Projects, Build, and the IDE#

This article describes how to set up projects in the QNX Momentics IDE, what types of projects to pick, and how to organize projects to perform effective builds from the IDE and command line. This article is based on IDE 4.5.1 and some IDE 4.6 features.

What do you need to know about the IDE project model?#

The QNX Momentics IDE is based on the open-source Eclipse platform and the open-source CDT project, which give the essential part of the functionality of projects and builds. The IDE uses the concept of a *workspace*, which is a user-specific writable directory on the local host. This directory itself should never be part of a version-control system shared between users, nor should it be located on a shared drive (unless you're sure only one user is using it). The IDE also uses the concept of a *project*, a container for source and binaries together with some configuration files, which usually is located in your workspace, and which can be shared between users using a version-control system.

Projects are flat; they cannot contain one another. However there is a concept of a *Working Set*, which lets you filter and group projects if you have too many of them in a workspace. There is also a special QNX Container project that lets you control and build sets of projects at the same time.

When you pick the location of your workspace and the names of the projects, be aware that these names can be used in the build, and **make** doesn't like directory and file names with spaces and funky characters in them, even it is fine with the IDE -- the build won't work with such paths.

If you've used Visual Studio before, you may think that a project is a virtual container that contains arbitrary files and directories, but this isn't the case with Eclipse. You can stretch an Eclipse project to use "Eclipse Links", but they have limited functionality. In general, you have to have a directory in the filesystem that contains the project root (for source and build output), and the same directory would contain the Eclipse project metadata. If you want to separate the project metadata from the source directories, you have to use folder links. You have an option to put a project inside or outside your workspace. You can use your imagination about how a project directory is created: you can check out the top level from one place, and subdirectories from another, you can use OS soft links, or some other means to create it.

The QNX Momentics IDE supports 3 project types: Makefile projects, QNX projects, and Managed projects.

Makefile projects#

A Makefile project would work for any project that has a Makefile (by default). Technically it can launch anything as an external builder in any folder. The IDE starts **make** ; after **make** exits, the IDE refreshes the workspace to show what you have created. You can change the **make** command and/or run specific **make** targets, but the IDE has no control over what **make** is doing.

Because the IDE doesn't know what it's building, it would have problems parsing source files (which it does internally to allow Navigation, Code Completion, Syntax Highlighting, Code Generation and Refactoring). So if you use a Makefile project, you have to tune the Indexer (the internal parser) to point it to where to find any missing **#includes** and what **#defines** your parser uses for conditional compilation. The process of figuring this out is called *Discovery* and can be controlled using the Discovery Options. If you know what includes and defines you're using, it's probably easier just to enter them directly (via **Project->Properties->C/C++ General -> Path and Settings**).

QNX projects#

A QNX project is special a flavor of a Makefile project with additional control over the **make**. To use a QNX project, you also have to use QNX recursive makefiles. QNX recursive makefiles follow certain conventions

for creating makefiles that use specific variables and use a specific layout (for details, see the Conventions for Recursive Makefiles and Directories chapter of the QNX Neutrino *Programmer's Guide*). These conventions allow the IDE to parse the makefile and provide UI control over makefile options and build variants. You can typically use a single QNX project to build one binary/library for several variants, such as x86 and PPC in debug, release, and profiling modes.

Managed projects#

A Managed project is a CDT-specific project that doesn't use makefiles, and all build settings are controlled by the UI. The inconvenience of it is its inability to perform a build of the project from the command line (although it *is* possible in simple cases with some extra setup files, or you can use a makefile generator). Also there are restrictions on what you can build and how, especially if you use special steps in the the build that involve other tools.

Original project creation#

Let's consider some scenarios where you'd create a project for the first time (in comparison to checking out a pre-made project, which we'll discuss later).

When you create a new IDE project, you have to pick from one of the following options:

1. This is new project, and you intend to create all the source in the IDE.
2. The source/structure exists already in the filesystem, and you want to "attach" an IDE project to it.
3. The source/structure exists in a version-control system, but not as an IDE project.

