of registers saved by Free mode more closely matches that of PRO mode.

Unterminated macro definitions (XC8-1514) The parser was compiling without error preprocessor macro definitions which were missing closing parentheses. This issue will be corrected when building with the C99 standard and the Clang front end.

Can't find space errors (XC8-1596, XC8-1273) When building for Enhanced Mid-range projects, a can't find space error might have been emitted where an absolute linear-memory object was positioned in the same bank as another absolute non-linear memory object, even though their addresses did not overlap.

Bogus signed to unsigned warning (XC8-1586) In some instances, when an integer expression used as an array index was promoted to an int, the generation of a implicit signed to unsigned conversion warning might have occurred. This issue will be corrected when building with the C99 standard and the Clang

front end.

Bad intermediate code (XC8-1560) In some cases, where a member of a structure had a typedef'd type, that was used in a previous definition before the structure, the code generator emitted the error message "bad intermediate code". This issue will be corrected when building with the C99 standard and the Clang front end, but for legacy projects an error is now emitted and the issue can possibly be worked-around by ensuring that the structure is the first definition to use the typedef'd type.

Syntax error reported for conditional operator (XC8-1536) In complex expressions involving the ternary operator, the compiler might have incorrectly issue a expression syntax error. This issue will be corrected when building with the C99 standard and the Clang front end.

Incorrect structure initializer not detected (XC8-1530) Code which initializes a structures with an integer type was not detected. This issue will be corrected when building with the C99 standard and the Clang front end.

Crash over include paths (XC8-1527) When compiling for C90 and using relative paths involving forward slashes in include statements, the compiler might have experienced a crash on some platforms. This issue will be corrected when building with the C99 standard and the Clang front end.

Array argument conflict (XC8-1467) Some arrays of pointers when passed to a function might have triggered an argument conflict with prototype error, even thought the type of the argument appeared to match that required by the prototype. This issue will be corrected when building with the C99 standard and the Clang front end.

Valid declarations marks as conflicting (XC8-1466) Declarations for an object using the `static` specifier that were accompanied by definitions of the same object but that omitted the `static` specifier produced a warning and error, stating that there had been a redeclaration of the object with a different storage class. This issue will be corrected when building with the C99 standard and the Clang front end.

Incorrect parsing of backslash in comments (XC8-1460) The handling of the backslash character when compiling for C90 was incorrect when used within comments. This issue will be corrected when building with the C99 standard and the Clang front end.

Debugging line issues (XC8-1438) Using a macro to represent the header file of a `#include` preprocessor directive resulted in incorrect line number information being contained in compiler debug output files (ELF or COFF). This issue will be corrected when building with the C99 standard and the Clang front end.

Macro expansion (XC8-1413) The preprocessor did not correctly expand preprocessor macros whose replacement text required several levels of subsequent macro expansion. This issue will be corrected when building with the C99 standard and the Clang front end.

Can't find space for large const objects (XC8-1404) When building for enhanced mid-range projects, `const` objects placed at an absolute address (using either `@` or `__at()`) had their maximum size incorrectly limited to the size of a program memory page.

Malformed hex constants (XC8-1393) Non-zero digits preceding the `x` character in what was intended to be a hexadecimal integer constant (e.g. `1xFF`) might have been incorrectly accepted by the compiler. This issue will be corrected when building with the C99 standard and the Clang front end.

Fixup error with __IT_INT_MASK (XC8-1382) On rare occasions, PIC18 projects building with the REALICE debugger enabled, might have experienced fixup errors for the symbol `__IT_INT_MASK`.

Can't generate code with offsetof macro (XC8-1374) Mid-range projects might have produced can't generate code errors when using the offsetof macro.

Undetected redefinitions (XC8-1373) In some instances, multiple definitions of the same local variable (which is not permitted by the C standard) was not detected by the parser. This issue will be corrected when building with the C99 standard and the Clang front end.

Crash when processing assembly source (XC8-1342) If an assembly module contained an empty psect that was positioned via an `ORG` directive, the driver might have crashed.

Code generator crash (XC8-1338) In PIC18 projects, initialization of arrays of structures containing array members using complex macros, might have caused the code generator to crash. This issue will be corrected when building with the C99 standard and the Clang front end.

Invalid array dimensions (XC8-1336) The compiler might not have issued an error when the dimension of an array was a constant expression with a negative value (e.g. `0 ? 1 : -1`). This issue will be corrected when building with the C99 standard and the Clang front end.

Unhelpful error (XC8-1325) An error message resulting from an unterminated `#if[n][def]` block might not have mentioned the name of the file in which the error was present.

BASEM undefined with printf (XC8-1296) Using the `(s)printf` format specifiers `%#08lx` might have resulted in an error for the undefined symbol `BASEM`.

Error with enumeration value (XC8-1284) Setting an enumeration value to be the size of a structure (using the `sizeof` operator) might have generated an error. This issue will be corrected when building with the C99 standard and the Clang front end.

5.10. Version 1.45

Bad initialization of linear-memory objects (XC8-1601) For enhanced mid-range projects that define multiple large objects that have been allocated to the linear address space, the initialization or clearing of those objects might not have been accurate.

Device information tables incorrect (XC8-1597) Macros in the device-specific headers that defined the address of entries in the "Device Information Area" and "Device Configuration Information" tables were incorrect.

Bad access to members after a pointer (XC8-1559) In projects built for enhanced mid-range devices, where a pointer was a structure member and that structure was located in and accessed via linear memory, access of the members following the pointer might not have worked as expected. This problem was only occur for some pointers, most likely pointers that only referenced a small number of data objects.

5.11. Version 1.44

Missing bank selection (XC8-1573) In PIC18-targeted projects that used an expression involving a 2-byte value in one bank added to a 3-byte pointer in a different bank, the bank of the destination might not have been correct selected and the upper-byte of the result was written to the wrong bank.

Crash on prototyping error (XC8-1572) Non-conforming programs that should have ordinarily emitted error (277) (relating to function prototyping), were instead causing the compiler parser to crash.

Warning on large Hexmate arguments (XC8-1426) Hexmate can now warn you if an argument read in via its options is too large. Previous, the option value might have been mis-read and led to unexpected results.

Incorrect fill (XC8-1423) Using Hexmate to fill unused locations with a four-byte fill value did not work as expected.

Undefined symbol in reentrant mode (XC8-1337) For PIC18 projects compiled using a reentrant model, the assignment of a 2-byte object to a 3-byte pointer might have resulted in a "plusw1" (or similar) undefined symbol error.

Wrong char specification macros (XC8-1322) The macros in `<limits.h>` relating to the sizes of `char` types were defined as if a `char` was signed. The `char` types are unsigned and the `CHAR_MAX` and `CHAR_MIN` macros have been updated with the correct values.

Wrong memory sizes displayed (XC8-1313) The size of the memory spaces reported in the compiler's memory summary did not reflect the use of `--ROM` or `--RAM` when these options used arguments that subtracted from the default memory (e.g. `--rom=default,-0-3ff`).

Missing read macro (XC8-1309) The `_READ_OSCCAL_DATA()` macro was not defined for those devices that used a oscillator calibration constant.

Unruly unlock sequences (XC8E-101) Source code which performed unlocking of peripherals, such as EEPROM or PPLSOCK unlock sequences, were not output in strict accordance with the device data sheet, resulting in the correct registers being written but not with the correct timing. This occurred most commonly in Free mode. Optimization of such sequences are no longer performed, and C source which performs the sequence should work as expected.

5.12. Version 1.43

Wrong ROM size for 18LF47K42 (XC8-1583) The compiler's chipinfo file indicated that there was less program memory present than on the actual device. The available memory now matches the device data sheet.

Wrong ROM size for 16(L)F19185 (XC8-1580) The compiler's chipinfo file indicated that there was more program memory present than on the actual device. The available memory now matches the device data sheet.

Variation in locally optimized code output (XC8-1575) Due to inconsistent bank-selections or banked access in the generated code, the instruction sequence could vary between builds that employed localized optimizations. The instruction sequences were functionally correct, but not consistent with the intent of localized optimizations.

Bogus pointer-conversion warnings (XC8-1567) The compiler produced erroneous warnings for pointer conversions that were entirely legal.

Bad flash macros (XC8-1566) Some PIC18 devices (viz. K40s, K42s and Q10s) were missing flash block write and erase sizes in the chipinfo file, resulting in flash macros (e.g. `_FLASH_WRITE_SIZE`) with bad values.

Syntax errors when copying structures (XC8-1562) When compiling for enhanced mid-range targets, the compiler generated code with assembly syntax errors for statements that copied a source structure located in program memory and a destination structure located on the software (reentrant) stack.

Linker errors from function pointer use (XC8-1558) Projects for midrange and especially baseline devices that used function pointers might have produced can't find space messages resulting from the `functab` psect being linked in a restrictive location. The linker now has more freedom in placing this psect and is less likely to generate memory errors.

Bad optimization around branches (XC8-1557) In some rare instances the PIC18 assembly optimizer factored out instructions common to both execution paths of a branch instruction even though these instructions affected the status flags checked by this branch instruction. This is no longer performed if

status is affected.

Registers clobbered by interrupts (XC8-1555) PIC18 projects that used either the hybrid or reentrant stack model and which contained code in interrupt functions might not have correctly saved all the temporary registers used in the interrupt routine, resulting in potential code failure in main-line code.

Bogus psect overlap message (XC8-1545) When building projects that used user-defined linker options to locate psects, there could have been psect overlap error messages (596) produced by the linker when the requested arrangement seemingly look valid.

Global variables not cleared (XC8-1544) In very rare circumstances, the entry point of the PIC18 runtime startup code was missing a bank selection instruction, which resulted in initialization of the wrong bank of memory before the main-line program was invoked. The conditions to trigger this situation were that the objects to be cleared or initialized were in a bank other than bank 0, this bank was the same as the bank that was selected when generation of the remainder of the program concluded, and there were no other variables to clear or initialize in the runtime startup code in any of the lower banks.

Incorrect signed right shifts (XC8-1543) The compiler produced incorrect code for signed right shifts by 1 bit for `signed long` or `signed short long` variables. PIC18 projects were not affected by this issue.

Bank selection error (XC8-1526) Code following some 3-byte pointer assignments in PIC18 projects might have been missing a bank selection instruction.

Compiler crash (XC8-1495) The compiler might have crashed when building PIC18 projects that defined a function that was never called but was referenced in hand-written assembly code.

Long free-mode build times (XC8-1428, XC8-1525, XC8-1554) For PIC18 projects, typically operating in Free mode, the assembler might have taken a inordinate amount of time to build when the source code contained `while(1)` loops at the end of other loop constructs.

5.13. Version 1.42

Missing PIC18 instruction from assembler (XC8-1550) The PIC18 assembler used a `bnv` instruction instead of `bnov`, as documented in the device data sheet. Both instruction mnemonics are now accepted.

Wrong coverity layout (XC8-1546) The coverity files supplied with the compiler were in the wrong directory layout.

Missing local optimizations (XC8-1540) When local optimizations were selected, these optimizations were not applied to functions that specified a return type of pointer to function.

Sloppy detection of conflicting function types (XC8-1538) The compiler did not detect conflicts between a function declaration and function definition when the definition was in a separate module and there was a difference in the return type specified.

Wrong bank selection after bit operations (XC8-1532) In PIC18 projects, the code following an `|=` or `&=` operation that used bit operands (including single bit bit-fields) might have accessed an operand in the wrong bank, resulting in code failure.

Incorrect initialization of linear-accessed objects (XC8-1529) Enhanced midrange projects that defined several initialized objects that were large enough to be accessed via linear memory might have initialized the objects with incorrect values.

Incorrect floating-point characteristics (XC8-1528) Macros in `<float.h>`, which describe the floating-point characteristics (e.g. `FLT_EPSILON`), were incorrect when the `float` type was set to 32-bits. See

also Migration Issues.

Better handling of external objects (XC8-1521) The compiler was not using bank specifiers in declarations for externally-defined objects to determine the required bank when accessing these external objects. See also Migration Issues.

Wrong bank selection after large-object access (XC8-1520) In PIC18 projects, code following that which accessed large, multi-bank objects, might have been missing bank selection instructions if the bank selected before accessing the multi-bank object was the same as that required after accessing the multi-bank object.

Bogus overflow warning (XC8-1519) Expressions which involved subtraction of the result of the `sizeof()` operator might have produced an arithmetic overflow warning when this was not expected.

Incorrect memory layout for enhanced midrange devices (XC8-1518) There were errors in description of banked and common memory for some enhanced midrange devices that might have triggered unexpected memory-related errors or warnings. The following devices and their LF counterparts in the PIC16F191*xx* range, where *xx* is 55, 56, 75, 76, 85, 86, 95, 96 and 97, were affected.

Overzealous assembler optimizer (XC8-1511) When optimizing a code involving a bit-skip over a GOTO and subsequent conditional code, the PIC18 assembler optimizer might have incorrectly removed an instruction (such as MOVLW, BSF, BCF, MOVWF or CLRF) in addition to the GOTO instruction.

Disregard of regsused pragma (XC8-1503) The compiler would not honor the `#pragma regsused` directive when used without a list of registers. The compiler subsequently assumed that the specified function would use all device registers rather than none.

Pointer believed to be unitialized (XC8-1492) When assigning to pointers inside structures that are themselves within other structures, the pointer target information was occasionally not propagating to the destination of the assignment. In such situations, the compiler assumed that the destination pointer was NULL and dereferencing this pointer would have produced the incorrect result.

Driver not handling off-chip code offsets (XC8-1481) The compiler was not able to correctly handle situations where a code offset higher than the top of internal (on-chip) memory had been specified, as is possible for PIC18 devices with the external memory interface.

Bogus warnings in library code (XC8-1476) Building projects that contained interrupt functions that called standard library routines might have produced inappropriate warning messages, such as warning 1496, `arithmetic on pointer to void yields Undefined Behavior`.

Unable to find space for const objects (XC8-1469) For an enhanced midrange devices, all constants were grouped into a single psect, which potentially could have grown to exceed the size of the largest contiguous block of free space. Const objects are now allocated to individual psects so that they can more easily be allocated memory.

Pointer truncated (XC8-1449) For enhanced mid-range devices, in some circumstances pointers that are members of structures and that point to objects located in bank1 or higher addresses might have incorrectly been dereferenced.

Can't generate code error (XC8-1448) In rare instances, a function that returned a `void` pointer in memory shared by one of that function's parameter variables might have triggered a `looping around allocGlobals()` error.

Bogus errors regarding function type checks (XC8-1446) Where a pointer to an incomplete structure type was referenced in a function declaration, the compiler incorrectly reported that the function declaration conflicted with the function's definition.

Huge objects not properly initialized (XC8-1445) In PIC18 projects, `const` arrays that were defined with more than 32768 elements and that were required to be assigned zero at startup were not properly cleared.

Truncated pointers to void (XC8-1443) For PIC18 projects that made use of pointers to void and function pointers, the pointers to void might have been made smaller than required and were therefore unable to correct dereference some objects.

Incorrect error message (XC8-1440) If an attempt to initialize an absolute eeprom-qualified object was made, the error message for this illegal operation incorrectly referred to RAM instead of EEPROM.

Code generator hangs processing loops (XC8-1433) Some projects that continued more than one non-nested infinite loop might have caused the code generator to hang.

Incorrect function return value (XC8-1416) For projects targeting any device, excluding PIC18 devices, and in rare situations where a function returned one of its parameters and the memory allocated to that parameter and the return value partially overlapped, the return value of that function might have been invalid.

Corruption when copying zero bytes (XC8-1409) Projects targeting enhanced midrange devices and that used the `memcpy()` library function might have experienced data being corrupted when a zero size was passed to this function.

Incorrect results for expressions with indirect-access (XC8-1408, XC8-1487) Expressions in PIC18 projects that involved an operand that indirectly accessed RAM objects, e.g. array access, and that also involved a constant operand might have produced incorrect results due to corruption of the WREG by the code that loaded the FSR register.

Wrong bit values shown in the MPLAB X IDE (XC8-1312, XC8-1418) Absolute-addressed bit variables might not have been accurately displayed in the MPLAB X IDE debugger. This affected SFRs as well as user-defined bits.

Memory error when using absolute variables (XC8-1301) In rare cases, the compiler might not have been able to allocate space for RAM-based variables in projects that also defined absolute variables, even when there appeared to be adequate memory for all objects.

5.14. Version 1.41

Missing errata workarounds (XC8-1512) The default errata workarounds applied for the PIC18LF6585/6680/8585/8680 devices, did not agree with their PIC18F counterparts. The DAW and FETCH workarounds might not have been applied for these devices.

