MPLAB® XC8 C Compiler Version 2.50 Release Notes for AVR® MCU

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

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

1. Overview

1.1. Introduction

This release of the Microchip MPLAB® XC8 C compiler is a minor update that contains several new features, bug fixes, and new device support.

1.2. Build Date

The official build date of this compiler version is the 26 July 2024.

1.3. Previous Version

The previous MPLAB XC8 C compiler version was 2.49, a functional safety compiler, built on 23 January 2024. The previous standard compiler was version 2.46, built on 5 January 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

The MPLAB XC8 C Compiler for AVR MCUs tools are written and distributed under the GNU General Public License (GPL) which means that its source code is freely distributed and available to the public.

The source code for tools under the GNU GPL may be downloaded separately from Microchip's website. You may read the GNU GPL in the file named `license.txt` located the `avr/doc` subdirectory of your install directory. A general discussion of principles underlying the GPL may be found here.

Support code provided for the header files, linker scripts, and runtime libraries are proprietary code and not covered under the GPL.

This compiler is an implementation of GCC version 5.4.0, binutils version 2.26, and uses avr-libc version 2.0.0.

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 AVR MCU devices at the time of release. See `avr_chipinfo.html` (in the compiler's `doc` directory) for a list of all supported 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 version 5.0 or later 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.

The XC Network License Server is a separate installer and is not included in the single-user compiler installer.

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 includes a Licenses window (Tools > Licenses) to visually manage roaming.

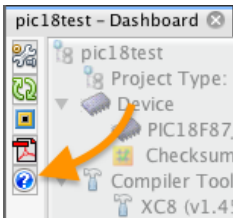
1.9.1. Resolving Installation Issues

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

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

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

1.10. Compiler Documentation



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

If you are building for 8-bit AVR targets, the MPLAB® XC8 C Compiler User's Guide for AVR® 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.

2. Documentation Updates

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

New or updated AVR documentation in this release:

- None

The *AVR® GNU Toolchain to MPLAB® XC8 Migration Guide* describes the changes to source code and build options that might be required should you decide to migrate a C-based project from the AVR 8-bit GNU Toolchain to the Microchip MPLAB XC8 C Compiler.

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 AVR® 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 - AVR® MCUs* (DS50003108) 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 *Hexmate User's Guide* is intended for those running Hexmate as a stand-alone application.

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

2.1. Inline-functions Option

The `-finline-functions` option requests that all functions be considered for inlining, even if they are not declared `inline`. The compiler heuristically decides which functions are worth integrating in this way.

If all calls to a function are integrated and that function is declared `static`, the function is normally not output as assembler code in its own right.

This option is automatically enabled when the optimization level is set to `3` or `s`.

2.2. No-data-init Option

The `-mno-data-init` option disables runtime startup data initialization, preventing uninitialized objects from being cleared and initialized objects from being assigned their initial value. Use this compiler option in preference to the `-Wl,--no-data-init` linker option, as the compiler will omit several symbols if it knows that data initializations is not required. This might reduce the size of the generated code.

Use this option with caution, as any code that relies on objects being cleared or containing their initial value at startup might fail.

2.3. W: Enable/disable warning Option

The `-W`*`msg`* option enables warning and advisory messages. The *`msg`* argument is the name of a message issued by the compiler. The special message argument `all` indicates that all messages

should be enabled. This option has precedence over the `-w` option, which disables all warning messages, so it can be used to re-enable selected messages when `-w` has been used.

The `-Wno-`*`msg`* form of the option disables the indicated message. Where the *`msg`* argument corresponds to a warning or advisory message, this message will never be issued by the compiler or assembler. If it corresponds to an error or is not recognized, the compiler will indicate via an error that this operation is not permitted.

The name of the message is printed in compiler warnings. For example, the following is an example of a `return-type` warning.

```
main.c:12:1: warning: control reaches end of non-void function [-Wreturn-type]
```

In the case where `-Wno-`*`msg`* has been issued and *`msg`* is a number that corresponds to a warning that has been promoted to an error through the `-Werror` option, then that message will not be issued by the compiler, nor will any error corresponding to that message.

2.4. Keep Attribute

The `keep` attribute can be used with variables and functions to ensure that they are not removed from the output, even if they have not been referenced in the program.

While the `used` attribute ensures that unused symbols are output to the object module, those symbols might still be removed by the linker's garbage collection if their sections have not been used. The `keep` attribute ensures that the symbol will be present in the final output without having to disable garbage collection or pass the linker an undefined symbol reference.

3. What's New

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

3.1. Version 2.50

New Device Support Support is now available for the following AVR devices: AVR16DU14, AVR16DU20, AVR16DU28, AVR16DU32, AVR32DU14, AVR32DU20, AVR32DU28, AVR32DU32, AVR64DU28 and AVR64DU32.