New Project (#1): Pick one of the project types described above. Use the **File->New...** menu, pick **C or C++ Project**, and then:

- For a QNX project, pick **QNX C Project** (or **QNX C++ Project** for C++). On the first page, pick the build variants (for example, x86 Debug & Release).
- For a Makefile project, pick **C Project** (or **C++ Project** for C++). Select **Makefile** on the left. Pick **QNX Toolchain** on the right. Click **Finish**.
- For Managed project, pick **C Project** (or **C++ Project** for C++). Select one of the project types or templates on the left, other than **Makefile**. Pick **QNX Toolchain** on the right. Click **Finish**.

Attaching to an existing folder(#2): Pick one of the project types described above for your project. Open the corresponding wizard as described in #1, but don't proceed any further. The first wizard page asks you to choose between using the default location and picking one. Uncheck **Use default location**. Select the location of your existing project using the **Browse** button. Follow the wizard as in #1. Alternatively, you can create the project in the default location and later attach other directories using link folders. See Example 4, below.

Checking out from source control(#3): Pick one of the project types described above for your project. If the whole project is in one directory in the version-control system, you can use the **Check Out As...** action of the SVN or CVS plugin to check it out. Use **Check out as a project configured using New Project Wizard** and pick the wizard of your choice. For a QNX project, make sure you uncheck **Generate default file** and **Generate Makefiles** (these aren't in version 4.5; you need to revert to the base to restore makefiles after checking out). If you want a partial checkout, see the next section.

Checking out a partial source tree#

Here's how to create a project by checking out from several folders from a version-control system (this example uses SVN).

Let's say you have a folder in svn called **big_project**, and it has 100 subfolders, each of them representing different binaries, but **big_project** has a Makefile and some other common folders, such as "public_includes" that you need to compile your subfolder **my_binary**.

- Follow the instructions for checking out a folder as a new project (**big_project**) but unselect **Checkout recursively** (SVN can't check out one file; it has to be the whole folder).
- Now find the **my_binary** subfolder in SVN, right-click and choose **Find/Check Out As...**, and then select **Check out as folder into existing project**. Click **Next**.
- Select the previously created project. Click **Finish**.
- If you need any other subfolders, repeat the process.
- Switch to the C/C++ Perspective.

If you checked out more than one project that shares an SVN folder as its project root, you can't commit the **.project** file back to SVN, or else you'll overwrite the **.project** file for those original projects with the one for your combined project. To ensure you don't accidentally check it in, we recommend that you add it to svn's list of files to ignore, preventing it from showing up as changed resource.

Sharing projects#

When you have created a project, you may want to "share" the settings so the next person can just check it out as an Eclipse project. If the given project root matches with exactly one folder in the source control system, you may commit project metadata files (**.project** and **.cproject**). If your project is attached to version control, but you don't want them to be committed, you have to add those files to the "ignore" list.

QNX projects share most of the options in makefiles, however some options such as the current build variants are user-specific (i.e. not in the project metadata). You can make them "shared" by enabling **Share project properties** in the Main tab of the QNX project properties. Some metadata is stored in files other than **.project** and **.cproject**. For example, the **Check Dependencies On/Off** settings are manifested as additional Makefiles (two in IDE 4.5, one in IDE 4.6).

Checking out existing Eclipse projects#

Source-control IDE integrations (CVS and SVN) can detect the presence of the Eclipse project metadata in the repository and check out pre-made projects. You can search recursively and check out all existing projects from the selected folder or choose a specific project. You can also create a special file called a *Team Project Set* (**.psf**) that contains a set of projects that you want everybody to check out. Team sets can be located in the source control repository as well, and the IDE will recognize them and should be able to check out all specified projects automatically.

Quick and Dirty project creation#

If you don't want to build using the IDE, but you suddenly want to use the debugger or profiler, or find memory corruption, you can create a simple project in the IDE for these tools to use. To do this:

- Create an empty Makefile project.
- Copy or link the source files into the project:
  - To create a link, create a new directory in the project. In the new directory dialog, select **Advanced**, and then pick **Link to folder**.
- Copy or link binaries and libraries into the project.
- Create a launch configuration for the selected tool.
- If you want source navigation, and the source files aren't in the same location where they were compiled, you need to enable **Source Path Lookup**; edit the Source tab of the launch configuration to provide a lookup.

