

Appendix A

Hungarian for .NET Programs

Version 1.0
September 5, 2003
By: Paul Yao and David Durant

Time is a software developer's most precious resource. For that reason, anything that can help save time – at any phase of the project – while maintaining or increasing productivity is a good thing. When we started to work on this book¹, we began by collaborating on a set of standards to minimize the differences between our coding styles. As the project moved forward, we refined and updated various elements of this file. We offer this set to you, our readers, as a starting point for your own coding standards. (To get an editable copy of this file in Microsoft Word format, email info@pauyao.com.)

After the book is published, we expect to continue updating the various elements of this file and maintaining it on our web site (<http://www.pauyao.com>), and welcome your comments and suggestions. To submit a comment or suggestion, click on any hyperlink in a machine-readable version of this file. Each hyperlink is numbered, to help us understand the specific part of the document which triggered your response. For general comments, general suggestions, or new types to include, click on this link: [\[Comment A.1\]](#)

Goals:

- 1 Enhance code readability. [\[Comment A.2\]](#)
- 2 Make elements of the sample code – classes, properties, methods, events, fields – clearly distinct from those elements of the Compact Framework. [\[Comment A.3\]](#)
- 3 Work equally well in both C# and Visual Basic .NET. [\[Comment A.4\]](#)
- 4 Establish standards which would work equally well in Compact Framework code as well as desktop .NET Framework code. [\[Comment A.5\]](#)

Guidelines

These standards are for a coding project that exclusively uses Microsoft's .NET technology, specifically the C# and VB .NET languages. These are, in short,

¹ The books mentioned above (for the PDF version of this file) are: *Programming the .NET Compact Framework in C#* and *Programming the .NET Compact Framework in VB.NET*, both by Paul Yao and David Durant, and both published by Addison-Wesley.

recommendations that apply specifically to .NET projects. These standards were created with the following guidelines in mind: [\[Comment A.6\]](#)

- Case Insensitive – C# is case-sensitive, but VB .NET is not. For that reason, we avoided any convention that relied on distinguishing upper-case from lower-case names. The single exception to this rule involves the names for constants, and for members of enumerations, which shall be all upper-case. [\[Comment A.7\]](#)
- Hungarian Naming – Although this style of naming has lost favor with some, it has always been one that we like. For that reason, our naming standard relies heavily on Hungarian. [\[Comment A.8\]](#)
- The Visual Studio .NET forms designer suggests names, some of which can be left alone but others are likely to change. For example, labels on forms are given names like label1, label2, label3, etc. In many cases, programmers have no interest in changing these. As much as possible, the naming scheme described here works with the default names provided by the forms designer. [\[Comment A.9\]](#)
- Avoid terse abbreviations - both for prefixes and for variable names – in favor of making them more meaningful. While terse names make it easier to type, the problem they create is that they are somewhat hard to remember. For example, the name for `TextBox` is `text` rather than the shorter, but less understandable, `txt`. [\[Comment A.10\]](#)
- Include elements in the Compact Framework. The focus of this book is the Compact Framework, as such we avoid naming things that are not in the Compact Framework. Because one of the design goals of the Compact Framework was to maintain consistency with the desktop framework, this approach means that these guidelines can be extended as future versions of the Compact Framework become available. [\[Comment A.11\]](#)

.NET Naming Standards

The Microsoft .NET documentation provides some guidelines to start with. Of particular interest is a section titled "Design Guidelines for Class Library Developers." This strict set of rules helps insure that new class libraries – custom controls, SQL data adapters, etc. – are easily understandable to programmers who are already familiar with .NET conventions. You can read the details of these guidelines here: [\[Comment A.12\]](#)
<http://msdn.microsoft.com/library/en-us/cpgenref/html/cpconnamingguidelines.asp>.