Shipped DFPs The following are the AVR-specific DFP versions that ship with the compiler: ATautomotive_DFP v3.1.73, ATmega_DFP v3.0.158, ATtiny_DFP v3.0.151, AVR-Dx_DFP v2.5.294, AVR-Ex_DFP v2.9.197, XMEGAA_DFP v2.2.54, XMEGAB_DFP v2.3.149, XMEGAC_DFP v2.2.40, XMEGAD_DFP v2.3.50, and XMEGAE_DFP v2.2.47

DIVA module support (XC8-3321) For AVR devices that implement a 32-bit Divide Accelerator (DIVA) module, which can perform 32-bit signed or unsigned integer hardware divisions and 32-bit shifts and rotates in hardware, the compiler can employ routines which set up, initiate, and obtain results from this module. The DIVA feature is automatically enabled when available, so that C division, shift and modulo expressions will invoke the DIVA routines. The `-mdiva` driver option makes this action explicit. The `-mno-diva` option can be used to disable this feature, having these operations instead performed by conventional library routines that will be larger and slower than the DIVA routines.

Disable warnings (XC8-3308) A new `xc8-cc` 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.

Keep attribute (XC8-2648) The compiler now implements a `keep` attribute that can be used with variables and functions to ensure that they are not removed from the output, even if they have not been referenced in the program.

3.2. 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.

3.3. Version 2.46

New Device Support Support is now available for the following AVR devices:

None.

Shipped DFPs The following are the AVR-specific DFP versions that ship with the compiler: ATautomotive_DFP v3.1.73, ATmega_DFP v3.0.158, ATtiny_DFP v3.0.151, AVR-Dx_DFP v2.3.272, AVR-Ex_DFP v2.8.189, XMEGAA_DFP v2.2.54, XMEGAB_DFP

v2.3.149, XMEGAC_DFP v2.2.40, XMEGAD_DFP v2.3.50, and XMEGAE_DFP v2.2.47.

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.

MISRA-compliant headers (XC8-3279) To ensure that the compiler can conform to language specifications such as MISRA, the headers of the non-functional safety compiler have been updated to ensure that the `<builtins.h>` header file is automatically included by `<xc.h>`, that `builtin_avr_flash_segments` is defined only on architectures that support `__memx` (that is, all architectures except avrtiny), and that declarations of the builtins are provided in avr-libc.

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

Memory configuration based on fuse settings (XC8-3240) The toolchain is now aware of certain fuse settings specified in source code and automatically configures the available device memory (flash regions and RAM) based on these fuse values. The `-mfuse-action` option controls this action.

3.4. Version 2.45

Universal license manager (XC8-3175, 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, XC8-2951) The compiler binary files for macOS are now universal, offering native support for both Intel- and M1-based machines.

Reduced floating-point library sizes (XC8-3112, XC8-3071) Improvements have been made to floating-point library functions, including `sinf()`, `cosf()`, `tanf()`, `pow()`, `sqrt()`, `expf()`, `log1fp()`, and `nextafterf()`, which see a reduction in code size for these routines.

New device support Support is now available for the following AVR parts: AVR16EA28, AVR16EA32, AVR16EA48, AVR32EA28, AVR32EA32, AVR32EA48, AVR16EB14, AVR16EB20, AVR16EB28, and AVR16EB32.

3.5. Version 2.41

Bootrow support (XC8-3053) The compiler will place the content of any sections with a `.bootrow` prefix at address 0x860000 in the HEX file. These sections are used for BOOTROW memory, which is designed for the storage of keys and other secure information that should only be accessible to a bootloader.

Redundant return elimination (XC8-3048) The compiler will now eliminate redundant `ret` instructions in functions with a tail jump when custom linker scripts are used. This is a similar optimization to that performed previously, but it is now performed on all orphaned sections, even if a custom linker script is used and the best fit allocation scheme is not in play.

Time type change (XC8-2982, 2932) 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()`.

New nop (XC8-2946, 2945) The macro `NOP()` has been added to `<xc.h>`. This macro inserts a `nop` no-operation instruction into the output.

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.

Trampolined calls (XC8-2760) The compiler can now perform replacement of long-form call instructions with shorter relative calls when the relative forms of the instructions would normally be out of range of their destination. In this situation, the compiler will attempt to replace `call` with `rcall` instructions to a `jmp` instruction that will 'trampoline' execution to the required address. In the below example, there are two calls to `foo()` that end up relaxed to relative calls.

```
  rcall tramp_foo   ;was call foo
  ...
  rcall tramp_foo
  ...
tramp_foo:
  jmp foo
```

This optimization, along with similar program flow optimizations, is controlled by the `-mrelax` option.

3.6. Version 2.40

New device support Support is now available for the following AVR parts: AT90PWM3, AVR16DD14, AVR16DD20, AVR16DD28, AVR16DD32, AVR32DD14, AVR32DD20, AVR32DD28, AVR32DD32, AVR64EA28, AVR64EA32, and AVR64EA48.

Improved procedural abstraction The procedural abstraction (PA) optimization tool has been improved so that code containing a function call instruction (`call`/`rcall`) can be outlined. This will only take place if the stack is not used to pass arguments to nor obtain return value from the function. The stack is used when calling a function with a variable argument list or when calling a function that takes more arguments than there are registers designated for this purpose. This feature can be disabled using the `-mno-pa-outline-calls` option, or procedural abstraction can be disabled entirely for an object file or function by using the `-mno-pa-on-file` and `-mno-pa-on-function`, respectively, or by using the `nopa` attribute (`__nopa` specifier) selectively with functions.

