Module17: Dyalog.Net

§ 17.1 Revealing the .NET Framework

Vade mecum, but be prepared to meet some deeper shades of gray ☻.

§§ 17.1.1 Getting Microsoft .NET

The Microsoft .NET Framework is the new low-level platform on which all Windows applications, and, with the immanent appearance of Windows Vista, even operating systems, are supposed to be built. Once upon a time Windows was built from DOS, the *Disk Operating System* software backbone of 80's PCs. DOS itself was probably (see http://museum.sysun.com/museum/cpmhist.html) modified CP/M. CP/M was originally developed around 1975 for Intel 8080 and Zilog Z80 on Intel's 8080 emulator under DEC's TOPS-10 operating system.

Dyadic Systems and Zilog Inc. developed Dyalog *APL* as a joint venture around 1980 ☺.

By the beginning of the 90's, the Microsoft Windows *Application Programmers Interface* (API) had appeared, based essentially on **C** functions in *Dynamic Link Libraries* (.DLL files). This was intended to replace DOS completely as the new basis for application development. Windows and its API slowly evolved away from DOS. Around the mid 90's, VB/VBA and OLE/COM were introduced as the new foundations (language *cum* interface) upon which all user level applications were supposed to be built. These dreams are still being realised today.

But now the intension is to replace all of these platforms with a new platform, called Microsoft .NET, which is built 'over' the Windows API and is based essentially on libraries of **C#** functions. Other languages, such as Dyalog APL, VB.NET, C++, Jscript, COBOL, FORTRAN, Python, RPG, Pascal, SmallTalk, Perl, Oberon and Eiffel, can also contribute libraries as equal partners because of the **common language specification** at the entrance to Microsoft .NET Framework functionality. The .NET base class libraries, or *assemblies*, comprising over 30 .DLL files, contain a huge array of functions embedded in *classes*, grouped within *namespaces* by area of application. These functions can all be called through a highly object-oriented approach by a growing a number of programming languages, including Dyalog APL. Documentation is found at http://msdn.microsoft.com/netframework/gettingstarted/default.aspx.

The Dyalog APL interface to .NET (Dyalog.Net) has been available from Dyalog version 9.5 onwards. To run .NET a computer requires Windows 2000 or Windows XP Professional together with the Microsoft .NET Framework (version 1 + SP1 or version 2). Both are freely installable from Microsoft downloads at http://www.microsoft.com/downloads/Search.aspx?displaylang=en via **dotnetfx.exe**. The .NET platform will be an integral part of the next and subsequent operating systems from Microsoft.

You can check your .NET framework installation level from [Control Panel][Add or Remove Programs], or from registry entry HKEY_LOCAL_MACHINE\Software\Microsoft\NET Framework Setup\NDP\v1.1.4322\SP, or by looking in directory ..\WINDOWS\Microsoft.NET\...

§§ 17.1.2 Assemblies (à), Namespaces (ñ) and Classes (¢)

When Microsoft .NET is installed on your computer, you will find, a directory called something like ..\Microsoft.NET\Framework\v1.1.4322\ in your Windows directory. This directory contains about 30 DLLs that together (or in stand-alone subsets) form the *substance* of the .NET Framework and contain almost all of the Microsoft-supplied functionality available to .NET programmers.

===== ASSEMBLIES (à) in the .NET Framework =====

mscorlib.dll
System.dll
System.Configuration.Install.dll
System.Data.dll
System.Data.OracleClient.dll
System.Design.dll
System.DirectoryServices.dll
System.Drawing.dll
System.Drawing.Design.dll
System.EnterpriseServices.dll
System.EnterpriseServices.Thunk.dll
System.Management.dll
System.EnterpriseServices.Thunk.dll
System.Management.dll
System.Messaging.dll
System.Runtime.Remoting.dll
System.Runtime.Serialization.Formatters.Soap.dll
System.Security.dll
System.ServiceProcess.dll
System.Web.dll
System.Web.Mobile.dll
System.Web.RegularExpressions.dll
System.Web.Services.dll
System.Windows.Forms.dll
System.XML.dll
cscompmgd.dll
ISymWrapper.dll
Microsoft.Jscript.dll
Microsoft.VisualBasic.dll
Microsoft.Vsa.dll

All word phrases here in green may be used, in one way or another (in character string arguments, as methods/functions, as properties/variables, …) in Dyalog.Net code. There are over 14,000 new unique dot-qualified strings available for inclusion in your programs in Dyalog.Net.

> It's like going from the Roman alphabet to Chinese script, or from ⎕AV to Unicode, or from {the set of all letters} to {the set of all phrases}, or from the safe set of integers, ℤ, all-be-they of infinite number aleph null ($\aleph_0$), to the wild real numbers, ℝ, or the beautiful complex numbers, ℂ, both of aleph one ($\aleph_1$)!

The .NET framework is highly object-oriented. An *instance* of an object is generally created from a *class* which holds the object creation code. A class represents a species of object - like the Dandelion (*Taraxacum officinale*) represents all the dandelions in your garden. Essentially, each **.NET assembly** (a logical .DLL) contains a number of **.NET namespaces** that each contains many **.NET classes** (or object blueprints). Classes contain *members* – these members include **methods**, **properties**, **fields** and **events**.

Here is a list of almost all the .NET namespaces from almost all the DLLs in the Microsoft .NET Framework (version 1.1). Their content comprises the .NET base class library.

===== NAMESPACES (ñ) in .NET Framework =====

System
System.CodeDom
System.CodeDom.Compiler
System.Collections
System.Collections.Specialized
System.ComponentModel
System.ComponentModel.Design
System.ComponentModel.Design.Serialization
System.Configuration
System.Configuration.Assemblies
System.Configuration.Install
System.Data
System.Data.Common
System.Data.Odbc
System.Data.OleDb
System.Data.OracleClient
System.Data.SqlClient
System.Data.SqlServerCE
System.Data.SqlTypes
System.Diagnostics
System.Diagnostics.SymbolStore
System.DirectoryServices
System.Drawing
System.Drawing.Design
System.Drawing.Drawing2D
System.Drawing.Imaging
System.Drawing.Printing
System.Drawing.Text
System.EnterpriseServices
System.EnterpriseServices.CompensatingResourceManager
System.EnterpriseServices.Internal
System.Globalization
System.IO
System.IO.IsolatedStorage
System.Management
System.Management.Instrumentation
System.Messaging
System.Net
System.Net.Sockets
System.Reflection
System.Reflection.Emit
System.Resources
System.Runtime.CompilerServices
System.Runtime.InteropServices
System.Runtime.InteropServices.CustomMarshalers
System.Runtime.InteropServices.Expando
System.Runtime.Remoting
System.Runtime.Remoting.Activation
System.Runtime.Remoting.Channels
System.Runtime.Remoting.Channels.Http
System.Runtime.Remoting.Channels.Tcp
System.Runtime.Remoting.Contexts
System.Runtime.Remoting.Lifetime
System.Runtime.Remoting.Messaging