Rather than provide the copious detail of those design guidelines, we have summarized the key points here. This book generally follows these guidelines, with a few exceptions which are noted later. Understanding these guidelines will help you write code that is generally more readable and maintainable than if you did not. [\[Comment A.13\]](#)

<i>Type</i>	<i>Convention</i>	<i>Notes and Examples</i>
<i>Containing Elements</i>		
Namespace [Comment A.14] s	Combine the following elements together for namespace names in public libraries: <ul style="list-style-type: none"> • Company Name • Technology Name 	Example: Microsoft.WindowsCE.Forms

<i>Type</i>	<i>Convention</i>	<i>Notes and Examples</i>
	<ul style="list-style-type: none"> • Feature • Design 	
Class [Comment A.15] ^s	<ul style="list-style-type: none"> • Use nouns or noun phrases for classes • For derived classes, create compound names for classes by appending base class name at end of new class name. 	<p>Examples of nouns as class names:</p> <ul style="list-style-type: none"> • Control, Form, TextBox <p>Examples of derived class names:</p> <ul style="list-style-type: none"> • DateTextBox • MainForm
<i>Contained Elements</i>		
Attribute [Comment A.16] ^e	<ul style="list-style-type: none"> • Use suffix of "Attribute" for custom attribute classes. 	<p>Example:</p> <p>HelpFileAttribute, AuthorAttribute</p>
Enumeration [Comment A.17] ⁿ	<ul style="list-style-type: none"> • Do not use "Enum" in name of enumeration 	<p>Examples:</p> <ul style="list-style-type: none"> • CombineMode • FontStyle • GraphicsUnit • Keys • DayOfWeek • FileShare
Event Argument Class [Comment A.18] ^s	<ul style="list-style-type: none"> • Use suffix of EventArgs for event argument class • 	<p>Examples:</p> <ul style="list-style-type: none"> • RowUpdatedEventArgs • KeyEventArgs • MouseEventArgs
Event Handler [Comment A.19] ^r	<ul style="list-style-type: none"> • Use suffix of "EventHandler" for event handler itself 	<p>Examples:</p> <ul style="list-style-type: none"> • TreeViewEventHandler • KeyEventHandler
Event Handling Method [Comment A.20] ^d	<ul style="list-style-type: none"> • Provide protected method OnXXX where XXX = event name 	<p>Examples:</p> <ul style="list-style-type: none"> • OnParentChanged • OnRowUpdated • OnKeyDown • OnMouseMove
Event Name [Comment A.21] ^s	<ul style="list-style-type: none"> • Use verbs for event names 	<p>Examples:</p> <ul style="list-style-type: none"> • ParentChanged • EnabledChanged • GotFocus • LostFocus
Exception [Comment A.22] ^s	<ul style="list-style-type: none"> • Use suffix of "Exception" for exception names. 	<p>Examples:</p> <ul style="list-style-type: none"> • ArithmeticException • DivideByZeroException • IndexOutOfRangeException
Interface [Comment A.23] ^e	<ul style="list-style-type: none"> • Use prefix of "I" for interface names. • Use nouns, noun phrases, or adjectives that describe behavior. 	<p>Examples:</p> <ul style="list-style-type: none"> • ICollection • IComparer • IDictionary • IEnumerable • IEnumerator • IList
Method [Comment A.24] ^s	<ul style="list-style-type: none"> • Use verbs or verb phrases 	<p>Examples:</p> <ul style="list-style-type: none"> • Activate

<i>Type</i>	<i>Convention</i>	<i>Notes and Examples</i>
		<ul style="list-style-type: none"> • Close • Invoke • ShowDialog
Method Parameter [Comment A.25] s	<ul style="list-style-type: none"> • Use descriptive parameter names • Do not use Hungarian naming • Use type-based parameter names sparingly 	Example:
Property [Comment A.26]	<ul style="list-style-type: none"> • Use noun or noun phrase • Do not use Hungarian naming 	Examples: <ul style="list-style-type: none"> • BackColor • Font • Text • Width
Static Field [Comment A.27]	<ul style="list-style-type: none"> • Use noun, noun phrases, or noun abbreviations for the name • Use Hungarian naming 	Examples:
Fields [Comment A.28]	<ul style="list-style-type: none"> • Do not use Hungarian naming – good names describe semantics not type 	

Hungarian Naming

Hungarian naming refers to a style of creating meaningful variable names. An MSDN Technical Article gives this explanation for how this approach got its name: [\[Comment A.29\]](#)

"It came to be known as "Hungarian notation" because the prefixes make the variable names look a bit as though they're written in some non-English language and because Simonyi is originally from Hungary." – Note from Dr. Gui, November, 1999. [\[Comment A.30\]](#)

Rather than repeat all the details that are so well described in this article, here is a link to a reprint of Charles Simonyi's original white paper on this naming convention for readers who wish to learn more: <http://msdn.microsoft.com/library/en-us/dnvsngen/html/hunganotat.asp>. [\[Comment A.31\]](#)