Improved internal sorting of symbols (XC8-1494) Part of the compiler's internal sorting of symbols was dependent on a variable's name. Recompiling source code where the only change was the names of variables might have resulted in those variables being allocated to different memory locations and hence a variation in the final output file. This reliance has been removed.

Bad pointer comparisons (XC8-1414) A Can't Generate Code error might have been issued for mid-range device projects that compared two pointers referencing RAM objects.

Incorrect access of pointer parameters (XC8-1411) In rare circumstances when building PIC18 projects, the compiler incorrectly interpreted the location of function parameters that were pointer objects, and it produced code which accessed the pointer in the wrong bank.

5.15. Version 1.40

Bogus error with interrupts in CCI mode (XC8-1435) For PIC18 projects built in CCI mode and that contained a low- and high-priority interrupt function defined using the `__interrupt()` specifier, the compiler erroneously reported that there were multiple interrupt functions defined at interrupt-level 2.

5.16. Version 1.38

Can't generate code errors using hybrid stack model (XC8-1431) For PIC18 projects using the hybrid stack model, can't generate code messages might have been produced for functions using the compiled stack that had a function pointer parameter and that pointer had function targets which were compiled using the reentrant model.

Incorrect pointer content shown in MPLAB X IDE (XC8-1415) For projects that targeted non-PIC18 devices and that defined a 1-byte-wide pointer that included NULL as one of its targets, the MPLAB X IDE debugger might not have shown its contents correctly.

Incorrect indirect addition (XC8-1402) Enhanced mid-range projects that performed a 16-bit addition of two dereferenced pointer values might have seen corruption of an SFR register resulting in an incorrect result.

Registers unavailable error when accessing SFRs (XC8-1391) In some circumstances for projects using a PIC18 device that is affected by the BSR15 errata and that have SFRs in bank15, accessing these SFRs might have resulted in a `(1466) registers unavailable for code generation` error.

Assembler hang (XC8-1390, XC8-1396) In some circumstances for PIC18 projects when the compiler is operating in Free mode with assembler optimizations enabled, the assembler might have hung.

Faster Hexmate (XC8-1385) The time required to merge HEX files using the Hexmate utility has been significantly reduced.

Section intcode overlap (XC8-1381) Projects compiled using PIC18 devices affected by the FETCH or 4000 errata, and which defined both low- and high-priority interrupt functions, might have produced the compiler message `(596) segment "intcode" overlaps segment "intcodelo"`. This was due to a miscalculation in the size of the psect which contained the high-priority interrupt code.

Speedy multiplication failure (XC8-1372) For PIC18 projects which have speed optimizations enabled, an incorrect bank selection might have produced a 16-bit multiplication error.

Upper pointer byte corruption (XC8-1369) In rare circumstances in projects targeting PIC18 devices in which there is a function that returns a 24-bit pointer, the upper byte of this pointer might be corrupted.

Incorrect variable startup clearing (XC8-1320) In rare circumstances for mid-range and baseline projects that had an object located immediately before the compiler temporary variables and this object had to be zeroed by the runtime startup code, and the runtime startup code had to perform other variable initialization, the zeroed object might have been corrupted.

5.17. Version 1.37

Reentrant specifier ignored (XC8-1379) The `__reentrant` function specifier might have been ignored and subsequently the function may not have been made reentrant. A warning would have been issued if this situation was encountered.

Local object overlap with invariant functions (XC8-1376) For PIC18 projects using stable-object (invariant) functions, there might have been an overlap in local objects between main-line and interrupt code resulting in data corruption.

Incorrect bank selected for external bits (XC8-1375) In some circumstances the compiler generated

incorrect bank-selection code, when accessing external bit variables located in the banked memory of PIC10/12/16 parts. This only affected bit variables defined outside C code, the most notable examples being the `__powerdown` and `__timeout` bits used to identify causes for reset.

Parameter corruption (XC8-1362) For extremely complex expressions involving function calls (including implicitly called library routines) a call to one function might have corrupted an already loaded parameter for another function.

Bogus warning with offsetof macro (XC8E-88) Use of the `<stddef.h>` macro `offsetof` might have resulted in the compiler emitting an unnecessary warning regarding arithmetic overflow.

5.18. Version 1.36

Variables incorrectly shown as program memory objects in the IDE (XC8-1370) Some absolute, non-`const` variables were being shown as program memory objects in the MPLAB X IDE watch window. These will now correctly appear as data memory objects.

Const parameters incorrectly accessed (XC8-1365) For PIC18 projects, functions that had parameters specified with `const` might not have been accessed correctly.

Wrong bank selection when right shifting by four (XC8-1364) For Baseline and mid-range devices, an incorrect result might have been obtained for expressions that shifted a 16-bit-sized object right by 4 bits and assigned the result to an object that was located in a different bank.

Code generator crash (XC8-1363) In some rare situations involving the intensive use of function pointers, XC8 might have crashed.

Access to const array incorrect in stable-object mode (XC8-1361) In some circumstances for PIC18 project when stable-object mode (formerly called invariant mode) is used, `const` objects may not have been accessed correctly via a pointer.

Unavailable global variables (XC8-1359) In projects where functions have been inlined by the compiler, some variables unrelated to those functions might appear as "Unavailable" in the MPLAB X watch window.

Code generator crash (XC8-1357) Certain questionable code sequences were causing the code generator to crash. An internal error message is now issue instead.

Output changes with each build (XC8-1345) Code that uses structures with pointer members has, on rare occasions, resulted in some pointer target lists that vary with each build. This manifested itself in build sizes that varied with each build when using identical source code.

Unsigned OS X installer (XC8-1343, XC8E-79) The MPLAB XC8 installers for OS X were not code signed, requiring the OS X security level to be lowered to allow installation of the compiler. All OS X installers are now signed.

Incorrect pointer assignment (XC8-1341) In some situations with PIC18 projects, the upper byte of a 2-byte pointer might have been incorrect assigned.

The hanging of Cromwell (XC8-1339) In some situations, possibly when using user-created libraries, the cromwell application might have hung if it was unable to resolve relative file names.

Apparent reset when using inline functions (XC8-1335) In some instances where a function using the `inline` specifier were not inlined, the function was incorrectly being removed. This resulted in a call to address 0x0, which would have appeared like a software reset.

Incorrect bit-field assignment (XC8-1334) For some mid-range and baseline projects, assignments to a 1-

bit wide bit-field variable of an integer might have stored an incorrect value in the bit-field.

Corrupted page-selection bits on entry to interrupt (XC8-1333) This issue affected enhanced baseline devices with interrupts. The page-selection bits in the interrupt's STATUS register might have been in an unknown state upon entry. Any code that relied on these bits might have been adversely affected.

Illegal directive warning (XC8-1329) The C preprocessor was emitting a warning when it encountered an unrecognized preprocessor directive that was excluded from compilation by an outer preprocessor conditional directive.

Errors when using EEPROM features (XC8-1324) The compiler did not support EEPROM access for those devices which used the NVMREG register interface. This has been added a new feature in this release, see New Features.

Local variables marked as 'out of scope '(XC8-1323) On some rare occasions, some local variables would appear as 'out of scope 'in the MPLAB X IDE watch window.

Wrongly formatted Messages from OBJTOHEX (XC8-1317) The compiler utility OBJTOHEX did not format its messages according to that specified by the driver options `--errformat`, `--warnformat`, and `--msgformat`.

No error for multiple interrupt functions (XC8-1316) The compiler was not detecting multiple instances of an interrupt function at the same interrupt priority.

Bogus NOPs placed after labels in data space (XC8-1315) Compiling code for PIC18 devices that are affected by an errata workaround that require NOPs to be inserted after labels might have had NOPs added after labels in some data space psects. This might have triggered a message `(944) data conflict at address....`

Bad indirect bit-field access (XC8-1314) The wrong bank might have been selected when indirectly accessing a 1-bit wide bit-field in a structure and converting/assigning to a byte sized object in PIC18 projects. This issue was likely to only occur where the structure involved was greater than 128 bytes in size.

Bad copy of large structures (XC8-1311) The code generated to copy structures greater than 128 bytes in size might have corrupted data memory and not correctly copied the upper structure members.

Assembler hangs with trivial ISR (XC8-1308) PIC18 projects that defined an interrupt function that contained only an infinite loop might have caused the assembler to hang when optimizing this code.

Misleading data usage in the MPLAB X IDE (XC8-1305) Programs that used the software stack may have shown an unusually large data usage in MPLAB X IDE that did not match the data usage reported by the compiler.

Incorrect allocation of local objects (XC8-1304) For projects using the compiled stack for auto objects and where function duplication was prevented using the `#pragma interrupt_level`, stack allocations of that non-duplicated function might have overlapped with that of other functions in the main-line call-graph.

No warning given for invalid floating-point specification (XC8-1302) The compiler was mistakenly permitting the floating-point suffix, `f`, to be applied to decimal integer literal constants.

Fletcher 8 checksums (XC8-1297) Projects that used publicly-available or written-in-house algorithms to perform the Fletcher 8 checksum algorithm (8-bit calculation; 16-bit result) might have seen that the checksum generated by their code did not match that produced by Hexmate due to an overflow issue. The algorithm implemented by Hexmate has been updated and its result confirmed to match that

produced b the code now published in the MPLAB XC8 User's Guide. If you do require the old value produced by Hexmate for legacy projects that are being recompiled with no code changes, use algorithm #9.

Parser crash (XC8-1283) In certain circumstances for extraordinary long C source lines, the compiler was crashing instead of exiting gracefully.

Code generator crash (XC8-1275) In rare circumstances, projects that used the compiled stack and defined a function called only indirectly from both main-line and interrupt code might have crashed the code generator.

No warning given for redundant pointer access (XC8-1274) The compiler was not given any notification that some expressions involving pointers with no targets were being optimized away by the compiler. Unassigned pointers might now trigger messages 1498, 759 or 760.

Missing logical operation (XC8-1271) For PIC18 projects that contain expressions involving a right shift by 8 bits and a logical OR with a constant, the resulting expression might have erroneously been optimized away. This only affect projects compiled in PRO mode.

CMF error (XC8-1257) Some programs that have multiple user defined psects of the same name may produce the error `(1437) CMF error: segment (text) already defined`.

Too much warning (XC8-1231) In some circumstances, message `(1467) read-only target may be indirectly written via pointe`r was displayed excessively.

5.19. Version 1.35

MPLAB X IDE update If you encounter the MPLAB X IDE error `The program file could not be loaded` for projects that use the compiler's (default) ELF output, please update your IDE to at least version, 2.30. There was an issue in the ELF loader that triggered this error and prevented projects from being debugged, but which has been resolved. If you cannot update the IDE, switch your project settings to COFF output.

Limited comment lines in listing file (XC8-1294) The assemblers were limiting the number of lines of comments in the assembly listing files to around 5000 lines. All subsequent comments were not output. This issue did not affect the generated code.

Crash with invariant functions (XC8-1285) For enhanced mid-range projects that contained at least one function qualified as `invariant`, expressions involving a comparison with a pointer that had never been assigned a value might have cause the code generator to crash.

Non-functioning error format option (XC8-1282) The `-E` option, when used without specifying a filename, was not toggling the compiler warning and error message formats to the machine-readable form. This issue did not affect use of this option when a filename was specified.

Incorrect data sizes in PIC18 memory summary (XC8-1277) The total number of data bytes shown by the memory summary was higher than the size used by the program. This affected only PIC18 projects and might have resulted in a summary showing more than 100% utilization of the available data space.

Conditional assembly directives (XC8-1266) The PIC18 assembler would have produced an error if the assembler optimisers were enabled and the ELSIF directive was used in inline assembly.

Can't Generate Code message initializing multi-dimensional array (XC8-1265) Code for baseline and mid-range projects that defined multi-dimensional `const auto` arrays might have triggered a can't generate code error.

Incorrect assignment of constant to bit-field (XC8-1259) When assigning a constant to a multi-bit structure bitfield in a PIC18 project, the value might have been assigned to an address in the wrong data bank.

Using ternary operator with bit-fields (XC8-1258) PIC18 projects using the ternary operator (`?:`) whose condition expression tested a single-bit bit-field and whose result was assigned to a different bit within the same structure might have giving an incorrect result, e.g. `bits.out = bits.in ? 0 : 1;`.

Alert system configuration not working (XC8-1256) The `--ERRORS` driver option, which limits the maximum number of errors per application, and the use of environment variables that specify the message formats, were not working correctly.

Incorrect bank selection in functions passed a byte (XC8-1255) If the first instruction generated from the first statement in a function was a MOVFF instruction, and that function's first parameter was a byte that was stored in WREG, the compiler output following the `MOVFF` instruction might have been produced based on incorrect knowledge of the currently set bank. This might have caused code failure in the code following the MOVFF instruction in that function or in the code following a call to that function.

CPP crash, primarily under Windows 7 (XC8-1253) In rare and specific circumstances, the preprocessor (CPP) might have crashed. This was only been reported under some variants of Windows 7.

Can't generate code error calling external reentrant functions (XC8-1251) C code calls to external (hand-written assembly) functions that are declared as reentrant would have triggered a Can't Generate Code message in PIC18 projects.

Undefined symbol error (XC8-1249) In rare circumstances the PIC18 assembler optimizer might have incorrectly removed some code labels. This might have resulted in an undefined symbol error from the linker.

Phase error (XC8-1248) A phase error might have been produced if the assembler optimizers were enabled and a psect contained so much hand-written assembly or special instructions (like `NOP` etc.) as to fill a program memory page on the device. A more accurate Can't find space message is now issued in such (rare) circumstances.

Missing register definitions (XC8-1239) Declarations for the `PLUSW`*x*, `PREINC`*x*, `POSTINC`*x* and `POSTDEC`*x* SFRs were missing from the device-specific header files for PIC18FxxJ9x parts. Attempts to use these symbols in hand-written assembly code would have resulted in an undefined symbol error. The missing definitions would not have affected code produced by the compiler.

Inability to set breakpoint on first statement (XC8-1232) In some situations it was not possible to set a breakpoint on the first executable statement inside `main()`.

Fixup error access bits (XC8-1230) In some midrange PIC projects, comparison of `bit` objects may have resulted in a fixup error.

Right shift clobbering operand (XC8-1224) Code in PIC18 projects that right shifted the value held by a `signed char` variable by 2 or 3 bit positions might have clobbered the variable operand.

No .dep file produced (XC8-1220) The `--SCANDEP` option was not producing a `.dep` file in addition to the `.d` dependency file. The `.dep` file contains both user and system header file information (unlike the `.d` file, which contains only user header file information).

Long subtraction bank selection issue (XC8-1219) Code that performed a long subtraction and assigned the result to a temporary variable might have failed if the bank of the left subtraction operand was in the same bank as the temporary result location, and the bank of the right subtraction operand was in a

different bank. This only affected non-enhanced mid-range PIC devices operating in PRO mode.

Bogus pointer target message (XC8-1192) For those PIC devices which allow `eeprom`-qualified variables, a bogus error might have been issued if assigning both `eeprom`-qualified structure members and non-`eprom`-qualified structure members to an object.

(U)INT64 type defined but not supported (XC8-1189) The `GenericTypeDefs.h` header allowed objects to be defined with the `INT64` or `UINT64` types. These types defaulted to a 32-bit C type, which was misleading. These types are no longer defined by the header.

Failure to break after inline delay (XC8-1186) Attempts to break at, step over, or run to the source line immediately following an inline delay built-in function (e.g. `__delay_ms()`) might have failed. This is a silicon issue which affects the ability of some devices to stop execution at a hardware breakpoint that is placed immediately following a branch instruction that branches to that breakpoint location. Such an instruction was sometimes used by the compiler's inline delays. The compiler will no longer use this instruction in delays when compiling for a device that is known to exhibit this issue. This issue will affect any `BRA` instruction with the operand described. See also new errata workaround.

Bad Bit-field assignment (XC8-1184) Code in PIC18 projects that performed an assignment, assignment-OR, or assignment-AND of a one-bit-wide bitfield to another one-bit-wide bitfield might have failed if the bit-fields were part of a structure in an array.

Can't generator code error with const initialization (XC8-1171) When trying to initialize a `const`-qualified object with a value expression that also contained a `const`-qualified object, a Can't generate code error message might have been issued.

Indirect calls from mainline and interrupt code (XC8E-30, XC8-132) A function was not able to be indirectly called from both main-line code and interrupt code when compiling for PIC18 targets. This previous known issue has been corrected and is now longer pertinent.

5.20. Version 1.34

Code warnings with library code (XC8-1227) With the warning level set to -3 or lower, an `expression generates no code` warning might have been displayed for the `printf()` (or other library) functions. These warnings are no longer emitted for code that is part of a library function.