System.Runtime.Remoting.Metadata
System.Runtime.Remoting.Metadata.W3cXsd2001
System.Runtime.Remoting.MetadataServices
System.Runtime.Remoting.Proxies
System.Runtime.Remoting.Services
System.Runtime.Serialization
System.Runtime.Serialization.Formatters
System.Runtime.Serialization.Formatters.Binary
System.Runtime.Serialization.Formatters.Soap
System.Security
System.Security.Cryptography
System.Security.Cryptography.X509Certificates
System.Security.Cryptography.Xml
System.Security.Permissions
System.Security.Policy
System.Security.Principal
System.ServiceProcess
System.Text
System.Text.RegularExpressions
System.Threading
System.Timers
System.Web
System.Web.Caching
System.Web.Configuration
System.Web.Hosting
System.Web.Mail
System.Web.Mobile
System.Web.Security

System.Web.Services
System.Web.Services.Configuration
System.Web.Services.Description
System.Web.Services.Discovery
System.Web.Services.Protocols
System.Web.SessionState
System.Web.UI
System.Web.UI.Design
System.Web.UI.Design.WebControls
System.Web.UI.HtmlControls
System.Web.UI.MobileControls
System.Web.UI.MobileControls.Adapters
System.Web.UI.WebControls
System.Windows.Forms
System.Windows.Forms.Design
System.Xml
System.Xml.Schema
System.Xml.Serialization
System.Xml.XPath
System.Xml.Xsl
Microsoft.CSharp
Microsoft.JScript
Microsoft.VisualBasic
Microsoft.Vsa
Microsoft.Win32

Each namespace contains a number of classes that can instantiate objects. Altogether there are over 700 classes in Microsoft .NET. One library of namespaces, `mscorlib.dll`, contains the *core* classes from which many other common classes inherit behaviour and characteristics. The .NET namespaces to be found in `mscorlib.dll` are:

===== NAMESPACES (ñ) in `mscorlib.dll` Assembly =====

System
System.Collections
System.Configuration.Assemblies
System.Diagnostics
System.Diagnostics.SymbolStore
System.Globalization
System.IO
System.IO.IsolatedStorage
System.Reflection
System.Reflection.Emit
System.Resources
System.Runtime.CompilerServices
System.Runtime.InteropServices
System.Runtime.InteropServices.Expando
System.Runtime.Remoting
System.Runtime.Remoting.Activation
System.Runtime.Remoting.Channels
System.Runtime.Remoting.Contexts
System.Runtime.Remoting.Lifetime

System.Runtime.Remoting.Messaging
System.Runtime.Remoting.Metadata
System.Runtime.Remoting.Metadata.W3cXsd2001
System.Runtime.Remoting.Proxies
System.Runtime.Remoting.Services
System.Runtime.Serialization
System.Runtime.Serialization.Formatters
System.Runtime.Serialization.Formatters.Binary
System.Security
System.Security.Cryptography
System.Security.Cryptography.X509Certificates
System.Security.Permissions
System.Security.Policy
System.Security.Principal
System.Text
System.Threading
Microsoft.Win32

In the entire .NET Framework base class library there are over 30 assemblies, altogether containing over 100 namespaces. These 100 or so namespaces together contain over 700 classes. These 700 or so base classes (and their instantiated objects) are all immediately available to Dyalog.Net programmers.

The namespaces are organised hierarchically. This avoids name clashes and helps to approximately categorize the **function**al (or **object**ive) nature of the contents.

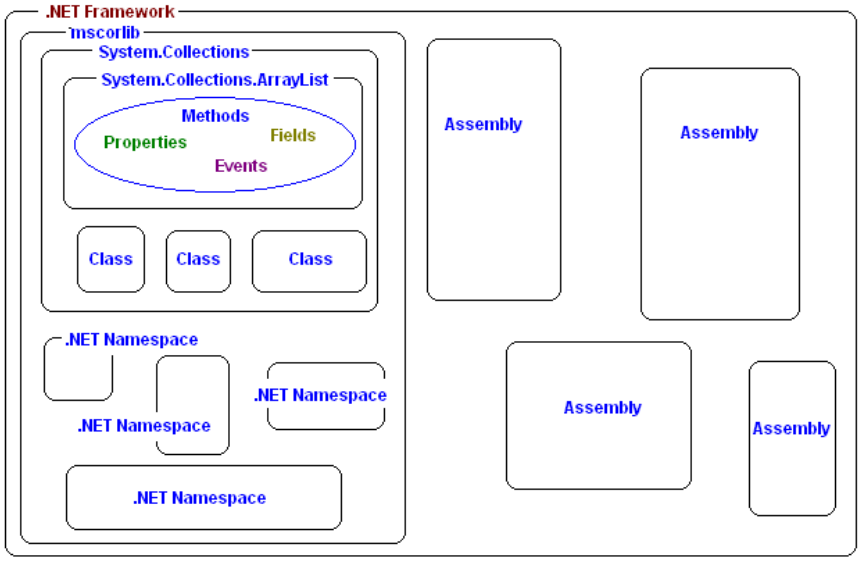
The `System.Collections` namespace in `mscorlib.dll`, for example, contains about 25 classes.

===== CLASSES (¢) in `System.Collections` Namespace =====

System.Collections.ArrayList
System.Collections.BitArray
System.Collections.CaseInsensitiveComparer
System.Collections.CaseInsensitiveHashCodeProvider
System.Collections.CollectionBase
System.Collections.Comparer
System.Collections.DictionaryBase
System.Collections.DictionaryEntry
System.Collections.Hashtable
System.Collections.ICollection
System.Collections.IComparer
System.Collections.IDictionary
System.Collections.IDictionaryEnumerator
System.Collections.IEnumerable
System.Collections.IEnumerator
System.Collections.IHashCodeProvider
System.Collections.IKeyComparer
System.Collections.IKeyedCollection
System.Collections.IList
System.Collections.KeyComparer
System.Collections.Queue
System.Collections.ReadOnlyCollectionBase
System.Collections.SortedList
System.Collections.Stack

Classes are used to create objects which have properties and methods. In the interests of application efficiency, some information about a class is not carried around with the class itself but is kept separately in its *MetaData* which is stored in the assembly's corresponding type library, or .TLB file. Typically, a class itself has no `GetMethods` method. Instead, you have to use the `GetType` method to instantiate a *reflection* of the original object in order to list the methods. The reflected object may be of 'dataType' `System.RuntimeType`. This object inherits the niladic `ToString` method that reports the dataType of the original object as, for example, `System.Collections.ArrayList` in the case of an instance of `ArrayList`. It also has `GetMethods`, `GetProperties` and `GetFields` methods that describe the members of the original instance of the `System.Collections.ArrayList` class. These methods access the MetaData which contains member names, dataTypes and method calling information.

Classes contain Methods, Properties, Fields and Events.



If we create an instance of the `System.Object` class then the default display form of the instance also happens to be `System.Object`, and the **dataType** of the instance is also called `System.Object`. However, these three names are logically distinct. The name of the class need not be identical to the dataType description of the instance, which need not be identical to the object display form. Thus, for example, the `GetType` method of an instance of the `ArrayList` class returns an instance of an object of the `System.Type` class whose dataType is reported as `System.RuntimeType` and whose display form is the full class name of the original instance, *viz* `System.Collections.ArrayList`. The `ToString` method is inherited by most objects. Its result is of type `System.String` and is returned to APL as a simple character vector. This is the default display form of an object and often spells out the

dataType (or type) of the object. In Dyalog version 11, the display form of an object may be set to any arbitrary character array via the new System Function ⎕DF.