Hungarian naming is not a standard. In other words, you will not find an ISO standards committee – or, for that matter, an ANSI / IEEE / ECMA committee – that dictates what prefixes you can and cannot use. While there are many lists of suggested prefixes – including this appendix – none claim to be the definitive one that supersedes all others. [\[Comment A.32\]](#)

Hungarian naming is, instead, a style. Lists (like this one) that call themselves "standards" are really just starting points for project-specific data types, because with few exceptions every programming project has its own unique set of types. Where you get the greatest benefits from Hungarian are on projects where all agree to use the same set of types. It provides a common language so that any team member can look at any code and understand it. [\[Comment A.33\]](#)

We have a friend who used to work at Microsoft. He states that the disciplined use of Hungarian naming – and a project-specific set of types and associated Hungarian prefixes – allowed him to transfer to the Excel team during a crunch period and be productive very quickly. [\[Comment A.34\]](#)

This Book

We use Hungarian naming throughout this book for data types. We do so because we find it makes our code more readable, and therefore more understandable. [\[Comment A.35\]](#)

The use of Hungarian also provides another important benefit: it allows you to quickly find names in an IntelliSense list. IntelliSense is, of course, a feature of the Visual Studio .NET text editing windows. Just about every time you type a dot operator, it pops up with possible choices for the next part of an operation. [\[Comment A.36\]](#)

For example, suppose you are writing some code and need to know the name of a string variable that is a data member of the class you are writing. The "old-fashioned" way to find the variable name would be to do a search, or to scroll to the top of your source file (or wherever you normally put such things). When you get there, you might find a list of string names defined like this: [\[Comment A.37\]](#)

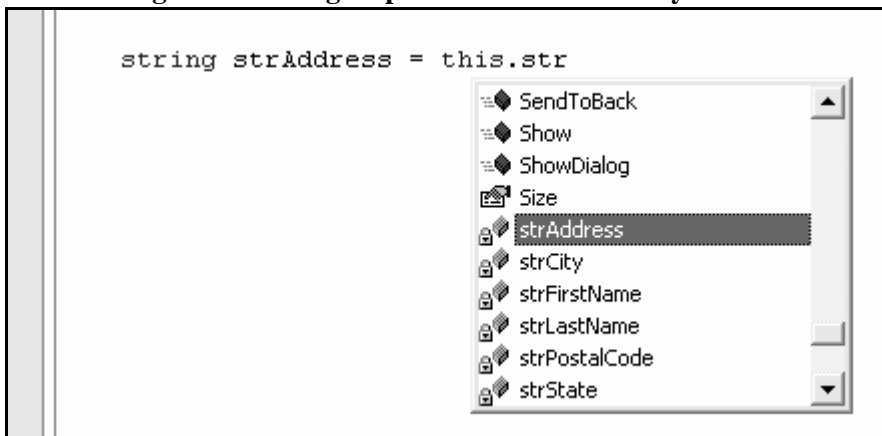
```
string strFirstName;  
string strLastName;  
string strAddress;  
string strCity;  
string strState;  
string strPostalCode;
```

Another alternative is to ask IntelliSense to help you. By typing

```
this.str
```

in the text editor window, you give IntelliSense enough details to help you remember the specific name that you were looking for. Figure B-1 shows how the IntelliSense window showing the available data members that starts with a "str" prefix. [\[Comment A.38\]](#)

Figure B-1 Hungarian Naming helps IntelliSense remind you of variable names



The “m_” Prefix for Private Data

A variation to the Hungarian style that we adopt in this book is the use of an additional prefix – m_ – for private class data members. This helps us instantly distinguish public data from private data. Where it is particularly helpful is in allowing a public property and an associated private data member with the same name – the only difference being the private data member has the m_ prefix. Here is an example: [\[Comment A.39\]](#)

```
private bool m_bThreadContinue; // Continue flag.

public bool bThreadContinue // Continue property.
{
    get { return m_bThreadContinue; }
    set { m_bThreadContinue = value; }
}
```

The user of this class always uses the public bThreadContinue property, which gets translated into the private m_bThreadContinue only visible to our class. [\[Comment A.40\]](#)

The Forms Designer creates private data members for you – one for each control created in a form. We do not use the m_ prefix for such private data members. [\[Comment A.41\]](#)

