I thought the CLI Datatypes afficionados might appreciate the following interchange. Your comments are certainly welcome.

-Ed

>From @IBM.COM:DMY@YKTVMH3 Wed Mar 13 10:50:09 1991
Return-Path: <@IBM.COM:DMY@YKTVMH3>
Date: Wed, 13 Mar 91 10:16:51 EST
From: DMY@IBM.COM
Original To: edbark@cme.nist.gov
Copies: chi@osf.org, schaffert@crl.dec.com, YEMINI@YKTVMH3

Ed, Your note concerning "RPC, CLIPC and subsetting" was forwarded to me. Overall, I think that it does a good job on narrowing the gap between RPC and CLIDT, CLIPC. I believe that both T2 and T5 RPC agree that there should be one set of mandatory datatypes, for both RPC and CLIDT/CLIPC.

Although I agree with a majority of the comments, I have a few questions:

6. I agree that there should be a standard as to how to indicate date-and-time. The question is, however, whether or not it should be a primitive datatype or a derived one. Why can't it be a defined-type included in an appendix ? Perhaps what is needed is two appendices: one that is suggested and one that is mandatory: i.e., all implementations must support certain derived types such as date-and-time. One could view this as an interface which is implicitly imported by all other interfaces. (see your comment 31).

If this approach is adopted, there is some question as to what should be a mandatory derived type and what should be a primitive: in my opinion, there is perhaps more basis for making Object-Identifier a primitive and date-and-time a derived type. What about complex?

The reason that Object-Identifer is more of a primitive than time-and-date is that it needs a new primitive characterizing operation: create a unique object id. The characterizing operations on time-and-date, however, can be implemented in terms of record, etc. characterizing operations. This serves as a good way of distinguishing primitive versus derived datatypes.

7. I am not sure what you are suggesting. there are a bunch of related types: bit, octet, bitstring, private. Some subset of these are needed. Personally, I am not sure which ones. I am also unclear as to the meaning of private. What are its intended semantics? What are its characterizing operations. I am afraid that such a data type is really something that any application can use for whatever it wants, without much other semantics. If so, call a spade a spade: uninterpreted set of bits.

10. I agree with you concerning scaled. But others within the RPC group don't.

13. I believe that Null is another derived type that all implementations must support. It can be defined as a state datatype without any literals (if such a thing is allowed) or a state with a single literal. however, I do not understand your comment "what is the value of an OUT parm which is not returned?" In my view, there is no such beast. If your application wants to make an out parm optiional, there are two ways to do so. One way to define the parm as a choice of the type sometimes returned and null. The other option is to define an exceptional return which returns a different number of parms.

16. I need to more carefully consider your position on subtyping. At first glance, I think I can go along with this view. however, I am not at all sure how often anyone ever wants to extend state types etc. I think that because subtyping lends itself to some clean mathematical rigor, language designers sometimes tend to overdo it -- to supply more machinery than language users really need.

24. why is table a list of records instead of a set of records. I would think of it as a set of records, with the restriction that there be at most one element with a given key. Modeling it as a list implies a nonexistant ordering.

Concerning attributes, I believe that we (the RPC group) need to document our overall aims concerning attributes. I agree that most attributes do not concern communication and should therefore not be RPC specific.

Thanks, Dan

Reply To: DMY@IBM.COM
Copies: chi@osf.org schaffert yemeni
Subject: Your comments on my comments on CLID subsets

I am beginning to think that most of the really active participants in this effort are coming to one or two differences of opinion which we can resolve. The committees as a whole, on the other hand, seem to have widely differing views.