17.1.2.1 What assembly do you think the System.ServiceProcess namespace is probably in? What namespace contains the System.ServiceProcess.ServiceControllerPermissionEntry class? (Version 11 has extended ⎕NL to facilitate the latter question.)

Hint: Use Google or consult http://msdn.microsoft.com/library/.

§§ 17.1.3 Using ⎕USING

Occasionally namespaces have members that are spread across multiple assemblies. In particular, the System namespace is spread over mscorlib.dll and system.dll. A .NET namespace is a logical design-time naming convenience, used mainly to organize classes in a single hierarchical structure. From the viewpoint of the runtime, there are no namespaces. Nevertheless, treating an assembly as a namespace receptacle is a welcome convenience. But it is therefore not easy programmatically to get a list of namespaces from an assembly name and so the namespace name together with its assembly origin must be specified before it can be utilized.

Dyalog APL contains a new system variable called ⎕USING. It is a bit like ⎕PATH which redirects APL to the location of some function that is in another APL namespace. ⎕USING (closely analogous to the using directive in C#) redirects APL to the location of some class which is in some particular .NET namespace which is in some particular assembly.

```
⎕USING←'ñ1,à1' 'ñ2,à2' ...        ⍝ Use .NET namespace ñi from assembly ài
```

In a new clear workspace ⎕USING≡0⍴⊂''. APL namespaces and programs inherit their local value of ⎕USING from the parent space, like ⎕PATH. Unlike ⎕PATH, the System Variable ⎕USING cannot be saved in the .DSE Session file.

⎕USING is a vector of character vectors (APL dataType VecCharVec) each character vector of which contains two parts separated by a comma. The first part specifies the case-sensitive name of a .NET namespace, and the second part specifies the name of a DLL file, thus

```
⎕USING←,⊂'NetNamespace,C:\..\Assembley.dll'
```

The primary assembly in the .NET framework, containing the most commonly used namespaces, is mscorlib.dll. The namespace in this assembly with the most commonly used classes is named the System namespace. Its content is exposed to APL by setting

```
⎕USING←,⊂'System,mscorlib.dll'
```

This particular namespace, and only this space, may be exposed by simply typing ⎕USING←'System', or even just ⎕USING←'' as long as you thereafter prefix everything with "System.". We shall use the verbose first form for clarity, and because all other namespaces have to be treated this way.

This establishes the basic starting point for Dyalog.Net. If, for example, classes in the namespace System.Windows.Forms are also to be invoked in Dyalog.Net code then the entry

```
⎕USING,←⊂'System.Windows.Forms,System.Windows.Forms.dll'
```

must be added to the local list of namespace paths.

17.1.3.1 Set ⎕USING in such a way that all the classes in .NET namespace System and .NET namespace System.Windows.Forms may be used directly from Dyalog APL.

§ 17.2 Exploring the .NET Interface

§§ 17.2.1 Examining Classes

In the impressive hierarchical structure of .NET there is another level above the assembly level, and another level beyond that as well as levels below the class level and the object hierarchies generated in a program. Therefore it is easy to get lost in the framework.

Processes⊃Application Pools⊃AppDomains⊃.NET Threads⊃Assemblies⊃.NET Namespaces⊃Classes⊃Objects⊃Members

We shall occasionally indicate whether something is an assembly by `à`, a namespace by `ñ` or a class by `¢`.

Consider `AppDomain` (`¢`) in `System` (`ñ`) in `mscorlib.dll` (`à`). This class represents an application domain, which is an isolated environment where applications execute.

```
      RSc←AppDomain                          ⍝ Class in System ñ in mscorlib.dll à
```

The class has a `CurrentDomain` property that gets the current application domain for the current thread object. (`Thread` (`¢`) in `System.Threading` (`ñ`) in `mscorlib.dll` (`à`) creates and controls a thread, sets its priority, and gets its status.)

```
      RSc←AppDomain.CurrentDomain    ⍝ Property of AppDomain ¢ returning a domain object
```

`CurrentDomain` returns an object of dataType `System.AppDomain`.

```
      RVec←AppDomain.CurrentDomain.GetAssemblies    ⍝ Property returns objects
```

This object has a `GetAssemblies` method that returns a vector of objects of dataType `System.Reflection.Assembly` representing the assemblies currently loaded in the application domain, each with a different display form containing a long string of individual information such as assembly name and version.

```
      TVec←AppDomain.CurrentDomain.GetAssemblies.GetType⊂⍬    ⍝ Vec of objects
```

The ubiquitous (inherited) niladic `GetType` method of `Assembly` `¢` in `System.Reflection` `ñ` in `mscorlib` `à` allows one to discover the dataType of the vector of assembly objects.

```
      VecRVec←AppDomain.CurrentDomain.GetAssemblies.GetTypes ⍝ Mirrors
```

The niladic `GetTypes` method of `Assembly` `¢` in `System.Reflection` `ñ` in `mscorlib` `à` returns a vector of vectors of objects (`VecRVec`) of dataType `System.Type` which describe all the dataTypes found in the assembly. So, for example,

```
      ⎕USING←'System,mscorlib.dll' ↵
            'System.Windows.Forms,System.Windows.Forms.dll'¶
      {(⍬⍴⍵)1⍴⍵}AppDomain.CurrentDomain.GetAssemblies
mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
bridge110, Version=11.0.0.0, Culture=neutral, PublicKeyToken=eb5ebc232de94dcf
dyalognet, Version=11.0.0.0, Culture=neutral, PublicKeyToken=eb5ebc232de94dcf
System.Windows.Forms, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c54e089
System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
System.Drawing, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
```

Note the appearance of two Dyalog-specific assemblies. Dyalog APL (version 11) communicates with the .NET framework via the Dyalog-distributed interface libraries, bridge110.dll and dyalognet.dll.

```
      ⍴¨AppDomain.CurrentDomain.GetAssemblies.GetTypes
 2320   178   6   2220   1783   294
```

```
      VecRVec←AppDomain.CurrentDomain.GetAssemblies.GetExportedTypes
```

The related `GetExportedTypes` method of `Assembly` `¢` returns six subsets of `System.Type` objects containing information about public classes in each of the six assemblies.

```
      ⍴¨AppDomain.CurrentDomain.GetAssemblies.GetExportedTypes
```

```
 1287   35   4   1053   872   186
      3↑⊃AppDomain.CurrentDomain.GetAssemblies.GetExportedTypes
System.Object  System.ICloneable  System.Collections.IEnumerable
```

```
      VRV←AppDomain.CurrentDomain.GetAssemblies.GetExportedTypes.Module
```

Each of these objects is of dataType `System.Type` and has a `Module` property that returns an object of dataType `System.Reflection.Module` whose default display form describes the assembly in question.

```
      3↑⊃AppDomain.CurrentDomain.GetAssemblies.GetExportedTypes.Module
CommonLanguageRuntimeLibrary  CommonLanguageRuntimeLibrary  CommonLanguageRuntimeLibrary
      ⎕CS⊃⊃AppDomain.CurrentDomain.GetAssemblies.GetExportedTypes.Module
      )NS
#.[System.AppDomain].[System.Reflection.Assembly].[System.RuntimeType].[System.Reflection.Module]
```

17.2.1.1 Objects of dataType `System.Type` have some properties that return simple string or Boolean values. Investigate the properties `IsClass`, `IsPublic`, `Name`, `Namespace` and `FullName`. Write an expression that returns a vector of objects of dataType `System.RuntimeType` which describe all the public classes in the first assembly in the current domain.

