WG14 N1542
Reply To The Attention Of: Barry Hedquist
Email: beh@peren.com

ISO/IEC JTC1 SC22/WG14 AND INCITS PL22.11
MAY 25 - 27, 2010 MEETING MINUTES

Meeting Sponsor: USA (ANSI) / CERT

Meeting Location:
Courtyard by Marriott, Boulder/Louisville
948 West Dillon Road
Louisville, CO 80027 USA
Phone: +1.303.604.0007
Fax: +1.303.604.0047

Host Contact information:

David Keaton
CERT
1630 30th Street #311
Boulder, CO 80301
Tel: +1.303.800.2938
Fax: +1.720.306.3512
Email: dmk@cert.org

Meeting Dates: 25-27 May, 2010

Meeting Times:

25 May 2010 09:00–12:00 13:30–17:00
26 May 2010 09:00–12:00 13:30–17:00
27 May 2010 09:00–12:00 13:30–17:00

1. OPENING ACTIVITIES

1.1 Opening Comments (N1455) (Benito, Keaton)

David Keaton welcomed us to the Boulder/Louisville area, described the meeting and hotel facilities. Several local restaurants within walking distance of the hotel. Lunch break will be from 12:00 - 13:30. Breakfast served every morning starting at 6:00 AM. Morning and afternoon breaks will be at 10:30, and 14:30.

1.2 Introduction of Participants/Roll Call

Name	***Organization***	***NB***	***Comments***
John Benito	Blue Pilot		WG14 Convener
Blaine Garst	Apple	USA	

David Keaton	CERT	USA	PL22.11 Chair
Tana L. Plauger	Dinkumware, Ltd	USA	
P. J. Plauger	Dinkumware, Ltd	USA	
Jim Thomas	HP	USA	
Hans Boehm	HP	USA	
Paul McKenny	IBM	USA	
Michael Wong	IBM	USA	
Rajan Bhatka	IBM	USA	
Clark Nelson	Intel	USA	
Herb Sutter	Microsoft	USA	
Larry Wagoner	NSA	USA	
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	
David Svoboda	CERT	USA	
Larry Jones	Siemens		WG14 Project Editor

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.

We have limited teleconferencing for Atomics, 2 – 4 PM local time today, and tomorrow morning. Login information is on the WG14 Wiki.

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

http://www.incits.org/inatrust.htm
and
http://www.incits.org/pat_slides.pdf

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.

Barry Hedquist is Meeting Secretary.

1.4 Approval of Previous Minutes - Florence (N1449) (Hedquist)

Several comments for typos, etc.

Minutes were modified per editorial changes and approved.

Final Minutes are **N1493.**

1.5 Review of Action Items (Hedquist)

Note: Check these items against submitted documents for closure.

ACTION: Larry Jones: Review use of headers required for the function declaration shown in the synopsis, and any others required to make the declaration valid.
OPEN

ACTION: Convener to find out from SC22 what we can do / not do with the existing TR, then look at how to proceed from there w/r/t N1351 and possible changes to TR18037.
DONE – for a substantial technical change we can request to republish.

ACTION: David Svoboda to explore an approach to encode and decode pointers with Ulrich.
OPEN

ACTION: PJ to write a rationale for N1371, Thread Unsafe Standard Functions.
DONE - N1458

ACTION: PJ to write a proposal for incorporating errno, as applicable w/r/t N1371
DONE - N1458

ACTION: PJ to write a rationale for Threads, N1372.
DONE - N1458

ACTION: Tom Plum, along with David Svoboda, Nick, Clark, Blaine, to work up a new paper for new keywords based on N1403, Support for Attributes.
DONE – N1453

ACTION: Clark will ask Hans Boehm to add an explanation of memory sequencing to the rational re: N1411, Memory Model Rationale.
OPEN – no document yet

ACTION: Clark to work on a clearer formulation of what the rules are w/r/t N1409, Aliasing and Effective Type.
OPEN

ACTION: Convener to discuss the resolution of N1448 with the Submitter, Steve Adamczyk.
OPEN

ACTION: Blaine Garth to obtain a release of the slides presented, GCFlorence.pdf, from Apple.
DONE

1.6 Approval of Agenda (N1474) (Benito)

Revisions to Agenda:
Added Items:

5.26 – N1451, Blocks Proposal, Garth

5.27 – 12207 Static Assertion in for loops

15.28 – 12205 Anonymous Structures

15.29 – Interchangeable vs. aliasing

Deleted Items: None
Agenda approved as modified.

1.7 Distribution of New Documents (Benito)
None

1.8 Identification of National Body Delegations (Benito)
US

1.9 Identification of PL22.11 Voting Members (Tydeman)

See PL22.11 Minutes, following these minutes.

1.10 Next Meeting Schedule (N1436) (Benito)

The Fall 2010 meeting and lodging will be held Nov 1-5, 2010, in Geneva, Chicago, USA.

Comfort Inn & Suites Geneva
1555 East Fabyan Parkway
Geneva, ILL
USA
Tel: +1.630.280.8811
Fax: +1.630.208.7844
Email: sales@comfortinngeneva.com
Web Site: www.ComfortInnGeneva.com

Special Rates: $75/night - Standard Room; $81/night - Small Suite
The Special Rate is NOT available if reservations are made on the website.
Includes a Continental Breakfast, and Internet access.

WG21 and PL22.16 will be meeting the following week, Nov 8-13, at Fermi National Accelerator Laboratory in nearby Batavia, ILL. The laboratory is just a few miles from this hotel.

2. LIAISON ACTIVITIES

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

Santa Cruz: Major work item now is the C1X revision. N1390 is The Convenor's Report. MISRA C is now a Category C Liaison.

Where are we going for our schedule and feature set for C1X?
Blaine expected us to be able to mull over the content of his proposals for six months, and push a draft out in Batavia.
David expressed that CERT supports the atomics proposals, and that we are still tracking some things down. He's changed his mind on how long things need to bake, and would prefer to take a less ambitious schedule.
Tom wants to take a more ambitious track, pointing out that C historically has focused on that which exists, rather than that which does not. Further, there are projects waiting for C1X to get out before they can proceed.
PJ agreed with all that Tom said.
Blaine sees potential problems already in the area of the library version for atomics.
Douglas does not think we're done, and does not want to stick to an artificial deadline, even though it presents the opportunity to present new proposals.
David sees two doors. One says no more new proposals, the second says no more things put into the standard. The Study Group will not finish its work until April 2011, at the earliest, possibly Dec 2011. WG14 is not holding up the Security Study Group.
Paul wants to maintain an aggressive schedule.
JB explained the ballot process for the normal steps. By doing a combined 'CD Registration and Ballot' we skip a step of doing a Registration Ballot followed by a CD Ballot.

How do we want to proceed? Bill wants to vote out something this week. Tana sees a difference between what's in the draft and adding new features. Wants to keep to the schedule that we now have. JB: If we don't vote out a CD this week, we need to be very busy between now and Batavia. We have to really nail things down in Batavia.
David wants to do the work to get the things that need to be in the Standard ready for Batavia.
Tom: The way to stop the churn is to vote out a CD.
Blaine is very concerned about the state of atomics in C1X right now. Does not see that we are offering much guidance for the future. Thinks that C1X will miss a huge hole without the atomics language support. Sees an opportunity to make a difference in how C is seen by the world at large. The decision should be feature driven rather than schedule driven.
Tom would like to take a straw poll. How to frame the question? JB feels that a poll on whether or not we want a CD at this meeting answers a lot of questions.

Straw Poll: Do we want to produce a CD at this meeting ?
6, 5, 1 – no consensus to do so - yet

Are there those who voted NO doing so in order to consider a language solution for atomics? Unknown.

Tom willing to reopen atomics for another day as long as it is compatible with the C++ semantics.

2.2 WG21 / PL22.16 (N1432) (Plum, Benito, Sutter)

WG21 has published a Final Committee Draft, FCD 14882, and is now in FCD Ballot. The ballot closes 26 July, 2010.

WG14 published a Liaison Statement to WG21 (N1432), reaffirming its intention to remain on schedule for C1X (N1392).

2.3 PL22 Programming Languages (Plum)

PL22 now meets twice a year via teleconference. Jun 10, 2010, is next meeting via teleconference. Another meeting will be held in late July to establish US positions to SC22 Plenary.

2.4 Linux Foundation (Stoughton)

No Report

2.5 Austin Group (Stoughton)

Santa Cruz: TR balloted and approved with comments. Ballot resolution underway for the next 1-2 meetings. Several Issues raised via Austin Group Aardvark reports requesting WG14 action.
ACTION: Nick will put together a summary of Austin Group issues requesting WG14 input.
No immediate time constraint.

2.6 WG11

Boulder: No Report. Needs new Convener.

2.7 WG23 (Benito)

Santa Cruz: Just finished NB comments on PTR Ballot. DTR will go out in about 3 weeks. Language dependent annexes for ADA and SPARK are underway. Possible work in Fortran. Work is proceeding on a C language annex with assistance from Tom Plum and Robert Secord. Next WG23 meeting will be outside Venice, Italy the week prior to the C meeting in Florence.

Florence: WG23 met last week, TR balloting is complete, and should go to ITTF soon. Next meeting is in Hawaii.

