



Application Note

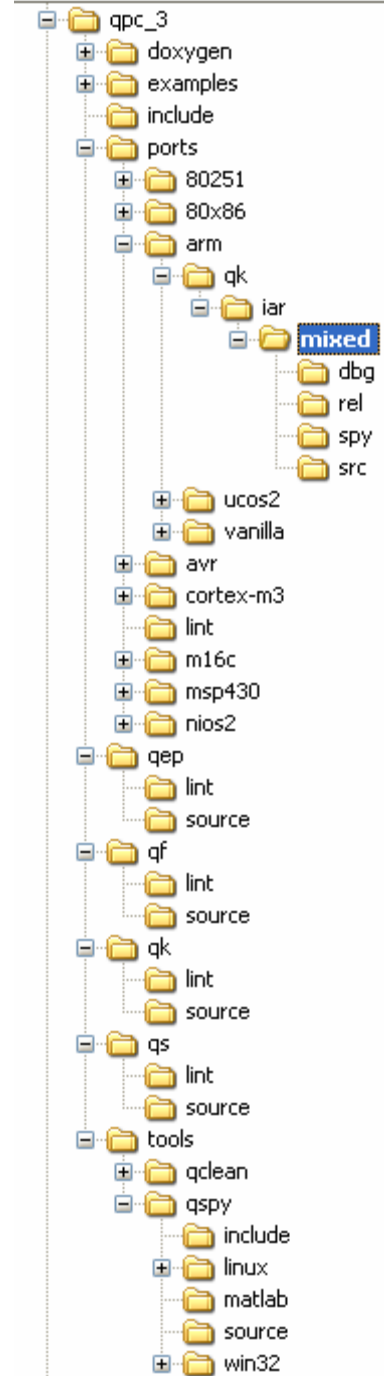
QP Directory Structure

Document Version C
May 2007

Quantum **L^eaP^s**TM, LLC
www.quantum-leaps.com

Copyright © 2002-2007 Quantum **L^eaP^s**, LLC. All Rights Reserved.

All parts of this document and the referenced software are protected by copyright law and can be used only with a valid license, as described in this document. Any other use or distribution of the referenced software or any part of this document is illegal.



1 Introduction

This document describes a new directory structure used for organizing Quantum Platform (QP) code starting from release 3.3.xx. At a minimum, this document should address the question:

"Where can I find a port of a given QP component or an example application?"

It is difficult to plan for all the possible combinations of embedded processors, operating systems, compilers, compiler options, and so on. On the other hand, managing this variety is an essential part of the embedded software development. The hierarchical directory structure described in this Application Note should take care of current and future expansions.

2 QP Root Directory

Every QP implementation resides in the separate directory branch, called henceforth QP Root Directory. You can freely choose the name of the QP Root Directory, however, the recommended name for QP/C 3.x.yy is **qpc_3** and for QP/C++ 3.x.yy is **qpcpp_3**. In the following sections, the QP Root Directory will be referred to as `<qp>/`, where the angle brackets symbolize the optional name of this QP Root Directory.

NOTE: the directory structure described here applies equally to C, C++, and potentially other implementations of QP.

NOTE: this document uses a Unix-style directory separator '/'. For the Windows hosts, please replace the forward slash with the backward slash '\'.

The essential element of the design is that the QP Root Directory can be “plugged into” any branch of a hierarchical file system and you should be able to move the QP Root Directory around, or even have multiple versions of the QP Root Directories. (The latter possibility could be very useful while working with a Version Control System, where you would be able to check-out QP source code into different work directories). The ability to relocate the QP Root Directory means that only relative paths are allowed in Makefiles, scripts, workspaces, or project files.

