ISO/IEC JTC1/SC22
Languages
Secretariat: CANADA (SCC)

ISO/IEC JTC1/SC22
N 1069

DECEMBER 1991

TITLE: Summary of Voting and Comments Received on CD 11404: Information Technology - Programming Languages - Common Language-Independent Datatypes

SOURCE: Secretariat ISO/IEC JTC1/SC22

WORK ITEM: JTC1.22.17

STATUS: New

CROSS REFERENCE: N970

DOCUMENT TYPE: Summary of Voting

ACTION: For information to SC22 Member Bodies. See attached.

Address reply to: ISO/IEC JTC1/SC22 Secretariat
J.L. Côté
Treasury Board Secretariat
140 O'Connor St., 10th Floor, Ottawa, Ontario, Canada, K1A 0R5
Tel.: (613)957-2496 Telex: 053-3336 Fax: (613)996-2690

SUMMARY OF VOTING ON:

Letter Ballot Reference No: SC22 N970
Circulated by :JTC1/SC22
Circulation Date :1991-06-12
Closing Date :1991-09-27, ext to 91-11-29

SUBJECT: CD 11404: Information Technology - Programming Languages - Common Language-Independent Datatypes

---

The following responses have been received:

'P' Members supporting the CD, without comments : 05

'P' Members supporting the CD, with comments : 01

'P' Members not supporting the CD : 05

'P' Members abstaining : 00

'P' Members not voting : 09

Secretariat Action:

The Comments received will be forwarded to WG11 for review and recommendation for further processing of CD 11404.

ISO/IEC JTC1/SC22 LETTER BALLOT SUMMARY

PROJECT NO: JTC1.22.17

SUBJECT: CD 11404: Information Technology - Programming Languages - Common Language-Independent Datatypes

Reference Document No: N970
Circulation Date: 1991-06-12

Ballot Document No: N970
Closing Date: 1991-09-27
extd to 1991-11-29

Circulated To: SC22 P,O,L

Circulated By: Secretariat

SUMMARY OF VOTING AND COMMENTS RECEIVED

	Approve	Disapprove	Abstain	Comments	Not Voting
'P' Members					
Belgium	( )	( )	( )	( )	(✓)
Brazil	(x)	( )	( )	( )	( )
Canada	( )	(x)	( )	(x)	( )
China	( )	( )	( )	( )	(✓)
Czechoslovakia	(x)	( )	( )	( )	( )
Denmark	(x)	( )	( )	( )	( )
Finland	( )	( )	( )	( )	(✓)
France	( )	( )	( )	( )	(✓)
Germany	( )	(x)	( )	(x)	( )
Iran	( )	( )	( )	( )	(✓)
Italy	(x)	( )	( )	( )	( )
Japan	( )	(x)	( )	(x)	( )
Netherlands	( )	( )	( )	( )	(✓)
New Zealand	(x)	( )	( )	(x)	( )
Romania	( )	( )	( )	( )	(✓)
Sweden	( )	( )	( )	( )	(✓)
Switzerland	( )	( )	( )	( )	(✓)
UK	( )	(x)	( )	(x)	( )
USA	( )	(x)	( )	(x)	( )
USSR	(x)	( )	( )	( )	( )
'O' Members					
Argentina	( )	( )	( )	( )	( )
Austria	( )	( )	( )	( )	( )
Australia	( )	( )	( )	( )	( )
Cuba	( )	( )	( )	( )	( )
Hungary	( )	( )	( )	( )	( )
Icelan	( )	( )	( )	( )	( )
India	( )	( )	( )	( )	( )
Korea	( )	( )	( )	( )	( )
Poland	( )	( )	( )	( )	( )
Portugal	( )	( )	( )	( )	( )
Singapore	( )	( )	( )	( )	( )
Turkey	( )	( )	( )	( )	( )
Thailand	( )	( )	( )	( )	( )
Yugoslavia	( )	( )	( )	( )	( )

VOTE ON COMMITTEE DRAFT	11404
Date of circulation 1991-06-12	Reference number
Closing date for voting 1991-09-27	ISO/~~IEC~~ JTC1 /SC 22 N 970

attachment 1
to SC22 N1069

ISO/~~IEC~~ JTC 1 /SC 22 Title Languages Secretariat Canada, SCC	Circulated to P-members of the committee for voting on registration of the draft as a DIS, in accordance with 2.4.3 of part 1 of the IEC/ISO Directives

Please send this form, duly completed, to the secretariat indicated above.

CD 11404

Title Information Technology - Programming Languages - Common Language - Independent Datatypes

☐ **We agree to the circulation of the draft as a DIS in accordance with 2.5.1 of part 1 of the IEC/ISO Directives**

☒ **We do not agree to the circulation of the draft as a DIS**
The reasons for our disagreement are the following (use a separate page as annex, if necessary)

P-member voting CANADA	
Date 91 09 19	Signature D. LANGLOIS

FORM 8 (ISO)

CANADIAN COMMENTS ON DOCUMENT NO. JTC1/SC22 CD11404 N970

TITLE: Information Technology - Programming Languages - Common Language - Independent Datatypes

- -

Canada does not approve the above document for the following reasons:

The Canadian National Body is ready to reverse its vote if its suggestions are accepted. The essential points are indicated in underlined characters in the following text.

General considerations

Canada strongly believes there should be an abstract data type for adequate processing of alphanumeric strings (extendable to ideographic, if structures are adequately defined).