Boulder: TR passed, revision in process.

2.8 MISRA C

Category C Liaison. N1471 on the agenda.

2.9 Floating-point Study Group (Thomas)

Santa Cruz: A study group has been formed, headed up by Jim Thomas, to address Floating Point issues to provide input to the work of IEEE 754. The group meets by teleconference and has about 17 participants. A wiki is available. Contact Jim Tomas, or the Convener for additional information. Working documents being developed. Goal is to produce a proposal for a Type 2 Technical Report.

Boulder Discussion: Jim Thomas. Nothing new. Making slow but steady progress, meeting via teleconference twice a month. Contact Jim Thomas for details.

2.10 Secure Coding Study Group (Keaton)

Florence: See N1450
Anyone interested in this report, or this SG, should contact David Keaton.

Boulder Discussion: Plans to ask WG14 for a Type 2 TR, after C1X is complete. Tentative meeting in Mid July. Contact David for details.

2.11 Other Liaison Activities

None

3. EDITOR REPORTS

3.1 Report of the Rationale Editor (Benito)

The proposal paper's author is responsible for writing the rationale.

Convener is adding text to the rational.

3.2 Report of the Project Editor (N1426) (Jones)

Open Issues:
1. N1371 - 21.5 changes need to be completed.
2. The index probably needs work, particularly for the newly added material.
3. The list of changes from the previous edition needs to be populated.

WP is up to date for the above 3 items.

4. The library synopses needs to be examined and extraneous headers removed.

JB emphasized we want to do a combined CD Registration and Ballot, which is what the SC22 Secretariat (ANSI) wants to see.

Clark asked what was the state of 'raw strings'. It was withdrawn after problems came up while incorporating the paper, and the consensus was to leave the feature out of the WP. We decided in Markham, via straw poll, to withdraw the proposal.
Tom believes that the Unicode portions should have been left intact. Clark: We don't have the UTF-8 literals. Markham minutes said we will wait to see what C++ does. It's now likely a C++ compatibility issue.

ACTION: Clark to look into the issue of UTF-8 literals w/r/t inclusion into C1X for C++ compatibility.

3.3 Status of TR 24731-1, Bounds Checking, Static (Meyers)

TR has format problems that need to be resolved prior to publishing.

3.4 Status of TR24732, Decimal Floating Point (Kwok)

TR 24732 is now published. Type 2, must be paid for. We have some comments on it. Edison is willing to do the work.

Do we need a new Editor? Yes. Mike will check.

3.5 Status of TR 24731-2, Bounds Checking, Dynamic (Benito)

Florence: Convener is coordinating with Andrew Josey, Austin Group, for editor support so we can make the final changes need to get this published.
Boulder: A Chapter was missing. ITTF made comments. We asked Andrew Josey to help, he tried, gave it back. Larry, David, and John will sit down and try to figure it out. ITTF is allowing us some flexibility in getting this done.

4. FUTURE

4.1 Future Meeting Schedule

Spring 2011 - Open
Fall 2011 - Portland (tentative)

Future meetings could be at INCITS, Santa Cruz, or we can get sponsors for Europe, but right now we have no meetings scheduled for 2011. 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

4.3 Future Mailings

None

Post Boulder:	25 Jun, 2010
Pre Batavia:	11 Oct, 2010

5. DOCUMENT REVIEW

5.1 Extensible Generic Math Functions (N1340) (Plauger)

Santa Cruz Notes:
This is the same paper we discussed Santa Cruz, along with N1401, Type Generic Macro Support. N2401 builds on N1340. When that paper was presented, Ulrich Drepper mentioned that the GNU compiler also implements features that can be used to implement type-generic macros. For the committee's consideration, those features are described in this paper as well as an alternate approach.

Discussion:
Clark presented this paper, and does not believe the GCC approach is a direction we should go in. He believes we should go in a direction closer to Plauger's earlier paper, N1340. That paper proposes a construct that resembles a function call, including a comma separated list of associations between the type-name and a function. It proposes a new alternative for the grammar primary-expression called generic-selection, with a new language construct _Generic.

How general or how specific to the type expressions need to be? Unknown at this time.
Why not adopt the EDG approach? (See N1340). EDG's implementation is really only suitable for floating point, and does not extend to atomics. Clark's proposal is a plausible extension of the EDG approach to include covering atomics. It does not touch on the preprocessor.

Where do we want to go with this? The only concern expressed thus far is one of macro expansion. No one has done tgmath this way? GCC has, but it's brand new. Concern about the lack of real world use cases. Implementation of tgmath to date is basically through compiler magic. Greater use of tgmath is likely with the coming of atomics and decimal floating point. We can either come up with a mechanism that's usable to implementers, or leave it to them to figure out the compiler magic.

Blaine finds the style of overloading very troublesome, and we are inventing rather than standardizing existing practice. It's interesting, and we should pursue a solution, but perhaps use a real overloading mechanism.

PJ: Perhaps we are overreaching with atomics. Expect to have more implementation experience later. Agrees that we are venturing into invention.

How desirable is generic support? We crossed that bridge in C99. It's there. tgmath is a flamingo with no legs, it has nothing to stand on.

Where do we go from here? Pursue this or not. Tom believes that decision has already been made, and does not want to see it dropped on the floor.

Santa Cruz Straw Poll: Do we want to go in the direction suggested by N1340?

Yes – 12
No – 4
Abs – 8

Boulder: N1340 defers to N1441. See 5.9.

5.2 C and C++ Alignment Compatibility (N1447) (Crowl)

N1447 presents what is believed to be the consensus proposal of the C and C++ compatibility mailing list, and is a revision to N1414 discussed in Santa Cruz. It incorporates detailed proposed wording for both C and C++ standards.

The revision to C++ is now in FCD Ballot, and none of the proposed C++ wording in this document, N1447, is contained in the C++ FCD.

Santa Cruz Notes:

[From the paper's Introduction (N1414)] The compatibility between the C and C++ threading facilities is important to many members of the respective communities. There are at least three levels of compatibility: critical, important, and desirable. Of these, the C and C++ committees should commit to achieving critical and important compatibility. This paper analyses the compatibility between current draft standards, and recommends several actions to improve that compatibility.

The most useful kind of compatibility is when an application header using thread facilities can be included by and used from both languages. Furthermore, it is desirable for C++ programs to be able to use C++ syntax with objects from that header. The recommendations within this paper support that goal. [end of intro material]

[Discussion]

Paul McKenney presented the paper from Lawrence Crowl. The paper proposes a number of recommendations for specific issues. Herb Sutter has put together a teleconference to discuss and attempt to resolve issues of incompatibility. There will likely be several teleconferences. JB encouraged those who have concerns or interests in the issues presented here to get involved with the teleconferences.

Robert pointed out that adoption of new facilities, such as threads, opens up potential for new security / vulnerability issues, implying somebody should be concerned about it. Tom pointed out that adding new portability features will always increase vulnerabilities. Doug believes the only *real issue would be that of one thread being able to access another thread. There is a larger issue of where the computing model is going, multi-threaded parallel processing, et al.*

Herb will broadcast an invitation to C, C++, POSIX groups to participate, however such a teleconference has no official standing.

Florence Discussion:
Paul McKenny – alignas Operator. Paul covered the proposed changes to C. Blaine concerned about not having a more complete solution, although he supports the concepts of this paper. There are no specific objections to this paper. No decision at this meeting.

Boulder Discussion: Paul walked us through N1447. The proposed changes for C++ are Not in the current FCD.

Fundamental Alignment – Proposes to constrain alignments to powers of two.

Alignof Operator – no proposal

Declaring Alignment

Spell the C1X keyword as "_Alignas", and revert the C++ proposal to a keyword:

```
decl-specifier:
        alignment-specifier
alignment-specifier:
        alignas ( type-id )
        alignas ( constant-expression )
```

Introduce a system header, <stdalign.h>, analogous to <stdbool.h>, that creates an identical spelling in both C and C++ for the keyword. Example:

```
#ifndef __cplusplus
#define alignas _Alignas
#define __alignas_is_defined 1
#endif
```

Alignment semantics

Define each individual alignment to zero to have no effect. That is, when there are multiple alignments specified, only alignments to zero have no effect.
A portable program cannot make use of relaxed alignment, so we propose that it remains disallowed in C and in C++.
Require that a definition or a tentative definition specifies the alignment; other declarations shall have the same alignment or no alignment.

Alignment Allocation

Update the C++ working draft section 1.2 [intro.refs] to refer to the 2011 C standard. This date may need to change.

Details of proposed wording for both C and C++ are contained in the paper.

It is unknown at this time whether the C++ proposed changes have been submitted as ballot comments for the C++ FCD.

Straw Poll: Adopt N1447 to the C1X WP.
13, 0, 0

Decision: Adopt N1447 to the C1X WP.

5.3 Mutexes in C and Compatibility with C++ (N1437) (Williams)

Background: From N1473
The current C1x draft (WG14 N1425) has a single mutex type (mtx_t), the properties of which are specified when each instance is initialized. On the other hand, the C++0x draft (WG21 N3000) has 4 distinct mutex types (std::mutex, std::timed_mutex, std::recursive_mutex and std::recursive_timed_mutex), each with predefined properties.