Incorrect structure addresses used (XC8-1218) In baseline or midrange projects, code that accessed a member of an absolute structure might have used an incorrect address.

Interrupt routine may clobber temp (XC8-1217) When compiling reentrant functions for enhanced mid-range projects, the compiler was inadvertently setting a temporary flag in memory that had not been allocated memory. This could have corrupted other temporary variables. Use of this flag with the software stack was not necessary and is no longer performed.

Wrong stack frame estimate (XC8-1208) In some instances, particularly when invariant optimization was turned on, the code generator did not detect if the stack frame for a function was too large. This may have resulted in assembler operand errors.

Bad call to subtype when disabling libraries (XC8-1205) Using the compiler option to disable C libraries resulted in the error `bad call to subtype`. The compiler requires these libraries to be linked in at all times and the option to disable this feature has been removed. See also Migration Issues.

Bad bank selection (XC8-1196) Expressions involving indirect assignments and comparisons might have

evaluated incorrectly due to the wrong bank being accessed. This only affected PIC18 devices.

Switch statement failure in lined functions (XC8-1187) Functions which were successfully inlined and which contained `switch()` statements might have failed. This would have affected only mid-range devices and only when the chosen switch strategy (as indicated by the assembler list file) was `direct_byte` or `jumptable`. Functions that contain `switch()` statements implemented with these strategies are no longer inlined.

Checksums not matching (XC8-1183) When requesting a checksum for PIC18 projects, the checksum value might not have agreed with runtime calculation of this value if there were unused locations within the memory over which the checksum was calculated. The option to fill unused memory is now automatically applied when using the checksum option. This makes the checksum consistent with that calculated for other devices.

Inaccurate memory summary with absolutes (XC8-1179) The memory summary of data usage printed by the compiler was too large when there were absolute variables that partially overlapped each other in memory.

Outside data space messages (XC8-1177) In some circumstances PIC18 projects using an SFR might have resulted in a warning that the SFR lay outside of available data space.

Too many symbols error (XC8-1176) The parser may have run out of space for symbols when compiling complex modules. The number of symbols the parser can handle has been increased.

Programming ELF file for PIC18F6410 fails (XC8-1175) Programming a PIC18F6410 might have failed when using the ELF/DWARF compiler output, particularly when the `--runtime=+config` option was enabled.

Can't generate code error using memcpy (XC8-1173) A can't generate code error message was sometimes emitted for calls to `memcpy()` when compiling for enhanced mid-range devices.

Assembler crash (XC8-1172) In some circumstances, the PIC10/12/16 assembler could have crashed when processing a `REPT` directive that contained a bit skip instruction.

Incorrect address argument passed to variadic function (XC8-1169) If there are more than one variadic functions that each pass their argument list to the same additional function and the argument list references pointers to objects in different memory space, these pointers may be incorrectly passed.

Wrong %p printf output (XC8-1167) When using the `%p printf()` placeholder to print an address and the address was 3-bytes wide, the top byte of the address might not have been printed and subsequent `printf()` arguments misread and misprinted. This only affected projects compiled for PIC18 targets.

Out of bounds message (XC8-1165) Building PIC18 projects that used the `--ROM` option together with the `--CODEOFFSET` option may have produced in an error stating that the ROM ranges were out of bounds.

Driver crash with unusual file name (XC8-1164) On Windows platforms the XC8 driver might have crashed when an input file name had no extension.

Incomplete support for _CLEAR_EEIF macro (XC8-1163) Use of the `_CLEAR_EEIF()` macro for PIC18Fx7K90 devices resulted in compilation errors. This macro now correctly supports these devices.

Missing bank selection with absolute arrays (XC8-1162) When an array defined at an absolute address is indexed with a variable that is also absolute, a bank selection instruction might have been omitted, causing the wrong location to be accessed. This only affected compilation for PIC18 targets.

Incorrect address initial value (XC8-1161) For mid-range PICs, where the address of a `const` object was required during initialization of another object, the calculated address might have been malformed.

Incorrect compliment/subtraction (XC8-1160) Expressions in PIC18 projects that involved a complement, or possibly a subtraction (most often in Free mode), might have produced incorrect results due to a badly-positioned, bank-selection instruction following an INCFSx instruction.

Incorrect relational comparison of pointers with addresses (XC8-1158) Relational comparison of some pointers with addresses might have failed for PIC18 targets when the address had a different size to the pointer.

Inconsistent preprocessor operations (XC8-1155) The results of preprocessor right shift, division and modulus operations when the operands are signed are implementation defined, but did not match the result of the equivalent operations when performed by the code generator. Expressions in the `#if` preprocessor directive may have produced unexpected results. The results from such preprocessor and code generator expressions now agree.

Functions called from interrupts not inlined (XC8-1153) Functions defined with `inline` and that were called from an interrupt function might not have been inlined when this was possible.

Linker output in wrong directory (XC8-1152) The `l.obj` temporary object file was always being placed in the current working directory, and its location could not be changed by the compiler's `--OUTDIR` option. It is now output in the same directory used by other temporary files.

memcpy copies incorrect number of bytes (XC8-1151) For enhanced mid-range devices, calls to the `memcpy()` function that were encoded to use the inbuilt `memcpy` routine (speed optimisations enabled) may have failed to copy the required number of bytes if the count argument was a variable rather than a constant expression.

Incorrect memory savings report (XC8-1150) When building for PIC18 devices in Free Mode, the estimated savings of using PRO mode were sometimes calculated incorrectly.

Bad const object access when using code offset (XC8-1149) The code produced for PIC18 projects using the `--codeoffset` option with an offset above 0xFFFF might have incorrectly accessed any `const` objects defined in the project.

Inaccurate delays (XC8-1148) For older PIC18 devices that use the 4000 errata workaround, the inbuilt `_delay()` routines did not delay long enough.

Incorrect long intialisation (XC8-1145) PIC18 projects built in Free mode might not correctly initialise `long` objects if the initial value requires performing a 16-bit multiplication.

Bad pointer comparison with NULL (XC8-1144, XC8-1159, XC8-1170) In cases where a pointer containing `NULL` was assigned to a larger-sized mixed-target pointer, the resulting pointer might have had not compared equal to `NULL`.

Bad indirect assignments (XC8-1142) Expressions that involved cascading assignments and where the source and destination operands were accessed indirectly via a pointer might have been incorrect when compiling for an enhanced mid-range device.

Bad code produced for indirect access of arrays in structures (XC8-1141) The code produced for complex controlling expressions in `if()` statements that indirectly accessed a structure member that that was an array might have not correctly loaded the W register, causing code to fail. This only affected PIC18 devices.

Bad shifts (XC8-1140) Shift operations performed for mid-range devices might have failed due to the WREG being clobbered during the operation.

Bad pointer deference (XC8-1139) In PIC18 projects where the generated code loaded the FSR register with

an address that was obtained via a pointer deference plus an offset, it might have clobbered a previously loaded value of WREG and led to code failure. This might have occurred for code that accessed an array/pointer inside a structure, for example.

Cromwell crash (XC8-1137) Programs that contained absolute variables whose name began with `_B`, `_H`, `_I` or `_L` might have cause the cromwell application to crash.

Incorrect boolean assignment (XC8-1136) In situations involving an assignment of a boolean expression to a single-bit bit-field, the wrong value might have been assigned.

Undesirable psect created using __section (XC8-1135) If a `const` object was allocated a new section using the `__section()` specifier, the new psect in which the object would have been located used a psect flag that prevent it from being linked to an absolute address using a linker option.

Assembler hangs (XC8-1133) An interrupt function containing only an infinite loop with an empty body might have caused the PIC18 assembler to become stuck in a loop.

Incorrect pointer initialisation (XC8-1131) Initialisation of a pointer object (at its definition) may have been incorrect when the pointer was assigned different targets during subsequent execution of the program and these new targets were in different memory spaces to the initialisation object whose address was taken.

Bad indirect call from interrupts (XC8-1130) For mid-range devices (excluding enhanced mid-range devices), calling a function indirectly from interrupt code but not calling the same function indirectly from main-line code, might have resulted in the wrong address being called.

Delays not using clearing watchdog timer (XC8-1128) The `_delaywdt()` (and related) delay functions did not use a CLRWDT instruction for some requested delay values or times. A CLRWDT instruction is now always used.

Wrong branch taken for comparison of decremented value (XC8-1125) For expressions which performed a relational comparison of a decremented variable with another quantity, for example `if(x-- <= y)`, the code may have taken the wrong branch if `x` was an `unsigned char` and had the value 0, or `x` was a `signed char` and had the value 0x80. Code would not have failed for equality comparisons (== or `!=`) or for larger operand types.

Incorrect access of external objects (XC8-1118) The compiler might have accessed external objects (for example those defined in assembly code but used in C code) in the wrong bank. This could have occurred in many situations; however, this was unlikely to occur.

Crash defining large data objects (XC8-1117) In programs with very large amounts of initialised data the code-generator might have run out of memory creating its internal data structures. The way the compiler allocates memory for these internal objects has been refactored and improved.

Unused functions trigger CGC errors (XC8-1110) In rare situations where there was a deep tree of unused functions, Can't Generate Code messages may have been issued.

Incomplete cascaded assignment (XC8-1108) A complex statement that performed multiple cascade assignments to `volatile` variables might have failed to correctly perform assignments to all the variables.

Incorrect branch with inline functions (XC8-1107) For PIC18 projects that used the `inline` specifier on functions that generated conditional branch instructions, the destination label of some of these branches might have been incorrect.

Incorrect debugging for static objects (XC8-1104) A deficiency in the compiler's debug information might

have resulted in debuggers confusing the displayed information for a static object if there was more than one such object with the same name.

Incorrect signed relational comparison (XC8-1103) Relational comparison of any signed integers with constants, where the leading byte of the constant was 0x80, might have failed.

Undesirable access of bitfields (XC8-1098) Code that accessed an 8-bit wide bit-field was producing sub-optimal code. This may have caused a failure if this was in a situation where the bit-field was an SFR that must be accessed in a particular way to trigger a hardware event, such as for `PMCON2bits.PMCON2` which controls aspects of the flash memory.

Wrong structure names used in the IDE (XC8-1095) If different structure types that contain equivalent members (in terms of member type, size and order) each contain a structure with the same name but with different member names, MPLAB X might have shown the members of these types using the wrong names.

Wrong integer-to-pointer conversion (XC8-1090) When integer members of structures were converted to pointers, the MSB of the integer might not have been copied over to the destination pointer. This issue could have affected mid-range devices.

Bogus warning with const function parameters (XC8-1086) When calling functions that had parameters specified as `const`, there might have been warning #1467 (pointer used for writes includes read-only target "*") produced when there was apparently no pointer in use. This warning has been suppressed.

Inaccurate small literal floating-point values (XC8-1083) 24-bit literal floating-point values below approximately 5.88E-39 were assumed to be the value 0.0. This did not affect calculations involving float-point variables, nor any 32-bit floating-point values.

Undefined temp symbol (XC8-1081) When building for PIC18 devices and using the reentrant stack model, the symbol `wtemp` might have been undefined when code operated on pointers that only pointed to `NULL`.

Bogus unused variable warning (XC8-1080) If all dereferences of a pointer variable with only one target were optimized to direct memory accesses of that target, the compiler would have misleadingly warned that this pointer was not used.

Corrupted printf output (XC8-1079) For PIC18 projects that made heavy use of the string functions `atof()` and `printf()`, the formatted output from `printf()`'s `%s` placeholder might have had corrupted characters if the strings used by these function were represented by a mix of 16- and 24-bit addresses.

Software stack not correctly unwound after call (XC8-1078) A reentrant function stores its return value on the stack. If this return value was used by some calling expressions and not used by other calling expressions, then the value was not removed from the stack for those cases where it was not used. This only affected the functional integrity of the generated code when such calls occurred within a loop. This behavior could also have affected debugging, such as watching variables with automatic storage that were stored on the stack.

Can't find space for large object (XC8-1074) The PIC18 code-generator might not have been able to allocate space for a large aggregate object containing pointers if there were any absolute variables or memory reservations in general-purpose banked memory.

Bad managed-stack call (XC8-1068) The code-generator might have generated the wrong code for a managed-stack function call (enabled with `--runtime=+stackcall`), resulting in the error `(800) undefined symbol "entry__functionName`, where `functionName` is the called function. PIC18

devices were not affected by this issue.

In-line assembly not preserved (XC8-1063) The PIC18 assembler was not correctly preserving hand-written in-line assembly code. The code might have been modified by some compiler optimizations.

Wrong access of equated symbol (XC8-1054) The first use of an equate (or set) symbol in PIC18 assembly code might have produced an instruction with the wrong RAM access bit. This was only ever likely to affect hand-written assembly code.

Incorrect assignment-add (XC8-1028) Expressions in PIC18 projects performing an assignment-add (+=) of a `signed char` to a `signed short` and where these operands were in different banks might have accessed the wrong address and produced an incorrect result.

Large addresses with __at() not accepted (XC8-1025) The compiler did not permit using values greater than 0x7FFF with `__at()` in CCI mode. This limitation has been removed and large addresses can now be used.

Unable to write flash memory (XC8-1020) The assembler's procedural abstraction optimization may have interfere with the flash write unlock sequence used by mid-range PIC devices that can write to their own flash program memory. This would have only been seen in PRO mode.

Free mode failure of multiply called functions (XC8-1018) Expression containing multiple calls to the same function, especially the functions associated with division or multiplication, might have failed for PIC18 projects in Free mode.

Error issued for printf %p placeholder (XC8-989) In some instances, using the `printf() %p` format specifier might have incorrectly caused the compiler to issue message #975.

Warning for absolute object overlap (XC8-988) The compiler did not produce a warning if an absolute object was defined at an address that overlapped with other absolute objects. A warning is now produced.

Wrong ID location sizes shown (XC8-987) The number of bytes of ID Location memory shown in the memory summary was less than that used by the project.

First byte-sized parameter to reentrant function (XC8-980) For reentrant external functions, passing a 1-byte argument as the first parameter may have produced incorrect results or an inability to generate code. Enhanced mid-range devices resolve this by passing such an argument via the software stack, not in WREG. Argument passing for PIC18 devices still operates the same way, but changes were made to ensure it works correctly.

Inaccurate small floating-point results (XC8-971) Operations that produced floating-point results whose IEEE-format exponent value prior to packing were small (approximately less than 14 for 24-bit doubles; 22 for 32-bit doubles) may have been corrupted by the floating-point pack routines.

Wrong offsetof() result (XC8-952) For enhanced mid-range devices, the `offsetof()` macro might have truncated its result to a single byte when used on members of large (> 255 byte) structures.

Driver crash with malformed options (XC8-948) When compiling on Windows platforms, if `--chip=` was omitted when providing the chip name (i.e. the chip name was listed without the option wrapper), the driver might have crashed.

Compiler crash with pointer conversion (XC8-944) Under some circumstances, code that contained an unsafe conversion between pointer types, would have crashed the compiler.

Incorrect function sizes shown in map file (XC8-933) Interrupt function sizes reported in the map file were often incorrect if the body of the interrupt function was located away from the interrupt vector.

Inaccurate sizes might also have been reported for ordinary functions if the psect that contained their code was split into several smaller psects. A more accurate means of obtaining function sizes has been employed.

Incorrect conversion of pointer to integer (XC8-628) In some cases where a one-byte pointer is cast and assigned to 2-byte integer the compiler would generate incorrect code.

Failure to extract modules (XC8-465) Using LIBR to extract modules from library files produced an error when the modules were originally stored with path information.

Incomplete structure initialization (XC8-280) If the first members of a non-auto structure was written to by an assignment in a function, the remaining members might not have been cleared (or initialised if they are assigned a value at their definition).

Warning issued when buddy function is called (XC8-246) The compiler now warns of potential code failure when it detects that a function has called one of its 'buddies'. Compiled-stack functions that can be called indirectly by the same function pointer are known as buddies. All buddies have their parameter memory aligned so that their parameters can be loaded without knowing exactly which function was called. A function calling a buddy will corrupt its own parameters. This does not affect functions that use the software stack.

Variables not initialised (XC8-228, XC8-1168) If the first use of a variable is in a `switch()` controlling expression and there were no conditional branches prior to this statement in the function, the variable might not have been initialised (either cleared or assigned a non-zero value) at startup.

Error produced for empty initialisation list (XC8-225) The compiler was producing an error for initial value lists that consisted solely of an empty pair of braces. A warning is now issued in such situations and the initial values are assumed to be zero.

No warning for absolutes out-of-bounds (XC8-186, XC8-187) Absolute objects positioned at an address that did not exist in the device did not elicit a warning from the compiler.