Code coverage macro The compiler now defines the macro `__CODECOV` if a valid `-mcodecov` option is specified.

Memory reservation option The `xc8-cc` driver will now accept the `-mreserve=space@start:end` option when building for AVR targets. This option reserves the specified memory range in the either the data or program memory space, preventing the linker from populating code or objects in this area.

Smarter smart IO Several improvements have been made to the Smart IO functions, including general tweaks to the printf core code, treating the `%n` conversion specifier as an indepen-

dent variant, linking in vararg pop routines on demand, using shorter data types where possible for handling IO function arguments, and factoring common code in field width and precision handling. This can result in significant code and data savings, as well as increase the execution speed of IO.

3.7. Version 2.39 (Functional Safety Release)

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

3.8. Version 2.36

None.

3.9. Version 2.35

New device support Support is available for the following AVR parts: ATTINY3224, ATTINY3226, ATTINY3227, AVR64DD14, AVR64DD20, AVR64DD28, and AVR64DD32.

Improved context switching The new `-mcall-isr-prologues` option changes how interrupt functions save registers on entry and how those registers are restored when the interrupt routine terminates. It works in a similar way to the `-mcall-prologues` option, but only affects interrupt functions (ISRs).

Even more improved context switching The new `-mgas-isr-prologues` option controls the context switch code generated for small interrupt service routines. When enabled, this feature will have the assembler scan the ISR for register usage and only save these used registers if required.

Configurable flash mapping Some devices in the AVR DA and AVR DB family have an SFR (e.g. FLMAP) that specifies which 32k section of program memory will be mapped into the data memory. The new `-mconst-data-in-config-mapped-progmem` option can be used to have the linker place all `const`-qualified data in one 32k section and automatically initialize the relevant SFR register to ensure that this data is mapped into the data memory space, where it will be accessed more effectively.

Microchip Unified Standard Libraries All MPLAB XC compilers will share a Microchip Unified Standard Library, which is now available with this release of MPLAB XC8. The *MPLAB® XC8 C Compiler User's Guide for AVR® MCU* no longer includes the documentation for these standard functions. This information can now be found in the *Microchip Unified Standard Library Reference Guide*. Note that some functionality previously defined by avr-libc is no longer available. (See Library functionality.)

Smart IO As part of the new unified libraries, IO functions in the printf and scanf families are now custom-generated on each build, based on how these functions are used in the program. This can substantially reduce the resources used by a program.

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. (See smart-io-format Option for more details.)

Placing custom sections Previously, the `-Wl,--section-start` option only placed the specified section at the requested address when the linker script defined an output section with the same name. When that was not the case, the section was placed at an address chosen by the linker and the option was essentially ignored. Now the option will be honoured for all custom sections, even if the linker script does not define the section. Note, however, that for standard sections, such `.text`, `.bss` or `.data`, the best fit allocator will still have complete control over their placement, and the option will have no effect. Use the `-Wl,-T`*`section=addr`* option, as described in the user's guide.

3.10. Version 2.32

Stack Guidance Available with a PRO compiler license, the compiler's stack guidance feature can be used to estimate the maximum depth of any stack used by a program. It constructs and analyzes the call graph of a program, determines the stack usage of each function, and produces a report, from which the depth of stacks used by the program 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 AVR parts: ATTINY427, ATTINY424, ATTINY426, ATTINY827, ATTINY824, ATTINY826, AVR32DB32, AVR64DB48, AVR64DB64, AVR64DB28, AVR32DB28, AVR64DB32, and AVR32DB48.

Retracted device support Support is no longer available for the following AVR parts: AVR16DA28, AVR16DA32 and, AVR16DA48.

3.11. Version 2.31

None.

3.12. Version 2.30

New option to prevent data initialisation A new `-mno-data-init` driver option prevents the initialisation of data and the clearing of bss sections. It works by suppressing the output of the `do_copy_data` and `do_clear_bss` symbols in assembly files, which will in turn prevent the inclusion of those routines by the linker.

Enhanced optimizations A number of optimization improvements have been made, including the removal of redundant return instructions, the removal of some jumps following a skip-if-bit-is instruction, and improved procedural abstraction and the ability to iterate this process.

Additional options are now available to control some of these optimizations, specifically `-fsection-anchors`, which allows access of `static` objects to be performed relative to one symbol; `-mpa-iterations=`*`n`*, which allows the number of procedural abstraction iterations to be changed from the default of 2; and, `-mpa-callcost-shortcall`, which performs more aggressive procedural abstraction, in the hope that the linker can relax long calls. This last option can increase code size if the underlying assumptions are not realized.

New device support Support is available for the following AVR parts: AVR16DA28, AVR16DA32, AVR16DA48, AVR32DA28, AVR32DA32, AVR32DA48, AVR64DA28, AVR64DA32, AVR64DA48, AVR64DA64, AVR128DB28, AVR128DB32, AVR128DB48, and AVR128DB64.

Retracted device Support Support is no longer available for the following AVR parts: ATA5272, ATA5790, ATA5790N, ATA5791, ATA5795, ATA6285, ATA6286, ATA6612C, ATA6613C, ATA6614Q, ATA6616C, ATA6617C, and ATA664251.