This discrepancy was highlighted by Lawrence Crowl in his paper on C and C++ compatibility in the thread library (WG14 N1423, WG21 N2985), and has since led to much discussion on the C and C++ compatibility reflector, and a series of informal teleconferences. This paper has come out of these discussions.

Two primary issues are discussed:

1. *Sharing of mutexes between C and C++ code, and*
2. *Performance of mutexes in C code.*

Florence Discussion:
David Keaton walked us through N1473, which contains proposed wording for the revision. This paper does not reflect an implementation. It is standardese. PJ does not believe there is anything here that is markedly better than existing practice. Without evidence of this being a better way, PJ is opposed to this paper. Blaine would like to take a look at the performance issues, and sample implementation in the next day, and come back to this later. Herb proposed a teleconference on this paper at a later date, which was agreeable. Herb is available after May 10. Tom pointed out that WG14 is highly unlikely to approve anything for which there is no implementation, or field experience.

Boulder Discussion:
N1473 is unchanged since Florence. C++ has four kinds of mutex objects, C has one. In cases where C and C++ code, using mutexes, are used together, there is a clear incompatibility. One answer to that problem is don't do that. The paper proposes adopting the C++ model of using four types. Herb pointed out there are also semantic issues between the C and C++ approaches depending on whether or not the operation is recursive. Why is that a problem? mtx_t is designed to be recursive. Wanting to use it in a non-recursive way is bad design. C++ has overloading, making a fundamental difference between the two. One should not expect to be able to use the C++ mutexes in C.

Herb sees this as two issues: 1) C/C++ compatibility, and 2) Correctness of code. PJ and Blaine disagree with #2. They do not see a correctness of code issue. Hans only concern is really the compatibility issue.

PJ: If we want an all inclusive compatible library between C and C++ we have much more work to to. That's not what people do. You should not want to mix languages. I/O streams is a clear example. Mixed I/O between C and C++ is typically done by using one or the other rather than mixing the two. That model should be the same here. Hans sees this as a different issue. Blaine sees the four types used in C++ as overly complex, and forces him to thoroughly review his entire design to make sure they are used correctly. He want something that is much simpler to use, that does what he wants. The C proposal is based on existing uses of mutex.

Another argument is one of space and performance, but there's no real data to indicate that argument has real merit.

Tom: We have discussed this several times during the C/C++ compatibility teleconferences. He sees the burden of proof on the request for C to change, i.e. the C++ approach. He believes that C++ approach is based on a best guess, by experts, that the C++ approach will solve coming issues in the marketplace. He does not see the burden of proof as having been met.

Blaine: The performance material in the paper does not represent real world use of mutexes.

There is a sense that the C++ approach is too much 'invention' to solve problems that are not real world. That approach conflicts with C's goal to standardize proven existing practice, and keep it simple.

Herb: Isn't there a benefit to a programmer to choose whether or not the mutex is recursive or not? I don't want to be forced to have to use both kinds, making them try to work together. PJ: Use the greatest common denominator.

Herb is using an iPad. How many mutexes does it have? Blaine knows, but won't tell.

Paul: Deadlock analysis tools would also have to deal with every type of mutex.

David: Can't find an argument that the type based solution (C++) has any application in the real world.

Rajan asked why mutex_lock (7.24.4.3) could return busy. PJ: don't know enough to answer that question.

Straw Poll: Adopt N1437 into the WP.
3, 10, 4 - no consensus

5.4 Updates to the C++ Memory Model Based on Formalization (N1480) (N1446) (McKenney)

Florence Discussion: Paul McKenny presented N1446. Blaine expressed concerned over some of the wording. David K concerned about the proposed wording as well, re: paper changed to "might be" changed to "is". (5.1.2.4p2). Paul believes N1425 addresses Blaine's concern. David's concern is that the proposed words Invalidate the base memory.   Clarify the scope as normal memory? Then we have to define normal memory. What happens to other forms of legal C declared memory needs to be addressed. Who will do that? Blaine thinks several of these papers that deal with atomics needs to be better understood and viewed w/r/t Blaine's N1462. Mike Wong believes a better understanding is needed w/r/t what atomics bring to C. Blaine suggest deferring discussing these papers until tomorrow, but Paul sees nothing in Blaine's paper that resolves the items they have been discussing. Defer to later discussion.

Boulder: N1446 has not changed since Florence.
Blaine: What does object refer to? Atomic object? Paul: Object refers to any object.
If a pointer is being shared with other threads, atomics should be used. Otherwise a data race may occur.
Blaine: There can be a data race due to improper ordering, or using the wrong fence. We do not have initialization semantics for an object w/r/t other threads, Paul: It's all handled according to

the 'rules below'. That case would fall out according to the rules, would likely be a data race, i.e. undefined behavior.
The issue really has to do with changing 'might be' to 'is'. Blaine believes the change is too restrictive. Larry suggested changing 'assigned to' to 'stored in'. Both Blaine and Paul are happy with that. Hans has an editorial change to the second paragraph. Larry will fix. "last value assigned' should be 'last value stored'.

This paper needs a new number: N1480.

Straw Poll: Adopt N1480 modulo edits above into the WP.
13, 0, 0
Decision: Adopt N1480 modulo edits per usual into the WP.

5.5 Dependency Ordering for C Memory Model (N1444) (McKenney)

N1444 proposes an addition to the C memory model and atomics library for inter-thread data-dependency ordering. This proposal is based on a similar proposal for C++ (WG21 N2664), most of which was adopted for the revision to C++. This proposal admits a trivial implementation, limiting significant implementation investment to those compilers and platforms where that investment will be recovered.

SC22WG14.12019:
Douglas Walls: If I understand the proposal correctly it adds: explicit program support for inter-thread data-dependency ordering. Programmers will explicitly mark the root of tree of data-dependent operations, and implementations will respect that ordering.
It appears to add another enum to memory order that may be supplied to operations for which data-dependency semantics are permitted. As I've noted in the past I do not fully understand all of these various forms of memory ordering operations.

As this potentially aids performance for TSO I do support it. Though I'd rather see vast simplification to only one memory ordering for the standard.
--- End SC22WG14.12019 ---
Ed: TSO means Total Store Ordering. "Total Store Ordering is the name for the ordering implemented on multi-processor SPARC systems. TSO basically guarantees that loads are visible between processors in the order in which they occurred, the same for stores, and that loads will see the values of earlier stores." Explanation provided by Douglas Walls.

Florence Discussion: Tuesday
Paul McKenney presented an overview of N1444, which evolved into a discussion of details with Blaine regarding the types of hardware that could be supported, and memory ordering schemes. Why was 'lock' changed to 'mutex'? To be consistent with the same change made to C++0X. The term is specific to the facilities provided in C and C++. Blaine thinks the atomics section needs to be reviewed simply for clarity. Clark wants to know if everyone is on board about the concepts presented in this paper. David K thinks it looks good. Blaine thinks his atomic proposal is an add on. Tana points out that C++ is moving forward, so if we have a comment on it, we should make it. Tom suggests we have no other alternative approaches before us. Blaine thinks the level of detail is fine for those who need it, but not for the C Standard, but also agrees with Tom. Tom generally agrees with the paper presented. Walking through the proposed wording is likely to be unproductive. If anyone has comments, forward them to Paul and Clark.

Boulder Discussion:
N1444 is unchanged since Florence.
Blain raised a number of 'what if' scenarios, that Paul suggested they amounted to design flaws.

Straw Poll: Adopt N1444 into the WP.
17, 0, 1
Decision: Adopt N1444 into the WP.

5.6 Rationale for C Language Dependency Ordering (N1461) (McKenney)

N1461 lays out a rationale for the memory model, including dependency ordering. There are many parallel programming models, and different applications will be best served by different programming models. As always, use the right tool for the job at hand. This document's focus is on the case where the right tool for the job is the multithreaded shared-memory programming model, as this is the programming model that stresses the memory model, which is the topic at hand.

Boulder Discussion:
N1461 is Rationale for N1444. As N1444 has been accepted as is, there is no need to discuss this paper.

5.7 Explicit Initializers for Atomics (N1482) (N1445) (McKenney)

Florence Introduction: C/C++ Compatibility Issue

N1445 describes a compatibility issue between C++ and C for atomics. Unlike C++, C has no mechanism to force a given variable to be initialized. Therefore, if C++ atomics are going to be compatible with those of C, either C++ needs to tolerate uninitialized atomic objects (ed - that's not going to happen), or C needs to require that all atomic objects be initialized (ed - that's what we have to do). There are three basic cases to consider: 1) C static variables, 2) C on-stack auto variables, and 3) C dynamically allocated variables.

Florence Discussion: [Wed]
Paul McKenny presented and overview of N1445. Any objections to the general concept – all atomics in C be initialized? None.

Boulder: Paul lead us through the salient features of N1445, same as that reviewed in Florence. Needs rationale for the macros proposed for C.   Needs C wording for note w/r/t memory_order_relax. Blain feels that the use of atomic_init will under certain circumstances be onerous for users. Jim Thomas expressed concern regarding macros for extend integer types. Tom believes that would be an extension, and need not be defined in the standard. Larry Jones want to see a separate clause for atomic_init, with more information as a generic function.