3 Top-level Directories

Within the QP Root Directory, we currently have the following subdirectories:

```
<qp>/          - QP-root directory for Quantum Platform (QP/C) v3.0
|
+-gpl.txt       - GNU General Public License
+-doxygen/     - HTML documentation generated from the source (with Doxygen)
+-examples/    - QP/C examples
+-include/     - QP/C platform-independent include files
+-ports/       - QP/C ports (header files and libraries)
+-qep/         - QEP/C component (source files and ports)
+-qf/          - QF/C component (source files and ports)
+-qk/          - QK/C component (source files and ports)
+-qs/          - QS/C component (source files and ports)
+-tools/       - Host-based tools, such as CleanCode, Q-Spy, etc.
```

4 Directory doxygen

Contains HTML documentation of the code extracted from the platform-independent source code by means of the Doxygen Documentation System [Doxygen 05].

```
<qp>/          - QP-root directory for Quantum Platform (QP/C) v3.0
|
+-doxygen/     - HTML documentation generated from the source (with Doxygen)
| +-examples/  - Code snippets used as examples
| +-html/      - HTML documentation files
| | +-index.html - the main HTML index page to start browsing the QP documentation
| +-images/    - Images used in the HTML documentation
| +-Doxyfile   - Doxygen configuration file used to generate the HTML
| +-qp.h       - QP Header file with comments about QP components, contains
|              also release notes.
```

Among others, the HTML documentation contains **release notes** for all QP components. (Please follow the “Revision History” links from the index page located in `<qp>/doxygen/html/index.html`.)

5 Directory include

Contains platform-independent header files of the QP components and for QP v3.1.xx contains:

```

<qp>/          - QP-root directory for Quantum Platform (QP/C) v3.0
+--include/    - QP/C platform-independent include files
  +-qassert.h  - Quantum assertions
  +-qep.h      - Quantum Event Processor QEP/C
  +-qf.h       - Quantum Framework QF/C
  +-qk.h       - Quantum Kernel QK/C
  +-qs.h       - Quantum Spy QS/C
  +-qqueueue.h - Quantum Event Queue (optionally used in QF ports)
  +-qmpool.h   - Quantum Memory Pool (optionally used in QF ports)
  
```

NOTE: The <qp>/include/ directory should be placed on the include path for the C compiler to compile the application code dependent on QP.

6 Ports Directory

The directory <qp>/ports/ contains platform-dependent header files and libraries to be used by QP applications. Managing this directory structure is complicated because of the large number of options available, such as embedded processors, compilers, operating systems, and evaluation boards that QP can support. Each of those options represents really a dimension in a multi-dimensional space of options.

The ports directory structure described in this section is designed for extensibility, so adding new ports should be easy and should have no impact on the existing ones. Every dimension in the multi-dimensional option space is represented as a level of nesting in a hierarchical directory structure, and therefore each dimension can be extended independently on the others. Also, the directory branch for each port is individually customizable, so each branch can represent only options relevant for a given CPU, operating system, compiler, etc.

6.1 CPU Directories

The first level of nesting within the ports directory is the **CPU architecture**. Examples of CPU architectures are: ARM, Cortex-M3, 80x86, AVR, MSP430, M16C, etc. Please note that even though the traditional ARM and the new ARM Cortex-M3 are related, the differences are significant enough to require a separate directory branch for ARM and Cortex-M3. The following listing illustrates some of the CPU directories:

```

<qp>/          - QP-root directory for Quantum Platform (QP/C) v3.0
+--ports/      - QP/C ports (header files and libraries)
  +-80x86/     - generic 80x86 processor, both real and protected mode
  +-arm        - ARM architecture (traditional ARM/THUMB architecture)
  +-avr/       - AVR architecture
  +-cortex-m3/ - Cortex-M3 architecture
  +-msp430/    - MSP430 architecture
  . . .
  
