WG14 N1587
INCITS PL22.11/11-002
Reply To The Attention Of: Barry Hedquist
PL22.11 Secretary
Email: beh@peren.com

ISO/IEC JTC1 SC22/WG14 AND INCITS PL22.11
MAR 14-18, 2011, MEETING MINUTES

Meeting Location:
BSI
389 Chiswick High Road
Room 507
London
W4 4AL
United Kingdom
Host Contact information:

Rod Grealish
E-mail: rod@grealish.freeserve.co.uk

Meeting Dates: 14 - 18 March, 2011

Meeting Times:
14 March 2011 09:00–12:00 13:30–16:30
15 March 2011 09:00–12:00 13:30–16:00
16 March 2011 09:00–12:00 13:30–16:30
17 March 2011 09:00–12:00 13:30–16:30
18 March 2011 09:00–12:00

1. OPENING ACTIVITIES

1.1 Opening Comments (Grealish, Benito)

John Benito welcomed us to the BSI offices in the Chiswick area, west of London, described the meeting facilities. Several local restaurants are within walking distance of the facility, and there is a cafeteria in the building on Floor #1. Lunch break will be from 12:00 - 13:30.

1.2 Introduction of Participants/Roll Call

Name	***Organization***	***NB***	***Comments***
John Benito	Blue Pilot		WG14 Convener
Blaine Garst	Apple	USA	
David Keaton	CMU/SEI/CERT	USA	PL22.11 Chair
Tana L. Plauger	Dinkumware, Ltd	USA	
P. J. Plauger	Dinkumware, Ltd	USA	
Jim Thomas	HP	USA	
Rajan Bhatka	IBM	Canada	HOD - Canada

Steve Montgomery	MISRA Liaison	UK	
Douglas Walls	Oracle	USA	HOD – USA, PL22.11 IR
Barry Hedquist	Perennial	USA	PL22.11 Secretary
Tom Plum	Plum Hall, Inc.	USA	
Fred Tydeman	Tydeman Consulting	USA	
Larry Jones	Siemens		WG14 Project Editor
Nick Stoughton	Usenix	USA	Austin Group Liaison
Bill Seymour	Seymour	USA	
Derek Jones	Knowledge Software	UK	
Willem Wakker	ACE	Netherlands	HOD – Netherlands
Nat Hillary	LDRA	USA	
Joseph Myers	CodeSourcery	UK	HOD - UK
Lawrence Crowl	Google	USA	phone
Hans Boehm	HP	USA	phone
Mike Wong	IBM	Canada	phone
Niall Douglas	NED Productions Ltd.	Ireland	
Martin Sebor	Cisco	USA	

1.3 Procedures for this Meeting (Benito)

The meeting Chair, John Benito, announced the procedures are as per normal. Everyone is encouraged to participate in straw polls.

INCITS PL22.11 members reviewed the INCITS Anti-Trust Guidelines at:

http://www.incits.org/inatrust.htm.

All 'N' document numbers in these minutes refer to JTC1 SC22/WG14 documents unless otherwise noted.

Straw polls are an informal mechanism used to determine if there is consensus within the meeting to pursue a particular technical approach or even drop a matter for lack of consensus. Participation by everyone is encouraged to allow for a discussion of diverse technical approaches. Straw polls are not formal votes, and do not in any way represent any National Body position. National Body positions are only established in accordance with the procedures established by each National Body.

The term "WP" means Working Paper, the latest draft version of the revision to the ISO/IEC 9899 C Standard, also known as C1X.

Emphasis for this meeting is resolution of Ballot Comments from the CD Ballot.

Barry Hedquist, PL22.11 Secretary, is the Recording Secretary for the meeting.

1.4 Approval of Previous Minutes - Batavia (N1541) (Hedquist)

Several comments for typos, etc.

Minutes were modified per editorial changes and approved.

Final Batavia Minutes are **N1557.**

1.5 Review of Action Items and Resolutions (Hedquist)

ACTION: Convener to discuss the resolution of N1448 with the Submitter, Steve Adamczyk.
CLOSED – Question covered in a footnote.

ACTION: David Keaton to work with Joseph Myers and write a proposal w/r/t SC22WG14.12205, Anonymous Structures.
CLOSED N1549

ACTION – Tom Plum to research whether char should be a standard integer type.
CLOSED – No Change

ACTION – Larry to look into non-editorial issue #6, harmonization of the description of signal handling.
OPEN

ACTION – WG21 liaison to clarify WG21 concerns regarding non-support of atomic floating point operations.
CLOSED – Neither group made any changes.

1.6 Approval of Agenda (N1552) (Benito)

Revisions to Agenda: Martin Sebor
Added Items: None
Deleted Items: None

Agenda approved as modified.

1.7 Distribution of New Documents (Benito)
None

1.8 Identification of National Body Delegations (Benito)
US, Canada, UK, Netherlands

1.9 Identification of PL22.11 Voting Members (Tydeman)

See PL22.11 Minutes, following these minutes. 15 of 18 members present.

1.10 Information on Next Meeting Schedule (Nxxxx) (Benito) (TBD)

The Fall 2011 meeting will be held at TBD.

2. LIAISON ACTIVITIES

2.1 WG14 / PL22.11 (Benito, Walls, Keaton)

London: PL22.11 No Report, WG14 call for Convener answered by John Benito of the US.

2.2 WG21 / PL22.16 (Plum, Benito, Sutter)

The ballot for the WG21 C++ Final Committee Draft, FCD 14882, closed on 26 July, 2010. The committee will meet next week in Madrid to finalize Ballot Resolution, and *plans* to vote out an FDIS at that meeting. Tom pointed out that much remains to be done, so it's a bit difficult to predict what will happen in Madrid.

2.2 Austin Group (Stoughton)

No formal report. Comments submitted on CD1 Ballot.

2.4 PL22 Programming Languages (Plum)

PL22 meets twice a year via teleconference. A meeting will be held in late July 2011 to establish US positions to SC22 Plenary.

2.5 WG11
Revision of Part I produced, but very little interest in further work. Part I is in SC22 Ballot

2.6 WG23 (Benito)

A revision to their TR is underway. Meeting in Madrid next week.
For further information, contact John Benito.

2.6 MISRA C (Montgomery)

MISRA working on Rev 3 of their C Guidelines, based on C99, and correcting issues in prior versions. Public review version expected by June, 2011. Let Steve know if you wish to take part in the review.

2.7 Floating-point Study Group (Thomas)

Jim Thomas. Nothing new. Making slow but steady progress, meeting via teleconference once a month. There is a Wiki. Contact Jim Thomas for details.

2.8 Secure Coding Study Group (Keaton)

Work is progressing. Will propose a Technical Specification by Sept next year (2012). Meeting here, London, on Friday. Everyone is invited.

2.10 Other Liaison Reports - none

3. EDITOR REPORTS

3.1 Report of the Rationale Editor (Benito)
Benito expects to have a rationale ready by the next meeting.

3.2 Report of the Project Editor (Jones)

6. The description of signal handling in 7.14.1.1p5 should be harmonized with the description in 5.1.2.3p5.

*The description of a signal handler in p2 should probably note that functions called indirectly via standard library functions like **abort** (when **SIGABRT** is being caught) are also considered to be part of the handler.*

9. The index probably needs work, particularly for the newly added material.

If anyone has an item to add to the index, let Larry know.

London: No new items to report.

3.3 Report on Editorial Meetings Held (Benito, Jones)

None held, WP was in ballot.

4. FUTURE

4.1 Future Meeting Schedule

Fall 2011 - Portland (tentative) October? September looks better. Week of 26th

Future meetings could be at INCITS, Santa Cruz, or we can get sponsors for Europe. What does the group want to do? How can we get the experts we want to attend meetings? Teleconferencing is economical, but technical discussions can get lost. Jim noted that teleconference seems to work best if everyone is conferencing. Tom works with a group that is split 50/50 present / teleconference.

4.2 Future Agenda Items

Per Normal

4.3 Future Mailings

Post London:	18 April, 2011
Pre Fall:	TBD

5. CD Ballot Resolution (N1548, N1553, N1554)

Applicable documents:
N1548: ISO/IEC CD 9899, Committee Draft Balloted, included in pre-meeting mailing.
N1553: NB Comments
N1554: Austin Group Comments

US 1 Generic Selection
Editorial - ACCEPT

US 2, 7.12.1
Comment: "... the value corresponding to the error ..." is missing the correspondence.

Proposed Response: The correspondence is: “invalid” => EDOM; “divide-by-zero” => ERANGE; “overflow” => ERANGE; “underflow” => ERANGE. It might be better as a table.

RESPONSE: Rejected – The Standard is clear as written.

US 3 – Moved to Editorial, REJECTED. The Standard is clear enough.

US 4, 6.5.3.4 – Rejected, no consensus to make this change.

US 5 Editorial - Accepted

US 6 Editorial - Rejected. Already there.

US 7 Editorial - Rejected. Already there.

US 8 Editorial - Accepted

US 9 Editorial - Accepted

US 10, 7.3.9.3
Comment: If the CMPLX macros are not useable in static initialization, then they have little value.
Proposed Response: Remove “Recommended practice” and change “should” to “shall”.

The comment is true.

RESPONSE: ACCEPT (UNANIMOUS CONSENT)

US 11, F.10.3.5
Comment: It is ambiguous if ilogb(NaN) is outside the range of the return type. The correct value for ilogb(NaN) is NaN. But, since NaN is not representable in int, “invalid” should be raised and an unspecified value returned. But, 7.12.6.5 specifies FP_ILOGBNAN as the return value (which some people say is in the range of the return type). Same problem applies to zero and infinity.

Proposed Response: Add a 3rd paragraph: ilogb(x), for x zero, infinite, or NaN, raises “invalid” and returns the value as specified in 7.12.6.5.

RESPONSE: ACCEPT (UNANIMOUS CONSENT)

US 12, 6.10.8.3
Comment: Need a way to distinguish freestanding from hosted.
Proposed Response: __STDC_FREESTANDING__ The integer constant 1, intended to indicate a freestanding environment.

