# Architecture/Toolchain-independent Information

## Support Overview

IAR EW integration is provided as follows:

- Project importing and options extraction facilities depend on toolchain (EWARM, EW430, EWSTM8 etc.) and its version. For more information refer to toolchain-specific sections below.

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

- Test Configurations for launching Unit & App Testing on C-SPY Simulator:
  - Run IAR EW Tests (Batch Template) - Uses EW-generated batch scripts (.cspy.bat) to launch simulator.
  - Run IAR EW Application with Mem Monitoring (Batch Template) - uses EW-generated batch scripts (.cspy.bat) to launch simulator.
- Test Flow recipes  (associated with Test Configurations) that have integrated test results reading:
  - Build and run tests on IAR C-SPY Simulator.
  - Build and run tests on IAR C-SPY Simulator using EW-generated batch scripts (.cspy.bat).
  - Build and run application on IAR C-SPY Simulator.
  - Build and run application on IAR C-SPY Simulator using EW-generated batch scripts (.cspy.bat).
- Following published properties are available in IAR-specific Test Configurations/Flows:
  - "Target architecture" (arch) - used in non-"(Batch Template)" Test Configurations.
  - "C-SPY execution backend options" (bkend_opts) - required to correctly execute the Test Binary on target; used in non-"(Batch Template)" Test Configurations.
  - "Original IAR EW project folder name" (prj_dir_name) - the name of the project file direct parent folder; defaults to original project name; must be set to locate the EW-generated batch script; used in "(Batch Template)" Test Configuration versions.
  - "Original IAR EW project name" (prj_name) - provided in case the Eclipse project name doesn't match the original one; used in "(Batch Template)" Test Configuration versions.
  - "IAR EW project Build Configuration (leave empty for EW < 7.0)" (bld_cfg) - the name of EW project build configuration to be used (e.g., "Debug" or "Release"); must be set for more recent EW versions to locate the EW-generated batch script used for running applications in IAR C-SPY Simulator (because for that versions the file name of EW-generated batch script contains the name of EW project build configuration), for older EW versions should be left empty; used in "(Batch Template)" Test Configurations.
  - "Adjust CSpyBat command (enable only for IAR EW < 6.1)" (adj_cspy_cmd) - specifies if EW-generated batch script should be adjusted using special additional adjusting step; for more recent EW versions such adjustment is not required and the value should be left to "false" (the default); used in "(Batch Template)" Test Configurations.
  - "Driver DLL" (drv_dll) - the hardware debugger driver DLL required by CSpyBat; by default this is "<arch>sim.dll" (e.g., "430sim.dll" for EW430, "rxsim.dll" for EWRX); for EWARM 5.3 and 5.4, the default value ("armsim.dll") is correct, but it must be changed to "armsim2. dll" for newer versions of EWARM (EWARM 5.4 contains both DLLs); used in non-"(Batch Template)" Test Configurations".
  - "Flash loader specification file" (flash_loader) - the flash-loader specification file to pass to CSpyBat; depending on EWARM version this may be an '.xml' or a '.board'/ '.flash' file (builtin configs are available in "EW_DIR/arm/config/flashloader"); used in all IAR-specific Test Configurations/Flows; set it when using flash-loader and non- "(Batch Template)" Test Configurations, when using "(Batch Template)" Test Configuration versions set it only for EWARM 5.3 due to issue with this EWARM (don't set it for newer EWARMs; this will trigger the duplicate flash-loader specification error).

There are more Test Configurations mentioned in toolchain-specific sections. These are configured manually and intended only as a backup solution when EW-generated C-SPY batch script isn't available or in other problematic cases. Please use the forementioned "Batch Template" Test Configurations whenever possible.

Standard test configurations, such as "Generate Unit Tests", "Generate Stubs", etc., can also be used for IAR projects and are recommended to help ensure code quality.

**Support for Environments with Multiple IAR EW Installations**

Starting with C++test 9.1, extensive changes have been applied to support environments where multiple IAR EW installations co-exist. The information about which IAR version/installation to use is crucial for enabling C++test to correctly build and test your projects.

IAR does not recommend using internal project data to obtain information about IAR version/installation, so it must be deduced by from the IAR Windows registry entries and other primary information, such as EW_DIR or PATH environment variable values. We strongly recommend setting the EW_DIR variable to the EW installation folder path and allow the other method to operate as a C++test backup mechanism.

C++test uses these versioning methods both when importing projects and when scanning '.ewp' project files. With each method and environment setting, C++test will check the availability of chosen/deduced EW installation in the registry; if it is not properly registered, C++test will not be able to work with it.

During a single session (application run), C++test can work with only the single chosen EW version/installation. To change this, you need to restart C++test with the modified environment.

# Requirements

- Compiler and C-SPY executables must be added to the PATH variable.
- Runtime Library Build Configuration files require the EW_DIR environment variable to be available and set to the EW install location.
- "Build and run tests on IAR C-SPY Simulator" and  "Build and run application on IAR C-SPY Simulator" Test Flow recipes require the EW_DIR environment variable to be available and set to the EW install location and Simulator back-end options to be provided manually (by editing Test Flow properties).

- For more recent EW versions "Build and run tests on IAR C-SPY Simulator using EW-generated batch script template" and "Build and run application on IAR C-SPY Simulator using EW-generated batch script template" Test Flow recipes require the name of EW project build configuration to be set (using "IAR EW project Build Configuration" Test Flow property).
- The selection/configuration of C++test's Runtime Library features places certain requirements on the configuration of IAR's C/C++ runtime library (DLib). You must ensure that these require-ments are met; most importantly, transporting of test results using the file channel (the default) and post-conditions/assertions messaging require the DLib to provide "stdio"-compatible FILE and printf interfaces. Thus, you should use the "Full" "Semihosted" DLib configuration or use the "Custom" and ensure that appropriate features are available.

