

MPLAB® XC8 C Compiler Version 1.20 Release Notes

Produced on Wednesday, 12 June 2013

**THIS FILE CONTAINS IMPORTANT INFORMATION RELATING TO THIS COMPILER.
PLEASE READ IT BEFORE RUNNING THIS SOFTWARE.**

Overview	2
Introduction	2
Previous Versions	2
System Requirements	2
Devices Supported	2
Editions and License Upgrades	2
Installation and Activation	3
Compiler Documentation	3
Customer Support	3
Documentation Updates	4
New Documents	4
User's Guide Clarifications and Corrections	4
Checksums	4
Local Variable Qualification	4
Assembly Code Deviations	4
Device Family Macros	5
Disabling Optimizations	5
Fletcher's Algorithm	5
New Error/Warning Messages	5
What's New	5
Version 1.20	5
Version 1.12	6
Version 1.11	6
Migration Issues	9
Version 1.20	9
Version 1.12	9
Version 1.11	10
Fixed Issues	12
Version 1.20	12
Version 1.12	15

Version 1.11	15
Known Issues	24
Microchip Errata	30

1. Overview

1.1. Introduction

This is a major update to the Microchip MPLAB® XC8 C compiler.

This compiler version introduces a significant new feature: generation of ELF/DWARF debugging files. In addition, there is new part support, and fixes to a number of reported bugs.

Previously, two HI-TECH C compilers supported Microchip 8-bit PIC devices: HI-TECH C Compiler for PIC10/12/16 MCUs and HI-TECH C Compiler for PIC18 MCUs. MPLAB XC8 replaces both these compilers and supports all 8-bit PIC devices released as of this time.

1.2. Previous Versions

The previous version MPLAB XC8 C compiler was 1.12, released November 2012.

The last released version of HI-TECH C Compiler for PIC10/12/16 MCUs was 9.83, released in September 2011. The previous HI-TECH C Compiler for PIC18 MCUs was 9.80, released in October 2011.

1.3. System Requirements

The MPLAB XC C compilers and the licensing software they utilize are available for a variety of operating systems, including Microsoft Windows XP Professional SP3/Vista SP1/Windows 7 Professional; Ubuntu 9.10; or Mac OS X 10.5. The compiler may also run on the various other Linux distributions, such as Oracle Enterprise Linux 5, Ubuntu 8.x and 10.04, Red Hat Enterprise Linux.

If you are running a license server, only computers with operating systems supported by the compilers may be used to host the license server. The license server does not need to run on a server version of the operating system. And specifically, Microsoft Windows Server platforms are *not* supported by the compiler.

1.4. Devices Supported

This compiler supports all known 8-bit PIC 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.

1.5. Editions and License Upgrades

The MPLAB XC8 compiler can be activated in several different operating modes: Free, Standard and PRO. The Standard and PRO modes are licensed modes and require a key for activation obtained after purchase. These two modes offer two levels of improved optimization features com-

pared to the Free mode. A compiler operating in Free mode can be operated indefinitely without a license. You can change to any operating mode (license permitting) from the MPLAB X IDE project properties (or MPLAB IDE v8 Build Options) on a build-by-build basis.

The compiler can be evaluated in any licensed mode for 60 days free of charge. After the evaluation period has expired, the compiler reverts to the Free mode and the trial optimization features are disabled.

You can choose the operating mode or choose to evaluate the compiler during the installation and activation process.

1.6. 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 MPLAB IDE v8.10 (or later) or MPLAB X 1.2 (or later) before installing these tools. Quit the IDE before installing the compiler. Run the `.exe` (Windows), `.run` (Linux) or `.app` (OS X) compiler installer application, e.g.

`XC8-1.00.11403-windows.exe` and follow the directions on the screen. The default installation directory is recommended. If you are using Linux, you must install the compiler using a terminal and from a root account. Install using a Mac OS X account with administrator privileges.

The option "Update MPLAB IDE to use XC8 for existing C18 projects", if set, will change your MPLAB IDE settings so that all existing legacy projects designated to use either MPLAB C18 or MPLAB C Compiler for PIC18 MCUs, will, by default, use this MPLAB XC8 compiler in C18 compatibility mode. To revert this change if you select it here, you will need to manually adjust the toolsuite settings for legacy projects that should continue to use the MPLAB C18 or MPLAB C Compiler for PIC18 MCUs compilers. This option has no effect if you do not have legacy projects.

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

The XC License Manager now supports roaming of floating network licenses. Aimed at mobile users, this feature allows a floating license to go off network for a short period of time. Using this feature, you can disconnect from the network and still use your MPLAB XC compiler. See the doc folder of the XCLM install for more on this feature.

MPLAB X IDE v1.40 includes a Licenses window (Tools > Licenses) to visually manage roaming.

1.7. Compiler Documentation

The compiler's user's guide covers all aspects of the compiler's operation, as well as other useful information. Check the well-populated index for your search term. A How-to chapter gives answers to common questions relating to the compiler and writing programs.

1.8. Customer Support

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

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

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

2. Documentation Updates

2.1. New Documents

The user's guide associated with this 1.20 version has been updated. The guide details new compiler features and expands the description of some existing features.

A PDF version of the PIC18 peripheral library document has been added to the compiler docs directory. This indicates those functions which are relevant to each device and the library interface.

2.2. User's Guide Clarifications and Corrections

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

2.2.1. Checksums

The information contained in the user's guide relating to the generation of checksum values (Sections: *4.8.17 --CHECKSUM: Calculate a Checksum* and *8.5.1.5 -CK*) does not clearly indicate that when selecting any of the Fletcher algorithms, this selection overrides any width specified. Algorithm 7 (Fletcher 8) will always reserve 2 words for the checksum value; Algorithm 8 (Fletcher 16) will reserve 4 words for the checksum value.

2.2.2. Local Variable Qualification

In sections *5.4.8.2 Near Type Qualifier* and *5.4.8.3 Far Type Qualifier*, they indicate that `auto` objects cannot use either the `near` or `far` specifiers. This text should indicate that *all* local variables, even static local objects, cannot use the `near` or `far` specifiers. The same objects also cannot be made absolute (see Section *5.5.4.1 Absolute Variables in Data Memory*).