3.13. 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, the `<builtins.h>` header file, which is automatically included by `<xc.h>`, has been updated. This header contains the prototypes for all in-built functions, such as `__builtin_avr_nop()` and `__builtin_avr_delay_cycles()`. Some built-ins may not be MISRA compliant; these can be omitted by adding the define `__XC_STRICT_MISRA` to the compiler command line. The built-ins and their declarations have been updated to use fixed-width types.

3.14. Version 2.20

New device support Support is available for the following AVR parts: ATTINY1624, ATTINY1626, and ATTINY1627.

Better best fit allocation The best fit allocator (BFA) in the compiler has been improved so that sections are allocated in an order permitting better optimization. The BFA now supports named address spaces and better handles data initialization.

Improved procedural abstraction The procedural abstraction optimizations are now performed on more code sequences. Previous situations where this optimization might have increased code size have been addressed by making the optimization code aware of the linker's garbage collection process.

Absence of AVR Assembler The AVR Assembler is no longer included with this distribution.

3.15. Version 2.19 (Functional Safety Release)

None.

3.16. 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.

The `#pragma nocodecov` may be used to exclude subsequent functions from the coverage analyzis. Ideally the pragma should be added at the beginning of the file to exclude that entire file from the coverage analyzis. Alternatively, the `__attribute__((nocodecov))` may be used to exclude a specific function from the coverage analyzis.

Device description files A new device file called `avr_chipinfo.html` is located in the `docs` directory of the compiler distribution. This file lists all devices supported by the compiler. Click on a device name, and it will open a page showing all the allowable configuration bit setting/value pairs for that device, with examples.

Procedural abstraction Procedural abstraction optimizations, which replace common blocks of assembly code with calls to an extracted copy of that block, have been added to the compiler. These are performed by a seperate application, which is automatically invoked by the compiler when selecting level `2`, `3` or `s` optimizations. These optimizations reduce code size, but they may reduce execution speed and code debugability.
Procedural abstraction can be disabled at higher optimization levels using the option `-mno-pa`, or can be enabled at lower optimization levels (subject to your license) by using `-mpa`. It can be disabled for an object file using `-mno-pa-on-file=`*`filename`*, or disabled for a function by using `-mno-pa-on-function=`*`function`*.
Inside your source code, procedural abstraction can be disabled for a function by using `__attribute__((nopa))` with the function's definition, or by using `__nopa`, which expands to `__attribute__((nopa,noinline))` and thus prevents function inlining from taking place and there being abstraction of inlined code.

Lock bit support in pragma The `#pragma config` can now be used to specify the AVR lock bits as well as the other configuration bits. Check the `avr_chipinfo.html` file (mentioned above) for the setting/value pairs to use with this pragma.

New device support Support is available for the following parts: AVR28DA128, AVR64DA128, AVR32DA128, and AVR48DA128.

3.17. 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.

Const objects in program memory The compiler can now place `const`-qualified objects in the program Flash memory, rather than having these located in RAM. The compiler has been modified so that `const`-qualified global data is stored in program flash memory and this data can be directly and indirectly accessed using the appropriate program-memory instructions. This new feature is enabled by default but can be disabled using the `-mno-const-data-in-progmem` option. For avrxmega3 and avrtiny architectures, this feature is not required and is always disabled, since program memory is mapped into the data address space for these devices.

Standard for free 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.

Welcome AVRASM2 The AVRASM2 assembler for 8-bit devices is now included in the XC8 compiler installer. This assembler is not used by the XC8 compiler, but is available for projects based on hand-written assembly source.

New device support Support is available for the following parts: ATMEGA1608, ATMEGA1609, ATMEGA808, and ATMEGA809.

3.18. Version 2.00

Top-level Driver A new driver, called `xc8-cc`, now sits above the previous `avr-gcc` driver and the xc8 driver, and it can call the appropriate compiler based on the selection of the target device. This driver accepts GCC-style options, which are either translated for or passed through to the compiler being executed. This driver allows a similar set of options with similar semantics to be used with any AVR or PIC target and is thus the recommended way to invoke the compiler. If required, the old `avr-gcc` driver can be called directly using the old-style options it accepted in earlier compiler versions.

Common C Interface This compiler can now conform to the MPLAB Common C Interface, allowing source code to be more easily ported across all MPLAB XC compilers. The `-mext=cci` option requests this feature, enabling alternate syntax for many language extensions.

New librarian driver A new librarian driver is positioned above the previous PIC `libr` librarian and the AVR `avr-ar` 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 or AVR 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.

4. Migration Issues

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

4.1. Version 2.50

None.

4.2. Version 2.49 (Functional Safety Release)

None.

4.3. Version 2.46

None.

4.4. Version 2.45

None.

4.5. Version 2.41

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.

4.6. Version 2.40

None.

4.7. Version 2.39 (Functional Safety Release)

None.

4.8. Version 2.36

None.

4.9. Version 2.35

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.

Inappropriate speed optimizations Procedural abstraction optimizations were being enabled when selecting level 3 optimizations (`-O3`). These optimizations reduce code size at the expense of code speed, so should not have been performed. Projects using this optimization level might see differences in code size and execution speed when built with this release.