Wednesday: Paper has changed as a result of several email exchanges, and was posted to the Wiki, WG14 main page. There are additional refinement details that will be provided later today.
Is ATOMIC_VAR_INIT a TG macro? No.
atomic_init does not avoid data races. There is no way to avoid this w/o affecting C++ capability.

N1482 is the revision to N1445 per the above discussions.
Clark likes the words used for ATOMIC_VAR_INIT for the words used, "Concurrent access…" be used in the description for atomic_int. Editorial. Delete the "..where it applies.."

Straw Poll: Adopt N1482 into the C1X WP, modulo the usual editorial changes.
14, 0, 1
Decision: Adopt N1482 into the C1X WP, modulo the usual editorial changes.

5.8 Atomic Proposal (N1485) (N1476) (N1473) (Garst)

N1476 is a revised version of N1473, which was a revision of N1452.

Florence Introduction: N1452 is an alternate proposal for atomics that differs from the one already adopted for C1X. It adds language type qualifier.

SC22WG14.12013
Blaine Garst: My N1452 is incomplete - a discussion of allocated memory w.r.t. atomic objects is necessary, and a revision should be expected before discussion. As it stands, I don't think allocated memory is fully consistent with existing C Standard atomic library objects since they seem to start out with undefined values but the process of establishing initial values before visibility elsewhere is not defined yet is essential for determinism. This seems bad. Requiring "0" initial values (for integers at least) and that calloc implement memory-store barriers seems sufficient, but may be too costly - a new "alloc" with alignment, realloc, memory-store barrier, initial values, may be the best alternative. Complete treatment may require new library support, but let it be minimal instead of maximal.

Florence Discussion:
Blaine Garst presented N1452. His view of using the language to program directly to the hardware with regard to synchronization of operations across multiple threads of control. The paper is essentially a white paper that could be a starting point for development of a more complete proposal. It consists of adding an '_atomic' type qualifier to volatile marked memory with a specified set of results. The concept described here does not have wide industry use. Tom favors a sequentially consistent model for C and C++.

Clark: Three questions. 1) if added, do we need to describe interaction w/library atomics. 2) if compiler is free to reject the request, what are the implications on strict conformance, 3) how would this impact C language schedule.

Blaine: 1) compatible with atomics library, this is just an add on. 2) Blaine is OK with that, others may not be. Tom does not support compilation error, 3) not clear.

Tom does not agree with going in the direction of a compile time error if the feature is not implemented.

Is there an implementation? No. At least not one that can be talked about.
Rajan likes the simplicity of the proposal, speaking as a programmer. Douglas expressed support as well.

[WED]

SC22.WG14.12062, Blaine Garth (excerpted)
One revision:

1) __atomic should apply to all objects, and not just the ones the compiler writer decides upon. This requires a strategy for atomic assigns of larger-than-word sized structs. This is not hard, I *can and did sketch a commonly used algorithm for dynamically grabbing a lock, but dedicating some extra storage is another option, so there are several acceptable resolutions to make this reasonable.*

There are two serious issues to resolve:

1) should += (et al.) have the strongest sequentially-ordered semantic (matching C++ atomic template use), or is a weaker ordering acceptable, as I have indicated in the proposal.

I'll speak with Hans about this since he has strong opinions across years of wider experience than me. I feel pretty strongly, though, that the operations should map to the hardware and not add expensive yet rarely used value. For obvious reasons I'm going to worry about ARM as much as Intel, while fondly remembering PPC as well - I have stories to tell about G5 instruction execution traces...

2) need to further specify structure behavior, in particular field stores. E.g. What does an inner __atomic mean to a structure declared with __atomic?

With some thought, I think that

a) operations on the whole structure are obviously atomic, as in assignment. Initialization of auto storage does not need to be atomic since it can't be shared, but that might be an optimization instead of a specification.
b) operations on field members are probably not, unless they are also specifically marked __atomic. This lets initialization happen on a per-field basis.

I hope to invite the interested-all to participate, not sure what forum. I'll make that my first item tomorrow, because I think the discussion on __atomic was all but 30 seconds from wrapping up when the hook came out :-) Feel free to sign yourself up right now if you haven't already been talking on this topic in committee.

Blaine
---- end excerpted SC22.WG14.12062 ---------------

[Wed]
Q to Blaine from JB. What is it we've agreed to? Blaine covered his view of the technical aspects. JB: We want to get closer to C++ compatibility. Blaine: We are moving forward in that direction.

N1473 is a revision to an earlier paper, N1452, same subject. It proposes to add an _atomic type qualifier to volatile marked memory.

Boulder Discussion:

Proposal is to add a type qualifier _Atomic. Blain has had several conference calls with Hans, Lawrence, Paul, and others on the details of this proposal. There are still details to work out

Blain covered some background leading into this proposal, and his view that a language formulation was needed for atomics, in addition to the library features. Without language support, C programmers would be forced to use only the library components. This proposal addresses only language features. A good deal of effort has gone into making sure that nothing here conflicts, or is otherwise incompatible, with C++0X. There are two areas where the proposal disagrees with what C++ is doing.

C++ seems to require extra unspecified bits for locking, or to use a side table of locks. Blain is unsympathetic to having to allow for all implementations for atomics. An _Atomic int is not always the same as atomic _int. Requiring a provision to have hidden lock bits seems unnecessary, and complicates the proposal. A side table of locks is all that is necessary. Addressing the needs of unknown, TBD, future hardware architectures is not practical.   If you really want atomic _int, use the library. True that ultimate efficiency is thrown out with the type qualifier _Atomic.

The treatment of atomic struct differs between C and C++. C treats every field in a struct to be atomic if the struct is atomic. C++ does not. Clark: In C++ there are no atomic structure data types. Blain: That differs from what Lawrence has been saying. Clark believes that Lawrence is talking about a future extrapolation of the C++ Standard. Tom: The C++ FCD has no such feature. Where's the conflict? Herb believes doing so (creating an atomic struct) is wrong anyway, since it will create data races. Blain agrees that is a possibility. The perceived conflict between C and C++ does not exist here. C++ does not allow access to members of an atomic struct, but there is disagreement on whether or not creating an atomic struct should be done at all. Better answer may be to leave access to structs undefined, i.e. reading or writing to struct members is undefined.   A normal struct containing an _Atomic member is OK.   The implementation can give a different alignment to that member. What does _Atomic struct mean to the field members? Cannot promote _Atomic down through the members. (revision to paper).

Treatment of unions is also an issue between C and C++. Unions are not allowed to hold any atomic numbers in C++ if it is not trivially constructible.   That prohibition does not apply to C since trivially constructible does not apply to C. From a C perspective, that's too prohibitive.   We need to resolve that. Paul has a proposal on that?

Tom now feels like the differences between the C / C++ atomics library and the language features as proposed here are greater than he initially believed them to be.

Clark: We should get a sense of where we want to go with this proposal. We are spending time discussing this without having a sense of consensus to pursue it.

Straw Poll: Do we want to get this into C1X ?
Yes: 6
No: 5
Abstain: 5
No consensus to add N1476 to C1X.

Discussion:
Paul, IBM likes this, but not for the WP.
Jim, HP has the same concern.
Herb, MSFT does not see this fitting into C1X.
Blain, Apple is very interested in seeing this happen.

Thursday Afternoon
N1485 is an extensive revision to N1473 presented earlier this week. It contains compatibility with C++. Is the devil still in the details, or have the details been worked out? The proposal now is complete. Removing anything may require adding wording. Blain walked through the High Level Description.

Clark: Is there a syntax to ask if an atomic type is lock free? There is a macro to check that. That answer does not satisfy Clark.
atomic_int works with assignment operators like C++

Tom is opposed to the addition of a new ternary operator (?=:). Paul agrees. David believes the operator simplifies the code, thereby making it more secure. Paul disagrees. Since this operator is intended to be the only mechanism for doing compare-and-swap on atomic bitfields, is the objection only for the operator. Yes. PJ believes it is very dangerous to introduce an new operator in a language. Would like to see all of this stuff go into a TR.

Blaine can have an implementation w/in 6 months. PJ: that's not enough time, or enough experience.

Blaine is not opposed to doing a TR, but believes that right now, we have an opportunity to do something significant. He does not want to lose this over a ternary operator, and is willing to remove it. Other issues? Floating point – certain operations. double equal. If C gets the same behavior as C++, this is not an issue.

Tom: A view of why it is important to do something now. The subset of what we are trying to do with atomics is pretty meager. If programmers have only the library functions to use, it makes C look second class w/r/t C++. Getting language tools gives the programmer greater flexibility. Do we want C1X programmers to be able to use operators on atomic data? Tom is happy with Blaine's proposal with the ternary operator dropped.

Jim: This is not really something that's mature enough to bring to a Standard.

JB sees three choices:
1) vote in now, work edits, CD in Nov
2) don't take atomics now, vote out a CD now, move on
3) vote in now, CD now.

Straw Poll: Adopt N1485 into C1X WP, modulo removal of #6 (ternary operator)
6, 2, 5 – consensus to adopt N1485
Decision: Adopt N1485 into C1X WP, modulo removal of #6 (ternary operator).