2.2.3. Assembly Code Deviations

Section *6.4.1 Assembly Instruction Deviations* does not list the format for the index form of the `MOVIW` and `MOVWI` instructions. MPLAB XC8 uses the form:

```
MOVIW [k]FSRn
```

for both instructions.

2.2.4. Device Family Macros

When compiling for PIC18 devices, a macro that indicates the device family is defined by the compiler. For example, when compiling for a PIC18F2610, the corresponding family macro defined is `_18F2X1X_FAMILY_`. To determine the family macro relevant to your device, turn on verbose compilation and observe the `-D` options passed to the preprocessor, `cpp`. Alternatively, you may look for the `FAMILY` field in the `picc-18.ini` file in the compiler's `dat` directory.

2.2.5. Disabling Optimizations

The user's guide indicates that the `--OPT` option (or the MPLAB X IDE project properties equivalent widget) can be used to disable optimizations. Note that in order to disable intrinsic features of OCG, you must also run the compiler in Free mode. Not all OCG features can be disabled.

2.2.6. Fletcher's Algorithm

The code for the fletcher 16-bit algorithm indicated in the user's guide contains a `return` statement that should not be present. The extraneous statement is towards the end of the `fletcher16` function and is marked and bolded in the following:

```
sumB = (sumB & 0xFFFF) + (sumB>>16);  
return sumB << 16 | sum; // remove me  
sumB <<= 16;
```

2.3. New Error/Warning Messages

The following message is not present in the compiler's user's guide.

1457 local variable "*" cannot be make absolute (Code Generator)

You cannot specify the address of any local variable, whether it be an auto, parameter or static local object.

```
int pushState(int a) {  
    int cnt __at(0x100); // oops -- you cannot specify an address  
    ...  
}
```

3. What's New

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

3.1. Version 1.20

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

ELF/DWARF Debugging Previously, COFF debug files were produced to allow source-level debugging in MPLAB IDE. The compiler can now produce ELF/DWARF files, although

the default output format is still COFF. ELF/DWARF files are not compatible with MPLAB IDE v8 but will be the preferred format if you are using MPLAB X IDE as they have fewer limitations. You must be using a version of MPLAB X IDE that supports this file format for the MPLAB XC8 compiler. Not all aspects of DWARF are implemented in this compiler release. See the [Known Issues section](#) for more details.

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

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

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

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

3.2. Version 1.12

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

3.3. Version 1.11

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

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

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

operation of prior versions of the compiler was equivalent to the `rich` setting. Use the `rich` setting if you need unused symbols to be included in the link stage.

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

3.4. Version 1.10

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

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

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

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

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

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

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

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

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

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

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

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

3.5. Version 1.01

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

3.6. Version 1.00

Psect merging and splitting Two new PSECT directive flags have been added to allow splitting or merging of psects by the assembler optimizer. Now, by default, no splitting or merging takes place, but the use of the `split=allow` and `merge=allow` flags can indicate that these optimizations can take place. See the assembly language chapter in the user's guide.

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

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

4. Migration Issues

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

4.1. Version 1.20

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

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

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

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

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

4.2. Version 1.12

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

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

Implicit function in-lining The implicit in-lining of function performed as an assembler optimization is now disabled by default. The assembler optimizer has, in the past, in-lined small assembly routines as part of its optimizations. You can re-enable it using the driver option

`-A-auto_inline`. This change does not affect in-lining of C functions, controlled using the `inline` specifier.

4.3. Version 1.11

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

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

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

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

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

4.4. Version 1.10

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

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

Deprecation of in-line assembly header files The PIC10/12/16 header files that were usable from assembly that was in-line with C code have been removed. These header files were

previously included via the generic file `<caspic.h>`. The content of these files will be included once you include `<xc.h>` in your C source code.

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

4.5. Version 1.01

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

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

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

4.6. Version 1.00

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

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

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

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

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

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

5. Fixed Issues

The following are corrections that have been made to the compiler. These may 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 of benefit if you need to contact support.

5.1. Version 1.20

Bad conditional code (XC8-640) An error in an assembler optimization that dealt with a bit-test-and-skip instructions and FSR manipulations caused the sense of some conditional control statements to be inverted.

Phase errors (XC8-590, XC8-691, XC8-49) The assembler optimizer was removing redundant page selection instructions, but other parts of the assembler were not taking this into account. This created a mismatch in expected and actual sizes of some jump instructions which led to phase errors being produced. This issue affected hand-written and compiler-generated assembly code.

Absolute function placement (XC8-203) The address specified for functions made absolute was rounded down to the nearest `0x10` value. So, for example, if you attempted to place a function at address `0x56`, it was actually located at address `0x50`. This only affected programs compiled for PIC18 devices.

Missing errata NOP instructions (XC8-692) The assembler was not correctly adding in NOP instructions that form part of the `fetch` errata workarounds for some PIC18 devices. (See the user's guide `--ERRATA` option section for more information.) Delay routines that required these instructions were known to run too fast. A consequence of this fix is that code size will increase for devices that require the `fetch` errata workarounds.

Wrong `UINT24_MAX` value (XC8-565) The value for this `<stdint.h>` macro was higher by 1 than it should have been.

Bad variable access after calls (XC8-694) The compiler was, on occasion, placing a bank selection instruction before the call to a routine. This bank selection was not properly being tracked and may have resulted in incorrect access of objects once in the called function.