So far characters are processed with the intrinsic assumption that their ordering is code-dependent in practice. It is known that sorts may do special processing for acceptable orders to be produced, but other programs, processes (hardware and software) are totally unable to deal consistently with these orders, which have to be processed by all aplications. Furthermore, search-oriented operations on character strings are painful. This pain is also tightly linked to the lack of consistency of ordering and character-oriented operations.

Canada has designed an Ordering Standard (Preliminary Standard CAN/CSA Z243.4.1) applicable to all comparison operations. It is possible to structure alphabetically data in such a way that it will be processable directly by machines (for searching and ordering - in other words for flexible data comparisons) and produce adequate cultural results. In order to relieve the applications from the burden of reinventing the wheel, Canada believes that the above-mentioned structures shall be defined in an abstract manner and that languages implement these structures accordingly when the requirement exists for a given language.

POSIX already has provisions for processing these structures for ordering and comparisons via different mechanisms but there is a lack of formal definitions for them in all contexts and particularly for programming languages.

The following discussion also addresses directly Issue 3 presented in Outstanding Issues that precede the Foreword of the CD.

Specific comments

Section 7.1.4 presents operations of the Character Type. If it stays as it is, the result will be the perpetuating of a catastrophe that still plagues modern data processing. Word processing generally knows how to search or compare character strings (although not consistently from application to application), but data processing has failed so far to be able to consistently compare alphabetic data, because of a lack of proper definitions for string comparisons in programming languages, which extend to technology at other levels (data bases, sort and merge programs, record indexing mechanisms, hardware disk searches and so on).

For example, if what the CD says is true, compound family names, written in different legal ways, will never be ordered the right way:

- "La Bonté" (with one space) would most of the time be ordered before "Labadie" because character SPACE comes in general before SMALL LATIN LETTER B in most character-sets given by the «repertoire» identifiers. This is incorrect for all Westerners, whatever their culture. There is very limited need for present naive ordering but the result is not guaranteed to be consistent across coding schemes, and it certainly leads to numerous practical mistakes in the real world even when used by skilled programmers (it is not because you don't find "La Bonté" in a list that it is not there, it might be because you did not think about the not-so-naively-interpretable result of the "naive" sort). And alphabetical data can not be positionally ordered, which is the case in the above example in naive ordering character by character. The CHARACTER data type is defined with such a positional ordering implied.

Moreover, the Canadian Preliminary Standard Z243.4.1 on Character Ordering could not be implemented system-wise with this kind of primitive datatype. No other actual culturally valid ordering scheme (without talking about telephone book data base management which is more application-dependent because order may vary and be multiple for a given record, necessitating record duplication and preprocessing any way, so that it is out of the question here) could work with such a so-called ordered datatype as presented in the CD. Canada requires the present CHARACTER type shall be declared as intrinsically unordered.

As required in SHARE Europe National Language Architecture White Paper, operations required on alphabetical strings do not limit to Equal, InOrder or Successor, but processibility can be more complex, including EqualIfExact ("CÔTÉ"="CÔTÉ"), EqualExceptSpecials ("Mc Arthur"<=>"McArthur"), EqualExceptCase ("CÔTÉ"<<=>>"côté") and RoughlyEqual ("côté"<<<=>>>"COTE"). SHARE Europe went farther by talking about fuzzy equality but the latter is, in this case only, application-dependent (for cases including telephone book making and beyond) and is not an immediate requirement here.

precited Canadian standard, by the intermediary of its expert on internationalisation, Mr. Alain LaBonté, nominated to WG20 as Canadian representative).

Properties remained to be exactly defined in a formal way. Operations are described above (see SHARE Europe Requirements)

Properties of such a data type are similar to 3-part floating point numbers and allows the operations presented before as a requirement: if the mantissa is truncated a floating point number is still processable with less precision; if only the exponent remains you can still work on its order of magnitude; if only the sign bit remains, you can still tell if it is negative or positive; here it is the same thing: each part defines a given precision level, from the highest (pure alpha) to the lowest (specials). Canada requires that such a flexible data type for adequate processing of character strings be defined, in the same fashion that the type REAL allows easy internal representation and storage of numbers in a directy processable form. .

It is to be noted that such a datatype would be consistent in a given culture and be totally independent of existing codings if well implemented. Conversion from/to Standard Character Coding would be necessary for exchange with culturally-different environments only out of process time, for capturing or outputing data. Then instead of a repertoire-identifier it needs a culture-identifier. Fortunately it is always possible to transform a PROCESSABLE STRUCTURED ALPHABETIC to a CHARACTER string because all information is preserved at its richest precision; if precision is less (if only the 1st element of the structure is preserved for example), the essence of the string is maintained and still transformable to CHARACTER.

Finally it could be extended to non-alphabetic scripts: Here it is called alphabetic because it definitely applies to all alphabetic scripts (including Arabic, which could have another analog concept for case, given the 4 forms of each character). Other elements (that is, more than 4) could exist in some alphabetic scripts (the number would be dependent on the culture and be implicit for this given culture).

It is foreseen that the structure could easily be applied to ideographic scripts, particularly with the now recognized international requirement of Han unification by Japan, Korea and China: part 1 for normalized character (taking into account traditional, simplified or variant characters), part 2 for determining the exact character in case of quasi-homographs in comparisons, eventually a part 3 (to be determined if a need exists for eventual extra finer discriminants) and part 4 for the special characters imbedded in the text.