While assembling this set of standards, the m_ prefix prompted the most discussion. We both found it useful, but wanted to make sure it would be reasonable for Compact Framework code. We found it widely used in pre-.NET technologies, including C++ projects (for the MFC class library and the ATL template library), Visual Basic, and in Active Server Pages. [\[Comment A.42\]](#)

What convinced of its applicability to the Compact Framework was its use in various .NET-based technologies, including Rotor, the shared source code implementation of the Common Language Infrastructure (CLI). We also found that the Compact Framework itself uses m_ for all of its private data, which are all not documented, but which become visible with a tool like ILDASM.EXE. [\[Comment A.43\]](#)

Hungarian Prefixes for CTS Value Types

The Common Type System (CTS) defines a set of types that all languages must support to be .NET-compatible. C# supports types outside this specification, most of which are unsigned types. In the .NET documentation, these are marked as "not CLS-compliant." To enhance code portability, we non-CLS-compliant types. The following table includes only CLS-compliant types. [\[Comment A.44\]](#)

<i>.NET Class</i>	<i>C# Alias</i>	<i>Visual Basic .NET Alias</i>	<i>Size</i>	<i>Prefix</i>
System.Boolean [Comment A.45]	bool	Boolean	1 byte	b
System.Char [Comment A.46]	char	Char	2 bytes	ch
System.DateTime [Comment A.47]		Date	8 bytes	date
System.Decimal	decimal	Decimal	16 bytes	dec

[Comment A.48]				
<i>INTEGER TYPES</i>				
System.Byte [Comment A.49]	byte	Byte	1 byte	byt
System.Int16 [Comment A.50]	short	Short	2 bytes	sh
System.Int32 [Comment A.51]	int	Integer	4 bytes	i
System.Int64 [Comment A.52]	long	Long	8 bytes	l
<i>Platform-Specific Pointer or Handle Types</i>				
System.IntPtr [Comment A.53]			4 bytes	iptr ²
<i>FLOATING POINT TYPES</i>				
System.Single [Comment A.54]	float	Single	4 bytes	sin
System.Double [Comment A.55]	double	Double	8 bytes	dbl
<i>OTHER TYPES</i>				
System.Guid [Comment A.56]			16 bytes	guid

Hungarian Prefixes for System Classes [\[Comment A.57\]](#)

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
Count of bytes [Comment A.59]	cb	cbReturnBuffer, cbLength
Count of characters [Comment A.60]	cch	cchMaxName, cchSearchBuffer
EventArgs [Comment A.61]	e	e - the standard event argument for all event handlers.
Exception [Comment A.62]	ex	ex - the standard exception argument for all exceptionhandlers
Index for loops [Comment A.63]	i	iItem, iFont, iFile
Object [Comment A.64]	obj	objDebugInput, objItem, objCollectionStart
String [Comment A.65]	str	strPath, strFileName, str

² Generally speaking, we do not use this prefix for IntPtr. Instead, we use the prefix for the type being represented by the IntPtr variable. For example, a Win32 window handle will have a prefix of hwnd instead of iptr. This accomplishes the goal of making our code more readable and to reduce the coding errors that arise from misusing variables.

Delegate [Comment A.66]	dele	deleFileFound, deleAddItem
--	------	----------------------------

Hungarian Prefixes for System.Text Classes

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
StringBuilder [Comment A.67]	strbld	strbldInputBuffer, strbldErrorLog
Decoder [Comment A.68]	decode	decodeFileBuffer, decodeHTML
Encoder [Comment A.69]	encode	encodeDisplayBuffer, encodeConvert

Hungarian Prefixes for System.Collection Classes

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
<Any> [Comment A.70]	a<type>	int [] ai = new int[10]; // Array of integers. ai[0] = 1; ai[1] = 2;
ArrayList [Comment A.71]	al<type>	ArrayList ali = new ArrayList(); // Integer ArrayList ali.Add(1); ali.Add(2);
BitArray [Comment A.72]	bitarr	bitarrDisplayFlags, bitarrCollectionType
HashTable [Comment A.73]	hash	hashFileNames, hashLeftover
Queue [Comment A.74]	queue	queueSendRequest, queueWaitingList
Stack [Comment A.75]	stack	stackFunctionCall, stackUndoCommands

Hungarian Prefixes for Microsoft.Win32 Classes (Desktop Framework Only)