17.2.1.2 Check that the `System.Math` class is available for use when `⎕USING≡''`.

Hint: Look at the 'Name Category' (`⎕NC`) of `⊂'Math'`.

§§ 17.2.2 Examining Methods

In Microsoft .NET there are no functions or variables outside classes. Some methods in some .NET classes, such as those considered in §§17.2.1, may be used without explicitly creating instances. They have some public methods and properties accessible directly from the class namespace. `System.Math` is such a class. It is a container for some simple mathematical functions which may be called directly from the class. A few basic mathematical methods and fields are immediately available from the `Math` class, assuming that the `System` namespace is on the `⎕USING` path.

```
Math.((Sin 0.5)(Asin 0.5)) ↦ 1 ¯1○0.5 ↦ 0.4794.. 0.5235...
Math.((Log 0.5)(Exp 0.5)) ↦ (⊗0.5)(*0.5) ↦ ¯0.6931.. 1.648...
Math.Abs⍣¯5+⍳9 ↦ |¯5+⍳9 ↦ 4 3 2 1 0 1 2 3 4
Math.(E PI) ↦ (*1)(○1) ↦ 2.7182.. 3.1415...
```

Remember that `..` and `...` were defined as single symbols.

17.2.2.1 Load the MetaData for the `Math` class and try using some of the methods in the class.

Hint: Right click MetaData in WS Explorer to load the .TLB file
Make sure [View][Type Libraries] in WS Explorer is checked.

Most .NET classes are used by creating an instance of the class. In .NET, to examine programmatically the information associated with members of a class (methods, properties, etc..), you have to create an instance of the class and then use the `GetType` method to create an associated object of dataType `System.Type` and examine the results of its `GetMethods` and `GetProperties` methods. Information about the original object-generating class is extracted from the assembly Type Library file (.TLB) and is reported via a `Type` object instantiated from the `Type` class.

In typical OO style, the .NET Framework deals in classes and objects and methods and properties... (Events may be used as methods via `4⎕NQ` … and from version 10.1 onwards it is possible to declare events on `NetType` objects created with Dyalog APL.)

```
      RSc←⎕NEW ¢ …                    ⍝ Create (instantiate) an instance of the class
```

In Dyalog version 11 instances may be created from classes with the ⎕NEW system function. This monadic function takes a class as the first parameter of its argument, followed by a second parameter if required. For example, assuming the *System* namespace is visible,

```
      DT←⎕NEW DateTime (2006 7 4)
```

creates a new instance specifically relating to Independence Day of the *DateTime* class. The *MethodList* and *PropList* properties of this instance return a list of its methods and properties. The list of methods (with further dataType information) may be discovered from MetaData or, equivalently, from expressions such as

```
      2↑DT.GetType.GetMethods ⍬
System.DateTime Add(System.TimeSpan)  System.DateTime AddDays(Double)
```

17.2.2.2 Find the minimum and maximum dates allowed for a *DateTime* object.

Returning to the *ArrayList* class, given that ⎕USING is set appropriately, eg

```
      ⎕USING←'System,mscorlib.dll' ↵
             'System.Collections,System.Collections.dll'¶
```

an instance of the *ArrayList* class may be created by the statement

```
      AL←⎕NEW ArrayList
```

and the methods associated with this object may be obtained using its *MethodList* property. Alternatively, a more detailed description of each method may be obtained from the *GetMethods* method of the reflected object. Unsurprisingly (by now) the *GetMethods* method returns a vector of *objects*. The display form of each object (dataType *System.Reflection.RuntimeMethodInfo*) gives information about a method and its syntax.

```
      2↑AL.GetType.GetMethods ⍬
 Int32 get_Capacity()  Void set_Capacity(Int32)
```

Thus each class (object creation program inside its own space) contains a number of methods (functions). For example, the *System.Collections.ArrayList* class contains the following methods.

== METHODS in *System.Collections.ArrayList* Class ==

```
  System.Collections.ArrayList Adapter       (System.Collections.IList)
                          Void AddRange      (System.Collections.ICollection)
                         Int32 Add           (System.Object)
                         Int32 BinarySearch  (Int32, Int32, System.Object, System.Collections.IComparer)
                         Int32 BinarySearch  (System.Object, System.Collections.IComparer)
                         Int32 BinarySearch  (System.Object)
                          Void Clear         ()
                System.Object Clone         ()
                       Boolean Contains      (System.Object)
                          Void CopyTo        (Int32, System.Array, Int32, Int32)
                          Void CopyTo        (System.Array, Int32)
                          Void CopyTo        (System.Array)
                       Boolean Equals        (System.Object)
  System.Collections.ArrayList FixedSize     (System.Collections.ArrayList)
      System.Collections.IList FixedSize     (System.Collections.IList)
System.Collections.IEnumerator GetEnumerator (Int32, Int32)
System.Collections.IEnumerator GetEnumerator ()
                         Int32 GetHashCode   ()
  System.Collections.ArrayList GetRange      (Int32, Int32)
                   System.Type GetType       ()
                         Int32 IndexOf       (System.Object, Int32, Int32)
                         Int32 IndexOf       (System.Object, Int32)
                         Int32 IndexOf       (System.Object)
                          Void InsertRange   (Int32, System.Collections.ICollection)
                          Void Insert        (Int32, System.Object)
                         Int32 LastIndexOf   (System.Object, Int32, Int32)
                         Int32 LastIndexOf   (System.Object, Int32)
                         Int32 LastIndexOf   (System.Object)
  System.Collections.ArrayList ReadOnly      (System.Collections.ArrayList)
      System.Collections.IList ReadOnly      (System.Collections.IList)
```

```
                         Void RemoveAt      (Int32)
                         Void RemoveRange   (Int32, Int32)
                         Void Remove        (System.Object)
System.Collections.ArrayList Repeat        (System.Object, Int32)
                         Void Reverse       (Int32, Int32)
                         Void Reverse       ()
                         Void SetRange      (Int32, System.Collections.ICollection)
                         Void Sort          (Int32, Int32, System.Collections.IComparer)
                         Void Sort          (System.Collections.IComparer)
                         Void Sort          ()
System.Collections.ArrayList Synchronized  (System.Collections.ArrayList)
    System.Collections.IList Synchronized  (System.Collections.IList)
                System.Array ToArray       (System.Type)
             System.Object[] ToArray       ()
               System.String ToString      ()
                         Void TrimToSize    ()
                        Int32 get_Capacity  ()
                        Int32 get_Count     ()
                      Boolean get_IsFixedSize()
                      Boolean get_IsReadOnly()
                      Boolean get_IsSynchronized()
               System.Object get_Item       (Int32)
               System.Object get_SyncRoot   ()
                         Void set_Capacity  (Int32)
                         Void set_Item      (Int32, System.Object)
```

Names of methods are repeated in this list when dataTypes of argument parameters vary. Each line specifies one way in which the method may be called. The situation often arises in APL, but behind the scenes. Consider, for example, the dyadic primitive functions ⍳ and ~. Their arguments may be numeric or character. Under the covers, APL checks which and applies the required algorithm. Dyadic ⍴ can accommodate a right argument of many different dataTypes and, as of version 11, the arguments to ∧ and ∨ may be Boolean or integer. The arguments to ! may be integer or real, etc… Unlike APL (and VBScript ...), but like most mainstream low-level programming environments such as FORTRAN, VB and C#, .NET requires us to be more explicit about possible types of arguments to and results of methods. In fact it might be useful to include in APL documentation the explicit calling options associated with each primitive function. However, to specify the precise structure of every intermediate array in an APL application would be a thankless task.