Finally, Canada believes that additionally to the BIT datatype be defined a BIT STRING Datatype. Many systems level parameters require the use of a bit mask or a bit string (more than 1 bit at a time). This could be addressed by use of integer or character string datatypes, however consideration of a BIT STRING datatype would be worthwhile.

The CHARACTER type as presently defined in the CD is useful for transporting character data across cultures or for presentation or input (not process) purposes, as long as the coding is identified. But another data type is needed to allow consistency of comparison operations on character strings in the real world. Such a new data type would easily allow for the operations described in the previous paragraph, if it were structured like this:

[-1]. Number of elements in the structure (here 4 for Western languages but this number may vary in Eastern languages)
[0]. Culture/Language Ordering identifier (for interpretation and transport)
1. Pure alphabetic Base
2. Diacritic Info
3. Case Info
4. Special character Info

Hence a string like "La Bonté" would be actually structured as follows (the following could be implemented in different fashions but each part would bear the same abstract information):

[-1]. 4 elements in the structure
[0]. French (based on Ordering Standard CAN/CSA Z243.4.1)
1. labonte
2. <acute><nd><nd><nd><nd><nd><nd>
3. <up><low><up><low><low><low><low>
4. <Logical Zero> <Position 3><SPACE>

A discussion on the previous structure and how, once concatenated (or processed in different phases in other implementations), the resulting 4-part string can be used system-wise for any type of comparison (by the way processable by any engine able to do dumb binary comparisons) is given in Canadian Preliminary Standard CSA Z243.4.1 (to be noted here that reverse discrimination on accents in part 2 of the example follows the French dictionaries practice and is compatible with English, which does not deal with accents in such a precise way: we have designed the Canadian Standard to allow a maximum of Western languages at once: French, English, German, Dutch, Italian, Portuguese and so on [Spanish and Scandinavian will need adaptation of tables but will use the same structure].

This new data type will need a name: Canada suggests PROCESSABLE STRUCTURED ALPHABETIC. It would be described (We took as model: Date-and-Time): Processable Structured Alphabetic is a family of datatypes whose values are parsed elements of alphabetic data in various resolutions. It is independent of the input-ouput coding of alphabetic data but identifies the culture assumed for processing.

The Canadian National Body is not sure about the exact description. and syntax it would take in the CD for consistency with the rest of the document but see no particular difficulty at first glance. We leave it to the experts that are more familiar with the exact terms of it (Canada is offering to help in making understood this new but field-proven concept explained in the

VOTE ON COMMITTEE DRAFT 11404	
Date of circulation 1991-06-12	Reference number ISO/~~IEC~~ JTC1 /SC 22 N 970
Closing date for voting 1991-09-27	

attachment 2
to SC22 N1069

ISO/~~IEC~~ JTC 1 /SC 22
Title
Languages

Secretariat Canada, SCC

Circulated to P-members of the committee for voting on registration of the draft as a DIS, in accordance with 2.4.3 of part 1 of the IEC/ISO Directives

Please send this form, duly completed, to the secretariat indicated above.

CD 11404

Title Information Technology - Programming Languages - Common Language - Independent Datatypes

☐ We agree to the circulation of the draft as a DIS in accordance with 2.5.1 of part 1 of the IEC/ISO Directives

☒ ~~XX~~ We do not agree to the circulation of the draft as a DIS
The reasons for our disagreement are the following (use a separate page as annex, if necessary)

We think that the Committe Draft 11 404 needs further discussion within WG 11. At least the eleven outstanding issues should be clarified before circulating a DIS.

P-member voting
DIN Germany

Date
Berlin, 1991-09-19

Signature M. Kutschke

FORM 8 (ISO)

11 89 3 09

VOTE ON COMMITTEE DRAFT 11404	
Date of circulation 1991-06-12	Reference number ISO/~~IEC~~ JTC1 /SC 22 N 970
Closing date for voting 1991-09-27	

attachment 3
to SC22N1069

ISO/~~IEC~~ JTC 1 / SC 22
Title
Languages

Secretariat Canada, SCC

Circulated to P-members of the committee for voting on registration of the draft as a DIS, in accordance with 2.4.3 of part 1 of the IEC/ISO Directives

Please send this form, duly completed, to the secretariat indicated above.

CD 11404

Title Information Technology - Programming Languages - Common Language - Independent Datatypes

☐ We agree to the circulation of the draft as a DIS in accordance with 2.5.1 of part 1 of the IEC/ISO Directives

☒ We do not agree to the circulation of the draft as a DIS
The reasons for our disagreement are the following (use a separate page as annex, if necessary)

Reason for Japanese Negative Vote

Many issues listed in the CD shall be resolved before circulating it as a DIS.

P-member voting
Japan

Date : 1991-09-06

Signature Ikuo Nakata
Ikuo Nakata

FORM 8 (ISO)

90 E 89 11

VOTE ON COMMITTEE DRAFT 11404	
Date of circulation	Reference number
Closing date for voting 1990-09-27	ISO/IEC JTC 1 /SC 22 N970

attachment 4 to SC22 N1069

P-members have an obligation to vote

ISO/IEC JTC 1 /SC 22 Title Common Language-Independent Datatypes Secretariat SCC	Circulated to P-members of the committee for voting

Please send this form, duly completed, to the secretariat indicated above.

