As of 13.04.2026

Embedded C++ Advanced: Object-Oriented Programming for Microcontrollers with C++/EC++ - Live Online Training

The growing complexity of embedded software applications and the ever increasing performance of hardware has resulted in C++ being more and more used in embedded systems. Depending on the application, quality features like safety and security, performance, resource consumption etc. have to be taken into account.

Objectives

You know the application and efficiency of advanced C++ constructs (namespaces, templates, exception handling, runtime type identification, new style casts, multiple inheritance, memory management).

You are able to make informed decisions on using these constructs in your application.

You have an overview of the elements and mechanisms of the STL (Standard Template Library) and can use them.

You can adapt patterns (state patterns, Singleton patterns, observer patterns, smart pointer patterns and layer patterns) to your applications and implement them.

Participants

The EC++ advanced training addresses programmers, software developers, software designers and software architects who use advanced C++ constructs in embedded software development.

Requirements

Basic knowledge of object oriented C++ is a must; basic knowledge of UML is an advantage.

Live-Online-Training

* Price per attendee, in Euro plus VAT

Training code: LE-EC++FOR

Face-To-Face - English

Date	Duration
08.06. – 11.06.2026	4 days

Live Online - German

Date	Duration
07.09. – 10.09.2026	4 days

Face-To-Face - German

Date	Duration
08.06. – 11.06.2026	4 days

MICRO CONSULT

As of 13.04.2026

Embedded C++ Advanced: Object-Oriented Programming for Microcontrollers with C++/EC++ - Live Online Training

Content

C++ for Embedded Applications

- History
- Specific quality requirements on embedded software
- Recommendations and rules (coding guidelines)
- C++ compiler basics
- Outlook: C++ idioms and clean code development
- Practical tip with important references

Summary - Basic C++ Constructs and Efficiency Considerations

- Class and object
- Class elements
- Modifiers for data, functions and objects
- Class function pointer using a state machine implementation as an example
- Class relations (association, aggregation, composition and inheritance)
- Virtual functions and interfaces (implementation and access)
- Extended C++ constructs

Namespaces and Efficiency Considerations

- Using (nested) namespaces in program code
- Namespace alias, anonymous namespace, Koenig lookup, inline namespace
- Applying the software architecture to program code
- C++ standard namespace std
- Application examples and recommendations for using these features in embedded software
- Exercise: Integrating namespaces in existing program code based on the architecture

Single Inheritance, Multiple Inheritance and Efficiency Considerations

- Programming single inheritance and multiple inheritance (with interfaces)
- Issues related to multiple inheritance, solutions
- Virtual inheritance
- Assembler, memory and runtime analysis and optimization
- Application examples and recommendations for using these features in embedded software
- Exercise: Using and programming multiple inheritance, virtually as an option

Exception Handling and Efficiency Considerations

- Exception handling - definition and programming
- Exception classes and hierarchies
- User exceptions
- C++ system exceptions
- Nested exception handling
- Assembler, memory and runtime analysis and optimization
- Application examples and recommendations for using these features in embedded software
- Exercise: Integrating exception handling in the existing application

Memory Management and Efficiency Considerations

- Memory segments (BSS, stack, heap) for objects, comparison
- Dynamic memory management with new and delete, with and without exception handling
- Operator overload of new and delete
- Pool allocation pattern
- Placement new
- Identifying risks and avoiding pitfalls
- Assembler, memory and runtime analysis and optimization
- Application examples and recommendations for using these features in embedded software
- Exercise: Creating and deleting an object dynamically on the heap

Runtime Type Identification (RTTI) and Efficiency Considerations

- RTTI - definition and programming
- type_info class
- Consequences in use
- Relation to exception handling and new style casts

- Assembler, memory and runtime analysis and optimization
- Application examples and recommendations for using these features in embedded software
- Exercise: Using RTTI for class identification at runtime

Type Conversion with New Style Casts and Efficiency Considerations

- Static, dynamic, const and reinterpret cast
- The right choice for use
- Relation to RTTI and exception handling
- Assembler, memory and runtime analysis and optimization
- Application examples and recommendations for using these features in embedded software

Templates and Efficiency Considerations

- Template functions
- Template class and object
- Template parameters and alias
- Inheritance and interfaces with template classes
- Practical tips: Static versus dynamic polymorphism
- CRTP (curiously recurring template pattern)
- Template specialization and (implicit versus explicit) instantiation
- Type traits and concepts
- Variadic template functions and classes
- Perfect forward
- Assembler, memory and runtime analysis and optimization
- Application examples and recommendations for using these features in embedded software
- Exercise: Programming a template class for use in the observer pattern context of the application

Smart Pointer, Recommended Use and Efficiency Considerations

- Smart pointers - specifics and variants
- Programming own smart pointers
- C++ smart pointers
- Lambda functions
- Assembler, memory and runtime analysis and optimization
- Application examples and recommendations for using these features in embedded software

C++ Standard Library: Containers, Iterators and Algorithms, Recommended Use and Efficiency Considerations

- Basic concept
- Containers, iterators, adapters, algorithms
- Specific container types and their differentiation
- Containers for sequential, sorting and specific applications
- Function objects (functors)
- Lambda functions
- Allocator class
- Assembler, memory and runtime analysis and optimization
- Application examples and recommendations for using these features in embedded software
- Exercise: Using a container class in the observer pattern context of the application

Callback

- Embedded software architecture guidelines
- Embedded software quality features
- Software layer pattern: embedded software layer architecture
- Synchronous and asynchronous software architecture
- Unidirectional and bidirectional communication
- Callback structure with and without operating system resources

Hardware Drivers and Interrupts in C++

- Object-oriented concepts and programming of standard peripheral drivers
- Object-oriented concepts and programming of interrupt handler
- Register bank access
- Callback structures in an interrupt context
- Application examples and recommendations for using these features in embedded software
- Exercise: Integrating a hardware driver and an interrupt service in the application

Select C++ Library Objects

- std::string and std::string_view
- iosteam and iomanip

- std::stringstream
- std::function, std::optional, std::variant and std::any
- Application examples and recommendations for using these features in embedded software

Practical Exercises in the Workshop

- You will use the STM32CubeIDE development environment along with the STM32 NUCLEO-F746ZG evaluation board, based on an Arm Cortex®-M3 microcontroller throughout the exercise (watch application).

MicroConsult Plus:

- Participants can download their exercises and the solutions developed by MicroConsult for this workshop.
- You get all C++ examples in electronic format and can easily adjust them to your development environment.
- You get a checklist with scalable recommendations for using C++ in embedded software.
- You get a tool and software component overview for developing embedded software.
- You get helpful notation overviews for UML and SysML.