Concurrent C/C++

Version 7.3 Release Notes (PowerMAX)

November 2006

0890497-7.3

READ ME BEFORE INSTALLING THIS PRODUCT

concurrent

Contents

1.0.   Introduction

Concurrent C/C++ utilizes Edison Design Group's C++ front end and Concurrent's Common Code Generator (CCG) technology to produce highly-optimized object code tailored to Concurrent systems running PowerMAX OS™.

There are several compiler switches that provide a degree of compatibility with previous drafts of the ANSI C++ Standard, USL versions 2.1 and 3.0 C++ compilers (also known as **cfront**), Kernighan and Ritchie C and SVR4 C, in addition to compatibility with the ANSI C++ and C languages.

This release combines into a single compiler what were, previous to release 5.1, separate C and C++ compilers. Release 5.1 was the "next" release after Concurrent C 4.3 and Concurrent C++ 3.1. This release does not require un-installing previous versions of the C and C++ compilers. It installs in its own unique location and provides a mechanism for supporting the use of multiple releases.

This release provides a command-line-based program development environment for building complex projects. Alternatively, the user may use NightBench™ as an interface with this program development environment.

As of release 5.1, C/C++ no longer ships with USL iostream and complex libraries. They are available separately.

2.0.   Documentation

Table 2-1 lists the Concurrent C/C++ 7.3 documentation available from Concurrent.

Table 2-1. Concurrent C/C++ Version 7.3 Documentation

Manual Name	Pub. Number
Concurrent C/C++ Reference Manual	0890497-040
Concurrent C/C++ Version 7.3 Release Notes (PowerMAX)	0890497-7.3

Copies of the Concurrent documentation can be ordered by contacting the Concurrent Software Support Center. The toll-free number for calls within the continental United States is 1-800-245-6453. For calls outside the continental United States, the number is 1-954-283-1822 or 1-305-931-2408.

Additionally, the manuals listed above are available:

- online using the X Window System™ utility, **nhelp**

- on the Concurrent Computer Corporation web site at **www.ccur.com**

3.0. Prerequisites

Prerequisites for Concurrent C/C++ Version 7.3 are as follows:

3.1. Software

- PowerMAX OS 4.3 or later
- X Window System[1]

3.2. Hardware

- Computer Systems:

    Power Hawk™ 620 and 640

    Power Hawk 710, 720 and 740

    Power Hawk 910 and 920

    PowerStack™ II and III

    Night Hawk® Series 6000

    TurboHawk™

    PowerMAXION™

- Board-Level Products:

    Motorola® MVME2604

    Motorola MVME4604

---

1 This is a prerequisite only for the the Concurrent C/C++ online help package (`c++73help`). It is not necessary for proper operation of the compiler. See "System Installation" on page 4 for more information about these packages.

4.0. System Installation

The Concurrent C/C++ product consists of a number of standard PowerMAX OS software packages and utilizes the standard PowerMAX OS product installation mechanism, **pkgadd** (see **pkgadd(1)**).

The package names are:

invoker
Only one release of the **invoker** package may be installed on a system. It is compatible with the current and all previous versions of the compiler, so the latest release may always be used with previous releases. An **invoker** package of the same or later release must be installed before installing the **c++73** and **c++73help** packages. Because only one release of **invoker** may be installed on a system, a designation of the release name is not included in the package name.

NOTE

To uninstall an older version of **invoker** without uninstalling products that depend on it, use the "**pkgrm -a check invoker**" command.

c++73runtime
If you only need to run dynamically-linked programs built with the C/C++ 7.3 compiler, this is the only package that needs to be installed. Multiple releases may be installed at the same time. Previous releases of the C/C++ compiler named this package **c++runtime** (without any designation of the release in the name), requiring the use of the **-a check** option to **pkgadd** in order to install multiple releases. To reduce confusion when shipping and installing multiple releases of the runtime, a designation of the release name was added to the package name.

c++73
The **c++73** package contains the Concurrent C/C++ 7.3 compiler. An **invoker** of the same or later release must be installed before installing this package. Multiple releases may be installed at the same time.

c++73help
The **c++73help** package contains the online manuals for the Concurrent C/C++ compiler. The installation of **c++73help** is not necessary for the proper operation of the compiler. Multiple releases may be installed at the same time.

The package names are case-sensitive.

NOTE

The above packages must be installed in the order shown.

In addition, the optional **c++73cfront** package containing USL iostream and complex libraries should be installed after the **c++73** package.