Library functionality The code for many of the standard C library functions now come from Microchip's Unified Standard Library, which might exhibit different behaviour in some circumstances compared to that provided by the former avr-libc library. For example, it is no longer necessary to link in the lprintf_flt library (`-lprintf_flt` option) to turn on formatted IO support for float-format specifiers. The smart IO features of the Microchip Unified Standard Library makes this option redundant. Additionally, the use of _P suffixed routines for string and memory functions (e.g. `strcpy_P()` etc..) that operate on const strings in flash are no longer necessary. The standard C routines (e.g. `strcpy()`) will work correctly with such data when the const-data-in-program-memory feature is enabled.

4.10. Version 2.32

None.

4.11. Version 2.31

None.

4.12. Version 2.30

None.

4.1. Version 2.29 (Functional Safety Release)

None.

4.2. Version 2.20

Changed DFP layout The compiler now assumes a different layout used by DFPs (Device Family Packs). This will mean that an older DFP might no work with this release, and older compilers will not be able to use the latest DFPs.

4.3. Version 2.19 (Functional Safety Release)

None.

4.4. Version 2.10

None

4.5. Version 2.05

Const objects in program memory Note that the by default, `const`-qualfiied objects will be placed and accessed in program memory (as described here). This will affect the size and execution speed of your project, but should reduce RAM usage. This feature can be disabled, if required, using the `-mno-const-data-in-progmem` option.

4.6. Version 2.00

Configuration fuses The device configuration fuses can now programmed using a `config` pragma followed by setting-value pairs to specify the fuse state, e.g.

```
#pragma config WDTON = SET
#pragma config BODLEVEL = BODLEVEL_4V3
```

Absolute objects and functions Objects and functions can now be placed at specific address in memory using the CCI `__at(address)` specifier, for example:

```
#include <xc.h>
int foobar __at(0x800100);
char __at(0x250) getID(int offset) { ... }
```

The argument to this specifier must be a constant that represents the address at which the first byte or instruction will be placed. RAM addresses are indicated by using an offset of 0x800000. Enable the CCI to use this feature.

New interrupt function syntax The compiler now accepts the CCI `__interrupt(num)` specifier to indicate that C functions are interrupt handlers. The specifier takes an interrupt number, for example:

```
#include <xc.h>
void __interrupt(SPI_STC_vect_num) spi_Isr(void) { ... }
```

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.

5.1. Version 2.50

Scanf return value (XC8-3355) If the `[` conversion specifier (to match a nonempty sequence of characters from a set of expected characters) is used in format strings with the `scanf()` family of functions, the return value of those functions might be in error.

Printing length zero strings (XC8-3317) Calls to `snprintf()` with an "n" argument of 0 were writing characters to the string array, which should not have occurred. This has been corrected.

5.2. Version 2.49 (Functional Safety Release)

None.

5.3. Version 2.46

Missing chip information (XC8-3223) Chipinfo (HTML) files were not being generated for Device Family Packs (DFPs). The chipinfo files shipped with the compiler were present and correct.

EEPROM update failure (XC8-3150) Calls to the `eeprom_update_byte()` routine on AVR EA devices silently failed, leaving the addressed byte unchanged. This routine has now been corrected and the fix ships in the latest DFPs.

5.4. Version 2.45

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

Internal error with arrays of structures (XC8-3069) When multi-dimensional array members of a structure were processed, the address space qualifier was not correctly propagated to the array. This led to a mismatch in the address space qualifier information and an internal compiler error. This situation has been corrected.

Bad writes to uninitialized streams (ML-353, XC8-3100) If the standard output/error streams were not explicitly setup using `FDEV_SETUP_STREAM` or `_init_stdout`/`_init_stderr`, attempting to write to them resulted in undefined behavior. This also affected writes from stdlib functions, such as `perror()`. Any writes to these streams before they have been initialized will now be ignored.

Unsupported modifier (XC8-2505) The avr-libc library did not support the * modifier in printf-style conversion specifiers, for example "%.*f". This is now supported with the introduction of the Microchip Unified Standard Library.

Multiple uninitialized warnings (XC8-2409) The compiler was issuing multiple identical warning messages when encountering a const array that was not initialized. The message should have been issued just the once, which is now the case when this situation occurs.

5.5. 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.

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.

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 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 character 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.

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.

5.6. Version 2.40

Too relaxed (XC8-2876) When using the -mrelax option, the compiler was not allocating some sections together, resulting in less optimal code sizes. This might have occurred with code that used the new MUSL libraries or with weak symbols.

Mapping feature not disabled as stated in warning (XC8-2875) The const-data-in-config-mapped-progmem feature is reliant on the const-data-in-progmem feature being enabled. If the const-data-in-config-mapped-progmem feature was explicitly enabled using the option and the const-data-in-progmem feature was disabled, the link step failed, despite a warning

message stating that the const-data-in-config-mapped-progmem feature had been automatically disabled, which was not entirely correct. The const-data-in-config-mapped-progmem feature is now fully disabled in this situation.

