# MPLAB® XC8 C Compiler Version 3.00 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.

## Contents

# 1 Overview

## 1.1 Introduction

This release of the Microchip MPLAB® XC8 C compiler is a major update that includes significant new features, as well as bug fixes and expanded device support.

## 1.2 Build Date

The official build date of this compiler version is December 4, 2024.

## 1.3 Previous Version

The previous MPLAB XC8 C compiler version was 2.50, built on July 26, 2024.

## 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 18.1.8 based upon LLVM 18.1.8.

## 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® 10 and 11, Ubuntu® 20.04, macOS® 13.2 (Ventura) and 12.5 (Monterey), and Fedora 34. Binaries for Windows have been code-signed. Binaries for macOS have been code-signed and notarized.

The MPLAB XC Network License Server is available for a variety of 64-bit operating systems, including Microsoft Windows 10 and above; Ubuntu 20.04 and above; or macOS® 13.2 (Ventura) and above. The server may also run on various other operating systems including Windows Server, Linux distributions, such as Oracle® Enterprise Linux® and Red Hat® Enterprise Linux as well as older versions of supported operating systems. However, the MPLAB XC Network License Server is not tested on these operating systems. The MPLAB XC Network License Server can be run on Virtual Machines of the supported OS using a virtual machine license for network licenses (SW006021-VM). All 32-bit versions of the MPLAB XC Network Server are discontinued starting from version 3.00.

## 1.7 Devices Supported

This compiler supports all available 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. This release of the MPLAB XC Functional Safety Compiler supports the Network Server License.

See the *Installing and Licensing MPLAB XC C Compilers* (DS50002059) document for information on license types and installation of the compiler with a license.

## 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` (macOS) 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 macOS 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.

### 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 user's guides shipped with the compiler. These can be opened from links in the HTML page that opens in your browser when clicking the blue help button in the MPLAB X IDE dashboard, as indicated in the screenshot.

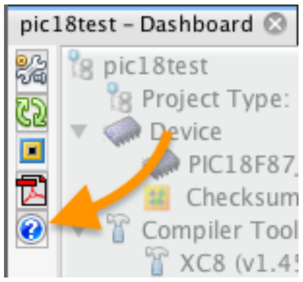

Figure 1: Where to access documentation from within the MPLAB X IDE

If you are building for 8-bit PIC targets, the MPLAB® XC8 C Compiler User's Guide for PIC® MCU contains information on those compiler options and features that are applicable to this architecture.

## 1.11 Customer Support

You can 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.

This release includes the following new or updated PIC documentation:

- *Microchip Unified Standard Library Reference Guide* (revision D)
- *MPLAB® XC8 C Compiler User's Guide for PIC® MCU* (revision J)
- *MPLAB® XC8 PIC® Assembler User's Guide* (revision E)
- *Hexmate User's Guide* (XC8-3601 - revision D)

As part of (XC8-3475), the release notes shipped with v3.00 and later versions of this compiler have been prepared using new tools. The content should remain unchanged but might be presented differently. Information regarding changes to version 1.x compilers has been removed but can be obtained from the release notes for any v2.x or v1.x compiler.

The *MPLAB® XC8 C Compiler User's Guide for PIC® MCU* describes the operation and options associated with the new `xc8-cc` driver.

The *Microchip Unified Standard Library Reference Guide* describes the behavior of and interface to the functions defined by the Microchip Unified Standard Library, as well as the intended use of the library types and macros. Some of this information was formerly contained in the *MPLAB® XC8 C Compiler User's Guide for PIC® MCU.* Device-specific library information is still contained in this compiler guide.

If you are just starting out with 8-bit devices and the MPLAB XC8 C Compiler, the *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 now distributed with the compiler.

The *MPLAB® XC8 PIC® Assembler User's Guide* is intended for those written assembly-only projects. If you are using assembly with C code, instead refer to the Assembler section in the *MPLAB® XC8 C Compiler User's Guide for PIC® MCU.*

The *Hexmate User's Guide* is intended for those running Hexmate as a stand-alone application. If you are using Hexmate as part of the MPLAB XC8 C compiler, instead refer to the Hexmate section in the *MPLAB® XC8 C Compiler User's Guide for PIC® MCU.*

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

## PIC18 Program Memory Symbols

Microchip PIC18 devices have various regions in their program memory address space in addition to the program flash memory (PFM). These regions contain device and configuration information of utility to user projects, and can be read using table reads, the same mechanism employed by the compiler to read constant data stored in PFM. The compiler defines special `extern const` symbols mapped to these regions in the device-specific headers. These symbols can be used by C programs to easily read the contents of these regions and are available once you include <xc.h> in your source files.

The regions represented and examples symbols defined for these regions are described below:

**User IDs** Words in the program memory space designated as ID locations where the user can store checksum or other code identification numbers. These words can be programed using the `#pragma config` directive. Example symbol definitions provided:

```
extern const unsigned _IDLOCs[8] __at(0x200000);
extern const unsigned _IDLOC0 __at(0x200000);
extern const unsigned _IDLOC1 __at(0x200002);
extern const unsigned _IDLOC2 __at(0x200004);
```

**Configuration Words** Words and bits that allow the user to setup a device with choices of oscillators, resets, memory protection options, etc. These words can be programed using `#pragma config` directive. Example symbol definitions provided:

```
extern const unsigned char _CONFIGs[10] __at(0x300000);
extern const unsigned char _CONFIG1L __at(0x300000);
typedef union {
    struct {
        unsigned FEXTOSC :3;
        unsigned :1;
        unsigned RSTOSC :3;
    };
} _CONFIG1Lbits_t;
extern const _CONFIG1Lbits_t _CONFIG1Lbits __at(0x30000);
extern const unsigned char _CONFIG1H __at(0x300001);
typedef union {
    struct {
       unsigned CLKOUTEN :1;
       unsigned PR1WAY :1; unsigned CSWEN :1;
       unsigned FCMEN :1;
    };
} _CONFIG1Hbits_t;
extern const _CONFIG1Hbits_t _CONFIG1Hbits __at(0x30001);
```

**Device Information Area (DIA)** The DIA contains contains information such as calibration data for the internal temperature indicator modules, stores the Microchip Unique Identifier words and the fixed voltage reference voltage, etc. Example symbol definitions provided:

```
extern const unsigned _DIA[32] __at(0x3F0000);
extern const unsigned _DIA_MUI[8] __at(0x3F0000);
extern const unsigned _DIA_MUI0 __at(0x3F0000);
extern const unsigned _DIA_MUI1 __at(0x3F0002);
extern const unsigned _DIA_MUI2 __at(0x3F0004);
extern const unsigned _DIA_EUI[10] __at(0x3F0010);
extern const unsigned _DIA_EUI0 __at(0x3F0010);
extern const unsigned _DIA_EUI1 __at(0x3F0012);
extern const unsigned _DIA_EUI2 __at(0x3F0014);
```

**Device Configuration Information (DCI)** The DCI holds information about the device which is useful for programming and bootloader applications. Example symbol definitions provided:

```
extern const unsigned _DCI[5] __at(0x3FFF00);
extern const unsigned _DCI_ERSIZ __at(0x3FFF00);
extern const unsigned _DCI_WLSIZ __at(0x3FFF02);
extern const unsigned _DCI_URSIZ __at(0x3FFF04);
extern const unsigned _DCI_EESIZ __at(0x3FFF06);
extern const unsigned _DCI_PCNT __at(0x3FFF08);
```

**Revision ID and Device ID** Unique identifiers for the device and its revision. Example symbol definitions provided:

```
extern const unsigned _DEVID __at(0x3FFFFE);
extern const unsigned _REVID __at(0x3FFFFC);
```

## Register Usage

The following complements the information already contained in *Section 5.6 Register Usage* in the User's Guide.

Note that the compiler determines the size of all `const` data defined by a program. For total sizes less than 64k and when the `mediumconst` psect is used to hold this data, the TBLPRTU register only needs to be initialized at runtime startup and does not need to be modified during program execution. For total sizes less than 256 bytes, the psect holding this data (`smallconst`) is positioned so that both the upper and high table pointer registers (TBLPTRU and TBLPTRH) can be assumed to hold the value assigned at runtime startup. As a consequence, any code that directly or indirectly modifies the table pointer registers that the compiler assumes will hold their value during program execution could lead to code failure. Such modification could occur if the program assigns an address to a pointer that is not of an object in the normally accessible program memory space. If you do need to assign such an address, or you must manually modify the table pointer registers in any way, save the content of the TBLPTRH and TBLPTRU registers as required before the assignment, and restore the registers with their saved values afterward. If in doubt as to which table pointer registers need saving, you can save all of them using code such as:

```
#include <xc.h>
uint24_t saved_tblptr = TBLPTR;
// place code that modifies TBLPTR here
TBLPTR = saved_tblptr;
```

### Address-mask Macro

The macro `_PTR3_ADDR_MASK` can be used to obtain the address bits encoded within certain pointers when building for PIC18 devices. The macro is defined as a mask that when ANDed with a 24-bit mixed-space address will strip away any bits used to identify the memory space of the pointer target, leaving just the address in that memory space.

## 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.

### Version 3.00

**New device support** No new PIC devices are supported by this release.

**Shipped DFPs** The following are the PIC-specific DFP versions that ship with the compiler: MCPxxxx_DFP v1.7.176, MCVxxxx_DFP v1.5.154, PIC10-12Fxxx_DFP v1.7.178, PIC12-16Cxxx_DFP v1.6.168, PIC12-16F1xxx_DFP v1.7.242, PIC16F1xxxx_DFP v1.25.389, PIC16Fxxx_DFP v1.6.156, PIC18Cxxx_DFP v1.5.163, PIC18F-J_DFP v1.8.164, PIC18F-K_DFP v1.13.292, PIC18F-Q_DFP v1.25.433, and PIC18Fxxxx_DFP v1.6.159.

**XCLM update (XC8-3479)** The XCLM license manager has been updated to version 3.22.

**LSP at your service (XC8-3466)** The compiler now provides an implementation of a server conforming to the Language Server Protocol (LSP) using clangd. This server can be used by IDEs to obtain some intelligence regarding XC8 source code.

**PIC18 program memory region symbols (XC8-3451)** Symbols have been added to the device-specific C header files that can be used for read-only, direct access to special program memory regions on PIC18 devices. These regions are the configuration registers, user-ID locations, DCI and DIA tables, and device and revision IDs. See Documentation Updates in these notes for more information.

**Clang updated (XC8-3016)** The Clang compiler front end has been updated from version v4.0.1 to v18.1.8. This update includes numerous bug fixes and improvements to diagnostics and warnings, along with updated support for C language standards. These changes include:

- The warning controlled by `-Wstrict-prototypes` is new and diagnoses deprecated declarations and definitions of functions without a prototype where the behavior in C2x will remain correct.
- The warning controlled by `-Wtautological-compare` will now look through member and array access to determine if two operand expressions are the same.
- The new warning controlled by `-Wsizeof-array-div` catches cases like `int arr[10]; ...sizeof(arr) / sizeof(short)`, and the existing warning controlled by `-Wsizeof-pointer-div` catches more cases.

- The new warning controlled by `-Wxor-used-as-pow` warns where it looks like the xor operator, `^`, was used to mean exponentiation
- The new warning controlled by `-Wimplicit-const-int-float-conversion` (enabled by default and controlled by `-Wimplicit-int-float-conversion`) warns on implicit conversion from an integer constant to a float type.
- The new warning controlled by `-Wpointer-to-int-cast` warns about C-style casts of pointers to an integer type too small to hold all possible values.
- The warning controlled by `-Wdeprecated-non-prototype` will diagnose cases where the deprecated declarations or definitions of a function without a prototype will change behavior in C2x; and additionally diagnose calls which pass arguments to a function without a prototype.
- (XC8-3596) The compiler will no longer issue a warning (controlled by `-Wmissing-prototype`) about `main()` having a missing prototype.
- Clang now appropriately issues an error when a definition of a C function without a prototype and with no arguments is an invalid redeclaration of a function with a prototype. For example, `void f(int); void f() {}` is now properly diagnosed.
- The diagnostic controlled by `-Wimplicit-function-declaration` now defaults to an error in C99 and later standards.
- The diagnostic controlled by `-Wimplicit-int` now defaults to an error in C99 and later standards.
- Using `#warning`, `#elifdef,` and `#elifndef` that are incompatible with C standards before C2x/C++2b are now warned via CCI or `the -Wpedantic` option.
- The diagnostic controlled by `-Wint-conversion` for implicit int <-> pointer conversions now defaults to an error in all C language modes but may be downgraded to a warning with `-Wno-error=int-conversion`, or disabled entirely with `-Wno-int-conversion.`
- The diagnostics controlled by `-Wimplicit-function-declaration` and `-Wimplicit-int` now default to an error in C99.
- The diagnostic controlled by `-Wincompatible-function-pointer-types` now defaults to an error in all C language modes but may be downgraded to a warning with `-Wno-error=incompatible-function-pointer-types`, or disabled entirely with `-Wno-incompatible-function-pointer-types.`
- The `--language=assembler-with-cpp` option is enabled for all hand-written assembly source files; the `--language=c` option is enabled for all C source files.
- The following table shows which Clang options are enabled by default for the different C standards.

| Option | C90 | C99 | CCI |
|---|---|---|---|
| `-Wno-gcc-compat` | X | X | X |
| `-std=c90` | X | | |
| `-std=c99` | | X | |
| `-Wconversion` | X | X | X |
| `-pedantic` | | | X |
| `-Werror=gnu-case-range` | | | X |
| `-Wgnu-case-range` | X | X | |

Further information can be found in the Clang release notes, at
`https://releases.llvm.org/`*`X`*`.0.0/tools/clang/docs/ReleaseNotes.html`,
where *X* spans versions 5 through 18.

