

QP™ State Machine Frameworks

What is it?

QP™ is a family of lightweight, open source, state machine frameworks for embedded microprocessors, microcontrollers, and DSPs. The QP frameworks enable implementing systems of concurrent

hierarchical state machines (UML statecharts) directly in readable and maintainable C or C++ **without big tools.**



Current versions of QP are: QP/C™ and QP/C++™, which require about 4KB of code (ROM) and a few hundred bytes of RAM, and the ultra-lightweight QP-nano™, which requires only 1-2KB of code and just several bytes of RAM.

For all intents and purposes, QP can be viewed as a **modern event-driven operating system** that replaces or augments traditional OS or RTOS, so QP is clearly much more than just state machines. QP was designed for efficiency, determinism, and very small footprint—especially in RAM. All QP frameworks can run standalone (on “bare metal” microprocessors, microcontrollers, or DSPs). The QP/C and QP/C++ frameworks can also work with a conventional OS/RTOS.

The QP frameworks are described in great detail in the book **Practical UML Statecharts in C/C++, 2nd Edition: Event-Driven Programming for Embedded Systems** by Dr. Miro Samek, (ISBN: 978-0750687065). The book has a companion website at www.state-machine.com/psicc2.

Why do you need it?

Most software developers are accustomed to the basic sequential control, in which a program (a “superloop” or a task in an traditional RTOS) waits for events in various places in its execution path by either actively polling for events or passively blocking on a semaphore or other such RTOS mechanism. Though this approach is functional in many situations, it doesn’t work very well when the system must timely react to multiple events whose arrival times and order one cannot predict. The fundamental problem is that while a sequential task is waiting on one kind of event, it is not doing any other work and is **not responsive** to other events.

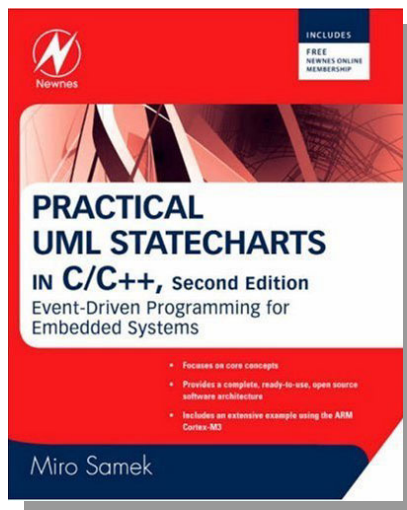
A long known, better alternative is to structure the software around the **event-driven programming model**, which requires a distinctly different way of thinking than conventional sequential programs. Event-driven systems are naturally divided into the application, which actually handles the events, and the supervisory **event-driven**

framework, which waits for events and dispatches them to the application. The control resides in the event-driven framework, so from the application standpoint, the control is **inverted** compared to a traditional sequential program. To remain responsive, the event-driven application cannot block, but must quickly return control after handling each event. Thus, the execution context cannot be preserved in the stack-based variables and the program counter as it is in a sequential task. Instead, the event-driven application becomes a **state machine**, or actually a set of collaborating state machines that preserve the context from one event to the next in the static variables.

In the embedded systems space, this event-driven paradigm traditionally required sophisticated design-automation tools, such as Rational Rose-RT or I-Logix Rhapsody (now both acquired by IBM). But the really important part of any such tool is not the flashy GUI for drawing the diagrams. The most valuable part of those systems is the **event-driven, state machine framework** that each of those tools contains and which forms the basis for the automatic code generation.

The open source QP™ state machine frameworks are in many respects similar to the frameworks buried in all commercially successful tools. The main difference is that the QP frameworks are not concerned with facilities for animation of state machines and are not biased toward mechanical code generation (although QP makes a great code-generation target). Instead, the QP frameworks are designed for direct, manual coding of event-driven applications.

When you start using QP, you will quickly discover that coding even the most involved state machines is really a non issue. Your problems will change. You will no longer struggle with 15 levels of convoluted `if-else` statements, and you will stop worrying about semaphores or other such low-level RTOS mechanisms. Instead, you will start thinking at a **higher level of abstraction** about events, states, and state transitions. After you experience this **quantum leap** you will find that programming can be much more fun. You will never want to go back to the “spaghetti” code or the raw RTOS.



“Practical Statecharts” has been an indispensable reference for my embedded systems development work...”