⎕*NEW* is not the only way in which instances of classes may be created. Classes often have methods that return instances. For example, the *DateTime* class has a property (niladic function) called *Now*. *Now* returns an object of dataType *System.DateTime*.

17.2.2.3 Use the *IsLeapYear* method of an instance of the *DateTime* class created by the *Today* property to determine whether or not the year 3000 is a leap year.

§§ 17.2.3 Examining Properties

Normally, classes instantiate objects whose methods and properties are then used and changed. Consider, *DateTime* ¢, in *System* ñ, in *mscorlib.dll* à. This class has a property (niladic function) called *Now*. *Now* returns an object of dataType *System.DateTime*. (This is *dataType* as met in Module 0. In .NET there is a *Type* class whose purpose is to yield object dataType information.)

```
      DT←DateTime.Now
```

DT is an object with about 90 methods and 59 properties,

```
      ⍴¨DT.(MethodList PropList) ↳ (,90)(,59)
```

and whose dataType is discovered from the default display form (see ⎕*DF*) of the object returned by the (in this case niladic) *GetType* method (which is inherited ubiquitously from *System.Object* ¢).

```
      ⍕DT.GetType ↳ 'System.DateTime'
```

GetType returns an object of type *System.Type*. This object has an *Assembly* property whose display form contain the name of the assembly from which the current type has come.

```
      8↑⍕DT.GetType.Assembly ↳ 'mscorlib'
```

(Note that the *Type* class has a monadic *GetType* method that takes a *String* argument.)

The object returned by the *Assembly* property of the instance of the *Type* class representing the *DT*,

```
      Ass←DT.GetType.Assembly
```

itself has a *GetTypes* method that returns a vector of objects (*RVec*) representing all classes in the assembly.

```
      ⍴All←Ass.GetTypes ↳ 2373
```

Some of these classes are *Enumerations* – their principal purpose is to supply alternate names for values of an underlying primitive instance. All the objects in vector *All* have a niladic method, *IsEnum*, which returns a Boolean value indicating whether the corresponding object in *All* is an Enumeration.

```
      ⍴Enums←(All.IsEnum)/All ↳ 384
```

Taking the first such type, the *GetFields* method gets all the field names for the first Enum in the list.

```
      ⍴Enums[1].GetFields ⍬ ↳ 17
```

A field is a member of an object or class and represents a variable associated with the object or class. For example, if the 23rd enumeration is the *System.DayOfWeek* enumeration

```
      ⊢⍕Enums[23] ↳ 'System.DayOfWeek'
```

then the fourth field in the enumeration happens to be the *Tuesday* public static field.

```
      ⍕4⊃Enums[23].GetFields ⍬ ↳ 'System.DayOfWeek Tuesday'
```

17.2.3.1 Use an instance of *DateTime* ¢ to find the day of the week today.

17.2.3.2 Verify that the object returned by *Assembly* (of dataType *System.Reflection.Assembly*) has a *Location* property that returns a string containing the directory in which the assembly resides.

17.2.3.3 Find how many classes there are in *System.Web.dll*.

Hint: Create a new instance of, say, *System.Web.UI.Control* or *System.Web.Mail.MailMessage*.

As with object methods, the dataTypes associated with object properties make up an essential part of their specification. Therefore, in the interests of clarity, we talk about **dataType** – an adjective describing what type of data structure an object conforms to. You might think of it as a very sophisticated (albeit non-existent) version of monadic ⎕*DR*. For simple objects like the number 9, the dataType might be *System.Int16*. *System.Int16* is actually a *value type* – a light-weight class which is treated as a value rather than a full-blown class in most situations. On the other hand, the dataType of a more complicated object such as an instance of the *ArrayList* class is usually described in the same words as the namespace-qualified name of the class itself - *System.Collections.ArrayList*. *System.Collections.ArrayList* is called a *reference type* – a full-blown object passed around by reference to it (shallow copy) rather than by making a copy (deep copy).

= PROPERTIES in *System.Collections.ArrayList* Class =

```
Int32                Capacity
Int32                Count
Boolean              IsFixedSize
Boolean              IsReadOnly
Boolean              IsSynchronized
System.Object        SyncRoot
System.Object        Item                 [Int32]
```

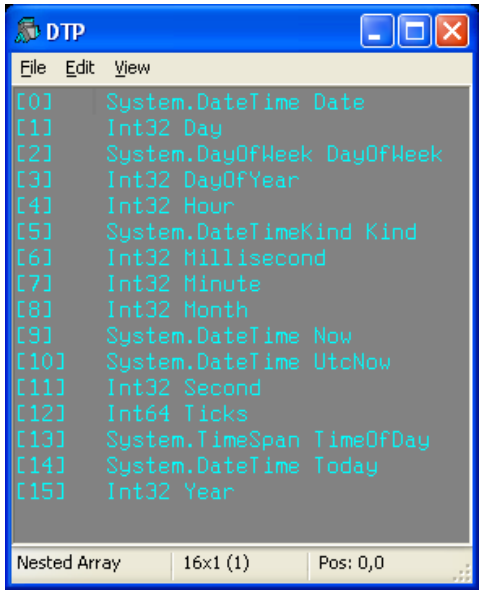
Note that the *Item* property looks different from the others - it seems to take an argument! *Properties* are often like shared variables and are accessed through get_.. and set_.. control functions. *Fields* are more like simple APL variables. *Methods* are like locked monadic or niladic functions – sometimes

cameleon-like, ie either! *Events* are treated like methods in Dyalog.Net and cannot as yet be assigned callback functions.

The `GetProperties` method returns objects representing the properties of a `DateTime` object, with their dataTypes.

```
      ⍕3↑⊃DT.GetType.GetProperties ⍬ ↳ 'System.DayOfWeek DayOfWeek'
      ⍴DTP←{(⍵⍴⍴⍵)1⍴⍵}DT.GetType.GetProperties ⍬ ↳ 16 1
```

Thus the `DayOfWeek` property returns an object of type `System.DayOfWeek`, whose display form is the actual day of the week relating to the instance date, whereas `Day` contains a simple integer day number (Int32 => `ZSc`) relating to the instance date.




```
      DT.DayOfWeek.ToString ⍬ ⍝ CVec
Tuesday
      ⍕DT.DayOfWeek           ⍝ CVec
Tuesday
      ⎕FMT DT.DayOfWeek       ⍝ CMat
Tuesday
      DT.DayOfWeek            ⍝ RSc
Tuesday
```

17.2.3.4 With a new instance of `DirectoryInfo` ¢ from `System.IO` ñ, call the `GetFiles` method with argument `'*.*'` to get (objects representing) all the files in a given DOS directory. Then read the `Name` and `CreationTime` properties of the vector of instances, dataType `System.IO.FileInfo`, to access the file details.