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
RegistryKey [Comment A.76]	regkey	regkeyRoot, regkeySoftware, regkeyVer10

Hungarian Prefixes for System.Windows.Forms Classes

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
Button a.k.a "Command Button" [Comment A.77]	cmd	cmdOk - OK button cmdReset - Button to reset form
CheckBox [Comment A.78]	chk	chkWordWrap, chkToolBar, chkCloseWhenDone
ComboBox [Comment A.79]	combo	comboStates, comboTaxSettings

ContextMenu [Comment A.80]	cmenu	cmenuMain, cmenuOptions
Control [Comment A.81]	ctrl	ctrlParent
DataGrid [Comment A.82]	dgrid	//a "Data-Bound Grid" dgridPhoneList
DomainUpDown [Comment A.83]	dupdown	
Form [Comment A.84]	form	formMain - main form frm - temp variable
<form used as a dialog box> [Comment A.85]	dlg	dlgFileOpen dlgFileSave
HScrollBar [Comment A.86]	hscroll	hscrollLines, hscrollElevation
ImageList [Comment A.87]	ilist	ilistFile, ilistEdit
Label [Comment A.88]	label	label1 - default label name labelStatus
ListBox [Comment A.89]	lbox	lboxFileNames
ListView [Comment A.90]	lview	lviewVisitors, lviewSymptoms
MainMenu [Comment A.91]	menu	menuMain, menuNew
MenuItem [Comment A.92]	mitem	mitemFileOpen, mitemEditCopy
NumericUpDown [Comment A.93]	nupdown	nupdownMaxItems, nupdownPassengers
Panel [Comment A.94]	panel	panelColorScroll, panelDiscountGroup
PictureBox [Comment A.95]	pbox	pboxPreview, pboxThumbnail
ProgressBar [Comment A.96]	pbar	pbarDownload, pbarRecalc
RadioButton [Comment A.97]	opt	"opt" stands for "Option button", as in optRed, optGreen, optBlue
StatusBar [Comment A.98]	status	statusMain, statusHelpWindow
TabControl [Comment A.99]	tab	tabFileDialog, tabOptionsDialog
TabPage [Comment A.100]	tpage	tpageOptions, tpageFormat
TextBox [Comment A.101]	text	textName textFirst textPassword
Timer [Comment A.102]	timer	timerPowerSave

ToolBar [Comment A.103]	tbar	tbarFile, tbarOptions
ToolBarButton [Comment A.104]	tbutton	tbuttonFileOpen, tbuttonEditCopy
TrackBar [Comment A.105]	tbar	tbar
TreeView [Comment A.106]	tview	tvwDirectory tvwOrgChart
VScrollBar [Comment A.107]	vscroll	vscrollRed, vscrollGreen, vscrollBlue

Hungarian Prefixes for Microsoft.WindowsCE.Forms Classes (Windows CE only)

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
InputPanel [Comment A.108]	sip	sipMain
MessageWindow [Comment A.109]	msgwnd	msgwndTextBox, msgwndSearchDone
Message [Comment A.110]	msg	msgIn

Hungarian Prefixes for System.Drawing classes

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
Bitmap [Comment A.111]	bmp	bmpThumb, bmpNext
Brush [Comment A.112]	br	brRed, brGreen, brYellow
Color [Comment A.113]	color	colorBlack, colorActiveWindow
Font [Comment A.114]	font	fontCurrent, fontArial
Graphics [Comment A.115]	g	g - Standard for Graphics object
Icon [Comment A.116]	icon	iconHelp, iconQuestion, iconMain
Image [Comment A.117]	img	imgApp, imgThumb
Pen [Comment A.118]	pen	penRed, penBlack
Point [Comment A.119]	pt	ptStart, ptEnd
PointF [Comment A.120]	ptf	ptfName, ptfString
Rectangle [Comment A.121]	rect	rectStory
RectangleF [Comment A.122]	rectf	rectfPreview

Region [Comment A.123]	reg	regScroll
Size [Comment A.124]	sz	szBackground
SizeF [Comment A.125]	szf	szfStringSize