Please refer to the "Installing Add-on Software" chapter in the *System Administration Volume I* (0890429) manual and the *PowerMAX OS Release Notes* for instructions on software installation.

4.1.   Installation Options

When you run **pkgadd(1M)** to install the Concurrent C/C++ packages, you are prompted to enter the name of the directory for installation. If you want to install in the default directory, **/usr/opt**, just press the <Enter> key at the prompt. Otherwise, enter the name of the directory where you want Concurrent C/C++ installed. If this directory does not exist, the installation procedures attempt to create it for you.

NOTE

Older versions of Concurrent C/C++ would install in **/** if a directory was not specified. Recent releases of Concurrent C/C++ will install in **/usr/opt** if no directory is specified, *not **/***.

4.2.   Multiple Releases and Pre-5.1 Releases

Release 5.1 of Concurrent C/C++ introduced the capability of having multiple releases installed on the system simultaneously. This allows the system administrator and users greater flexibility in upgrading to new compiler versions that may have incompatibility requiring recompilation or source modification. It will also allow the installation of beta releases without interfering with those users who need to continue using the official version of the compiler.

If a pre-5.1 release of C or C++ is already installed on the system when 5.1 or later is installed, the older release will not be overwritten. Instead, a pseudo-release called **pre5.1** is created for accessing this previous release. After installing 7.3, the following commands will default to release **7.3**:

- **hc**
- **cc**
- **cc++**
- **c++**
- **analyze**
- **report**

If the user wishes to use the pre-5.1 versions of the tools, he or she may:

- Set the environment variable PDE_RELEASE to pre5.1, or
- Set the user's default release using the command:

c.release -rel pre5.1 -u

In addition, the system administrator may issue the command:

c.install -rel pre5.1 -p

This will cause the six commands named above to invoke the **pre5.1** versions, while **ec**, **ec++**, and **c.*** will continue to invoke the 7.3 versions. Note that this setting is overridden by the **–d** option to **c.install** should the system administrator change the system default release.

5.0. Changes in This Release

The following changes were made in Concurrent C/C++ Version 7.3.

5.1. Invoking the compiler

To use the release 5.1 and later of the C/C++ compiler, it is necessary to add the path **/usr/ccs/bin** to your PATH environment variable. The C compiler is invoked by the **ec** command, and the C++ compiler is invoked by the **ec++** command.

If an earlier C or C++ compiler is still installed on the system, the system administrator may choose to configure the previous compiler commands (**cc**, **hc**, **cc++**, **c++**, **analyze**, and **report**) to invoke the pre-5.1 release. However, by default, they will invoke the latest release. The user may override this default behavior by setting the PDE_RELEASE environment variable to either **pre5.1**, **5.4**, or another release name, or by using the **c.release** command to select the user-specific default release.

5.2. Debug Information: location lists

Traditionally, debug information associates one location with a variable. However, modern optimizing compilers (such as Concurrent's) may keep a variable in a variety of locations over the variable's lifetime. For example, a variable stored in memory may be kept in a register during the execution of a loop, or a variable may be bound to several registers over the course of its lifetime. Another drawback of the traditional approach is that a location may not always contain the value of the variable. The location may contain other values at any point where it is not required for the variable, such as before the varaible is first set, or after it is last used. At such times, examination of the variable with the debugger would produce incorrect values.

This release of the compiler implements a more detailed description of variable locations called "location lists". These lists specify where a variable is located as a function of the program counter. Thus, while using the debugger, if the user is stopped inside a loop and the variable has been copied into a register, the register will be used to examine the variable's value. When the user is stopped outside the loop, and the variable exists only in memory, then the memory location will be used to examine or modify the variable. If a variable is in a register, but the user is stopped at a location when the register contains some other value, such as after the last use of the variable, then the debugger will indicate that the variable has no valid location.

It is recommended that NightView 5.4 or later be used to debug programs compiled with location lists.

The **-g** option has been enhanced for location lists to give the user more control over the amount of debug information the compiler generates:

-g0 (*none*)   No debug information is generated. (This is the default if no **-g** option is specified.)

-g1 (*lines*)   Only line number information is generated. (In previous releases, this was the default when no **-g** option was specified.)

-g2 (*simple*)   Variables are described in only the most common location, and for the entirety of their defined scope. (In previous releases, this was the default when the **-g** option was specified.)

-g3 (*full*)   Full location lists are generated for variables. (This is the default if the **-g** option is specified without a numeric modifier.)

