About EW430

Support Overview

The following compiler/environment versions are supported:

- IAR EW430 (Embedded Workbench for MSP430) 4.2x (C and C++ static analysis only).
- IAR EW430 (Embedded Workbench for MSP430) 5.3x (C-only).
- IAR EW430 (Embedded Workbench for MSP430) 5.4x (C++ supported for static analysis only)
- IAR EW430 (Embedded Workbench for MSP430) 6.1x (C and C++ static analysis only)

IAR EW integration is provided as follows:

- Project importing is only possible using the `cpptesttrace` utility (see Importing Projects, for more info).

The following components are provided to facilitate testing IAR Embedded Workbench projects:

- Compiler configurations for specific supported versions of IAR compiler for MSP430 (listed above).
- "IAR_icc430.mk" Runtime Library Build Configuration file for building the Runtime Library using 'make'.
- Test Configurations prepared for launching Unit Testing on C-SPY Simulator:
  - Run IAR MSP430 Tests - a pure manual simulator configuration.
  - Run IAR MSP430 Application with Mem Monitoring - a pure manual simulator configuration.

Known Limitations

Placing IAR memory attributes in front of pointer declarations is not fully supported. In the following example, *Declaration A* and similarly written declarations will be analyzed as if they were written like *Declaration B*:

Declaration A

```
__data20 int * ptr;                    /* 'ptr' variable in 'data20'
                                       memory; points to an integer in
                                       default memory */
```

Declaration B

```
int __data20 * ptr;                    /* 'ptr' variable in default
                                       memory; points to an integer in
                                       'data20' memory */
```

This limitation affects projects with the following configuration:

- '430X' target processor core
- medium or large data memory model
- enabled IAR language extensions.

The following declarations are handled correctly and may be used instead of unsupported construction mentioned above:

```
int * __data20 ptr1;                   /* 'ptr1' variable in 'data20'
                                       memory; points to an integer in
                                       default memory */

int __data20 * __data16 ptr2;          /* 'ptr2' variable in 'data16'
                                       memory; points to an integer in
                                       'data20' memory */
```

- By default, UBROF libraries and object files are produced with the same extension (.r?? and .r43 for EW430 5.3x-5.4x). This prevents C++test from distinguishing between objects and libraries, which causes the user libraries to be removed from the linking command-line. The following sections provide a more detailed description and solutions to this limitation.
- Versions of Xlib shipped with IAR compilers for MSP430 versions 5.3x-5.4x have a known issue that may cause failures during C++test's LSI phase. Xlib will generate the following message if the problem occurs: "Error [14]: Unknown tag: E0". Update Xlib to a newer version to resolve this issue by updating xlink to version 5.2.6.19 or later. Please contact IAR support for information on performing this action.

Why User Libraries May Be Removed During Linking

C++test uses an option-filtering mechanism to substitute instrumented objects for original user objects in linking command lines. In formats like ELF and COFF, the object and library file extentions differ by default (.o/.a, .obj/.lib), which C++test depends on to properly emply the option-filtering mechanism. UBROF libraries and object files, however, have the same extension (.r??) by default, which results in C++test removing libraries from linking command-lines. This was solved for system libraries by introducing an option-filtering pattern based on their observed systematized naming convention (dl430*.r43). User libraries, however, are still removed.

Preventing Libraries From Being Removed During Linking

There are two solutions to this problem:

1. Right-click on the project and choose **Properties> Parasoft> C++test> Build Settings** and manually add your libraries in the **Linker options** field in the options section (recommended).
2. Establish a library naming scheme (extension or name pattern) and create a Custom Compiler Configuration.

If you choose option #2, insert the following option-filtering rule into the c/cpp.psrc files:

```
edgtk.optionConfig name=*<your_pattern> regexp=true casesensitive=false tags=linker
```

just before

```
edgtk.optionConfig name=*.r43 regexp=true casesensitive=false

tags=linker,excludable,object_file
```

This issue also applies to externally built C++test Runtime Libraries. To learn more about Custom Compilers, see Configuring Testing with the Cross Compiler.