

C++/SX: Compatibility/Migration Notes

Description

This document describes notes on compatibility and migration between C++/SX Rev.070 or later, and C++/SX Rev.069 or before, concerning the following subject.

- [C++ Program Using the Standard C++ Library](#)

Notes

- *This document corresponds to C++/SX Rev.070 or later.*
- *If you use self compiler, **sxc++**, **sxcc** should be read **c++**, **cc**.*
- *Related document is "C++/SX programmer's Guide (G1AF28E)".*

Proprietary Notice

The information disclosed in this document is the property of NEC Corporation (NEC) and/or its licensors. NEC and/or its licensors, as appropriate, reserve all patent, copyright and other proprietary rights to this document, including all design, manufacturing, reproduction, use and sales rights thereto, except to the extent said rights are expressly granted to others.

The information in this document is subject to change at any time, without notice.

All product, brand, or trade names in this publication are the trademarks or registered trademarks of their respective owners.

1. C++ Program Using the Standard C++ Library

1.1. Overview

When an object file (.o file), that was generated by C++/SX Rev.069 or before, is applied to by one of the two conditions described in "1.2 C++ Program Using the Standard C++ Library", and it is linked by C++/SX Rev.070 or later, the linking may failed with the unresolved symbols. To avoid this incompatibility, the object file must be re-compiled.

In general, a C++ program using the Standard C++ Library, can be re-compiled for the template instantiation at the linking. Therefore, almost all C++ programs using the Standard C++ Library can be re-compiled. However, when it is not possible to re-compile, please see "1.4.2 When you cannot re-compile your C++ program".

Example of unresolved symbols error at the linking:

When an object file a.o, generated from a C++ program using the Standard C++ Library a.cpp below, by C++/SX Rev.069 with setting an environment variable SX_BASE_CPLUS is given,

```
% cat a.cpp
#include <iostream>          // using <iostream> in Standard C++ Library
using namespace std;
main()
{
    cout << "Hello World." << endl;
}
```

and a.o is linked by C++/SX Rev.070, the linking fails as follows.

```
% sxc++ a.o
undefined          first referenced
symbol             in file
_Getctype          a.o
_Tolower           a.o
_Toupper           a.o
std::ios_base::clear(std::ios_base::_Iostate, bool) a.o
...
```

1.2. Condition of the Program That Falls Under This Incompatibility

If an object file of the C++ program that uses the Standard C++ Library is applied to by one of the following two conditions at the compilation, the incompatibility may occur.

- (1) It was generated by compiling with setting the environment variable **SX_BASE_CPLUS**.
- (2) It was generated by compiling with **-Tnoauto**. (**-Tauto** is default.)

If (1) applies, refer to the sections "1.3 How to Check Whether an Object File Falls under the Incompatibility", and "1.4 How to Avoid the Incompatibility".

If (2) applies, the program must be re-compiled. Please refer to "1.4 How to Avoid the Incompatibility".

In C++, the path-names of the compiler and compiler options of the compilation are saved in a template information file in order to enable the re-compilation for the template instantiation.

Usually, the standard-installed compiler (/SX/usr/bin/sxc++) is used, but the non-standard-installed compiler (ex. /SX/opt/sxc++/rev069/bin/sxc++) is used by setting SX_BASE_CPLUS. When you used the non-standard-installed compiler (old compiler) by setting SX_BASE_CPLUS, the template instantiation using new Standard C++ Library will not be done because the old compiler's path name is saved and the old compiler's Standard C++ Library is taken. When **-Tnoauto** is specified, any template information file is not generated. Therefore, the linking fails when one of the above two conditions is applied.

1.3. How to Check Whether an Object File Falls under the Incompatibility

Whether an object file falls under the condition (1) in "1.2 Condition of the Program That Falls Under This Incompatibility" can be checked by the following three subjects.

- (a) Was the object file generated by C++/SX Rev.069 or before?
- (b) Is the object file the C++ program using the Standard C++ Library?
- (c) Was the object file generated with setting SX_BASE_CPLUS at the compilation?