See more details in Example 4, below.

Build properties#

QNX projects#

You can adjust the build properties for a QNX project using Project Properties or by modifying the makefile. Be aware that if you modify the makefile manually, you can make it unrecognizable by the IDE, and it won't be able to properly update it in future.

Example of things you can do in Project Properties:

- Add extra libraries and library paths (**Extra libraries** from the drop-down menu in the Linker tab)
- Add extra includes (**Extra include paths** from the drop-down menu in the Compiler tab)
- Define macro variables for the whole project
- Define macros and includes for one file (Click **Advanced** and pick a file on the left)
- Select more than one platform to build for (Build Variants tab)
- Create another build variant (such as Profiling)
- Change a **make** variable (in the Make Builder tab, override the Build Command; for example, **make DEBUG=1**)
- Add custom compiler and linker options
- Change the name of the output binary/library

Managed projects#

You can adjust the build properties for Managed projects using the Project Properties.

Example of things you can do in Project Properties:

- Add extra libraries and library paths (**C/C++ Build->Settings->QCC Linker->Libraries**)
- Change output options, such as add Debug, Optimize, or Instrumentation options (**C/C++ Build->Settings->QCC Compiler->Output Control**). Some options require changes in both the compiler and linker.
- Add custom linker or compiler options
- Add another build variant (build configuration) (**Manage Configuration...** button in any page of the C/C++ Build)
- To set an individual file's options, use the same properties, but on the file/folder.

Other things:

- To exclude a file from a build, select the file and select **Exclude from build** from the context menu.
- To include a folder in the build, it has to be a "source folder". Otherwise, you can click on the folder, select **Properties**, and then uncheck **Exclude from build** in the C/C++ Build page.
- You can choose Internal Build or External Make build with make generation. This is controlled from **C/C++ Build->Tool Chain Editor->Current Builder**.

Makefile projects#

For a Makefile project, you can change the location where the build starts from, and the **make** arguments (and even the command to launch **make** itself). You can also change environment variables for the **make** invocation in the Environment subcategory of the C/C++ Build options. You can change the same variables automatically from the Settings tab, if you're using QNX naming conventions for the **make** variables. If the variables are defined in **make** itself, environment variables can't override them, unless you use **make -e**.

All other options you set in your makefile.

Building from the IDE#

To do a simple build for an active configuration, select one or more project, and then select **Build Project** from the context menu or from the main Project menu. To clean one or more projects, select **Clean Project**.

For QNX and Makefile projects, you can create and build specific **make** targets. Use the context menu for that.

For Makefile and Managed projects, you can create several build configurations (for example debug, release, x86, arm or any combinations or these), you can switch the default active configuration and build using the Build Configurations menu. You can also set a global preference to build one configuration or all when you do a build on a Managed project (in IDE 4.6 or later).

A *container project* is a specific, complex, quasi-hierarchical project that lets you combine a set of projects in order to quickly switch between different build configurations. The root of a container project is always a container. A container's children are *configurations*. A configuration is a set of projects of various types (including other containers). Each member of a configuration has two important attributes specific to the container environment: a *target* and a *variant*. These are very generic attributes; their interpretation is completely defined by a particular configuration member. For example, for a QNX C project, the target is a build target, such as **build, clean**, and the variant can be something like **x86/release, x86/debug**. To build any desirable configuration, just select the container configuration from the pull-down menu. For Managed projects, you can use Working Sets to build them together.

Setting up automated command-line builds#

To set up automated builds in version 4.5 of the IDE, you have to use **make**. If you're using a Managed project, you have to use Gnu Make Build, which generates makefiles for you. If a project doesn't depend on anything, just run **make** in a root (or appropriate) directory. If you want to build several projects, you have to create an external makefile that references all subprojects. For details, see the IDEBuildFAQ.