RESPONSE: REJECTED: __STDC_HOSTED__ indicates what is asked for.

US 13 - Editorial. Rejected. Arranged as is.

US 14 - Editorial. Rejected. The proposed change confuses the issue.

US 15, Sec 4;p4
Comment: A program that violates C's syntax should not be translated.

Proposed Response: Add "Recommended practice – The implementation should not successfully translate a preprocessing translation unit that violates any syntax (has an erroneous program construct)."

Little support for this change. Martin and Fred in favor. Martin believes that some guarantees be offered. Most compilers already have an option to convert warnings to errors.

RESPONSE: REJECTED – No consensus to make this change.

US 16, Sec 4;p8
Comment: It would be nice if a programmer could find out how to invoke an implementation in Standard C conformance mode.

Proposed Response: Add to end of sentence: and how to invoke the implementation in conforming mode.

We made a decision years ago not to talk about switches and plugs. Joseph believes this is in play already. Tom believes that the burden of proof is on the implementer, and would like to see this added. The documentation could be onerous.

STRAW POLL: Adopt US 16.
Result: 6-9-3 No Consensus
RESPONSE: REJECTED – No Consensus.

US 17, Clarifications to Anonymous Structures and Unions (N1549) (Keaton)
Sec 6.7.2.1;p8 & 13

N1549 proposes a resolution to US 17, and matches the proposed resolution presented therein.

Comment: Anonymous structures and unions need minor clarification.
Proposed response: Changes along the lines of N 1549 should be adopted.

Joseph Myers pointed out that the current wording of the C standard could be interpreted to mean that a typedef-name could be used to declare an anonymous structure or union. This was not intended because it is a gratuitous difference from C++.

Joseph also pointed out that some of the wording makes it unclear whether a flexible array member can be preceded by only anonymous structures and unions. The confusion occurs because a flexible array member must be preceded by a "named member." It was intended that anonymous structures and unions bring named members into the scope of their parent structure or union (recursively).

The following wording is proposed to eliminate these problems.

2.1 Changes to subclause 6.7.2.1

In paragraph 8, change the third sentence to the following.

> If the struct-declaration-list does not contain any named members, either directly or via anonymous structures or anonymous unions, the behavior is undefined.

In paragraph 13, change the first sentence to the following.

> An unnamed member whose type specifier is a structure specifier with no tag is called an *anonymous structure*; an unnamed member whose type specifier is a union specifier with no tag is called an *anonymous union*.

RESPONSE: ACCEPTED (UNANIMOUS CONSENT)

US 18, Editorial. Accepted

US 19, Sec 7.3, 7.15, 7.18

Comment: There are headers that define macros "complex", "bool", "alignas" for keywords "_Complex", "_Bool", "_Alignas" etc.

But we could not find a header defining the macro "noreturn" for "_Noreturn".
Nor could we find a header defining the macro "thread_local" for "_Thread_local".

We think there should be.

Proposed Response: Add header files along the lines of <stdbool.h> to define the noreturn and thread_local macros.

Discussion: Where to put these is not clear. We want a header file that brings in only the macros wanted, not all. _Generic (new C1X keyword) falls into the same category. Tom: does not want a separate header file for _Generic, because it is specialized. Would like to see a header file for _Noreturn, that would allow for the C++ version. Jim: Do not need a header file to define _Noreturn. _Thread_local can go into threads.h. Nick: suggests attribute.h, as _Noreturn is the remaining attribute when we were considering attributes. alignas.h is another possibility.
Joseph: stdnoreturn.h (first 8 characters must be unique). Preferred over noreturn.h.
Rajan: needs to check on the 8 character requirement. IBM may limit its names to 8 characters.

Response: ACCEPT with MODIFICATION.
Add stdnoreturn.h for _Noreturn. (IBM may come back on this.) Put _Thread_local in threads.h.

US 20, Sec 6.4.1
Comment: Why is "alignof" a new keyword, instead of "_Alignof" with a header to define alignof macro? Seems needlessly inconsistent

Proposed Response: Change the “alignof” keyword to “_Alignof” and add a header file along the lines of <stdbool.h> to define the alignof macro.

Discussion: Why did we do this? No one can really recall, but this is a mistake. Does not matter.

Response: Accept with Modification, use stdalign.h as the header.

US 21, 7.17.6;p1
Comment: It was never the intention to require that the atomic_* types be defined in terms of the _Atomic keyword, and this paragraph causes major problems with C++ compatibility. The atomic_* types must be implementable as struct so that they can serve as base classes for their atomic<*> counterparts.

Proposed Response: For each line in the following table, the atomic type name behaves the same as the corresponding direct type. (NOTE: The atomic type name may be a typedef for the direct type, or it may be a struct.)

Tom: There are three ways to spell the same type, as proposed above. Wants "behaves the same as" to mean "the same as". PJ wants the wiggle room to make it a struct. Blaine agrees. Lots of discussion around the words and whether or not we really want these to be the same. Same representation and alignment with the standard footnote.

RESPONSE: Accept with Modification, along the lines above.

ACTION: Larry to propose words for US-21.

US 22, US 23, Thread Support in the Library (N1551) (Boehm)

In response to US 22, and US 23**,** N1551 proposes to address a number of library issues that arise as the result of the introduction of threads in C1X. The author notes that not all the precise wording needed in adopting this proposal is included, but that it can be derived from the POSIX standards. The intent of this paper is to more closely align C1X threads with existing practice.

Hans Boehm, Michael Wong, and Lawrence Crowl joined us via telecom. Many minor issues, and some major items that require a policy decision by the Committee.

US – 22
N1551: Issue #6 STDIO – Add POSIX functions: flockfile, funlock, getc_unlocked, putc_unlocked, getchar_unlocked, putchar_unlocked.

Blaine believes leaving it ambiguous is OK. Users will find out quickly enough, and the current state reflects current practice. PJ: This is adding a bit too much. Bill: Agrees with PJ. Blaine: Move toward an Annex K solution in the future, but going here now is a huge step. Nick believes the current draft requires stderr to use locking. PJ: Folks who use this scenario use a single thread environment, and the problem goes away.

There are no real words here, so we will need a directional, along-the-lines-of, vote in order to proceed with this.

Blaine would like to add some of the proposed words, but none of the proposed functions. Nick agrees. Hans agrees. Words developed, see N1551.

DECISION: Adopt the first paragraph of proposed wording for STDIO, N1551. (7.21.2;p6) (UNANIMOUS CONSENT)

Nick: Proposes words to flag a known problem in multi-threaded environment.

POSIX sends signals to specific threads, and it is no longer specified how the signal() function works in the presence of multiple threads. In its place, the sigaction() API is introduced, along with a set of supporting functions (e.g. sigwait, sigqueue).

See http://pubs.opengroup.org/onlinepubs/9699919799/functions/V2_chap02.html#tag_15_04 for more information.

In POSIX, the raise() API is defined in terms of pthread_kill, which sends a signal to a specific thread.

The problem is that when an asynchronous signal occurs, which thread should handle it? Should all threads receive it? When a thread synchronously raises an signal, should another thread be allowed to handle it?

At very least, we should make signal undefined.
Agreement by everyone.

DECISION: Add to 7.14.1.1, at the beginning of Para 2: "Use of this function in a multi-threaded program results in undefined behavior." (UNANIMOUS CONSENT)

There is a question of whether or not to say the same thing to raise(). My feeling is that raise() is OK, and implementations can be left to do the right thing with this. However, I would not object to raise also becoming unspecified/undefined or

Nick is prepared to write words for sigaction etc for WG14. However, he does not expect the WG to want to handle it for the current draft. Perhaps such a text might want to come in as a Technical Specification?

US 23

US 23 - N1551: Issue #1: strerror, strtok, rand and asctime.
These functions are not thread-safe. Wants to invoke the reentrant versions of these functions, strerror_r, etc. Using the "_s" functions is also a possibility, but not as well established. PJ has a list of functions that have writable static storage, thus subject to issues in a multithread environment. About 30-40 functions. The issues raised by Hans are only the tip of the iceberg. Nick: POSIX has a list of 78 thread unsafe functions, but most are POSIX functions. But only these four are common to the C Standard as thread unsafe functions. It's not possible to write them in a thread-safe way.

Tom: What has C++ done about these functions? Nothing. Tom does not see what else can be added at this date. David agrees, but also agrees that using the "_s" functions is a good solution. rand() is not included because it's a not very useful anyway.

Blaine sees the only practical solution is to list thread-safe functions in an annex, but we have little room to dictate the thread-safeness of library functions.

PJ sees using the POSIX solution set is a better way to go. JB pointed out that we have customers other than POSIX. Nick: specify which library functions are NOT thread-safe. PJ agrees, would rather list the thread-unsafe functions that adding the "_r" reentrant functions.

Hans does not want to lose the strerror function for threads. It's needed, but must be thread-safe. Wants to require strerror_s with threads. Future direction? Nick prefers moving strerror_s into the main body of the Standard.

The tools are there to solve this problem. strerror_s exists is Annex K, and in POSIX. For practical purposes, this is not really an issue.

Tom suggested that the Project Editor has the discretion to add a footnote for clarification.

Hans: the existing wording regarding data races is unclear. We can't list everything a programmer should not do. PJ: putting prohibitions in the standard give testers something to go gunning for.

Willem: Possible rationale material.

Summary: No support for a change. Should choose annex K if you use the Threads option. The tools already exist.

STRAW POLL
Q: Do we want to add thread-safe functions for strerror, strtok, rand and asctime to the main body of the Standard.
Result: 7 – 14 - 1
Decision: No consensus to add these functions.

Discussion has moved to memcpy. It's related to the first proposed wording change "Library functions…".
Decision: Ed – footnote. "For example, a memcpy implementations…"

US 23 - N1551 Issue #2: Replace wording for strtok, strerror, and rand.
Wording is clear for users, but Hans believes it is too liberal for a multi-threaded implementation. Blaine believes that is the case only for the unwise.