CD 11404

Title **Common Language-Independent Datatypes**

Please put a cross in the appropriate box(es)

Approval of the draft:

☒ as presented

☒ with comments as given below (use separate page as annex, if necessary)

- ☐ general
- ☐ technical
- ☐ editorial

☐ **Disapproval of the draft for reasons below (use separate page as annex, if necessary)**

- ☐ Acceptance of these reasons and appropriate changes in the text will change our vote to approval

☐ **Abstention** (for reasons below)

This vote of approval is one of 'approval in principle'. The current work being undertaken by WG 11 on Common Language-Independent Procedure Calling Mechanisms (Work Item JTC1.22.16) is still at an early stage and it is therefore not possible to assess if there will be any significant interaction between the two standards.

The current work on bindings covered under Work Item JTC1.22.21 is also relevant and New Zealand would welcome the addition of an informative Annex indicating the drafter's opinions and attitude towards potential interactions between those work items before final voting takes place.

P-member voting Standards Association of New Zealand

Date 1991-09-11

Signature D. L. Andrews
International Coordinator
FOR. K. Hopper.

FORM 8 (ISO/IEC)

1 E 90 02 5 000

VOTE ON COMMITTEE DRAFT	11404
Date of circulation: 1991-06-12	Reference number: ISO/IEC JTC1 /SC 22 N 970
Closing date for voting: 1991-09-27	

Attachment 5 to SC22 N1069

ISO/IEC JTC 1 / ISC 22

Title: Languages

Secretariat: Canada, SCC

Circulated to P-members of the committee for voting on registration of the draft as a DIS, in accordance with 2.4.3 of part 1 of the IEC/ISO Directives

Please send this form, duly completed, to the secretariat indicated above.

CD 11404

Title: Information Technology - Programming Languages - Common Language - Independent Datatypes

☐ We agree to the circulation of the draft as a DIS in accordance with 2.5.1 of part 1 of the IEC/ISO Directives

☒ We do not agree to the circulation of the draft as a DIS
The reasons for our disagreement are the following (use a separate page as annex, if necessary)

The intention of WG 11 was to circulate this CD for comment only, because further work is necessary on the remaining open issues, before proceeding to DIS.

P-member voting: UNITED KINGDOM

Date: 12 September 1991 Signature: J.A. Tate

FORM 8 (ISO)

attachment 6 to
SC22 N1069

VOTE ON COMMITTEE DRAFT	11404
Date of circulation 1991-06-12	Reference number ISO/~~IEC~~ JTC1 /SC 22 N 970
Closing date for voting ~~1991-09-27~~ 1991-11-24	

ISO/~~IEC~~ JTC 1 / SC 22 Title Languages Secretariat Canada, SCC	Circulated to P-members of the committee for voting on registration of the draft as a DIS, in accordance with 2.4.3 of part 1 of the IEC/ISO Directives

Please send this form, duly completed, to the secretariat indicated above.

CD 11404

Title Information Technology - Programming Languages - Common Language - Independent Datatypes

☐ We agree to the circulation of the draft as a DIS in accordance with 2.5.1 of part 1 of the IEC/ISO Directives

☒ XX We do not agree to the circulation of the draft as a DIS
The reasons for our disagreement are the following (use a separate page as annex, if necessary)

See attached comments

P-member voting USA (ANSI)	Signature Stacy M. Leistner
Date November 21, 1991	

FORM 8 (ISO)

590 E 89 11

U.S. Comments on CD 11404: Language-Independent Datatypes

Item: 1
Title: Make CLID a Reference Model
Rating: Major
Clause: 1, 5, 10.1, Annex A
Rationale:

The relationship between the CLI datatypes and CLIPC and RPC has led to the inclusion of a "minimum datatype list" in the current draft. This minimum list makes no sense for programming language conformance, as France has pointed out, nor for the PCTE, and may not make sense for unforeseen uses of the CLID. This is largely because CLID is intended as a reference model and what "information processing entities" may be expected to comply is open-ended. Accordingly, the concept of "how an information processing entity may comply directly" is not well-defined.

The question of which datatypes must be supported by CLIPC/RPC should be left to those standards (jointly).

Proposed Change:

1. Modify clause 1, first sentence, to read:
"This International Standard defines a reference model for datatypes, specifying the nomenclature and shared semantics ..."
2. Move the second, third and fourth paragraphs of Clause 1 to Clause 6.
3. In 5.1, strike paragraphs 2 and 3, defining total and partial direct compliance. (Direct compliance means "uses the model and its syntax consistently with the CLID".)
4. In 5.1, modify bullet ii) to make clear that all additional types defined by an information processing entity must be either type-declarations or new types defined by type-templates. Add a caveat Note to the effect that types not defined by type-declarations will not be useable in the CLIPC/RPC environment.
4. In 5.2, 2nd paragraph, strike "total" in all occurrences, and strike "and outward" in the first line. In the last line, strike everything after "an inward mapping".
5. In 5.2, strike the 3rd paragraph.
6. In 10.1, strike bullet iii)
7. Delete Annex A.

Item: 2
Title: Relate type of compliance to type of conforming entity
Rating: Major
Clause: 5
Rationale:

It is unclear what kinds of information processing entities are expected to have direct or indirect compliance.

For example, are programming languages expected to be in indirect compliance? (It seems that all of datatypes to be used in the CLIPC/RPC must have a mapping to every language. Is that the inward mapping or the outward mapping or both?)

Similarly, is the POSIX language-independent specification supposed to comply directly or indirectly?

Does "mapping standard" mean a standard containing the mapping from a given language to/from the CLID? Or is it something else?

The proposed change for this item assumes that the removal of total and partial compliance was accepted in Item 1.

