Doc. No.	X3J16/96-0151R2
	WG21/N0969R2
Project:	Prog. Lang. C++
Date:	11 July 1996
Reply to:	Dag Brück
	dag@dynasim.se
Ref. Doc.	WG21/N0928

Core-3 working group: exception handling issues.

Issue 647

Amend the working paper 15.1 [except.throw] paragraph 4 (beginning) by changing:

> The memory for the temporary copy of the exception being thrown is allocated in an implementation-defined way.

to

> The memory for the temporary copy of the exception being thrown is allocated in an unspecified way.

Issue 541

Amend the working paper 15.3 [except.handle] by adding the following paragraph after paragraph 11:

> Exceptions thrown in destructors of objects with static storage duration or in constructors of namespace-scope objects are not caught by a function-try-block on **`main().`**

Issue 542

Amend the working paper 15.3 [except.handle] by replacing paragraph 2 including editorial box with the following text:

> A *handler* is a match for a *throw-expression* with an object of type **`E`** if
>
> - The handler is of type cv$_{opt}$ T or cv$_{opt}$ T&, and **`T`** and **`E`** are the same type (ignoring the top-level *cv-qualifier*s), or
>
> - The handler is of type cv$_{opt}$ T or cv$_{opt}$ T&, and **`T`** is an unambiguous public base class of **`E`**, or
>
> - The handler is of type cv$_{opt}$ T* cv$_{opt}$, and E is a pointer type that can be converted to the type of the handler by a standard pointer conversion (4.10) not involving conversions to pointers to private or protected or ambiguous base classes, or a qual-ification conversion (4.4), or a combination of these two.
>
> Footnote: **`handler.is(cv_opt_T_star_cv_opt) && E.is(pointer type) && handler.type().standard_pointer_conversion(E, 4.10) && !find_if(T.standard_pointer_conversion(E), conversion(private_base ∧ protected_base ∧ ambiguous_base) | conversion(qualification))`**

Issue 587

Amend the working paper 15.3 [except.handle] paragraph 1 by appending:

> The exception declaration shall not denote a pointer or reference to an incomplete type, other than **`void*, const void*, volatile void*`** or **`const volatile void*.`**

Issue 648

No change to working paper. Whether the stack is unwound before or after **`terminate()`** is called is implementation-defined, not unspecified.

Issue 588

Amend the working paper 15.4 [except.spec] by adding the following text to paragraph 1 (after the example):

> A type denoted in an exception-specification shall not denote an incomplete type. A type denoted in an exception-specification shall not denote a pointer or reference to an incomplete type, other than **`void*, const void*, volatile void*`** or **`const volatile void*.`**

Issue 631

Amend the working paper in the following way:

1. Remove paragraph 5 of 15.4 [except.spec].
2. Split paragraph 2 of 15.4 [except.spec] after the first sentence, giving paragraphs 2a and 2b.
3. Append the following text to paragraph 2a of 15.4 [except.spec]:

> A diagnostic is only required if the sets of *type-id*s are different within a single translation unit.

Issue 657

Follows from resolution of issue 631.

Issue 649

Amend the working paper 15.5.1 [except.special] by deleting:

> when the implementation's exception handling mechanism encounters some internal error.

Issue 651

Amend the working paper 15.5.2 [except.unexpected] paragraph 1 by changing

> is called (_lib.exception.unexpected_).

to

> is called (_lib.exception.unexpected_) immediately after completing the stack unwinding for the former function.

Single definition of uncaught_exception()

Amend the working paper by changing the “Returns” part of 18.6.4 [lib.uncaught] to:

> Returns: true after completing evaluation of a throw-expression until completing initialization of the exception-declaration in the matching handler (_except.uncaught_). This includes stack unwinding (_except.ctor_).

and changing the contents of 15.5.3 [except.uncaught] to:

> See 18.6.4 [lib.uncaught].

Unexpected handler during stack unwind

Amend the working paper by changing paragraph 1 of 18.6.2.4 [lib.unexpected] to:

> Called when a function exits via an exception not allowed by its exception-specification (_except.unexpected_).
>
> Effects: Calls the unexpected_handler function in effect immediately after evaluating the throw-expression (_lib.unexpected.handler_).

Terminate handler during stack unwind

Amend the working paper by changing the Effects part of paragraph 1 of 18.6.3.3 [lib.terminate] to:

> Effects: Calls the terminate_handler function in effect immediately after evaluating the throw-expression (_lib.unexpected.handler_).

No incomplete type in throw-expression

Amend the working paper 15.1 [except.throw] by changing paragraph 3 to:

> A throw-expression initializes a temporary object of the static type of the operand of throw, ignoring the top-level cv-qualifiers of the operand's type, and uses that temporary to initialize the appropriately-typed variable named in the handler. The type of the throw-expression shall not be an incomplete type, nor a pointer or reference to an incomplete type, other than **`void*, const void*, volatile void*`** or **`const volatile void*.`** Except for these restrictions and the restrictions on type matching mentioned in _except.handle_, the operand of throw is treated exactly as a function argument in a call (_expr.call_) or the operand of a return statement.