5.9 General support for type-generic macros (N1441) (Nelson)

Santa Cruz Introduction:

N1401, Type-generic Macro Support, builds on N1340. When that paper was presented, Ulrich Drepper mentioned that the GNU compiler also implements features that can be used to implement type-generic macros. For the committee's consideration, those features are described in this paper as well as an alternate approach.

Santa Cruz Discussion:
Clark presented this paper, and does not believe the GCC approach is a direction we should go in. He believes we should go in a direction closer to Plauger's earlier paper, N1340. That paper proposes a construct that resembles a function call, including a comma separated list of

associations between the type-name and a function. It proposes a new alternative for the grammar primary-expression called generic-selection, with a new language construct _Generic.

How general or how specific to the type expressions need to be? Unknown at this time.
Why not adopt the EDG approach? (See N1340). EDG's implementation is really only suitable for floating point, and does not extend to atomics. Clark's proposal is a plausible extension of the EDG approach to include covering atomics.   It does not touch on the preprocessor.

Where do we want to go with this? The only concern expressed thus far is one of macro expansion. No one has done tgmath this way?   GCC has, but it's brand new. Concern about the lack of real world use cases. Implementation of tgmath to date is basically through compiler magic. Greater use of tgmath is likely with the coming of atomics and decimal floating point. We can either come up with a mechanism that's usable to implementers, or leave it to them to figure out the compiler magic.

Blaine finds the style of overloading very troublesome, and we are inventing rather than standardizing existing practice. It's interesting, and we should pursue a solution, but perhaps use a real overloading mechanism.

PJ: Perhaps we are overreaching with atomics. Expect to have more implementation experience later. Agrees that we are venturing into invention.

How desirable is generic support? We crossed that bridge in C99. It's there. tgmath is a flamingo with no legs, it has nothing to stand on.

Where do we go from here? Pursue this or not. Tom believes that decision has already been made, and does not want to see it dropped on the floor.

Do we want to go in the direction suggested by N1340?

Yes – 12
No – 4
Abs - 8
--- End Santa Cruz Discussion ---

Florence Introduction: N1441 contains proposed wording to specify the _Generic construct proposed in N1404. In Santa Cruz, there was consensus (12-4-8) to adopt the approach presented in N1404.

SC22WG14.12021:
Douglas Walls, indicated support for this paper over N1340.

SC22WG14.12011:
Douglas Gwyn, n1441; what about use on types for which no default or….
I would urge the Committee to minimize invention of new linguistic features for C. Indeed, C99 type-generic macros ought to be revisited to determine if they have been sufficiently useful to offset their difficulty in complete specification and implementation. Similarly for sizeof(VLA), which is harder to make good use of than simply using the expression that determined the VLA length in the first place.

Some invention may be needed when there is a demonstrated need for a feature, not portably available under C99, across a wide variety of platforms. But it should not add substantial complexities.

[end SC22WG14.12011]

SC22WG14.12015
Herb Sutter: Strongly supports DG position above.

Florence Discussion:
John Parks presented N1441, proposed wording for type generic macros. The general sense is that all of this stuff (type generic macros) is a kludge, but we seem to be stuck with them.

Boulder:
N1441 is unchanged from Florence. This facility is close to the EDG model, but better. N1441 contains the standardese needed to carry forward the concepts we agreed to for N1340.

Straw Poll: Adopt N1441 into the C1X WP.
12, 0, 0
Decision: Adopt N1441 into the C1X WP.

5.10 Comparison Macros (N1459) (Plauger)

Florence Introduction: C/C++ Compatibility Issue
The words in the current C1X WP say nothing about whether or not the two arguments in a comparison macro must have the same real-floating type. The same macros in the C++ FCD require the types be the same. We should either change the C1X WP to require the same types, or file a comment against the C++ FCD to not require the same type. PJ prefers the first choice.

Reflector: WG14.12023: Douglas Walls argues that C99 places no requirement that the two types be the same, and prefers the second option, filing a comment against the C++ FCD. Oracle's implementation allows different types.

Florence Discussion: [Wed]
PJ presented the background of the comparison macros, and the incompatibility between C++0X and C99. Bill is willing to file a comment, if that's what we want to do. HP and Oracle want to be able to use different types, which is what the C Standard presently allows. PJ views doing so requires a degree of compiler magic, but agreed to let the types be different after further discussion. Agreement that the C Standard may not be 'clear'. If the types are mixed, they can promoted to the common type. Tom: maybe we need to say the usual type balancing rules apply. It was never intended that compile errors result if the types are different. PJ regarded this as sufficient direction.

N1459 is a revision to N1442 discussed in Florence and reflects the view that the two argument types need not be the same type. The current FCD 14882 (C++0X) requires the two types to be the same. We believe the C++ FCD should be aligned with the existing C Standard. PJ has submitted a comment for the FCD.

Boulder Discussion:
Not clear what the right answer is. PJ believes the intent was that the types need not be the same. Such clarification could go into the rationale.

Straw Poll:
Add the following from the bottom of N1459 to he C1X WP.
"The two arguments to each function need not be of the same real floating type. "
No Objection.

Decision:
Add the following from the bottom of N1459.
"The two arguments to each function need not be of the same real floating type. "

5.11 Comparison Macro Argument Conversion (N1472) (Thomas)

N1472 calls for clarification in the C Standard for function like macros (Clause 7.12.14, Comparison Macros). The result of the comparison can differ according to when the comparison is made. If the comparison is made prior to the conversion of an argument (per the evaluation mode of that argument), the answer may differ from that obtained if the comparison is made after the conversion.

Three alternatives are proposed:
1. Specify that widened arguments of comparison macros must be narrowed (like classification macros).
2. Specify that widened arguments of comparison macros must not be narrowed (like relational operators).
3. Clarify that whether widened arguments of comparison macros are narrowed is unspecified.

Of the three, Jim Thomas prefers #2.
Clark: Why is 'do nothing' not an option.
Jim: Essential because the result would not match the specification in the case described above.
PJ likes #1 or #3. 'may' be narrowed is OK. #3 leaves it to the user.

Straw Poll: N1472, Adopt proposal #3 into the C1X WP

12, 1, 0

Decision: Adopt N1472 Proposal #3 into the C1X WP

Essentially leaves it unspecified.

5.12 Subsetting the Standard (N1460) (Plauger)

Santa Cruz: N1395, Wakker, Netherlands Contribution on the Growth of Programming Language Standards, SC22 N4484.

N4484 discusses the growing set of features being added to Standards for programming languages, and the problems created by requiring the implementation of those features on all systems, particularly specialized embedded processers. Creating a conformance category called a Conforming Subset Implementation, which would allow conformance to a subset of the entire standard, would help. An example in the C Standard is 'freestanding', however some believe more flexibility is needed.

Santa Cruz Discussion: In Santa Cruz, Plauger expressed support for the concept, others expressed concern of its practicality. No action was taken.

At the SC22 Plenary, the paper was discussed, and the following resolution (09-16) adopted:

Netherlands Contribution on Growth of Programming Language Standards JTC 1/SC 22, noting the discussion on SC 22 N 4484 (Netherlands Contribution on the Growth of Programming Language Standards), recommends its Working Group Conveners to distribute SC 22 N 4484 on the Growth of Programming Language Standards to their Working Groups as a JTC 1/SC 22 recommended document on guidance to address the issues identified in this document.

Florence Introduction:
N1443 proposes three feature test macros be added to the C1X WP:

*__STDC_NO_COMPLEX_*
*__STDC_NO_THREADS_*
*__STDC_NO_VLA_*

and suggests the Committee also consider a similar macro for atomics either as part of threads, or separately.

N1460 is a revision to N1443 discussed in Florence. It proposes the following predefined macro names:

*__STDC_NO_COMPLEX_*
*__STDC_NO_THREADS_*
*__STDC_NO_VLA_*

Boulder: PJ reviewed the Florence discussion of N1460. Per that discussion, he added <stdatomic.h> to threads as part of __STDC_NO_THREADS_. JB noted that VLAs are scattered throughout the Standard, and identifying exactly where the exclusion applies may be difficult.

Jim: Should the rational address the use of the convention __STDC_NO_feature_ . It's a negative macro. PJ agreed to do so.

ACTION: PJ to add word for the Rationale on the use of __STDC_NO_feature_

Douglas: Concerned that these features impact the portability of programs. Those who use VLAs will find that their programs don't work.
Clark: Implementations that have not implemented VLAs won't run those programs anyway,

Straw Poll: Adopt N1460 to the WP.
11, 1, 1, 0

5.13 Further Subsetting the Standard (N1471) (Montgomery)

In Florence, MISRA was asked if they had any additional items to consider for subsetting in C1X. N1471 is in response to that question, and proposes the following components be made optional:

*__STDC_NO_TGMATH_*
The integer constant 1, intended to indicate that the implementation shall accept only a strictly conforming program that does not include the header <tgmath.h>.

*__STDC_NO_FN_ID_*
The integer constant 1, intended to indicate that the implementation shall accept only a strictly conforming program in which no function declarator includes an identifier list even if empty.

*__STDC_NO_FAM_*

The integer constant 1, intended to indicate that the implementation shall accept only a strictly conforming program that does not use flexible array members.

Boulder: The second item, *__STDC_NO_FN_ID_*, is not understood. It seems to read the opposite if what we believe to be the intent – remove all old style function declarations, and accept only prototype function declarations. These words do not match that intent.