```

6.1.1 Operating System Directories

The second level of nesting, under the CPU architecture, is the **operating system** used to support QP. For example, in the ARM architecture, QP can operate with the Quantum Kernel (QK) [QL 05], with uC/OS-II Real Time Kernel [Labrosse 02], or in plain-vanilla setting without any underlying operating system (main()+ISR).

```

<qp>/          - QP-root directory for Quantum Platform (QP/C) v3.0
|
+-ports/       - QP ports (header files and libraries)
|
+-arm/         - ARM processor, ARM, THUMB, and mixed ARM/THUMB
|
|  +-qk/       - QK kernel
|  +-ucos2/    - uC/OS-II kernel
|  +-vanilla/  - Plain-vanilla (no kernel - main()+ISR)
|  . . .
|
+-80x86/       - generic 80x86, both real and protected mode
|
|  +-dos/      - DOS operating system (main()+ISR)
|  +-qk/       - Quantum Kernel
|  +-linux/    - Linux operating system
|  +-win32/    - Windows (Win32) operating system
|  . . .
|  . . .

```

6.1.1.1 Compiler Directories

The next level of nesting, under each operating system directory, is the directory for the **compiler** used. For example, ARM code can be compiled with the GNU gcc, with RealView, or with the IAR compilers.

```

<qp>/          - QP-root directory for Quantum Platform (QP/C) v3.0
|
+-ports/       - QP/C ports (header files and libraries)
|
+-arm/         - ARM processor, ARM, THUMB, and mixed ARM/THUMB
|
|  +-qk/       - QK kernel
|  |  +-gnu/    - GNU compiler
|  |  +-realview/ - ARM RealView compiler
|  |  +-iar/    - IAR ARM compiler
|  +-ucos2/    - uC/OS-II kernel
|  +-vanilla/  - Plain-vanilla (no kernel - main()+ISR)
|  . . .
|
+-80x86/       - generic 80x86, both real and protected mode
|
|  +-dos/      - DOS operating system (main()+ISR)
|  |  +-tcpp101/ - Turbo C++ 1.01 compiler
|  |  +-bcpp45/  - Borland C++ Builder 4.5
|  +-qk/       - Quantum Kernel
|  +-linux/    - Linux operating system
|  +-win32/    - Windows (Win32) operating system
|  . . .
|  . . .

```

6.1.1.2 CPU-Mode Directories

Each compiler can emit code for various **modes of the CPU**. For example, a compiler for the ARM architecture can produce ARM code, THUMB code, or a mixed ARM/THUMB code (some modules in ARM, some in THUMB) that requires interworking. These different modes can use different policies for interrupt disabling, ISR structure, initialization, etc., and therefore each of them requires a separate branch.

```

<qp>/          - QP-root directory for Quantum Platform (QP/C) v3.0
+-ports/       - QP/C ports (header files and libraries)
  +-arm/       - ARM processor, ARM, THUMB, and mixed ARM/THUMB
    +-qk/      - QK kernel
      +-gnu/    - GNU compiler
      +-realview/ - ARM RealView compiler
      +-iar/    - IAR ARM compiler
      +-arm/    - ARM mode
        +-make.bat - Batch script to build QP libraries
        +-qep_port.h - QEP/C platform-dependent include file
        +-qf_port.h - QF/C platform-dependent include file
        +-qk_port.h - QK/C platform-dependent include file
        +-qs_port.h - QS/C platform-dependent include file
      +-thumb/  - THUMB mode
      +-mixed/  - mixed ARM/THUMB mode with interworking
    +-ucos2/    - uC/OS-II kernel
    +-vanilla/  - Plain-vanilla (no kernel - main()+ISR)
    . . .
  +-80x86/     - generic 80x86, both real and protected mode
    +-dos/     - DOS operating system (main()+ISR)
      +-tcpp101/ - Turbo C++ 1.01 compiler
      +-l/     - Large memory model
        +-make.bat - Batch script to build QP libraries
        +-qep_port.h - QEP/C platform-dependent include file
        +-qf_port.h - QF/C platform-dependent include file
        +-qs_port.h - QS/C platform-dependent include file
      +-s/     - Small memory model
      +-bcpp45/ - Borland C++ Builder 4.5
    +-qk/      - Quantum Kernel
    +-linux/   - Linux operating system
    +-win32    - Windows (Win32) operating system
    . . .
  