**Software interrupt vector tables (XC8-2991)** The compiler can now provide an interrupt vector table and dispatch mechanism in software for any device that supports interrupts but does not implement a hardware VIC module. This allows such devices to be programmed as if they had one vector for each interrupt source and so benefit from reduced interrupt latency. A software interrupt vector table is created when a `__flags()` argument is used with any `__interrupt()` specifier.

## Version 2.50

**New Device Support** Support is now available for the following PIC devices: 16F18013, 16F18014, 16F18023, 16F18024, 18F04Q20, 18F05Q20, 18F14Q20, and 18F15Q20.

**Shipped DFPs** The following are the PIC-specific DFP versions that ship with the compiler: MCPxxxx_DFP v1.7.176, MCVxxxx_DFP v1.5.154, PIC10-12Fxxx_DFP v1.7.178, PIC12-16Cxxx_DFP v1.6.168, PIC12-16F1xxx_DFP v1.7.242, PIC16F1xxxx_DFP v1.25.389, PIC16Fxxx_DFP v1.6.156, PIC18Cxxx_DFP v1.5.163, PIC18F-J_DFP v1.8.164, PIC18F-K_DFP v1.13.292, PIC18F-Q_DFP v1.25.433, and PIC18Fxxxx_DFP v1.6.159.

**Disable assembly warnings in code (XC8-3309)** A new PIC Assembler directive has been introduced to control warning and advisory messages from assembly code. The `ERRORLEVEL` directive takes a comma-separated list of arguments. The arguments 0, 1, and 2 enable all warning and advisory messages, disables advisory messages, and disables both advisory and warning messages, respectively. Individually numbered messages can be disabled by prefixing the message number argument with a negative sign. Prefixing with a positive sign will enable the message. See the Documentation Updates section for more information.

**Disable warnings (XC8-3308)** A new `xc8-cc` driver and `pic-as` assembler driver option can be used to enable and disable warning and advisory messages. The `-Wno-`*msg* form of the option disables the message. The `-W`*msg* form of the option enables the indicated message. The special message argument `all` indicates that all messages should be enabled or disabled. See the Documentation Updates section for more information.

**Optimization of subtraction assignments (XC8-3297)** Projects targeting enhanced mid-range devices using optimization level `-O2` or less might see an improvement in code size and execution speed for expressions using subtraction assignments.

**Expanded undefined interrupt feature (XC8-3170)** Programs that target devices without a Vectored Interrupt Controller (VIC) module can now use the `-mundefints` option to assign an instruction to any unassigned vectors. The compiler will issue a warning when this automatic assignment occurs.

## Version 2.49 (Functional Safety Release)

**Disabled dongle (XC8-3312)** To improve compile times, the functional safety compiler installer instructs the installed license manager to omit the check for a dongle license. Dongle licenses are not available with functional safety compilers, so this does not impact on the compiler's usability.

## Version 2.46

**New Device Support** Support is now available for the following PIC devices: 16F13113, 16F13114, 16F13115, 16F13123, 16F13124, 16F13125, 16F13143, 16F13144, 16F13145, 18F24Q24, 18F25Q24, 18F26Q24, 18F45Q24, 18F46Q24, 18F55Q24, and 18F56Q24.

**Shipped DFPs** The following are the PIC-specific DFP versions that ship with the compiler: MCPxxxx_DFP v1.7.176, MCVxxxx_DFP v1.5.154, PIC10-12Fxxx_DFP v1.7.178, PIC12-16Cxxx_DFP v1.6.168, PIC12-16F1xxx_DFP v1.7.242, PIC16F1xxxx_DFP v1.23.382, PIC16Fxxx_DFP v1.6.156, PIC18Cxxx_DFP v1.5.163, PIC18F-J_DFP v1.7.159, PIC18F-K_DFP v1.13.292, PIC18F-Q_DFP v1.23.425, and PIC18Fxxxx_DFP v1.6.159.

**Dongle license option (XC8-3281)** The compiler installer now provides an option to control whether checks are made for a USB dongle when the compiler operates. When not using a dongle license, ensure the dongle license option is disabled when installing the compiler, as this might improve compile times.

**Hash XOR option (XC8-3262)** A new `xor` sub-option to the `-mchecksum` driver option has been added. This sub-option takes a hexadecimal argument, which may have an optional `0x` prefix, and is XORed with the hash result before it is stored. It is only applied to CRC algorithm results and is ignored for other algorithms.

**XCLM Upgrade (XC8-3259)** The XCLM license manager has been upgraded to version 3.20.

**Warning promotion option (XC8-3238)** A new command-line option, `-W[no-]error`, has been added to the both the Compiler and PIC Assembler driver applications. This optionpromotes warning messages into errors. Alternate forms of the option can suppress any conversion for particular warnings. This option can control numbered warnings, which are produced by most compiler applications, as well as string-identified warnings produce by the Clang front end.

**Improved switch processing (XC8-3156)** When processing `switch()` statements with `char` arguments and using the C99 standard, the Clang front end was producing output that the compiler could not properly optimize. This has been improved so that `switch()` statements operating on small-typed arguments will produce less code.

## Version 2.45

**New device support** Support is now available for the following PIC parts: 16F17154, 16F17155, 16F17156, 16F17174, 16F17175, 16F17176, 16F18154, 16F18155, 16F18156, 16F18174, 16F18175, 16F18176, 18F06Q20, 18F16Q20, 18F24Q71, 18F25Q71, 18F44Q71, 18F45Q71, 18F54Q71, and 18F55Q71.

**Universal license manager (XC8-3176, XCLM-224)** The macOS version of the license manager used with the compiler is now universal, offering native support for both Intel- and M1-based machines. The Linux version of the license manager now requires at least version 2.25 of glibc.

**Mac universal binaries (XC8-3168, XC8-3116, XC8-2951, XC8-2693)** The compiler binary files for macOS are now universal, offering native support for both Intel- and M1-based machines.

**New errata workarounds (XC8-3148)** By default, the compiler will now enable the `BSR63` errata workaround when building for the following devices: 18F25K83, 18F26K83, 18LF25K83, and 18LF26K83.

**Stack bound symbols (XC8-3145)** New symbols have been added whose values represent the upper and lower bounds of the memory reserved for the software data stacks when either the hybrid or reentrant stack models have been requested. For the main stack, these symbols are: `__stack_lo` and `__stack_hi`. For the interrupt stack on enhanced mid-range devices, they are: `__int_stack_lo` and `__int_stack_hi`. For PIC18 devices, the symbols for high- and low-priority interrupt priorities are respectively: `__inthi_stack_lo` and `__inthi_stack_hi`, and `__intlo_stack_lo` and `__intlo_stack_hi`. These symbols are C accessible and can be used at runtime to ensure that the address of the stack pointer has not moved outside the reserved stack memory. The symbols will be automatically declared when you include `<xc.h>` into your source. They could be used in code similar to:

```
#include <xc.h>
```

```
if(FSR1 < (unsigned short)&__stack_lo)
   stackError();
if(FSR1 >= (unsigned short)&__stack_hi)
   stackError();
```

**Hexmate comment support (XC8-3144)** Hexmate now supports the presence of comments in HEX files. A comment is considered to be any line containing a sequence of characters that does not include a colon character, any sequence of characters before a colon character that starts a valid record, or any sequence of characters that includes a colon character that is not followed by a valid record. Although comments are permitted in input files, they are not generated in any output.

**New NOPs (XC8-3114)** To accommodate those devices which require a hand-written `NOP` instruction encoded as 0x0000 in errata workarounds, the C built-in function `__nop()` and the assembler instruction `NOP` will now use encoding 0x0000. Previously, these used the opcode 0xF000. The encoding for any NOP instructions added by a compiler-applied errata workarounds remains unchanged; however, new macros have been provided to allow the desired encoding of the `NOP` instruction to be conveyed to the compiler. These are:

`__nopf000()` and `nopf000` to ensure an encoding of 0xF000
`__nop0000()` and `nop0000` to ensure an encoding of 0x0000; and
`__nopffff()` and `nopffff` to ensure an encoding of 0xFFFF

**Library improvements (XC8-3046)** Code and data improvements have been made to many C99 library functions supplied by the Microchip Unified Standard Library. Improvements were made to functions in the `<stdio.h>`, `<ctype.h>`, `<time.h>`, `<string.h>`, `<stdlib.h>`, `<math.h>`, and `<stdint.h>` libraries.

## Version 2.41

**New device support (XC8-2954, XC8-2734)** Support is now available for the following PIC parts: 18F26Q71, 18F46Q71, and 18F56Q71.

**Smarter smart IO (XC8-2986)** Improvements have been made to the Smart IO functionality. These factor common code in field width and precision handling, resulting in code and data savings in the generated IO functions, as well as increases in their execution speed.

**Time type change (XC8-2982)** The C99 standard library type, `time_t` has been changed from a `long long` to an `unsigned long` type, which gives code-size improvements in some time-related functions, such as `mktime()`.

**Update to XCLM (XC8-2944)** The license manager used with the compiler has been updated and is now more responsive when checking the compiler's license details.

**C99 support for pedants (XC8-2917)** The `-Wpedantic` option has been extended to support projects conforming to the C99 language standard. It will warn when forbidden language extensions have been used in a project.

**Smaller printing functions (XC8-2960)** The size of auto variables used in integral conversions for the C99 standard library formatted printing functions has been optimised to suit the values these function are expected to print. This will reduce the memory footprint of these functions and improve their performance when programs print only smaller integral types.

**OTP register indication (XC8-2825)** Any configuration or user-id registers that are one-time programmable (OTP) will be indicated as such in the device-specific HTML documentation for relevant devices. Check the `pic_chipinfo.html` and `pic18_chipinfo.html` files, both located in the `docs` directory under your compiler' installation directory.

## Version 2.40

**New device support** Support is now available for the following PIC parts: 16F17114, 16F17115, 16F17124, 16F17125, 16F17126, 16F17144, 16F17145, 16F17146, 16F18015, 16F18025, 16F18026, 16F18044, 16F18045, 16F18046, 16F18054, 16F18055, 16F18056, 16F18074, 16F18075, 16F18076, 16F18114, 16F18115, 16F18124, 16F18125, 16F18126, 16F18144, 16F18145, and 16F18146.

**Formated IO optimization** Many improvements have been made to the vfprintf-related code, which forms the basis to most of the formatted output functions. These changes allow for finer granularity of customization, reducing the inclusion of redundant code. The usage of the `strcmp()` function in the runtime detection of format conversion specifiers has also been simplified and in some cases eliminated, further improving code efficiency.

**Faster compilation speed** Compilation speed improvements have been made that will be particularly noticeable when projects use a large number of string literals. This change might also prevent out-of-memory errors from occurring on the machine hosting the compiler.

**Simplified atexit function** The C99 standard library function `_atexit()` has been greatly simplified to reduce code size.

**Optimal selection** of move instruction When building projects that target PIC18 devices that have the `movffl` instruction, the compiler may have used this larger instruction in places where the smaller `movff` instruction could have been used.

**MPASM-compatible operand masking** The PIC Assembler can now be put into a mode where it will automatically truncate all instructions operands, similar to how they are masked when using MPASM. The mode is controlled by a new linker option, which can be specified from the compiler driver, using `-Wl,--fixupoverflow=error|warn|lstwarn|ignore`. The `ignore` setting will have the PIC Assembler closely mimic the behaviour of MPASM, truncating all operands to suite the instruction silently. The `warn`, `lstwarn` or both these settings, colon separated, will have the linker truncate operand values to suite the instruction, but also issue a warning on the console, in the assembler list file, or in both, respectively, whenever truncation occurs, so that situations where truncation was not expected can be investigated. Truncation will occur for all operands of all instructions to a size suitable for the instruction.

## Version 2.39 (Functional Safety Release)

**Network Server License** This release of the MPLAB XC8 Functional Safety Compiler supports the Network Server License.

## Version 2.36

None.