Hungarian Prefixes for System.Data. classes

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
ConstraintCollection [Comment A.126]		
ConstraintException [Comment A.127]	ex	
DataColumn [Comment A.128]	dcol	
DataColumnChangeEventArgs [Comment A.129]	e	
DataColumnChangeEventHandler [Comment A.130]		
DataColumnCollection [Comment A.131]		
DataException [Comment A.132]	ex	
DataRelation [Comment A.133]		
DataRelationCollection [Comment A.134]		
DataRow [Comment A.135]	drow	
DataRowChangeEventArgs [Comment A.136]	e	
DataRowChangeEventHandler [Comment A.137]		
DataRowCollection [Comment A.138]		
DataRowView [Comment A.139]		
DataSet [Comment A.140]	dset	
DataTable [Comment A.141]	dtab	
DataTableCollection [Comment A.142]		
DataRowView [Comment A.143]		
DataManager [Comment A.144]		
DataRowViewSetting [Comment A.145]		
DataRowViewSettingCollection [Comment A.146]		
DBConcurrencyException [Comment A.147]	ex	
DeletedRowInaccessibleException	ex	

[Comment A.148]		
DuplicateNameException [Comment A.149]	ex	
EvaluateException [Comment A.150]	ex	
FillErrorEventArgs [Comment A.151]	e	
FillErrorHandler [Comment A.152]	e	
ForeignKeyConstraint [Comment A.153]		
InRowChangingEventException [Comment A.154]	ex	
InvalidConstraintException [Comment A.155]	ex	
InvalidExpressionException [Comment A.156]	ex	
MissingPrimaryKeyException [Comment A.157]	ex	
NotNullAllowedException [Comment A.158]	ex	
PropertyCollection [Comment A.159]		
ReadOnlyException [Comment A.160]		
RowNotInTableException [Comment A.161]	ex	
StateChangeEventArgs [Comment A.162]	e	
StateChangeEventHandler [Comment A.163]		
SyntaxErrorException [Comment A.164]		
UniqueConstraint [Comment A.165]		
VersionNotFoundException [Comment A.166]	ex	

Hungarian Prefixes for System.Data.SqlServerCe classes

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
SqlCeConnection [Comment A.167]	conn	
SqlCeCommand [Comment A.168]	cmd	
SqlCeDataReader [Comment A.169]	drdr	
SqlCeException	ex	

[Comment A.170]		
SqlCeError [Comment A.171]	err	
Engine [Comment A.172]	db	
SqlCeDataAdapter [Comment A.173]	dapt (what about sqladpt ??)	
SqlCeCommandBuilder [Comment A.174]	cbld	

Hungarian Prefixes for System.IO classes

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
FileStream [Comment A.175]	fs	

Hungarian Prefixes for System.Threading classes

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
AutoResetEvent [Comment A.176]	autoevent	
ManualResetEvent [Comment A.177]	manevent	
Mutex [Comment A.178]	mutex	
Thread [Comment A.179]	thread	
Timer [Comment A.180]	timer	
WaitHandle [Comment A.181]	whand	

Hungarian Prefixes for System.Xml Classes

<i>Class</i>	<i>Prefix</i>	<i>Examples</i>
NameTable [Comment A.182]	xntable	
XmlAttribute [Comment A.183]	xattr	
XmlAttributeCollection [Comment A.184]	xattcoll	
XmlCDataSection [Comment A.185]	xcdatasec	
XmlCharacterData [Comment A.186]	xchardata	
XmlComment [Comment A.187]	xcom	

XmlDeclaration [Comment A.188]	xdecl	
XmlDocument [Comment A.189]	xdoc	
XmlDocumentFragment [Comment A.190]	xdfrg	
XmlElement [Comment A.191]	xel	
XmlEntityReference [Comment A.192]	xeref	
XmlImplementation [Comment A.193]	ximp	
XmlNamedNodeMap [Comment A.194]	xnm	
XmlNamespaceManager [Comment A.195]	xnsmanager	
XmlNodeChangedEventArgs [Comment A.196]	e	
XmlNodeReader [Comment A.197]	xnreader	
XmlParserContext [Comment A.198]	xcontext	
XmlProcessingInstruction [Comment A.199]	xpi	
XmlQualifiedName [Comment A.200]	xqname	
XmlResolver [Comment A.201]	xresolver	
XmlSignificantWhitespace [Comment A.202]	xsigwhite	
XmlText [Comment A.203]	xtext	
XmlTextReader [Comment A.204]	xtr	
XmlTextWriter [Comment A.205]	xtw	
XmlUrlResolver [Comment A.206]	xuresolver	
XmlWhitespace [Comment A.207]	xwhite	