DFP changes to correctly access NVMCTRL (XC8-2848) The runtime startup code used by AVR64EA devices didn't take into account that the NVMCTRL register was under Configuration Change Protection (CCP) and was not able to set the IO SFR to the page used by the const-data-in-config-mapped-progmem compiler feature. Changes made in AVR-Ex_DFP version 2.2.55 will allow the runtime startup code to correctly write to this register.

DFP changes to avoid flash mapping (XC8-2847) A work-around for a problem with the flash-mapping device feature reported in the AVR128DA28/32/48/64 Silicon Errata (DS80000882) has been implemented. The const-data-in-config-mapped-progmem compiler feature will not be applied by default for affected devices, and this change will appear in AVR-Dx_DFP version 2.2.160.

Build error with sinhf or coshf (XC8-2834) Attempts to use the `sinhf()` or `coshf()` library functions resulted in a link error, describing an undefined reference. The missing function referenced has now been included in the compiler distribution.

Build errors with nopa (XC8-2833) Using the `nopa` attribute with a function that has had its assembler name specified using `__asm__()` triggered error messages from the assembler. This combination is not possible.

Variadic function failure with pointer arguments (XC8-2755, XC8-2731) Functions with a variable number of arguments expect 24-bit (`__memx` type) pointers to be passed in the variable argument list when the const-data-in-progmem feature is enabled. Arguments that were pointers to data memory were being passed as 16-bit objects, causing code failure when they were ultimately read. When the const-data-in-progmem feature is enabled, all 16-bit pointers arguments are now converted to 24-bit pointers.

strtoxxx library functions failing (XC8-2620) When const-data-in-progmem feature was enabled, the `endptr` parameter in the strtoxxx library functions was not updated properly for source string arguments not in program memory.

Alerts for invalid casts (XC8-2612) The compiler will now issue an error if const-in-progmem feature is enabled and the address of a string literal is explicitly cast to data address space (dropping the `const` qualfiier), for example, `(uint8_t *) "Hello World!"`. A warning is issue if the address might be invalid when a `const` data pointer is explicitly cast to data address space.

Placement of uninitialized const objects (XC8-2408) Uninitialized `const` and `const volatile` objects were not being placed in program memory on devices that map all or part of their program memory into the data address space. For these devices, such objects are now placed in program memory, making their operation consistent with other devices.

5.7. Version 2.39 (Functional Safety Release)

None.

5.8. Version 2.36

Error when delaying (XC8-2774) Minor changes in the default Free mode optimizations prevented constant folding of operand expressions to the delay built-in functions, resulting in them being treated as non-constants and triggering the error: `__builtin_avr_delay_cycles expects a compile time integer constant.`

5.9. Version 2.35

Contiguous allocation using __at (XC8-2653) Contiguous allocation of multiple objects places in a section with the same name and using `__at()` did not work correctly. For example:

```
const char arr1[] __attribute__((section(".mysec"))) __at (0x500) = {0xAB, 0xCD};
const char arr2[] __attribute__((section(".mysec"))) = {0xEF, 0xFE};
```

should have placed `arr2` immediately after `arr1`.

Specifying section start addresses (XC8-2650) The `-Wl,--section-start` option was silently failing to place sections at the nominated start address. This issue has been fixed for any custom-named sections; however, it will not work for any standard sections, such as `.text` or `.bss`, which must be placed using a `-Wl,-T` option.

Linker crashes when relaxing (XC8-2647) When the `-mrelax` optimization was enabled and there were code or data sections that did not fit into the available memory, the linker crashed. Now, in such a circumstance, error messages are issued instead.

No no-falling-back (XC8-2646) The `--nofallback` option was not correctly implemented, nor documented. This can now be selected to ensure that the compiler will not fall back to a lower optimization setting if the compiler is unlicensed, and will instead issue an error.

Inappropriate speed optimizations (XC8-2637) Procedural abstraction optimizations were being enabled when selecting level 3 optimizations (`-O3`). These optimizations reduce code size at the expense of code speed, so should not have been performed.

Bad EEPROM access (XC8-2629) The `eeprom_read_block` routine did not work correctly on Xmega devices when the `-mconst-data-in-progmem` option was enabled (which is the default state), resulting in EEPROM memory not being read correctly.

Invalid memory allocation (XC8-2593, XC8-2651) When the `-Ttext` or `-Tdata` linker option (for example passed through using a `-Wl` driver option) is specified, the corresponding text/data region origin was updated; however, the end address was not adjusted accordingly, which could have led to the region exceeding the target device's memory range.

Crash with over-attributed function (XC8-2580) The compiler crashed if a function was declared using more than one of the interrupt, signal or nmi attributes, e.g., `__attribute__((__signal__, __interrupt__))`.

Invalid ATtiny interrupt code (XC8-2465) When building for ATtiny devices and the optimizations were disabled (`-O0`), interrupt functions may have triggered `operand out of range` assembler messages.

Options not being passed through (XC8-2452) When using the `-Wl` option with multiple, comma-separated linker options, not all of the linker options were being passed to the linker.

Error indirectly reading program memory (XC8-2450) In some instances, the compiler produced an internal error (`unrecognizable insn`) when reading a two byte value from a pointer to program memory

5.10. Version 2.32

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.