Proposed Change:

Add the following text to the end of the note in Clause 5. "Programming language standards are expected to comply indirectly through a mapping standard for that language which directly complies to the CLIDT. Language independent standards could comply directly themselves or indirectly via a mapping standard."

Add to Clause 5.3 after the first sentence. "Programming language mapping standards shall require both an inward and outward mapping to the CLIDT. All other mapping standards shall have at least an outward mapping to the CLIDT."

Item: 3
Title: Add late-binding concept
Rating: Major
Clause: 6, 7 (many places), 8
Rationale:

In procedure calls and record types, because CLID regards bounds, sizes, and other such parameters as part of the type identification, the types of some arguments or fields may be dependent on the values of other arguments or fields. This feature is referred to in programming languages as "late binding". It is necessary to numerous mathematical procedure definitions, among others. The CLID does not currently provide for this feature.

Proposed Change:

1. Change the production for value-expression in Clause 8 to include "late-binding".

2. Remove the asterisk in 7.3.7, i.e. change the productions for upperbound and lowerbound to read:

```
upperbound = value-expression .
lowerbound = value-expression .
```

This requires either a corresponding change in the syntax for select-range in clause 7.2 (see comment 12 below) or a renaming of the syntactic objects lowerbound and upperbound in 7.3.7.

3. Add the concept "late binding" to Clause 6. The following draft wording should be considered:

6.x.x Late-Bindings

A late-binding identifies a value which is the value of another component of some generated-datatype in which the late-binding occurs. Syntax:

```
late-binding = late-binding-primary { "." component-reference } .
late-binding-primary = field-identifier | argument-name .
component-reference = field-identifier | "*" .
```

A datatype-designator x is said to "involve" a particular instance of a late-binding if x contains the instance and there is no component datatype y of x which contains the instance. Thus, exactly one

datatype-designator "involves" a given instance of a late-binding. Any datatype-designator which involves a late-binding shall identify a component of some generated-datatype.

The late-binding-primary shall identify a (different) component of some generated-datatype which contains the datatype which involves the late-binding. The component so identified is said to be the "primary component", and the generated datatype of which it is a component is said to be the "primary datatype". The primary datatype shall be either a procedure-type or a record-type. When the primary datatype is a procedure-datatype, the late-binding-primary shall be an argument-name and shall identify an argument of the primary datatype. When the primary datatype is a record-type, the late-binding-primary shall be a field-identifier and shall identify a field of the primary datatype.

When the late-binding contains no component-references, the value of the late-binding shall be the value of the primary component. Otherwise, the "current reference" shall comprise the late-binding-primary, and identify the primary component and its value, and the following paragraph shall be applied (recursively) to determine the value of the late-binding.

The datatype of the current reference shall be a record-type, a choice-type, or a pointer-type. If the datatype of the current reference is a record-type or a choice-type, then the next component-reference shall be a field-identifier and shall identify a field of the current reference. In this case, a new reference shall comprise the current reference syntax, plus the next component-reference, and its value shall be the value of that field of the current reference which is identified by the next component-reference. If the current reference is a pointer-type, the next component-reference shall be an asterisk. In this case, a new reference shall comprise the current reference syntax, plus the asterisk, and its value shall be that obtained by dereferencing the pointer-value of the current reference. In either case, when the late-binding contains no further component-references, the value of the late-binding shall be the value of the new reference. Otherwise, the new reference shall become the current reference and the rules of this paragraph shall be applied thereto.

NOTES:

1. The datatype which involves a late-binding must be a component of some generated-datatype, but that generated-datatype may itself be a component of another generated-datatype, and so on. The primary datatype may be several levels up this hierarchy.

2. The primary component, and thus the primary datatype, cannot be ambiguous, even when the late-binding-primary identifier appears more than once in such a hierarchy. By the scope rules, the meaning of the identifier X in a datatype-designator (y) which involves X as a late-binding-primary is the meaning of X as declared in y, or else the meaning of X in the datatype-designator which immediately contains y. So, in following the hierarchy upward, there is a "most recent declaration" of X, and that is the one to which the late-binding refers.

3. In the same wise, an identifier which may be either a value-identifier or a late-binding can be resolved by application of the same scope rules. If the identifier X is found to have a "declaration" anywhere within the outermost datatype-designator which contains y, then that declaration is used. If no such declaration is found, then a declaration of X in a "global" context, e.g. as a value-identifier, applies.

Item: 4
Title: Resolve Null/undefined or discard them
Rating: Major
Clause: 7.1.13 and 7.1.14
Rationale:

There is NO consensus on the usefulness or meaning of Null and Undefined. What currently appears in the draft encountered objections from the U.S. and France on the CD-registration ballot which were not resolved.

This issue must be resolved before the CD becomes a DIS.

The US believes it is possible that an "absent" value of every datatype is isomorphic to the CLIDT WD5 Null datatype.

Undefined, unknown, and other variants of Null with the possible exception of "absent" are implementation/representation concerns. As such they can be treated as distinguished values of the state type using choice. The fundamental property distinguishing "absent" is that of any item it is present or not. The other interpretations of Null do not appear to have this universality.

Proposed Change:

Remove clause 7.1.14 Undefined

In clause 7.1.13, chance the title of this clause to "Void".
In the "Syntax", delete "null-literal = "nil"".
In the "Values", replace the text by "none".
In the "Properties", replace the text by "none".
In the "Operations", replace the text by "none".
Replace the text of Note(1) in clause 7.1.13 by the following.
"Datatype Null is used for example as the type of a procedures arguments which has no arguments. The alternative of a choice when the choice has no values."