When all the three conditions are fulfilled for an object file, the object file may fall under the incompatibility. Exceptionally, when the object file is in an archive file (.a file), it falls under the incompatibility, when the two conditions (a) and (b) are fulfilled, irrespective of (c).

How to check these three subjects are mentioned below.

- (1) How to know the revision number of C++/SX compiler which generated the object file.

You can get the revision number from the template information file (.ti file) corresponding to the object file. The template information file is automatically created under the same directory as where the object file is output, by the automatic instantiation feature of C++/SX compiler at the compilation. If a program does not use template feature, i.e. it does not use the Standard C++ Library, there is no template information file corresponding to the object file.

There are two three-digit numbers in the first line of the template information file. The second number is the revision number of C++/SX which compiled the object file.

Example:

```
% ls a.o a.ti
a.o    a.ti
% cat a.ti
#rev:001:069
...
```

a.ti is the template information file of a.o. The second number in the first line is 069. Therefore, the revision number of C++/SX compiler which generated a.o is Rev.069.

If an object file has no corresponding template information file, it means the object file does not use the Standard C++ Library, that is, it does not fall under this incompatibility.

- (2) How to know whether the C++ program uses the Standard C++ Library or not

A C++ program using the Standard C++ Library includes the following files under the directory /SX/usr/include/C++ (cross compiler) or /usr/include/C++ (self compiler) by **#include**.

algorithm	bitset	complex	deque	exception	fstream
functional	iomanip	ios	iosfwd	iostream	istream
iterator	limits	list	locale	map	memory
new	numeric	ostream	queue	set	sstream

stack	stdexcept	streambuf	string	typeinfo	utility
valarray	vector	fstream.h	iomanip.h	iostream.h	memory.h
new.h	stl.h				

You can know whether the file is included or not by the output of **-M** (which outputs a list of file dependency lines).

Example:

```
% sxc++ -M a.cpp | grep C++
a.o:/SX/usr/include/C++/iostream
a.o:/SX/usr/include/C++/istream
a.o:/SX/usr/include/C++/ostream
a.o:/SX/usr/include/C++/ios
a.o:/SX/usr/include/C++/xlocnum
...
```

The above output shows that a.o has dependency on iostream and some files, that is, a.cpp uses the Standard C++ Library.

(3) How to know whether the object file was generated with setting the environment variable **SX_BASE_CPLUS**

You can know this by referring to the template information file corresponding to the object file.

When the path name of **ccom** which appears on the line starting " cmd:" is not /SX/usr/lib0/ccom (cross compiler) or /usr/lib0/ccom (self compiler), the object file was generated with setting the environment variable **SX_BASE_CPLUS**.

Example:

```
% ls a.o a.ti
a.o a.ti
% cat a.ti
#rev:001:069
cmd: /SX/opt/sxc++/rev069/lib0/ccom -prelink_exec -cpp ...
...
```

The object file a.o was generated with setting the environment variable **SX_BASE_CPLUS** because the path name of **ccom** in the template information file a.ti is not /SX/usr/lib0/ccom.

Example:

```
% cat b.ti
#rev:001:069
cmd: /SX/usr/lib0/ccom -prelink_exec -cpp ...
...
```

The object file b.o was generated without setting the environment variable **SX_BASE_CPLUS** because the path name of **ccom** in the template information file b.ti is /SX/usr/lib0/ccom, and this means b.o does not fall under this incompatibility.

1.4. How to Avoid the Incompatibility

1.4.1. Re-compilation (Recommended)

The incompatibility can be dissolved by re-compiling the corresponding source file by C++/SX Rev.070 or later. In general, source files and object files of the C++ programs using the Standard C++ Library are placed

in the circumstance where the source files can be re-compiled at any time for the template instantiation. Therefore, almost all C++ programs using the Standard C++ Library can be re-compiled.

1.4.2. When you cannot re-compile your C++ program

You should link your object files with **-Ys** compiler option by C++/SX Rev.070 or later on the cross compile environment where C++/SX Rev.067, Rev.068, or Rev.069 are multi-instance installed. Specify the installed path name of the old compiler with **-Ys**. You can link your object files with the Standard C++ Library which is under the specified path.