STRAW POLL: David proposed: The <function> is not required to avoid data races with other calls to <function>. Substitute for <function>, strerror, strtok, and rand, respectively.
Result: Lots – 0 – 2
DECISION: Adopt words as above.

US 23 - N1551: Issue #3: Same issue as #2 for srand.
Joseph recommends adding for rand and srand the same basic wording as above.

STRAW POLL: Add the following words as appropriate to each of the functions rand and srand.
"The rand and srand functions are not required to avoid data races with other calls to rand and srand."

Poll: Unanimous Consent

DECISION: Add the following words as appropriate to each of the functions rand and srand.
"The rand and srand functions are not required to avoid data races with other calls to rand and srand."

US 23 - N1551: Issue #4: setjmp/longjmp (N1556)
Agreement by Hans and Nick to say something. Nick likes to POSIX wording, but Hans was a little uncomfortable with the wording. Blaine believes the proposed wording adds clarity.
David: Changing this requires a change to Annex L.

STRAW POLL: Adopt proposed wording in N1551, as modified below in N1566:
Result: UNANIMOUS CONSENT

At 7.13.2.1 para 2, Change:

The longjmp() function shall restore the environment saved by the most recent invocation of setjmp() in the same thread, with the corresponding jmp_buf argument. If there is no such invocation, or if the function containing the invocation of setjmp() has terminated execution in the interim, or if the invocation of setjmp() was within the scope of an identifier with variably modified type and execution has left that scope in the interim, the behavior is undefined.

with

The longjmp() function shall restore the environment saved by the most recent invocation of setjmp() in the same invocation of the program, with the corresponding jmp_buf argument. If the most recent invocation of setjmp() with the corresponding jmp_buf occurred in another thread, or if there is no such invocation, or if the function containing the invocation of setjmp() has terminated execution in the interim, or if the invocation of setjmp() was within the scope of an identifier with variably modified type and execution has left that scope in the interim, the behavior is undefined.

Also, at para 4, change:

After longjmp is completed, program execution continues as if the corresponding invocation of the setjmp macro had just returned the value specified by val.

To:

After longjmp is completed, thread execution continues as if the corresponding invocation of the setjmp macro had just returned the value specified by val.

Result: 11 – 0 – 7

DECISION: Add the proposed wording contained in N1566 as above.

Issue #5: malloc/free
minor issue. adds text describing data race behavior. David: word tweek. Larry has it.
Poll: UNANIMOUS CONSENT

US 24 - N1521, mtx_try,
Remove mtx_try from the WP. Hans knows of no implementations that use it, believes if it's there it will either be ignored or used wrong. PJ will not go to the wall to defend mtx_try. It's in his implementation, it's tested, and it works. Douglas is in favor of removing it. We considered this in Batavia, and there was no consensus to remove mtx_try.

STRAW POLL: Adopt US 24 (N1521, remove mtx_try)
Result: 11 – 5 – 5

DECISION: Consensus exits to Adopt US 24 (N1521, remove mtx_try)

US 25, Clause 7. See Also CA 1
Comment: The _Atomic type qualifier should be removed. It is redundant, and its use needlessly hinders C++ compilation of code.

Proposed Response: Remove _Atomic type qualifier.

Discussion: C1x_atomics_1_2.doc. Initial assumption in this paper, that

int func(_Atomic(int))

can have two meanings. It cannot. It can only be a func that takes an atomic int.

STRAW POLL: In favor of US-25, and CA-1 as written.

Result: 4-14-4

RESPONSE: Rejected, no consensus to make this change.

CA 1, Clause 7
Comment: The _Atomic type qualifier should be removed. It is redundant, and its use needlessly hinders C++ compilation of code.

Proposed Response: Remove _Atomic type qualifier.

Discussion: C1x_atomics_1_2.doc. Initial assumption in this paper, that

int func(_Atomic(int))

can have two meanings. It cannot. I can only be a func that takes an atomic int.

STRAW POLL: In favor of US-25, and CA-1 as written.

Result: 4-14-4

RESPONSE: Rejected, no consensus to make this change.

CA-2
Comment: Remove atomic_address in C1x.

Proposed Change: We (WG21) have removed atomic_address in C++0x. This was removed because it was a base class of the pointer specialization, which leads to no type safety.

STRAW POLL: Accept CA-2

Result: Unanimous Consent

RESPONSE: ACCEPT

CA-3

Comment: The current draft supports too many compound operations like atomic divide assign, atomic float for arithmetic operations. It is trying to be too general making every compound operators atomic. C++ selectively narrowed the operations, based on what current hardware will not have trouble supporting.

Proposed Response: Until we specify what they mean, what are the traps, we would prefer that C1x limits it to the same list as C++0x. Additional operations can be added. Original C1x paper implies that these operations can be written as if it is written with a compare exchange loop and that might work, but we need to understand it better. The limited set of operations that C++ supports is listed in Table 1 below these comments.

Here is the limited set of operations that C++ supports that we would like C to restrict::

	type made atomic			
Operations	bool	T*	integral	others
is_lock_free	Y	Y	Y	Y
load, store, exchange, compare_exchange (weak+strong)	Y	Y	Y	Y
fetch_add, +=, fetch_sub, -= ++,--		Y	Y	
fetch_or, |=, fetch_and, &=. fetch_xor, ^=			Y	

Y – REQUIRED
Blank – permitted to implement, not required
NOTE: Table above corrected to move ++,-- to above it's prior (incorrect) position.

Lawrence: need very specific semantics, and C's are not tight enough. Sees this as a 'vast pit of undefined behavior'. Is there existing practice for this material? Tom leaning toward agreeing with the proposal since due to lack of existing implementations and C/C++ compatibility. Blaine: agrees we should focus on existing practice, but also see this as a matter of time for implementing. Blaine believes it can be implemented. Derek: does not believe we should downgrade ourselves w.r.t. C++ for these items. David: The real issue is what we can do in hardware. Lawrence: The floating point operations are undefined, there is no way to protect yourself. PJ: We've indulged in massive invention. The whole machinery is huge and complex. But, really wants to leave this in, and live with the consequences. It will be a mess for at least five years.

STRAW POLL: Adopt CA -3.
Results: 5-7-8
Decision: No Consensus to Adopt CA – 3.

CA 4, Sec 6.10.8.3 (also C1x_atomics_1.2.doc)
Comment: There is a current macro that says if you have __STDC_NO_THREADS__ defined, then you don't need to provide the stdcatomic.h header. These are different things. Specifically, threads belong to the OS and atomics belong to the hardware. In embedded systems, you want hardware support and not have the OS come along for the ride.

Proposed Response: Separate __STDC_NO_THREADS__ from stdcatomic.h

Excerpt from paper, item #4:

There is a current macro that says if you define __STDC_NO_THREADS, then you don't need to provide the stdatomic.h header.

The integer constant **1**, intended to indicate that the implementation does not support atomic types (including the **_Atomic** type qualifier and the **<stdatomic.h>** header) or the **<threads.h>** header.

Threads and atomics are different things. Specifically, threads belong to the OS and atomics belong to the hardware. The OS should provide common locking implementations so that different compilers don't do incompatible things. In embedded system, you want hardware support and not have OS come along for the ride. We should separate __STDC_NO_THREADS from stdatomic.h. We suggest __STDCATOMIC__ be defined if the system provides atomics and the atomic header.

We would have atomics in all cases, with or without threads. Add a macro to make both atomics and threads optional? STDC_NO_THREADS and STDC_NO_ATOMIC ? Whose OK with this? CA, Michael, Lawrence, Hans, all OK.

STRAW POLL: Split out atomics from __STDC_NO_THREADS__, and add __STDC_NO_ATOMICS__.

Result: 15 – 2 – 2

DECISION: CA 4: Split out atomics from __STDC_NO_THREADS__, and add __STDC_NO_ATOMICS__.
Side Note: Crowl will recommend that threads be taken out of C++ freestanding in Madrid.

CA 5, Sec 5.1.2.4
Comment: Remove atomic to atomic assignment. C++0x has removed it because people may think the assignment is like transactional memory, but it is not.

Proposed response: none.
We don't have proposed words for this comment, and the Section is quite large.

ACTION: Michael Wong to propose wording for this comment
DONE: There is no problem here. Withdrawn.

CA 6
Comment: Align C mutex types with C++ mutex types.
Rationale: C++ mutex types were designed to make that compatibility possible. It will be embarrassing if we don't have the same mutex type. Originally, they were not placed probably because people did not want to assume a C syntax. Now that there is, this makes this argument moot. C mutex are local objects and while we may put wrapper around that because we require member functions, this will make condition variables fail to work with that. Condition variables only work with the C++ mutex type. If we further export these as inline functions, it also breaks down. We believe the C++ design leads to better performance, especially when we start scaling the system. [Hans and Lawrence may have some personal anecdotes and experience to back this up]. What is supplied by OS facility usually is too slow because it tries to be fair and does not scale well.
PJ: There was never an intention for C/C++ threads to be interoperable, or interchangeable. We have no words for this comment, nor do we have any evidence of implementation experience.

Why not POSIX? Because the threads library specified in the WP is more portable, and on more different types of system than POSIX.

US 24 has an alternate proposal, i.e. remove the mtx_try

STRAW POLL: Adopt CA 6
Result: 6-10-5 No Consensus for making this change.
DECISION: No Consensus to Adopt CA 6. We have already considered this issue in prior meetings. There was no consensus then to adopt that approach, and there is no consensus now.

NL 1, Sec 6.7.5
Comment: Comment on Section 6.7.5 - Alignment specifier
It would be 'natural', certainly for a language like C, if the alignment specification is part of the type specification and not, as proposed, as part of the declaration specifier. The proposed _Alignas specifier prevents the proper propagation and use of alignment information in the compiler. The argument for the current choice is that the cost of taking the type specifier approach would be very costly for C++; we do not consider this to be a valid argument: in many other places a difference between C and C++ is justified by the reasoning that C and C++ are two different languages, each with their own users and application areas, so why is it so necessary that in the _Alignas case the languages are the same

Proposed Response: NONE

Several spoke to the difficulty of making the alignment specification part of the type specification, however IBM is already doing this. Could do it for objects now, and later for types. Some implementation experience here.