For #1, we know of no undefined behavior associated with TGMATH. Not hearing any support for these items. Put this proposal on hold until we figure out what we are going to do with tgmath, and discussion with Montgomery.

Thursday: The first element for TGMATH is consistent with what we've done this week. There are also type generic macros in math.h, that could prove to be more difficult than those in tgmath.h. However, given the work we've done, MISRA may want to review these items in light of that work.

ACTION: Convener to discuss N1471 in light if the new work we've done this week for the C1X WP with Steve Montgomery / MISRA.

5.14 Completeness of Types (Feather) (N1439)

N1439 proposes changes to the definitions of types to clarify that completeness of a type is a 'scoped property of a type', it can vary from place to place in a translation unit, without altering the type itself. While that's clear from the current text, it's not clear from the definition.

Florence Discussion:
Clive is not available. Tom: In most places where we say an object type, we should be saying a complete object type. Completeness is a scope property, once a type is completed, it is a complete type. There are function types and object types. Within an object type, it may be complete of not. Once it is completed, it is a complete type. It can change back to an incomplete type at the end of the block. Larry believes Clive is correct, but it's a lot of work.

Boulder Discussion:
There is no opposition to Clive's comments. Tom believes that this paper says what we meant.

Straw Poll: Adopt N1439 into the C1X WP.
13, 0, 0 – Consensus exists.
Decision: Adopt N1439 into the WP for C1X.

5.15 Constant Expressions (Adamczyk) (N1448)

Florence Introduction:
N1448, asks if the following code is valid in C

```
int i = 1 || (2, 3);
int main() {
  return 0;
}
```

A literal reading of the C99 Standard suggests the comma operator is valid because it appears in an subexpression that is not evaluated, which differs from C89.

Florence Discussion: Douglas Walls is opposed to this change. Tom: The net of Steve's proposal is the compiler's would have to issue a diagnostic where in the past they did not. Tom suggested that the expression above may not even be a 'constant expression'. If it's not, the comment about the comma operator does not matter. Steve suggests that since the initialization is static, it must be a constant expression. Tom changed his mind. David K points out there is a footnote in 6.6;p11 that addresses this.

ACTION: Convenor to discuss this with Steve A.

Boulder Discussion:
No discussion in Boulder.

5.16 Supporting the noreturn Property in C1X (N1478) (N1453) (Svoboda)

N1453 proposes to add a new keyword, _Noreturn for use by functions. Doing so could tell the compiler to assume the function cannot return, and allow optimization without regard to what would happen if the function did return.
Example:

```
_Noreturn void fatal (void);
void fatal() {
exit (1);
}
```

This is supported by MSVC and GCC.

Boulder Discussion:
David Keaton presented N1453. C++0X has 'noreturn'. Tom concerned about creating a keyword that could be used as a built in macro, but favors the proposal. Leave it to the implementer?

_Noreturn needs to be added to the keyword table.

The paragraph, "Add a new paragraph", that says "..if the implementation cannot be certain.." is not typically something we say. Tom: C++ calls the behavior undefined. Keaton agrees to delete that requirement. Make the second sentence "Recommended Practice". Cannot use "shall" in an RP.

Straw Poll: Propose something along the lines of N1453 for inclusion into the WP.
No Objection

ACTION: David to revise the wording for N1453, moving the second sentence to Recommended Practice, and other changes as needed, such as adding _Noreturn to the keyword table.

DONE – N1478 contains revised wording.

Straw Poll: Adopt N1478 to the WP
13, 0, 0

Decision: Adopt N1478, noreturn, to the WP

5.17 Type Punning Example (N1463) (Tydeman)

N1463 proposes to add an example of type punning, which is mentioned in a footnote in C99, Clause 6.5.2.3, Structure and Union Members; and words for the Rationale, clarifying that type punning differs from type conversion.

Boulder Discussion:
Jim concerned that the initialization of complex shown in the proposed example is not the really right way to do this. Clark: Why do we want to add a bad example?

Straw Poll: Adopt N1463 into the C1X WP.
1, 7, 5 – no consensus to adopt N1463

5.18 errno and threads (N1462) (Tydeman)

Florence Introduction:
N1427 proposes to specify, for threads, each thread has its own errno.

Florence Discussion:
The initial state of the errno is indeterminate. The proposed footnote tells us nothing. Suggested removing it.

N1462 is a revision to N1427 discussed in Florence, proposing that each thread has its own errno, and that the initial state for a thread's errno is indeterminate.

Boulder Discussion:
Some existing implementations have errno as part of thread local storage, others do not.

Straw Poll: Adopt N1462 into the C1X WP
12, 0, 1
Decision: Adopt N1462 into the C1X WP

5.19 Floating Point Flags to errno Correspondence (N1470) (Tydeman)

N1470 proposes to clarify the correspondence between math errors, floating point exception flags, and errno.

Boulder:
Is this material normative? It seems to be as presented, and probably should not be.

Straw Poll: Adopt N1470 into the C1X WP
1, 10, 2 – no consensus

5.20 ilogb (N1467) (N1428) (Tydeman)

Florence Introduction: Contains three proposals regarding ilogb.
Background: As a function that maps reals to reals, sqrt(-1.0) has no usefully definable result, and the operand is invalid, so is a domain error. In like vain, as a function that maps reals to integers, ilogb(0.0) has no usefully definable result, so it should be a domain error. In addition, IEEE-754-2008 requires ilogb(NaN or infinity or 0) to raise invalid.

Florence Discussion:

What do we really want to do about new 754 standard. Do we really want to put 754 requirements in the C standard? Clark considers this the nose of the camel. PJ does not want to tinker until we get a final report, in the works, w/r/t 754 impacts on the C Standard. No support for Proposal 1 or 2.

Proposal 3 in this paper addresses an incorrect statement in C99 (really a DR). F.9.3.5. PJ thinks the case is so unlikely, has not happened in any machine he knows about. Jim T believes the statement is OK as is.

Boulder: N1467 is a revision to N1428 discussed in Florence addressing a contradiction between C99 (7.12.6.5) and C99 Appendix F (10.3.11), i.e. Proposal 3.
Why isn't the existing text sufficient? Jim: The text between Annex F and C99 is contradictory. Just reading Annex F, w/o reading C99, can create some confusion. PJ thinks this is a change of existing behavior, and is opposed to making the change. Jim: There's two ways to interprect this. PJ: the main body says you can trap if you want, but it's not required. Requiring it is a change.

Straw Poll: Adopt 1467
1, 7, 4 – no consensus

5.21 fabs (N1486) (N1466) (N1429) (Tydeman)

Florence Introduction: Defect Report
Background: IEEE-754-1985 suggests and IEEE-754-2008 requires fabs(NaN) to affect just the sign bit.

Florence Discussion:
PJ disagrees. As with the earlier remark, PJ does not want to tinker with this. Different nose of the same camel. Meta rule: Changes between 754-1985, and 754-2008 will wait until the FP Study Group is finished with their report.

Boulder: N1466 appears to be a revision to N1429, which in Florence we said we did not want to deal with. N1486 is a revision to N1466.
NaN is not a number, but it can have a sign bit. Annex F does not yet track the new IEEE Std. Why to we want to make any changes to it?
The sentence: ".. the fabs functions in provides.." should be "..in math.h.."

Straw Poll: Adopt N1486, modulo several edits, into the C1X WP
Clark wants the poll split to break out the F.3 bullet

Straw Poll: Adopt the edit to F.3 in N1486 into the C1X WP
6, 4, 2 - convener says that's consensus.
Decision: Adopt the edit to F.3 in N1486 into the C1X WP

Straw Poll: Add f.4.10.2 edits to N1486
2, 3, 6 – no consensus

5.22 negate (N1465) (N1430) (Tydeman)

Florence Introduction: Defect Report
Background: IEEE-754-1985 suggests and IEEE-754-2008 requires negate(NaN) to affect just the sign bit.

Florence Discussion:
Same meta rule as above. Wait on FP Study Group. PJ prefers to leave it alone.

Boulder: N1465 seems to be a revision of N1430 discussed in Florence, same proposed wording.
PJ: This is a compiler operation? Yes. Somebody else's problem.

Straw Poll: Adopt N1465 into the C1X WP
3, 8, 1 – no consensus

5.23 Creation of complex value (N1464) (N1431) (Tydeman)

Florence Introduction: Three proposals on the creation of complex values.

Problem:

*(x + y*I)*

will NOT do the right thing if "I" is complex and "y" is NaN or infinity. It does work fine if "I" is imaginary. Users and library implementers have noticed this deficiency in the standard and have been surprised that there is no easy to use portable way to create a complex number that can be used in both assignment and static initialization. Three solutions are proposed.

Florence Introduction:
SC22WG14.12018
Douglas Walls: I support this proposal.
I prefer solution #3 macros over proposal #2 due to the ambiguous case talked about in the footnote. I think that footnote will need to become normative, not a footnote. But maybe I don't understand :-) I dislike proposed solution #1 as type punning can lead to aliasing issues.
--- end SC22WG14.12018 –

Florence Discussion:
Proposal 1: uses type punning
Proposal 2: invention from EDG
Proposal 3: adds new macros. HP has been shipping this solution for several years.