```

NOTE: The Port Mode directory `<qp>/ports/<cpu>/<os>/<compiler>/<mode>` should be placed on the include path for the C compiler to compile the application code dependent on QP.

6.1.1.2.1 Build-Type Directories

Finally, the QP library code can be compiled with different **compile-time switches and optimization options**. For example, a Debug version might contain the symbolic debug information, the Release version might use aggressive optimizations, and the Spy version might include the Quantum Spy [QL 05d] instrumentation.

```

<qp>/          - QP-root directory for Quantum Platform (QP/C) v3.0
+-ports/       - QP/C ports (header files and libraries)
  +-arm/       - ARM processor, ARM, THUMB, and mixed ARM/THUMB
    +-qk/      - QK kernel
      +-gnu/    - GNU compiler
      +-realview/ - ARM RealView compiler
      +-iar/    - IAR ARM compiler
      +-mixed/  - mixed ARM/THUMB mode with interworking
        +-dbg/  - Debug build
          +-libqep.r79 - QEP debug library
          +-libqf.r79 - QF debug library
          +-libqk.r79 - QK debug library
          +-libqs.r79 - QS debug library
        +-rel/  - Release build
          +-libqep.r79 - QEP release library
          +-libqf.r79 - QF release library
          +-libqk.r79 - QK release library
          +-libqs.r79 - QS release library
        +-spy/  - Spy build (Quantum Spy instrumented)
          +-libqep.r79 - QEP spy library
          +-libqf.r79 - QF spy library
          +-libqk.r79 - QK spy library
          +-libqs.r79 - QS spy library
      +-make.bat - Batch script to build QP libraries
      +-qep_port.h - QEP/C platform-dependent include file
      +-qf_port.h - QF/C platform-dependent include file
      +-qk_port.h - QK/C platform-dependent include file
      +-qs_port.h - QS/C platform-dependent include file
    +-arm/      - ARM mode
    +-thumb/    - THUMB mode
  +-ucos2/     - uC/OS-II kernel
  +-vanilla/   - Plain-vanilla (no kernel - main()+ISR)
  
```

NOTE: The application code must link with the appropriate version of the QP libraries located in the build directory `<qp>/ports/<cpu>/<os>/<compiler>/<mode>/<build>`.

6.1.1.2.2 The make.bat Batch Script for Building QP Libraries

Whenever possible, the Generally Available QP™ distribution contains very straightforward **batch files** for building the QP™ libraries. Typically, the QP libraries are built just once at the beginning of the project and used unchanged henceforth. The simple batch scripts are ideal for this type of use, since they don't require any additional tools (such as the make utility, proprietary Integrated Devel-

opment Environments and so on). The following listing shows the location of the make.bat batch scripts.

```

<qp>/ - QP-root directory for Quantum Platform (QP/C) v3.0
+-ports/ - QP/C ports (header files and libraries)
+-arm/ - ARM processor, ARM, THUMB, and mixed ARM/THUMB
  +-qk/ - QK kernel
    +-gnu/ - GNU compiler
    +-realview/ - ARM RealView compiler
    +-iar/ - IAR ARM compiler
      +-mixed/ - mixed ARM/THUMB mode with interworking
        +-make.bat - Batch script to build QP libraries
      +-arm/ - ARM mode
        +-make.bat - Batch script to build QP libraries
      +-thumb/ - THUMB mode
        +-make.bat - Batch script to build QP libraries
    +-ucos2/ - uC/OS-II kernel
    +-vanilla/ - Plain-vanilla (no kernel - main()+ISR)
  +-80x86/ - generic 80x86, both real and protected mode
  +-dos/ - DOS operating system (main()+ISR)
  +-qk/ - QK kernel
    +-tcpp101/ - Turbo C++ 1.01 compiler
    +-l/ - Large memory model
    +-make.bat - Batch script to build QP libraries
  
```

The provided make.bat batch script supports the different build configurations, which you can specify as an argument to the make.bat batch file. The default target is "dbg". Other targets are "rel", and "spy" respectively. The following table summarizes the targets accepted by the make.bat batch files.