§ 17.3 Digging into .NET

§§ 17.3.1 Windows Forms

Now that we know how to create instances of .NET classes by `⎕USING` the `System` namespace

```
      (⎕NEW DateTime(3↑⎕TS)).ToString ⍬ ↳ '11/04/2006 00:00:00'
```

and understand that objects may be created with different *constructors*

```
      (⎕NEW DateTime(6↑⎕TS)).ToString ⍬ ↳ '11/04/2006 10:07:21'
```

and can recognise some different categories of objects

```
      DI←⎕NEW IO.DirectoryInfo(⊂'C:\windows')⋄⎕NC⊂'DI' ↳ 9.3
```

and appreciate some of the idiosyncrasies/niceties of .NET and the Dyalog calling syntax

```
      (⊃DI.(GetFiles⊂'*.exe')).FullName ↳ 'C:\windows\alcrmv.exe'
```

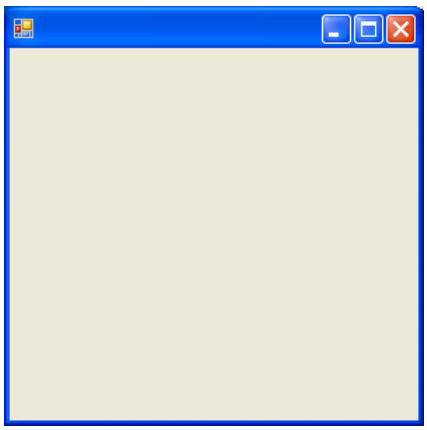
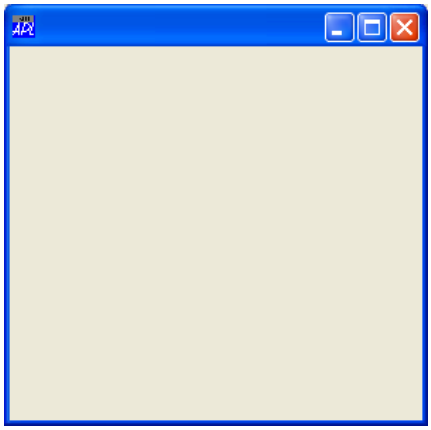
and know how to read syntax from MetaData (but beware of `WS FULL` for this large assembly), we should be able to build applications based on .NET classes. The Dyalog supplied workspaces in ..\samples\winforms\ give some excellent examples as does the *Dyalog.Net Interface Guide*. Here we just skim the surface.

In order to create a Form in .NET it is necessary to access `System.Windows.Forms` ñ.

```
      ⎕USING←'System.Windows.Forms,System.Windows.Forms.dll'
```

We can then immediately create a `Form`, and make it `Visible`

```
      F←⎕NEW Form ⋄ F.Visible←1
```


compared with `'F2'⎕WC'Form'`


The only visible difference is the default Icon. The differences between Windows GUI and Windows .NET Forms begin to diverge from this close (guess why) start.

The GUI *Form* has an *OnTop* property which becomes *TopMost* in .NET. The *Caption* property becomes the *Text* property and the *Size* and *Posn* properties are each a combination of two properties:

```
F.(Height Width)←200 300 ⍝ was Size
F.(Top Left)←100 200     ⍝ was Posn
```

There is also a *Location* property which has dataType *System.Drawing.Size*. This is more like the old *Posn* property in the sense that it accepts the *Top* and the *Left* coordinates in one gollop. But in order to achieve this we have to create an instance of *System.Drawing.Size* ¢ via *Point* ¢. One way of constructing an instance of *System.Drawing.Point* ¢ is with a *⎕NEW* object parameter of dataType *System.Drawing.Size* which is what we are trying to create in the first place. Luckily there is also a constructor with (Int32, Int32) for X and Y.

```
⎕USING,←⊂'System.Drawing,System.Drawing.dll'
F.Location
Pt←⎕NEW Point (10 10)
F.Location←Pt             ⍝ was Posn
```

17.3.1.1 Set the *Form* size in one statement using the *ClientSize* property.

A *Button* object, an instance of *Button* ¢, of dataType *System.Windows.Forms.Button* and display form *System.Windows.Forms.Button, Text:*..., may be created by

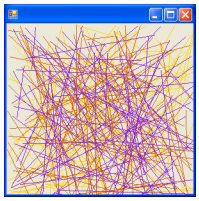
```
B←⎕NEW Button
```

.NET does not really have the concept of namespace hierarchies. For naming convenience, namespaces appear to be arranged hierarchically, but in fact, if a namespace called *A.B.C.D* exists in .NET then this does **not** imply that any of *A*, *A.B* or *A.B.C* has to exist. The .NET way of assigning a parent-child relationship to the *Form* and the *Button* is by way of the *Controls* property of a *Form* which supplies an instance of the class *System.Windows.Forms.Control* which has an *Add* method.

```
⊃F.Controls.MethodList ↳ 'Add'
```

This method takes an object as its argument and adds it to the collection of controls comprising the children of *F*.

```
F.Controls.Add B
```



17.3.1.2 Trace the *∇scribble∇* function below and use MetaData to verify the comments.

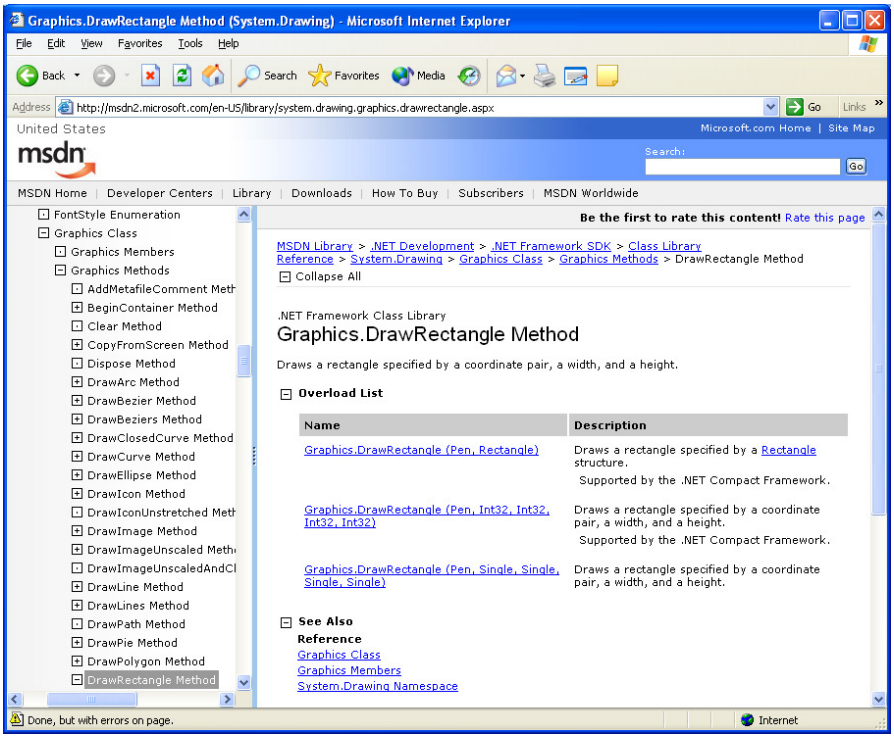
```
     ∇ scribble;F;GR;PB                                  ⍝ Scribble lines on a Form.
[1]    ⎕USING←'System.Windows.Forms,System.Windows.Forms.dll' ⍝ Initiate System.Windows.Forms
[2]    ⎕USING,←⊂'System.Drawing,System.Drawing.dll'     ⍝ and System.Drawing namespaces.
[3]    F←⎕NEW Form                                      ⍝ Create inst.of System.Windows.Forms.Form ¢
[4]    F.Visible←1                                      ⍝ and make it visible.
[5]    PB←⎕NEW PictureBox                               ⍝ Create instance of ..Forms.PictureBox ¢
[6]    PB.Size←F.Size                                   ⍝ and make Size same as Form.
[7]    F.Controls.Add PB                                ⍝ Add control to Form.
[8]    GR←PB.CreateGraphics                             ⍝ Create Graphics obj with CreateGraphics m
[9]    GR.{DrawLine(Pens.Gold,⍵)}¨↓?100 4⍴300           ⍝ Run DrawLine m of ..Drawing.Graphics ¢
[10]   GR.{DrawLine(Pens.Chocolate,⍵)}¨↓?100 4⍴300     ⍝ using Chocolate p of System.Drawing.Pens ¢
[11]   GR.{DrawLine(Pens.BlueViolet,⍵)}¨↓?100 4⍴300    ⍝ to create a System.Drawing.Pen object.
[12]   F.Close                                          ⍝ Close the Form.
     ∇
```