5.3.   Debug Information: --compress_debug_info

When compiling under the PDE (using the **c.*** tools or NightBench), there is a new option available:

    --compress_debug_info

This option will direct the compiler to record type definitions in the PDE's database rather than in each individual object file. The result is much smaller debug information in the final executable. Without the same types defined repeatedly from each object file, the NightView debugger will operate much faster.

5.4.   Enhancements to c.script

The **c.script** tool, which is used by NightBench to save an environment to a file, no longer generates a **/bin/sh** script for re-creating an environment. Instead it generates a **/usr/ccs/bin/c.sh** script. The **/bin/sh** script was slow because each command it invoked had to open, modify, and update the database file. The **/usr/ccs/bin/c.sh** tool only needs to open the database one time, interpret all the commands, then close the database when done. Apart from the script running much faster, this change should be transparent to the user unless the user manually edits the script to make changes.

5.5.   static_cast between enum types allowed

The standards committee ruled in Core Issue 128 that static_cast between enums is allowed.

The compiler now allows that.

```
enum E1 { ke1 };
enum E2 { ke2 };
int main () {
  static_cast<E2>(ke1);   // Now allowed
}
```

5.6.   Operator keywords in #if directives

The compiler now correctly processes operator keywords (like compl or new) in #if directives when alternative tokens are enabled (see the **--[no]_alternative_tokens** option).

For example:

```
#if 0 or 1    // Now tests true when alternative tokens are enabled
#if delete 0  // Still invalid, but different diagnostic
#endif
#endif
```

5.7.   Value-initialization

Value-initialization, a refinement of default-initialization introduced in TC1 of the C++ standard (and Core Issue 178), is now implemented. It ensures that all fields of non-POD classes without user-written constructors are properly initialized, by zeroing the storage for the object before calling the constructor.

5.8.   GNU C compatibility option

The compiler now supports command-line switches **--gcc** and **--no_gcc** that provide some language compatibility with GNU C compilers. The compatibility is neither exact (i.e., some extensions may have a slightly different meaning) nor complete (some extensions are not implemented). This compatibility is only provided in C mode; in fact, the command-line switches imply C mode even when no **--c** option is specified.

The set of extensions enabled with the **--gcc** option initially includes:

- variable length arrays
- extended designated initializers
- compound literals (extended over C99 to allow more kinds of initializations)
- extended variadic macros
- identifiers with dollar signs
- C++-style comments
- digraphs
- long long (and related extensions provided by C99)
- hexadecimal floating-point constants
- nonconstant aggregate initializers for automatic variables
- the typeof operator
- the __inline__ (and inline) and __extension__ keywords
- __FUNCTION__ and __PRETTY_FUNCTION__
- zero-sized arrays
- the \e escape sequence
- empty structs and structs with flexible array members
- attributes
- &&label
- goto *expression
- builtin functions
- statement expressions i.e., ({...})
- multi-line strings

- binary conditional operators (`"x ?: y"`)
- extern inline
- the sizeof operator is applicable to `void` and function types and evaluates to the value one
- arithmetic on pointers to `void` and function types is possible (they behave much like `char*`)
- variables can be redeclared with different top-level cv-qualifiers (the new qualification is merged into existing qualifiers)
- locally declared labels
- the (ternary) conditional operation produces an lvalue if its second and third operand are lvalues
- the comma operator produces an lvalue if its second operand is an lvalue
- certain casts preserve the lvalueness of their operands
- case ranges
- union casts
- implicit conversions between integer and pointer types, and between incompatible pointer types, with a warning
- implicit conversions between pointer types that drop cv-qualifiers
- typedef initializers
- **`--preinclude_macros`** option (like the GCC **`--imacros`** feature)
- do-nothing casts to struct or union types
- return expressions in void functions
- oversized bit fields are turned into ordinary fields
- old-style definitions with unpromoted parameter types can follow function prototypes with those same parameter types
- `_Bool` keyword
- `"?"` operators with mixed `void`/non-`void` operands

5.9. `__alignof__` equivalent to `__ALIGNOF__`

The compiler now accepts `__alignof__` as an alternative spelling of `__ALIGNOF__`. The two are exactly equivalent, but the former was added to increase compatibility with other C++ compilers. The `__ALIGNOF__` spelling remains available.

5.10. `#warning` directive

The compiler now supports a `#warning` directive that is similar to `#error`, except that it generates a warning instead of a catastrophic error.

```
#warning Issue this warning
```

This new directive is not recognized in strict ANSI mode.