5.11. Version 2.31

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

5.12. Version 2.30

Global labels misplaced after outlining (XC8-2299) Hand-written assembly code that places global labels within assembly sequences that are factored out by procedural abstraction might not have been correctly repositioned.

A relaxing crash (XC8-2287) Using the `-mrelax` option might have caused the linker to crash when tail jump relaxation optimizations attempted to remove `ret` instruction that were not at the end of a section.

Crash when optimizing labels as values (XC8-2282) Code using the "Labels as values" GNU C language extension might have caused the procedural abstraction optimizations to crash, with an `Outlined VMA range spans fixup` error.

Not so const (XC8-2271) The prototypes for `strstr()` and other functions from `<string.h>` no longer specify the non-standard `const` qualifier on returned string pointers when the `-mconst-data-in-progmem` feature is disabled. Note that with avrxmega3 and avrtiny devices, this feature is permanently enabled.

Lost initializers (XC8-2269) When more than one variable in a translation unit was placed in a section (using `__section` or `__attribute__((section))`), and the first such variable was zero initialized or did not have an initializer, initializers for other variables in the same translation unit that were placed in the same section were lost.

5.1. Version 2.29 (Functional Safety Release)

None.

5.2. Version 2.20

Error with long commands (XC8-1983) When using an AVR target, the compiler may have stopped with a file not found error, if the command line was extremely large and contained special characters such a quotes, backslashes, etc.

Unassigned rodata section (XC8-1920) The AVR linker failed to assign memory for custom rodata sections when building for avrxmega3 and avrtiny architectures, potentially producing memory overlap errors

5.3. Version 2.19 (Functional Safety Release)

None.

5.4. Version 2.10

Relocation failures (XC8-1891) The best fit allocator was leaving memory 'holes' in between sections after linker relaxation. Aside from fragmenting memory, this increased the possibility of there being linker relocation failures relating to pc-relative jumps or calls becoming out of range.

Instructions not transformed by relaxation (XC8-1889) Linker relaxation did not occur for jump or call instructions whose targets become reachable if relaxed.

Missing <power.h> functionality (XC8E-388) Several definitions from `<power.h>`, such as `clock_div_t` and `clock_prescale_set()`, were not defined for devices, including the ATmega324PB, ATmega328PB, ATtiny441, and ATtiny841.

Missing macros The preprocessor macros `_XC8_MODE_`, `__XC8_VERSION`, `__XC`, and `__XC8` were not automatically defined by the compiler. These are now available.

5.5. Version 2.05

Internal compiler error (XC8-1822) When building under Windows, an internal compiler error might have been produced when optimizing code.

RAM overflow not detected (XC8-1800, XC8-1796) Programs that exceeded that available RAM were not detected by the compiler in some situations, resulting in a runtime code failure.

Omitted flash memory (XC8-1792) For avrxmega3 and avrtiny devices, parts of the flash memory might have been left un-programmed by the MPLAB X IDE.

Failure to execute main (XC8-1788) In some situations where the program had no global variables defined, the runtime startup code did not exit and the `main()` function was never reached.

Incorrect memory information (XC8-1787) For avrxmega3 and avrtiny devices, the avr-size program was reporting that read-only data was consuming RAM instead of program memory.

Incorrect program memory read (XC8-1783) Projects compiled for devices with program memory mapped into the data address space and that define objects using the PROGMEM macro/attribute might have read these objects from the wrong address.

Internal error with attributes (XC8-1773) An internal error occurred if you defined pointer objects with the `__at()` or attribute() tokens in between the pointer name and dereferenced type, for example, `char * __at(0x800150) cp;` A warning is now issued if such code is encountered.

Failure to execute main (XC8-1780, XC8-1767, XC8-1754) Using EEPROM variables or defining fuses using the config pragma might have caused incorrect data initialisation and/or locked up program execution in the runtime startup code, before reaching `main()`.

Fuse error with tiny devices (XC8-1778, XC8-1742) The attiny4/5/9/10/20/40 devices had an incorrect fuse length specified in their header files that lead to linker errors when attempting to build code that defined fuses.

Segmentation fault (XC8-1777) An intermittent segmentation fault has been corrected.

Assembler crash (XC8-1761) The `avr-as` assembler might have crashed when the compiler was run under Ubuntu 18.

Objects not cleared (XC8-1752) Uninitialized static storage duration objects might not have been cleared by the runtime startup code.

Conflicting device specification ignored (XC8-1749) The compiler was not generating an error when multiple device specification options were used and indicated different devices.

Memory corruption by heap (XC8-1748) The `__heap_start` symbol was being incorrectly set, resulting in the possibility of ordinary variables being corrupted by the heap.

Linker relocation error (XC8-1739) A linker relocation error might have been emitted when code contained a `rjmp` or `rcall` with a target exactly 4k bytes away.

5.6. Version 2.00

None.

6. Known Issues

The following are limitations in the compiler's operation. These may be general coding restrictions, or deviations from information contained in the user's manual. The bracketed label(s) in the title are that issue's identification in the tracking database. This may be of benefit if you need to contact support. Those items which do not have labels are limitations that describe *modi operandi* and which are likely to remain in effect permanently.