What class owns the `Add` method?

17.3.1.3 Trace the `∇_Grid2∇` function in supplied workspace ..\samples\winforms\winforms.dws.

An `Edit` object becomes a `TextBox` class in Dyalog.Net, a `Grid` object becomes a `DataGrid` class but a `Label` is still called a `Label` and a `StatusBar` is still called a `StatusBar`. ☺

The definitive guide to .NET framework class libraries is MSDN (MicroSoft Dot Net), available on-line or as a download.



17.3.1.4 Assign a simple `∇show∇` function to the `onClick` property of Button `B`. Trace the line

```
Application.Run F
```

and compare with `⎕DQ`. Notice that the message argument is a 2-vector of *objects*.

17.3.1.5 Trace the .NET-laced `∇RUN∇` function in workspace ..\samples\winforms\gdiplus.dws, watching out for the `Timer` ¢ instance.

Hint: Trace `Application.Run Form1` rather than using the Session implicit `⎕DQ`.

17.3.1.6 Play a pretty game of Tetris in workspace ..\samples\winforms\tetris.dws, then trace the *exhibition-quality* Dyalog.Net code. Take care with the `onTick` event and multi-threading when tracing.

§§ 17.3.2 Communications

Communications is a big word these days. Once upon a time it might have covered simply the notion of messages to the user, as in

```
      ⎕USING←'System.Windows.Forms,System.Windows.Forms.dll'
      MessageBox.Show⊂'This in itself is the message.'
```

Press OK.

```
OK
```

'Communications' might have included requests for information about the local environment.

```
      ⎕USING←'System'
      ⍎IO.Directory.GetCurrentDirectory ↳ 'C:\Dyalog\DWS'
```

which was covered by using `⎕NA'kernel32|GetCurrentDirectoryA U4 >0T'` or before that from `⎕CMD'cd'`. Further 'communications' with 'the system' could be exemplified with

```
      ⍎IO.Directory.(GetParent GetCurrentDirectory) ↳ 'C:\Dyalog'
```

or

```
      IO.Directory.(GetDirectoryRoot GetCurrentDirectory) ↳ 'C:\'
```

or

```
      Environment.CurrentDirectory ↳ 'C:\Dyalog\DWS'
```

as above (except it does not return an object and it can be assigned), or

```
      Environment.CommandLine
"C:\Program Files\Dyalog\Dyalog APL 11.0\dyalog.exe"
```

which is the same as `#.GetCommandLine` in Dyalog GUI terms, or

```
      Environment.UserName ↳ 'ADENNY'
```

giving the same as `⎕AN`.

But there are many other properties and methods in the Framework Class Library that give information not readily available in raw APL – although probably accessible via `⎕NA`.

```
      Environment.MachineName ↳ 'JCM5032483'
      Environment.(OSVersion Version)
 Microsoft Windows NT 5.1.2600 Service Pack 1  2.0.50215.44
      Environment.GetLogicalDrives
 A:\  C:\  D:\  K:\  L:\  M:\  O:\  P:\  Q:\  R:\  U:\  X:\  Y:\  Z:\
```

Sometimes APL gives information not directly available from Microsoft .NET such as the inverse of a matrix, and sometimes APL just is not that concerned:

```
        ↑(Int16 Int32 Int64).(MinValue MaxValue)
       ¯32768                 32767
  ¯2147483648           2147483647
-9223372036854775808  9223372036854775807
```

Once upon a time extraction of data from a file system or database might have been classed as communications. In .NET the `System.Data`… namespaces contain facilities for ODBC, SQL…

But we all know that these days communications is bigger and wider than all that. It means radio, TV, postal services and transport. But in particular for computing it means *eMail* and the *Internet*.

The .NET Framework has a namespace in the base class library `System.dll` called `System.Net` and another called `System.Net.Sockets`. These cover most of the TCP/IP functionality available through Dyalog `TCPSocket` objects. For example the `System.Net.Sockets.Socket` class has a `Send` method which is similar to the `TCPSend` method in the Dyalog GUI. But the `System.Net` namespace has a lot more functionality. For example, there are classes relating to Authentication, Cookie control and HTTP handling and a namespace `System.Net.Security` relating to security issues. The framework class library also has as a number of other assemblies, such as `System.Web.dll`, entirely devoted to Internet issues and `System.Web.Mail` relating to email issues.

Consider, for example, the `TCPGotAddr` event of `TCPSocket` objects in the Dyalog GUI. This event may be used to report the IP address associated with a host name. Alternatively, the DOS command C:\WINDOWS\system32\nslookup.exe may be used to find the same information. This information is retrieved via a Domain Name Server (DNS) located somewhere on the visible network. It is the job of this server to maintain an up-to-date list of site names (domains) and their IP addresses.

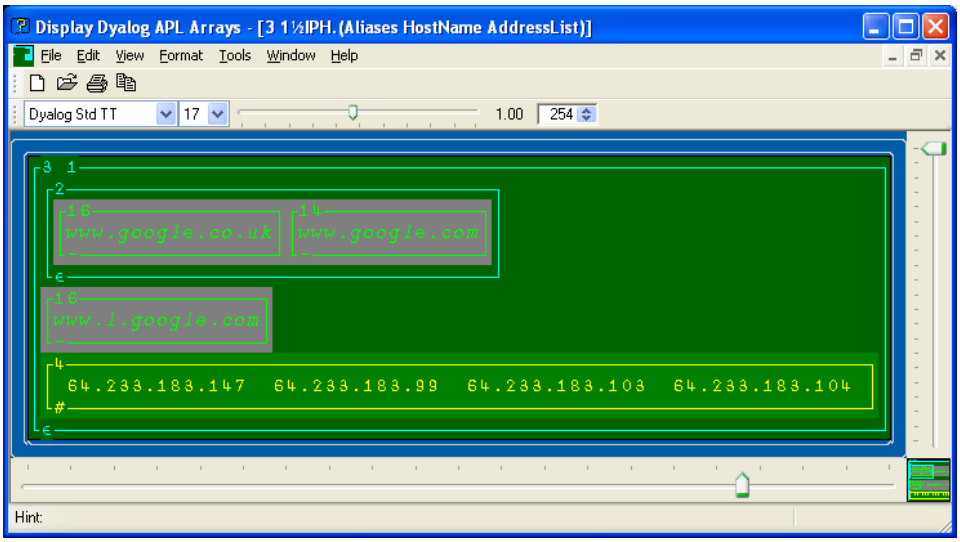
In .NET the `System.Net.Dns` class is a method called `BeginGetHostByName` that takes a URL parameter and returns asynchronously, having made contact with a DNS server, an object containing information about the URL in question.

```
      ⎕USING←'System,system.dll'
      URL←'www.google.co.uk'
      RAR←System.Net.Dns.BeginGetHostByName URL(⎕NS'')(⎕NS'')
```