Software Version	Build command
Debug (default)	make
Release	make rel
Spy ^(*)	make spy ^(*)

Table 6.1 Commands for building the Debug, Release, and Spy software versions.
^(*)The Spy configuration requires the QS™ component, which is distributed and licensed separately from the QP Generally Available (GA) release.

6.1.1.2.3 Platform-Specific Source Directory

Most QP ports require some small amount of platform-specific C (or C++) source code to implement the platform-specific callbacks (see ["QP Programmer's Manual"](#) [QL 07a]).

The platform-specific source code is placed in the directory <qp>/ports/<cpu>/<os>/<compiler>/<mode>/<build>/src/, as illustrated in the following listing:

```

<qp>/ - QP-root directory for Quantum Platform (QP/C) v3.0
  |
  
```



```

+-ports/                - QP/C ports (header files and libraries)
  +-arm/                - ARM processor, ARM, THUMB, and mixed ARM/THUMB
    +-qk/               - QK kernel
      +-gnu/            - GNU compiler
      +-realview/      - ARM RealView compiler
      +-iar/            - IAR ARM compiler
        +-mixed/       - mixed ARM/THUMB mode with interworking
          +-dbg/        - Debug build
          +-rel/        - Release build
          +-spy/        - Spy build (Quantum Spy instrumented)
        +-src/         - Platform-specific source directory
          +-qk_port.s79 - Platform-specific source code for the port
        +-make.bat     - Batch script to build QP libraries
        +-qep_port.h   - QEP/C platform-dependent include file
        +-qf_port.h    - QF/C platform-dependent include file
        +-qk_port.h    - QK/C platform-dependent include file
        +-qs_port.h    - QS/C platform-dependent include file
      +-arm/           - ARM mode
      +-thumb/        - THUMB mode
    +-ucos2/          - uC/OS-II kernel
    +-vanilla/        - Plain-vanilla (no kernel - main()+ISR)
  +-80x86/            - generic 80x86, both real and protected mode
  +-dos/              - DOS operating system (main()+ISR)
  +-qk/               - QK kernel
    +-tcpp101/       - Turbo C++ 1.01 compiler
      +-l/           - Large memory model
        +-src/       - Platform-specific source directory
          +-qk_port.c - Platform-specific source code for the port
        +-make.bat   - Batch script to build QP libraries
  
```

The platform-specific source files are considered conceptually a part of the respective QP component and therefore they include the package-scope header file. For example qk_port.c includes qk_pkg.h, which is part of the QK component source code (see upcoming Section 7). The latter file comes from the appropriate Port-Mode directory described in Section 6.1.1.2, which is selected by the compiler include option (typically -I) that is pointed to the desired Port-Mode directory <qp>/<cpu>/<os>/<compiler>/<mode>.

NOTE: The compiler include path must contain the <qp>/ports/<cpu>/<os>/<compiler>/<mode>/-<build>/src/ directory during the QP library build. However, the application code does **not** need have any knowledge of this directory.

7 QP-Component Directory Structure

The directory structure of each QP component, such as QEP, QF, QK, or QS, (see [QP 07a, QL 07b]) follows the same pattern and will be here explained for the QF component.

<qp>/	- QP-root directory for Quantum Platform (QP)
+ -qep/	- QEP component
+-lint/	- QEP configuration files for PC-lint™
+-source/	- QEP platform-independent source code (*.C files)
+ -qf/	- QF component
+-lint/	- QF configuration files for PC-lint™
+-source/	- QF platform-independent source code (*.C files)
+ -qk/	- QK component
+-lint/	- QK configuration files for PC-lint™
+-source/	- QK platform-independent source code (*.C files)
+ -qs/	- QS component
+-lint/	- QS configuration files for PC-lint™
+-source/	- QS platform-independent source code (*.C files)

The QF-component directory <qp>/qf contains the platform independent source files (*.C files) in the subdirectory <qp>/qf/source, as well as some support subdirectories, such as the lint/ subdirectory with the configuration files for linting the QF component.