Item: 5
Title: Revise Choice to relate alternatives to tag values
Rating: Major
Clause: 7.3.1
Rationale:

The Choice datatype as currently provided does not support relationship between the alternative "fields" and values of any tag-type. This version of Choice, therefore, does not support the Pascal or Ada concept of variant-record very well. Moreover, it makes the handling of a Choice datatype in an interchange very difficult: How does the "caller" or "source" identify to the "service" (e.g. RPC or CLIPC) which alternative applies to the current value? How is a "tag" associated with a "field-name"?

The Ada/Pascal model solves these problems by associating the alternatives with values of some datatype. This model should be used in the CLID.

Proposed Change:

1. In 7.3.1, revise the syntax to read:

```
choice-type = "choice" "(" selection-type [ "=" discriminant ] ")"
                    "of" "(" alternative-list ")"  .
selection-type = datatype  .
discriminant = value-expression  .
alternative-list = alternative { "," alternative }  .
```

```
alternative = selection-values ":" alternative-type  .
selection-values = select-list | "*"  .
alternative-type = datatype  .
```

2. In 7.3.1, replace the second sentence under Components with: "Each datatype is labelled by a list of selection-values drawn from the value space of the selection-type." The selection-type shall be any discrete datatype.

And replace the second paragraph under components with the text of the Components paragraph from 7.2.2, and the following:

"No value of the selection-type shall appear among the selection-values of more than one alternative. The "*" may appear in at most one alternative and refers to all values of the selection-type which do not appear in any other alternative."

3. Add Notes to 7.3.1:

NOTE - This generator represents the Pascal/Ada variant-record concept, and allows the C discriminated union to be supported by a slight subterfuge. E.g.:

```
choice( state(a1, a2, a3) ) of (
    a1: real,
    a2: integer,
    a3: boolean
)
```

NOTE - The useful forms of value-expression for the discriminant are late-binding or parametric-value which resolves to late-binding, but there is no reason to exclude the use of constants.

Item: 6
Title: Define the model of Pointer type
Rating: Major
Clause: 7.3.3 (Pointer)
Rationale:

1. The model of a pointer-type must be explained. WD5 introduces the term "variable", in lieu of "instance", but the notion "variable" is not a datatype and is not defined. The use of "variable should be removed since it implies the use of an update operation which does not appear anywhere else in the standard.
2. The characterizing operations on a pointer type must include some kind of "assignment", in the same way that Array types have Replace operations. This notion would lead to explaining the behavior of multiple pointers to the same value.

Proposed Change:

The model of the pointer datatype appropriately belongs in the rationale document for the CLIDT.

Delete clause 3.41 (definition of "variable")

In clause 7.3.3:

Values: First sentence, delete "a computational variable to which is associated"
Last line of first paragraph, change "any variable" to "any instance of the element datatype"
Move the second sentence of the last paragraph to the end of the first paragraph. Delete the first sentence of the last paragraph.

Description: Change "a variable containing a value" to "an instance"

Operations (Dereference): Change "possessed by the variable which the" to "identified by the ".
Delete "identifiers" at the end.

NOTE: Replace "variable" by "instance of the element datatype"

Item: 7
Title: State criteria for inclusion of datatypes in CLID
Rating: Minor
Clause: 6, possibly 5
Rationale:
The standard does not identify the criteria for determining which datatypes and datatype generators are defined in clause 7 and Annexes B and C. What is the criterion which makes a datatype a CLI Datatype, as opposed to a "user-defined" datatype?

Proposed Change:
Add a new paragraph 4 in Clause 6.1 and a note following this paragraph.
New text:

> The datatypes included in this standard are "common", not in the sense that they are directly supported, i.e., "built in" to many languages, but in the sense that they are common and useful generic concepts among professional users of datatypes.
>
> NOTE
>
> As an example, List belongs in the standard not merely (or even primarily) because it is a datatype supported by such languages as LISP, but because it is an extremely common conceptual datatype. This is illustrated by the fact that one can hardly find a textbook on data structures that ignores it.

Item: 8
Title: Make support of datatypes a clause
Rating: Editorial
Clause: 6.3 and 6 generally
Rationale:
The concept of "support" of a datatype is critical to the use of the CLID. It should be elevated to a clause by itself. And all the elements describing "support" and "preservation of properties" in the subclauses of clause 6 should be gathered into this new clause.

Proposed Change:
1. Create a new Clause X, titled Support of Datatypes, immediately before or after Clause 10 (Mappings).
2. Move clause 6.3 second paragraph to Clause X.
3. Move the last paragraph of each of 6.3.1, 6.3.2, 6.3.3, 6.3.4 and 6.3.6 to Clause X. Move the last 2 paragraphs of 6.3.5 to clause X.

Item: 9
Title: Fix syntax to allow trailing attributes and subtypes
Rating: Minor

Clause: 7
Rationale:

1. It is desirable to permit attributes to be added at the end of productions rather than in front of the objects they modify, but they would be ambiguous in several instances.

2. Syntactic ambiguities arise from the trailing position of subtype specifications, e.g.

List of Set of State(on, off): size(10).

Does the "size(10)" modify the List or the Set?

