

# smx++

**smx++ is recommended for C++ projects. With smx++, an application can be coded exclusively in C++. There is no need to call smx C functions directly.**

smx++ is a C++ class library built on smx. It consists of 21 classes, as shown in the class hierarchy chart on the back. Most classes correspond directly to smx objects and most class methods correspond directly to smx functions.

New classes and methods were added to *improve* the API. (See sidebar.) Also, unnecessary smx objects and functions have been omitted. The net result is an interface that is friendlier and more intuitive than the smx API.

This interface also opens up the world of context sensitivity, function overloading, virtual functions, modularity, and inheritance to the embedded system programmer. It is more than a C++ wrapper — it is an environment.

For example, the user may create a unique object derived from the smx++ Task class. It would include all smx++ objects which are associated with the task. The derived task is completely modular and includes all data and methods which are required by the task. For example, a serial communications task object might include an interrupt service routine, an smxLsr object, an smxBucket object, and an smxExchange object.

For existing C smx applications, smx++ allows an easy transition to C++. Since smx underlies smx++, direct calls to smx are still possible, permitting a mixture of C and C++

---

## More Than a Wrapper

Some C++ experts might be tempted to develop their own C++ API for smx. We recommend smx++ for C++ programmers from expert to novice because it is more than just a wrapper:

1. *this* support: C++ compilers do not assume a multitasking environment. When a task is created it is not in its own environment. Hence its *this* pointer must be captured and saved (in its tcb). When the task runs, its *this* pointer is restored. (Where, depends upon C++ compiler internals.)
2. When used as intended, C++ code does frequent object creations and deletions via the *new()* and *delete()* operators. C++ compilers implement these via the heap functions, *malloc()* and *free()*. This can lead to severe heap fragmentation and slow, indeterminate allocations. These are unacceptable in embedded systems which have limited RAM and require fast, deterministic operation. To overcome this problem, smx++ permits overloading *new()* and *delete()* with fast block pools. Overloading may be done on a class by class basis to permit exact-size blocks or on a class group basis to reduce pools. Since fast blocks also do not have associated control blocks, very efficient memory utilization can be achieved.
3. Lsr object: The smx C API has no equivalent. Adds power and additional features to smx. It allows the user to put lsr routines in any segment, and lsr's can access as many parameters as desired.
4. TimerLsr object: Integrates the smx timer and lsr into a user-friendly whole which is easier to use and understand than in the C API for smx.
5. Task object: Has integrated main function and hooked exit/entry routine that make tasks easier to use than with the C API for smx. Also adds numerous methods to perform operations that would otherwise require directly accessing TCB fields.
6. Message object: Goes far beyond the capabilities of the MCB. Tracks size and other details automatically. Also adds extra error checking abilities. This is a container rather than a wrapper and has the benefits of a container.
7. Copy constructors allow the user to easily make copies of complex objects such as multi-level queues.
8. Global C++ object support: Initialization, creation, and destruction.

---

usage. This evolutionary approach allows *incrementally* migrating to C++. New functionality can be added using

the smx++ API, and old functionality can be gradually converted, as convenient.

