QNX Momentics Plug-in

This topic explains installation and basic setup instructions for the C++test plugin into QNX Momentics. In this section:

Introduction

Use the Eclipse plug-in mechanism to integrate C++test into your QNX Momentics IDE. A link file will be added to the following location:

```
<QNX Installation Root>/QNX Momentics/com.parasoft.xtest.cpptest.link.
```

The link file contains the path to the C++test plugin, which informs the QNX Momentics IDE that the plug-in should be loaded at startup. After installation, a dedicated C++test perspective will be added in Workbench. C++test views will also be added to the C/C++ perspective.

Prerequisites

- QNX Software Development Platform (see IDE Support for supported versions)
- See Compilers for supported hosts and additional requirements specific to your compiler.

Installation

1. Run the C++test plugin installation executable.
2. Choose the setup language.
3. Click **Yes** when a dialog asks whether you want to install C++test.
4. Click **Yes** after you have read and agreed to the license information.
5. Choose the QNX target platform, then click **Next**.
6. Enter your QNX Momentics installation directory (if not detected automatically), then click **OK**.
7. Enter the desired destination directory for the C++test Extension files, then click **Next**.
8. Close the IDE if it is open, then click **OK** to close the dialog reminding you to close this program. C++test will then start copying files and installing the necessary files into Momentics. A dialog box with a progress indicator will open and indicate installation progress. When the installation is complete, a notification dialog box will open.
9. Click the **OK** button to close the notification dialog box.

During the installation process, C++test attempt to create a link file in the following QNX Momentics IDE's embedded Eclipse location:

```
<QNX Installation Root>/QNX Momentics/com.parasoft.xtest.cpptest.link
```

The link will contain the path to the C++test installation directory. This is the only change that the C++test installation makes inside the QNX Software Development platform distribution.

Launch the QNX Momentics IDE (with the C++test plugin installed) as you normally would.

Uninstalling

Use the Control Panel's **Add or Remove Program** functionality to uninstall the C++test QNX Momentics IDE plug-in. Alternatively, you can run the Windows C++test installation executable and enable the **Remove** option.

You cannot uninstall the C++test plug-in by removing the `<QNX Installation Root>/QNX Momentics/com.parasoft.xtest.cpptest.link` lin k file. If you remove the file, you will still need to manually remove the C++test installation. The Windows registry's Install Shield information will still remain.

Licensing

You can run the C++test plug-in using a local (machine-locked) license or a network license served from Parasoft DTP or from a standalone instance of Parasoft License Server. See Licensing for details on how to configure the license.

Working with QNX Projects

In order to perform static analysis or unit testing, C++test must collect the following information from your QNX project:

- Paths to source files
- Compilation command lines
- Linker command lines

The information is collected from the QNX Momentics IDE project structure description by scanning make files.

Scanning Make Files

1. Select your project and run a C++test test configuration.
2. The build process starts, but instead of using the actual compilation/linking command lines, the make file executes the command lines prefixed with the option scanning utility. The command line is generally modified as follows:
    - Original compilation command line: `qcc <options> <file>`
    - C++test prefixed compilation command line: `options_scanner <options> qcc <options> <file>`
3. The option scanner analyzes the command line and stores the compiler/linker options for further reuse.

After the build process is completed, C++test will have all the necessary data collected.

Project Configuration

Before you start testing, ensure that the project's compiler/linker options source is correctly set. In most cases, C++test automatically detects the necessary settings. The compiler/linker flags source settings are specified in the Build Settings screen. To access the build settings:

1. Right-click the project's Navigator node and choose **Properties** from the shortcut menu.
2. Expand the **Parasoft> C++test** category in the left pane and choose the **Build Settings** category.

The following configuration options are available:

Options Source

Choose a strategy for collecting compiler/linker switches from the drop-down menu.

For testing of QNX projects directly, choose **Use Options from QNX build system**. The C++test options extractor is designed to scan QNX make files. The compiler/linker executable name, compiler/linker command lines, and the system environment used to start the compilation/linking process are scanned. All these settings are later used during the C++test analysis.

Project Settings

- **Build command line:** Enter a command line that will launch your build system. This system can be abstract, but by default, the make-based one is assumed and the appropriate command is preset. This command executes your make on the project's Makefile to scan the project's compilation and linking options. Thus, a special scanner (hidden behind the ${CPPTEST_SCAN} macro) is substituted for C/C++ compilers and the linker; the -i and -B make options are very useful (see Accounting for Make Varieties for more information).
- **Build working directory:** Specify the directory from which the build command is launched.
- **Dependency file(s):** Specify all files that should be checked for changes each time a test/build action is performed. If one of these files is found to be modified, the build command will be reexecuted. You should enter all files that are sources of project's options (or influence them). Typically, this is just your Makefile.
- **Reset cache** button: Lets you clear all scanned options and force rescanning on the next test action (see Accounting for Make Varieties for exceptions).