Response: REJECTED – No Consensus to implement at this time.

NL 2, Annex F
Comment: This (normative) section refers to IEC 60669:1989, while there is a new version of this standard by the summer of 2011 (well in advance of adoption of the C1X standard). C1X must refer to the new floating-point standard. Separate question: is it the intention to include the exchange formats (especially binary16 - half float) as a fully required data type once the new floating-point standard is referenced? If not, should this be added to the Embedded-C specification as there is some interest in this in the embedded C world?

Proposed Response: None

We intentionally made this choice, but we also have a study group underway to create a binding to the new IEEE standard as a Technical Specification in the future.

RESPONSE: REJECTED for this revision. However there is a Study Group within WG14 looking at this issue with plans to create a C binding to the new IEEE Standard as a Technical Specification.

BSI 1, Sec 5.1.2.3;p5
Comment: 5.1.2.3#5 describes parts of program state when a signal occurs. However, it is defined in terms of objects, which does not cover the floating-point environment.
Depending on the operating system, the floating-point environment on receipt of a signal may be set to a default environment or it may be the environment in effect when the signal was delivered; the latter may not be a state that was ever in effect in the abstract machine because

code sequences for some operations may change the rounding mode temporarily, then restore it. It seems best to leave the choice explicitly unspecified. (This means signal handlers cannot reliably use floating point; if that is to be permitted, feholdexcept and fesetenv would need to be documented as safe to call from signal handlers.)

Proposed Response: In 5.1.2.3, insert ", as is the floating-point environment (7.6)" after "unspecified", and insert ", as does the floating-point environment if it is modified and not restored before exit from the handler" after "undefined".

Discussion: Rajan is OK with this change.

DECISION: Adopt BSI 1 (UNANIMOUS CONSENT)

BSI 2
Comment: There are some places where alignof needs to be handled similarly to sizeof, for consistency and to reflect existing practice, but with appropriate adjustments for when VLA size expressions are involved
Proposed Response:
In 6.5.3.4#3, change the second sentence to "Expressions in the operand are not evaluated, and the result is an integer constant.".

In 6.6#3, footnote 115, change "sizeof" to "sizeof or alignof".

In 6.6#6, insert "alignof expressions," before "sizeof expressions", and change "sizeof operator" to "sizeof or alignof operator".

In 6.6#8, change "and sizeof expressions" to "alignof expressions, and sizeof expressions", and change "a sizeof operator" to "an alignof operator or a sizeof operator".

In 6.9#3, change "a sizeof operator" to "an alignof operator or a sizeof operator".

In 6.9#5, change "a sizeof operator" to "an alignof operator or a sizeof operator".

DECISION: ACCEPT BSI 2 (UNANIMOUS CONSENT)

BSI 3, 6.7;p3 (N1565) (Stoughton)
Comment: 6.7#3 says "a typedef name can be redefined to denote the same type as it currently does", and redefining otherwise is a constraint violation, but in the case of VLAs it may not be known until runtime whether the types will in fact be the same. The suggested solution of diagnosing that a violation at runtime is possible should be stated in a footnote.

Proposed Response: In 6.7#3, after "same type as it currently does" add a footnote "If identity of the types depends on the values of variable length array size expressions, the implementation may generate a diagnostic that a constraint violation could occur depending on the values at runtime.".

Discussion:
Tom has concerns about this. It seems overly fussy to flag what we consider to be benign retypedefing rules. Rajan, has already implemented this, suggests making it Implementation Defined.

ACTION: Bill Seymour: Produce alternate words for a resolution to BSI 3 - DONE

Proposed Words:

Change:

If an identifier has no linkage, there shall be no more than one declaration of the identifier (in a declarator or type specifier) with the same scope and in the same name space, except that a typedef name can be redefined to denote the same type as it currently does and tags may be redeclared as specified in 6.7.2.3.

To:

If an identifier has no linkage, there shall be no more than one declaration of the identifier (in a declarator or type specifier) with the same scope and in the same name space, except:

* a typedef name can be redefined to denote the same type as it currently does if that type is not a variably modified type

* it is implementation defined whether a typedef name can be redefined to denote the same type as it currently does when that type is a variably modified type

* tags may be redeclared as specified in 6.7.2.3.

Update to Proposed Change in BSI 3

6.7;p3, Changes

Change:

If an identifier has no linkage, there shall be no more than one declaration of the identifier (in a declarator or type specifier) with the same scope and in the same name space, except that a typedef name can be redefined to denote the same type as it currently does and tags may be redeclared as specified in 6.7.2.3.

To:

If an identifier has no linkage, there shall be no more than one declaration of the identifier (in a declarator or type specifier) with the same scope and in the same name space, except:

* a typedef name can be redefined to denote the same type as it currently does if that type is not a variably modified type

* it is implementation defined whether a typedef name can be redefined to denote the same type as it currently does when that type is a variably modified type

* tags may be redeclared as specified in 6.7.2.3.

Discussion: Delete bullet 2 above, modify it? Leave it? Unspecified? Can't tell if it's the same type.

STRAW POLL: Words above for BSI 3, without Bullet 2 (delete bullet 2).

9-3-4 Consensus

DECISION: Adopt above words for BSI 3, w/o bullet 2.

BSI 4
Comment:
6.7#5 defines a "definition" of an identifier, saying that for an enumeration constant or typedef name it is "the (only) declaration of the identifier". The "(only)" is no longer accurate now typedef redefinition is allowed; it seems natural to say that the first declaration in such a case is the definition (an alternative would be to say that all are definitions).

Proposed Response:
In 6.7#5, replace the last bullet point with two bullet points:
• for an enumeration constant, is the (only) declaration of the identifier;
• for a typedef name, is the first or only declaration of the identifier.

DECISION: ACCEPT BSI 4 (UNANIMOUS CONSENT)

BSI 5
Comment:
6.7.1 is missing a constraint that _Thread_local may not be used on a function declaration. (This usage makes no sense and disallowing it is existing GNU __thread practice. For function definitions this is already disallowed by 6.9.1#4 but it should also be disallowed for declarations that are not definitions.)

Proposed Response:
In the Constraints in 6.7.1, add a new paragraph after paragraph 3: "_Thread_local may not be present in the storage class specifiers in a declaration of a function.".

DECISION: ACCEPT BSI 5 (UNANIMOUS CONSENT)

BSI 6
Comment:
6.7.2.1#18 says "As a special case, the last element of a structure with more than one named member may have an incomplete array type; this is called a flexible array member.". It should be made clear that this allows structures where all previous members are anonymous structures or unions, by virtue of 6.7.2.1#13.

Proposed Response:
At the end of 6.7.2.1, add a new example:
Because elements of anonymous structures and unions are considered to be members of the containing structure or union, the following example has more than one named member and is a valid use of a flexible array member:

```
struct s
 {
   struct
   {
    int i;
   };
  int a[];
};
```

DECISION: ACCEPT BSI 6 (UNANIMOUS CONSENT)

BSI 7
Comment:
6.7.9#15 says "An array with element type compatible with a qualified or unqualified version of wchar_t may be initialized by a wide string literal, optionally enclosed in braces. Successive wide characters of the wide string literal (including the terminating null wide character if there is room or if the array is of unknown size) initialize the elements of the array.". But 6.4.5 now defines wide string literals to include char16_t and char32_t literals, and the initialization wording needs updating to allow each kind of wide string literal to initialize the associated kind of array.

Proposed Response:
Change 6.7.9#15 to read "An array with element type compatible with a qualified or unqualified version of wchar_t, char16_t or char32_t may be initialized by a wide string literal, optionally enclosed in braces. The wide string literal must have array element type (as defined in 6.4.5) compatible with the unqualified version of the element type of the array being initialized. Successive elements of the array specified in 6.4.5 for the wide string literal (including the terminating null element if there is room or if the array is of unknown size) initialize the elements of the array."

DECISION: ACCEPT BSI 7 (UNANIMOUS CONSENT)

BSI 8
Comment:
6.10.9#1 refers to removal of the L prefix, if present, from a string literal inside _Pragma. This should now handle the new types of string prefixes added in C1X.

Proposed Response:
In 6.10.9#1, change "L prefix" to "u8, u, U or L prefix

STRAW POLL: Adopt proposed response to BSI 8
Result 14 – 2 - 5

DECISION: ACCEPT BSI 8

NEEDS C++ Liaison – same issue.

BSI 9, 7.1.2;p4
Comment:
7.1.2#4 says "The program shall not have any macros with names lexically identical to keywords currently defined prior to the inclusion.".
There is however a related issue that this does not address: a macro lexically identical to a keyword could be defined after the standard header is included, but with the definition being in effect when a macro defined in the standard header is expanded, and the expansion of the macro in the standard header could use the keyword that is defined as a macro.
Thus, either such definitions of keywords as macros should be disallowed whenever a macro from a standard header is expanded, or all macro definitions in standard headers need to use alternative implementation-specific keywords in the reserved namespaces such as __void. In the latter case, examples in the C standard such as the required definition of assert in 7.2#1, the possible definition of the cbrt type-generic macro in 6.5.1.1#5 and the possible definitions of CMPLX, CMPLXF and CMPLXL in 7.3.9.3#5 should not show the use of keywords outside the reserved namespaces.

(The Rationale (pages 100 and 101 in version 5.10) discusses uses for defining keyword names as macros, but I think this should still be made undefined behavior if such a macro definition is in effect when a macro from a standard header is expanded

Proposed Response:
In 7.1.2#4, add "or when a macro defined in a standard header is expanded" at end of last sentence.

DISCUSSION: PJ supports this, although many will ignore it.

DECISION: ACCEPT BSI 9 (UNANIMOUS CONSENT)

BSI 10 - Editorial. Accepted

BSI 11

Comment:
It seems clear from the standard text that the scanf %% format is required to skip white-space in the input stream: that %% acts differently from single ordinary characters in the format string and you need to use %1[%] to match just a single % without white-space. However, implementations differ in this regard, so it would be useful to add an example to make this clearer to implementers.