## Version 2.35

**New device support** Support is available for the following PIC parts: 16F15254, 16F15255, 16F15256, 16F15274, 16F15275, and 16F15276.

**Warning for Block Table Read Protection** The compiler will now issue a warning when it detects that any of the Block Table Read Protection configuration bits have been enabled. Enabling these bits might adversely affect variable initialization and reading constants from program memory.

**Warning when enabling Storage Area Flash** The compiler will now issue a warning when it detects that the Storage Area Flash has been enabled via the relevant configuration bits.

With this area enabled, the user must ensure this memory is not used for program code, typically done by reserving program memory when building the project.

**Constant caching** The `-fcacheconst=[on|off|auto]` PRO-mode option enables new optimizations that can automatically cache objects qualified `const` in any unused data memory, accessing them from this location rather than from program memory.

**Smarter IO** The compiler will now analyze program code for calls to the scanf family of functions in the same way it has done for the printf family of functions, customising the available scan functions' features so that it is optimal for how it will be used in the program. This feature is fully automatic.

**Smart IO assistance option** When analyzing calls to smart IO functions (such as `printf()` or `scanf()`), the compiler cannot always determine from the format string or infer from the arguments those conversion specifiers required by the call. Previously, the compiler would always make no assumptions and ensure that fully functional IO functions were linked into the final program image. A new `-msmart-io-format=`*fmt* option has been added so that the compiler can instead be informed by the user of the conversion specifiers used by smart IO functions whose usage is ambiguous, preventing excessively long IO routines from being linked.

**Dynamic memory allocation** The compiler now supports dynamic memory allocation onto a heap via the usual `<stdlib.h>` functions, such as `malloc()`. Memory can be released, using the `free()` function, but the management of unused memory is rudimentary. A new `-mheap` option can be used to reserve memory for the heap. Only use dynamic memory allocation when required, and then avoid unnecessary allocation and freeing of memory blocks.

**Bank selection for indirect access** The `BANKISEL` assembler directive is now available for mid-range and Baseline devices. This directive will set the IRP bit (mid-range) or STATUS bits (Baseline) appropriately for a given symbol argument so that indirect access of that object can be made. For all other devices, this directive will be ignored.

**Generation 4 tool support** The compiler now supports the selection of all the generation 4 debug tools (e.g. MPLAB ICD4, PICkit 4, and SNAP) through the `-mdebugger` option. The corresponding predefined macros are indicated in the User's Guide.

**String support in initialization directives** The assembler will now accept a string argument to the `DB`, `DW`, and `DDW` directives. The string can be specified with either single or double quote characters with the backslash used to escape special characters, for example:

```
DB 'directive\'s usage'
DW "a terminated string",0
```

**setjmp/longjmp support** The C90 and C99 library functions `setjmp()` and `longjmp()` are now available for enhanced mid-range devices. PIC18 devices have had support for these functions in the past, but due to device constraints, the functions are still not implemented for Baseline and other mid-range devices.

**CONFIG assembler examples** The chipinfo HTML files have been updated to show example usage of the `CONFIG` assembler directives used with the PIC Assembler to set the device configuration bits.

**Wider use of reverse-word hashes** The Hexmate reverse-word processing hash calculation feature can now be applied to Fletcher algorithms. This feature was previously applicable to only CRC and SHA algorithms. It has no affect on the additive or subtractive algorithms.

**Final XOR with Hexmate hashes** Hexmate can now be asked to perform an XOR of a hash result with a specified value. Use `o` or `O` followed by the value as an additional argument to Hexmate's `-CK` option.

**I16HEX file support in Hexmate** Hexmate has been extended to read type 2 records, as well as a new `--ssa` option to specify the Start Segment Address used by type 3 records. INHX16 files can be written by using the new `inhx16` argument to the `--format` option.

**Hexmate II** The Hexmate application shipped with the compiler is now built from a new code base. Although the application's features and the options that drive it are mostly identical to those in the previous incarnation, the parsing of its options is much improved. See also Migration Issues.

**Microchip Unified Standard Libraries** All MPLAB XC compilers will share a Microchip Unified Standard Library. The MPLAB XC8 compiler has shipped with this library for several versions, but with this release, the available functions have been expanded and the operation of some functions has been standardized. The *MPLAB® XC8 C Compiler User's Guide for PIC® MCU* no longer includes the documentation for these standard functions. This information can now be found in the *Microchip Unified Standard Library Reference Guide*, included with this release.

## 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 analyzes 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.

## Version 2.31

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

## 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.

## 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.

## 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 `r`*n* 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`.

## Version 2.19 (Functional Safety Release)

None.

## Version 2.10

**Code Coverage** This release includes a code coverage feature that facilitates analyzis 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 as arguments to 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.

## Version 2.05

**More bits for your buck** The macOS 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 macOS.

**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.

## 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 minimize 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:

| Functions | |
|---|---|
| `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:

| Functions | |
|---|---|
| `tgamma` | `strftime` |
| `tgammaf` | |
| `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.

### Versions 1.XX

For information regarding changes made to version 1.xx compilers, see the release notes of any compiler version prior to v3.00.

## 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.

### Version 3.00

**Improved messaging (XC8-3131)** The Clang front end was reporting fatal errors as `std::logic_error`. It now reports more descriptive fatal errors.

**Clang updated (XC8-3016)** The Clang compiler front end has been updated from version v4.0.1 to v18.1.8. This update includes changes to diagnostics and warnings. See Clang updated (XC8-3016) in the *What's New* section for information on what has changed.

**PIC18 pointer format (XC8-2990)** The format of the address stored in 24-bit wide mixed-target pointers has changed for PIC18 projects. Previously, bit #21 was used to determine the target address space; now bit #23 indicates if the target is in data or program memory. Any code that makes an assumption as to the numerical value of an address in such a pointer will need to be reviewed.

**HI-TECH C discontinued (XC8-2989, XC8-3437, XC8-3438, XC8-3422)** Support for the HI-TECH C language, libraries, and compiler drivers has been discontinued. Compiling for C90 or ANSI C is no longer conflated with HI-TECH C eccentricities and will be based on a more conventional C standard. Typically, C90 will now impose fewer restrictions compared to the old HI-TECH C language. Note the following changes.

- The architecture-specific drivers, `picc` and `picc18`, have been renamed to `driver` and `driver18`, respectively, and the middle driver, `xc8`, has been folded into the architecture-specific drivers. In this manner, these old HI-TECH legacy drivers will cease to be available.
- The preprocessing and parsing (into p-code) of C90 source code by the `p1` and `cpp` applications is now performed by Clang. There might be differences in how C90 programs are preprocessed; for example, C++ style comments (`//`), which are not part of the C90 standard, are not interpreted in every circumstance when using Clang.
- All `float`, `double`, and `long double` floating-point types use the IEEE753 32-bit format. The 24-bit floating point type is no longer supported.
- An empty function prototype used in a declaration is now interpreted to mean that no function parameter information has been provided, rather than meaning the function has no parameters, as was inconsistently assumed by the old p1 application.
- The `_Bool` type can now be used with C90 programs.
- The `long long` types are now 64-bit C99 standard integer types when building for PIC18 and Enhanced Mid-range devices and the C89/C90 standard. All other devices will continue to have `long long` types restricted to 32 bits wide.
- The `__at()` construct can now be placed before or after the variable identifier in a definition regardless of the C standard selected.
- Projects that need to conform to the C89/C90 standard may use dynamic memory allocation via the standard library memory management functions. Any restrictions that were imposed by the target device will, however, still remain.
- Case ranges (*`lo_value`* `...` *`hi_value`*`:`) are not a feature of the C90 or C99 standards and are no longer supported.
- Objects defined with anonymous structures or unions can now be initialized regardless of which C standard has been selected.
- Floating-point constants can now be expressed as hexadecimal digits and a binary exponent, initiated with either p or P, regardless of which C standard has been selected.
- The integer size used by all preprocessor arithmetic is now 64-bits, regardless of device or which C standard has been selected.
- The `HI_TECH_C` has been removed; the `__CLANG__` macro is always set; the `__STDC_VERSION__` is new and is defined for both C90 and C99.
- The use of unnamed function parameters, which was a C23 language extension supported by the old `p1` application, is not supported by Clang. Empty initializers and binary literal integers, both C23 language extensions, continue to be supported.
- The `-fshort-double`, `-fshort-float`, and `-mc90lib` options have no effect other than to issue a warning.

- The `-msmart-io-format` and `-MF` options are now available to projects built to the C89/C90 language standard.
- Use of the `__CONFIG`, `__IDLOC`, `__IDLOC7`, and `__PROG_CONFIG` macros will trigger a warning.
- The following legacy headers and include files will no longer be supplied with the compiler or DFPs: `pic/include/p18cxxx.h`, `pic/include/pic_cas_chip_select.h` and `pic/include/proc/as<device_name>.h`. The `pic/include/legacy/` directory has also been removed. The following headers are still shipped with the compiler, but including them will trigger a warning stating that they are deprecated: `pic/include/aspic.h`, `pic/include/caspic.h`,`pic/include/htc.h`, and `pic/include/__at.h`.

**Wider pointer scope (XC8-2870)** The compiler now supports the ability for pointers to point to and access all regions of the PIC18's program memory address space. This includes the Device Information Area (DIA).

## Version 2.50

**Pointer refactoring (XC8-3202)** Much of the compiler source code responsible for pointer analysis and memory allocation has been refactored so that the PIC18 and baseline/mid-range code generators now share a common, revised pointer-processing code base. This change should not impact how projects are built.

## Version 2.49 (Functional Safety Release)

**New message emitted (XC8-3420)** In situations where the compiler has detected a pointer to an `__eeprom` type that has been assigned the address of an object that is not qualified `__eeprom`, the error previously emitted by the compiler has been changed to a different advisory message.

## Version 2.46

**Message type changed (XC8-3257, 3252)** The warning messages issued when `#pragma config` configuration bits settings enabled storage area flash (1604) or enabled block table read protection (1605) have be converted to advisory messages. If the configuration bit settings associated with these warnings were intentional, there was no way to suppress these warnings, and in functional safety applications, where warnings might be converted to errors, this would have prevented the project from being built. Continue to monitor build output for these advisories.

**Calling a different Hexmate (XC8-3237)** The compiler will now call the shipped Hexmate utility when required, rather than rely on internal Hexmate functionality. This change will be largely transparent, other than a Hexmate command line appearing in verbose build outputs.

## Version 2.45

**Removal of devices (DTD-59)** The following parts are no longer supported by the compiler, and data for them will not appear in any subsequent DFPs: 16C54A, 16CR57B, and 16CR58A.

**Removal of devices (DEVXML-4227)** The following parts were never released and preliminary compiler support for them has now been retracted: MCP19120, MCP19121, MCP19126, MCP19128, MCP19132, MCP19133, and MCP19625.

**Changed NOPs (XC8-3114)** To accommodate those devices which require a hand-written `NOP` instruction encoded as 0x0000 in errata workarounds, the C built-in function `__nop()`

and the assembler instruction `NOP` will now use encoding 0x0000. Previously, these used the opcode 0xF000. The encoding for any `nop` instructions added by a compiler-applied errata workarounds remains unchanged; however, new macros have been provided to allow the desired encoding of the `nop` instruction to be conveyed to the compiler. These are:

`__nopf000()` and `nopf000` to ensure an encoding of 0xF000
`__nop0000()` and `nop0000` to ensure an encoding of 0x0000; and
`__nopffff()` and `nopffff` to ensure an encoding of 0xFFFF

**Deprecation of HI-TECH support (XC8-2958)** A future version of MPLAB XC8 will discontinue support for legacy HI-TECH C90 source code and libraries. Beginning with version 2.45 of the compiler, advisory messages indicating that HI-TECH C syntax, libraries, and drivers are deprecated will be issued when appropriate, so that programmers can consider code migration to the C99-compliant syntax and the Microchip Unified Standard Libraries. This message will be issued whenever the `xc8`, `picc`, or `picc18` legacy drivers are directly executed. Note that this future change will not affect standard C90-compliant code; only legacy HI-TECH code syntax and libraries will no longer be supported. To prepare for this future change, effective immediately, projects targeting Baseline and Mid-range devices shall by default use the C99-compliant Microchip Unified Standard Library when compiling for C99. Users may revert to the legacy HI-TECH C90 library by using the `-mc90lib` option; however, this will trigger the advisory message mentioned above. If code is ported from the legacy C90 HI-TECH language to C99, the following changes are relevant.