5.11. Semantic analysis of friends of class templates

The standard currently specifies that semantic analysis of a friend function defined in a class template is done when the enclosing class is instantiated, even if the friend function is never used. Most implementations, however, defer such analysis until the function is used. As a result, there is a body of code that requires such deferral to compile. We believe that the standard should be revised to require such deferral and have changed the compiler accordingly pending consideration of the issue by the standards committee.

```
template <class T> struct A {
  friend void f(A a) {
    g(a);
  }
};

A<int> ai;
```

5.12. Indirect flexible array members in C99 mode

The default C99 mode of the front end now accepts structure definitions whose last field has a struct type containing a flexible array member.

For example:

```
struct F { int i; int a[]; };
struct S {
  int n;
  struct F f;  /* Now accepted in default C99 mode. */
};
```

This is still an error in strict C99 mode.

5.13. Command-line option to allow `long long` in strict mode

Except in C99 mode, the use of `long long` results in a diagnostic in strict mode. The **`--long_long`** option can now be used to override this behavior so that code that uses `long long` can be compiled without diagnostics in strict mode. The **`--long_long`** option must follow the strict mode option on the command line.

5.14. Limiting the amount of instantiation context output in diagnostics

In certain cases, the context information provided for errors during template instantiations can become very lengthy. In many cases much of the context information is unnecessary. The compiler now provides a mechanism to limit the volume of context output. The "context limit" is the maximum number of context entries to be displayed as part of a diagnostic message and may be set with the **`--context_limit`** command-line option.

A value of zero is used to indicate that there is no limit. If the number of context entries exceeds the limit, the first and last *N* context entries are displayed where *N* is half of the context limit. The following is an example of the output when a portion of the context is suppressed using **`--context_limit=2`**. The default limit is 10.

```
detected during:
  instantiation of "void f(T) [with T=int ***]" at line 4
  [ 20 instantiation contexts not shown ]
  instantiation of "void f(T) [with T=int]"
```

5.15. __ALIGNOF__ no longer requires parentheses around argument expressions

The __ALIGNOF__ (or __alignof__) operator now more closely matches the standard sizeof operator in that it can be followed by an expression that is not parenthesized. For example:

```
int a = __ALIGNOF__ a;
```

This change makes the __ALIGNOF__ extension more compatible with similar extensions in other compilers.

5.16. C99, GNU C mode: lowering of bool increment/decrement

In C99 mode and GNU C mode, when a bool value is incremented or decremented it should be set to 1 (true) or 0 (false), respectively. The generated code for those cases now does that.

5.17. GNU C: variable-length array fields of local structs

The GNU C compiler accepts variable-length array fields in local struct declarations, which may result in size and offset computation being delayed until run time. However, it common for this feature to be used simply to indicate that the last field of a structure is a flexible array member. To enable the latter idiom, the compiler now accepts such declarations in GNU C mode, but it discards the given length with a warning, and sets the length of the array to zero. For example:

```
void f(int n) {
  struct S {
    int a[n]; /* Warning: equivalent to "int a[0];". */
  };
}
```

5.18. GNU C compatibility: "auto" ignored in file scope

In GNU C mode, the front end ignores the auto specifier with a warning when it appears on a file-scope declaration.

```
auto int i;  // Warning in GNU C mode; error otherwise
```

5.19. GNU C: incomplete array parameter types

In GNU C mode, the compiler now accepts the following code:

```
void f(int x[][]) {}
```

Within the definition of the function, the parameter x is treated as being a pointer to an incomplete type (e.g., it cannot be subscripted).

5.20. GNU C: --short_enums option

The compiler now accepts the **`--short_enums`** option in GNU C mode. It indicates that all the enumeration types should be treated as if they were declared with the `packed` attribute (i.e., the underlying type should be the smallest integer type that can accommodate the enumerator constants).

5.21. GNU C compatibility: parameter names in unprototyped declarations

The compiler now accepts parameter names in unprototyped function declarations that are not definitions. A warning is issued (in strict C mode an error is issued).

```
void f(i);  /* Accepted with a warning in GNU C mode. */
```

5.22. GNU C mode: typedefs with sign or size modifiers

The compiler now accepts the modifiers `short`, `long`, `signed` and `unsigned` on `typedef` types that are synonyms for builtin integer or floating-point types.

```
typedef int Int;
typedef unsigned Int UInt; /* Accept in GNU C and pcc modes; error
                              in other modes. */
```