- Assertion failure using --ROM or --RAM options (XC8-517)** When attempting to reserving memory that was outside your target device's on-chip memory, these options may have caused an assertion failure.
- Invalid access via pointer (XC8-643)** In some situations, the compiler was not detecting that the upper TBLPTR register was being changed from its assumed state. As a result, code that dereferenced a pointer with both RAM and ROM targets may have corrupted this register and triggered a subsequent failure in the code.
- Assembler crash with bad options (XC8-510)** If options were passed directly to the assembler application but no file names were present, the assembler issued an error and continued processing. The error produced in this situation is now a fatal error to prevent the crash. This issue would have only affected users driving the assembler directly.
- Assembler crash when specifying functions as inline (XC8-589)** In some instances, when a function was declared `inline` the assembler crashed.
- Can't Generate Code message associated with unused objects (XC8-636)** In cases where a pointer was not used but was assigned the address of an object, this message may have been emitted. Optimizations associated with such code are now restricted and the message will not be issued.
- Parser crash with enumerated types (XC8-637)** The parser may have crashed when scanning code associated with enumerated types. This may have been associated with taking the address of enumerated objects.
- Qualifiers silently ignored for local objects (XC8-670)** Objects which are local to a function (including `auto` and static local objects) cannot be qualified as `far` or `near`. The compiler was not indicating that the specifier was being ignored. A warning is now issued for such definitions.
- Absolute addresses silently ignored for local objects (XC8-652)** Objects which are local to a function (including `auto` and static local objects) cannot be made absolute. The compiler was not indicating that the `@` or `__at()` construct was being ignored. A warning is now issued for such definitions.
- Linear memory allocation of objects (XC8-501)** Absolute objects that specified a linear memory address may not have been allocated correctly if the object's equivalent banked addresses mapped into the common memory. This only affected enhanced mid-range devices and objects that were not large enough to be automatically allocated to linear memory. The compiler now correctly allocates these objects.
- Identical case label values in a switch statement (XC8-493)** The compiler was not detecting the use of more than one identical `case` label value inside a `switch()` statement. This is now correctly identified and will trigger an error.
- `__DEVICENAME__` not defined (XC8-659)** This predefined macro was not being defined by the compiler. This is now defined unconditionally.
- Procedural abstraction of in-line assembly code (XC8-660)** The assembler optimizer was performing procedural abstraction on assembly code that was placed in-line with C code. The

optimizer should not perform *any* optimization of in-line assembly code. The optimizer has been prevented from performing these optimizations.

BSR register not recognized by regsused pragma (XC8-663) Any attempt to list the BSR register in the `regsused` pragma would have resulted in an error. This has been corrected and this register may now be used with this pragma for those devices that implement this register.

Allocation of PIC18 far variables (XC8-582) In some instances, `far`-qualified variables may have been linked at address 0, not in the memory defined as the far RAM.

Indexing array with constant expression failure (XC8-648) With enhanced mid-range devices only, any operation that involved adding a integer constant to an address (typically this will be an array access with a constant integer index) may have caused subsequent code to fail. Such code was not correctly reporting its use of WREG.

Initialization of large objects on enhanced mid-range device (XC8-672) Initialization of large objects (such as arrays or structures) that contained a pointer may not have been assigned the appropriate values by the runtime startup code. This only affected PIC16F1xxx devices.

Uninitialized const objects not assigned zero (XC8-553) If `const` objects were not initialized, they were not being automatically assigned the value 0 by the compiler. These objects had no memory reserved for them at all and this may have resulted in them appearing to overlap with other `const` objects. This is now corrected; any `const` object that does not have an initial value is implicitly assigned 0 by the compiler.

Assertion failure reserving RAM (XC8-662) In some instances, reserving RAM when using any non-enhanced mid-range part using the `--RAM` option would result in an assertion failure `hi >= lo`.

Integers allowed with specifiers (XC8-549) The integer constants usable with specifiers such as `__at()` were limited to 16-bit values. These values can now be specified as 32-bit values.

Multiply defined symbols (XC8-516) If code used the `__section()` specifier with variables, there may have been symbols contained in the runtime startup code that were defined more than once, producing an error. (Such symbols might be `clear_ram` or `clrloop`, for example.) This duplication has been corrected.

__mediumconst symbol undefined (XC8-621) In some circumstances, particularly when using the peripheral library, the compiler may have produced an undefined symbol error for `__mediumconst`.

Functions not inlined (XC8-521) When a function qualified as `inline` was called from `main()`, it was never inlined and a regular call was made. Inlining should now take place for any suitably qualified function called from `main()`.

Incorrect return instruction (XC8-525) For some baseline devices (most notably the 12F529T39A and 12F529T48A), the compiler may have attempted to use the non-existent `RETURN` instruction instead of a `RETLW` instruction.

Partial access to ID locations (UDBC-678) The compiler did not allow access to the entire ID location for devices that implement this memory as being 14-bits wide (e.g., the PIC16F1503). This has been corrected and you may now program all the bits of these ID locations.

Code failure accessing absolute objects straddling a bank (XC8-601) The wrong bank may have been selected when accessing absolute objects that straddle a bank boundary. Code may have failed when accessing the addresses in banks following that of the bank of the object's base address.

5.2. Version 1.12

Compile times and crash (XC8-127) A sorting issue related to pointer variables may have significantly increased the compilation time of projects. Not all projects were affected by this issue. This issue may also have caused the code generator to run out of memory, or even crash in some situations.

Compile times (XC8-498) A further issue affecting compilation times was corrected. This issue affected the assembler when `--ASMLIST` was used. (This option is on by default when using the IDE.)

Installer operation The installer program was not correctly setting the write permissions for some files. This was reported on Windows XP, but may have affected other platforms.

Bank selection issue (XC8-494) In an expression such as:

```
A = B + C;
```

where A and C are unsigned 16-bit objects in different banks and B is an unsigned char, a bank select instruction may have been omitted resulting in the wrong value being assigned to the destination variable.

5.3. Version 1.11

Compilation times (XC8-441) Large projects, particularly those targeting PIC18 devices, may have experienced increased compilation times. This was due to the compiler processing more symbols. A new option has been added, and enabled by default, to limit the symbol list. The new option is `--PARSER`. See the New Features section for more information.

Looping around pointGraphComplete() Error (XC8-442) A non-deterministic pointer-related issue was causing this error to be printed, and terminating compilation.

Debugging absolute objects (XC8-447) Some SFRs and absolute addressed objects residing in RAM would appear as being located in program memory during a MPLAB X or v8 debugging session. The content of these values would, thus, be incorrect. Absolute symbols should now appear to be located in their correct memory space and their contents shown correctly.

Parsing of config pragma arguments (XC8-452) The parsing of arguments to the `#pragma config` directive was erratic when it came to quoted arguments. The quote character `' '` is now a token delimiter and you may quote the arguments to this pragma, e.g. `"WDTEN=ON"`

Doing so will avoid any macro substitutions by the preprocessor. If you have defined macros for `ON`, `OFF` or any other token used by the `config` pragma, consider quoting the pragma arguments or moving the pragmas to a module that is not exposed to your macros.

Incorrect access of array of strings (XC8-454) Code which used a variable to access an element of a string array may have failed when the index was non-zero.