- Any components of the HI-TECH library (e.g. functions, types, etc.) that are not defined by the C99 standard will not be available, including the following functions: `cgets()`, `eval_poly()`, `cputs()`, `ftoa()`, `getche()`, `isascii()`, `isdig()`, `itoa()`, `ltoa()`, `strichr()`, `stricmp()`, `strnicmp()`, `strrichr()`, `stristr()`, `toascii()`, `udiv()`, `uldiv()`, `utoa()`, and `xtoi()`.
- The equivalent Microchip Unified Standard Library versions of HI-TECH C90 library functions are typically more feature rich and more accurate, hence they often use more data and code memory. This might mean, particularly for Baseline devices, that use of the functions will exceed the device memory.
- The `getch()` and `getchar()` functions are both supported by the Microchip Unified Standard Library; however, the return type for these are an `int`, compared to the `char` return type used with the HI-TECH C90 library.
- As the Microchip Unified Standard Library supports dynamic memory allocation, Baseline and Mid-range devices using this library can define a heap and use functions like `malloc()` and `free()`. When used, the maximum heap size for these targets will be restricted to the largest available space in one bank. Use of dynamic memory allocation is not recommended, particularly for small-memory devices.
- The internal buffer that temporarily holds the result of each format conversion before printing has been reduced from 80 to 16 characters for formatted output functions, like `printf()`, when they are used with Baseline targets.
- The internal buffer that temporarily holds an individual sequence of characters related to a single conversion has been reduced from 80 to 16 characters for formatted input functions, like `scanf()`, when they are used with Baseline targets.
- Baseline devices will not be able to make use of the `struct tm` type, defined in `<time.h>`.
- The size of `intptr_t` and `uintrptr_t` will scale, according to the target device.
- The `atexit()` function will support the registration of only 3 functions, not the 32 functions specified by the C99 standard.

- Baseline and Mid-range devices do not support 64-bit integral types. The `intmax_t` and `uintmax_t` types, therefore, are 32-bit integral types and any 64-bit type (for example `uint64_t`) or macro (for example `INT_FAST64_MAX`) or function (for example `lldiv()`) will not be available.

## Version 2.41

**Less cryptic error messages (XC8-3056)** The compiler will now issue different error messages when encountering some situations that were previously described with messages, such as `Looping around allocGlobals`, which have no meaning to the user. The new messages indicates that `XC8 has encountered an internal error and must quit` and indicates what needs to be done.

**Misleading diagnostic messages (XC8-2950)** In compiler output diagnostic messages that were not associated with any particular file, a misleading line number might have been shown. Line number information is no longer shown in these situations.

**Inaccurate fma functions removed (XC8-2913)** The C99 standard library `fma()`-family functions (`<math.h>`) did not compute a multiply-add with infinite precision to a single rounding, but instead accumulated rounding errors with each operation. These functions have been removed from the supplied library.

**Generation of Hex files (XC8-2885)** The generation of Hex files from linked object files is now performed by functionality that is part of Hexmate. This duty was previously performed by the Objtohex utility, which is no longer shipped with the compiler. This change should not impact any projects, as the Objtohex utility was only ever indirectly called by the compiler driver.

**More function splitting (XC8-2884)** If the code generated for a function is larger than the size of a program memory page, the compiler could use a feature to allow it to span multiple pages. Previously, this feature was tied to the optimization level, even though it is more accurately a work-around for limitations in the hardware. The compiler now uses the function-splitting feature whenever possible, even when optimizations are not enabled. This will reduce the incidence of `can't find space` errors relating to program memory in some circumstances.

**Different undefined symbol messages (XC8-2206)** When there were many undefined symbols reported by the linker, they were displayed in a message on one line in a way that was difficult to read. This has been updated so that each undefined symbol is reported in a separate error message.

## Version 2.40

**Better warnings with DFP incompatibles (XC8-2854)** The compiler issued warning (1020) regarding unknown attributes when using a device family pack (DFP) that was incompatible with (typically, more recent than) the currently selected compiler. A new message, stating, `The selected Device Family Pack (DFP) contains features not implemented in this compiler version; consider using an alternate DFP or a more recent compiler release`, will now be issued.

**Code coverage macro availability (XC8-2804)** The compiler previously set the `__CODECOV` macro to `__CC_NONE` when the code coverage feature was disabled. Now, this macro is not defined at all when code coverage is disabled. Other states of this macro remain the same.

**Removal of math functions (XC8-2757)** The `exp2()`, `exp2f()`, and `exp2l()` family of library functions have been removed from the standard C library shipped with the compiler. These functions did not always give correct results.

**Warnings with direct header inclusion (XC8-2135)** The compiler was not correctly warning if a device-specific header file, for example `<pic18f8722.h>`, was being directly included into source code. A warning is now issued in such circumstances. If the compiler reports this warning in your projects, ensure you include the top-level header file `<xc.h>` instead.

## Version 2.39 (Functional Safety Release)

None.

## Version 2.36

None.

## Version 2.35

**Data memory reservation (XC8-2711)** Some PIC18F'Q devices have data banks that can be dedicated to a particular function (e.g. page reads and writes of program memory, sending and receiving CAN messages). Previously, such banks might have been included with general-purpose data memory and been available to the compiler for allocation of ordinary objects. These banks are now reserved and no longer available to the compiler. If desired, you may create absolute-addressed variables (using the `__at()` specifier) in these reserved memory regions.

**errno not being set (XC8-2682)** Several `<math.h>` functions (for example `acos()`) were not setting the `errno` object when required, such as on a domain error. Code in your project that made checks on this object might now behave differently. The functions now act in accordance with the C99 language standard.

**Removal of non-standard functions (XC8-2591)** The `finite`, `finitef`, `toascii`, and `isascii` functions are extraneous to the C99 language standard and have been removed from the compiler distribution. These functions were never documented. In addition, the non-standard `<conio.h>` header has been removed.

**Handling of string-to bases (XC8-2420)** To ensure consistency with other XC compilers, the XC8 string-to functions, like `strtol()` etc., will no longer attempt to convert an input string if the base specified is larger than 36 and will instead set `errno` to `EINVAL`. The C standard does not specify the behaviour of the functions when this base value is exceeded.

**Hexmate II** The Hexmate application shipped with the compiler is now built from a new code base. In addition, parts of the Hexmate code have been incorporated into the compiler drivers directly, meaning that the compiler no longer needs to execute the Hexmate application when building projects. If you run the compiler in verbose mode, a Hexmate application command line, like that shown here:

```
/Applications/microchip/xc8/v2.32/pic/bin/hexmate
@/tmp/hexmate_xcNZDOQKH.cmd [ -W-3
--edf=/Applications/microchip/xc8/v2.32/pic/dat/en_msgs.txt main.hex
-E1 -Omain.hex -logfile=main.hxl -addressing=1
-fill=0xFFFF@0x800:0xFFF -ck=800-FFF@20+0w1g2p0r0s0.1 -break=300000 ]
```

will no longer be seen. The stand-alone Hexmate application is still shipped with the compiler and if required can be run explicitly as it had in the past, allowing full access to its options or its use without a compiler. The log file output, when requested, shows more complete information relating to the tasks Hexmate performed.

**Microchip Unified Standard Libraries** All MPLAB XC compilers will share a Microchip Unified Standard Library. The MPLAB XC8 compiler has shipped with this library for several versions, but with this release, the available functions have been expanded and the operation of some functions has been standardized. The *MPLAB® XC8 C Compiler User's Guide for PIC® MCU* no long includes the documentation for these standard functions. This information can now be found in the *Microchip Unified Standard Library Reference Guide.*

## Version 2.32

None.

## Version 2.31

None.

## 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.

## Version 2.29 (Functional Safety Release)

None.

## 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.

## Version 2.19 (Functional Safety Release)

None.

## 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.

## 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 recognized. As an alternative, use the macros that define the optimization level, such as `__OPTIM_LEVEL`.

## 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 standardized 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.

### Versions 1.XX

For information regarding migration issues relating to version 1.xx compilers, see the release notes of any compiler version prior to v3.00.

## 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.

Note that some device-specific issues are corrected in the Device Family Pack (DFP) associated with the device. See the MPLAB Pack Manager for information on changes made to DFPs and to download the latest packs.

## Version 3.00

**(XC8-3576)** When using the `-mcodeoffset` option to link PIC18 applications to addresses above 0xFFFF and those applications defined less than 256 bytes of `const` data, this data might have been incorrectly accessed.

**Wrong access bank address (XC8-3533)** When assigning the address of an SFR in the access bank to a pointer, the compiler might have assumed that the SFR was in bank 0, resulting in the wrong address being assigned.

**Can't find far space (XC8-3522)** A `can't find space` linker error might have occurred for some PIC18 projects that used the `__far` qualifier, even though the `-maddrqual` option was set to ignore the qualifier.

**No code offset for Baselines (XC8-3521)** With Baseline devices that did not have interrupts, the `-mcodeoffset` option did not shift the `reset_vec` psect (which contains the code placed at the reset vector) as expected. If the targeted Baseline device did implement interrupts, the option correctly shifted the `intentry` psect (which is used to hold the code executed at the interrupt vector). All relevant psects are now shifted if this option is used; however, note that the compiler will not generate the runtime startup code to initialize the oscillator from the calibration constant in such a case. It is expected that the code linked to the reset location (address 0x0) will provide this functionality. This option worked as expected for all other device families.

**Too much warning promotion (XC8-3510)** If the compiler was invoked with a `-Werror=`*`num`* option, the Cromwell application promoted all warnings it issued to errors, regardless of the warning number specified in the option.

**Bad __interrupt() arguments accepted (XC8-3493)** The compiler was allowing any prefix and/or suffix with the `low_priority` and `high_priority` interrupt arguments, for example `__interrupt(AAhigh_priority)` would have been an acceptable substitute for `__interrupt(high_priority)`. The compiler now strictly requires `high_priority`, `__high_priority`, `low_priority`, or `__low_priority` keywords.

**Bad addresses accepted (XC8-3492)** The compiler was interpreting address arguments to the `__at()` specifier as signed and allowing negative addresses to be passed. The compiler now expects this argument to be unsigned and will produce an error if a negative value is given.

**Bogus error with const parameters (XC8-3490)** When projects defined function parameters as `const` and the `-maddrqual` option was set to `require`, the compiler might have issued can't find space memory error.

**Data corruption with indirect access (XC8-3482)** For projects targeting PIC18 devices, assignments that dereferenced a `void` pointer cast to be a pointer to a 24-bit integer type might have resulted in corrupted data.

**Cast of shift operand ignored (XC8-3423)** In some circumstances, when the left-hand operand of a bit-wise right shift operator was cast to a larger type in order to avoid an undefined result, the compiler would ignore the conversion and perform the shift on the smaller type. An `undefined shift` warning was issued; however, the generated code could potentially produce the undefined result. The compiler now correctly performs the type conversion before shifting.

**Text data clash error (XC8-3421)** Building XC8 or PIC-AS projects that linked in a library containing more than one object-code module that defined text-based psects with the same name would incorrectly issue a `Clash in psect text data` error.

**Compiler crash with nested compound literal (XC8-3350)** The Clang application might have crashed when encountering code such as the following initialization:

```
struct S {int x; int y;};
struct T { struct S a; };
struct T t = (struct T) { .a = (struct S){} };
```

**Failure to open file (XC8-3294)** On rare occasions, the compiler was unable to rename some temporary files, resulting is errors such as `unable to rename temporary` and `(141) can't open input file`. The Clang application has been updated and includes fixes that address issues such as this.

**Can't initialize structures (XC8-3280)** The compiler was issuing a `Can't generate code` error for the initialization of some compound structures specified as `const`.

**Bad handling of large enumerated values (XC8-3233)** Clang was incorrectly assuming that enumerated values larger than an `int` would be promoted to a `long` type and evaluated some expressions (such as `sizeof()`) based on that assumption. If large enumerated values are encountered, they are now truncated to an `int` type and a warning will be issued by the code generator.

**Enumerated values not promoted (XC8-2912)** The compiler was not applying the C language's default argument promotions to enumerated values. Variadic functions, such as `printf()`, reading arguments that were small enumerated values might have done so in error.

**Printing long doubles (XC8-2582)** The `printf()`-family length modifier `L` applied to a floating-point conversion specifier was ignored by the compiler. It can now be used to allow `long double` types to be printed.

**Crash when discarding output (XC8-1858)** When the output of the compiler was specified as `/dev/null`, the compiler issued a generic error message and crashed. Now a dedicated error message is issued and the compiler will exit gracefully.

## Version 2.50

**Bad inline assembly translations (XC8-3432)** When targeting enhanced mid-range devices, the use of `bra` instructions in inline assembly might have resulted in a linker fixup error after the instruction was transformed. Such transformations of inline assembly are no longer performed.

**Read failure of buffers (XC8-3413)** The compiler does not include CAN and other special buffers in the GPR (General Purpose Register) data RAM available for the allocation of objects. However, if a pointer is assigned an integer value, assumed to be the address of such a buffer, dereferencing that pointer might read from program memory, not data memory. This issue might have been encountered when using `memcpy()`, for example, and it only affects PIC18 projects.

**Calls to NULL pointers (XC8-3407)** The improved pointer analysis performed by this release allows the compiler to more accurately determine when function pointers might hold a `NULL` value and correctly refrain from generating code for function calls made using such pointers. However, in very specific circumstances where the compiler previously issued the advisory `(2098) indirect function call via a NULL pointer ignored` and optimized away the function call, the message will no longer be emitted and the call will be encoded. This might mean a slight increase in code size, but with more reliable operation.