The following is an example to execute C++/SX Rev.071 and specify the Standard C++ Library in C++/SX Rev.069.

Example:

```
% sxc++ a.o b.o -Ys,/SX/opt/sxc++/rev069      (Linking with the Standard C++
                                                Library of C++/SX Rev.069)
```

[Compiler Option]

Format:

-Ys,*directory-name*

Description:

Option	Suboption	Description	Default
-Ys , <i>directory-name</i>		Specifies the directory where another C++/SX compiler system whose Standard C++ Library is to be used for the compilation and linking is.	/usr /SX/usr

Note:

- This option can be specified to C++/SX compiler Rev.070 or later. When you use this option, please execute C++/SX compiler Rev.070 or later.
- Multi-instance installation of the two revisions of C++/SX compiler, one is Rev.070 or later, and another is one of the Rev.067, Rev.068, or Rev.069, by the system administrator is required in advance.

1.5. Q & A

This section presents some supplementary explanation for this incompatibility in Q & A format.

[Language specification/Terms]

(1) **What is the Standard C++ Library?**

It is the C++ library, conforming ISO/IEC 14882:1998 standard, provided by C++/SX. This library contains various class libraries such as **iterator** class corresponds to a loop in C++ language, **iostream** class for stream I/O, or etc. The Standard C++ Library is constructed from header files that define and declare templates and archive files that include entities of functions and data that are referred to from the templates.

(2) What is a template?

It is one of the C++ language specifications. Programmers define the parametric form of a function or a class as a template and compiler instantiates the function or the class according to the template and type information given as template parameters. The instantiated function or class can be treated as well as the non-template function or class. Using the template enables programmers to automatically generate various types of functions or classes from one template and increases their productivity.

(3) What is an instantiation?

An instantiation is an entity of the function or the class that is generated from a template by compiler.

Example:

```
template <class T> T func(T x, T y) {           // a function template
    return x + y;
}
...
double a, b, c;
int    m, n, i;
...
a = func(b, c);           // reference(1) to the template
m = func(n, i);           // reference(2) to the template
...
```

When a template is declared and referenced in a program as above, compiler generates entity of the functions like below, for the references (1) and (2) respectively. These functions are called instantiations.

```
double func(double x, double y) {           // an instantiated function of double
    return x + y;
}
int func(int x, int y) {                    // an instantiated function of int
    return x + y;
}
```

(4) What is template instantiation?

A sequence of compiler's processes to generate the instantiation from a template is called "template instantiation". C++/SX automatically scans all the object files, determines in which object file the instantiation should be generated, and re-compile the corresponding source files so that each of the necessary instantiations can be generated in the appropriate object file by the re-compilation.

[On dealing with this incompatibility]

(1) Is there a case in which no source file is available and it is impossible to re-compile, although the recompilation is needed at the linking?

Every object file of the C++ programs using a template always has the potential to be required the re-compilation at the linking for the template instantiation. Therefore, there should be the corresponding source file in the same environment as the object file, so that the programs can be maintained, in general.

(2) What should I do for object files in an archive file?

Object files(.o file) in an archive file(.a file) are not targeted on by automatic template instantiation process. Therefore, they cannot be re-compiled automatically at the linking. If the two conditions "Was the object file generated by C++/SX Rev.069 or before?" and "Is the object file the C++ program using the Standard C++ Library" in "1.3 How to Check Whether an Object File Falls under the Incompatibility" are fulfilled for the object files in the archive file, they should be re-compiled and the archive file should be updated.

(3) What should I do when I cannot find .ti file?

If an object file has no corresponding template information file and it has never been removed by users, it means the object file does not use the Standard C++ Library, that is, it does not fall under this incompatibility.

(4) When I linked C++ programs using the Standard C++ Library, a message such as “re-compiling: ...” was displayed and a source file is automatically re-compiled. Why?

At the linking of the C++ programs using a template, the source files may be automatically re-compiled for the template instantiation. This is default behavior and there is no problem.

[Others]

(1) Does a C program fall under this incompatibility?

No. this incompatibility does not occur in C programs because the Standard C++ Library cannot be used in C programs.