Unsupported short long message using CCI (XC8-429) When using the CCI, a warning may have been issued indicating that the short long type was not supported. The header files that referenced this type have been updated and use a plain long type when compiling for the CCI.

Undefined symbols with bitwise operations (XC8-424) Some bitwise operations, for example `|` or `&`, when used in functions that were in the interrupt call graph, may have produced code that contained references to undefined temporary symbols. Such symbols would look similar to `i2u49_41`.

Can't generate code with printing floats (XC8-108) For some placeholders associated with `float` types, a can't generate code error may have been triggered with `(s)printf`. This has been corrected.

Detection of incomplete types (XC8-109) The parser was not detecting definitions using incomplete types, for example:

```
typedef struct foo foo_t; // where foo has not been defined
foo_t x;
```

Code which is defined in such a way will now trigger an error from the parser.

Can't generate code for library string routines (XC8-413) Some string library functions may have caused "can't generate code errors". These have been adjusted to ensure correct compilation.

Assignment to volatile bytes (XC8-427) In some operating mode, when assigning '1' to a `volatile` byte variable, it may not have been updated atomically (with one write rather than clear and increment instruction). This problem did not affect absolute objects and may have only caused runtime problems if the byte was accessed from main code and interrupt.

--ROM option The `--ROM` option was not processing its arguments correctly which may have resulted in it not reserving the memory specified.

5.4. Version 1.10

Using far with Functions (XC8-337) If the `far` qualifier was used with functions, it confused the compiler into thinking the function identifier was that of a variable destined for RAM. This may have triggered an error, if no far memory was defined, or caused a runtime code failure.

Non-functional --ADDRQUAL option (XC8-293) The `reject` suboption of this option was not working correctly for PIC10/12/16 device. The `require` and `request` suboptions

were not working correctly for PIC18 devices. This option should now work as expected for all devices.

Conversion of integer to bit-field (XC8-353) The compiler was treating single-bit bit-fields as a boolean rather than an integer quantity when it came to assignment. Code that assigned a byte or larger integer to a single-bit bit-field may have failed.

Can't Generate Code for Duplicated Functions (XC8-358) The compiler may have issued Can't Generate Code error messages in situations where a function was duplicated in interrupt and main-line code and this function used pointers.

Handling of incomplete types (XC8-374) The parser was producing incorrect type information where a type used in a declaration was, at that point, incomplete, e.g. if you used a structure tag in a declaration, but that tag had not been defined. The code generator would subsequently crash if such incomplete type information was encountered. types should always be defined before they are used, but the compiler will no longer crash on such situations.

Testing of volatile bytes (XC8-388) The code that tested `volatile` bytes variables for (in)equality was using a `MOVWF x, f` instruction. This was not in keeping with the spirit of the `volatile` keyword and could also play havoc with certain registers, such as the TMR0 register, on some baseline devices, such as the 10F222, which require a read delay after being written. This code sequence is no longer used and access of SFRs will be well behaved.

Pointer assignment failure (XC8-342, XC8-343) In some circumstances, assignment of a structure member address to a pointer may have resulted in an incorrect destination pointer size being determined by the compiler. This would have resulted in subsequent pointer dereferences being invalid. This problem was detected only with assignments to pointer parameters as part of a function call, but could have potentially occurred elsewhere.

Pointer access failure with duplicated functions (XC8-377) If a function that has a pointer parameter is called from main-line and interrupt code (i.e. it is duplicated by the code generator), the compiler may have issued a "looping around allocGlobals", or "can't generate code" error message. In other situations, the code may have compiled, but accesses to the pointer may have been incorrect. The issue could have affected any device with interrupts.

Undefined btemp symbol (XC8-371) In some circumstances, exacerbated by the use of a debugger and devices with small amounts of RAM, the internal compiler symbol `btemp` may not have been defined and an error results. This symbol is now correctly defined.

Incorrect access of high program memory (XC8-363) The compiler was not correctly setting the TBLPTRU register for accesses of absolute-addressed `const` data located above 0xFFFF in PIC18 program memory.

Incorrect configuration bit/user ID settings (XC8-385) Attempts to program the configuration bit or user ID settings may have failed due to a sorting bug that may have resulted in the bits being programmed in the wrong order. The issue was inconsistent, but could affect all devices.

Improved error relating to IDLOC (XC8-384) If non-hex digits were used in for any nibble in the `IDLOC()` macro a confusing error was issued. A new message (#1436) has been created for this situation.

5.5. Version 1.01

Looping around allocGlobals error (XC8-318) This error may have been triggered, but is not specific to any particular code sequence. The compiler has been updated to ensure this trigger will not result in this error.

Access of char arrays (XC8-304) If the index expression used with an array of `char` consists of a non-constant expression (e.g. a plain variable) from which is subtracted a constant (e.g. `myArray[idx-1]`), then the index calculation may have been incorrect and the wrong element accessed. This issue mostly affected PIC18 devices and arrays no larger than a bank. It may have affected enhanced mid-range parts with array sizes larger than 256 bytes. The issue has been resolved for all devices.

Code jumps to wrong location (XC8-295) If the assembler optimizer is enabled, assembly code that directly wrote to the program counter may have been abstracted which may have caused a jump to the wrong location and code failure. C code affected would produce a lookup table of some description, but it unlikely that C code would trigger this issue. Hand-written assembly would only be affected if the option to optimize assembly source files was enabled.

Duplicated SFRs (XC8-319) Some SFR bit-field structures had duplicate entries in the device-specific header files and have been removed.

Bit objects on baseline devices Uninitialized global bit objects may not have been zeroed for baseline devices, and in some instances, the generated startup code may have corrupted other memory locations or caused the device to restart.

Call graph inconsistencies The call depth of some functions was not correctly indicated in the call graph shown in the assembly list file. The total amount of auto, parameter and temporary variable usage was also incorrectly displayed for some functions in the call graph.

Wide bit-fields in header files (XC8-289) Some header files were being generated with SFR bit-fields that were wider than a byte. This is not allowed by the compiler. Such definitions are no longer contained in the header files.

Wrong external memory access (XC8-286) The `--EMI` options was not correctly being processed and this may have meant that access to external memory may have been incorrect. Only several PIC18 devices have such memory.