Compiler Settings

The Compiler settings area lets you specify the compiler set/tools used for compiling the project's sources and building the test executable. Settings include:

- **Family:** Select the appropriate family from all the currently registered compiler families. Use QNX® GCC 5.x.
  Click **Autodetect** to allow C++test to automatically detect the compiler family based on the compiler executables and compiler version regular expression that is stored in the compiler's configuration directory. Autodetect may not always produce the expected results because of the wide variety of compilers used with embedded solutions.
- **C compiler**: Specify the C compiler executable.
- **C++ compiler:** Specify the C++ compiler executable.
- **Linker:** Specify the linker executable.

Options

Specify any additional compiler or linker options you want to include, as well as the location of the runtime library.

- `-DPARASOFT_CPPTEST` is added to the compiler options by default. You can add any additional options after that.
- You can add options, but not remove them (unless you can overwrite the original options with negatives, such as undefine symbols, etc.).
- The `${cpptest:original_options}` variable can be used to access the original project options.

Also see Specifying Options in the C++test Project or File Options Panel.

Static Analysis

Before you start static analysis, verify that the appropriate compiler settings are used in the Project Properties panel's **Parasoft> C++test> Build Settings** area. In most cases, it is best to set the Options source as **Use Options from QNX build system**. After confirming that compiler settings are correct, you can start the analysis.

> **Learning More**
>
> For general information on performing static analysis with C++test, see Static Code Analysis and Flow Analysis.

Runtime Testing

This section covers both Unit Testing and Application Monitoring.

Unit Testing

Building the C++test Runtime Library for Testing QNX Projects

For embedded testing, the C++test runtime library must be cross-compiled for the chosen platform before you can perform unit testing. The built-in "Run QNX Momentics Tests" Test Configuration does this building automatically.

If you need to build C++test runtime library manually, follow the instructions in Working with the C++test Runtime Library — using `QNX_5.mk` as the target configuration.

Test Configuration for QNX Unit Testing

C++test provides a Test Configuration template designed specifically for testing QNX Projects. This Test Configuration is in Builtin> Embedded Systems> QNX> Run QNX Momentics Tests. It covers all steps required for full unit testing of QNX Projects:

- Building the C++test runtime library
- Building the test executable
- Executing the test executable on remote QNX system
- Collecting results

This Test Configuration is only a template; it needs to be customized to reflect your remote QNX system configuration as follows:

1. Choose **Parasoft> Test Configurations**.
2. Open the **Builtin> Embedded Systems> QNX** tree node.
3. Right-click **Run QNX Momentics Tests** and choose **Duplicate**.
4. Select the duplicated Test Configuration that now appears under **User-defined**.
5. Open the **Execution** tab
6. Modify the Test Configuration properties to reflect your remote QNX system configuration:
    - **QNX target:** Target host address
    - **QNX target test directory:** Directory on target host where test will be executed
    - **QNX target user name:** User name that will be used to connect to target host

This test configuration is based on rcp (allow remote copy) and rsh (allow remote shell execution) tools available in Windows. The remote QNX system needs to be configured to allow remote calls from rcp and rsh. It is also possible to modify test execution flow and use different tools instead of rcp and rsh.

Debugging Test Cases

Use Eclipse Internal debugging mode. For more details see:

- Configuring Debugger Settings.
- Debugging in Various Embedded Development Environments.

> **Learning More**
>
> For general information on generating and executing unit test cases with C++test, Test Creation and Execution.

Application Monitoring

Test Configuration for QNX Application Monitoring

C++test provides a Test Configuration template designed especially for running a QNX application with memory monitoring: **Builtin> Embedded Systems> QNX> Build and Run Application with Memory Monitoring for QNX Momentics**. This Test Configuration provides all steps required for running QNX application monitoring, including:

- Application building
- Executing the application on a remote QNX system
- Collecting results

This Test Configuration is only a template; it needs to be customized to reflect your remote QNX system configuration. To customize it:

1. Choose **Parasoft> Test Configurations**.
2. Expand **Builtin> Embedded Systems> QNX**.
3. Right-click **Build and Run Application with Memory Monitoring for QNX Momentics** and choose **Duplicate** from the shortcut menu.
4. Select the new Test Configuration that is added to the **User-defined** category.
5. Open the **Execution** tab
6. Modify the Test Configuration properties to reflect your remote QNX system configuration:
    - **QNX target**: The target host address.
    - **QNX target test directory:** The directory on the target host where the application will be executed.
    - **QNX target user name:** The user name that will be used to connect to target host
    - **QNX target PHOTON environment variable value:** For GUI applications, you need to specify this value in order to display your application GUI in phindows session.

This Test Configuration is based on scp (allow remote copy) and rsh (allow remote shell execution) tools available in Windows. The remote QNX system needs to be configured to allow remote calls from rcp and rsh. It is also possible to modify test execution flow and use different tools in place of rcp and rsh.

Learning More

For general information on performing application monitoring and runtime error detection with C++test, see Runtime Error Detection.