Proposed Change:

1. Delimit the element-types of generated-types:
   a. In 7.3.3: pointer-type = "pointer" "to" "(" base ")" .
   b. In 7.3.4: set-type = "set" "of" "(" element ")" .
   c. In 7.3.5: list-type = "list" "of" "(" element ")" .
   d. In 7.3.6: bag-type = "bag" "of" "(" element ")" .
   e. In 7.3.7:
      array-type = "array" "(" array-index-list ")" "of" "(" element ")" .
   f. In 7.3.8: table-type = "table" "(" key ")" "of" "(" element ")" .
   g. In 7.3.9 and in 8.2, delete the production for "components" and:
      defined-generator = generator-name ["(" parameter-list ")"]
      "of" "(" component-list ")" .

2. Make the corresponding change to explicit-subtype in 7.2.6:
   explicit-subtype = "restricted" "to" "(" subtype-definition ")" .

3. Move the attributes to behind the types.
   a. In 7:
      datatype-designator = CLI-datatype [type-attributes] .
   b. In 7.1.16:
      procedure-declaration =
      "procedure" procedure-name "(" [argument-list] ")"
      "raises" "(" exception-list ")" [procedure-attributes] .

   Argument attributes and field attributes should be left as is.

Item: 10
Title: Uniform specification of relative-error
Rating: Minor
Clause: 7.1.6, 7.1.10, 7.1.11, 7.1.12
Rationale:

The means of specification of "relative-error" and "precision" should be consistent. The current draft specifies values for Real and Complex as a Real number, and values for Scaled and Date-and-Time as two integers.

Proposed Change:

Make all four types use a common precision specification. We prefer the (radix, factor) form.

Item: 11
Title: Add properties to Private type
Rating: Minor
Clause: 7.1.15
Rationale:

The Private type defined in 7.1.15 is described as having two functionalities:

a) passing a "bit-string" which has a complex internal structure

through a CLID-conforming service, such as RPC, without the service having to "understand" its contents, and

b) describing a "handle", or similar object, whose structure and representation is known to one of the communicating entities but not to the other.

The first function is better performed by some kind of "bit-string" datatype, possibly conforming to the ASN.1 "octet string" model. In any case, the presumption is that the sender, the recipient and the service must preserve the length and ordering of the bit-string in order for the object to retain its meaning. This implies that this form of Private should be defined to have such properties.

The second function, in which the object is meaningless to all parties but one, requires the same "bit-string" length and ordering properties on the part of all but the knowledgeable routine. This implies that any interface definition would describe the object-type as Private, and only the "marshalling" service on the "knowledgeable" end would be aware of the true underlying datatype.

Proposed Change:

1. Move Private to Annex B and define it as derived from Array of Bit or List of Bit, having fixed size and only the characterizing operation Equal.
2. Mention the marshalling service concerns in a Note.

Item: 12
Title: Change syntax for omitted bounds
Rating: Minor
Clause: 7.2.1, 7.2.2, 7.2.3, 7.2.5, 7.3.7
Rationale:

The asterisk should not be used to mean both an externally specified bound in Array cases (7.3.7) and no bound in select-range specifications.

Proposed Change:

Allow the upperbound and lowerbound in select-range to be empty, i.e.

```
select-range = [ lowerbound ] ".." [ upperbound ] .
```

and modify the text of 7.2.1, .2, .3 accordingly, replacing the words "an asterisk" with "not present".

Replace the size production in 7.2.5 with:

```
size-subtype = base ":" size "(" size-specification ")" .
size-specification = minimum-size | minimum-size ".." [ maximum-size ] .
maximum-size = value-expression .
```

and modify the text of 7.2.5 under Components accordingly.

Item: 13
Title: Value-notation for declared datatypes
Rating: Minor
Clause: Annex B
Rationale:

There is no way to define a specific value notation for a declared datatype. As in the RPC document, it is clearly necessary to have a notation for character-string values and bit-string values, and it may be desirable to add such notations for other declared-datatypes.

Proposed Change:

1. Define a value notation for character-string.
2. Define a value notation for bit-string.

3. Define a standard value-notation mechanism for values of declared-datatypes in general, e.g. <type-name>.<value>.

Item: 14
Title: Make Annex B and C part of the main text
Rating: Minor
Clause: Annex B, Annex C
Rationale:
The inference drawn by nearly every reader of the draft is that the datatypes in Annex B and Annex C are considered to be "of lesser importance". Because they are not primitive, they are relegated to an Annex. This is not an acceptable disposition of CharacterString and BitString and possibly others. They are first-rank datatypes, even if they are not primitive.

Proposed Change:
1. Make a new clause X, following clause 8, titled: Derived Datatypes and Generators.
2. Make the current Annex B clause X.1.
3. Make the current Annex C clause X.2.
4. Consider whether some members of Annex B or C should be discarded, specifically B.5 (integer modulo).

Item: 15
Title: Remove Annex D
Rating: Minor
Clause: Annex D
Rationale:
Annex D, which deals with representational attributes, is outside the scope of the CLI Datatypes. It cannot be normative in the CLID and deserves attention as a useful standard for representation attributes for CLI Datatypes, which may be useful in CLIPC and RPC implementations.

Proposed Change:
1a. Remove Annex D from the draft.
b. On page 1, strike the last paragraph, except for the last sentence.

Item: 16
Title: Add ISO 2375 to Annex E
Rating: Minor
Clause: Annex E
Rationale:
ISO 2375 defines a registration mechanism which effectively defines character repertoires. It should be included in Annex E.

Proposed Change:
Add the following line to Annex E:
ISO 2375:??? Procedures for registration of escape sequences

Item: 17
Title: Pointer to Procedure
Rating: Question
Clause: 7.1.16 and 7.3.3
Rationale:

Does the type Pointer to Procedure(<arguments>) have the proper value space? Do the characterizing operations work correctly? Should Pointer-to-Procedure be a distinguished type in Clause 7 or Annex C?