Pointer arrays in COFF not shown correctly (XC8-178) Arrays of pointers, in certain circumstances, appeared incorrectly in MPLAB's (X or 8) watch window when using COFF output. The functionality of code was unaffected. The representation of pointers in COFF still has limitations, and it is recommended you use ELF/DWARF output for debugging where possible.

Incorrect expansion of macro next to ## (XC8-96) If the preprocessing token following a ## operator matched a defined macro then the token was being expanded. The C Standard indicates that such tokens must not be expanded.

5.21. Version 1.33

Wrong bank access in indirectly-called function (XC8-1124) Some functions might not have selected the correct bank when they were called indirectly. This could occur when there was a relatively complex expression to determine the function address. This only affected enhanced mid-range devices devices.

Indirect function call corrupts function parameters (XC8-1123) When performing an indirect call to a function and the function pointer itself had to be read indirectly, a temporary variable might have been allocated memory that had the same address as the already-loaded parameters. This would corrupt the parameters and result in code failure. This only affected enhanced mid-range devices devices.

Assembler crashes when inlining code (XC8-1119) If a function was specified as `inline` and this function called other functions that were also inline, the assembler may have entered an endless loop. The same symptom may have occurred if the inlined function contained a loop.

Float/double size in invariant libraries (XC8-1109) The size of `float`/`double` types in an instruction-invariant library were those specified when the library was linked with a project, not on the sizes specified when the library was built.

Instruction-invariant optimizations falsely enabled (XC8-1102) When the `all` suboption to `--OPT` was used, the instruction-invariant optimizations were also enabled. This optimization is now only enabled if it is explicitly requested via a `+invariant` suboption. Note that for some projects, MPLAB X IDE used `--OPT=all` as the base to the optimizations option. For these projects using the 1.32 compiler, customers would have inadvertently used the instruction-invariant optimizations.

Incomplete instruction-invariant coverage (XC8-1101) The `invariant` sub-option of the driver option `--OPT`, was not forcing all functions to be invariant.

Improper access of objects assigned linear-memory address (XC8-1096) For enhanced mid-range PIC devices, code generated for absolute objects that were given an address in the linear address space and whose size was smaller than the size of a bank, may have been accessed incorrectly.

Can't generate code in switch expression (XC8-1041) For PIC18 devices, the compiler may not have been able to generate code when the first element of a larger-than-a-bank-sized array was accessed in a `switch()` controlling expression.

Code generator crashes trying to handle recursive call (XC8-1033) For PIC18 or enhanced mid-range devices, which support reentrancy, the compiler may have crashed in some projects involving recursive function calls.

Wrong bank selected for comparison statements (XC8-1031, XC8-1040) For PIC18 projects using the compiled stack, an `if()` statement comparing a `signed char` variable with a `signed int` variable may have failed. When the variables being compared resided in a bank different to that used for temporary variables, the bank of the temporary may not have been selected, resulting in the comparison failing. This problem is known to have affected the `printf()` library routine.

5.22. Version 1.32

Incorrect pointer assignment (XC8-1067) In some cases the code-generator was incorrectly assigning to a 3-byte pointer a 1-byte address that was a member of a structure and which was also accessed by a pointer.

Incorrectly initialized structures (XC8-1057) For PIC18 devices, when initializing structures that contained a `char` array with a string less than the size of the array, the initialization may have caused structure members following the array to be corrupted.

Code generator crash (XC8-1056) In rare circumstances, complex projects that made heavy use of function pointers may have caused the code generator to crash.

Bad short long addition result (XC8-1052, XC8-1064) For PIC18 devices, the result of `short long` additions when the operands were in different banks may have been incorrect.

Software UART writes not working (XC8-1048) The PIC18 peripheral library function `WriteUART()` was not sending the necessary start and stop bits and also truncated the data sent out to the TX pin.

Assembler crash (XC8-1046) In rare circumstances for baseline and mid-range targets, when procedural abstraction optimizations were enabled, the assembler may have crashed.

Assembly errors when using LOCAL macro labels (XC8-1043) In rare instances, use of the `LOCAL` directive in assembly macros may have resulted in a syntax or lexical error. This would affect macros

where the `LOCAL` directive was the first token in the macro definition.

Wrong bank selected with complement (XC8-1035) In some cases for PIC18 targets the compiler was selecting the wrong bank when performing a bitwise complement of a word-sized object in banked memory.

Incorrect memory access (XC8-1023) In extremely rare instances, code that accessed memory locations may have accessed them in the wrong bank.

Bad call to typeSub() error (XC8-1021, XC8-981, XC8-1027) In rare situations, complicated expressions which indirectly called functions via a pointer and where that pointer was a member of a structure that was passed to a function as an argument, a 'bad call to typeSub' error might have occurred.

Bad indirect structure member access (XC8-1006) Indirect access of pointer members within a structure may have failed if the structure pointer had been previously assigned a function return value. This only affected PIC18 devices.

Bad bank selected in addition (XC8-1005) For PIC18 targets, where an unsigned byte was added to a 2-byte location and the destination and two addition operands were in different banks, the operands might have been read from the wrong bank, which would have corrupted the result.

Parser crash after emitting errors (XC8-998, XC8-1037, XC8-982) Situations, such as where a function declaration was badly formed or there were other errors in the source code, might have caused the parser to crash after generating one or more errors indicating the offending line of code.

Can't Generate Code error (XC8-947) A Can't Generate Code message may have been issued for baseline parts when a single-byte pointer was indirectly assigned to a two-byte destination pointer.

Driver crash when using unorthodox memory reservation option (XC8-916) When reserving memory on an enhanced mid-range PIC, if the reservation range included the device's common memory the compiler's driver might have crashed.

Undefined symbol errors using interrupt_level pragma (XC8-871) When using the `interrupt_level` pragma with functions called indirectly, there may have been undefined symbols, such as `i0fpbase` or `fp__funcName`.

Bogus warnings (XC8-870, XC8-863, XC8-787) Warnings indicating a degenerate or mismatched comparison were sometimes issued in situations where this was not the case. This was most likely when the `sizeof` operator was used in expressions and an offset was subtracted from this operator's value. In other instances, an unused-variable warning was issued for temporary symbols generated by the compiler.

Cromwell crash (XC8-217) In rare situations, the cromwell application may have crashed for large projects.

Can't Generate Code error (XC8-176) In some instances, the compiler was not able to generate code for an initializer that implicitly converted an address to an integer type.

Code generator crash with multiple assignments (XC9-172) Large chains of assignments caused the code generator to crash (e.g. `a=b=c=d=...`). This has been corrected for the case where the intermediate destinations have no side-effects.

External memory missing for 18F97J94 (UDBC-740) No external memory was specified for this device in the relevant INI file. This prevented some compiler memory options from allowing you to add extra memory when building.

5.23. Version 1.31

Note about Debugging (MPLABX-2129) Note that an issue in MPLAB X IDE has resulted in poor debugging experiences when using MPLAB XC8 and source code that uses inlined functions. At times, more than one step action is required to advance one C source line, and sometimes stepping might have resulted in jumps to unexpected parts of the program. This has been corrected in version 2.05 of the IDE.

Bad address formation (XC8-1017) For enhanced mid-range devices, the formation of the upper byte of an address might have been incorrect when the target was in banks higher then bank 1. This might have affected increment/decrement of a pointer; loading a temporary variable from a software-stack-based pointer; or indirect assignment of a pointer.

Bad pointer assignment (XC8-1015) For enhanced mid-range devices, where a pointer is assigned to another pointer, the source pointer is byte-wide and on the software stack (reentrant mode), and the destination pointer is 2-bytes wide and is *not* on the software stack, then the upper byte of the destination pointer may be loaded with an incorrect value.

Inline delays causing crash (XC8-1009) In situations where the inline delay routine was used and no code followed in the same function that performed a call or jump to another page, the delay may have jumped to the wrong location and the program crash.

No warnings for stack overflow (XC8-1012) A warning indicating an overflow of the data stack was not produced for some reentrant functions which defined too much stack-based variables. A warning is now issued. Note that the MPLAB XC8 User's Guide incorrectly lists the PIC18 stack limit as 255 bytes. This limit is 127 bytes.

Undefined/redefined symbols for reentrant functions (XC8-1011) For functions compiled using the reentrant model, some assignments may have defined the wrong local symbol for a branch. Errors indicating undefined or multiply defined symbols, or relative branch/call offsets out of range may have been issued by the compiler.

Indirect access of structure members (XC8-1010) In some instances, pointer arithmetic on the address of structure members could be calculated incorrectly.

Incorrect pointer size with return statements (XC8-999) In rare instances, pointers returned by functions might have been made the wrong size.

Can't Generate Code errors with pointer structure members (XC8-451) Where a pointer is a member of a structure, it may have been made larger than necessary, or code that dereferenced it might have been less than optimal. In some situations, the compiler might have issued Can't Generate Code messages when dereferencing the pointer.

Program resets (XC8-970) Under some circumstances inlining nested functions calls might have resulted in a call to reset and program failure.

Debug information missing from .asm modules (XC8-1003) Line number information was missing in generated COF files for assembly files that used a `.asm` file extension. Files with a `.as` extension were not affected.

Bad labels for switch code (XC8-996) For PIC18 devices, if a `switch` statement was used in a reentrant function that was called by an interrupt function, the code generator created invalid label names for the cases.

Syntax error with stack-object assignments (XC8-997) Expressions involving multiple assignments of stack objects in reentrant functions on PIC18 devices, might have resulted in a syntax error from the assembler.

Incorrectly merged code (XC8-994) When a statement in the true part of an `if` statement and a statement in the false (else) part was the same except for a conversion (cast) of a symbol from signed to unsigned (or vice versa), the compiler might incorrectly consider these statements identical and merge the generated code.

Overlap of interrupt code (XC8-983, XC8-945) On an enhanced baseline devices with oscillator calibration enabled in the runtime startup code and an interrupt service routine defined, a linker warning (596) about segment overlap might have been issued. The warning would have indicated generated code that was corrupted and could result in code failure.

Overlap of interrupt code (XC8-979) Programs compiled for PIC18 targets that include a low- and high-priority interrupt function might have had the psects `intcode` and `intcodelo` overlap. This was especially the case when compiling in Free or Standard modes.

Byte comparison failure (XC8-978, XC8-990) Comparison of a byte expression with a constant might have failed when compiling for PIC18 devices. This would have only affected expressions whose results were stored in temporary variables. The temporary variable might have been incorrectly accessed in common memory when it was located in banked memory.

Bad pointer assignment (XC8-977) For PIC18 targets, when assigning from one pointer to another and the destination pointer was 2-bytes wide, the upper byte of the pointer might have been cleared rather than assigned the correct value.

Syntax error with left shifts (XC8-973, XC8-974) Some PIC18 expressions involving a left shift of a single bit quantity by a constant value might have produced a syntax error due to a malformed assembler instruction.

Bad parameter load (XC8-959) In rare instances, where a function call is required to obtain the argument to another function, there may be a corruption of the called-second function's parameters.

Cromwell crash (XC8-966) Programs that contain an inline-qualified function that was successfully inlined by the compiler and with no 'outlined 'version generated, might have caused cromwell to crash during ELF/DWARF generation.

Bad call to typeSub error with pointers (XC8-960) For code that defined function pointers and these pointers were not assigned a valid address of an object, the error 'Bad call to typeSub() 'might have been produced.

Assembler crash with large routines (XC8-955) For midrange PIC devices compiling extremely large routines that would never actually fit on the target device, the assembler might have crashed. A memory error is now reported in such circumstances.

Ignored __section specifier (XC8-562, XC8-951, XC8-950, XC8-852, XC8-851) In some instances, the use of the `__section` specifier was ignored, or might have resulted in the wrong section being used. This would have occurred when objects using custom sections were used across multiple source modules. Bogus warnings issued when using this specifier have also been suppressed. The previous limitations associated with use of the `__section` specifier have been lifted as a result of this fix. You no longer need use the specifier with declarations.

Driver crash when adjusting memory ranges (XC8-936) When complex memory reservation ranges were specified (particularly those that contained duplicate or overlapping ranges) or when hand-written assembly code defined absolute psects, the compiler driver may have crashed.

Code generator crash when using regused pragma (XC8-928) If the `regused` pragma was used with with no registers specified, the compiler might have crashed. This only affected compilation for baseline and

all midrange devices.

Driver crash when reserving common memory (XC8-1014) When reserving addresses for an enhanced midrange device and the addresses were within the common memory range, the compiler driver might have failed an assertion.

Looping around allocGlobals error (XC8-832, XC8-991, XC8-946) In some instances where a structure contained a pointer member, the error 'looping around allocGlobals' may have been issued.

Non-detection of multiple use of codeoffset (XC8-509) When compiling for PIC18 devices, using more than one `--CODEOFFSET` option triggered an error; when compiling for all other devices, it was silently ignored. Now, for all devices, the duplicate option is ignored provided it does try to move the origin of ROM lower than it is currently set; otherwise, a warning is produced.

Seemingly insane can't find space messages (XC8-502) Bogus can't find space errors might have been produced for structure or array objects that contained pointers, even though there was ample memory remaining. The message would have been triggered by objects whose size (assuming the pointers were 3-bytes wide) was larger than 256 bytes, but whose final size was less than 256 (after the pointers sizes were determined).

Missing warning (XC8-491) The compiler was not producing a warning when code applied the `sizeof` operator to objects of incomplete type e.g. an array with no declared size.

Error messages referencing inappropriate source code (XC8-490, XC8-207) Some error and warning messages are not related to specific lines of code; however, even in these circumstances the compiler would print a file name and line number (which would typically be the last file and line of source processed). Some messages now include the string "Non line specific message:" instead of a filename to ensure it is clear that there is no single statement that is at fault.

Bogus warning when comparing function types (XC8-464) The compiler might have issued warnings indicating a mismatch in type when comparing a function that has an empty parameter list and a function that has a `void` parameter (all else being equal). These are now considered equivalent types.

Parser crash (XC8-460) When erroneous code such as the following (attempted use of an incomplete structure definition) was encountered:

```
struct a {
int f;
/* missing close-curly } */;
struct a b;
```

the parser component crashed instead of terminating gracefully.

Memory errors when using sort function (XC8-449) The buffer size used by the `sort` function has been reduced for non-PIC18 devices to prevent can't find space messages on some devices. Consider adding the source for this function to your project if you want to customize the buffer size for your application and device.

Detection of square bracket misuse (XC8-443) The compiler will now flag an error if square, array-subscript brackets (as opposed to round brackets) have been mistaken used to call a function.

In-line assembly degrading debugging (XC8-329) Setting breakpoints on lines of code appearing after `#asm` ... `#endasm` blocks was often not possible, or breakpoints did not match the correct assembly instructions. Use of `asm()` statements did not cause this issue.

Can't Generate Code when calling functions via initialized pointers (XC8-238) The compiler better handles situations where pointers containing NULL were been used to call functions, or absolute (literal

constant value) functions were called. A warning might be triggered if such a call is made.

Spurious warnings concerning invalid variable locations (XC8-221) If the warning level threshold was lowered, spurious warnings about library functions were being printed. Messages appeared similar to "invalid variable location detected: ___aslmul - ___aslmul (warning)"

Assembly crash with in-line assembler (XC8-182) In some instances when in-line assembly contained a GOTO instruction, the assembler may have crashed. This affected only PIC18 targets.

Can't Generate Code messages with comparisons (XC8-164) A Can't Generate Code error might have been issued for complex relational (greater than) comparisons of `int` types.

Failure with indirect comparison (XC8-13) Code that indirectly accessed any 4-byte object and performed a == or != comparison might have failed. This might be expected when integer or floating point quantities were converted to boolean values.

Missing symbol when calling functions indirectly (XC8-968) When compiling for any baseline or midrange device, functions that are only called indirectly and only from an interrupt might not have been added to the indirect function calling table. This would have resulted in an undefined symbol error.

5.24. Version 1.30

Incorrectly merged code (XC8-924) When a statement in the true part of an `if` statement and a statement in the false (else) part was the same except for a conversion (cast) of a symbol, the compiler might incorrectly consider these statements identical and merge the generated code.

MPLAB X Watch objects in wrong memory space (XC8-935) In some instances, uninitalized (BSS) variables would be reported as being in program memory in the MPLAB X watch window, and the wrong value displayed.

Build delay The compiler's license manager (XCLM) was updated to revision 1.23. This disables the RLM option to automatically query the network for a network server when a node locked file was not found. This will prevent a several-second delay with each invocation of the compiler.

Recursive call to function error (XC8-913) In some instances the compiler falsely reported that a compiler library function was called recursively.