**Function duplication (XC8-3408)** The improved pointer analysis performed by this release might result in the compiler now duplicating some compiled-stack functions and issuing the advisory `(1510) non-reentrant function * appears in multiple call graphs and has been duplicated by the compiler` where it previously did not. In some rare instances, this action might be redundant; however, the compiler will err on the side of caution.

**Bad pointer comparison (XC8-3386)** Some projects targeting PIC18 devices might have produced "can't generate code" errors for expressions comparing pointers. In rarer instances, such comparisons might have compiled but produced code that corrupted memory.

**Data corruption when shifting (XC8-3371)** For projects targeting enhanced mid-range devices, the compiler might have omitted a bank selection instruction when building expressions that shifted SFRs operands. This might have caused data corruption of an object at the same address offset in a different data bank.

**Crash with const members (XC8-3364)** The use of `const` with structure members where the structure has been initialized via compound literal expressions might have caused the Clang frontend to crash.

**Omission of indirect call (XC8-3356, XC8-3155)** The compiler was omitting some calls made indirectly through a function pointer. The message `indirect function call via a NULL pointer ignored` was emitted in this situation. The compiler is now correctly generating the function call graph, allowing code to be generated for all valid calls.

**Crash on missing argument (XC8-3272)** Where the source code contained a function that had been prototyped with at least one parameter, and the function was called according to that prototype but then later called without any arguments, the compiler might have issued an internal fatal error.

**Looping in allocation (XC8-3183)** In rare circumstances, enhanced mid-range projects that indirectly accessed an object larger than a bank might have triggered an `Internal error - Looping around allocGlobals()` error when being built.

**Bad assignment of shifted value (XC8-3172)** In PIC18 projects that contained an assignment of a multiplication by 2 (which the compiler optimizer converted to a left bitwise shift), and if the assignment was to an object larger than a bank, the result might have been written to the wrong location.

**Memory error when const caching (XC8-2983)** When building large projects that contained `const` arrays of pointers to `const` and the cache const feature was active, the compiler might have exited with a `can't find space` memory error.

**Bogus warning when reserving memory (XC8-2482)** When an entire region of memory (such as a data bank) was reserved (e.g. using the `-mram` option) and an externally defined object was located in that reserved region, the compiler was always issuing the warning, `2028 external declaration for identifier "*" doesn't indicate storage location`, despite the usage of the appropriate storage specifier.

**More allocation looping (XC8-2324, XC8-1964)** In projects using enhanced mid-range or PIC18 that called functions with a pointer as a parameter, the code generator might have encountered an issue and exited, displaying an `Internal error - Looping around allocGlobals()` error message.

**Crash on recursion (XC8-2309)** Programs that contained recursion and used the compiled stack might have caused the code-generator to crash.

**Viewing pointer targets in the IDE (XC8-1758)** When debugging enhanced mid-range projects that included structures with members that were pointers, the MPLAB X IDE might have shown incorrect values for the member pointer targets in its debugger views.

**Corrupted return value (XC8-1646)** In projects targeting enhanced mid-range devices that included a function that took a single byte-sized parameter and returned a type larger than a byte, insufficient memory might not have been reserved for the return value, which could then have become corrupted.

**Yet more allocation looping (XC8-1607)** PIC18 projects that made heavy use of pointers and that enabled the local optimization feature might have triggered an `Internal error`

- `Looping around allocGlobals()` error when being built.

**Pointer made too small (XC8-1450, XC8-1287)** Projects targeting enhanced mid-range devices with more than 16 banks of RAM and that used structure members that were pointers to other structures linked at an address requiring a two-byte pointer, might have failed to run correctly due to the structure pointer being made one byte in size.

**Error with interrupt pragma (XC8-1303)** Projects that targeted enhanced mid-range devices and that made use of the `interrupt_level` pragma might have triggered an internal error when being built, indicating `Internal error - no stack allocated to function`.

**Call expression as argument (XC8-1055)** Projects targeting enhanced mid-range devices might have encountered issues when being built. Specifically, if a project contained function calls with arguments that were themselves function calls or calls to an internal library routine, an `Internal error - no stack allocated to function` might have been triggered.

## Version 2.49 (Functional Safety Release)

None.

## Version 2.46

**Removed instructions acting on BSR (XC8-3253)** For projects targeting enhanced mid-range devices that had assembly optimizations enabled, where there were consecutive instructions in hand-written assembly modules or inline assembly that had BSR as an operand, the assembler optimizer might have incorrectly removed one or more of these instructions.

**Tris oversight (XC8-3216)** The internal compiler assembler and PIC-AS incorrectly reported that the `tris` instruction was illegal for enhanced mid-range targets. This instruction is now permitted with these devices.

**Temps moved the long way (XC8-3217)** The `lowdata` psect flag was missing from the compiler-supplied `temp` psect when building for PIC18 devices. This meant that any `movff` pseudo instructions that operated on any temp objects (`btemp` symbol) might have been transformed into the long-form `movffl` instruction, unnecessarily increasing code size.

**Headers removed from DFP (XC8-3167)** The `<__at.h>`, `<builtins.h>`, `<cci.h>`, `<errata.h>`, `<htc.h>`, `<language_support.h>`, `<xc.h>`, and `<xc.inc>` header files are no longer included in Device Family Packs (DFPs). These headers are not relevant at the DFP level and are more specific to a compiler version. This change will not affect most projects; however, it might correct situations where building with a new compiler but an old DFP prevented access to new compiler features.

## Version 2.45

**Roaming license failure (XCLM-235)** Roamed licenses failed to work correctly on Linux platforms using glibc versions later than 2.28.

**Fixup on GOTOs (XC8-3147)** The PIC Assembler was not always processing the operand to the `GOTO` instruction correctly. This might have resulted in hand-written assembly code for Mid-range and Baseline devices that used this instruction triggering fixup errors. One consequence of the fix for this issue is that instructions like `GOTO $+`*offset* will trigger a fixup error if this instruction is located in any but the lowest page of memory. The operand for this instruction would need to be masked to avoid the error. Fixup errors like this can also be avoided entirely in hand-written assembly code by using the `-Wl,--fixupoverflow` option.

**Crash when generating SHA (XC8-3146)** When the compiler was instructed to write summaries to a file and compute a SHA hash of the generated HEX-file, it might have resulted in the driver crashing. Now, the compiler will write any computed SHA1 or SHA256 to a separate file (using `.sha1` or `.sha256` extensions respectively).

**Undefined symbol on indirect call (XC8-3124, XC8-3085)** In certain circumstances when using the compiled stack, making an indirect function call using an unassigned function pointer that is a member of a structure, might have caused a compiler error for an undefined and unrecognizable symbol.

**Hexmate range restriction gives error (XC8-3123)** In some cases, an input HEX file range restriction specified with unqualified hexadecimal addresses starting at zero (for example, `r0-b,input.hex`) could have been misinterpreted by Hexmate, resulting in an error, `cannot determine start address`.

**Limited irq arguments (XC8-3122)** The compiler did not allow all forms of C-style integer radices for values given to the `__irq()` argument to `__interrupt()`, for example 31 was accepted, but 0x1F was not. All usual radices can now be used for this argument.

**Crash on unterminated macro (XC8-3113)** The PIC Assembler might have exited with an out of memory error if processing assembly source that defined a macro without an `ENDM` end macro directive.

**Indirect calls ignored (XC8-3096)** For code that involved indirect calls using function pointers defined in structures, the compiler might have issued an `indirect function call via a NULL pointer ignored` message and erroneously omitted code to perform the function call.

**Bogus warning with library code (XC8-3094)** In certain circumstances, the compiler unnecessarily warned about indirect function calls via a NULL pointer for some standard C library code, such as `atexit()`.

**Error with byte parameters (XC8-3084)** When building PIC18 projects that defined a function with a byte-sized parameter, compilation might have stopped with an error 5 (or`(703) bad GN` error).

**Assembler errors badly handled (XC8-3082)** When converting the object code of a program to a HEX file and there was a clash in text data, the resulting error messages from the PIC Assembler might have been incorrect and the driver might not have halted execution of the toolchain. The errors have been improved to more accurately describe the cause of the clash, and the assembler will flag that it has failed.

**Error when excluding startup files (XC8-3065)** In special cases where no startup files were requested (`-Wl,-nostartfiles`), some linker options were not being specified, resulting in `psect "*" was never defined, or is local` link errors, for example.

**Trivial Clang inconsistencies (XC8-3058)** The Clang (C99 parser) generated printf/scanf usage pragmas in a non-deterministic order. This did not adversely affect the generated code in any way, but might have resulted in inconsistencies across builds.

**Errors when restricting RAM (XC8-3052)** When restricting the amount of RAM available for PIC18 devices that implement special RAM memory at high addresses, such as that memory used for CAN, the compiler might have issued undefined symbol errors for the symbols `__smallconst` and/or `__mediumconst`.

**Unaccepted interrupt base addresses (XC8-3051)** The compiler issued the error `(1318) invalid argument to attribute/specifier "__interrupt"` when using a high value of `base` address specified in the `__interrupt()` construct, which was being treated as a signed value. These base addresses are now permitted.

**Incomplete VA functions cause error (XC8-3028)** Functions with variable argument lists that did not use `va_start()` in the body of the function might have caused the compiler to exit with a `looping around allocGlobals()` error.

**Cryptic message (XC8-2922)** The Clang frontend issued a cryptic `default: Empty` message when encountering a null statement located outside a function definition. This message has been suppressed.

**Clang crash (XC8-2883)** The Clang frontend might have crashed when processing expressions where a string was placed in brackets, for example, `char c[100] = ("Hello World");`

**Advisory when option is repeated (XC8-2705)** When more than one instance of some options were issued, the compiler either generated an error or silently ignored all but the last instance. For relevant options, the compiler will now issue an advisory message (2101), alerting you to the fact that the option has been specified multiple times. Note that some options, like the `-mreserve` option, were intended to be used more than once. Their repeated use does not present conflicting information and will not trigger such an advisory.

**Array fails to build (XC8-2445)** A `const` array with exactly 0x8000 elements failed to build when using C99, yielding the error, `(740) array dimension must be larger than zero`.

**Erroneous pointer error (XC8-2300)** Mid-range projects that defined two structures that differed only in name and with one structure qualified `__eeprom` might have triggered error (1402) `a pointer to eeprom cannot also point to other data types` if a pointer was assigned the address of the structure in EEPROM.

**Case ranges in C99 (XC8-1855)** In previous compiler versions, the C99 Clang front end silently ignored `case` ranges in a `switch()` statement. Clang now accepts and generates code for case ranges. If the CCI is enabled, a warning is issued, although valid code will be produced.

**Truncated IDLOCx values (XC8-1570)** For some PIC18 devices, the `IDLOC`*x* values specified using the `#pragma config` directive were being truncated from 8 to 4 bits wide. The full `IDLOC`*x* value will now be programmed, provided you are using the latest DFP.

**Incorrect conflicting structure definition error (XC8-1407, XC8-3097)** Code that defined a structure based on another structure type that was incomplete due to it being located in a different module, might have caused the compiler to exit with a conflicting declarations error (1098).

## Version 2.41

**Dongle issues on Ventura (XC8-3088)** Dongles used to license the compiler might not have been properly read on macOS Ventura hosts, resulting in licensing failures. Changes to the XCLM license manager correct this issue.

**Configuration programming of new devices (XC8-3072)** The configuration registers for new devices that had these registers located in Data Flash Memory were programmed in such a way that an extra byte was located outside the configuration memory area. This extra byte will no longer be programmed.

**Linker error with unused absolute functions (XC8-3068)** The compiler might have issued a linker error indicating that the psect associated with an absolute function was never defined when that absolute function was only called from a function that itself was never used. The names of such psects consist of the function's assembly-domain name sufficed with `_text`, so for example an absolute function `rv()` will be placed in a psect called `_rv_text`.

**Bad memmove() code (XC8-3067)** For projects targeting PIC18 devices, invalid pointer comparisons between addresses in the `memmove()` library function might have ignored overlapping source/destination array addresses. This code might additionally have triggered a `can't generate code` error. The `memmove()` library routine has been updated to ensure that memory is correctly copied for PIC18 devices.

**Linker crash (XC8-3042)** The linker might have crashed when the object file it was linking contained linker classes that had no name.

**Bad manual relocation of software stack (XC8-3033)** The runtime startup code generated for PIC18 programs using a software stack initialized the mainline-code stack pointer with a constant literal value (calculated by the compiler) instead of a relocatable symbol associated with the `stack` psect. As a result, any manual relocation of the `stack` psect would have reserved memory for the stack in the desired location, but the stack-pointer would not have been initialized to point to this memory. The runtime startup code has been modified so that any manual relocation of the `stack` psect is correctly honoured. Note that in normal use, where the `stack` psect was linked using the default linker options, there was no code failure. Additionally, the stack pointer was correctly setup using the appropriate symbols when entering interrupt functions.