— Dr. Haitham Hindi, Research Staff, Palo Alto Research Center (PARC)

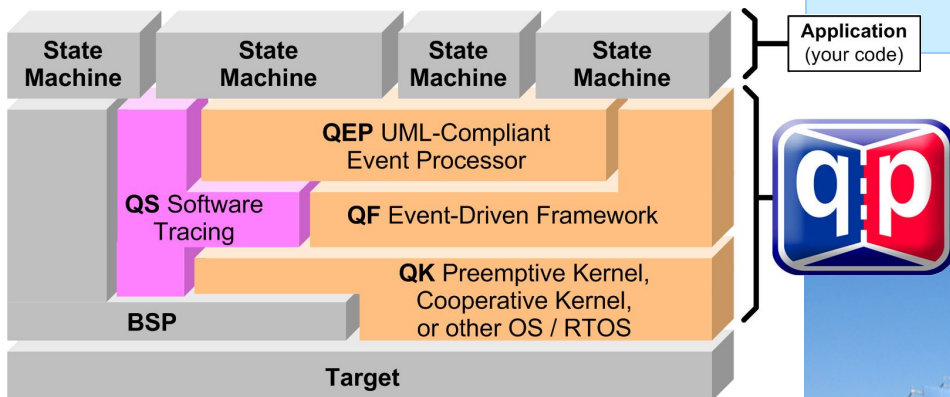
“After trying out a couple of different CASE tools, we came to the conclusion that expensive round-tripping UML tools were cumbersome and did not fit our way of working. However, the innovative QP software to map UML state machines to C/C++ code was exactly what we were looking for...”



— Henrik Bohre, Embedded Systems Consultant, GotCom AB, Göteborg, Sweden

What makes it great?

Due to the inversion of control, the **QP™** frameworks can offer benefits that no RTOS can, such as zero-copy event management, publish-subscribe event delivery, automatic garbage collection for events, and powerful software tracing for unprecedented visibility into a running application. All this is possible, because unlike a conventional RTOS, the **QP** framework completely controls the lifecycle of events and state machines. Interestingly, even though **QP** raises the level of abstraction way above any RTOS, it is actually **smaller than a conventional RTOS** both in CPU time and memory space required. **QP** consists of the following components:



“ The QP implementation is very compact and efficient yet provides lots of features for such a small package. The QP code is well organized and allowed us to focus on our design and functionality instead of reinventing the wheel with another custom state machine. One of the best parts about this model is that the maintenance turns out to be much easier without adding lots of spaghetti code to add in something new... Without QP, I don't believe we could have delivered on our given schedule dates with the same level of quality. ”



— Jeff Karau, Sr. Software Engineer, General Dynamics C4 Systems



QP helps generate clean solar power from Stirling Energy SunCatchers

QEP Universal UML-compliant event processor (**QEP™**) that enables direct coding of UML statecharts in highly maintainable C or C++, in which every state machine element is mapped to code precisely, unambiguously, and exactly once (**traceability**). QEP fully supports **hierarchical state nesting**, which enables reusing behavior across many states instead of repeating the same actions and transitions over and over again.

QF Highly portable, event-driven, real-time framework (**QF®**) for concurrent execution of state machines specifically designed for real-time embedded (RTE) systems.

QK Tiny **preemptive** non-blocking run-to-completion kernel (**QK™**) designed specifically for executing state machines in a run-to-completion (RTC) fashion.

QS Software tracing system (**QS™**) that enables live monitoring of event-driven **QP** applications with minimal target system resources and without stopping or significantly slowing down the code.

Who's using it?

QP™ has strong user community in variety of industries, such as consumer electronics, medical devices, industrial control, wired and wireless communications, networking equipment, research, defense, robotics, power generation, RFID, and many others. Some of the companies that acquired commercial **QP** licensees are shown on the right.



QP™ Features and Benefits

FEATURE	QP / C	QP / C++	QP-nano	FEATURE	QP / C	QP / C++	QP-nano
UML-Compliant Event Processor (QEP™)				Real-Time Framework (QF®)			
Highly maintainable and traceable boilerplate state machine mapping to C or C++	✓	✓	✓	Support for active object computing with the following number of active objects:	63	63	8
Full support for hierarchical state nesting	✓	✓	✓	Deterministic thread-safe execution	✓	✓	✓
Full support for automatic entry/exit action execution on arbitrary state transition topology	✓	✓	✓	Low-power architecture with idle-task callback	✓	✓	✓
Full support of nested initial transitions	✓	✓	✓	Direct event delivery with FIFO policy	✓	✓	✓
Number of states limited only by code space in ROM	✓	✓	✓	Direct event delivery with LIFO policy	✓	✓	
Support for events with arbitrary parameters	✓	✓		Publish-subscribe event delivery	✓	✓	
Support for events with fixed-size parameters			✓	Dynamic events with zero-copy event passing	✓	✓	
Extremely small data requirements (RAM footprint) 1 pointer-to-function per state machine	✓	✓	✓	Automatic recycling of dynamic events	✓	✓	
Very small code size (ROM footprint)	0.6-1.5KB	0.8-2.0KB	0.5-1.0KB	Efficient zero-copy deferring and recalling events	✓	✓	
Support for simpler non-hierarchical Finite State Machines with entry/exit actions	✓	✓	✓	One-shot time events (one-shot timers)	✓	✓	✓
Fully reentrant event processor code with minimal stack requirements	✓	✓	✓	Periodic time events (periodic timers)	✓	✓	
Source code 98% compatible with MISRA guidelines and passing strict analysis with PC-lint	✓	✓	✓	Thread-safe event queues for task-to-ISR comms	✓	✓	
Software Tracing (QS™)				Thread-safe memory partitions (fixed size heaps)	✓	✓	
Thread-safe software tracing for all QP components	✓	✓		Platform Abstraction Layer (PAL) for portability	✓	✓	✓
Data compression minimizing the buffering and bandwidth requirements	✓	✓		PAL supports integration with a OS/RTOS	✓	✓	
Generic logging of application-level activities	✓	✓		Cooperative prioritized "vanilla" kernel	✓	✓	✓
Sophisticated runtime filtering of records based on record-type and object type	✓	✓		Small, scalable code size (ROM footprint)	2-4KB	2-4KB	.5-2KB
Precise time-stamping of trace records	✓	✓		Assertion-based error handling	✓	✓	✓
Symbolic information in the trace	✓	✓		Preemptive Run-to-Completion Kernel (QK™)			
Robust HDLC-like data protocol	✓	✓		Preemptive, priority-based, execution of RTC tasks	63	63	8
Decoupled data logging and transmission for flexibility and portability	✓	✓		Extremely fast run-to-completion event processing without blocking	✓	✓	✓
Portable host-based application (QSPY) with full source code	✓	✓		Single stack for all tasks and interrupts	✓	✓	✓
QSPY MATLAB® interface	✓	✓		Allows using compiler-generated ISRs	✓	✓	✓
Supported Middleware				Highly portable to most CPUs and compilers	✓	✓	✓
Open source lwIP TCP/IP stack	✓	✓		Priority-ceiling mutexes	✓	✓	✓
SEGGER emWin / Micrium uC/GUI embedded GUI	✓	✓		Extended context switch for coprocessors	✓	✓	
				Thread-local storage	✓	✓	