Reset to main in debugger (XC8-926, XC8-1008) The compiler produced a debugging image lacked sufficient information to support the MPLAB X feature that breaks the debugger at main() on restart. Note that changes to resolve these issues were also made in MPLAB X IDE and will be available in the v2.06 release.

Bad right shift (XC8-938) For all devices, some expressions involving a right shift of an unsigned object that was cast to be a signed object, might have produced an incorrect result.

Bad pointer access (XC8-908) Dereferencing a pointer with program and data space targets may have resulted in the wrong value. This issue is known to have caused bad output when using printf().

Copying const structures (XC8-806) Code that indirectly copied (via a pointer) a structure larger than 4 bytes in size to another structure may have failed if the source structure was in program memory and the source pointer had both RAM and program memory targets. This affected PIC18 devices only.

Duplicate advisory messages printed (XC8-840) Advisory messages might have been issued twice when compiling code in some situations.

Bad code after indirect data memory access (XC8-839) Code which indirectly accessed RAM using an

FSR may have overwritten the content of WREG, resulting in subsequent code failure. This would have only affected enhanced mid-range devices.

Truncation of Operand Value messages when accessing linear objects (XC8-837, XC8-889) For enhanced mid-range devices that accessed objects placed in linear memory, the wrong address may have been accessed. The warning "truncation of operand value" might have been issued for such code.

Compiler crash when copying structure pointers (XC8-779) When casting a pointer to a structure to another structure pointer, the compiler may have become stuck in a recursive loop (and ultimately crash) if the source and destination structure types both had a self-referential pointer member at the same member position in the structure.

String labels (STR_x) defined more than once (XC8-527) Some string labels may have been reused resulting in a multiply defined label error from the compiler. String labels look similar to `STR_2`, `STR_10` etc.

Can't Generate Code errors when casting structure pointers (XC8-722) This error may have been issued when casting pointers to packed structures to other pointer types.

Inappropriate pointer warning issued (XC8-599) The message "a pointer to eeprom cannot also point to other data types" may have been issued in some situations that were not appropriate.

Linker errors when adjusting program memory ranges (XC8-522) If compiling for an enhanced mid-range device and overriding the default program memory with an option similar to: `--ROM=0-7ff` (options in which the default memory is removed and new ranges are specified), the `STRING` linker class was not being allocated the specified memory. Linker errors such as 'psect "strings" not specified in -P option' might have resulted and string objects might have be linked at invalid addresses.

Undefined symbols using interrupt_level pragma (XC8-535, XC8-133, XC8-3) If a function was only called in interrupt code and the `interrupt_level` pragma was applied to it, it was not output at all, resulting in "undefined symbol" errors.

Bad Pointer conversion (XC8-761) Implicit conversion of a 2-byte pointer to a 3-byte pointer might have failed in some situations. The upper byte of the destination pointer was always being cleared.

Bad pointer comparisons (XC8-604) In some situations, comparisons of 3-byte pointers with `NULL` may have been incorrectly performed.

Errors using ## preprocessor operator (XC8-526) When expanding preprocessor macros whose expansion involved the concatenation operator, ##, and the argument to such a macro was a floating-point number which had a sign character and one and only one exponent digit (e.g. 1.23e+1), the expansion may have failed and generated compile-time errors.

Bad Pointer conversion (XC8-761) Implicit conversion of a 1-byte pointer to a 3-byte pointer might have failed in some situations.

Driver crash when reserving memory (XC8-688) The compiler may have crashed if reserving program memory that included all the configuration memory. It is now no longer possible to reserve any of the configuration, idloc, or useridloc memory using the `--ROM` option. These address ranges are masked from any range specified using this option.

Can't Generate Code errors and __section specifier (XC8-506) Indirectly accessing objects defined using the `__section()` specifier may have produced this error for any mid-range or baseline device.

Bogus Can't Find Space messages (XC8-85) In situations (typically when writing bootloaders) where the entire memory which is normally allocated to a linker class is reserved, the linker might have displayed

error messages indicating that psects could not be allocated memory, even if those psects had a size of zero. These errors are now suppressed in this situation.

Assembler crash with missing labels (XC8-123) The assembler may have crashed if optimising code that had jumps or calls to labels that were not defined in the assembly code.

Can't Generate Code errors (General) (XC8-505) The compiler now performs an extra code generation step before issuing these errors. This might suppress such errors for expression that involve (implicit or explicit) function calls and register allocation.

Looping Around allocGlobals error (XC8-844) Some expressions created temporary variables which were not correctly allocated memory. This caused an endless loop which resulted in a "looping around allocGlobals" error to be produced.

Corruption of const data (XC8-865) The psects that contained absolute `const` objects were being scanned by the assembler and errata NOPs added after any label in those psects. This corrupted the data these psects held. The names of the psects used to hold these data have changed from ending in `_text` to ending in `_const` and they are no longer scanned by the assembler. This issue only affected devices that required the FETCH or 4000 errata workarounds (see table at the end of this document).

General code failure (XC8-866) Some code sequences involving `char` and `int` types may fail, particularly when casting from one type to another. This only affected mid-range and baseline devices. While a general fault, it was unlikely to affect programs.

Can't Generate Code when indirectly calling functions (XC8-867) This error messages might have been issued when calling functions via a pointer which is initialized only with a literal constant or NULL. This issue only affected PIC18 devices.

Compilation of source files with the same name (XC8-768) An error was generated when compiling an MPLAB X IDE project that contained two or more C source files with the same filename (but stored in different directories). Such projects are now allowed to compile. Operation on the command-line is similarly less restricted, provided there will be no clash between the intermediate P1 files generated from these source files.

Complement operator used with boolean results (XC8-167) The complement of a boolean value, for example `~( ! foobar)`, was returning a boolean type rather than the integral promoted type. This may have produced incorrect values in expressions.

Error reading CMF (XC8-872) A user defined psect (either in assembly or created with the `__section()` directive) that was not associated with a linker class (either in a command-line option or in the psect directive) produced a CMF file that could not be read by the Cromwell application. It caused Cromwell to emit the message "error reading CMF: no token at index 6".

Arguments to indirect calls not correctly passed (XC8-242) If more than one function was called indirectly via the same pointer and these functions had pointer parameters, in some instances the parameters might not have be passed correctly.

Incomplete memory summary (XC8-503) Psects that are created using the `__section()` specifier did not appear in the memory summaries issued by the compiler. These are now included in an "unclassed" part of the psect listing.

Can't Generate Code errors with structure pointer members (XC8-417) In some situations, code involving pointers defined in structures produced Can't Generate Code messages. This was most prevalent when the pointer was passed to a function as an argument.

Bank selection issue with bitfield member (XC8-877) When accessing a bit-field located at position 6 in the structure, and the expression contains another memory object in another bank, a bank selection instruction may have been omitted resulting in the wrong bit-field location being accessed.

Incorrect XOR of single bit-field object (XC8-842) XORing single-bit-wide bit-field objects with the value 1 was always producing the value 1.

Incorrect memory reservation with user-defined RAM psects (XC8-466) The compiler was reserving memory during the C code generation step for psects in hand-written assembly code that were not absolute and overlaid. Only absolute and overlaid psects should memory reserved at the code generation stage, as described in the XC8 User's Guide. This may have resulted in bogus out-of-memory errors.

Code failure with shift and type conversion in the same expression (XC8-887) Expressions that involve a variable being shifted by multiples of 8 and a conversion to smaller integer type (whether implicit or a cast) may have accessed the variable in the wrong bank and led to code failure.

Truncation of Operand Value messages with compound assignments (XC8-873) Compound assignments of a constant to objects that are in linear memory, e.g. `a=b=100;` might have produced a warning "truncation of operand value" message with subsequent code failure. This only affected enhanced mid-range devices.

Incorrect delays and Truncation of Operand Value messages (XC8-843) When using the inbuilt delay routine `_delay()` and delay values above 1792, the delay time may have been in error and warning messages indicating truncation of operand value may have been issued.

Bad Fill assignment (XC8-890) In some instances when the `--CHECKSUM` option was being used, a `--FILL` option was implicitly created by the driver to ensure consistent checksum results. The fill command passed to Hexmate was not correctly formed and resulted in a Hexmate error: (941) bad "-FILL" assignment.

Parser crash on division/modulus by zero (XC8-886, XC8-888) Some preprocessor expressions involving a division or modulus by a literal value of zero were not being detected and crashed the parser. The parser now issues an error message for such situations.

Illegal sized arrays accepted (XC8-898) Array declarations that specified an array size and the size expression involved the sizeof() operator were not flagged as an error if the size expression evaluated to be a negative value. An error is now issued for such code.

Missing warning or error messages (XC8-899) Some warnings issued by the compiler may have been lost and not appear during a build. It is possible that builds with this compiler release will produce more warnings than with previous builds.

Address of objects returning NULL (XC8-76) For programs with less than 256 bytes of `const` data (const objects being placed in the `smallconst` psect), taking the address of a `const` object may have compared equal to `NULL`. The first byte of the space allocated to the `smallconst` psect is now reserved so that this situation can no longer occur.

Error when using booleans in integer expressions (XC8-131) The unary + and - operators may now be used with either `bit` variables or operators that produce a boolean value (such as ==, `!` and >). The boolean values are promoted to an `int` then used in the usual way.

Negative signs with negative floating-point values (XC8-135) In some instances, the negative sign was not being printed for negative floating-point values. Note that the printf code is customized with each build based on the placeholder your program uses, so not all programs would have been affect by this issue.

5.25. Version 1.21

Incorrect indirect access of absolute objects (XC8-847) Pointers assigned the address of absolute objects might have been too small to correctly access the target object. This only affected PIC18 devices.

Wrong pointer sizes (XC8-824, 791, 830) In some situations pointers may have assumed incorrect sizes which lead to code failure.

Bad code after addition (XC8-823) Code which required the addition of an stack-object address with a small literal constant may have clobbered the value in held in WREG. Any subsequent code relying on WREG may have failed. This only affected enhanced mid-range PIC devices.

Over-enthusiastic optimization of goto to branch instructions (XC8-808) Regardless of the state of the optimization option, the PIC18 assembler would always consider changing GOTO instructions to branch instructions for hand-written assembly modules (this did not affect in-line assembly code).

Assembler crash with GOTO instruction (XC8-805) In some instances, the use of a constant literal address as the operand of a GOTO instruction caused the PIC18 assembly optimizer to crash. This is only likely to have affected hand-written assembly code.

Syntax errors with bit expressions (XC8-810) Assigning the result of an AND between two bit types to a bit type destination, may have resulted in syntax errors. This only affected PIC18 targets.

Compiler crash with assembly-only projects (XC8-822) For projects that contain only hand-written assembly code, the compiler driver may have crashed. This only affected compilers running under the Window OS.

Spurious message when building (XC8-783) Programs containing functions with identifiers ending in `alt` or `nosup` could have caused the compiler driver to emit messages making reference to `process_nosup_syms`. The messages relate to a "mistaken identity" of symbols used by the linker to convey specific linking information; they do not indicate wrongly generated code.

Assembly optimizer alters code sequences affecting volatile SFRs (XC8-798) The assembly optimizer can partially abstract code sequences. Typically, this is of no concern, but for time-sensitive instruction sequences (e.g. special instruction sequences to initiate a flash write), this may cause unexpected behavior. The compiler has been updated to ensure that access of all volatile special function registers are not abstracted. This only affects mid-range PIC devices.

Assembler optimization produces bad code for if() statements (XC8-825) When an if() expressions involved two tests for equality and the body of the if() was an assembly sequence that was only 1 instruction long, an assembly optimization may have corrupted this code sequence causing code failure. This only affected PIC18 devices where assembler optimizations were enabled.

Wrong bank access with shift code (XC8-828) Code involving right shifts of long objects, where the object being shifted and the variable containing the shift amount are in different banks, was not correctly registering the bank selected. Subsequent code may have accessed the wrong memory location. This only affected PIC18 target devices.

Crash when using large command lines (XC8-845) Executing the compiler with command line arguments totaling more than 8192 characters in length may have caused the compiler driver to crash.

Wrong bank access in code loops (XC8-567) For some code sequences that contained loops, tracking the currently selected bank when it was changed inside the loop may have failed. This would have caused wrong variable locations to be accessed on subsequent iterations of the loop.

Wrong pad values used with download option (XC8-760, 762) The `download` suboption to the `--RUNTIME` option pads certain records in the HEX file. The pad values used for mid-range and baseline

devices were specified using the wrong byte order. This may have led to programming errors. This issue did not affect the operation of the code in any way.

Crash when using #fi directive (XC8-771) The `#fi` preprocessor directive is valid when using C18 compatibility mode. Outside of this mode, use of this directive should flag an error. Instead, the preprocessor was crashing when this directive was encountered.

Incomplete initialization of absolute const arrays (XC8-770) When compiling for PIC18 devices, the uninitialized elements or members of partially initialized absolute `const` array or aggregate objects, may not have been zeroed. This issue did not affect objects that were not absolute.

Incorrect byte return value from functions (XC8-769) For functions that returned a byte and which did not use the regular call stack (this implies that the `stackcall` suboption to `--RUNTIME` was enabled), expressions that used this return value may have read it from the wrong location. This would have resulted in incorrect results from expression that used the return value.

Recursive function call error (XC8-437) Code using the `modf()` library function may have triggered the error message *recursive function call to "___ftpack"*. The source code for this function has been modified so that this message is no longer generated.

Function return value corrupts other data or is corrupted by other code (XC8-344) Insufficient memory was allocated for a function's return value if that value was greater than the size of the auto-parameter block for that function. Specifically this issue only affected PIC18 functions which returned a pointer type and when the size of this pointer was larger than 1 byte in size. This return value may have corrupted other memory locations or have been corrupted itself by other functions or interrupt routines.

Incorrect pointer size calculated for duplicated function's return value (XC8-461) If a function returning a pointer was duplicated because it was called from more than one call graph, there were instances where the size of the pointer returned by the function were incorrect.

Inlined code crashes (XC8-804) If code that was inlined (specified `inline`) contained more than one jump or more than one call to the same destination label, then only the first jump or call instruction was fixedup to use the new inlined label. If the execution path in the inlined code took any of the latter jump or call paths, this would have caused execution to crash. This issue has been corrected and inlined functions may contain multiple jumps or calls to the same destination label.

Failure to load ELF file in MPLAB X IDE (XC8-784) If `inline` functions where used in a program, the ELF files produced may not have loaded into the Windows version of MPLAB X IDE.

Can't find space for zero bytes message (XC8-800) The use of some `bit` SFRs on 16F1xxx enhanced PIC devices (for example `SEG43COM3` on the 16F1947) may have resulted in a compiler error indicating that it couldn't find space for zero bytes.

Missing errata workaround NOP instructions (XC8-803) The assembler failed to insert a NOP instruction at the entry point of an assembly function defined for PIC18 devices that suffers from the FETCH or 4000 errata issues.

5.26. Version 1.20

Bad conditional code (XC8-640) An error in an assembler optimization that dealt with a bit-test-and-skip instructions and FSR manipulations caused the sense of some conditional control statements to be inverted.

Phase errors (XC8-590, XC8-691, XC8-49) The assembler optimizer was removing redundant page selection instructions, but other parts of the assembler where not taking this into account. This created a

mismatch in expected and actual sizes of some jump instructions which led to phase errors being produced. This issue affected hand-written and compiler-generated assembly code.

Absolute function placement (XC8-203) The address specified for functions made absolute was rounded down to the nearest 0x10 value. So, for example, if you attempted to place a function at address 0x56, it was actually located at address 0x50. This only affected programs compiled for PIC18 devices.

Missing errata NOP instructions (XC8-692) The assembler was not correctly adding in NOP instructions that form part of the `fetch` errata workarounds for some PIC18 devices. (See the user's guide `--ERRATA` option section for more information.) Delay routines that required these instructions were known to run too fast. A consequence of this fix is that code size will increase for devices that require the `fetch` errata workarounds.

Wrong UINT24_MAX value (XC8-565) The value for this `<stdint.h>` macro was higher by 1 than it should have been.

Bad variable access after calls (XC8-694) The compiler was, on occasion, placing a bank selection instruction before the call to a routine. This bank selection was not properly being tracked and may have resulted in incorrect access of objects once in the called function.

Assertion failure using --ROM or --RAM options (XC8-517) When attempting to reserving memory that was outside your target device's on-chip memory, these options may have caused an assertion failure.

Invalid access via pointer (XC8-643) In some situations, the compiler was not detecting that the upper TBLPTR register was being changed from its assumed state. As a result, code that dereferenced a pointer with both RAM and ROM targets may have corrupted this register and triggered a subsequent failure in the code.