7.1 Source Directory

The source directory contains the platform-independent *.C files and the package-scope header file qf_pkg.h.

<qp>/	- QP-root directory for Quantum Platform (QP)
+ -qf/	- QF component (source files and ports)
+-source/	- QF platform-independent source code (*.C files)
+-qa_ctor.c	
+-qa_defer.c	
+-qa_fifo.c	
.	
+-qf_pkg.h	- QF platform-independent package-scope include file
.	
+-qte_rarm.c	

All source modules include the package-scope header file qf_pkg.h, which in turn includes the platform-dependent, component-port file qf_port.h. The latter file comes from the appropriate Port-Mode directory described in Section 6.1.1.2, which is selected by the compiler include option (typically -I) that is pointed to the desired Port directory <qp>/ports/<cpu>/<os>/<compiler>/<mode>.

8 Examples Directory

Now the question is “Where do I find the example code?” We decided to have a completely separate directory branch <qp>/**examples/** for examples running on different CPUs, operating systems, compilers, compiler options, and evaluation boards.

The structure of the <qp>/examples branch closely mirrors the structure of the <qp>/ports directory, except that it adds one more level for various example applications.

The following listing shows the QDPP (Quantum Dining Philosophers) example for ARM, vanilla port (bare metal target board), IAR compiler, mixed ARM/THUMB mode:

```

<qp>/          - QP-root directory for Quantum Platform (QP/C) v3.0
+-examples/    - QP/C examples
  +-arm/       - ARM processor
    +-vanilla/ - Plain-vanilla CPU (no kernel - main()+ISR)
      +-iar/    - IAR ARM compiler
        +-mixed/ - mixed ARM/THUMB mode with interworking
          +-qdpp-at91eb40a - QDPP example for the AT91EB40A evaluation board
            +-dbg/      - Debug build (runs from RAM)
              +-qdpp.d79 - executable image
              +-...
            +-rel/      - Release build (runs from ROM)
              +-qdpp.hex - downloadable image in Intel hex format
              +-...
            +-spy/      - Spy build (runs from RAM)
              +-qdpp.d79 - executable image
              +-...
          +-Makefile   - make file to build the QDPP example
          +-at91.h
          +-at91eb40a_ram.xcl - IAR ARM linker command file
          +-qdpp.h
          +-main.c
          +-isr.c
          +-philos.c
          +-table.c
          +-cstartup_ram.s79 - IAR ARM assembler module containing RAM startup code
          +-cstartup_rom.s79 - IAR ARM assembler module containing ROM startup code
          +-vectors.s79 - IAR ARM assembler module containing exception vectors
          +-qdpp_dbg.ewp - IAR project file to debug the QDPP example
          +-qdpp_dbg.eww - IAR workspace file to debug the QDPP example
          +-qdpp_spy.ewp - IAR project file to debug the QDPP example with Q-Spy
          +-qdpp_spy.eww - IAR workspace file to debug the QDPP example with Q-Spy
  +-...
  +-...
  +-...
  
```

9 Tools Directory

The directory `<qp>/tools/` contains the host-based tools that support target code development. Whenever it is possible, the host-based tools are provided for the Windows and Linux hosts. Currently, the Windows compiler used is the Visual C++ 6.0, while Linux version uses the standard g++.