Error on assembly directives after including header file (XC8-285) After including the `xc8.inc` header file into assembly code, some assembly directives, e.g. the `DS` directive, may have generated an error. This was due to SFR names conflicting with the directive's name. The header files will on longer use any SFR name that conflicts with a directive. This will mean that SFR names may not always match those listed in the device data sheet.

- Bad access at start of function (XC8-94)** The code at the beginning of function could have accessed the wrong address (bank) for a variable if the compiled stack was built up over multiple banks and WREG was used to pass a function argument.
- Incorrect RAM ranges with 12F1501 (XC8-274)** The RAM ranges associated with this device did not reserve the memory from 50-6F. Allocation of user-defined objects to this memory may have caused code failure.
- Bad results with 24-bit expressions (XC8-227)** The code generated for some arithmetic and bitwise operations involving 24-bit integers were missing some necessary bank selections which may have generated wrong results.
- Compiler crash with conditional operator (XC8-117)** Assigning a pointer the result of an expression using nested conditional statements could have caused the compiler to crash for PIC18 devices.
- Bad results with long arithmetic (XC8-241)** The results of some long arithmetic expressions may not be correctly assigned due to a bank selection bug.
- Build errors in MPLAB X IDE (XC8-101, XC8-104)** Source file paths that contained spaces were being written to the dependency files without the spaces escaped. This resulted in these files containing erroneous targets and dependencies which may have resulted in incorrect builds. In addition, the preprocessor would generate a dependency file for assembly modules incorrectly assuming that the intermediate file to have a ".p1" extension and not ".obj" resulting in the same behavior.
- Crash with compiler-domain symbols (XC8-18)** User-define variables and functions should never start with an underscore character as such symbols are in the compiler's domain. Inappropriate use of these symbols in a program, however, was leading to a compiler crash. The crash no longer will occur; however, you should continue to avoid using symbols beginning with an underscore.
- MPLAB X IDE plugin** Previously the XC8 MPLAB IDE plugin overwrote the HI-TECH Universal Plugin such that the universal plugin no longer appeared in the toolsuite list. This is no longer the case. The XC8 v1.01 installer will install and reinstate both the XC8 and universal plugins so that they will both be selectable from the IDE.
- Bad if() code (XC8-58)** In some instances the assembler optimizer would incorrectly move `MOVLW` instructions which could cause some logical expressions (such as those used by if statements) to be incorrectly evaluated.
- Not honoring message disable (XC8-62)** When compiling for PIC10/12/16 targets the assembler was not honoring the `--MSGDISABLE` option. Thus, it was impossible to disable warning messages produced by this application. This issue did not affect other applications or any application when compiling for PIC18 targets.
- Bad call to subtype() Error (XC8-73)** In some instances where a function has a pointer type as its first parameter but the function is never called, a "bad call to typeSub()" error may occur.

Incorrect optimizations involving carry bit (XC8-77) The assembly optimizer may have incorrectly moved instructions that set the carry bit above other code that would subsequently clear carry before it was being used.

Bad optimization of indirect access (XC8-95) In some situations, expressions with the form `*ptr = *ptr op A;` may fail. In particular it is the optimization of assignment-operations when the destination is an indirect access which can lead to incorrect results.

Bad right shift code (XC8-105) In some instances for PIC18 targets, the code generated for right-shift a `signed long` operand would access the wrong file registers, giving an incorrect result.

Memory reservation using --CODEOFFSET (XC8-230) This option should have reserved memory in all linker classes associated with program memory. It was only reserving this memory from the CODE class but not for other classes that hold const objects. This was unlikely to cause issues since const objects are typically not allocated to low address; however, all classes are now adjusted.

Badly optimized if() code (XC8-75) The assembler optimizer may have procedurally abstracted code that erroneously contained return-style instructions. This may have caused code such as `if()` statements to be ignored and fail. This has now been corrected.

Redundant MOVLP instructions (XC8-49) Redundant MOVLP instructions were being produced for some call sequences for Enhanced mid-range devices.

5.6. Version 1.00

Bogus warning on -B option (XC8-11) If compiling under MPLAB v8, in some instances a warning indicating that the `-B` option was defunct was issued. This option is indeed defunct, but the compiler has been adjusted so that this is not produced when compiling under the IDE.

Parser crash (PICC18-618) If an unexpected attribute was used with function definitions, e.g. `__attribute__((const))`, the parser crashed instead of giving an error. This has now been corrected.

Parser crash (PICC-684, PICC-688, PICC18-596, PICC18-607, PICC18-616, PICC18-619, PICC18-620, PICC18-621) In some circumstances, the parser would crash when encountering illegal or unusual source code. This has been seen with code that accesses some structure members, passing malformed arguments to functions, or with non-prototyped (K&R) function definitions. Changes to the parser should prevent the application crashing.

Bad code associated with negation (PICC-652) In certain cases, negating a 16-bit integer may have overwritten WREG. This has been corrected.

Crash on structure with no members (PICC-597) If a structure was defined that had no members, the parser application crashed. This has been rectified.

Arrays of structures in MPLAB not watchable (PICC-544, PICC18-593) In some circumstances, the watch window in MPLAB IDE v8 would show elements of a structure array correctly. Changes have been made to the compiler output to correct this.

Redirecting function using psect pragma (PICC-514) There were instances where use of the `#pragma psect` would not appear to work as the name of the psect being redirected was changing as a result of psect merging by the assembler optimizer. A new system of identifying mergable and splittable psects was introduced, which will assist in reducing this problem.

MPLAB IDE popup about source code using #include (PICC18-565, PICC18-566) If you are using any of the `printf` family of functions, an MPLAB IDE popup may have appeared warning that "The project contains source files that use the `#include` directive from within a function to include source code". Although this would not have affected normal debugging operations in this particular instance, the trigger for message has been adjusted so that this message will no longer be produced.

Zeroing of config bits (PICC18-600) If you used the PIC18 pragma to program only the lower byte of a configuration location, the compiler may have zeroed the upper byte rather than use the default configuration value. Ideally, all configuration words should be specified in your programs or the `--RUNTIME` option to program the device with the default config words should be enabled. However, this zeroing of the upper byte will no longer occur.

Undefined symbol with empty loops (PICC18-524) The compiler can remove some condition code if the state of variables used in the controlling expressions are known. In some instances, where the true or false statements were an empty loop (e.g. `while`), the compiler may have deleted a label that was still being referenced by other code. This would have produced an undefined symbol, which might look something like `19`.