Supported Processors and Compilers

All members of the QP™ family of frameworks can be easily adapted to various microprocessor architectures and compilers.

PROCESSOR	COMPILER	QP / C	QP / C++	QP-nano
ARM7/ARM9	IAR EWARM, GNU-ARM	✓	✓	✓
ARM Cortex-M3	IAR EWARM	✓	✓	✓
TI MSP430	IAR EW430	✓	✓	✓
TI TMS320C28x	C2000	✓	✓	✓
Atmel AVR	WinAVR, IAR EWAVR	✓	✓	✓
Renesas M16C/R8C	HEW4/NC30	✓		✓
Renesas M32C	HEW4/NC308	✓		✓
Renesas H8S/300	HEW4/Hitachi C/C++	✓	✓	✓
Freescale ColdFire	IAR EWCF	✓	✓	
Freescale HC08	CodeWarrior HC(S)08			✓
Altera Nios II	Nios2-GCC (Nios II IDE)	✓	✓	
8051	IAR EW8051			✓
80251	Keil 80251	✓		
Microchip PIC18	MPLAB-C18	✓		✓
Microchip PIC24	MPLAB-C30	✓		✓
Cypress PSoC	PSoC-ICCM8C			✓

Supported Operating Systems/RTOSs

The QP/C and QP/C++ frameworks can also be easily adapted to work with conventional operating systems and RTOSs.

OS / RTOS	COMPILER	QP / C	QP / C++
Linux/BSD/QNX (POSIX)	GNU	✓	✓
Windows, WiCE (WIN32)	Visual C++, MinGW	✓	✓
VxWorks	GNU	✓	✓
ThreadX	Various compilers	✓	✓
FreeRTOS.org	Various compilers	✓	✓
Micrium µC/OS	Various compilers	✓	✓
eCos	GNU	✓	✓

A large, steadily growing number of **QP Development Kits** (QDKs) are immediately available for download from the Quantum Leaps web site. Each QDK contains the port of the specified QP framework plus example applications illustrating the use of QP on the specific processor architecture and compiler.

“ The QP frameworks have been adopted across the company and are used in all products on a variety of CPUs and OS platforms. ”

— Dr. Paul Montgomery,
Director of Engineering,
Novariant Inc., Fremont CA



Licensing QP™

All versions of QP™ as well as related software is available under the **dual-licensing** model, in which both the open-source software distribution mechanism and traditional closed-source licensing models are combined.

Open-Source Licensing

If you are developing and distributing **open source** applications under the GNU General Public License version 2 (GPLv2), then you are free to use the QP™ software under the GPLv2 license. Please note that GPLv2 Section 2 (b) requires that all modifications to the original code as well as all Derivative Works must also be released under the terms of the GPLv2 open source license.



About Quantum Leaps

Quantum Leaps, LLC is the provider of lightweight open source state machine frameworks. The company also offers training and consulting in UML state machines, real-time frameworks, and event-driven programming for embedded-systems.

Closed-Source Licensing

If you are developing and distributing traditional **closed source** applications, you might purchase one of Quantum Leaps commercial licenses, which are specifically designed for users interested in retaining the proprietary status of their code.

To read more about the licensing options, pricing, technical support, or to request a commercial license, visit www.state-machine.com/licensing.



Contact Information

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

Toll free: 866-450-LEAP
Fax: 919-869-2998
e-mail: info@quantum-leaps.com
web: www.state-machine.com
www.quantum-leaps.com