Assembler crash with bad options (XC8-510) If options were passed directly to the assembler application but no file names were present, the assembler issued an error and continued processing. The error produced in this situation is now a fatal error to prevent the crash. This issue would have only affected users driving the assembler directly.

Assembler crash when specifying functions as inline (XC8-589) In some instances, when a function was declared `inline` the assembler crashed.

Can't Generate Code message associated with unused objects (XC8-636) In cases where a pointer was not used but was assigned the address of an object, this message may have been emitted. Optimizations associated with such code are now restricted and the message will not be issued.

Parser crash with enumerated types (XC8-637) The parser may have crashed when scanning code associated with enumerated types. This may have been associated with taking the address of enumerated objects.

Qualifiers silently ignored for local objects (XC8-670) Objects which are local to a function (including `auto` and static local objects) cannot be qualified as `far` or `near`. The compiler was not indicating that the specifier was being ignored. A warning is now issued for such definitions.

Absolute addresses silently ignored for local objects (XC8-652) Objects which are local to a function (including `auto` and static local objects) cannot be made absolute. The compiler was not indicating that the `@` or `__at()` construct was being ignored. A warning is now issued for such definitions.

Linear memory allocation of objects (XC8-501) Absolute objects that specified a linear memory address may not have been allocated correctly if the object's equivalent banked addresses mapped into the common memory. This only affected enhanced mid-range devices and objects that were not large enough to be automatically allocated to linear memory. The compiler now correctly allocates these

objects.

Identical case label values in a switch statement (XC8-493) The compiler was not detecting the use of more than one identical `case` label value inside a `switch()` statement. This is now correctly identified and will trigger an error.

***__DEVICENAME__* not defined (XC8-659)** This predefined macro was not being defined by the compiler. This is now defined unconditionally.

Procedural abstraction of in-line assembly code (XC8-660) The assembler optimizer was performing procedural abstraction on assembly code that was placed in-line with C code. The optimizer should not perform *any* optimization of in-line assembly code. The optimizer has been prevented from performing these optimizations.

BSR register not recognized by regsused pragma (XC8-663) Any attempt to list the BSR register in the `regsused` pragma would have resulted in an error. This has been corrected and this register may now be used with this pragma for those devices that implement this register.

Allocation of PIC18 far variables (XC8-582) In some instances, `far`-qualified variables may have been linked at address 0, not in the memory defined as the far RAM.

Indexing array with constant expression failure (XC8-648) With enhanced mid-range devices only, any operation that involved adding a integer constant to an address (typically this will be an array access with a constant integer index) may have caused subsequent code to fail. Such code was not correctly reporting its use of WREG.

Initialization of large objects on enhanced mid-range device (XC8-672) Initialization of large objects (such as arrays or structures) that contained a pointer may not have been assigned the appropriate values by the runtime startup code. This only affected PIC16F1xxx devices.

Uninitialized const objects not assigned zero (XC8-553) If `const` objects were not initialized, they were not being automatically assigned the value 0 by the compiler. These objects had no memory reserved for them at all and this may have resulted in them appearing to overlap with other `const` objects. This is now corrected; any `const` object that does not have an initial value is implicitly assigned 0 by the compiler.

Assertion failure reserving RAM (XC8-662) In some instances, reserving RAM when using any non-enhanced mid-range part using the `--RAM` option would result in an assertion failure `hi >= lo`.

Integers allowed with specifiers (XC8-549) The integer constants usable with specifiers such as `__at()` were limited to 16-bit values. These values can now be specified as 32-bit values.

Multiply defined symbols (XC8-516) If code used the `__section()` specifier with variables, there may have been symbols contained in the runtime startup code that were defined more than once, producing an error. (Such symbols might be `clear_ram` or `clrloop`, for example.) This duplication has been corrected.

__mediumconst symbol undefined (XC8-621) In some circumstances, particularly when using the peripheral library, the compiler may have produced an undefined symbol error for `__mediumconst`.

Functions not inlined (XC8-521) When a function qualified as `inline` was called from `main()`, it was never inlined and a regular call was made. Inlining should now take placed for any suitably qualified function called from `main()`.

Incorrect return instruction (XC8-525) For some baseline devices (most notably the 12F529T39A and 12F529T48A), the compiler may have attempted to use the non-existent `RETURN` instruction instead of

a `RETLW` instruction.

Partial access to ID locations (UDBC-678) The compiler did not allow access to the entire ID location for devices that implement this memory as being 14-bits wide (e.g., the PIC16F1503). This has been corrected and you may now program all the bits of these ID locations.

Code failure accessing absolute objects straddling a bank (XC8-601) The wrong bank may have been selected when accessing absolute objects that straddle a bank boundary. Code may have failed when accessing the addresses in banks following that of the bank of the object's base address.

5.27. Version 1.12

Compile times and crash (XC8-127) A sorting issue related to pointer variables may have significantly increased the compilation time of projects. Not all projects were affected by this issue. This issue may also have caused the code generator to run out of memory, or even crash in some situations.

Compile times (XC8-498) A further issue affecting compilation times was corrected. This issue affected the assembler when `--ASMLIST` was used. (This option is on by default when using the IDE.)

Installer operation The installer program was not correctly setting the write permissions for some files. This was reported on Windows XP, but may have affected other platforms.

Bank selection issue (XC8-494) In an expression such as:

```
A = B + C;
```

where `A` and `C` are `unsigned` 16-bit objects in different banks and `B` is an `unsigned char`, a bank select instruction may have been omitted resulting is the wrong value being assigned to the destination variable.

5.28. Version 1.11

Compilation times (XC8-441) Large projects, particularly those targeting PIC18 devices, may have experienced increased compilation times. This was due to the compiler processing more symbols. A new option has been added, and enabled by default, to limit the symbol list. The new option is `--PARSER`. See the New Features section for more information.

Looping around pointGraphComplete() Error (XC8-442) A non-deterministic pointer-related issue was causing this error to be printed, and terminating compilation.

Debugging absolute objects (XC8-447) Some SFRs and absolute addressed objects residing in RAM would appear as being located in program memory during a MPLAB X or v8 debugging session. The content of these values would, thus, be incorrect. Absolute symbols should now appear to be located in their correct memory space and their contents shown correctly.

Parsing of config pragma arguments (XC8-452) The parsing of arguments to the `#pragma config` directive was erratic when it came to quoted arguments. The quote character ' `"` ' is now a token delimiter and you may quote the arguments to this pragma, e.g. `"WDTEN=ON"` Doing so will avoid any macro substitutions by the preprocessor. If you have defined macros for `ON`, `OFF` or any other token used by the `config` pragma, consider quoting the pragma arguments or moving the pragmas to a module that is not exposed to your macros.

Incorrect access of array of strings (XC8-454) Code which used a variable to access an element of a string array may have failed when the index was non-zero.

Unsupported short long message using CCI (XC8-429) When using the CCI, a warning may have been issued indicating that the `short long` type was not supported. The header files that referenced this

type have been updated and use a plain long type when compiling for the CCI.

Undefined symbols with bitwise operations (XC8-424) Some bitwise operations, for example `|` or `&`, when used in functions that were in the interrupt call graph, may have produced code that contained references to undefined temporary symbols. Such symbols would look similar to `i2u49_41`.

Can't Generate Code with printing floats (XC8-108) For some placeholders associated with `float` types, a Can't Generate Code error may have been triggered with (`s`)`printf`. This has been corrected.

Detection of incomplete types (XC8-109) The parser was not detecting definitions using incomplete types, for example:

```
typedef struct foo foo_t;  // where foo has not been defined
foo_t x;
```

Code which is defined in such a way will now trigger an error from the parser.

Can't Generate Code for library string routines (XC8-413) Some string library functions may have caused "Can't Generate Code errors". These have been adjusted to ensure correct compilation.

Assignment to volatile bytes (XC8-427) In some operating mode, when assigning '1' to a `volatile` byte variable, it may not have been updated atomically (with one write rather than clear and increment instruction). This problem did not affect absolute objects and may have only caused runtime problems if the byte was accessed from main code and interrupt.

--ROM option The `--ROM` option was not processing its arguments correctly which may have resulted in it not reserving the memory specified.

5.29. Version 1.10

Using far with Functions (XC8-337) If the `far` qualifier was used with functions, it confused the compiler into thinking the function identifier was that of a variable destined for RAM. This may have triggered an error, if no far memory was defined, or caused a runtime code failure.

Non-functional --ADDRQUAL option (XC8-293) The `reject` suboption of this option was not working correctly for PIC10/12/16 device. The `require` and `request` suboptions were not working correctly for PIC18 devices. This option should now work as expected for all devices.

Conversion of integer to bit-field (XC8-353) The compiler was treating single-bit bit-fields as a boolean rather than an integer quantity when it came to assignment. Code that assigned a byte or larger integer to a single-bit bit-field may have failed.

Can't Generate Code for Duplicated Functions (XC8-358) The compiler may have issued Can't Generate Code error messages in situations where a function was duplicated in interrupt and main-line code and this function used pointers.

Handling of incomplete types (XC8-374) The parser was producing incorrect type information where a type used in a declaration was, at that point, incomplete, e.g. if you used a structure tag in a declaration, but that tag had not been defined. The code generator would subsequently crash if such incomplete type information was encountered. types should always be defined before they are used, but the compiler will no longer crash on such situations.

Testing of volatile bytes (XC8-388) The code that tested `volatile` bytes variables for (in)equality was using a `MOVF x,f` instruction. This was not in keeping with the spirit of the `volatile` keyword and could also play havoc with certain registers, such as the TMR0 register, on some baseline devices, such as the 10F222, which require a read delay after being written. This code sequence is no longer used and

access of SFRs will be well behaved.

Pointer assignment failure (XC8-342, XC8-343) In some circumstances, assignment of a structure member address to a pointer may have resulted in an incorrect destination pointer size being determined by the compiler. This would have resulted in subsequent pointer dereferences being invalid. This problem was detected only with assignments to pointer parameters as part of a function call, but could have potentially occurred elsewhere.

Pointer access failure with duplicated functions (XC8-377) If a function that has a pointer parameter is called from main-line and interrupt code (i.e. it is duplicated by the code generator), the compiler may have issued a "looping around allocGlobals", or "Can't Generate Code" error message. In other situations, the code may have compiled, but accesses to the pointer may have been incorrect. The issue could have affected any device with interrupts.

Undefined btemp symbol (XC8-371) In some circumstances, exacerbated by the use of a debugger and devices with small amounts of RAM, the internal compiler symbol `btemp` may not have been defined and an error results. This symbol is now correctly defined.

Incorrect access of high program memory (XC8-363) The compiler was not correctly setting the TBLPTRU register for accesses of absolute-addressed `const` data located above 0xFFFF in PIC18 program memory.

Incorrect configuration bit/user ID settings (XC8-385) Attempts to program the configuration bit or user ID settings may have failed due to a sorting bug that may have resulted in the bits being programmed in the wrong order. The issue was inconsistent, but could affect all devices.

Improved error relating to IDLOC (XC8-384) If non-hex digits were used in for any nibble in the `IDLOC()` macro a confusing error was issued. A new message (#1436) has been created for this situation.

5.30. Version 1.01

Looping around allocGlobals error (XC8-318) This error may have been triggered, but is not specific to any particular code sequence. The compiler has been updated to ensure this trigger will not result in this error.

Access of char arrays (XC8-304) If the index expression used with an array of `char` consists of a non-constant expression (e.g. a plain variable) from which is subtracted a constant (e.g. `myArray[idx-1]`), then the index calculation may have been incorrect and the wrong element accessed. This issue mostly affected PIC18 devices and arrays no larger than a bank. It may have affected enhanced mid-range parts with array sizes larger than 256 bytes. The issue has been resolved for all devices.

Code jumps to wrong location (XC8-295) If the assembler optimizer is enabled, assembly code that directly wrote to the program counter may have been abstracted which may have caused a jump to the wrong location and code failure. C code affected would produce a lookup table of some description, but it unlikely that C code would trigger this issue. Hand-written assembly would only be affected if the option to optimize assembly source files was enabled.

Duplicated SFRs (XC8-319) Some SFR bit-field structures had duplicate entries in the device-specific header files and have been removed.

Bit objects on baseline devices Uninitialized global bit objects may not have been zeroed for baseline devices, and in some instances, the generated startup code may have corrupted other memory locations or caused the device to restart.

Call graph inconsistencies The call depth of some functions was not correctly indicated in the call graph shown in the assembly list file. The total amount of auto, parameter and temporary variable usage was also incorrectly displayed for some functions in the call graph.

Wide bit-fields in header files (XC8-289) Some header files were being generated with SFR bit-fields that were wider than a byte. This is not allowed by the compiler. Such definitions are no longer contained in the header files.

Wrong external memory access (XC8-286) The `--EMI` options was not correctly being processed and this may have meant that access to external memory may have been incorrect. Only several PIC18 devices have such memory.

Error on assembly directives after including header file (XC8-285) After including the `xc8.inc` header file into assembly code, some assembly directives, e.g. the `DS` directive, may have generated an error. This was due to SFR names conflicting with the directive's name. The header files will on longer use any SFR name that conflicts with a directive. This will mean that SFR names may not always match those listed in the device data sheet.

Bad access at start of function (XC8-94) The code at the beginning of function could have accessed the wrong address (bank) for a variable if the compiled stack was built up over multiple banks and WREG was used to pass a function argument.

Incorrect RAM ranges with 12F1501 (XC8-274) The RAM ranges associated with this device did not reserve the memory from 50-6F. Allocation of user-defined objects to this memory may have caused code failure.

Bad results with 24-bit expressions (XC8-227) The code generated for some arithmetic and bitwise operations involving 24-bit integers were missing some necessary bank selections which may have generated wrong results.

Compiler crash with conditional operator (XC8-117) Assigning a pointer the result of an expression using nested conditional statements could have caused the compiler to crash for PIC18 devices.

Bad results with long arithmetic (XC8-241) The results of some long arithmetic expressions may not be correctly assigned due to a bank selection bug.

Build errors in MPLAB X IDE (XC8-101, XC8-104) Source file paths that contained spaces were being written to the dependency files without the spaces escaped. This resulted in these files containing erroneous targets and dependancies which may have resulted in incorrect builds. In addition, the preprocessor would generating a dependency file for assembly modules incorrectly assuming that the intermediate file to have a ".p1" extension and not ".obj" resulting in the same behavior.

Crash with compiler-domain symbols (XC8-18) User-define variables and functions should never start with an underscore character as such symbols are in the compiler's domain. Inappropriate use of these symbols in a program, however, was leading to a compiler crash. The crash no longer will occur; however, you should continue to avoid using symbols beginning with an underscore.

MPLAB X IDE plugin Previously the XC8 MPLAB IDE plugin overwrote the HI-TECH Universal Plugin such that the universal plugin no longer appeared in the toolsuite list. This is no longer the case. The XC8 v1.01 installer will install and reinstate both the XC8 and universal plugins so that they will both be selectable from the IDE.

Bad if() code (XC8-58) In some instances the assembler optimizer would incorrectly move `MOVLW` instructions which could cause some logical expressions (such as those used by if statements) to be incorrectly evaluated.

Not honoring message disable (XC8-62) When compiling for PIC10/12/16 targets the assembler was not honoring the `--MSGDISABLE` option. Thus, it was impossible to disable warning messages produced by this application. This issue did not affect other applications or any application when compiling for PIC18 targets.

Bad call to subtype() Error (XC8-73) In some instances where a function has a pointer type as its first parameter but the function is never called, a "bad call to typeSub()" error may occur.

Incorrect optimizations involving carry bit (XC8-77) The assembly optimizer may have incorrectly moved instructions that set the carry bit above other code that would subsequently clear carry before it was being used.

Bad optimization of indirect access (XC8-95) In some situations, expressions with the form `*ptr = *ptr op A;` may fail. In particular it is the optimization of assignment-operations when the destination is an indirect access which can lead to incorrect results.

Bad right shift code (XC8-105) In some instances for PIC18 targets, the code generated for right-shift a `signed long` operand would access the wrong file registers, giving an incorrect result.

Memory reservation using --CODEOFFSET (XC8-230) This option should have reserved memory in all linker classes associated with program memory. It was only reserving this memory from the CODE class but not for other classes that hold const objects. This was unlikely to cause issues since const objects are typically not allocated to low address; however, all classes are now adjusted.

Badly optimized if() code (XC8-75) The assembler optimizer may have procedurally abstracted code that erroneously contained return-style instructions. This may have caused code such as `if()` statements to be ignored and fail. This has now been corrected.

Redundant MOVLP instructions (XC8-49) Redundant `MOVLP` instructions were being produced for some call sequences for Enhanced mid-range devices.