The following listing shows the structure of the `<qp>/tools` directory:

```

<qp>/          - QP-root directory for Quantum Platform (QP/C) v3.0
+-tools/       - QP host-based tools
  |--cleancode/ - CleanCode utility (C++)
  |--include/   - include files
  |--source/    - platform-independent source files
  |--win32/     - Win32-specific source files and Visual C++ workspace
  |   |--Debug/ - Debug build directory
  |   |--Release/ - Release build directory
  |   |   |--cleancode.exe - the Windows executable
  |--linux/    - Linux-specific source files and Makefile
  |   |--dbg/   - Debug build directory
  |   |--rel/   - Release build directory
  |   |   |--cleancode - the Linux executable
  |   |   |--Makefile  - Makefile for building the executable
+-qspy/       - Quantum Spy host utility (C++)
  |--include/   - include files
  |--source/    - platform-independent source files
  |--win32/     - Win32-specific source files and Visual C++ workspace
  |   |--Debug/ - Debug build directory
  |   |--Release/ - Release build directory
  |   |   |--qs.exe    - the Windows executable
  |--linux/    - Linux-specific source files and Makefile
  |   |--dbg/   - Debug build directory
  |   |--rel/   - Release build directory
  |   |   |--qspy     - the Linux executable
  |   |   |--Makefile - Makefile for building the executable
  .
  .
  .
  
```

10 Summary of Target-Resident Code Directories

All directories related to the ports of the target-resident code have similar structure. The following listing summarizes the similarities among directories <qp>/ports/ and <qp>/examples/:

```

<qp>/
+-ports/
  +-arm/
    +-vanilla/
      +-iar/
        +-arm/
          +-dbg/
          +-rel/
          +-spy/
          +-src/
          +-make.bat
        .
        .
      .
      .
    +-80x86/
      +-dos/
        +-tcpp101/
          +-l/
            +-dbg/
            +-rel/
            +-spy/
            +-src/
            +-make.bat
          .
          .
        .
        .
      .
      .
    .
    .
  .
  .

<qp>/
+-examples/
  +-arm/
    +-vanilla/
      +-iar/
        +-qdp-at91eb40a/
          +-arm/
            +-dbg/
            +-rel/
            +-spy/
          +-Makefile
        .
        .
      .
      .
    +-80x86/
      +-dos/
        +-tcpp101/
          +-qalc/
            +-l/
              +-dbg/
              +-rel/
              +-spy/
            +-Makefile
          .
          .
        .
        .
      .
      .
    .
    .
  .
  .

```

11 References

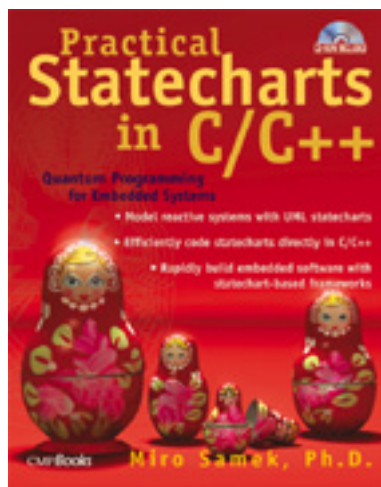
[C99]	ISO/IEC 9899 C Approved Standards, http://www.open-std.org/jtc1/sc22/open/n2794
[Doxygen 05]	Doxygen is a documentation system, http://www.doxygen.org
[Gimpel 04]	Gimpel Software, http://www.gimpel.com
[Labrosse 02]	Labrosse, Jean, J., "MicroC/OS-II The Real-Time Kernel, Second Edition", CMP Books 2002.
[QL 07a]	Quantum Leaps, LLC, "QP Programmer's Manual", http://www.quantum-leaps.com/doc/QP_Manual.pdf
[QL 07b]	Quantum Leaps, LLC, "QS Programmer's Manual", http://www.quantum-leaps.com/eval/QS_Manual.pdf

12 Contact Information

Quantum Leaps, LLC
103 Cobble Ridge Drive
Chapel Hill, NC 27516
USA

+1 866-450-LEAP (toll free)
+1 919-869-2998 (fax)

e-mail: info@quantum-leaps.com
WEB : <http://www.quantum-leaps.com>



"Practical Statecharts in C/C++",
by Miro Samek, Ph.D.
CMP Books 2002,
ISBN 1578201101