Proposed Response:
In 7.21.6.2, add another example: "The program
#include <stdio.h>

```
int main (void)
{
  int dummy;
  return sscanf ("foo \t %bar1", "foo%%bar%d", &dummy);
}
```

returns status 1, not 0, because input white-space is skipped when matching %%."

DECISION: ACCEPT BSI 11 (UNANIMOUS CONSENT)

BSI 12
Comment:
7.25.1#4 says that xtime "holds a time specified in seconds and nanoseconds", and has members "time_t sec;" and "long nsec;". But time_t is not required to count in seconds; 7.26.1#4 says "The range and precision of times representable in clock_t and time_t are implementation-defined.". time_t may count in units other than seconds; it may be a floating-point type, so if it counts in seconds it may have sub second resolution; it may not bear a linear relation to elapsed time. The xtime type is used by cnd_timedwait, mtx_timedlock and thrd_sleep, and set by xtime_get. xtime_get creates a valid xtime value, which apparently is to be interpreted in accordance with the base argument; the other functions use such a value, and do not have any base parameter to describe the interpretation. The description of the base argument refers to TIME_UTC, but the list of macros defined in this header does not include TIME_UTC.

I don't believe it makes sense to have the base argument to xtime_get, given that the semantics of time_t values (which must be the basis for those of xtime values) do not depend on any such value, and the only way to modify a time_t value to get a valid future time, and so a valid future xtime value, is through <time.h> functions (direct arithmetic on time_t values does not have defined effects). Thus I propose removing this argument.

Proposed Response:
Proposed change 1: In 7.25.1#4, change "holds a time specified in seconds and nanoseconds" to "holds a time specified as a nanosecond offset from a time_t value", with a footnote "Although the time_t value is given as time_t sec;, time_t does not necessarily count in seconds.".
Proposed change 2: In 7.25.7.1#1, remove the "int base" argument. In 7.25.7.1#2, remove "based on the time base". In 7.25.7.1#3, change "the nonzero value base, which must be TIME_UTC" to "a nonzero value". In Annex B.24, remove the "int base" argument to xtime_get.

DISCUSSION: Alternate wording possible? Add text to this.
ACTION: Joseph and Nick to add text to Proposed Response to merge the BSI and AG time related comments.
DONE: (N1564) See adopted words for BSI 12, AG 2, AG 9, AG 10, AG 11 below.

Proposed Response:
Remove xtime from threads.h
Change para 3 of <time.h> from
"... which are arithmetic types capable of representing times; and struct tm
which holds the components of a calendar time, called the broken-down time."

To:
" which are arithmetic types capable of representing times;
struct timespec
which is a structure type that holds a time specified in seconds and nanoseconds. The structure shall contain at least the following members, in any order.
time_t tv_sec; long tv_nsec;
and
struct tm
which holds the components of a calendar time, called the broken-down time."

Globally replace "xtime" with "timespec", the "sec" member with "tv_sec, and the "nsec" member with "tv_nsec". [Note the remainder of this ballot uses xtime where appropriate. The global edit suggested here should be applied to these ballot comments if this comment is accepted]

DECISION: Words adopted as above. (UNANIMOUS CONSENT)

BSI 13
Comment:
It appears 7.26.1#3 allows time_t and clock_t to be complex types. I see no good reason for this to be permitted.

Proposed Response:
In 7.26.1#3, change "arithmetic types" to "real types".

DECISION: ACCEPT (UNANIMOUS CONSENT)

BSI 14
Comment:
When are wide string library functions required to handle values of type wchar_t that do not represent any value in the execution character set, and when does using such values with a library function result in undefined behavior? This issue was raised directly and through the Austin Group; the Batavia minutes say "We are taking no action here" (N1541 6.31 item 1) but this still leaves the standard unclear.

The definition of "wide character" in 3.7.3 is "bit representation that fits in an object of type wchar_t, capable of representing any character in the current locale". I interpret the part after the comma as being descriptive of the type wchar_t, rather than constraining the definition of "wide character". That is, "wide character" includes all bit representations that fit in type wchar_t, whether or not they represent valid members of the execution character set.
The first problem here would seem to be the possible inclusion of trap representations; it seems better for only representations that represent values of type wchar_t to count as wide characters, and for only the integer value to be significant. That is, a wide character should be a value of type wchar_t, not a bit representation.

In turn, 7.1.1#4 defines a "wide string" to include all null-terminated sequences of wide characters (whether or not they represent execution character set members).
This implies semantics for certain cases. For example, 7.21.6#8 defines fprintf handling of the %ls format in terms of wcrtomb, and 7.28.6.3.3 specifies what happens in wcrtomb if "wc is not a valid wide character". The concept of "valid wide character" does not appear to be defined, but the intention must be a wide character not representing any value in the execution character set, since this is how 7.21.3#14 defines "encoding error".

In other cases, the document is silent about the handling of wchar_t values not representing any value in the execution character set.

In some cases, the semantics of wide characters are interpreted, but wcrtomb is not mentioned and so it is unclear whether encoding errors may occur. For example, the wide characters in an swprintf format string must be interpreted to identify conversion specifications. If a wide character is passed that does not represent a value in the execution character set, is this undefined behavior (possibly critical undefined behavior as an invalid argument to a library function), an encoding error, or a valid wchar_t value that must be passed through to the output? (For fwprintf, the values output to the stream are passed through wcrtomb, so the last case is not a possibility there.)

In other cases, such as wmemcmp, there is no interpretation of values. Should the statement from C90 AMD1 7.16.4.6

These functions operate on arrays of type wchar_t whose size is specified by a separate count argument. These functions are not affected by locale and all wchar_t values are treated identically. The null wide character and wchar_t values not corresponding to valid multibyte characters are not treated specially.
apply or was it deliberately removed?
My inclination is that whenever there is no need to interpret the string, and corresponding <string.h> functions accept all byte sequences (null-terminated as needed) following 7.23.1#3, all wchar_t sequences should be accepted by the wide character functions. One possibly tricky case is wcscoll/wcsxfrm, where the <string.h> functions accept all byte sequences (according to 7.23.1#3) even if they are not valid multibyte strings for the present locale, and no error returns are possible.
I propose a simple change to make it explicit that all wchar_t sequences are valid, but to allow

encoding errors where the strings may be interpreted and an error return is possible, as in the case of swprintf.

Proposed Response:
Proposed change 1: In 3.7.3, change "bit representation that fits in" to "value representable by".
Proposed change 2: In 7.28.1, add a new paragraph before paragraph 5: "Arguments to the functions in this subclause may point to arrays containing wchar_t values that do not correspond to members of the extended character set. Such values shall be processed according to the specified semantics, provided that it is unspecified whether an encoding error occurs if such a value occurs in the format string for a function in 7.28.2 or 7.28.5 and the specified semantics do not include passing the wide character through wcrtomb."

DECISION: ACCEPT BSI 14 (UNANIMOUS CONSENT)

BSI 15
Comment:
There are what appear to be name spaces (explicitly reserved or otherwise) used by various headers that are not listed in 7.30 but should be.

Proposed Response:

Proposed change 1: Between 7.30.3 and 7.30.4, add a subclause for <fenv.h>: "Macros that begin with FE_ and an uppercase letter may be added to the definitions in the <fenv.h> header.". Add footnotes referencing this new subclause to the sentences referring to such macros in 7.6#6, 7.6#8 and 7.6#10.

Proposed change 2: Between 7.30.6 and 7.30.7, add a subclause for <stdatomic.h>: "Macros, function names, typedef names and enumeration values that begin with ATOMIC_, atomic_ or memory_ may be added to the <stdatomic.h> header.".

Proposed change 3: Between 7.30.11 and 7.30.12, add a subclause for <threads.h>: "Function names, typedef names and enumeration values that begin with cnd_, mtx_, thrd_ or tss_ may be added to the <threads.h> header.".

DECISION: ACCEPT BSI 15 (UNANIMOUS CONSENT)

BSI 16
Comment:
J.2 lists (bottom of page 563) "The number of characters transmitted by a formatted output function is greater than INT_MAX (7.21.6.1, 7.21.6.3, 7.21.6.8, 7.21.6.10).". This is missing the wide character functions and the functions that output to strings instead of files; all of these have the same issue that there may be return values specified by the semantics that cannot be represented in the int return type.
(A similar issue applies to functions in Annex K. I have not tried to propose a fix there, though making the overflow cases into runtime-constraint violations may make sense. The asprintf-family functions in TR 24731-2 also have this problem.)

Proposed Response:
In the last item on page 563, change "characters" to "characters or wide characters" and change "transmitted" to "transmitted, written to a string, or that would be written to a string has the array size parameter been large enough". Add 7.21.6.5, 7.21.6.6, 7.21.6.12, 7.21.6.13, 7.28.2.1, 7.28.2.3, 7.28.2.5, 7.28.2.7, 7.28.2.9, 7.28.2.11 to the list of subclauses in that item.

DECISION: ACCEPT BSI 16 (UNANIMOUS CONSENT)

BSI 17
Comment:
scanf-family functions may have a format string with more than INT_MAX conversion specifiers. J.2 should list undefined behavior if one of these functions would need to return a value greater than INT_MAX.
(A similar issue applies to functions in Annex K. I have not tried to propose a fix there, though making the overflow cases into runtime-constraint violations may make sense.)

Proposed Response:
Add to J.2 an item "The number of input items assigned by a formatted input function is greater than INT_MAX (7.21.6.2, 7.21.6.4, 7.21.6.7, 7.21.6.9, 7.21.6.11, 7.21.6.14, 7.28.2.2, 7.28.2.4, 7.28.2.6, 7.28.2.8, 7.28.2.10, and 7.28.2.12).".

DECISION: ACCEPT BSI 17 (UNANIMOUS CONSENT)

BSI 18
Comment:
In view of the binary16 format in IEEE 754-2008, J.5.6 should explicitly note the possibility of additional floating types having less range and precision than float.

