Differences between the .NET Core interface and the .NET Framework interface:

- .NET Framework only works on Microsoft Windows; .NET Core is cross-platform.

- The .NET Framework interface works on both Unicode and Classic editions of Dyalog; the .NET Core interface only works on Unicode editions of Dyalog.

- Features that are in the .NET Framework Interface Guide but are not included in the .NET Core Interface Guide should not be assumed to work.

- The first release of the .NET Core interface has a very restricted threading model in which all calls to .NET members are made on the same thread. The main implications of this are:
    - .NET calls made on an APL thread other than thread 0 will be executed on the same (operating system) thread as all other .NET calls.
    - If a .NET object (for example, the FileSystemWatcher) raises an event on a thread in a pool, then the APL callback will run on a new APL thread but all calls back to the .NET object will be made on the interpreter's main thread.

  This thread model will definitely change as the .NET Core interface evolves.

- On Microsoft Windows it is not currently possible to use both the .NET Core interface and the .NET Framework interface at the same time. This means that, on Windows, the .NET Core interface has to be explicitly enabled. This is done by setting DYALOG_NETCORE=1 (the default is 0 for backwards compatibility). On Linux and macOS, DYALOG_NETCORE defaults to 1, so needs to be explicitly set to 0 in the (unlikely) event that you want to disable the .NET Core interface.

- The .NET Core interface automatically converts Tuples (https://docs.microsoft.com/en-us/dotnet/csharp/tuples) to and from nested arrays.

- The .NET Core interface automatically "associates" extension methods (https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods) with relevant classes.
    - When the .NET Core interface loads an assembly, it identifies all relevant extension methods and add them to its internal cache of members. These extension methods will then appear as any other method in the relevant objects.

- The .NET Core interface handles `⎕USING` differently to the way the .NET Framework interface does. If the assembly specification in a `⎕USING` statement does not end in "**.dll**" then the assembly is loaded by name (not by filename). This means that .NET Core can use its built-in binding mechanisms to determine from where to load the assembly. This is useful as in many cases it removes the need to know where an assembly is located. (implementation-wise we call Assembly.Load() not Assembly.LoadFrom().: