Hungarian Notation Reference

You are requested to follow some standard naming convention for variables and functions, objects etc used in your programs . Your programs should clearly outline the purpose of the variable along with its declaration . Also the variable names should be self descriptive. For example "pipewt" is a bad choice for a variable, as the use of that variable is not clear looking at its name.

The following page is meant as an introduction to one such standard naming convention that is widely adopted.

Example of an Identifier in Hungarian Notation:

CString* m_ps	NameFirst
Prefix	Qualifier

What is Hungarian Notation? Top

Hungarian is a naming convention for identifiers in code, especially, but not exclusively, the C++ and Java languages. Each identifier would have two parts to it, a *type* and a *qualifier*.

- **type**: the first characters of the identifier specify what type the object is an instance of. This is achieved by adopting part of the name of the type as a prefix on the identifier. The prefix is always entirely lower-case.
- **qualifier**: the remainder of the name of the identifier describes what the variable is used for. The qualifier portion can be one word or a run-on word. The qualifier begins with a capital to distinguish it from the type prefix.

Example:
```
String sNameFirst; //s is a prefix for String type
char cLetter; //c is a prefix for the char type
Button butPushMe; //but is a prefix for the Button type
```

More explanation about Hungarian notation.

Type Prefixes Top

Common Type Prefixes

| b | by y | c | C | d | dw | f | g_ | h | I | l | m_ | n | p | s str | sz psz | u | v | w | x | X |

Prefix	Meaning	Example	Notes
p	Pointer	`Finger* pRude;`	In most cases, *p* is combined with another prefix; the prefix of the type of object being pointed to. For example: `String* psName` is a pointer to a string object containing a name.

s
str
	String	`String sName;`
		`String strName;`
			This convention is generally used for first-class string classes.

sz
psz
	zero-terminated / null-terminated string	`char szName[16];`
		`char* pszName;`	

h	Handle	`HWND hWindow`	
c	Character (char)	`char cLetter;`	Sometimes *c* is used to denote a counter object.

by
y
	Byte or Unsigned Char	`byte byMouthFull;`
		`byte yMouthFull;`	

n	Integer (int)	`int nSizeOfArray;`	
f	Float	`float fRootBeer;`	
d	Double	`double dDecker;`	

b	Boolean	`boolean bIsTrue;`
		`BOOL bIsTrue;`
		`int bIsTrue;`
			An integer can store a boolean value as long as you remember not to assign it a value other than 0 or 1

u	Unsigned...		
w	Word or Unsigned Integer	`unsigned int wValue;`	
l	Long	`long lIdentifier;`	Sometimes *l* is appended to *p* to denote that the pointer is a long. For example: `lpszName` is a long pointer to a zero-terminated string.
dw	Unsigned Long Integer		

C
or just a capital first letteri
	Class	`Class CObject;`
		`Class Object;`
			C is used heavily in Microsoft's Foundation Classes but using just a capital first letter is emphasized by Microsoft's J++.

I	Interface (ususally a struct or class with only pure virtual methods and no member variables)	`class IMotion {`
		`public:`
		`   virtual void Fly() = 0;`
		`};`
			Used extensively in COM.

X	Nested Class	`class CRocket {`
		`public:`
		`   class XMotion:public IMotion {`
		`   public:`
		`      void Fly();`
		`   } m_xUnknown;`
		`}`
			Used extensively in COM.

x	Instantiation of a nested class.	`class CAirplane {`
		`public:`
		`   class XMotion:public IMotion {`
		`   public:`
		`      void Fly();`
		`   } m_xUnknown;`
		`}`
			Used extensively in COM.

m_	Class Member Identifiers	`class CThing {`
		`private:`
		`   int m_nSize;`
		`};`	

g_	Global	`String* g_psBuffer`	Constant globals are usually in all caps. The *g_* would denote that a particular global is not a constant.
v	Void (no type)	`void* pvObject`	In most cases, *v* will be included with *p* because it is a

			common trick to typecast pointers to void pointers.

Top of table

Type Prefixes for non-common Types

In many cases, you will have identifiers of non-standard types. In this case, it will not do to borrow prefixes from the list of standard prefixes. Instead, the programmer needs to invent his own and, even more importantly, remain consistent with his own notation. To invent a prefix, abreviate the type name in a *short* and *meaningful* fashion. If it is not possible to abbreviate and the type name is not too long, you can just use the type name as a prefix.

The Type Prefix will always be entirely lowercase and should reflect the name of the type by abbreviating it distinctively.

Example:
```
    Button buttonPushMe; //this is okay, but awkward
    Button butPushMe; //a better abreviation for the type prefix
    Button bPushMe; //BAD: can be mistaken for a Boolean
```

Standard Qualifiers Top

The Qualifier portion of the identifier can be anything but should accurately and concisely describe the purpose of the object. There is a set of standard qualifiers for variables used in commonly performed programming tasks.

Qualifier	**Explanation**
`Sav`	A temporary from which the value will be restored
`Prev`	A value that follows one behind a current value in an iteration (eg. linked list node pointer)
`Cur`	Current value in some iteration
`Next`	Next value in some iteration
`Nil`	A special illegal value that is distinguished from all legal values. Typically denotes a certain absense of a legal value.
`Min`	Smallest legal index in an array or list. Typically zero.
`Max`	A strict upper limit for legal indexes in an array or list.

Other Hungarian Naming Conventions Top

Identifiers Containing Multiple Words

If the identifier contains multiple words, then the first letter of each word in the name is capitalized.

Example:
```
    int nThisIsAReallyLongIdentifier
```

If the identifier contains adjectives that describe the purpose of the object, the adjective succeeds the word it describes.

Example:
```
    CString sNameFirst; //first name
    int nNumCowsBrown; //number of brown cows
```

Functions

Functions should start out with a capital letter and no type prefix. It is not unheard of to start a function with a lowercase letter however. Microsoft urges the use of a capital first letter though the Java awt libraries use a lower-case first letter on member functions.

Example:
```
     CString GetName (); //Microsoft C++ standard
     String getName (); //Sun Java standard
```

Macros and constants

Macros and constants should be entirely in capitals.

Example:
```
     #define MAXSIZE 100
     const int MAXSIZE = 100;
```

More Information About Hungarian Notation

Top

The need for a standard

Programmers typically do not name their identifiers randomly; Programmers commonly name their identifiers mnemonically. However, every programmer is unique and one mnemoic identifier name may mean a lot to one person and nothing to another. A *convention* is merely a codified style of choosing mnemonics.

One important reason for a standard naming convention is to make your code more easily read by other people. Additional information about an object can be coded into its identifier so that someone reading code does not have to frequently refer back to the objects declaration to find out what *type* it is. Furthermore, if the identifier relates to what the object is used for in the code, then reading code is greatly simplified because the reader does not have to infer it from how it is used.

Origins of Hungarian notation

Charles Simonya, chief architect at Microsoft is the originator of the Hungarian Notation standard, which is used extensively in Microsoft Windows code. Simonya first used the notation in 1972. The notation was refered to as *Hungarian* originally as a criticism. At first glance, identifiers using Hungarian Notation appear to be gibberish until the pattern is deduced. Friends of Simonya compared Simonya's notation convention to some obscure foreign language and since Simonya is Hungarian, that was the obscure foreign language refered to.

Since its inception in 1972, Hungarian Notation has been adopted by Xerox, Apple, 3Com, and of course Microsoft.