Proposed Response:
In J.5.6#1, add "or less range and precision than float" after "long double".

DECISION: ACCEPT BSI 18 (UNANIMOUS CONSENT)

BSI 19
Comment:
There are several instances of undefined behavior that are intrinsically hard for implementations to bound, by reason of the ABIs in use in practice or the limitations of hardware. These should be added to the list of critical undefined behavior in L.3#2.

Specifically:
• Modifying constant objects should be considered equivalent to operations using invalid pointers.
• The problems with invalid arguments to library functions also apply to symbols such as va_arg specified to be macros.
• Incompatible types, where not constraint violations, generally cannot be diagnosed without information often not available at link time, and if (say) one translation unit declares an object with a type occupying more memory than another translation unit defining the object, accesses from the first translation unit will be like using invalid pointers.

Proposed Response:
Proposed change 1: In the list in L.3#2, add "The program attempts to modify a string literal (6.4.5).".

Proposed change 2: In the list in L.3#2, add "An attempt is made to modify an object defined with a const-qualified type through use of an lvalue with non-const-qualified type (6.7.3).".

Proposed change 3: In the list in L.3#2, change "library function" to "library function or macro". Wording change to clarify the library or macro is a Standard library, or Standard macro. David: It's not really UB to pass an invalid argument to a macro.

Proposed change 4: "Two declarations of the same object or function specify types that are not compatible (6.2.7).".
Add: And object is modified, or the function is called
Open issues for all 4.

Defer

New Proposed Response: (N1568)

Proposed change 1:
Accept as written.

Proposed change 2:
Accept as written.

Proposed change 3:
In L.3#2, change "An argument to a library function . . ." to
"An argument to a function or macro defined in the standard library . . ."

Proposed change 4:

Part A:
Change "(6.3.2.3)" to "(6.3.2.3, and see 6.2.7)".

Part B:
"A store is performed to an object that has two incompatible declarations (6.2.7)."

DECISION: Adopt BSI 19, N1568, as above. (UNANIMOUS CONSENT)

AG 1 (AUSTIN GROUP)
Comment:
Since quick_exit() disallows signal handlers to be called, what happens if a signal corresponding to a computational exception is generated during execution of one of the functions registered by at_quick_exit()?

Proposed Words:
In 7.22.4.7, add at the end:
If a signal is raised while the quick_exit function is executing, the behavior is undefined.

Discussion: Why is this the case? Shouldn't the interrupt be handled gracefully? quick_exit does not cleanup as well as exit. The assumption is that a quick exit means 'get me out of here'.

DECISION: Adopt AG 1 Proposed Words as above. (UNANIMOUS CONSENT)

AG 2 (Also BSI 12, AG 9, AG 10, AG 11) (N1564) (Stoughton)

Comment:
POSIX already defines two different structures to hold time, one of which, struct timespec, is almost identical to the xtime structure. It would be appropriate to merge the xtime and timespec structures.

Proposed Response:
Remove xtime from threads.h
Change para 3 of <time.h> from
"... which are arithmetic types capable of representing times; and struct tm
which holds the components of a calendar time, called the broken-down time."

To:
" which are arithmetic types capable of representing times;
struct timespec
which is a structure type that holds a time specified in seconds and nanoseconds. The structure shall contain at least the following members, in any order.
time_t tv_sec; long tv_nsec;
and
struct tm
which holds the components of a calendar time, called the broken-down time."

Globally replace "xtime" with "timespec", the "sec" member with "tv_sec, and the "nsec" member with "tv_nsec". [Note the remainder of this ballot uses xtime where appropriate. The global edit suggested here should be applied to these ballot comments if this comment is accepted]

The above needs more work.
New proposed words (N1564):

Change para 3 of 7.26 <time.h> from
"... which are arithmetic types capable of representing times; and
struct tm
which holds the components of a calendar time, called the broken-down time."

To:
"... which are arithmetic types capable of representing times;
struct timespec
which is a structure type that holds a time specified in whole seconds and nanoseconds, and
struct tm
which holds the components of a calendar time, called the broken-down time.

Many of the timing facility functions accept or return time value specifications.
A time value structure timespec specifies a single time value, and contains at least the following members, in any order.

time_t tv_sec;
long tv_nsec;

The tv_sec member is a whole number of seconds since an implementation defined epoch. (footnote: the tv_sec field is a linear count of seconds and may not have the normal sematics of a time_t).

The tv_nsec member is only valid if greater than or equal to zero, and less than the number of nanoseconds in a second (1 000 million).

The time interval described by this structure is (tv_sec * 1E9 + tv_nsec) nanoseconds."

Globally replace "xtime" with "timespec", the "sec" member with "tv_sec, and the "nsec" member with "tv_nsec".

Change 7.25 <threads.h> para 4:

FROM:
and

xtime

which is a structure type that holds a time specified in seconds and nanoseconds.
The structure shall contain at least the following members, in any order.

time_t sec;
long nsec;

TO:

Inclusion of the <threads.h> header shall also make visible all of the symbols defined in <time.h>.

Move 7.25.7 to 7.26.4, changing the title from "Time Functions" to "Other Time Functions"

Add to 7.26.1 para 2:

TIME_UTC

which expands to a integer constant greater than 0 that describes one of perhaps several system time bases (footnote: implementations may define several other clocks, but are only required to support a real time clock based on Universal Coordinated Time)

Rewrite the xtime_get function as follows:

7.26.4 Other Time functions

7.26.4.1 The timespec_get function

Synopsis

#include <time.h>
int timespec_get(struct timespec *xt, int base);

Description

The timespec_get function sets the timespec object pointed to by xt to hold the current time based on the time base.

If base is TIME_UTC, the tv_sec field is set to the number of seconds truncated to a whole value since an implementation defined *epoch*.

The tv_nsec field shall be set to the integral number of nanoseconds, rounded to the resolution of the system clock. (footnote)

(footnote: Although a timespec object describes times with nanosecond resolution, the actual resolution in a timespec object is system dependent, and may be greater than 1 second).

Returns

If the timespec_get function is successful it returns the nonzero value base, otherwise, it returns zero.

Add to Future Library Directions a new clause between 7.30.11 and 7.30.12:

7.30.12 Date and Time <time.h>
Macro names beginning with TIME_ may be added to the macros defined in the <time.h> header.

DECISION: Adopt proposed changes as above. (UNANIMOUS CONSENT)

AG 3, 7.25.3.5; p1,2 (N1564) (Stoughton)
Comment:
"until after the time specified by the xtime object pointed to by xt"

It is not clear whether xt specifies an absolute time or elapsed
time from the start of the cnd_timedwait() call.

I.e. should applications just set xt to the length of the timeout,
or should they call xtime_get(), add the length to the returned
time, and then use that.

The equivalent POSIX function pthread_cond_timedwait() takes an
absolute time. There are good reasons for this: see the RATIONALE in
the POSIX description of the function.

Proposed Response:
Clarify whether xt is an absolute time or the length of the timeout.

New proposed words:

In 7.25.3.5, change:
Synopsis

```
#include <threads.h>
int cnd_timedwait(cnd_t *cond, mtx_t *mtx, const xtime *xt);
```

To:
Synopsis

```
#include <threads.h>
int cnd_timedwait(cnd_t *restrict cond, mtx_t *restrict mtx,
        const struct timespec *restrict abstime);
```

At para 2, change

"..until after the time specified by the xtime object pointed to by xt.."

to

"..until after the time specified by the timespec object pointed to by abstime, which represents an absolute time in time base TIME_UTC since the implementation defined

epoch."

DECISION: AG 3, Adopt proposed words as above. (N1564) (UNANAMOUS CONSENT)

AG 4
Comment: 7.25.3.6 says "If the mutex pointed to by mtx is not locked by the calling thread, the cnd_wait function will act as if the abort function is called."

This requirement means mutexes must keep a record of ownership, which affects efficiency, and is inconsistent with cnd_timedwait() whose description states "The cnd_timedwait function requires that the mutex pointed to by mtx be locked by the calling thread."

Proposed response:
Change 7.25.3.6
From:
"If the mutex pointed to by mtx is not locked by the calling thread, the cnd_wait function will act as if the abort function is called."

To:
"The cnd_wait function requires that the mutex pointed to by mtx be locked by the calling thread."

DECISION: ACCEPT AG 4 proposed response as above. (UNANIMOUS CONSENT)

AG 5 (N1564) (Stoughton)
Proposed words:
In 7.25.4.4, change

From:
Synopsis
```
#include <threads.h>
        int mtx_timedlock(mtx_t *mtx, const xtime *xt);
```

To:
Synopsis
```
#include <threads.h>
        int mtx_timedlock(mtx_t *restrict mtx, const struct timespec *restrict abstime);
```

in para 2, change

From:
until the time specified by the xtime object xt has passed.

To:
until after the time specified by the timespec object pointed to by abstime, which represents an absolute time in time base TIME_UTC since the implementation defined epoch.

DECISION: Adopt proposed words for AG 5. (UNANIMOUS CONSENT)

AG 6 (N1564) (Stoughton)
Add a new para (para 3) to 7.25.5.5 Description:

The program shall exit with an exit status of 0 after the last thread has been terminated. The behavior shall be as if the implementation called exit() with a zero argument at thread termination time.

DECISION: Adopt proposed words for AG 6. (UNANIMOUS CONSENT)

AG 7, AG 8 (N1564) (Stoughton)

Change 7.25.5.7 The thrd_sleep function

From:
Synopsis
#include <threads.h>
void thrd_sleep(const xtime *xt);
Description
The thrd_sleep function suspends execution of the calling thread until after the time specified by the xtime object pointed to by xt.
Returns
The thrd_sleep function returns no value.