On your issues:

6a. Date-and-Time. The reason why this must be a "primitive" type is that it has distinct abstract semantics and there is no agreement on a convenient representation type. There are people who will insist on one Real, one Integer, an Array of Integer, a NumericString (which I think should be a defined-type in Annex B), or several other forms of CharacterString. This is a perfect instance of something which is common, but supported differently by different languages and systems. You want to let the language, package, vendor or user choose the representation convention and develop the necessary marshalling routines. At the CLIPC local level, the vendor will probably choose the System370 or VAX or Cyber or POSIX-Real-Time system representation, whereas RPC is probably well-advised to use ISO 8601.
Ed Greengrass (X3T2), however, has espoused your view that a Date-and-Time type can be "conceptually" a Record type with defined ordering and interval operations, and without regard for its actual representation. Paul Rabin, on the other hand, effectively makes it a defined-type of Integer. The problem with either defined-type, in my view, is that all the defined-types in the standard up till now are datatypes which are clearly "constructible" from the existing primitive types and generators, with the addition of semantics and minor changes to the operations. Defining date-and-time this way, a la Greengrass or Rabin, is a horse of a different color: it's choosing a debatable construction which has nothing to do with the semantics of the datatype. (A case can be made for Rabin's Integer representation for Date, but it's harder for Date-and-Time because you have to decide on the smallest time unit, which POSIX specifies, making it easy for Paul but not for arbitrary applications.) The SQL troops and I may represent a truly minority view that date-and-time is a primitive datatype; but if so, it is incumbent on you, the majority, to agree on the text by which it becomes a defined-type. (Good luck!)

6b. Object-Identifier. I agree with your definition of "generic" Object- Identifier as a primitive type. I would say that it is the "undefined State datatype" which is the source of values for Pointer. (And "create a unique object id" was a characterizing operation for Pointer, once Allocate, then Associate, until the warfare over its definition was resolved by removing it from WD4.) This in turn implies that you don't need both Pointer and Object-Identifier.
All this arises from the premise that "datatype" is intrinsically a lower-level of abstraction than "object" in the OOPL sense. That is, ALL datatypes are representations of more abstract objects; and therefore, the "object" which an Object-Identifier identifies must have (be represented by) a datatype, and thus a value of the Object-Identifier type must be a value of datatype Pointer to <the object datatype>. And for those of the weak binding persuasion, the generic Object-Identifier type is then Pointer to Any. Conversely, it can be argued that many values of State/Enumerated datatypes are "object-identifiers", as are many CharacterString values. If, of course, you don't accept the premise, then we need to find a new semantic

basis for further discourse.

OSI Object-Id, on the other hand, is something entirely different. It is a specific datatype which is used to perform the "object identification" function for a particular collection of applications. And its List of Integer infrastructure is independently meaningful: each Integer is really a representation of a value of a distinct State datatype (the name-and-number form confirms this), and which State datatype that is depends on the POSITION of the integer value and everything which precedes it in the List. Thus, OSI-Object-Id (which is what I wanted to call the thing, precisely in order to distinguish it from the generic notion you describe) is a perfect candidate for a defined-type, although there may be some debate about the appropriateness of the "Integer" part. The Dewey Decimal System values and Codex-numbers for international law have similar structures and serve as "object-identifiers" for other major collections of computerized applications, as do Social-Security-Number, Exchange/Standard-Stock-Code, Family-name/Given-name, and countless others, which do not have the same structure. Unlike these others, however, OSI-Object-Id is directly referenced in the RPC and therefore should appear in the IDN/CLID standards themselves.

7. If "octet" is "an uninterpreted set of bits", then "private" or "opaque" emphasizes the "uninterpreted"; and "octet" or "bitstring" emphasizes the "set of bits". As long as we have a spade, I don't care what it is called.

13. You make my point. The problem with an OUT parameter which has no value is that you have to have Null in order to be able to construct the Choice. The exception return, IF supported (languages have trouble with this), is probably a more elegant solution.

24. You are absolutely right. Table is a Set, not a List. In fact, I think the whole point in the CLID was that Table is to Set (or Bag) as Array is to List - a kind of mapping or access mechanism. I am generally of the opinion, by the way, that Bag is the generic constructor (ASN.1 SET OF), and that Set, requiring uniqueness, is a useful special case. I have observed, however, that many disagree with this view.