Crash with undefined functions (PICC18-532) If a function that were not defined were assigned as the target of a function pointer, the compiler may have crashed. This crash has been fixed and now the linker will report the symbols as being undefined, as expected.

Errors indicated in unused files (PICC-428) When an error indicating a variable is too large is emitted, the name of the file that contained the variable definition may have indicated a source file that was not in the project, such as a library file.

Assignment to structure ignored (PICC-433) In situations where an assignment is made to a structure is followed by an assignment to a bitfield within that structure, the initial assignment may be removed. This would only occur in PRO mode. This has now been fixed and the compiler notes the distinction between the objects accessed.

Compiler crash and old library modules (PICC-573) The compiler may have crashed if some library modules were used from the SOURCES directory of the compiler. This directory no longer includes files from non-OCG compilers, as they are not compatible to OCG compilers and will not compile.

Undefined ?Fake symbol (PICC-589, PICC-581) This error may have occurred for code that indirectly called an external function defined in assembly code or was part of another build. Only functions that returned a value in memory would be affected by this issue. The error is no longer produced, but you are required to define a symbol that represents the memory

location where the return value will be stored. See the User's Guide on External Functions for more information.

Linker crash with assembler optimizers (PICC-648) If the assembler optimizers (psect merging) were enabled, in some situations the linker would crash if a psect was removed but was still referenced by a directive. This has been corrected and all assembly optimizations may be employed.

Compiler crash with bad memory ranges (PICC-608) If a memory range was specified using the `--RAM` option, or its equivalent in MPLAB IDE, and that range was excessively large, the compiler may have produced an assertion failure. In such situations, the compiler now prints a more informative error.

Cromwell error with MCPxxxx targets (PICC-642) If compiling for an MCP device, an error relating to the "prefix list" would have been triggered. This has been corrected and compilation can be performed for these devices.

Cromwell crash (PICC-493) Cromwell may have crashed if encountering information generated from static bit objects that are absolute.

Crash with malformed warning pragma (PICC-610) If the message number(s) were omitted from the `#pragma warning disable` directive, the compiler crashed. An error is now emitted if this situation is encountered.

Read of write-only registers (PICC-658) If an attempt was made to read a Baseline device write-only register, e.g. `TRIS`, the compiler would produce a can't generate code error message. A more specific message is now produced alerting you to the offending operation.

Can't generate code (PICC-683, PICC-499) The compiler was not able to generate code for some expressions involving logical or bitwise operation on `bit` objects. This has been corrected.

Prototype for `eeprom_write` (PICC-675) The prototype for `eeprom_write` incorrectly stated that this function returned a value. This was not correct and the prototype now indicates a return type of `void`.

CLRWDT appearing in `_delay` output (PICC-460) For some delay values, the `_delay` in-line function for PIC18 targets only produced `CLRWDT` instructions as part of the delay. This has been corrected and a `NOP` is used instead. For PIC18 targets, you have a choice of `_delay` or `_delaywdt` to implement in-line delays, which do not use and use, respectively, the `CLRWDT` instruction as part of the delay.

Optimization of volatile bits (PICC-606) The compiler was not generating the appropriate information to prevent the assembler optimizer from optimizing access to volatile bit objects. Although the code would have been functionally correct, subsequent optimizations may have caused runtime failures. This correct information is now generated and volatile bit access will not be optimized unexpectedly.

vprintf and vsprintf linker errors (PICC-524, PICC-619) Trying to use either of these functions may have resulted in the undefined symbols `__doprint`, `_vprintf` or `_vsprintf`. These routines are now fully supported in the libraries and can be called.

Upper bound of mktime (PICC18-126) The `mktime` function had a upper limit of the year 2020; higher values would return incorrect results. This limit has now been extended to the year 2038.

Bad call to subtype with regsused pragma (PICC18-392) If using the `regsused` pragma for a function before the definition of that function, this error occurred. This has been corrected and the pragma can be used at any point in the source code.

Looping around allocGlobals error (PICC-570) This error may have occurred in some situations, particularly when `NULL` pointers were being assigned in the source code. This has been corrected.

Read of wrong array element (PICC-542) When reading a `const` array using a constant index, in some situations, the wrong element may be read. This bug did not affect array access using a variable or other expression as the index.

Memory allocation An improved memory allocation scheme is used when variables are qualified as being in a particular bank or near and the `--ADDRQUAL` option is set to require. This will reduce the likelihood of Can't find space errors from the linker.

Code generator crash with undefined functions (PICC18-532) If a function pointer was assigned the address of a function that is defined in external code, the code generator may have crashed. External functions might include those defined in assembly source code. This only affect PIC18 targets.

Can't generate code errors with nested structures (PICC-628) This error may have occurred for code that accessed bit-fields within structures that themselves were members of other structures.

No stack allocated to function error (PICC-611, PICC-485, PICC-416, PICC-637) This error is now unlikely to be triggered by code. It was most likely to occur with complex expressions, particularly involving floating-point arithmetic.

Can't generate code errors with short long types (PICC-632) Some assignment operations on short long types, e.g. `|=`, `&=` and `*=` etc may have triggered a "can't generate code" error when compiling for baseline or mid-range parts. The ability to generate code for such expressions has been enhanced.

Parser crash with bad identifiers (PICC-622) The parser (`p1`) may have crashed if certain keywords were used as a variable identifier, e.g. if you were to try to define a variable with the name `eeeprom`. A crash may also have occurred if reserved keywords were used in place of variables in expressions. The crash has been corrected, but variable identifiers cannot be reserved keywords.

Symbol defined more than once for in-line assembly (PICC-563) If the `REPT` macro was specified in in-line assembly, the code generator was generating additional labels multiple

times. This triggered an error indicating the symbol was defined more than once. This has been corrected and `REPT` can safely be used in in-line assembly code.

RAM ranges for 16(L)F1507 The RAM ranges available when selecting this device extended further than the physical device implementation. This has been corrected.

6. Known Issues

The following are limitations in the compiler's operation. These may be general coding restrictions, or deviations from information contained in the user's manual. The bracketed label(s) in the title are that issue's identification in the tracking database. This may be of benefit if you need to contact support.