`RAR` is a namespace of dataType `System.Net.Dns+ResolveAsyncResult`. This object is used as an argument to the `System.Net.Dns` class method `EndGetHostByName`. When the URL has been resolved, this method returns an object of dataType `System.Net.IPHostEntry`. An instance of `System.Net.IPHostEntry` ¢ has properties `Aliases`, `HostName` and `AddressList` that give some basic information about the domain, including all the related names and addresses.

```
      IPH←Net.Dns.EndGetHostByName RAR
      3 1⍴IPH.(Aliases HostName AddressList)
 www.google.co.uk  www.google.com
www.l.google.com
 64.233.183.103  64.233.183.104  64.233.183.147  64.233.183.99
```

`Aliases` is of dataType `System.String[]` which in APL translates into VecCVec. `HostName` is of dataType `System.String` which in APL terms translates into CVec, and `AddressList` is of dataType `System.Net.IPAddress[]` which returns a vector of instances of the `System.Net.IPAddress` class.



Based on this functionality, Stefano Lanzavecchia has given the dotnet@dyalog.com group, amongst many other treasures, the following function which determines the IP addresses of all 3-letter .com domains.

```
     ∇ r←ss1;step;name;list;n;⎕USING;blocks;b;x;t
[1]    step←300    ⍝ blocksize
[2]    ⎕USING←'' 'System.dll'
[3]    name←{'www.',⍵,'.com'}
[4]    list←name¨,⊃∘.,/⎕A ⎕A ⎕A
[5]    r←list,[1.5]⊂''
[6]    blocks←((⍴list)⍴(step↑1))⊂⍳⍴list
[7]    t←⎕AI[3]
[8]    :For b :In blocks
[9]       ⎕←'n: '(⊃b)'/'(⍴list)
[10]       ⎕←'elapsed: '(0.001×⎕AI[3]-t)'estimated: '(0.001×(⍴list)×(⎕AI[3]-t)÷⊃b)
[11]       x←{System.Net.Dns.BeginGetHostByName ⍵(⎕NS'')(⎕NS'')}¨list[b]
[12]       r[b;2]←{0::⍬ ⋄ (System.Net.Dns.EndGetHostByName ⍵).AddressList.ToString}¨x
[13]   :EndFor
     ∇
```

17.3.2.1 Create a new instance of `MailMessage` ¢ in `System.Web.Mail` ñ. Assign suitable values to the `To`, `From`, `Subject` and `Body` properties of the object. Run the `Send` method belonging to `System.Web.Mail.SmtpMail` ¢ with the `MailMessage` object as its argument.

Hint: See the *Dyalog.Net Interface Guide* p20.
Note1: Lines in the `Body` string end in `⎕AV[3]` (LF) and the `Body` is terminated with `⎕AV[4]` (CR).
Note2: You might need to set the name of your SMTP relay mail server via `SmtpMail.SmtpServer`.

17.3.2.2 Use .NET to retrieve the string contents of a URL web site. Run the `Create` method of `System.Net.WebRequest` ¢ from `System.Net` ñ in `system.dll` à with an argument of some URL string such as 'http://www.dyalog.com'. The `GetResponse` method returns an object of dataType `System.Net.WebResponse`. This object has a method called `GetResponseStream` that returns an object of dataType `System.IO.Stream`. This object may then be used as a parameter when creating a new instance of `System.IO.StreamReader` ¢. Finally the `ReadToEnd` method of this instance returns a string containing the contents of the URL home page.

Hint: See the *Dyalog.Net Interface Guide* p21.
Note: You might need to create a suitable instance of `WebProxy` ¢ and assign it to the `Proxy` property.

§§ 17.3.3 Generalising APL Primitives

Many facilities in Dyalog Version 9 are replicated in Dyalog.Net. Super-succession has occurred many times before and in many different contexts. If Microsoft is correct then .NET is here to stay for the foreseeable future. It will replace the underlying methodology of the Dyalog GUI, Dyalog TCPSocket objects, APL Threads, `⎕NA`… , perhaps unnoticeably; like `⎕TS` changed is clock and `⎕AN` changed its data source.

Even primitive APL functions may be supplemented with .NET methods or replaced by .NET equivalents although the current mathematical offerings of .NET are far less extensive than those in APL 1. One might hope, for example, that complex arithmetic will be gifted to Dyalog APL through .NET although neither camp seems particularly interested. However, it is the simple basic grammar of APL and not the underlying algorithms which distinguish it from all the other less elegant languages.

Although APL originated as a notation for succinctly describing algorithms and was only later implemented as a computer language, it owes much to other computer languages in its later generations. For example the concepts of file systems, nested arrays, error trapping, control structures, multi-threading and the modern GUI interface are derived directly from other computer languages.

Error trapping in Dyalog.Net follows the OO style. An error encountered within .NET signals an error number `90`. This error may be trapped in the usual way with `⎕TRAP` or `:Trap`. `⎕DM` contains the usual diagnostic message but more details may be found from the properties (and display form) from the new system object, `⎕EXCEPTION`, which is an instance of `System.Exception ¢`.

17.3.3.1 Force an error in .NET and examine the properties of the `⎕EXCEPTION` object.

Many people are developing .NET classes to cover various areas of computing which are not found in the framework library, *eg* http://www.extremeoptimization.com/ or http://www.strangelights.com/fsharp/. Some of these extensions might one day be an intrinsic part of Dyalog APL. Microsoft .NET itself introduces some methods which extend basic arithmetic and Boolean functions to `DateTime` objects.

The meaning of adding days to dates or determining whether one date is greater than (after) another are intuitively clear and so .NET introduces methods such as `op_Addition` and `op_GreaterThan` that apply directly to instances of `System.DateTime ¢` and `System.TimeSpan ¢`. Dyalog has incorporated some of these 'operators' into primitive functions.

17.3.3.2 Experiment with APL primitives `+ - = ≠ > ≥ < ≤` as applied to `DateTime` and `TimeSpan` objects and compare with corresponding methods in these classes. Dyalog goes further and provides natural extensions to primitives `⍋ ⍒ ⌊ ⌈`. Experiment with derived functions (such as `⌈/`) as applied to dates.

Hint: See the *Dyalog.Net Interface Guide* p16.

17.3.3.3 Consider joining the dotnet@dyalog.com mailbox group and ask for the next module.