Optimizing C++ Code

Simple changes which reduce code size for C++ programs using MPLAB® XC32

June 2013

Wednesday, 26 June 13

In this mini-webinar we show three ways of reducing the size of code produced by the MPLAB XC32 C++ compiler.
In this mini-webinar we show three ways of reducing the size of code produced by the MPLAB XC32 C++ compiler.
Avoid Pure Virtual Methods

Virtual methods in C++ can be overridden by inheriting classes. To allow this to happen, virtual method tables and a considerable amount of code are used at runtime to determine the *actual* method to execute.

When a virtual method is not expected to be called in a class, you can make this method *pure* by replacing the method's body with an assignment of zero. If such a method *is* called, an error will be signalled. This error is handled by additional code the compiler must link in. You can avoid this error-handling code by never using pure virtual methods and instead provide a default implementation for the virtual method. It may be something as trivial as that shown in the code example on this slide.
Virtual methods in C++ can be overridden by inheriting classes. To allow this to happen, virtual method tables and a considerable amount of code are used at runtime to determine the actual method to execute.

When a virtual method is not expected to be called in a class, you can make this method pure by replacing the method's body with an assignment of zero. If such a method is called, an error will be signalled. This error is handled by additional code the compiler must link in. You can avoid this error-handling code by never using pure virtual methods and instead provide a default implementation for the virtual method. It may be something as trivial as that shown in the code example on this slide.
Virtual methods in C++ can be overridden by inheriting classes. To allow this to happen, virtual method tables and a considerable amount of code are used at runtime to determine the actual method to execute.

When a virtual method is not expected to be called in a class, you can make this method pure by replacing the method's body with an assignment of zero. If such a method is called, an error will be signalled. This error is handled by additional code the compiler must link in. You can avoid this error-handling code by never using pure virtual methods and instead provide a default implementation for the virtual method. It may be something as trivial as that shown in the code example on this slide.
Avoid Pure Virtual Methods

Virtual methods generate tables and code
Pure virtual methods should not be called
The compiler must add error-handling code
Use a default method implementation

Virtual methods in C++ can be overridden by inheriting classes. To allow this to happen, virtual method tables and a considerable amount of code are used at runtime to determine the actual method to execute.

When a virtual method is not expected to be called in a class, you can make this method pure by replacing the method's body with an assignment of zero. If such a method is called, an error will be signalled. This error is handled by additional code the compiler must link in. You can avoid this error-handling code by never using pure virtual methods and instead provide a default implementation for the virtual method. It may be something as trivial as that shown in the code example on this slide.
Another way of reducing the size of your program is to use the compiler option `-fno-rtti` to disable runtime type information. This information is embedded into your code for use with operations like dynamic casting and the `typeid` operator. If these operations are not being used in your code, then this information can be safely omitted from the output.

It is safe to use this option as an error will be generated if you **have** used runtime type information in your project. Note that if you include the standard template library (STL) header files, these often use runtime type information and this option cannot be used.
Another way of reducing the size of your program is to use the compiler option to disable runtime type information.

This information is embedded into your code for use with operations like dynamic casting and the `typeid` operator. If these operations are not being used in your code, then this information can be safely omitted from the output.

It is safe to use this option as an error will be generated if you *have* used runtime type information in your project. Note that if you include the standard template library (STL) header files, these often use runtime type information and this option cannot be used.
Another way of reducing the size of your program is to use the compiler option to disable runtime type information.

This information is embedded into your code for use with operations like dynamic casting and the `typeid` operator. If these operations are not being used in your code, then this information can be safely omitted from the output.

It is safe to use this option as an error will be generated if you have used runtime type information in your project. Note that if you include the standard template library (STL) header files, these often use runtime type information and this option cannot be used.
Disable runtime type information

- Used with `dynamic_cast<>` and `typeid`

- Use of runtime type information and this option flag a compilation error

Another way of reducing the size of your program is to use the compiler option to disable runtime type information.

This information is embedded into your code for use with operations like dynamic casting and the `typeid` operator. If these operations are not being used in your code, then this information can be safely omitted from the output.

It is safe to use this option as an error will be generated if you have used runtime type information in your project. Note that if you include the standard template library (STL) header files, these often use runtime type information and this option cannot be used.
A similar code reduction technique is to use the option to disable exception handling.

Extensive code must be produced by the compiler to handle exceptions. If this is not a program requirement, the compiler can be instructed to omit exception-handling code from the output.

Again, if you use exceptions and have specified this option, an error will be generated by the compiler.
A similar code reduction technique is to use the option to disable exception handling. Extensive code must be produced by the compiler to handle exceptions. If this is not a program requirement, the compiler can be instructed to omit exception-handling code from the output.

Again, if you use exceptions and have specified this option, an error will be generated by the compiler.
Disable Exception Handling

- fno-exceptions

- Disable exception support
- Used to handle runtime errors

A similar code reduction technique is to use the option to disable exception handling.

Extensive code must be produced by the compiler to handle exceptions. If this is not a program requirement, the compiler can be instructed to omit exception-handling code from the output.

Again, if you use exceptions and have specified this option, an error will be generated by the compiler.
Disable Exception Handling

- Disable exception support
  - Used to handle runtime errors
- Use of exceptions and this option flag a compilation error

A similar code reduction technique is to use the option to disable exception handling.

Extensive code must be produced by the compiler to handle exceptions. If this is not a program requirement, the compiler can be instructed to omit exception-handling code from the output.

Again, if you use exceptions and have specified this option, an error will be generated by the compiler.