5.23. How conversion functions compete with constructors

The C++ standards committee in closing Core Issue 243 without action affirmed a piece of the C++ standard we weren't sure was intended - that constructors win out over conversion functions in direct-initializations. Accordingly, we have changed the compiler to follow that rule, except in `cfront` mode.

```
extern "C" int printf(const char *, ...);
struct B;
struct A {
  A(B);
};
struct B {
  operator A() { printf("B::operator A()\n"); return A(*this); }
} b;
A::A(B) { printf("A(B)\n"); }
void f(A) {}
int main() {
  f(A(b));  // Prints A(B)
  f((A)b);  // Prints A(B)
  return 0;
}
```

5.24. GNU C mode: warning only on invalid uses of "`inline`"

In GNU C mode, declaring a variable `inline` causes the front end to only issue a warning (rather than an error). The keyword is ignored after the warning is issued.

```
int inline x;  /* Warning in GNU C mode; error in other modes. */
```

The use of `inline` in block-extern declarations of functions is treated similarly:

```
void f() {
  inline void g();  /* Warning in GNU C mode;
                       error in other modes. */
```

5.25. Template function with explicit arguments treated as simple function

In accordance with the proposed resolution to Core Issue 115, the compiler now treats a template function with explicit template arguments as a simple function if the arguments are sufficient to select a unique template function.

```
template <class T> void f(T);
int main() {
  &f<int>;  // Okay now
}
```

5.26. Select C99 features

The C99 features Variable Length Arrays and Complex Arithmetic have been added.

5.27. --[no]_vector_safe_prologs compile option

The **--vector_safe_prologs** compile option places a runtime test in function prologs to avoid executing AltiVec instructions in the prolog. This makes it possible for the user to write a single function that uses AltiVec instructions if available, or non-vector operations otherwise.

5.28. -Qall_variables_in_memory

This customer-requested option forces all variables to reside in memory.

WARNING

Since this suppresses holding variables in hardware registers,
code quality will be significantly impacted.

This option may be helpful in debugging problems in user code, as it allows setting address traps on all variables.

5.29.   Code quality improvements

A number of code quality improvements have been made.

6.0.   Cautions

6.1.   va_list structure format

When using PowerMAX OS 5.1 or later headers, **varargs** and **stdarg** use a different format for the
va_list structure. Because of this, the user should avoid passing va_list structures to routines that
are built with previous versions of the PowerMAX OS headers.

Code built with PowerMAX OS 5.1 or later headers can handle both the old and the new format of
va_list. There should therefore be no problem with old code, compiled with PowerMAX OS 5.0 and
earlier headers, passing va_list to code compiled with PowerMAX OS 5.1 headers.

For example, a dynamically-linked program that calls vfprintf and that is compiled under
PowerMAX OS 4.3 will have no problem running on a PowerMAX OS 5.1 or later system, calling the
vfprintf in the 5.1 or later version of **libc.so**. That same program compiled under PowerMAX OS
5.1 or later will not work if run on a PowerMAX OS 4.3 system. The vfprintf in the 4.3 version of
libc.so will not know how to handle the new va_list format.

6.2.   Retain source

Users are encouraged to retain the source for their applications. Major releases may have changes in the
object-file format which will require the recompilation of their programs. This release is one such
release. In the process of implementing additional ANSI/ISO C++ features, some changes to "name
mangling" and interfaces to runtime routines were necessary.

6.3.   curses.h and bool

The **curses.h** header file contains a bool type definition. In compilation modes where **bool** is a
keyword, this results in a compilation error. There are two workarounds. The user may turn off the **bool**
keyword with the **--no_bool** option, or he may use the following sequence when using the header file:

```
#define bool _curses_bool
#include <curses.h>
#undef bool
```

When using this second workaround, if the user makes reference to curses' bool type definition, the user
must use the name **_curses_bool** instead.

6.4.   Structure compares

The C 4.3 and earlier releases supported an enhancement that allowed the user to implicitly compare a
structure to zero (meaning all bytes of the structure are zero) in an if or while statement:

```
struct S {int a,b,c,d,e,f;} s;
...
if (s) {
...
```

As this enhancement is not present in any other major C compiler and is undocumented in the Concurrent C compiler, it is no longer present in C/C++ 5.1. The user can easily write a small function to test a structure for being zero should any code actually use this feature.

7.0. Direct Software Support

Software support is available from a central source. If you need assistance or information about your system, please contact the Concurrent Software Support Center at our toll free number 1-800-245-6453. For calls outside the continental United States, the number is 1-954-283-1822 or 1-305-931-2408. The Software Support Center operates Monday through Friday from 8 a.m. to 5 p.m., Eastern Standard Time.

You may also submit a request for assistance at any time by using the Concurrent Computer Corporation web site at **http://www.ccur.com/isd_support_contact.asp** or by sending an email to **support@ccur.com**.