6.1. MPLAB X IDE Integration

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

Array debug information (XC8-3157) The debug information produced by the compiler does not accurately convey the object type for arrays in the `__memx` address space. This will prevent observation of the object in an IDE.

6.2. Code Generation

No unique section allocation (XC8-3387) The `-fdata-sections` option does not create a uniquely named section for an object defined as `const volatile` and using the `progmem` attribute. It is instead assigned to the same generic section that would have been selected if the `-fdata-sections` was not used.

Custom section allocation failure (XC8-3256) Section-ref based allocation is not ignoring sections mapped to a different region, and is not handling absolute addressed sections in the linker script as it would orphaned sections with an absolute address. This might result in a linker error when using the `-mrelax` option and custom output sections that are mapped to a region other than `text`.

Errno not set (XC8-3239) The `strtod()` function does not correctly set `errno` to `ERANGE` when appropriate.

Section overlap (XC8-3185) The best fit allocator might ignore any linker-script-defined output sections that are mapped to the data memory region, leading to a possible overlap of input sections with linker-script-defined sections. There is no indication from the compiler if sections overlap.

Segfault with section-anchors option (XC8-3045) Program that defined functions with variable argument lists and that use the `-fsection-anchors` option might have triggered a `internal compiler error: Segmentation fault`.

Debug info out of sync (XC8-2948) When linker relaxation optimizations shrinks instructions (for example `call` to `rcall` instructions), source line to address mappings might not remain in sync when there is more than one shrink operation occurring in a section.

PA memory allocation failure (XC8-2881) When using the procedural abstraction optimizers, the linker might report memory allocation errors when code size is close to the amount of available program memory on the device, even though the program should be able to fit the available space.

Not so smart Smart-IO (XC8-2872) The compiler's smart-io feature will generate valid but sub-optimal code for the `snprintf` function if the const-data-in-progmem feature has been disabled or if the device has all of its flash mapped into data memory.

Even less smart Smart-IO (XC8-2869) The compiler's smart-io feature will will generate valid but sub-optimal code when the `-flto` and `-fno-builtin` options are both used.

Suboptimal read-only data placement (XC8-2849) The linker is currently not aware of the APPCODE and APPDATA memory sections, nor the [No-]Read-While-Write divisions in the memory map. As a result, there is a small chance that the linker might allocate read-only data in an unsuitable area of memory. The chance of misplaced data increases if the const-data-in-progmem feature is enabled, especially if the const-data-in-config-mapped-progmem feature is also enabled. These features can be disabled if required.

Object file processing order (XC8-2863) The order in which objects files will be processed by the linker might differ based on the use of procedural abstraction optimizations (`-mpa` option). This would only affect code which defines weak functions across multiple modules.

Linker error with absolute (XC8-2777) When an object has been made absolute at an address at the start of RAM and uninitialized objects have also been defined, a linker error might be triggered.

Short wake-up IDs (XC8-2775) For ATA5700/2 devices, the `PHID0/1` registers are only defined as being 16 bits wide, rather than 32 bits wide.

Linker crash when calling symbol (XC8-2758) The linker might crash if the `-mrelax` driver option is used when the source code calls a symbol that has been defined using the `-Wl,--defsym` linker option.

Incorrect initialization (XC8-2679) There is a discrepancy between where the initial values for some global/static byte-sized objects are placed in data memory and where the variables will be accessed at runtime.

Bad indirect function calls (XC8-2628) In some instances, function calls made via a function pointer stored as part of a structure might fail.

Inaccurate stack advisor messaging (XC8-2542, XC8-2541) In some instances, the stack advisor warning regarding recursion or indeterminate stack used (possibly through the use of alloca()) is not emitted.

Failure with duplicate interrupt code (XC8-2421) Where more than one interrupt function has the same body, the compiler might have the output for one interrupt function call the other. This will result in all call-clobbered registers being saved unnecessarily, and the interrupts will be enabled even before the epilogue of the current interrupt handler has run, which could lead to code failure.

Bad output with invalid DFP path (XC8-2376) If the compiler is invoked with an invalid DFP path and a 'spec' file exists for the selected device, the compiler is not reporting the missing device family pack and instead selecting the 'spec' file, which might then lead to an invalid output. The 'spec' files might not be up to date with the distributed DFPs and were intended for use with internal compiler testing only.

Memory overlap undetected (XC8-1966) The compiler is not detecting the memory overlap of objects made absolute at an address (via `__at()`) and other objects using the `__section()` specifier and that are linked to the same address.

Failure with library functions and __memx (XC8-1763) Called libgcc float functions with an argument in the `__memx` address space might fail. Note that library routines are called from some C operators, so, for example, the following code is affected:

```
return regFloatVar > memxFloatVar;
```

Limited libgcc implementation (AVRTC-731) For the ATTiny4/5/9/10/20/40 products, the standard C / Math library implementation in libgcc is very limited or not present.

Program memory limitations (AVRTC-732) Program memory images beyond 128 kb are supported by the toolchain; however, there are known instances of linker aborts without relaxation and without a helpful error message rather than generating the required function stubs when the `-mrelax` option is used.

Name space limitations (AVRTC-733) Named address spaces are supported by the toolchain, subject to the limitations mentioned in the user's guide section Special Type Qualifiers.

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.