PJ: we need a notation. Tom: likes #3, and the names are as good as anything. Blaine, consider an operator over a macro. Douglas support the macro solution, but could support an operator solution as well. Jim T: If compilers supported the imaginary type, operators would not be needed. PJ: That's the problem, Imaginary is optional.

Boulder: N1464 is a revision to N1431 based on adopting proposal #3. Three new macros are proposed:

CMPLX(x, y)
CMPLXF(x, y)
CMPLXL(x, y)

These macros would return the complex value x + *i**y created from a pair of real values, x and y.

Boulder Discussion:
Implementation has existed for at least 10 years.
Can they be used as static initializers. Yes, as long as the arguments can be static initialized.
PJ: Are sample implementations available that do not use imaginary I? Unknown. Don't know how to do this w/o imaginary I, and use static initialization. Believe it will be years before everyone does. Drop static initialization requirement. Make that a Recommended Practice (after Returns). The "as if" sentence s/be a note? No, editorial.

Straw Poll: Adopt N1464 per the edit changes to make the static initialization a Recommended Practice.
11, 0, 1 – consensus exists
Decision:

5.24 Assumed types in F.8.2, Expression Transformations, (N1468) (N1435) (Tydeman)

Florence Introduction: Defect Report

Problem: The integer constants should be double constants.

Add a paragraph to F.8.2 Expression transformations

While the following code shows (and assumes) ***double*** *type, the same issues apply to all floating types. Change existing F.8.2 integer constants (0, 1, 2) to double constants (0.0, 1.0, 2.0).*

Florence Discussion:
The reference should be F.9.2 in the current WP. Clark: See no motivation for this change. PH agrees with Clark. Fred has worked with people that have found this confusing.

Boulder: N1468 is a revision to N1435 discussed in Florence, same proposed change, for which there seemed to be no support in Florence.

Boulder Discussion:
N1468 provides code that demonstrates the problem. Jim: This is a good thing to do, given what we have to work with in C. Larry may be able to clarify

Straw Poll: Editor to adopt something along the lines of N1468, that addresses the identified, into the C1X WP.
11, 0, 1

Decision: Editor to adopt something along the lines of N1468, that addresses the identified, into the C1X WP.

5.25 F.9.2: x-y <--> x+(-y) (N1469) (N1399) (Tydeman)

Boulder: N1469 is related to N1399 discussed in Santa Cruz, for which no action was taken.

Jim: This is above and beyond 754. It does not match the model. It would hamstring optimizers.

Withdrawn by Fred.

5.26 Blocks Proposal (N1451) (Garst)

Florence Introduction: N1451 describes a feature called 'blocks', a form of closures, as already implemented by Apple. In C, the term 'block' is already well defined, so it suggests that the term 'closure' to avoid confusion with the existing term 'block'. Usage of terms such as _block, Block_copy, et al, in this paper, simply reflect existing practice by Apple, and would have to be changes to something like _closure, Closure_Copy, etc.

N1451 also cites N1270 as "Apple's Extensions to C" as a prerequisite. However, N1270 is a set of draft minutes, so the cite is wrong.

This paper is incomplete in that the proposed wording reflects 'block' terminology rather than something the Committee would need to decide on, such as 'closure'. Also, the cite needs to be corrected.

Florence Discussion:
[Wed]
Blaine went through a slide presentation on Blocks (On the Wiki, BlocksFlorence2010.pdf).

Tom: There is likely to be a involved discussion on how blocks relate to lambdas in C++.
Blaine: The main difference is that lambdas are not closures. It's harder to share information in lambdas. Nearly equivalent, but they are different facilities.

Then covered N1451, overview section. There are four basic elements:

1) a new form of compound type paralleling function types
2) a new closure literal construct that creates a pointer to a closure object,
3) a new storage class, __block, representing closure object scope duration, and
4) language primitives Block_copy and Block_release.

This is new material for most of us, and there is some doubt as to whether or not most even understand the paper. More discussion will take place at the next meeting. If anyone has specific questions, contact Blaine. The reference to N1270 should be N1370, Apple's Extensions to C.

Boulder: N1451 is unchanged from Florence.
Blaine presented N1451, pointing out that blocks are heavily used within Apple. They are somewhat similar to lambda's (C++), but those used in Apple are designed for C. It's possible for both to exist in the same compiler, but lambdas proposed in C++ would have to be modified to be used in C.
The paper needs editorial work, i.e. blocks are already defined in C, and the terminology would have to be changed to 'closures'. Apple does not follow the C function declarative rules.

Apple has a patent application in the works for the blocks concepts described in this paper.
Tom: Blocks is a pure extension to C as present in C1X. Why should we consider it now? Blaine: C++0X has a similar but different feature (lambdas). C has no similar feature, and it should. It would provide feature parity.

David: Also sees the ability to do parallel dispatch, important in multi-core systems. Blain agrees, but Apple does not do that.

Blaine see a lot of uses for blocks (closures) as being done by Apple now.

Clark: The drafting in the proposal is incomplete, very incomplete. And it's not clear how much work is needed. There is zero chance that making the changes will not affect the schedule.

Tom sees a good chance that people will want to implement this over the next few years as a pure extensions, but does not see how we can include this in C1X.

Straw Poll: Adopt something along the lines of N1451 into the C1X WP.
5, 8, 4 – No consensus

5.27 SC22WG14.12207, Static Assertion in 'for' Loops (N1425)

In SC22WG14.12207, Joseph Myers asks the following:

> As I read N1425, the following use of _Static_assert is valid.
>
> ```
> void
> g (void)
> {
>     int i = 0;
>     for (_Static_assert (1, "");
>         i < 10; i++);
> }
> ```
>
> There is no requirement [*] that the declaration in a "for" loop declare anything at all, so the constraint in 6.8.5#3 is not violated. (Recall that said constraint was clarified in response to DR#277, those I don't think the present issue depends in any way on that clarification.)
>
> Is this intended to be valid? It doesn't look like C++0X allows static assertions in this context.
>
> [*] Other than the general requirement on all declarations in 6.7#2, but
> clearly 6.7#2 should not apply to static_assert-declarations at all.

Did we mean for the above code to work? No. Let the editor 'fix' this. Yes.

ACTION: Larry Jones to correspond with Joseph and make changes to the WP as needed w/r/t SC22WG14.12207.

5.28 SC22WG14.12205, Anonymous Structures (N1425)

In SC22WG15.12205, Joseph Myers poses the following:

> The definition of anonymous structures and unions in N1425 appears to
> allow cases such as:
>
> ```
> typedef struct
> {
> ```

```
    int i;
} s0;

struct s1
{
    s0;
};
```

where the anonymous structure member is defined using a typedef for a structure with no tag (but not when it is defined using a typedef for a structure with a tag). Is this as intended? It is supported by some existing implementations, but not all (for example, GCC requires –fms -extensions to enable this case).

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.". Does this allow structures where all previous members are anonymous structures or unions?

The argument that it does is that "The members of an anonymous struct or union are considered to be members of the containing structure or union." (paragraph 13), so that in such a case it would be considered, by virtue of that sentence, to have more than one named member after all. The argument that it does not is that paragraph 8 says "If the struct-declaration-list contains no named members, no anonymous structures, and no anonymous unions, the behavior is undefined.", and this reasoning would make the words "no anonymous structures, and no anonymous unions" redundant in paragraph 8 (any such structure or union would either have contained, recursively, a named member, or resulted in undefined behavior itself). My inclination is that those words are indeed formally redundant, but helpful, and similar words should be added to paragraph 18 to clarify it. That is, examples such as the following are allowed.

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

It might of course be helpful for examples in C1X to cover these cases explicitly.

Boulder Discussion:
David: He's right, but it's not clear how to fix it.

ACTION: David K to work with Joseph Myers and write a proposal w/r/t SC22WG14.12205, Antonymous Structures.

5.29 SC22WG14.12250, Interchangeable v. Aliasing (Nelson)

In response to an action item in Santa Cruz regarding aliasing rules, Clark asks whether or not it makes sense to unify, or at least overlap, the concepts of interchangeable types and aliasing, or to leave them as is.   Interchangeable types refer to types that have the same representation and alignment requirements.

Boulder Discussion:
Clark: Compiler optimization does not always keep track of type distinctions. Thus, the alias rules are not implemented exactly as specified in the Standard.
David: Loosening the rules would create problems that goes beyond compilers.
Tom: The text cited is a non-normative note. (Understood). Is the text a help, or a hindrance. Not sympathetic to changing the rules.
Clark: Is anybody interested in bringing the rules together. No response.
Tom: The footnote is probably confusing. Maybe it should be scrapped.

No Action taken.

5.30 Additional Atomics Errata (N1484) (N1477) (McKenney)

N1477 lists errata for the specification of atomics in the current WP. Much of this is editorial, and was be handed off to the Editor, Larry Jones.

N1484 is a revision to N1477 containing proposed wording. What does it mean to initialize a union containing an atomic? That's well defined behavior because nothing is said about not allowing atomics? Does this code have to work in C++0X? Unknown. We don't have time to construct an answer to this, but we don't want to forget about it either. The first part is on hold (6.7.3p2).