**Undefined symbols when shrouded (XC8-3032, XC8-2880)** When using the `-mshroud` option, the compiler might have failed to build the project, issuing 'undefined symbol' errors. This option can now be used for all projects.

**Wrong assembly debug lines (XC8-3027)** The MPLAB X IDE would indicate the incorrect source line when stepping through code or stopping at breakpoints set in code contained in absolute, (`abs` flag) overlaid (`ovrld` flag) psects.

**Temp files not unique (XC8-3001)** Windows versions of the compiler might have created and tried to simultaneously use temporary files with the same name but that were meant to hold different information. This would have caused the build to fail, possibly with the error `(398) output file can't be also an input file`.

**Watching what? (XC8-2980, XC8-2371)** The type information of `const`-qualified objects was being incorrectly indicated in the MPLAB X IDE watch window. This would have impeded project debugging. The code being generated was not adversely affected.

**Better clean up (XC8-2962)** The compiler might have left behind temporary directories and files after executing. This behavior was dependent on the exact host operating system under which the compiler was being executed.

**Assembly EE data misplaced (XC8-2953)** The `edata` psect definition provided in the device-specific assembly include files for PIC18 devices incorrectly specified a `delta` value of 2 instead of 1. This would have resulted in any assembly output placed in this psect being located outside the valid address range of EEPROM data.

**Truncated vsprintf() output (XC8-2952)** In certain circumstances, the C99 `vsprintf()` library function would truncate formatted output in the supplied buffer to an empty string.

**Misleading diagnostic messages (XC8-2950)** In compiler output diagnostic messages that were not associated with any particular file, a misleading line number might have been shown. Line number information is no longer shown in these situations.

**Bad access of absolute booleans (XC8-2930)** Access of a `_Bool` variable positioned at an absolute address in PIC18 projects might have been to the wrong address, resulting in potential code failure.

**Missing errata indicators (XC8-2929)** The compiler-supplied header `<errata.h>` was missing definitions for the following PIC18 errata: `ERRATA_BRKNOP2`, `ERRATA_NVMREG`, and `ERRATA_BSR63`; and the following mid-range PIC errata: `ERRATA_CLOCKSW` and `ERRATA_BRANCH`. These are preprocessor macros that can be checked to see if an errata is in effect. The actual code work arounds were implemented correctly by the compiler.

**Warnings in library code (XC8-2926)** When building C99 projects, the compiler parser might have issued warnings relating to compiler library code. These warnings have been suppressed.

**Incorrect indication of memory allocation (XC8-2925)** Attempting to allocate `SIZE_MAX` bytes (or a value close to this) of memory using the standard library memory management functions (`malloc()` et al) incorrectly succeeded. It did not take into account that extra bytes were needed in addition to the block requested when using the simple dynamic memory allocation implementation. A `NULL` pointer will now be returned and `errno` set to `ENOMEM` in such situations.

**Bad handling of string conversion (XC8-2921, XC8-2652)** When a 'subject sequence' for conversion by `strtod()` contained what appeared to be a floating-point number in exponential format and there was an unexpected character after the `e`/`E` character, then where `endptr` had been provided, it was assigned an address that had it point to the character after the `e`/`E`, whereas it should have been pointing to the `e`/`E` itself, since that had not been converted. For example, `strtod("100exx", &ep)` should return 100.00 and set `ep` to point to the `"exx"` part of the string, whereas the function was returning the correct value but setting `ep` to point to the `"xx"` part of the string.

**Ignore-assignment flag ignored (XC8-2915)** The return value of C99 standard library formatted input functions such as `scanf()` etc., ignored the assignment-suppression flag (`*`) relating to their return value.

**Wrong values for pointer macros (XC8-2914)** The values of the standard library macros `PTRDIFF_MAX` and `PTRDIFF_MIN` (`<stdint.h>`) were incorrect.

**Inaccurate fma functions removed (XC8-2913)** The C99 standard library `fma()`-family functions (`<math.h>`) did not compute a multiply-add with infinite precision to a single rounding but instead accumulated rounding errors with each operation. These functions have been removed from the supplied library.

**Bad comparisons with NaN (XC8-2911)** The C99 standard library floating-point comparison macros, for example `isgreater()` and `isless()`, with the exception of `isunordered()`, might have given incorrect results if one or both of their arguments were `NAN`.

**Bad infinite() results (XC8-2910)** The C99 standard library floating-point classification function, `isfinite()` produced incorrect results for most arguments.

**Warning on nonexistent construct (XC8-2909)** The compiler was in some circumstances incorrectly warning about a constant conditional branch due to the possible use of "=" instead of "==" where it was obvious that no such usage was present. In these instances, the warning issued by the compiler was in response to an internal representation of the optimized code. These warnings have been suppressed.

**Read failure with no white space (XC8-2906)** If a white-space directive (space character(s)) was present in the format string used with the C99 library `scanf()` functions and no white-space characters were present in the input string, the input would not be correctly read. For example, the expression `sscanf("99", " %d", &val)` was incorrectly returning 0 and not assigning the value 99 to `val`.

**Error requesting byte hashes (XC8-2904, XC8-2905)** When compiling for word-addressed PIC devices (mid-range and baseline) and using the driver's `-mchecksum` option, if the `width` suboption was set to 1, for example:

`-mchecksum=0-7ffd@7ffe,offset=ff,width=1,algorithm=1`
an assertion error might have been triggered.

**Not honouring zero precision (XC8-2903)** The C99 printf-family conversion-specifiers `u`, `d`, `o`, and `x` would print one digit instead of none when zero precision has been specified in the conversion specifier. For example, the expression `sprintf(buf, "%.0u", 0)` should print no characters to the buffer, but was instead printing a `0` character.

**No point in output (XC8-2902)** Some floating-point values printed with the C99 `%a` conversion specifier would omit the decimal point in the output. For example, printing the value 1.0 using the conversion specifier `%.4a` should yield the result `0x1.0000p+0` but was instead producing `0x10000p+0`.

**References to <strings.h> removed (XC8-2901)** The `<strings.h>` header is a non-standard header that is not supported by XC8. Residual references to this header have been removed from the C99 standard library.

**Bad assignment of NULL (XC8-2895)** When building PIC18 projects, assigning `NULL` to a function pointer might have assigned something other than zero in some instances.

**Syntax error with NULL comparison (XC8-2882)** The compiler might have generated a syntax error for expressions that compared a function pointer with the `NULL` value.

**Incorrect integer addition (XC8-2878)** When baseline, mid-range or enhanced mid-range projects included an expression with the form `A = B + C`, where `A` was a 24- or 32-bit integer, and `B` and `C` were integers, for certain values of `B` and `C`, the result assigned to `A` might have been incorrect when compiled at optimization levels 0, 1 or 2.

**Incorrectly formatted zero output (XC8-2703)** The C99 standard library formatted printing functions attached a hexadecimal prefix (`0x` or `0X`) when the `#` flag was used with hexadecimal format specifiers (`x` or `X`) and when printing a value of zero, for example `%#x` should print `0`, not `0x0`.

**Bad conversion of hexadecimal floats (XC8-2626)** Functions in the `strtof()` family and `scanf()` family always converted a hexadecimal floating-point number missing an exponent part to zero. So for the statement `f = strtof("0x1.1", &endptr);` instead of assigning `f` the value 1.062500, it would assign 0. The floating-point formatted input conversions of `scanf()` were similarly affected.

**Pointer mismatch with structures (XC8-2444)** For enhanced mid-range projects, in some rare circumstances that involved pointers to structure members or member pointers, there may have been a mis-match in pointer sizes resulting in runtime failure.

**Errors with IO helpers (XC8-2164)** When building using the C99 standard, conflicting declaration error messages were emitted when attempting to use the `getch()` and `putch()` IO helper functions. These functions are only implemented as stubs by the compiler, but their prototypes are now included by `<stdio.h>`.

**Response to conflicting static usage (XC8-1961)** Functions that were declared as `static` but that were defined without using the `static` qualifier (external linkage) might have caused an undefined symbol error. The compiler now builds without error such functions, treating them as having internal linkage.

**Undetected undefined symbol (XC8-1938)** Depending on the order of the input C source files, an `undefined symbol` error might or might not be generated for situations where a function was defined as `static` in one source file and another function with the same name was used and declared `extern` in another file. As the `extern` declaration was missing a definition, an undefined symbol error should have been issued at link time.

**Erroneously taking a pee (XC8-1790)** The C99 scanf-family conversion specifiers `A`, `E`, `F`, and `G` would erroneously consume `p` and `P` characters in decimal number sequences present in the input. This issue did not affect the value converted for these specifiers, but subsequent conversions would not process the extra input characters already consumed and might assign incorrect values.

**Special numbers not recognised (XC8-1789)** The C99 `scanf()`-family functions' `A`, `E`, `F`, and `G` conversion specifiers did not recognize strings representing infinity or NaN, for example `"inf"` or `"nan"`. These values are now correctly interpreted.

**Missing character set specifiers (XC8-1785)** The C99 `scanf()`-family functions did not support the `%[`*c*`]` character-set conversion-specifiers, which allowed matches of a nonempty sequence of characters from a set of expected characters. These have now been implemented.

**Error not set for string conversions (XC8-1782)** The C99 `strtod()` family of functions (including `strtof()` and `strtold()`) did not set `errno` to `ERANGE` for values outside the representable range. This action is now performed.

**Bad conversion of hexadecimal floating-point (XC8-1781)** The C99 `scanf()`-family functions' `A`, `E`, `F`, and `G` conversion specifiers were only converting the leading zero of any hexadecimal floating-point constant. This resulted in an assignment of zero, and the remainder of the string from the `x`/`X` character was scanned according to the remainder of format string.

**Indirect calls removed (XC8-1368, 1947, 2658, 2669, 2738, 2782, 2837, 2981)** The compiler was being overly strict when comparing potential function pointer targets with a pointer's defined type. In some situations, insignificant albeit mismatching qualifiers used with function prototypes resulted in the internal assignment of some pointer targets (function addresses) being dismissed. This might have resulted in the compiler treating pointers as though they had no targets and contained `NULL`. In such cases, indirect calls using the pointer might have generated no code. The comparison rules have been refined and indirect calls to functions now take place as expected.

**Can't generate bit operations (XC8-1286)** When building enhanced mid-range projects, the compiler was occasionally unable to generate code that assigned the result of a bitwise operation between 1-bit-wide bitfield operands to a 1-bit bitfield destination.

**Clearing of volatile objects (XC8-1261)** Global variables qualified `volatile` that were written to before being read were not being zeroed by the runtime startup code. Although this did not affect the value read by the program, any side effects resulting from the zeroing performed by the startup code were not realised. This optimization will no longer occur for `volatile` objects.

**ICE support macro clash (XC8-1223)** When using a Microchip ICE debugger, the compiler might have defined macros with names that were in the programmer's name space. This might have caused the build to fail due to a clash in definition.

**Bogus warning on comparison (XC8-1185)** In certain expressions that compared a signed integer with a literal value of zero, the compiler may have incorrectly issued a degenerate unsigned comparison (765) warning. This warning has been suppressed.

## Version 2.40

**Bad printing (XC8-2861)** The compiler failed to detect usage of some C99 printf flags and length modifiers in conversion specifiers when the format string passed to the formatting function was not a string literal and the types of the arguments to be printed were analysed. This might have resulted in incorrectly printed output.

**Bogus warning for absolute far objects (XC8-2839)** The compiler issued a bogus warning, stating that objects were outside available data space, for the definition of `__far`-

qualified objects that were made absolute using the `__at()` construct. The warning will no longer appear.

**Bad access of const array (XC8-2836)** Access of `const`-qualified arrays where the index expression consisted of the subtraction of a literal value from a variable might have accessed data memory instead of program memory.

**Printf-style space flag ignored (XC8-2829)** The space (`' '`) and zero (`'0'`) flags were ignored when used with the `%e`, `%f`, or `%g` specifiers with printf-style IO functions, and space and zero padding was omitted from the output.

**Printf-style space flag ignored (XC8-2828)** The space flag (`' '`) was ignored when used with the `%d` or `%i` specifiers with printf-style IO functions, and space padding was omitted from the output.

**Errors printf hexadecimal floating point (XC8-2827)** Various issues relating to character counting, field widths, space and zero flags were present when using the `%a` or `%A` floating-point specifiers with printf-style IO functions, resulting in incorrect output.

**Bad branch targets silently encoded (XC8-2824)** For enhanced mid-range projects, the linker did not correctly identify when the operand to the `bra` instruction was out of range. This situation was most likely to affect hand-written assembly code. An error is now issued if erroneous branch code is encountered.