5.31. Version 1.00

Bogus warning on -B option (XC8-11) If compiling under MPLAB v8, in some instances a warning indicating that the `-B` option was defunct was issued. This option is indeed defunct, but the compiler has been adjusted so that this is not produced when compiling under the IDE.

Parser crash (PICC18-618) If an unexpected attribute was used with function definitions, e.g. `__attribute__ ((const))`, the parser crashed instead of giving an error. This has now been corrected.

Parser crash (PICC-684, PICC-688, PICC18-596, PICC18-607, PICC18-616, PICC18-619, PICC18-620, PICC18-621) In some circumstances, the parser would crash when encountering illegal or unusual source code. This has been seen with code that accesses some structure members, passing malformed arguments to functions, or with non-prototyped (K&R) function definitions. Changes to the parser should prevent the application crashing.

Bad code associated with negation (PICC-652) In certain cases, negating a 16-bit integer may have overwritten WREG. This has been corrected.

Crash on structure with no members (PICC-597) If a structure was defined that had no members, the parser application crashed. This has been rectified.

Arrays of structures in MPLAB not watchable (PICC-544, PICC18-593) In some circumstances, the watch window in MPLAB IDE v8 would show elements of a structure array correctly. Changes have been made to the compiler output to correct this.

Redirecting function using psect pragma (PICC-514) There were instances where use of the `#pragma psect` would not appear to work as the name of the psect being redirected was changing as a result of psect merging by the assembler optimizer. A new system of identifying mergable and splittable psects was introduced, which will assist in reducing this problem.

MPLAB IDE popup about source code using #include (PICC18-565, PICC18-566) If you are using any of the printf family of functions, an MPLAB IDE popup may have appeared warning that "The project contains source files that use the #include directive from within a function to include source code". Although this would not have affected normal debugging operations in this particular instance, the trigger for message has been adjusted so that this message will no longer be produced.

Zeroing of config bits (PICC18-600) If you used the PIC18 pragma to program only the lower byte of a configuration location, the compiler may have zeroed the upper byte rather than use the default configuration value. Ideally, all configuration words should be specified in your programs or the `--RUNTIME` option to program the device with the default config words should be enabled. However, this zeroing of he upper byte will no longer occur.

Undefined symbol with empty loops (PICC18-524) The compiler an remove some condition code if the state of variables used in the controlling expressions are know. In some instances, where the true or false statements were an empty loop (e.g. `while`), the compiler may have deleted a label that was still being referenced by other code. This would have produced an undefined symbol, which might look something like `l9`.

Crash with undefined functions (PICC18-532) If function that were not defined were assigned as the target of a function pointer, the compiler may have crashed. This crash has been fixed and now the linker will report the symbols as being undefined, as expected.

Errors indicated in unused files (PICC-428) When an error indicating a variable is too large is emitted, the name of the file that contained the variable definition may have indicated a source file that was not in the project, such as a library file.

Assignment to structure ignored (PICC-433) In situations where an assignment is made to a structure is followed by an assignment to a bitfield within that structure, the initial assignment may be removed. This would only occur in PRO mode. This has now been fixed and the compiler notes the distinction between the objects accessed.

Compiler crash and old library modules (PICC-573) The compiler may have crashed is some library modules were used from the SOURCES directory of the compiler. This directory no longer include files from non-OCG compilers, as they are not compatible to OCG compilers and will not compile.

Undefined ?Fake symbol (PICC-589, PICC-581) This error may have occurred for code that indirectly called a external function defined in assembly code or was part of another build. Only functions that returned a value in memory would be affected by this issue. The error is no longer produce, but you are required to define a symbol that represents the memory location where the return value will be stored. See the User's Guide on External Functions for more information.

Linker crash with assembler optimizers (PICC-648) If the assembler optimizers (psect merging) were enabled, in some situations the linker would crash if a psect was removed but was still referenced by a directive. This has been corrected and all assembly optimizations may be employed.

Compiler crash with bad memory ranges (PICC-608) If a memory range was specified using the `--RAM` option, or its equivalent in MPLAB IDE, and that range was excessively large, the compiler my have produced an assertion failure. In such situations, the compiler now prints a more informative error.

Cromwell error with MCPxxxx targets (PICC-642) If compiling for an MCP device, an error relating to the "prefix list" would have been triggered. This has been corrected and compilation can be performed for these devices.

Cromwell crash (PICC-493) Cromwell may have crashed if encountering information generated from static bit objects that are absolute.

Crash with malformed warning pragma (PICC-610) If the message number(s) were omitted from the `#pragma warning disable` directive, the compiler crashed. An error is now emitted if this situation is encountered.

Read of write-only registers (PICC-658) If an attempt was made to read a Baseline device write-only register, e.g. `TRIS`, the compiler would produce a Can't Generate Code error message. A more specific message is now produced alerting you to the offending operation.

Can't Generate Code (PICC-683, PICC-499) The compiler was not able to generate code for some expressions involving logical or bitwise operation on `bit` objects. This has been corrected.

Prototype for eeprom_write (PICC-675) The prototype for `eeprom_write` incorrectly stated that this function returned a value. This was not correct and the prototype now indicates a return type of `void`.

CLRWDT appearing in _delay output (PICC-460) For some delay values, the `_delay` in-line function for PIC18 targets only produced `CLRWDT` instructions as part of the delay. This has been corrected and a `NOP` is used instead. For PIC18 targets, you have a choice of `_delay` or `_delaywdt` to implement in-line delays, which do not use and use, respectively, the `CLRWDT` instruction as part of the delay.

Optimization of volatile bits (PICC-606) The compiler was not generating the appropriate information to prevent the assembler optimizer from optimizing access to volatile bit objects. Although the code would have been functionally correct, subsequent optimizations may have caused runtime failures. This correct information is now generated and volatile bit access will not be optimized unexpectedly.

vprintf and vsprintf linker errors (PICC-524, PICC-619) Trying to use either of these functions may have resulted in the undefined symbols `__doprint`, `_vprintf` or `_vsprintf`. These routines are now fully supported in the libraries and can be called.

Upper bound of mktime (PICC18-126) The `mktime` function had a upper limit of the year 2020; higher values would return incorrect results. This limit has now been extended to the year 2038.

Bad call to subtype with regsused pragma (PICC18-392) If using the `regsused` pragma for a function before the definition of that function, this error occurred. This has been corrected and the pragma can be used at any point in the source code.

Looping around allocGlobals error (PICC-570) This error may have occurred in some situations, particularly when `NULL` pointers were being assigned in the source code. This has been corrected.

Read of wrong array element (PICC-542) When reading a const array using a constant index, in some situations, the wrong element may be read. This bug did not affect array access using a variable or other expression as the index.

Memory allocation An improved memory allocation scheme is used when variables are qualified as being in a particular bank or near and the `--ADDRQUAL` option is set to require. This will reduce the likelihood of Can't find space errors from the linker.

Code generator crash with undefined functions (PICC18-532) If a function pointer was assigned the address of a function that is defined in external code, the code generator may have crashed. External functions might include those defined in assembly source code. This only affect PIC18 targets.

Can't Generate Code errors with nested structures (PICC-628) This error may have occurred for code that accessed bit-fields within structures that themselves were members of other structures.

No stack allocated to function error (PICC-611, PICC-485, PICC-416, PICC-637) This error is now unlikely to be triggered by code. It was most likely to occur with complex expressions, particularly involving floating-point arithmetic.

Can't Generate Code errors with short long types (PICC-632) Some assignment operations on short long types, e.g. `|=`, `&=` and `*=` etc may have triggered a "Can't Generate Code" error when compiling for baseline or mid-range parts. The ability to generate code for such expressions has been enhanced.

Parser crash with bad identifiers (PICC-622) The parser (`p1`) may have crashed if certain keywords were used as a variable identifier, e.g. if you were to try to define a variable with the name `eeprom`. A crash may also have occurred if reserved keywords were used in place of variables in expressions. The crash has been corrected, but variable identifiers cannot be reserved keywords.

Symbol defined more than once for in-line assembly (PICC-563) If the `REPT` macro was specified in in-line assembly, the code generator was generating additional labels multiple times. This triggered an error indicating the symbol was defined more than once. This has been corrected and `REPT` can safely be used in in-line assembly code.

RAM ranges for 16(L)F1507 The RAM ranges available when selecting this device extended further than the physical device implementation. This has been corrected.

6. Known Issues

The following are limitations in the compiler's operation. These may be general coding restrictions, or deviations from information contained in the user's manual. The bracketed label(s) in the title are that issue's identification in the tracking database. This may be of benefit if you need to contact support. Those items which do not have labels are limitations that describe *modi operandi* and which are likely to remain in effect permanently.

6.1. MPLAB X IDE Integration

MPLAB X IDE update If you encounter the MPLAB X IDE error `The program file could not be loaded` for projects that use the compiler's (default) ELF output, please update your IDE to at least version 2.30. There was an issue in the ELF loader that triggered this error and prevented projects from being debugged, but which has been resolved. If you cannot update the IDE, switch your project settings to COFF output.

MPLAB IDE integration If Compiler is to be used from MPLAB IDE, then you must install MPLAB IDE prior to installing Compiler.

MPLAB X IDE library builds See the New Features section for information pertaining to debugging library files.

Wrong referenced values shown (XC8-1758) Where a structure contains a member that is a pointer to another structure and that other structure contains a function pointer member, the IDE watch view might incorrectly show the values contained in the other structure.

Wrong parameter location shown (XC8-1565) In some cases, where the definitions of a function's

parameters were spread over multiple lines and the argument of the first parameter was originally stored in WREG, the debug information relating to the location of the parameter might have been incorrect and the IDE indicated an address `WREG0 (CPU)`, even if the parameter was moved to memory. As a workaround, consider placing the name of the function and its first parameter on the same source line, or use the option `-Xxc8 -W--dwarf-use-loclists=never`.

Debugging problems when using of macro for header name (XC8-1438) When including a header file whose name is specifier as a preprocessor macro, can result in incorrect line number information in the debugging files which would affect the ability to set breakpoints and step code. This issue only affects projects using the P1 parser.

Can't find space with absolutes (XC8-1327) In projects that used absolute addressed objects in program memory, a "cannot find space" error might have occurred, as a result of the assembler's psect-merging optimization. In such case, the issue can be worked-around by either disabling psect-merging (i.e. using the option `-Wa,-no_merge`) or by reserving the program memory occupied by those objects (using `-mrom` or `-mreserve`).

CMF error for maverick code (XC8-1279) Code or data that is not explicitly placed into a psect will become part of the default (unnamed) psect. Projects containing assembly code that were placed in such a psect caused the compiler to emit the error "(1437) CMF error: no psect defined for the line". As a workaround, place the assembly code in an appropriate psect. Using the default psect at any time is not recommended.

Use of XC8 in MPLAB IDE v8 is now deprecated (XC8-1228) As of MPLAB XC8 v1.34, the use of MPLAB 8 IDE is deprecated, and the installation of the DLL files that the IDE used to interface to XC8 are removed from the compiler installer.

ELF debugging issues Not all aspects of the ELF/DWARF debugging file have been implemented. MPLAB XC8 version 1.30 implements DWARF version 3 which allows for improved debugging. Only MPLAB X IDE supports ELF, but you must ensure that you are using a version of this IDE that can process ELF files produced by the compiler. The following are some of the issues that may not work as expected with this release of the compiler.

- **Unused variables (XC8-747)** will not be identified in the ELF file.
- **Constant propagation optimizations (XC8-744)** may affect which variables are watchable in the IDE or the values that are indicated in the Watch window.
- **In-line C functions (XC8-748)** will not be debuggable.
- **Procedural abstraction (XC8-749)** will affect the operation of breakpoints.
- **External memory variables (MPLABX-2004, MPLABX-2255, and others)** will not be displayed correctly in the Watch window.
- The *type* name (as opposed to the object's name) that appears for an anonymous structure or union typedef or an enumerated typedef will be shown as "." in the Watch window.
- The *type* name displayed for an identifier that was declared using a typedef type will be the identifier's semantic type rather than its typedef type in the Watch window.

6.2. Code Generation

Recursive calls incorrectly indicated (XC8-2451) The compiler might indicate that a function has been recursively called when indirectly called via a pointer that is also used to hold other function addresses at other times in the program. The compiler considers the targets of a pointer over the entire lifetime of the pointer object, which might fool the compiler into believing that recursion has taken place.

Not this array size (XC8-2445) Const-qualified arrays of bytes with a size of exactly 0x8000 will trigger an error indicating that the array dimension must be larger than zero. Larger or smaller arrays will compiler correctly.

Incorrect tracking of double indirection (XC8-2444) When a pointer variable is assigned an address indirectly via another pointer, the assigned address might not be correctly tracked by the compiler, resulting in an incorrect pointer size being used.

Size-of symbols not functioning (XC8-2412) The map file contains symbols, for example `__size_of_strncmp`, that suggest they might show the size of functions. These always indicate the value 0 and cannot be used to determine the size of any routine.

Missing delays (XC8-2345) When building for Baseline and Mid-range PIC devices, the `_delaywdt_us` and `_delaywdt_ms` delay builtins are missing.

Bad array access (XC8-2286) For projects that target PIC18 devices and that use expressions reading large arrays that can span multiple banks, array accesses using a constant array index might have used the wrong instruction. Using a pointer to the array (rather than the array identifier) and the same index will work correctly.

Read of volatile objects not performed (XC8-2273) If the true and false statements for an `if()` statement are identical, the compiler will try to simplify such situations. If the controlling expression inside the `if()` accesses seemingly redundant `volatile` objects, code to access to these objects might be incorrectly removed.

Wrong CRC write order (XC8-1997) The multi-byte CRCDATA register must be written in a specific byte order to ensure that the data is latched correctly. There is no guarantee that this will take place for C code that writes to this register as a whole. Code should instead write to the individual registers within CRCDATA (T, U, H, and L registers) in the intended order.

Bad tail merge optimization (XC8-1978) In rare circumstances where two or more PIC18 code sequences immediately before a jump to the same destination contain instructions that differ only in the banked/common access bit, these sequences might be incorrectly merged into one.

Undetected Division by zero in assembly (XC8-1960) Division of a literal constant by zero is not being detected in the assembler, which may cause the assembler to crash.

Illegal instruction not detected (XC8-1956) Using a `,w` operand to the `movwf` instruction does not trigger any error or warning, and the instruction is encoded use the Access Bank memory.

Right shift wrong (XC8-1941) Right-shifting a `signed long` variable by 16 bits can omit the sign extension, producing an incorrect result.

Incorrect pointer arithmetic (XC8-1940) Incrementing to a pointer to an array, e.g. `char (*ptr) [32]`, should add a value being the size of the array. Instead, it is adding the size of the array's elements.

Static function diagnostics (XC8-1938) Depending on the order of the input C files, one may or may not get an “undefined symbol” error for a function defined as `static` in one file but used and declared `extern` in another, but undefined at link-time.

Incorrect array sizes (XC8-1934) When compiling C99 programs, the compiler may make incorrect assumptions about the size of pointers used to define the size of an auto array.

Inappropriate and missing types (XC8-1886) The `<stdint.h>` header used by C99 builds defines `(u)int_fast16_t` as being types with a size of 4 bytes, whereas a type with a width of 2 bytes would be the fastest types. The `(u)int_fast24_t` types are missing entirely.

Case ranges ignored (XC8-1855) When the Clang front end is used for building, it silently ignores case ranges in `switch` statements and matches only the initial case value in the range. Clang is used when building for the C99 standard.

Unsupported directive (XC8-1817) The `DDW` assembly directive is not supported.

Bogus warning for absolute functions (XC8-1809) Functions that are made absolute might trigger the warning `(1262) object "*" lies outside available data space` when compiling for devices that have vectored interrupts, e.g. a PIC18F25K42. This warning can be ignored.

Generate no code warning (XC8-1803) Some of the C99 standard library character classification functions (viz. `isalpha()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `isspace()`, and `isupper()`) will cause the compiler to emit the warning `expression generates no code`. This is a side-effect of their implementation, and can be safely ignored.

FLT_ROUNDS macro not defined (XC8-1791) The `FLT_ROUNDS` macro, which should be defined by `<float.h>` is not defined and its use will result in an undefined symbol error.

Excessive consumption of decimals (XC8-1790) The A, E, F, and G conversion specifiers in the C99 `scanf()` functions will consume 'p' and 'P' in decimal numbers.

Special float strings not recognised (XC8-1789) The A, E, F, and G conversion specifiers in the C99 `scanf()` functions do not recognise strings representing infinity and NaN (not a number).