To:
7.25.5.7 The thrd_sleep function
Synopsis
#include <threads.h>
int thrd_sleep(const struct timespec *duration, struct timespec *remaining);

Description
The thrd_sleep() function shall cause the current thread to be suspended from execution until either the time interval specified by the duration argument has elapsed or a signal is delivered to the calling thread, and its action is to invoke a signal-catching function or to terminate the program. The suspension time may be longer than requested because the argument value is rounded up to an integer multiple of the sleep resolution or because of the scheduling of other activity by the system. But, except for the case of being interrupted by a signal, the suspension time shall not be less than the time specified by duration, as measured by the system clock TIME_UTC.

Returns
If the thrd_sleep() function returns because the requested time has elapsed, its return value shall be zero. If the thrd_sleep() function returns because it has been interrupted by a signal, it shall return a value of -1. If the remaining argument is non-NULL, the timespec structure referenced by it is updated to contain the amount of time remaining in the interval (the requested time minus the time actually slept). The duration and remaining arguments may point to the same object. If the remaining argument is NULL, the remaining time is not returned. If thrd_sleep( ) fails, it shall return a negative value.

DECISION: Adopt above for AG 7, AG 8 (UNANIMOUS CONSENT)

6. OTHER BUSINESS

6.1 C1X Schedule

We expect to submit the WP for DIS Ballot after this meeting, following an editorial review. The DIS ballot is five months. Since there is not much point in meeting during the ballot period, it will be at least six months before we can meet again. If the DIS ballot is approved, with no comments, we can skip the follow-on ballot (FDIS), and go straight to publication.

6.2 Potential C/C++ Compatibility Issues

Below is a list of items flagged as C++ "issues" by WG21 that may or may not have an impact on C.

The LWG and CWG references below are from the Library Working Group (LWG) and Core Working Group (CWG) issues lists within WG21, respectively. There are no corresponding WG14 documents. The link to these items is:
CWG: http://www.open-std.org/jtc1/sc22/wg21/docs/cwg_active.html
LWG: http://www.open-std.org/jtc1/sc22/wg21/docs/lwg-active.html

Are we OK with the items listed below?

LWG 1460 – OK (adds BOOL and pointer)

LWG 1461 – what should the macro prefix be? If we chose to add a prefix, C++ will follow. LC wants no change.
STRAW POLL – Adopt a prefix on the upper case atomic macros as in
Result: 1 - 13 – 6 No support from WG14 to adopt prefixes.

LWG 1459 – No support for this issue from WG14. Expect WG21 to resolve this issue as 'not a defect'.

LWG 2034 – wordsmithing issue, but we want our memory model to remain consistent with C++. We need to see the 'right' words. Provided by Hans

Modify 29.3 [atomics.order] p.3, so that the normative part reads:

There shall be a single total order *S* on all memory_order_seq_cst operations, consistent with the "happens before" order and modification orders for all affected locations, such that each memory_order_seq_cst operation that loads a value observes either the last preceding modification according to this order *S*, *A* (if any), or the result of an operation *X* that does not happen before *A* and that is not memory_order_seq_cst. [*Note*: Although it is not explicitly required that *S* include locks, it can always be extended to an order that does include lock and unlock operations, since the ordering between those is already included in the "happens before" ordering. — *end note*]

LWG 1524 – if adopted, the C words would be different, but along the same lines of. Joseph: what we've agreed to in N1551 resolves this issue for C. Hans is not sure, would like to look at this offline. Upon further review, there are no C issues here.

LWG 2037 – mostly a C++ internal issue, but wanted to point out that C++ is extending all of the free functions to include all of the atomics. OK with us.

LWG 1478 – clarifies that initialization is not atomic. We are already covered.

LWG 1456 – some int type stuff did not make it into C++, and so they are being added. We discussed this in Boulder. We decided not to do these in C.

LWG 1457 – C already has these, so it's a non issue for us.

LWG 1474 – atomic_compare_exchange, wordsmithing by C++, but no semantic change?
ACTION: Lawrence to tweak our words as needed to align with LWG 1474.
Blaine, some wordsmithing of footnotes to clarify the intent is still underway.

Proposed Words in N1567. No objection to the footnote words.
STRAW POLL – Add proposed words to footnote 133 in N1567.
Result: 15 – 0 - 3

DECISION: Adopt as applicable to C WP the existing proposed resolution to LWG 1474

http://lwg.github.com/issues/lwg-active.html#2034

LWG 1479 – are the functions pointer compatible – yes

LWG 1480 – need to get the ordering constraints right. We currently have the same original words, and can adopt the words proposed by C++.

LWG 2024 – internal C++, atomic pointers need to have a standard layout to make them compatible with C. Looks OK to us.

CWG 1176 – could use an equivalent C change (5.1.2.4;p10). Blaine likes the wording.
DECISION: Adopt the proposed wording as applicable. (UNANIMOUS CONSENT)

Recommend we (C++) change 5.1.2.4p10 as follows:
A release sequence <ins>headed by a release operation A</ins> on an atomic object M is a maximal contiguous sub-sequence of side effects in the modification order of M, where the first operation is ~~a release~~<ins>A and every subsequent operation either is performed by the same thread that performed the release or is an atomic read-modify-write operation.

This uses "headed by a release operation A" instead of "from a release operation A" in the C++ FCD. The wording here matches the uses, and I currently prefer it. I sent an email message to WG21 CWG asking for clarification.

Pete (C++ PE) has seen the intent of these words. Douglas: Looks right.

CWG 1177 – Our wording looks close to the proposed wording here, so we can adopt these words.

6.3 Thread Unsafe Standard Functions (N1371) (Crowl, Plauger, Stoughton)

Revisiting this paper, behavior of atexit in a multi-thread or single-thread environment. PJ – there is no way to specify this, and get it right, for a multi-thread environment. Hans wording makes it specifically unspecified to call atexit after exit.

The wording here is C++ wording, and has to be reworded to fit into C. We need words that we can look at.

ACTION: Larry to write up proposed wording for changes in item 18.4, Start and Termination, for atexit and exit.
DONE

6.4 Latency Reducing Memory Allocation in the C Standard Library (N1527) (Douglas)

Niall Douglas presented his paper, N1527. This paper points out the difference in growth between improvements to the speed required to access RAM and the improvement in the amount of RAM available. The growth of RAM available is exponential (250x) while improvement in access to RAM is linear (25x). Clearly the improvement in RAM access speed is lagging well behind the growth in RAM capability. The paper proposes new mechanisms for the C Library to improve RAM access speed. The code is relatively small, implemented, and available.
This material would be a good candidate for a Technical Specification for future consideration.

6.5 Compound Assignment, 6.5.16.2, fn 113, (N1567) (Garst)

A compound assignment of the form `E1` *op*`=` `E2` is equivalent to the simple assignment expression `E1 = E1` *op* `(E2)`, except that the lvalue `E1` is evaluated only once, and with respect to an indeterminately-sequenced function call, the operation of a compound assignment is a single evaluation. If `E1` has an atomic type, compound assignment is a read-modify-write operation with `memory_order_seq_cst` memory order semantics.

Edit paragraph 3 footnote 113) as follows. This change clarifies that the right-hand argument is evaluated only once. It also clarifies the intent that any exceptions that occur during intermediate failing computations should not accumulate in order to satisfy the equivalence of E1 op= E2 and E1 = E1 op (E2).

113) Where a pointer to an atomic object can be formed, and where `E1` and `E2` have integer type, this is equivalent to the following code sequence where ~~*T*~~ *T1* is the type of `E1` and *T2* is the type of `E2`:

```
~~T~~ T1 tmp = E1;
T2 val = (E2);
~~T~~ T1 result;
do {
      result = tmp op ~~(E2)~~ val;
} while (!atomic_compare_exchange_strong(&E1, &tmp, result));
```

with `result` being the result of the operation.

If E1 or E2 are floating types, then exceptional conditions or floating-point exceptions encountered during discarded evaluations of result must terminate the loop immediately or be discarded in order to satisfy the equivalence of E1 op= (E2) and E1 = E1 op (E2). For example, if Annex F is in effect and FLT_EVAL_METHOD defined as 0, the equivalent recovering code could be

```
#include <fenv.h>
#pragma STDC FENV ACCESS ON
fenv_t fenv;
T1 existing = E1;
T2 val = E2;
T1 result;
feholdexcept(&fenv);
for (;;) {
```

```
        result = existing_op_val;
        if (atomic_compare_exchange_strong(&E1, &existing, result))
                break;
        feclearexcept(FE ALL EXCEPT);
}
feupdateenv(&fenv);
```

If FLT EVAL METHOD is not 0 then T2 must be a type with the range and precision to which E2 is evaluated in order to satisfy the equivalence of E1 op= (E2) and E1 = E1 op (E2)

The code above has both an equivalence relationship and an example. Editorial to change/fix. This markup is difficult to read. The first (existing) note was wrong. This is a correction. Prefer to see it as "Change to read:" or Change From, To.
ACTION: Blaine to revise this morning.

Revised Wording: (N1567) (Garst)

fn 113) Where a pointer to an atomic object can be formed, and where E1 and E2 have integer type, this is equivalent to the following code sequence where *T1* is the type of E1 and *T2* is the type of E2:

```
T1* addr = &E1;
T1 tmp = *addr;
T2 val = (E2);
T1 result;
do {
    result = tmp op val;
} while (!atomic_compare_exchange_strong(addr, &tmp, result));
```
with result being the result of the operation.

If E1 or E2 are floating types, then exceptional conditions or floating-point exceptions encountered during discarded evaluations of result should be discarded in order to satisfy the equivalence of E1 op= (E2) and E1 = E1 op (E2). For example, if Annex F is in effect, the floating types involved have IEEC 60559 formats, and FLT_EVAL_METHOD defined as 0, the equivalent code would be

```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
fenv_t fenv;
T1 *addr = &E1;
T1 existing = *addr; // evaluate E1 only once
T2 val = E2;
T1 result;
feholdexcept(&fenv);
for (;;) {
    result = existing op val;
    if (atomic_compare_exchange_strong(addr, &existing, result))
                break;
    feclearexcept(FE_ALL_EXCEPT);
}
feupdateenv(&fenv);
```

If FLT_EVAL_METHOD is not 0 then T2 must be a type with the range and precision to which E2 is evaluated in order to satisfy the equivalence of E1 op= (E2) and E1 = E1 op (E2).
---- end of footnote ----

The revision to the footnote is non-normative, and for information/clarification. No straw poll needed.