This question should be considered when suggested revisions to Pointer and Procedure are considered as well.

Item: 18
Title: Add distinguished-name datatype
Rating: for consideration only
Clause: none
Rationale:

OSI defines a datatype "Distinguished Name", which is expected to be widely used in naming objects in distributed processing environments. This datatype should be in the CLID, to insure consistent definition.

Proposed Change:

Add a new subclause to clause 7.1:

7.1.x Distinguished-Name

Description: Distinguished Name is the datatype of the names of objects in the OSI directory (defined in ISO 9545).

Syntax: refer to ISO 9545

Values: refer to ISO 9545

Properties: non-numeric, unordered, discrete

Operations: Append, Detach, Equal, Last

```
Declare:  name-component = new CharacterString;
          with operations: Equal from CharacterString.
Append(x: distinguished-name, y: name-component):
distinguished-name is the new distinguished-name formed by appending
the relative name-component y to the distinguished-name x;
Detach(x: distinguished-name): distinguished-name formed by
removing the last name-component from the name-component sequence x.
If there is only one name-component in x, then the result is the
distinguished-name value ROOT;
Last(x: distinguished-name): name-component is the the
relative name-component value which is the last element of the
distinguished-name x;
Equal(x,y: distinguished-name): boolean =
    if x is the value ROOT and y is the value ROOT, then
        true
    else if x is the value ROOT or y is the value ROOT, then
        false
    else if name-component.Equal(Last(x), Last(y)) then
        Equal(Detach(x), Detach(y))
    else
        false.
```

Alternative: In a single clause in Annex B, define the name-component datatype as above, and declare:

```
type distinguished-name = list of name-component;
```

with the above description and operations (derived from list).

Item: 19
Title: Modify OSI Object-Identifier type
Rating: for consideration only
Clause: B.9
Rationale:

OSI-Object-Identifier should be a primitive datatype, as it is

in ASN.1, with a value syntax identical to that of ASN.1.

Proposed Change:

Move subclause B.9 into clause 7.1, with the following changes:

1. Replace the "Declaration" and "Parameters" paragraphs with:

```
Syntax:     object-identifier-type = "object" "identifier" .
            object-identifier-value = "{" objid-component-list "}" .
            objid-component-list = objid-component { objid-component } .
            objid-component = nameform | numberform | nameandnumberform .
            nameform = identifier .
            numberform = value-expression .
            nameandnumberform = identifier "(" value-expression ")" .
```

2. In the Values paragraph, replace "a finite sequence of integer values" with: "a non-empty finite sequence of values isomorphic to the integers" and strike "Any subsequence ... in the sequence".

3. Replace the Operations paragraph with:

```
Operations: Append, Length, Detach, Last, Equal.
    Declare:  object-id-component = new integer;
              with operations: Equal from integer.
    Append(x: object-identifier, y: object-id-component):
object-identifier is the new object-identifier formed by appending
the relative object-id-component y to the object-identifier x;
    Length(x: object-identifier): integer is the number of
object-id-component values in x;
    Detach(x: object-identifier): object-identifier, where
Length(x) > 1, is the object-identifier formed by removing the last
object-id-component from the object-id-component sequence x;
    Last(x: object-identifier): object-id-component is the the
relative object-id-component value which is the last element of the
object-identifier x;
    Equal(x,y: object-identifier): boolean =
        if Not(Length(x) = Length(y)) then
            false
        else if Not(object-id-component.Equal(Last(x), Last(y))) then
            false
        else if Length(x) = 1 then
            true
        else
            Equal(Detach(x), Detach(y)).
```

Alternative: Revise B.9 largely as indicated above, declaring

```
    type object-identifier = list of object-id-component;
```

with the above operations (derived from list).

Item: 20
Title: Procedure datatype
Rating: Major
Clause: 7.1.16
Rationale:

Removal of the "in | out | inout" syntax.
No need to determine if two procedure types are the same.

Proposed Change:

Annex F: Delete "direction = "in" | "out" | "inout""

Section 7.1.16:

Syntax: Delete "direction = "in" | "out" | "inout""

Values: Replace first 2 sentences by "Conceptually a procedure operates on a set of input values ("input-list") and produces a set of output values ("result-list"). It is outside the scope of the CLIDT to define the input and output values of a procedure.

Subtypes: Change the text of the first bullet to the following: "P is said to be formally compatible with Q if they have the same number of arguments, the number of elements in the "input-list" are the same, and the number of elements in the "output-list" are the same."

Change the text of the second bullet to the following: "If P is formally compatible with Q, and for every element in the "result-list" of Q, the element-type of the corresponding element-list of P is a (not necessarily proper) subtype of the Q element-type, then P is said to be a result-type of Q. If all of the argument-types in the result lists of P and Q are identical (none are proper subsets), then each is a result-subtype of the other.

Change the text of the third bullet to the following: "If P is formally compatible with Q, and for every element in the input-list of Q, the element-type of the corresponding element in the input-list of P is a (not necessarily proper) subtype of the Q element-type, then Q is said to be an input-subtype of P. If all of the argument-types in the input-lists of P and Q are identical (none are proper subtypes), then each is an input-subtype of the other.

Note(4): Remove occurrences of "in" and "out" from sample declarations.

Note(7): Second sentence, delete "These distinctions are supported by the syntax, but"