COMPUTING INTEGRITY
INCORPORATED
60 Belvedere Avenue
Point Richmond, CA 94801-4023
510.233.5400 Sales
510-233.5444 Support
510.233.5446 Facsimile

Hungarian Notation for OOABL

22 March 2006

The use of any form of Hungarian Notation is a topic certain to produce intense opinions, whether pro or con. Since CI intends to use a form of Hungarian Notation in its published examples and foundation classes, it seems worthwhile to review the reasons for this choice and the specifics of the standards that will be used.

Originally, Hungarian Notation was described by Charles Simonyi, a senior engineer at Xerox and later Microsoft. Since then, there have been many alternate variations on his core idea, some of which are rather different from the original intent. The core notion of Hungarian Notation is to apply a prefix to names in order to help identify "type". In the original formulation, later known as Apps Hungarian, the "type" was meant to be an indication of the variable's use or purpose. Subsequently, another major family of notation developed which is referred to as Systems Hungarian in which the "type" is intended to be a datatype. See http://en.wikipedia.org/wiki/Hungarian_notation for more discussion of the historical background.

The formalization used by CI, which is derived from one proposed a number of years ago by Peter Headland, extends the concept of "type" to include "direction" and "scope". "Direction" applies only to parameters and indicates whether the parameter is input, output, or input-output with respect to its context. "Scope" indicates the applicable scope of the variable. These three aspects combine to give one a significant amount of information about any particular variable without having to search for definitions and other references. It is believed that this very limited extension of type avoids the issues associated with systems using an open-ended extension of characteristics such as:

> a_crszkvc30LastNameCol : constant reference function argument, holding contents of a database column of type varchar(30) called LastName that was part of the table's primary key (example from the Wikipedia entry above).

In the CI formalization, the first character, which can be missing, is always direction, the second always scope (missing for database tables and fields), and the third and fourth indicate the type, so the prefix is limited to two to four characters from a well-defined set (chart provided at the end).

Hungarian Notation is often associated with the use of "CamelCase", i.e., mixed upper and lower letters to assist in identifying multiple words in a name, although there are many people who use CamelCase without advocating Hungarian Notation. Classic Hungarian Notation places the prefix in lower case immediately on the front of the word, e.g., chFirstName. CI's formalization utilizes a separating underscore to make the name and the type more readily distinguishable, e.g., ctt_Invoices and chn_Invoices.

Arguments against the use of Hungarian Notation usually focus on the question of readability. While this has some point in extreme cases such as the one quoted above, that example merely illustrates that any convention can be abused. Much of the issue of readability depends on what one

is used to. For example, Progress Software has historically used upper case for keywords. This is convenient when referring to isolated keywords in text, such as "OPEN QUERY", because it makes it clear that the words are "special", but in the context of code, there are many who feel that one is giving emphasis to the wrong part of the code, as if the keywords were shouting. For example, this fragment from the 10.1A 4GL Reference Manual, is shown there as:

```
DEFINE QUERY q-order FOR
    customer FIELDS (customer.cust-num customer.name customer.phone),
    order FIELDS (order.order-num order.order-date),
    order-line FIELDS (order-line.line-num order-line.price order-line.qty),
    item FIELDS (item.item-num item.item-name item.cat-desc).
OPEN QUERY q-order FOR EACH customer,
    EACH order OF customer,
    EACH order-line OF order,
    EACH item OF order-line NO-LOCK.
```

Converting this to lower case keywords, camel case names, no hyphens, but no Hungarian prefixes except the ones already in the PSC example, produces:

```
define query qOrder for
    Customer fields ( Customer.CustNum Customer.Name Customer.Phone ),
    Order fields ( Order.OrderNum Order.OrderDate ),
    OrderLine fields ( OrderLine.LineNum OrderLine.Price OrderLine.Qty ),
    Item fields ( Item.ItemNum Item.ItemName Item.CatDesc ).
open query qOrder for each Customer,
    each Order of Customer,
    each OrderLine OF Order,
    each Item of OrderLine no-lock.
```