Errno affects scans (XC8-1786) If the `errno` variable is non-zero before calling any of the C99 `scanf()` functions, then any of the following conversion specifiers will fail: A, D, E, F, G, O, S and X.

Character set conversions not supported (XC8-1785) The `%[]` character-set conversion specifiers in the C99 `scanf()` functions are not supported.

White space not counted (XC8-1784) The `%n` conversion-specifier in the C99 `scanf()` fails to count white-space in the input string.

Error not set for string conversions (XC8-1782) The C99 `strtod()` family of functions (including `strtof()` and `strtold()`) do not set `errno` to `ERANGE` for values outside the representable range.

Scanning hex floating-point (XC8-1781) The C99 `scanf()` `a`, `e`, `f`, and `g` conversion specifiers will only convert the leading zero of any hexadecimal floating-point constant. This results in an assignment of zero to the function arguments which receive the converted input, and the remainder of the string from the `x` character is scanned according to the format string.

Undetected bad config values (XC8-1771) If the argument to the `#pragma config` directive is a literal numerical value with a valid initial value followed by illegal characters, for example `#pragma config CONFIG2L=0todo`, the initial portion of the value is read and the remainder silently ignored. This might be encountered of you accidentally attempt to specify a binary value to this pragma, using a `0b` prefix.

fmod() and zero (XC8-1641) The `fmod()` library function is non-compliant when second argument is zero. Currently, it returns the first argument in this case. It should return either trigger a domain error or return zero.

Can't generate code for floating-point operations (XC8-1613, XC8-1614) Expression that perform operations on the results of two complex floating-point expression, e.g. use of the `||` operator in the following `(( ! *pd2) - d2) || (((f1)-- ) >= *pf3))`, might trigger can't generate code error messages.

String literal expressions (XC8-1610) Accessing a character of a string literal might produce an error message when using the initialize an object, e.g. `volatile int a = "abc"[1];`.

Too many side-effects (XC8-1587) Incorrect PIC18 code is being generated for expressions involving compound assignment operators and where the lefthand side has side-effects. In this cases, the side-effects of the lefthand side will occur twice.

IDLOC value truncation (XC8-1570) The IDLOC values for some devices, e.g. PIC18F8723, are currently limited to being 4-bits wide. The truncation can be removed by editing the `.cfgdata` file (located in the `dat` directory) corresponding to your device, and changing the mask value, bolded in this example:

`CWORD:200000:`**`F`**`:FF:IDLOC0`

to the required value, for example:

`CWORD:200000:`**`FF`**`:FF:IDLOC0`

Register over-writes with recursive functions (XC8-1563) In some binary expressions or sub-expressions located in a recursively-called function, the compiler may allocate a static register to one sub-tree that might be clobbered by a recursive function call made in the other sub-tree. In this case, the compiler will now emit a warning message to indicate that the register might be corrupted.

Bad intermediate code from typedefs (XC8-1560) In some cases where a member of a structure has a typedefed type that is used in a previous definition before the structure, the code generator will emit the error message "bad intermediate code". This can possibly be worked-around by ensuring that the structure is the first definition to use the typedef'd type.

Non-removal of unused variables (XC8-1480) If a global variable is defined outside of library code and has had its address taken by a non-library function that is not called, the variable has memory allocated, even if it has not otherwise been used.

Device oscillator calibration (XC8-1280) Oscillator calibration using `--RUNTIME=+config` will not work with new devices that obtain their calibration constant from a calibration word stored in flash program memory, such as the MCP19114/5 devices. Disable this runtime sub-option and refer to the device data sheet for instructions.

Bogus warning of arithmetic overflow when subtracting (XC8-1270, XC8-1585) A warning regarding an `arithmetic overflow in constant expression` might be issued by the compiler when the expression contains a literal subtraction. This warning can usually be ignored. A recent change in the default wanting level imposed by the MPLAB X IDE has seen this compiler warning issued more frenetically by the compiler.

Inline assembly not output in PRO mode (XC8-1268) In PRO mode, inline assembly code enclosed in `#asm` - `#endasm` is simply not output by the code generator. The presence of any instruction in an `asm("");` statement before the `#asm` assembly results in the correct behaviour. Alternatively, you can place the `#asm` block inside braces `{ }`.

Printing spaces to width (XC8-1214) When using a `%d` placeholder and a width specifier, the C90 implementation of `printf()` function did not print leading spaces in the output when the printed value had few characters that the specified width, so for example the format string `"%04.2d"` might print `"77"` instead of `"  77"`. The C99 `printf()` function is not affected by this issue.

Functions called from in-line assembly(XC8-1180) The code generator will not be able to identify a C function called only from in-line assembly code if the definition for that C function is placed before the assembly call instruction in the source file. Placing the function definition after the call is acceptable. If the function cannot be identified, no code will be generated for the function and the linker will issue

undefined symbol errors.

Printf modifiers 'h' and 'L' ignored (XC8-1166) A `printf()` conversion specification that uses the `h` or L modifiers will result in the specification itself being printed, e.g. `%hx` will print `hx`. This does not affect the `printf()` in the C99 libraries.

Messy cleanup (XC8-1087) Running an XC8 ports-support uninstaller might leave behind several directories in the compiler's main directory.

Redefinition of intrinsic functions (XC8-1061) It not possible to replace a standard library function which uses the `intrinsic` pragma with a user-defined function with the same name. Examples of standard library functions that might use this pragma are: all of the inline delay functions (such as `_delay()`), `memcpy()`, and `__va_start()`.

Can't generate code error (XC8-1055) Interrupt code which indirectly calls a function and uses the return value of this call as a parameter to another function has been seen to trigger can't generate code errors. Split the statement so the call is made and the result assigned to a temporary variable. Pass the temporary variable to the subsequent call.

Accessing flash data memory (XC8-1047) None of the supplied flash library routines or macros associated with flash support those devices that utilize flash data memory. Devices without flash data memory are not affected by this limitation.

Persistent memory check functions (XC8-1029) The `persist_check()` and `persist_validate()` functions that were provided previously do not work with the new memory allocations schemes used by the compiler. These functions have been removed from the libraries and are not available for use.

Can't generate code errors (XC8-1022) In Free and PRO modes, code which indirectly accesses nested structure members might produce can't generate code errors.

Indirect function calls (XC8-1000) For midrange and baseline devices, there is a limit on the number of functions that can be called indirectly via a pointer. Typically this will be about 120 functions, but this limit is dependent on where the functions are linked. Code might crash if this limit is exceeded. This does not affect enhanced mid-range devices.

Multiple-assignment expressions (XC8-995) The compiler can crash when compiling a statement that involves multiple assignments and the assignment operands have side effects (such as referencing `volatile` objects), e.g. `a = b = c = d = e =...` Break up offending statements into many small ones.

Fedora path variable (XC8-474) The path variable will not be updated when non-root users install the compiler under Fedora. If you wish for the compiler driver to be in your path, update your path variable manually after installation of the compiler.

No static local specifiers (XC8E-313) The `__near` and `__far` object specifiers cannot be used with `static` local objects.

Absolute variables in access bank memory (XC8E-138) PIC18 projects that locate absolute variables in the lower addresses of the access bank RAM might trigger a `can't find space error` for the psect `temp` in class `COMRAM`. If a project must define absolute objects, try locating them at a higher address.

Absolute initialized variables (XC8E-63) Variables which are absolute and which are not const cannot be initialized. The following example will not generate an error, but will not work as expected. Define the variable as absolute and initialize it in main-line code.

```
unsigned int varname @ 0x0200=0x40; // will not work as expected
```

Bank qualifiers (XC8E-62) Only `bankx` qualifiers for data banks 0 through 3 are supported by the compiler. (These are enabled using the `--ADDRQUAL` option). Use absolute variables to place objects in other banks, if required.

In-line assembly and labels (XC8E-61) Functions which are called from both main-line and interrupt code should not contain in-line assembly that defines assembly labels. Such labels will not be assigned the usual duplication prefix (`i1`, `i2` etc) and will result in multiply-defined symbol errors.

Switch strategies (XC8E-20) There is only one possible switch strategy currently available for PIC18 devices. It uses the `space` switch type. New strategies will be introduced in future compiler versions so that PIC18 devices have similar options to the baseline/mid-range devices.

Stack overflow (XC8E-11) When the managed stack is used (the `stackcall` suboption to the `--RUNTIME` option is enabled) in some situations the stack may overflow leading to code failure. With this option enabled, if a function call would normally overflow the stack, the compiler will automatically swap to using a lookup table method of calling the function to avoid the overflow. However, if these functions are indirect function calls (made via a pointer) the compiler will actually encode them using a regular call instruction and when these calls return, the stack will overflow. The managed stack works as expected for all direct function calls, and for all indirect calls that do not exceed the stack depth.

Non-reentrant library functions Some library functions, for example the `printf()` family of functions, are not reentrant and may fail if multiple instances of them are active at the same time. This limitation exists even if you specify a reentrant stack setting.

New scanf() implementation The compiler does not currently customise the new C99 `scanf()` function in the same way as it does with the `printf()` function. The provided implementation of `scanf()` is feature complete, and, as a result, is extremely large. When used, this function's footprint could exceed the program memory size on your device. It is recommended only for large PIC18 devices when space is not a critical issue.

Redirecting bss variables If the `#pragma psect` directive is used to redirect objects that normally reside in any of the `bss` psects, the runtime startup code will not be aware of this and will clear the memory that the variables would have ordinarily be allocated. At such an early stage, this should not affect program execution, but if all bss objects are redirected, an undefined symbol error will occur with PIC18 devices. Consider using the `__section()` specifier.

Unsupported functions The `strdup()` function is not supported. The `strftime()` function is not supported for baseline devices.

Installer execution On both Mac OS X and Linux, it is necessary to run the installer as root or with superuser privileges (using `sudo`, for example). If the installer is started without superuser privileges on Mac OS X, it will exit and display an informative message. In the same situation on Linux, the installer will fail when it attempts to write to directories for which it does not have adequate rights. The messages displayed will relate to these access failures. For correct operation, run the installer via `sudo`, or as the root user, on these systems.

PATH environment variable On Linux systems, the installer, by default, updates the `PATH` environment variable to include paths to the new executables being installed. If the installer is run via `sudo`, the default action will update the `PATH` variable of the user executing the `sudo` command. If the installer is run by root, the installer will only update root's `PATH` variable, and not the `PATH` variables of ordinary users. If installing the compiler while logged in as root, a better choice is to update *all* user `PATH` variables. Alternatively, skip the step to update the `PATH` variable in the installer, and manually update

the PATH variables of users who will use the software.

PIC12F529T39A/T48A memory restrictions The previous limitation which restricted memory to the first 4 RAM banks for user-defined variables has been lifted. Note, however, that the compiler will not allow you to define objects that span multiple banks on these devices.

Psect pragma and data psects As described in the manual, the #pragma psect directive should not be used to move initialized variables that would normally be located in one of the 'data' psects. The initial values in program memory and space for the variables themselves in RAM must be built up in a strict order. Using this pragma will violate this assumption. Consider using the __section() specifier.

Copying compiler header files The header files shipped with the compiler are specific to that compiler version. Future compiler versions may ship with modified header files. If you copy compiler header files into your project, particularly if you modify these files, be aware that they may not be compatible with future versions of the compiler.

Can't Generate Code messages When compiling for baseline devices, some complex expressions may cause compile-time errors (712) Can't Generate Code for this expression. The expressions should be simplified to work around this. This may require the use of additional variables to store intermediate results. This is most likely with long integer or floating-point arithmetic and particularly those devices with less than 4 bytes of common memory available.

Option and tris register access For baseline devices, the OPTION and TRIS registers must be written as a byte. Writing individual bits is not supported.

PIC17 support PIC 17 devices (for example, 17C756) are not supported by this compiler.

Configuration words (PIC18 parts only) The new device support introduced in PICC18 v9.80 will not automatically program the default values into the configuration words when no value is specified. If your project does not program all configuration words explicitly, select the option "Program the device with default config words" in the Linker tab.

Specifying configuration words on PIC10/12/16 devices The __PROG_CONFIG() and __CONFIG() macros can be used to specify the configuration words on PIC10/12/16 devices as well as PIC18 devices, but only when building for C90. The __PROG_CONFIG() macro must use a literal constant argument; you cannot use the configuration setting symbols with this macro. The __CONFIG() macro must only use the predefined configuration setting symbols and you may not not use a literal value with this macro.

rfPIC12 parts To use the rfPIC12 parts, for example the rfPIC12C509AF, you will need to specify to the compiler a part name in a format similar to RF509AF, for example. You can also use an alias like 12C509AF, for example. The full part name is also not appropriate when compiling from MPLAB IDE.

7. Device Errata

For 8-bit PIC devices, this release of the XC8 compiler recognizes the published silicon errata issues listed in the table below. Some of these issues have been corrected and no longer apply in recent silicon revisions. Refer to Microchip's device errata documents for details on which issues are still pertinent for your silicon revision. The compiler's chip configuration file records which issues are applicable to each device. Specific errata workarounds can be selectively enabled or disabled via the driver's -merrata command line option.

All these errata are PIC18 specific, except for the CLOCKSW and BRANCH errata, which applies to enhanced mid-range devices.

Name	Description	Workaround details
4000	Execution of some flow control operations may yield unexpected results when instructions vector code execution across the 4000h address boundary.	Each block of program code is not allowed to grow over the 4000h address boundary. Additional NOP instructions are inserted at prescribed locations.
FASTINTS	If a high-priority interrupt occurs during a two-cycle instruction which modifies WREG, BSR or STATUS, the fast- interrupt return mechanism (via shadow registers) will restore the value held by the register before the instruction.	Additional code reloads the shadow registers with the correct values of WREG, STATUS and BSR.
LFSR	Using the `lfsr` instruction to load a value into a specified FSR register may also corrupt a RAM location.	The compiler will load FSR registers without using the `lfsr` instruction.
MINUS40	Table read operations above the user program space (>1FFFFFh) may yield erroneous results at the extreme low end of the device's rated temperature range (-40o C).	Affected library sources employ additional `nop` instructions at pre- scribed locations.
RESET	A `goto` instruction placed at the reset vector may not execute.	Additional `nop` instruction inserted at reset vector if following instruction is `goto`
BSR15	Peripheral flags may be erroneously affected if the BSR register holds the value	Compiler avoids generating `movlb 15` instructions. A warning is issued if this

	15, and an instruction is executed that holds the value C9h in its 8 least significant bits.	instruction is detected.
DAW	The DAW instruction may improperly clear the CARRY bit (STATUS<0>) when executed.	The compiler is not affected by this issue.
EEDATARD	When reading EEPROM, the contents of the EEDATA register may become corrupted in the second instruction cycle after setting the RD bit (EECON1<0>).	The `EEPROM_READ` macro read EEDATA immediately.
EEADR	The result returned from an EEPROM read operation can be corrupted if the RD bit is set immediately following the loading of the EEADR register.	The compiler is not affected by this issue.
EE_LVD	Writes to EEPROM memory may not succeed if the internal voltage reference is not set.	No workaround applied
FL_LVD	Writes to program memory may not succeed if the internal voltage reference is not set.	No workaround applied
TBLWTINT	If a peripheral interrupt occurs during a `tblwt` operation, data can be corrupted.	Library routine `flash_write()` will temporarily disable all applicable interrupt-enable bits before execution of a `tblwt` instruction.
FW4000	Self write operations initiated from and acting upon a range within the same side of the	No workaround applied

	4000h boundary may fail based on sequences of instructions executed following the write.	
RESETRAM	Data in a RAM location can become corrupted if an asynchronous reset (e.g. WDT, MCLR event) occurs during a write operation to that location.	A warning will be issued if the length nvram psect is greater than zero bytes (persistent variables populate this psect).
FETCH	Instruction fetches can become corrupted after certain code sequences.	A `nop` instruction as added after `tblrd` instructions, returns, destinations of calls and gotos, and ISR vector addresses.
CLOCKSW	An instruction may be corrupted when switching from INTOSC to an external clock source. (Enhanced mid-range devices)	Switch to high-power mode immediately after reset.
BRANCH	The PC might become invalid when restoring from an interrupt during a BRA or BRW instruction. (Enhanced mid-range devices)	Branch instructions are avoided.
BRKNOP2	Hardware breakpoints might be affected by branch instruction.	Use 2 nops instead of `BRA <pc+1>`.
NVMREG	The program will access data flash rather than program flash memory after a reset, affecting runtime startup code.	The runtime startup code adjusts the NVMCON register to ensure that program memory is accessed by table read instructions.
BSR63	Corrupted execution of	Compiler avoids generating

	`movff` instruction when the BSR holds 63	`movlb 63` instructions. A warning is issued if this instruction is detected.