# Known Limitations

- There is no EW-shipped environment setup script, so you must either rely on the environment variables set by the EW installation program or set up the environment by other means.
- IAR extensions related to IAR memory attributes (e.g. '__data', '__data20', '__tiny', '__near_func') are not fully supported. This limitation may lead to errors during analysis, such as parse errors or instrumented code compilation errors. The limitation may also lead to code being analyzed as if it contained memory attributes different than the attributes in the original code.
- IAR C++ extensions of the following algorithms are unsupported:
    - class template partial specialization matching
    - function template parameter deduction

  Using IAR memory type attributes (e.g. '__code', '__data', '__data20', '__near_func') in conjuction with C++ templates may lead to the incorrect memory type attribute being substituted for the template parameter. This limitation does not impact EC++ because templates are not available in that mode. Because the IAR template-related C++ extensions mentioned in this section are used extensively in IAR DLIB headers, C++test may not accept code that uses IAR STL headers or may analyze the code imprecisely.
- Function signatures reported by C++test (e.g., in the Coverage view) may lack parts related to IAR type attributes. For example, coverage for the v `oid f(int __data16 *)` and `void f(int __data24 *)` functions may be displayed as coverage for two different function with the same v `oid f(int *)` signature.
- In some special situations, C++test may not accept analyzed code if IAR language extensions are disabled. You can bypass this limitation by enabling IAR language extensions. This can be done, for example, in the project settings inside EW or by adding the `-e` compiler option to C++test compiler options for the project:
    1. Select the project and choose **Project> Properties> Parasoft> C++test> Build Settings**
    2. Enter `-e` in the Compiler options field and click **Apply**.
- C++test will analyze C++ code in EEC++ (Extended Embedded C++) mode, regardless of the C++ mode used in the original project or build, if you use the IAR compiler configurations shipped with C++test. This limitation does not affect C++test support for EWSTM8. You can adjust C++test IAR compiler configuration to bypass this limitation by doing the following:
    1. Create a Custom Compiler Configuration (see Configuring Testing with the Cross Compiler, to learn more about Custom Compiler Configurations).
    2. In the `gui.properties` file, locate the line that begins with `cppCompilerCmdLine=` and change the `--eec++` compiler option to the required option. This line may have the following appearance:
       ```
       cppCompilerCmdLine=$(exe) $(filtered_opts) --eec++
       -I $(CPPTEST_INCLUDE_DIR) $(input) -o $(output)
       ```
       If the `gui.properties` file contains a line that begins with `cppCompilationModeOption=`, then you must also change the `--eec++` compiler option to the required option. This line may have the following appearance:
       ```
       ppCompilationModeOption=--eec++
       ```
    3. Make the following changes to the `cpp.psrc` file:

       Locate the lines that begin with `edgtk.preprocessorCommand` and `edgtk.gccAutoconfiguratorCommand` and change the `--eec++` compiler option to the required option. The lines may have the following appearance:
       ```
       edgtk.preprocessorCommand {exe} --preprocess=nl {out} {opts} --eec++ {in}
       edgtk.gccAutoconfiguratorCommand {exe} --preprocess=nl {tmpout} {opts} --eec++ --silent --
       predef_macros {out} {in}
       ```
- Support for compiler pragma directives is limited. Although pragma directives are accepted and generally retained in tested code during analysis, they are not interpreted. In some special situations, source code that contains compiler pragma directives may be analyzed as if the directives were not present or located in different places. This may lead to errors during analysis or imprecise analysis results. For dynamic analysis and projects containing C++ files, compilation warning messages related to pragma directives (e.g. `inline` or `function_effects`) used inside compiler header files may be printed to the C++test console. In most cases, these warnings indicate that the optimization level of tested code is different than the optimization level of the same code in the original project and can be ignored.
- IAR compilers provide several C++ modes (i.e., EC++, EEC++, C++, etc.) that you can enable with command line options. Enabling a C++ mode forces all source files present in compiler command line to be treated as C++ files compiled in the enabled mode, regardless of their filename extension. When creating a C++test project from a .bdf file that contains C++ mode selection options, C++test may select additional directories (such as `bin` or `inc`) from a compiler installation directory and treat them as linked folders. As a result, your analysis scope may contain additional files, which may lead to performance issues and unexpected messages in the C++test console. Remove the unwanted linked folders from the C++test project to avoid these issues. You can also prevent C++test from selecting additional directories when creating a new project from a .bdf file. For example, in the GUI:
    1. Choose **File> New> Project> Create project from a build data file**
    2. Choose a .bdf file and verify the settings
    3. Click **Next** and expand project contents to find unwanted folders
    4. Right click on a folder choose **Remove folder(s)** from the context menu
- The following keywords are not supported:.
    - Special IAR '__no_alloc_str()' and '__no_alloc_str16()' built-in operators.
    - Special C-RUN operators like '__as_get_base()', __as_get_bounds()' and '__as_make_bounds()'.

  Analyzing the code that contains usage of these IAR extended keywords will lead to parse errors during analysis.
- The following keywords have partial support:
    - Special IAR '__no_alloc' and '__no_alloc16' keywords.

  Analyzing the code that contains usage of these IAR extended keywords may lead to errors during analysis, such as parse errors or instrumented code compilation errors.
- For information about known limitations for various supported target architectures please refer to toolchain-specific sections below.