The contrast is hardly dramatic, but has the effect of making the table and field names more prominent than the ABL keywords. Now lets add the type of Hungarian Notation used by CI:

```
define query lqu_Order for
    tb_Customer fields ( tb_Customer.ch_CustNum tb_Customer.ch_Name tb_Customer.ch_Phone ),
    tb_Order fields ( tb_Order.in_OrderNum tb_Order.da_OrderDate ),
    tb_OrderLine fields ( tb_OrderLine.in_LineNum tb_OrderLine.de_Price tb_OrderLine.in_Qty ),
    tb_Item fields ( tb_Item.in_ItemNum tb_Item.ch_ItemName tb_Item.ch_CatDesc ).
open query qu_Order for each tb_Customer,
    each tb_Order of tb_Customer,
    each tb_OrderLine OF tb_Order,
    each tb_Item of tb_OrderLine no-lock.
```

Note that in this example, all references are to table or field names and therefore the scope component is considered meaningless and thus is missing. If one is unused to reading Hungarian Notation, the initial reaction is likely to be that this is less readable than the prior example, but consider the additional information provided. We know, for example, that the query is local to a procedure and that all of the tables and fields are from the database. Suppose, for example, some or all of the "tb_" prefixes where replaced with "ltt_" prefixes; then we would know that we were dealing with local temp-tables rather than database tables. Or perhaps an "mtt_" prefix telling us that the temp-table was local to a method. This is significant additional information that is otherwise not available without reading additional code.

Now consider a simple statement like:

```
IF NOT ans THEN DO:
```

In the all caps keyword form, the variable almost disappears. Converting case we get:

```
if not Ans then do:
```

And, adding the Hungarian prefix we get:

```
if not llg_Ans then do:
```

In this case, we certainly would have guessed that Ans must have been a logical variable or it wouldn't compile, but the "llg_" prefix tells us that it is a local variable scoped to the procedure, which tells us a great deal about where it might be assigned a value. For example, suppose a block of code like:

```
if lch_Type = "C" then do:
  update llg_Ans with frame fr_Decision.
end.
if not llg_Ans then do:
```

In a case like this, we can see that the value of llg_Ans might come from the update, but it might come from somewhere else. Consider what a different set of information this is than if the name of the variable was iplg_Ans, telling us that it was an input parameter and therefore its initial value came from outside the procedure. One prefix tells us to look to variable definitions for default values, the other to the calling program.

One of the classic values of this type of notation is the ability to have two otherwise identical variable names that are of different type. E.g, ctt_Invoice and chn_Invoice, the former a temp-table scoped to a class and the latter a handle to that temp-table, also scoped to the class. Or, lin_Quantity and lch_Quantity where the former is the variable used in computations and the latter is the form used in displays to allow special handling for units of measure.

To the programmer who has spent the last 10 years reading nothing but upper case keyword code, probably even the lower case keyword code looks funny, much less something with underscores and strange letters hanging out in front. But, work with code that way for a while and it becomes expected, normal … and useful. I personally was opposed to this sort of notation until I was persuaded of its advantages and now that I am used to it, I would leave it behind only very reluctantly.

Interestingly, while there are many who are opposed to Hungarian Notation, they will use aspects of it in their own code. E.g., there are a number of examples of this that have appeared in more recent code from PSC such as ttCustomer to indicate a Customer temp-table or qOrder in the code sample above. This is particularly notable in the series of whitepapers by John Sadd on the OpenEdge Reference Architecture (OERA) which includes examples such as eOrder (used for the "entity" or domain object temp-table), etOrder.i (the file containing the definition of eOrder, hBuffer (handle to Buffer), srcOrder (query and data-source for Order), sceOrder.p (file containing the data-source code), dsOrder (data set for Order), daOrder.i (include file with data access code for Order), daOrderValidate.p (procedure file with validation logic for Order), beEntity.p (procedure for business entity for Order), etc. Not all of these are variable names, of course, but the principle is the same. What is clear is that, while John Sadd has not adopted a universal notation for all variables and other names, that he has adopted and advocates a strong naming convention for a number of key elements of his formulation of OERA.