**Wrong interrupt entry point (XC8-2823)** For PIC10/12/14/16 based projects, the `intentry` psect, which contains the interrupt entry point, might not have been linked at the appropriate address when the interrupt function was defined in hand-written assembly code.

**Build error on trace macros (XC8-2795)** An undefined symbol error might have resulted when attempting debug builds of enhanced mid-range device projects that used any of the REALICE/ICE4 `__TRACE()` or `__LOG()` macros. These macros may now be used.

**Wrong memory access with PIC18 pointers (XC8-2781)** Pointers in PIC18 projects that were assigned the address of objects in program memory as well as objects in specialty GPR banks (such as CAN or Buffer RAM areas) when dereferenced would incorrectly access those GPR locations from program memory. This issue could affect library functions like `memcpy()`.

**Error with compiled stack directives (XC8-2759)** Use of the `FNARG`, `FNBREAK`, or `FNINDIR` assembler directives in hand-written assembly code incorrectly caused the error `unknown FNREC type` to be issued.

**Bad remainders (XC8-2751)** Certain combinations of floating-point arguments to the `remquof()` and `remainderf()` library functions gave an incorrect result.

**Too many traces (XC8-2741)** Using `TRACE()` or `LOG()` macros in more than one call graph in PIC18 projects resulted in the underlying trace functions being unnecessarily duplicated. This did not affect the program's or debugger's functionality, but did waste program memory.

**Error with structure initialization (XC8-2736)** Initializing a pointer to `uint8_t` as part of a designator-list that assigns a static local object resulted in a post-compilation error `Unable to find referred Decl in SymTab`.

**Quote characters stripped from options (XC8-2708)** The compiler driver was removing quote characters, even if they were escaped, from command line options that might need a quoted string argument, for example, when using the `-D` option. The user's guide has been updated to show how quoted strings can be used with this option.

**Unwarranted conflicting declaration error (XC8-2663)** The compiler might have erroneously produced a `conflicting declaration` error when encountering identical structure declarations in seperate modules and where one of these declarations was still incomplete.

**Indescribable error messages (XC8-2656)** On selecting a new DFP that was released after the compiler version being used, error messages stating that an error/warning as been generated but that no description is available might have been produced. The message description file is now shipped as part of the DFP so that new messages can be emitted in full.

**Can't generate code with structure copy (XC8-2643)** For projects targeting Baseline or (non-enhanced) mid-range devices, some expressions assigning one structure to another might have triggered a, `can't generate code` or `registers unavailable` error message.

**Error with array initialization (XC8-2594, XC8-2595)** The compiler erroneously issued a `too many initializers` error then crashed when encountering an array of structures with static storage duration that was initialized using a initializer list that did not initialize all the elements in the array, for example:

```
static const struct {
    uint8_t id;
    const char * name;
    uint8_t voltage_threshold;
}
struct_params[] = {
    [1] = {4u, "test", 128u}
};
```

**Weird non-digit characters printed (XC8-2586)** When printing certain floating-point powers of 10 with the `%f` specifier and C99 printf-style functions, the value might not have been displayed correctly, using non-digit characters such as a colon (`:`) .

**%F specifier ignored (XC8-2583)** If not used alongside the `%f` specifier, the C99 version of the printf-style conversion specifier `%F` was ignored by the compiler, resulting in output being missing for that specifier.

**Bad structure copy (XC8-2579)** For projects targeting PIC18 devices, code that assigned one structure to another via pointers was copied in an incorrect order when the size of the structure was between 5 and 128 bytes, the source structure was located in program memory, and the destination structure was stored in RAM.

**False cant-find-space error (XC8-2488)** The compiler might have incorrectly reported a `can't find space` error relating to psects destined for common/Access bank memory for project code that defined one or more functions that took a byte-sized argument passed via the W register and where that argument did not need to be stored to memory.

**Checksum option ordering (XC8-2441)** An error was issued when providing a `code` specification to the driver's `-mchecksum` option in any location other than at the end of the option's specification list.

**Crash with array access (XC8-2403)** The compiler might have crashed when encountering code where a `bit` type was being used as an index to an array.

**Warning when using shrouded libraries (XC8-2375)** When building projects that linked libraries (*.a) created using the `-mshroud` option, the warning, `use of the opt control "nolist" is deprecated; use the corresponding directive` was emitted. The compiler front end has been corrected to prevent producing the output that triggered this message.

**Pointer warning incorrectly issued (XC8-2315)** When indirectly writing to an object via a pointer, the compiler may have incorrectly emitted the warning, `pointer used for writes includes read-only target "*"` when there was never any address of a read-only target assigned to the pointer.

**Bad array subtraction (XC8-2286, XC8-2742)** For PIC18 projects, an incorrect result might have occurred in expressions where one element of an array that spanned more than one bank and is subtracted from another element of the same array and where the array indexes were constants.

**Errors when using assert() (XC8-2191)** Use of the C90 implementation of the `assert()` macro in compound statements might have resulted in `inappropriate "else"` errors being erroneously triggered.

**Error on absolute far access (XC8-2063)** The compiler issued can't-generate-code type errors for expressions that accessed `__far`-qualified objects that were made absolute using the `__at()` construct. Such access is now permitted.

**Internal error disabling duplication (XC8-1845)** In programs where the `#pragma interrupt_level` had been used to prevent function duplication, an internal error might have been issued.

**Stricter parameter checking (XC8-1815)** When the number of arguments passed to a function does not match its prototype, the compiler might have output a warning to this effect but might have subsequently output a `looping around allocGlobals` error. Except for externally defined functions, the compiler will now treat a mismatch in argument number as an error and output the message `number of arguments passed to function "%s" does not match function's prototype`.

**Warning with absolute functions (XC8-1808)** When positioning the ___at() construct after a function's name for C99 projects, the compiler may have given the warning, `GCC does not allow 'address' attribute in this position on a function definition [-Wgcc-compat]`. This form of absolute function definition was and remains valid, and it is now accepted without warning.

**Bad code access config words (XC8-1592)** Some expressions in PIC18 projects that involved accessed the configuration words using the TBLPTR registers might have resulted in a `truncation of operand value` warning being issued and incorrect code being generated.

**Warning with variable argument lists (XC8-1535,** XC8-2139) Use of the `va_start()` macro might have incorrectly caused the compiler to emit the warning, `arithmetic on pointer to void yields Undefined Behavior`.

## Version 2.39 (Functional Safety Release)

None.

## Version 2.36

None.

## Version 2.35

**Hexadecimal stack size ignored (XC8-2697)** The driver silently ignored the `-mstack` option when the stack size was specified using the hexadecimal prefix `0x`. Now, the driver will strictly enforce that the number be a decimal format. If it is not, a warning will be issued and the size will default to `auto` for the relevant stack.

**errno not being set (XC8-2682)** Several `<math.h>` functions (for example `acos()`) were not setting the `errno` object when required, such as on a domain error. These functions now set `errno` when required by the C99 language standard.

**Automatic reservations (XC8-2676)** If a size has not been specified for either the reentrant stacks or the heap, the compiler will silently reserve all remaining memory for this purpose. Now an advisory message is emitted to indicate that this has taken place.

**Software stack reservation (XC8-2675)** For projects targeting enhanced mid-range devices, if there are reentrant functions that are called only from an interrupt (not from main-line code), memory will be reserved for the software stack using by main-line code but not for the software stack using by interrupt code.

**Restricted call destinations (XC8-2665)** The PIC Assembler was encoding the `call` instruction with its operand XORed in an expression that ensured it could not be used to call a destination in a different page. This was the intended operation for calls generated from C code, but not for hand-written assembly code written for the PIC assembler. The `call` instruction is now encoded exactly as specified in the assembly source. To call destinations in another page, use the `PAGESEL` directive and mask the address. A new `PAGEMASK()` direction can do this. Alternatively, use the `fcall` instruction.

**Bogus warning for corruption (XC8-2659)** The compiler may have unnecessarily issued warning message 1481 (`call from non-reentrant function, "*", to "*" might corrupt parameters`) for functions that were not be affected by this issue.

**MB_CUR_MAX macro undefined (XC8-2657)** The `MB_CUR_MAX` macro (the maximum number of bytes in a multibyte character in the current locale) was undefined. It is now defined to be the value 1.

**Bad optimization of single instruction (XC8-2655)** For projects targeting PIC18 devices and that used optimization level `3` or `s`, any functions that were called from an interrupt and main-line code and that used the compiled stack and contained code that could be compiled to a single instruction (for example, `return 5;`), the compiler might have performed optimizations that result in the application not functioning correctly.

**Lack of warnings (XC8-2654)** No preprocessor `#warning` warning messages were issued for PIC-AS assembler source files using the C preprocessor.

**Bad assembly debug information (XC8-2645)** When building assembly projects targeting PIC10/12/16 devices, the linker would miscalculate the address associated with some lines of code within a psect that appears in more than one module. This would only affect debugging (such as setting breakpoints) and not the functionality of the assembled code.

**LIST lacking processor option (XC8-2625)** The assembler's `LIST` directive was not allowing the processor to be specified using the `p=`*device* argument. This is now permitted.

**Errors when building for REALICE (XC8-2623)** For projects using the C99 language standard and targeting an enhanced mid-range device, use of the `-mdebbugger=realice` option might have triggered multiple errors relating to trace functionality.

**Align directive breaking debugging (XC8-2618)** Usage of the `ALIGN` directive in a PIC-AS project might have caused incorrect line number information to be included in the ELF/DWARF file. This issue only affected debugging and did not result in any runtime failure.

**Invalid instruction accepted (XC8-2614, XC8-1956)** The assembler incorrectly allowed a destination register to be specified with the `movwf` instruction for all devices, for example `movwf 0x70,w`. This is no longer permitted.

**Extra trace macro (XC8-2610)** When compiling PIC18 projects and the option `-mdebugger=realice` was used, the compiler defined the `__IT` macro. This macro should only be defined by the IDE.

**Missed warning on extended instruction set (XC8-2609)** The compiler failed to detect and emit an error when the PIC18 extended instruction-set bit had been enabled by default configuration register programming. The enabling of this mode has always been reported when it was programmed explicitly through the configuration bits.

**Unnecessary bank selection (XC8-2606)** Code for mid-range devices that directly accessed SFRs located at address 0x0 or 0x1 (e.g. `INDF = 0x44`), might have triggered the inclusion of unnecessary bank selection instructions.

**Incorrect string conversions (XC8-2598)** The standard C99 library functions in the strtol family incorrectly converted strings beginning with `0x` followed by non-alphanumeric characters.

**In-line SFR bit macros (XC8-2592)** The SFR bit access macros previously supplied in the device-specific C header files (`.h` files) were unusable with in-line assembly code. This has now been corrected. For example:

```
#include <xc.h>
asm("bsf " RB4_bit);
```

**Inconsistent ptrdiff_t definition (XC8-2585)** The type represented by `ptrdiff_t` for C99 builds has been changed from a `long` type to an `int`. This is then consistent with C90 builds but also with how the compiler performs pointer subtraction. The result of any subtraction not representable in an `int` object constitutes undefined behaviour.

**vsprintf not customized (XC8-2581)** The compiler generates the code for many IO functions, based on how they are used in a program. The compiler was not properly customising the implementation of the `vsprintf()` function, resulting in the excess code being generated for this function. As part of this fix, the compiler now calls `vfprintf()`, rather than `vsnprintf()`, to perform the bulk of the printing operation.

**regsused pragma (XC8-2550)** In some instances, the `#pragma regsused` might not have worked as expected when building C99 based projects.

**Confused integer scan specifier (XC8-2506)** The `%i` conversion specifier when used with the C99 scanf family of functions was being incorrectly processed, as if it was `%d`.

**Incorrect encoding of move instruction (XC8-2504)** When using the `,0` or `,1` syntax to specify access or banked memory for a PIC18 `movwf` instruction (e.g. `movwf EECON2,0`), the assembler might have encoded it as a `movf` instruction.

**Misplaced absolute objects (XC8-2502)** When using the `-nostartfiles` option, absolute objects defined in C source might not have been linked at the specified address.

**Bogus error when using macro (XC8-2489)** The assembler might have shown an error for code that used an assembler macro with an argument beginning with `SET` or `EQU`.

**Defunct SFR aliases present (XC8-2486)** The device-specific headers and include files for PIC18FxxK42 devices contained incorrect SFR bitfield aliases. These have been removed.

**Relaxed parsing of config directives (XC8-2485, XC8-2599, XC8-2615)** To avoid interaction between the arguments of the PIC Assembler's `CONFIG` directive and definitions in device-specific header files, either the *setting* or *value* tokens or the *setting* `=` *value* expression can be surrounded by either double or single quotes to protect them from any macro substitution performed by the preprocessor, for example, `CONFIG "FOSC = ERC"`. The preprocessor will not alter anything inside the quotes.