5.31 intN_t and intN_t atomics (Thomas)

The paper proposes to add atomic versions of the types intN_t, as addressed in N1482. Note that the C++ FCD does not list atomic versions for these types. The implications of this proposal may be more far reaching than we realize.   Where does this stop? Are we drifting away from C++? C++ accomplishes this via static member functions, which C does not have. Clark believes we need to liaise with C++ to solve this problem. For this meeting, the most productive thing to do is probably nothing. Defer for now.

5.32 Comparison Macro Argument Conversion (N1487) (N1481) (Thomas)

This note follows up on N1472 and the committee decision to leave unspecified whether widened arguments to comparison macros are narrowed to their semantic type.
A primary goal of Annex F is predictable floating-point behavior for conforming IEEE implementations that use IEEE features and standard evaluation methods. Leaving the evaluation of the IEEE comparison macro arguments unspecified is inconsistent with this goal.
An alternative would be to specify the argument behavior only for Annex F implementation. Not narrowing widened arguments is the most natural interpretation of the C99 specification and consistent with the macros' being stand-ins for comparison operators, so is preferred. This argues to

1. Disallow narrowing widened arguments (though only for Annex F implementations)

There are significant implementations that narrow widened arguments, and ones that do not. If accommodating implementations is deemed necessary, then a fallback might be

2. Make the behavior implementation defined and deprecate narrowing widened arguments (only for Annex F implementation)

If the committee chooses to simply leave the argument conversion behavior unspecified, then the burden is on the user to make the macros predictable. Since predictability is a key goal of Annex F, an example or footnote should be provided - in Annex F - in order to alert the user to the problem and show how it might be avoided.

Discussion:
David: Option 1 is in the spirit of what we've done in the past.
Fred: Option 1 is also in the spirit of the original 754.
PJ: How to do this in the library ? Can't control the narrowing and the widening.
Fred: A conforming solution would be just to provide the long double version.
PJ: OK

Straw Poll: Adopt something along the lines of N1481 Option 1 into C1X
6, 0, 6
Decision: Adopt something along the lines of N1481, Option 1, into C1X.

Jim has proposed the following text to satisfy the above.

> In F.9 insert a subclause before current F.9.1:
>
> F.9.1 Comparison macros
>
> Inequality operators and their corresponding comparison macros (7.12.3) produce equivalent result values, even if argument values are represented in wider formats. Thus, comparison macro arguments represented in formats wider than their semantic types are not converted to the semantic types, unless the wide evaluation method converts operands of inequality operators to their semantic types. The standard wide evaluation methods, characterized by FLT_EVAL_METHOD equals 1 or 2 (5.2.4.2.2), do not convert operands of inequality operators to their semantic types.
>
> Rationale: 7.12.14 does not specify whether widened arguments of comparison macros are narrowed to their semantic types. F.9.1 specifies this behavior, in keeping with the Annex F goal of predictable floating-point behavior. Not narrowing widened arguments is consistent with the macros' being "quiet" versions of the comparison operators. For the standard wide-evaluation methods and most others, converting arguments to long double would enable portable implementation.

The above text is moved into N1487.

F.9 should be F.10, math.h. The 'Comparison Macro' clause would be F.10.11.

Straw Poll: Adopt N1487 into the C1X WP, modulo the physical placement called out in that paper.
9, 0, 3 – consensus
Decision: Adopt N1487 into the C1X WP, modulo the physical placement called out in that paper.

5.33 UTF-8 String Literals (N1488) (Nelson)

Proposal for UTF-8 string literals. Tom: When we pulled out raw string literals, due to waiting for WG21 to decide what to do, we did not submit UTF-8. This is compatible with C++. Tom likes

what Clark has done. Adding these is OK. Adding library conversions is not. Tom: We need help from the compiler, C++ has already done it, we should do likewise.
The document is N1488.

Straw Poll: Adopt N1488 into the C1X WP.
11, 0, 2
Decision: Adopt N1488 into the C1X WP.

5.34 DR 269 Fix (Tydeman)

Proposes a fix for dr269 which was deferred for consideration in a future revision. Larry: This is already required, but it takes a while to figure it out. Clark: Where does it say that it can't have padding bits. Larry: In the prior paragraph. Clark: OK. Paragraph I and 2 should be as parallel as possible. Editorial change to WP

5.35 DR 271 Fix (Tydeman)

Proposal by Clive to add defined behavior when desc is zero for either iswctype() of towctrans(). This is a tweek. Make it a N doc – N1491.

5.36 DR 319 Fix (Tydeman)

Do we want to make a rule where existing implementations to different things? Give it a doc number – N1492.

6. OTHER BUSINESS

6.1 C1X Schedule

What do we want to do with our schedule?

Straw Poll: Vote a CD out at this meeting.
This will require a very active editorial communication process to get the document ready
3, 8, 2 – no consensus to vote out a CD now.

Default: No CD until Batavia, the earliest.
We still need to work the editorial issues between now and Batavia.

1st meeting: 16 July

2nd meeting: 16 Aug

Logistics are TBD. Hopefully video and audio. Silicon Valley types can meet together.

Who: Tom Plum, Jim , Paul, David, Douglas, Blain, Larry, JB

If we get started quickly out of Batavia, we could start by the end of November 2010, meet in April 2011 for a Ballot Resolution Meeting.

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.

N1439 – Completeness of Types.
N1441 – General Support for Type Generic Macros.
N1444 – Dependency Ordering in the C Memory Model.
N1447 - C and C++ Alignment Compatibility.
N1459 – Comparison Macros, Proposal #2 for C (ed – the types need not be the same).
N1460 – Subsetting the Standard.
N1462 – Errno and Threads.
N1464 – Creation of Complex Type, modulo making static initialization a Recommended Practice v. being required.
N1468 - Assumed types in F.8.2, Expression Transformations, modulo editorial, along the lines of.
N1472 – Comparison Macro Conversion, Proposal #3.
N1478 – Supporting the noreturn Property.
N1480 - Updates to C++ Memory Model Based on Formalization.
N1482 – Explicit Initializers for Atomics.
N1485 – Atomics, modulo edits to remove item #6, ternary operator.
N1486 – fabs, adopt edit to F.3 only
N1487 – Comparison Macro Argument Conversion, modulo the physical placement (ed).
N1488 – UTF-8, String Literals

7.2 Review of Action Items

Carry Over
ACTION: Larry Jones: Review use of headers required for the function declaration shown in the synopsis, and any others required to make the declaration valid.
OPEN

ACTION: Clark will ask Hans Boehm to add an explanation of memory sequencing to the rational re: N1411, Memory Model Rationale.

ACTION: Clark to work on a clearer formulation of what the rules are w/r/t N1409, Aliasing and Effective Type.
OPEN

ACTION: Convener to discuss the resolution of N1448 with the Submitter, Steve Adamczyk.
OPEN

New

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

ACTION: Larry Jones to correspond with Joseph and make changes to the WP as needed w/r/t SC22WG14.12207.

ACTION: Convener to discuss N1471 in light if the new work we've done this week for the C1X with Steve Montgomery / MISRA.

7. THANKS TO HOST

The Committee expresses its great appreciation to David Keaton for hosting this meeting in Boulder, CO

Thanks also to Dinkumware for providing the Wiki.

8. ADJOURNMENT

Meeting adjourned at 1640, 27 May, 2010

PL22.11 Meeting
25-27 May, 2010

Meeting convened at 0900, 25 May, 2010 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
Tana L. Plauger	Dinkumware, Ltd – A	
P. J. Plauger	Dinkumware, Ltd – P	
Jim Thomas	HP – P	
Hans Boehm	HP - A	
Michael Wong	IBM - A	
Rajan Bhatka	IBM - P	
Paul Mc Kenney	IBM – A	
Clark Nelson	Intel – A	
Nat Hillary	LDRA – P	
Herb Sutter	Microsoft – P	
Douglas Walls	Oracle - P	PL22.11 IR
Barry Hedquist	Perennial – P	PL22.11 Secretary
Tom Plum	Plum Hall – P	
Fred Tydeman	Tydeman Consulting – P	PL22.11 Vice Chair
Larry Wagoner	NSA - P	
Prospective Members		
Larry Jones	Siemens PLM Software	WG 14 Project Editor

Agenda Approved as modified.

1. **INCITS Anti-Trust and Patent Guidelines**
   We viewed the slides located on the INCITS web site.

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

   http://www.incits.org/pat_slides.pdf

2. **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.

3. **Identification of PL22.11 Voting Members (Tydeman)**
   See attendance list above.
   13 PL22.11 voting members participated out of 15.

3.1 PL22.11 Members Attaining Voting Rights at this Meeting
Larry Wagoner, NSA.

3.2 Prospective PL22.11 Attending Their First Meeting
None

4. Members in Jeopardy due to Failure to return Letter Ballots.
Apple, HP, CERT

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

6. **Selection of US Delegation for 2010 / 2011 meetings**
Motion: The US delegation for WG14 meeting for the remainder of 2010 and 2011 will be all members and alternates for PL22.11 with the exception of JTC1 Officers. (Walls, Keaton)
(13, 0, 0, 2, 15)
Motion Passes.

6. Adjournment
Adjourned at 1640 local, 27 May 2010