In version 4.6 of the IDE, you can use the **mkbuild** command (it comes with the QNX Momentics Tool Suite 6.4.1), which launches Eclipse in headless mode (that is, without the UI) and lets you build projects in your workspace as you would from the UI. For more information, see the IDE User Guide.

Examples#

Example 1

Suppose your local source (c++) files are in the following structure:

```
-source
  -a
   +inc
   -b
     -mydir
       +src
       +out
        Makefile
```

You work in the **mydir** directory and just run **make**, which picks up libraries from other parts of the filesystem and picks header files from **inc** (and local ones inside **mydir**). To create a project, do the following:

- Select **New->C++ Project** (from the context menu in the Project Navigator).
- Name your project "mydir" (or any name you like).
- Uncheck **Use default project location**.
- Browse to pick the **mydir** directory from the filesystem.

- Select **Makefile->Empty Project**.
- Select **QNX Toolchain**.
- Click **Finish**.
- Select the new project and select **Properties...** from the context menu (right click).
- In the Properties dialog, select **C/C++ General->Paths and Symbols**.
- Select **GNU C++** and add the directory **/source/a/inc** as your include path (the internal parser needs this for code navigation, refactoring, syntax highlighting, etc.).
- Also if you know the default macros that **make** uses to build the source, add them here. For example, if you compile as **qcc -DDEBUG foo.c**, add the DEBUG macro)
- Run **Build Project**.

Example 2 (Makefile isn't in the root)

Suppose that the directory structure is the same as in example 1, except that the Makefile is in the **out** directory, and this is the build directory.

- Create the project as above.
- Open the Project Properties, and then click on *C/C++ Build*.
- As the Build directory, specify **${workspace_loc:/mydir/out}**.

Example 3 (Link for output directory)

Once again, let's use the same directory structure as in example 1, except that the **out** directory is outside of **mydir** (but the Makefile is still in there).

- Create the project as above.
- Select the project, and the select **New->Folder** from the context menu.
- In the New Folder dialog, click **Advanced**.
- Select **Link the folder in the file system**.
- Select the output folder outside of the project's **mydir**.
- Click **Finish**.

Example 4 (Links for everything)

This time, the source code is in directory A, the output binaries are in directory B, and the extra header files in directory C. All of them have a common root, D (for example **D/x/y/z/A**, **D/x/C**, **D/w/C**)

You want the project itself with its metadata to be somewhere else. You also want to share it, if possible.

Procedure

- Create an C++ Empty Makefile project.
- In the project dialog, use the default project location (in your workspace). Let's call it Project4. Click **Finish**.
- Now we're going to create link folders for A,B, and C.
  - In the project, select **New->Folder** from context menu.
    - Click **Advanced**, and then check **Link the folder in the file system**.
    - Select **Variables...**. (If you don't have this button because you're using an earlier version of the IDE, just select the path from the filesystem).
    - Add a new variable called ROOT_DIR and set its value to be the path in the filesystem pointing to directory D.
    - In the dialog, click **Extend...** and select subdirectory A.
  - You should see a path something like **ROOT_DIR/x/y/z/A** in the path. Click **Finish**.

    - Repeat this procedure for the output directory, B (but you don't need to create the ROOT_DIR variable again -- just click **Extend...**).
    - Repeat the procedure for the extra include directory, C. This is required so that the project settings can use a relative path; if you aren't going to share this project, you can omit this step.
- Now you need to add an extra include folder:
    - In the Properties, select **C/C++ General->Paths and Symbols**. Select **GNU C++**, and the add directory C as the include search path using the **Workspace...** button (it should be something like **/Project4/C**).
- Now let's say the makefile is in the folder A (source folder). We need to instruct IDE to run **make** from there:
    - Open the Project Properties, and then click on **C/C++ Build**.
    - As the Build directory, specify **${workspace_loc:/Project4/A}**

The **Build Project** and **Clean Project** commands should now work. You should see your binaries in the Binary container. And your includes in the Includes container. Open a source file and check that Include navigation works (double-click on an external include in the Outline view; you should navigate there).