**Incorrect configuration programming sizes (XC8-2476)** Some newer PIC18 devices now have their configuration settings located in DFM (Data Flash Memory), which are programmed as individual bytes. The compiler assumed that all PIC18 configuration settings were programmed in words and so padded or programmed with a default value an odd number of configuration bytes. For new devices that had an odd number of configuration bytes, a warning was issued when the padded bytes were programmed into the device. The new programming characteristics of configuration and user-id memory that is programmed using the `#pragma config` directive and the assembler `CONFIG` directive has been recorded in the device INI files.

**Bad interrupt context switch (XC8-2474)** When building enhanced mid-range projects that called reentrant functions from an interrupt routine, the compiler generated context save/restore code that might have been incorrect and could potentially cause data corruption.

**Config values misinterpreted (XC8-2464)** When building PIC-AS projects, named configuration values that begin with a digit (e.g. `8MHz`) would be incorrectly interpreted as a constant literal numeric value. The assembler now checks config values against all named values before trying to interpret them as a numerical value.

**Unimplemented option removed** (XC8-2463) The `-###` option was selectable for assembler projects targeting PIC devices but generated an error. This option can still be selected for AVR projects, but is no longer a valid option for PIC projects.

**Errors where they aren't (XC8-2443)** The use of macros that contained file/line directives (which may result from it containing preprocessor directives) may have caused any warning or error messages emitted by the assembler to reference the wrong file or line number.

**Unimplemented symbols (XC8-2412)** The compiler generated `__size_of_`*xxx* symbols (shown in the map file) that were assigned inaccurate values. These symbols are no longer generated.

**Broken branches (XC8-2378)** When building PIC-AS projects targeting enhanced mid-range devices, the use of the `bra` instruction with an operand that was not a constant might have resulted in incorrect encoding of that instruction.

**Unwelcome page selection (XC8-2346,** XC8-2604) For PIC-AS projects targeting mid-range and Baseline devices, page select instruction(s) may have been inserted prior to `call` or `goto` instructions that were located immediately after skip instructions, such as `btfsx`.

**Missing watchdog delay builtins (XC8-2345, XC8-2394)** The definitions for the watchdog variants of the timed delay builtins, those being `__delaywdt_us` and `__delaywdt_ms`, were missing when building for mid-range and Baseline devices.

**Redefining assert (XC8-2325)** It was not possible to redefine the `assert()` macro, which should have been performed each time `<assert.h>` was included and based on the current state of the `NDEBUG` macro. This is now possible.

**Bad branches not detected (XC8-2322)** No error was issued by the assembler when building PIC18 projects that contained branch instructions specifying a destination label that was not defined.

**Silent overlap of absolute objects (XC8-2268)** The compiler did not warn if an absolute object (defined using `__at()`) was positioned such that it overlapped with any addresses that were used internally by the compiler. A warning is now issued in such circumstances.

**Unoptimized interrupt routines (XC8-2179)** The interrupt functions defined for projects targeting Baseline devices might not have been optimized. Optimization of interrupt functions are now enabled for all devices that support interrupts.

**Unnecessary bank selection (XC8-2159)** When targeting Baseline and mid-range devices, unnecessary banking instructions may have been generated when accessing absolute objects located within unbanked memory.

**Wrong line information in messages (XC8-1962)** Code-related errors and warnings from the assembler might have indicated an incorrect C source line for some code sequences.

**No members? No build (XC8-1924)** In situations where a structure had been defined with no members, the compiler might have crashed.

**Bogus warning with character classification (XC8-1803)** Use of some C99-standard library character classification functions (viz. `isalpha()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `isspace()` and `isupper()`) caused the compiler to emit the warning `expression generates no code`. This was a side-effect of their implementation, and did not affect the code's operation. This warning will no longer occur.

**FLT_ROUNDS macro undefined (XC8-1791)** The `MB_CUR_MAX` macro (the floating-point rounding mode) was undefined. It is now defined to be the value 1, which implies rounding to the nearest representable value.

**Errno triggers scanf fail (XC8-1786)** If `errno` was non-zero before calling any of the scanf family of functions, then any of the `a`, `d`, `e`, `f`, `g`, `o`, `s` and `x` conversion specifiers would have failed to correctly read in valid input.

**Bad config ignored (XC8-1771)** The compiler was not detecting or warning against situations where a literal numerical value to a `#pragma config` directive had trailing garbage, for example `#pragma config CONFIG2L=0sillybilly`. An error will now be issued in such cases.

**Undefined assembly routines (XC8-1318)** In some instances, referencing a function that is defined in a separate assembly module in C source might result in an undefined symbol error.

## 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.

## 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.

## 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 Assembler 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 recognize 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.

## Version 2.29 (Functional Safety Release)

**Bad labels in powerup stub (XC8-2011)** The powerup assembly source provided to customize 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`.

## 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 macOS 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.

## Version 2.19 (Functional Safety Release)

None.

## 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 macOS.

**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.

## 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 pointer 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.

## 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.

### Versions 1.XX

For information regarding fixed issues relating to version 1.xx compilers, see the release notes of any compiler version prior to v3.00.

## 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.

#### 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.

**No pragma suggestions (XC8-3517)** The MPLAB Extensions for VS Code are not able to make suggestions for the arguments to any `#pragma config` directives.

**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.

### Code Generation

**Optimization error with hand-written assembly (XC8-3507)** In some instances, the optimizer might incorrectly encode flow control instructions, such as a `goto`, if the `PAGEMASK` macro has been used with the operand. This only affects hand-written assembly code and only when the `-fasmfile` option (**Assembler file** control in the MPLAB X IDE) has been enabled.

**Wrong source format accepted (XC8-3495)** The compiler is allowing C source files to be processed with the `-xassembler-with-cpp` option has been issued. This option should dictate that the source file is a pre-processed assembly file, regardless of its extension.

**Crash with unions (XC8-3458)** In some instances, compiling unions with function pointer members might cause the compiler to crash

**Crash with string address (XC8-3404)** Taking the address of a string literal, for example `&"oops"` can cause the compiler to crash.

**Parameter corruption (XC8-3377)** In cases where a function takes a byte-sized object as it first parameter and additional parameters involve an indirect function call to obtain the argument, the function's first argument might be incorrectly passed.

**Simple configuration expressions (XC8-3310)** At present, the argument expressions to the `CONFIG` assembler directive are not folded, meaning that these are limited to symbolic or literal values with no other operators.

**Invalid unlock sequence (XC8-3284)** The compiler might generate redundant bank section instructions in the UNLOCK sequence for NVM program flash Memory on some Enhanced mid-range devices. The generated code is functionally correct; however, the presence of an additional `movlb` instruction will invalidate the unlock sequence. Entering the require sequence as in-line assembly code works fine, as does repeating the first step, as in:

```
NVMCON2 = 0x55;
NVMCON2 = 0x55;
NVMCON2 = 0xAA;
NVMCON1bits.WR = 1;
```

**Lack of fixup with rcall (XC8-3266)** Fixup overflow messages are not being output in cases where the operand expressions to these instructions are evaluated by the linker to be out of range.

**Use of banned instruction (XC8-3232)** Even when employing the BSR63 errata workaround, a PIC18 `movlb 63` instruction might appear in the code generated for bitfield access.

**Merging of different code sequences (XC8-3230)** The compiler might consider assembly sequences that differ only in that one sequence has additional trailing instructions as identical. If these sequences were to be merged by the assembler optimizer, then the code could fail.

**C90 printf precision specifier (XC8-3213)** When using the C90 library, the precision specifier in a printf-type format string might be limited to a maximum value of 6. For example, the precision specifier in the following might fail: `sprintf(buf, "%.8u", 10);` This issue does not affect the C99 libraries.

**Successive cast failure (XC8-3196)** Code in PIC18 or Enhanced Mid-range devices that performs multiple successive casts on the one expression might yield an incorrect result when the reentrant stack model has been selected.

**FSR clobber (XC8-3158)** When a conditional operator needs to use an FSR register in both the condition expression and the true/false expressions, the compiler might allocate the same FSR register in both cases, resulting in the FSR being clobbered before it is used. For example, in `ptr1->select? ptr2->a : ptr2->b` loading `ptr2` to an FSR might clobber the previous load of `ptr1` to this register. This is most likely to occur when using optimization level 3.

**PIC-AS move transformation (XC8-3132)** The PIC Assembler might change hand-written `MOVFF` instructions to `MOVFFL` when the target addresses are out of range of the smaller instruction. For hand-written assembly code built using PIC Assembler, this transformation should not be taking place.

**Limited far aggregate access (XC8-3043)** The compiler will not be able to generate code for statements that either directly or indirectly access members within a structure declared as `__far`, nor will it accept a pointer to `__far`-qualified aggregate objects that is also in the far space itself.

**Structure pointer dereference failure (XC8-3041)** Dereferencing a pointer to a const-qualified structure where the pointer itself is the member of a const-qualified structure (`myConstStruct.structPtr-> myConstMember`) might fail when building for PIC18 devices.

**Over optimized conditional operator expressions (XC8-2949)** When the conditional operator (`? :`) is used in complex expressions and both true and false expression have a true value, e.g. `((testvar ? 1 : 1) && 0) ? 1 : 0`, the expression is oversimplified, resulting in the wrong result.

**Error with array return type (XC8-2933)** Functions returning an array type might trigger a `bad intermediate type` error.

**Corruption of stack-based objects (XC8-2874)** Instances have been seen where the memory allocated to the stack-based variables of one function is also used by stack-based objects of another function that is active at the same time. The memory for stack-based objects can be reused by other functions only when there is no chance of corruption. This issue only affects objects on the compiled stack. Specifying that affected functions should use the software stack (`__software` specified) will avoid any corruption.

**Out-of-range instruction targets not detected (XC8-2867)** The PIC Assembler is not detecting when any of the targets of conditional branch or call instructions (e.g. `bra`, `bz`, `rcall`) are out of range.

**Bogus warning for absolutes (XC8-2859)** Placing an absolute object at low data memory address on PIC18 devices might lead to the message, `(2084) absolute object "*" has an address that lies within memory utilized by the compiler and will likely lead to code failure`. This warning is not applicable.

**Bit-field casting ignored (XC8-2754)** Casting of bit-fields to a signed value is ignored by the compiler.

**Char type not flexible (XC8-2668)** The `-fsigned-char` (or `-fno-unsigned-char`) options current are silently ignored. The type of a plain char is unsigned char. The values for the `CHAR_MAX` and `CHAR_MIN` macros reflect this.

**Zero divide by zero (XC8-2602)** The result of a floating-point expression that uses runtime values of zero divided by zero returns Infinity, not NaN.

**IDE doesn't show all instructions (XC8-2597)** Only the first instruction of a multi-instruction assembly pseudo-instruction (e.g. `BANKSEL`) might appear in the MPLAB X IDE disassembly view. This does not affect the generates code or how it executes.

**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.

**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.

**Indirect function calls not encoded (XC8-2003)** In some cases, the compiler is not detecting that function pointers that are members of a structure have been initialized, and it will fail to encode subsequent function calls made using that pointer. The compiler might or might not issue a warning regarding the missed calls.

**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.

**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.

**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.

**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.

**White space not counted (XC8-1784)** The `%n` conversion-specifier in the C99 `scanf()` fails to count white-space in the input string.

**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];`.

**Corruption of auto objects (XC8-1608)** For complex projects targeting enhanced mid-range devices and using the compiled stack, the auto variables of functions that are concurrently active might be allocated to the same address, causing corruption of these objects.

**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.

**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 building using the C90 standard and 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. Building the project with the C99 standard selected will alleviate the issue entirely.

**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 `{ }`.

**Warning on conversion to shorter type (XC8-1246)** In some instances when building using the C90 standard, the compiler might issue a (752) conversion to shorter data type warning where this is not expected. Building using the C99 language standard will alleviate this issue entirely.

**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()`.

**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 previously provided functions `persist_check()` and `persist_validate()` 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 mid-range 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.

**Bank qualifiers (XC8E-62)** Only `bankx` qualifiers for data banks 0 through 3 are supported by the compiler. (These are enabled using the `-maddrqual` 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.

**Time zones** The `<time.h>` library functions assume GMT and do not support local time zones, thus `localtime()` will return the same time as `gmtime()`, for example.

**Haven't got time** The `time()` function is implemented for PIC devices, but always returns -1 (as described in the *Microchip Unified Standard Library Reference Guide* for situations when the target environment cannot determine the current time).

**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.

**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.

**Installer execution** On both macOS 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 macOS, 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 prescribed 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 15, and an instruction is executed that holds the value C9h in its 8 least significant bits. | Compiler avoids generating `movlb 15` instructions. A warning is issued if this 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. |

| Name | Description | Workaround details |
|---|---|---|
| `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 4000h boundary may fail based on sequences of instructions executed following the write. | No workaround applied |
| `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 `movff` instruction when the BSR holds 63 | Compiler avoids generating `movlb 63` instructions. A warning is issued if this instruction is detected. |