ELF debugging issues Not all aspects of the ELF/DWARF debugging file have been implemented in this compiler release. 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.

- Unused variables will not be identified in the ELF file.
- Constant propagation optimizations may affect which variables are watchable in the IDE or the values that are indicated in the Watch window.
- In-line C functions will not be debuggable.
- Procedural abstraction will affect the operation of breakpoints.
- Variables residing in both memory and registers may not be correctly displayed in the Watch window.
- Nested pointers may not be dereferenced correctly in the watch window, hence the values in reported in the Watch window will therefore be erroneous.
- Variables located in external memory will not be displayed correctly in the Watch window.
- In the Watch window the *type* name (as opposed to the object's name) that appears for a anonymous structure or union typedef or an enumerated typedef will be ".".
- 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.

Use of `__section()` specifier If you use the `__section()` specifier when defining an object, you must also use the same specifier with any declaration of the same object. Failure to do so may result in the specifier being ignored.

Support for `scanf` functions The compiler does not directly support any of the `scanf`-family functions, such as `scanf`, `sscanf`, `vscanf`, etc., and they are not compiled into the library files. Example source code for some of these functions can be found in the compiler's `sources` directory. These examples have not been verified to work on all devices and should be used with caution. Consider modifying this code to suite your application.

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.

No support for vsprintf (XC8-425) This function is not supported in the compiler libraries.

Previous attempts to use it would have resulted in undefined symbol errors. This function is now flagged as unsupported and its use will trigger an error to that effect.

Installer execution On both Mac OS X and Linux, it is necessary to run the installer as root or with superuser privileges (using `sudo`, for example). If the installer is started without superuser privileges on Mac OS X, it will exit and display an informative message. In the same situation on Linux, the installer will fail when it attempts to write to directories for which it does not have adequate rights. The messages displayed will relate to these access failures. For correct operation, run the installer via `sudo`, or as the root user, on these systems.

PATH environment variable On Linux systems, the installer, by default, updates the `PATH` environment variable to include paths to the new executables being installed. If the installer is run via `sudo`, the default action will update the `PATH` variable of the user executing the `sudo` command. If the installer is run by root, the installer will only update root's `PATH` variable, and not the `PATH` variables of ordinary users. If installing the compiler while logged in as root, a better choice is to update *all* user `PATH` variables. Alternatively, skip the step to update the `PATH` variable in the installer, and to update the `PATH` variables of users who will use the software, manually.

C18 code compatibility The C18 code compatibility feature is only a beta implementation. Legacy C18 projects may not compile, or compile with runtime errors when using XC8.

Changing toolsuite in MPLAB IDE v8 If the toolsuite associated with an MPLAB IDE v8 project is changed from the HI-TECH Universal toolsuite to the XC8 toolsuite, or vice versa, the selected options in the original toolsuite are not copied over to the new toolsuite. Please note all options before you switch and confirm the options in the new toolsuite.

IDE plugin overwrite (XC8-360, XC8-367) If an older compiler is installed after installing a newer compiler, the new IDE plugin may be overwritten. (The 1.01 compiler installer is known to overwrite the 1.10 plugin.) You can manually reinstall the new plugin without having to reinstall the compiler. Run the command:

```
regsvr32 "C:\Path_to\microchip\xc8\v1.10\bin\MPLABXC8.dll"
```

for example, replacing the path with that appropriate for your compiler installation.

Call to empty function after comparison (XC8-398) Code which calls a function containing no statements (an empty definition) which is preceded by an `if()` statement may fail for 16F1xxx parts. If the assembler optimizer is enabled, the call to the empty function, or subsequent code may fail. Adding any code, such as `NOP();`, to the empty function is enough to work around this issue.

Assignment to pointer member of a union (XC8-341) For PIC18 devices and when the compiler is operating in PRO mode, assignment of an address to a pointer that is a member of a

union which in turn is part of a structure may fail if the member is accessed via structure pointer `a` in the following example.

```
strPtr->myStruct.myUnion.myPtr = mySrcPtr;
```

Functions returning pointers (XC8-344) If a function returns a pointer, the size of this pointer is larger than 1 byte, and also larger than the total size of all the parameters passed to this function, then the memory used by the return value may not be allocated. The function's return value may overwrite other local variables and lead to code failure. Consider adding a dummy parameter to the function with the same size as the return value type.

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.

In-line delay values The delay value specified with the in-line delay function (`_delay`) is limited to 179,200 on PIC18 devices. With PIC10/12/16 devices, this value may be up to 50,659,000.

In-line delays On PIC10/12/16 targets, the only in-line delay function implemented is `_delay`. This uses NOP instructions, when required, as part of the delay sequence. There is currently no equivalent version of this function that uses the CLRWDT instruction. The function, `_delaywdt` which is available for PIC18 targets, does use the CLRWDT instruction.

Watching const values in MPLAB v8 MPLAB IDE v8 cannot correctly display in the Watch view any object qualified as `const` and stored as RETLW instructions in memory. Any `const` object compiled for a PIC18 device will be observable, and this issue does not affect MPLAB X IDE.

Use of hardware multiply The PIC18 hardware multiply instruction is not used for any 32-bit multiply operations. A library multiply routine is used for such operations. The hardware multiply instruction is used for 8- and 16-bit multiplication.

Disabling function duplication (XC8-3) The `interrupt_level` pragma current does not prevent duplication of functions called from both main-line and interrupt code.

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.

In-line assembly and labels 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.

Bank qualifiers Only `bankx` qualifiers for data banks 0 through 3 are supported by the compiler. (These are enabled using the `--ADDRQUAL` option). Use absolute variables to place objects in other banks, if required.

Memory Report for Absolute Variables If the user defines absolute variables (variables placed at an absolute address via the @ construct), and any of these variables overlap other variables in memory, the compiler will also count overlapping variables when it comes to reporting on memory usage. That is, if two, byte-sized variables are located at the same address, the report will indicate 2 bytes being used, rather than 1.

Out of bound variables in watch view If variables are defined for Enhanced Mid-range devices that are large and placed in the linear memory space, MPLAB IDE v8 shows these objects are being "out of bound". This is display issue and is not indicative of code failure.

Variables shown in wrong memory space If an object is made absolute and allocated address 0, it may only be visible in the program memory view of MPLAB IDE v8. This is purely an IDE view issue and does not impact on the code operation.