In a similar fashion, other PABL shops have adopted their own conventions over the years to provide special names for part of the overall nameset. Some of these are similar to the examples in the prior paragraph, e.g., hBuffer, and others are less clear, e.g., the old Varnet Order# and Order## to indicate secondary and tertiary buffers. One might suggest that all such naming conventions are an indication of the recognition of some virtue in a simple, standardized prefix or suffix naming system to identify names of a particular type or derivative names (as hBuffer is a handle to Buffer), but those adopting these conventions have declined such a standard universally.

While the current document is focused specifically on a particular prefix notation, it should be noted that the notion of standardized naming practices is a much broader issue of great importance. E.g., if one is going to adopt a Finder and Mapper object structure for data access, as will be described elsewhere, it is clearly poor practice to suddenly name something OrderGetter instead of OrderFinder, because significant information is lost. Similarly, if one has defined a type of Collection Class as SortedSet, then if one calls an object OrderSortedSet, it had better be an instance of that type of Collection Class.

This issue of information provided through standard naming practice lies at the heart of CI's use of Hungarian Notation. One is faced with a choice of whether or not to provide potentially useful information as a part of a name and how to represent that information. E.g., in the case of a variable local to a method which is the character version of an order quantity for display purposes, one could include none of this context information and call the variable simply OrderQty, but then one might have a naming conflict with the integer version which is used for computation and persistence. One could, of course, call it OrderQtyChar or some such, but then, if the other variable is left as OrderQty, that doesn't immediately identify what relationship it has to the longer name. Moreover, because one name is a subset of the other, one can have issues trying to do search and replace because the shorter form cannot be uniquely identified by some editors under some conditions. And, of course, neither lets one know whether the variable is local to a method or internal procedure, is a parameter, or anything about its presumed characteristics. Those associations can only be determined by searching for definitions and other references.

By contrast, simply calling these two variables mch_OrderQty and min_OrderQty, or whatever scope prefix is appropriate, conveys this contrast very compactly, consistently, and with facility for doing whatever one might need in search and replace. This, by the way, is one of the reasons for the convention of using a "tb_" prefix for database tables, which one might otherwise be tempted to leave unmodified, since it makes searching for them easy.

It is not expected that this short discussion will convince anyone with strong "anti-Hungarian" views to suddenly convert to using Hungarian Notation. It is hoped, however, that having some understanding of why people might choose to adopt this convention and its possible benefits will at least result in some tolerance of this "peculiarity" of the code we will be publishing.

The full current set of naming standards in use at CI follows. These standards have recently been updated to include prefixes appropriate to 10.1A and so may undergo additional revision with experience.

CI **STANDARD NAMING PREFIXES**

Direction	
	Not applicable
b	Both In & Out
i	In
o	Out

Scope	
	Not applicable
c	Class
g	Global (rare)
i	Internal Procedure
l	Procedure
m	Method
p	Parameter
s	Shared (avoid)
t	Trigger

Type	
bl	Block
br	Browser
bf	Buffer
bt	Button
by	Byte
ch	Character
ds	DataSource
da	Date
dt	DateTime
dz	DateTime-TZ
de	Decimal
dl	Delimited List
fr	Frame
hn	Handle
im	Image
ix	Index
in	Integer
lb	Label
lg	Logical
ln	Long
lc	LongChar
mp	Memory Pointer
mn	Menu
mi	Menu Item
pp	Persistent Procedure
pd	ProDataSet
qu	Query
rw	Raw
ri	Recid/Rowid
rc	Rectangle
sq	Sequence
sh	Socket Handle
vw	SQL View
st	Stream
sm	Sub-Menu
sp	SuperProcedure
sy	Symbol
tb	Table
tt	Temp-Table
vn	Variable Name
wh	Widget-Handle
wp	Widget-Pool
wn	Window
wt	Work-Table (Avoid)
xh	XML Handle