C++ Concurrency needs to know about this ??

Rajan: "..must terminate the loop..", How do you know it's in a loop?
Jim: For Annex F operations, all of the operations can be handled.

6.6 Behavior of realloc() with zero size (N1559) (Sebor)

Note: Although an issue is described, there are no proposed words.

The C99 standard specifies two distinct implementation behaviors for a call to the realloc(p, n) function when (p != NULL) and (n == 0) hold:

1) the call fails without attempting to allocate any storage and returns NULL

2) the function attempts to allocate storage and if the allocation is successful returns a pointer to the allocated space; otherwise, it returns NULL

Correct, but which of these exits is Implementation Defined.

These two behaviors seem to contradict each other. Issue came up in one of the secure coding meetings. POSIX defers to C. AG is seeking guidance from WG14 on this issue.

Larry: The intent never changed. Every time we try to specify what realloc does, we get it wrong.

Tom: There are good reasons to require the size allocator determine the size is greater than '0'.
Martin: Both positions exist. (accept 0, or greater than 0).

Rajan: IBM follows C99.
Douglas: Sun implemented POSIX. It's no big deal.
Others not sure what they've done.
Tom: The window has closed.

If realloc has failed, it returns NULL, and has not reallocated memory.

Correct the rationale. Add a footnote? Larry will look at it. Split out the failure case in the text.

No further action here.

6.7 Lifetime of locale string (N1561) (Sebor)

Words are proposed.

The C99 standard requires that the lifetime of the string referenced by the pointer returned from a successful call to setlocale() extend through the end of the program.

This requirement effectively prevents implementations from dynamically allocating the string (and freeing it the next time setlocale() is called).

The POSIX standard is considering an enhancement whereby implementations would be allowed to allocate the string dynamically and free it each time setlocale() is called. This is useful when the string is excessively long, typically when using "combined" locales or user defined locale databases.

This change is in TC 1 for IEEE 1003.

Larry: This is a reasonable thing to do.
Bill S: A good idea that is too late.
Nick: The IEEE TC 1 words are not the same as the words here.
Martin: Intentional, to better align with C.

No consensus for this change, at this time.

6.8 putc() and getc() macros (N1562) (Sebor)

The C99 standard specifies that the getc(), putc(), getwc(), and putwc() functions may be implemented as macros that evaluate their stream argument more than once, and discourages programs from invoking the functions (or macros) with an argument that has side effects.

To minimize the risk of security related defects due to this error the CERT C Secure Coding Standard requires that conforming programs avoid calling getc() or putc() with arguments that have side effects.

No consensus for this change, at this time.

6.9 N1563, DR# 017, Question 19 missing from C99 (Sebor, Jones)

The Response to DR #017, Question 19, is:

> If a fully expanded macro replacement list contains a function-like macro name as its last preprocessing token, it is unspecified whether this macro name may be subsequently replaced. If the behavior of the program depends upon this unspecified behavior, then the behavior is undefined.

The first sentence of the Response is reflected in J.1 of the standard below but there appears to be no corresponding wording in 6.10.3:

> When a fully expanded macro replacement list contains a function-like macro name as its last preprocessing token and the next preprocessing token from the source file is a (, and the fully expanded replacement of that macro ends with the name of the first macro and the next preprocessing token from the source file is again a (, whether that is considered a nested replacement (6.10.3).

In addition, the second sentence of the Response is not reflected in either Annex J.2 or anywhere in 6.10.3.

The absence of text in the body of the Standard is intentional. Take it out of the annex? Martin would prefer to see the example in the body.

There seems to be text missing from the document.
Tom: We are entitled to add an example because it's non-normative.

STRAW POLL: Add an example.
Result: 11-5-3 Consensus to add an example, but PE is instructed if issues arise, drop it.

7. RESOLUTIONS

7.1 Review of Decisions Reached

The following papers achieved consensus for adoption into the C1X WP. All are modulo edits as needed by the Project Editor.

N1553 – NB Comments - Responses of ACCEPT
US 1
US 5
US 8 - 11
US 17
US 18

CA 2

BSI 1
BSI 2
BSI 4
BSI 5 – 11
BSI 13 – 18

Responses of Accepted with Modification (AWM)
US 19 – 21
US 24
CA 4
BSI 3
BSI 12 (N1564)
BSI 19 (N1568)

N1554 – AG Comments - ACCEPT
AG 4

N1554 – AG Comments – Accept with Modification (AWM)
AG 1 - 3
AG 5 – 11

All other comments were considered, but lacked consensus by the Committee for adoption.

Non – NB or AG Comments decisions.

Review of all comments by NB.

US 1 ACCEPT

US 2	REJECT
US 3	REJECT
US 4	REJECT
US 5	ACCEPT
US 6	REJECT
US 7	REJECT
US 8	ACCEPT
US 9	ACCEPT
US 10	ACCEPT
US 11	ACCEPT
US 12	REJECT
US 13	REJECT
US 14	REJECT
US 15	REJECT
US 16	REJECT
US 17	ACCEPT
US 18	ACCEPT
US 19	AWM
US 20	AWM
US 21	AWM
US 22	AWM
US 23	AWM
US 24	AWM
US 25	REJECT

BSI 1	ACCEPT
BSI 2	ACCEPT
BSI 3	AWM
BSI 4	ACCEPT
BSI 5	ACCEPT
BSI 6	ACCEPT
BSI 7	ACCEPT
BSI 8	ACCEPT
BSI 9	ACCEPT
BSI 10	ACCEPT
BSI 11	ACCEPT
BSI 12	AWM
BSI 13	ACCEPT
BSI 14	ACCEPT
BSI 15	ACCEPT
BSI 16	ACCEPT
BSI 17	ACCEPT
BSI 18	ACCEPT
BSI 19	AWM

CA 1	REJECT
CA 2	ACCEPT
CA 3	REJECT
CA 4	AWM
CA 5	REJECT
CA 6	REJECT

NL 1	REJECT

NL 2 REJECT

RU 1 REJECT

AG 1 AWM
AG 2 AWM
AG 3 AWM
AG 4 ACCEPT
AG 5 AWM
AG 6 AWM
AG 7 AWM
AG 8 AWM
AG 9 AWM
AG 10 AWM
AG 11 AWM

7.2 Review of Action Items

Carry Over - NONE

New
ACTION – Editorial Review Group to meet on or about 4 April, 2011: JB, David Keaton, Blaine, Garst, Nick Stoughton, Douglas Walls, Jim Thomas, Larry Jones. Blaine can host. Comments can only be applied to Larry's edits; conforming to what we've voted on in this meeting. 1 PM at Apple. Expect to forward on or before 11 April. Allow no less than 6 months for another meeting.

ACTION – Convener forward the WP as revised in this meeting to the inquiry stage of ballot, DIS, 5 months, ITTF managed. If there are no "DISAPPROVE" votes, and no technical comments, WG14 should be able to publish. Even if an additional 2 month ballot is needed, the new C standard should be published in early in 2012.

8. THANKS TO HOST

The Committee expresses its great appreciation to BSI, Rod Grealish, for making arrangements for this meeting at BSI, Chiswick, London, UK.

The Committee expresses a special thanks to Lawrence Crowl (Google), Hans Boehm (HP), and Mike Wong (IBM) for their participation via teleconference, in spite of having to get up very early in the morning to do so.

Thanks to HP for providing the telephone bridge used for the teleconference.

Thanks also to Dinkumware for providing the Wiki.

9. ADJOURNMENT

Meeting adjourned at 12:20, Thursday, Mar 17, 2011.

PL22.11 Meeting
15 March 2011

Meeting convened at 16:18 local, 15 March 2011, by PL22.11 Chair, David Keaton.

Attendees:

Voting Members:		
Name:	**Organization:** **P – Primary, A - Alternate**	**Comments**
Blaine Garst	Apple - P	
John Benito	Blue Pilot - P	
David Keaton	CMU/SEI/CERT - P	PL22.11 Chair
P. J. Plauger	Dinkumware, Ltd – P	
Tana L. Plauger	Dinkumware, Ltd – A	
Jim Thomas	HP – P	
Rajan Bhatka	IBM - P	
Nat Hillary	LDRA - P	
Douglas Walls	Oracle - P	PL22.11 IR
Barry Hedquist	Perennial – P	PL22.11 Secretary
Tom Plum	Plum Hall – P	
Bill Seymour	Seymour - P	
Fred Tydeman	Tydeman Consulting – P	PL22.11 Vice Chair
Prospective Members		
Martin Sebor	Cisco	
Nick Stoughton	Self	

Agenda Approved as modified.
Meeting minutes from Batavia approved by unanimous consent. (PL22.11/10-004).

1. Selection and Review of US Delegation – done in Boulder

2. INCITS Anti-Trust Guidelines
We viewed the slides located on the INCITS web site.
http://www.incits.org/inatrust.htm

3. INCITS official designated member/alternate information.
Be sure to let INCITS know if your designated member or alternate changes, or if their email address changes. Send contact info to Lynn Barra at ITI, lbarra@itic.org.

4. Identification of PL22.11 Voting Members (Tydeman)
See attendance list above.
12 PL22.11 voting members participated out of 15.

4.1 PL22.11 Members Attaining Voting Rights at this Meeting
None

4.2 Prospective PL22.11 Attending Their First Meeting
Martin Sebor, Cisco

5. Members in Jeopardy

5.1 Members in jeopardy due to failure to return Letter Ballots.
HP

5.2 Members in jeopardy due to failure to attend Meetings.
LDRA

6. JTC1 SC22/WG14 Working Sessions
All members of PL22.11, who are not JTC1 Officers, are members of the US delegation to WG14.

7. **New Business**
None

8. Adjournment
Adjourned at 16:24 local, 15 March 2011