Wrong watch values MPLAB IDE v8 may show incorrect values for pointers and any object located after a pointer in a structure. The COFF debug format used by this IDE does not allow the compiler to specify the necessary size information for pointer variables. The compiler makes pointers a certain size based on their usage, whereas the IDE assumes all pointers are one fixed size. This can make viewing pointers "hit and miss". Any attempt to examine pointers in the IDE's Watch view may result in incorrect values being shown, or an "invalid" message being displayed in the view. In addition, if a pointer is a member of a structure, any member (of any type) located after the pointer in the structure's memory may also be shown incorrectly in the Watch window. The actual contents of the pointer and program execution are not affected. If you are using MPLAB X IDE, then use the new ELF/DWARF debug format, as this allows accurate size information to be specified and will eliminate the issue.

Switch strategies 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.

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.

SFRs for 12LF1840T48A The following are inconsistent in this device's header file.

- DACNSS bit not defined in DACCON0 register (118h)
- SSP1M0, SSP1M1, SSP1M2, SSP1M3, SSPEN, SSP1OV defined as SSPM0, SSPM1, SSPM2, SSPM3, SSPEN, SSPOV in SSP1CON1 register (215h)

SFRs for 16(L)F1507 The following are inconsistent in this device's header file.

- In NCO1ACCU (49Ah) NCO1ACC<23:20> not defined
- CWG1ASD register (695h) defined as CWG1CON2

Initialization of eeprom arrays Initialization of large arrays that are qualified `eeprom` may fail on enhanced mid-range devices. When the size of these arrays exceeds the size of the free

space in each GP RAM bank, the compiler tries to allocate the array to the linear RAM space, not EEPROM. Small arrays are not affected by this issue.

Initialization of auto structures Structures which are `auto` cannot be initialized. Simply define the structure and assign values separately.

EEPROM variable limitations There are some limitations to using variables qualified as `eprom`, aside from them being not supported at all on PIC18 devices. Avoid using these variables in complicated expressions and remember that code which accesses such objects is complex and very slow.

Size of EEPROM arrays Arrays that qualified as `eprom` and which are initialized may not have all initial values written to the EEPROM.

Constant index into int array If a constant is used as the index into an array of `const int` type, the element read may be incorrect. This issue does not affect arrays of different types, or when the index is a variable.

Absolute initialized variables Variables which are absolute and which are not `const` cannot be initialized. The following example will not generate an error, but will not work as expected. Define the variable as absolute and initialize it in main-line code.

```
unsigned int varname @ 0x0200=0x40; // will not work as expected
```

Stack overflow 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.

Main function size If the function `main` produces assembly code that is 0x1FF words long, this cannot be placed in the program memory and a "can't find space" error message is produced. Increasing or decreasing the size of the function will allow it to be positioned correctly and the error will not be displayed. This only affects Baseline PIC devices.

Functions called from in-line assembly 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.

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.

Indirect function calls The parameters to functions called indirectly via a pointer may not be passed correctly if the following conditions are met.

- A function is called indirectly from both main-line and interrupt code
- This function has parameters
- The function pointer used to call this function is also assigned the address of other functions at some point in the program; and
- This function pointer is initialized when it is defined.

All other instances of indirectly calling a function, even from interrupt and main-line code, will work as expected.

option and tris 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) is not supported by this compiler.

fast32 floats The option to select the fast 32-bit float or double library for PIC17 devices that was included in the PICC STD compiler is no longer available.

Procedural abstraction An instance has been seen, but not resolved, which appears to be related to the procedural abstraction optimization of the assembler. If suspect behavior is encountered, compiling for speed, instead of space, will disable this optimization and allow you to confirm if this is the underlying cause. This problem was reported for an Enhanced Mid-range device.

Pointer assignments In PRO mode, if a pointer assignment is made where one of the pointers is a member of a structure, the compiler may fail to copy the whole pointer, causing the pointer to register an incorrect address.

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. 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 use a literal value with this macro.

MPLAB 8 debugging information When compiling projects with multiple C files, MPLAB may throw up a warning dialog with the message beginning "The project contains source files that use the `#include` directive from within a function to include source code." This is due to the Omniscient Code Generation in the XC8 compiler, which can make multiple source files appear in the one COF output. It does not affect the operation of the actual project.

Selecting compiler mode When using the Free version of the XC8 compiler, the ability to select compilation modes in the plugin is disabled. If you switch to an older version of the compiler inside the MPLAB 8 plugin, you will not be able to change the compilation mode to Standard or PRO. To work around this behavior, close the plugin, then reopen it. This should allow other compilation modes to be selected.

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.

MPLAB IDE integration If Compiler is to be used from MPLAB IDE, then you must install MPLAB IDE prior to installing Compiler.

7. Microchip 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 `--ERRATA` command line option. All these errata are PIC18 specific, except for the CLOCKS`W` 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.

Name	Description	Workaround details
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 (>1FFFFh) may yield erroneous results at the extreme low end of the device's rated temperature range (-40o C).	Affected library sources employ additional NOP instructions at pre- scribed locations.
RESET	A GOTO instruction placed at the reset vector may not execute.	Additional NOP instruction inserted at reset vector if following instruction is GOTO
BSR15	Peripheral flags may be erroneously affected if the BSR register holds the value 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.

Name	Description	Workaround details
DAW	The DAW instruction may improperly clear the CARRY bit (STATUS<0>) when executed.	The compiler is not affected by this issue.
EEDATARD	When reading EEPROM, the contents of the EEDATA register may become corrupted in the second instruction cycle after setting the RD bit (EECON1<0>).	The EEPROM_READ macro read EEDATA immediately.
EEADR	The result returned from an EEPROM read operation can be corrupted if the RD bit is set immediately following the loading of the EEADR register.	The compiler is not affected by this issue.
EE_LVD	Writes to EEPROM memory may not succeed if the internal voltage reference is not set.	No workaround applied
FL_LVD	Writes to program memory may not succeed if the internal voltage reference is not set.	No workaround applied
TBLWTINT	If a peripheral interrupt occurs during a TBLWT operation, data can be corrupted.	Library routine flash_write() will temporarily disable all applicable interrupt-enable bits before execution of a TBLWT instruction.

Name	Description	Workaround details
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.