XSHARP : XBASE AND .NET TO THE MAX

Fabrice Foray
fabrice@xsharp.eu

Southwest Fox 2019 – Phoenix AZ

Overview

- FoxPro Dialect
  - Class Definition
  - DBF Console App
- .NET Extensions to xBase
  - Type Check Expression, Var/Local Implied, Dynamic
  - Const, Default, Nullable Type access, Yield
  - Checked/UnChecked/Using/Scope
  - Lock/Fixed/UnSafe
  - Switch Statements
  - Try/Catch/Finally
  - Anonymous & Lambda Expression
- Extensions
  - Extension Methods
  - Generics
  - LINQ
- Async / Await

FoxPro Dialect

FoxPro Dialect

- Class Definition Sample
  - `DEFINE CLASS` / `ENDDEFINE`

- `USE` / `SCAN / APPEND`
  - Standard FoxPro Commands

Memory and Vars

Memory and Vars

- Type Check
  - IS / ASTYPE
- VAR / LOCAL IMPLIED
  - Compile-Time resolution
- DYNAMIC
  - A little bit like USUALs…

Memory and Vars

- DEFAULT
  - `oPerson:FirstName DEFAULT "First"`
    - Set a default value if the left element is NULL
- ?
  - `oEmptyPerson?:FirstName`
    - Conditional Access, no crash even if oEmptyPerson is NULL
- CONST / INITONLY
    - Change the way Fields can be set and modified

    - Let's see a Sample

Yield

- Yield
  - Any enumerator
  - ForEach or LINQ
  - Let’s see a Sample

Statement Blocks

BEGIN

- BEGIN CHECKED
  - The **checked** keyword is used to explicitly enable overflow checking for integral-type arithmetic operations and conversions.

- BEGIN UNCHECKED
  - The **unchecked** keyword can be used to prevent overflow checking.

BEGIN

- BEGIN USING
  - Provides a convenient syntax that ensures the correct use of **IDisposable** objects.
  - The **using** statement calls the Dispose method, and it also causes the object itself to go out of scope as soon as Dispose is called. Within the using block, the object is read-only and cannot be modified or reassigned.

BEGIN

- BEGIN SCOPE
  - Define a block statement
  - All defined LOCALs in that block, only exist in that block
  - Let's see a Sample

BEGIN

- BEGIN FIXED
  - statement prevents the garbage collector from relocating a movable variable
  - sets a pointer to a managed variable and "pins" that variable during the execution of the statement.

BEGIN

- BEGIN LOCK
  - Marks a statement block as a critical section
  - Ensures that another thread does not enter that block.
    - If so, it will wait, until the object is released at END
  - More to come with Async

SWITCH

- SWITCH … CASE
  - DO CASE replacement
    - Except that the expression is only evaluated *once*
  - More like Switch/Case in C#, C++
    - No fall-through
      - So no Break needed
  - Ok, let's see

Try Catch

- Try Catch Finally
  - Open a statement block, which specify handlers for different exceptions.
  - And, optionally, an exit statement block whatever the reason of the exit
  - Let's see a Sample

Anonymous & Lambda

Anonymous Methods

- Delegate … reminder
  - It is a Reference type, like a Method signature

```
DELEGATE DoubleDelegate( d AS REAL8) AS REAL8
```

  - Now, DoubleDelegate is a Type
    - `LOCAL namedMethod AS DoubleDelegate`
  - If MultiplyBy2 is a Function with the right prototype
    - namedMethod := MultiplyBy2
  - And you can use the DELEGATE for a Function call
    - namedMethod(8.0)

Lambda Expression

- So, and Anonymously ?
  - Define the DELEGATE
  - Define the reference holder
  - Write that code, a bit like a CodeBlock

- Very usefull with in List<T> for example

Extensions

Extension Methods

- Enable you to "add" methods to existing types without creating a new derived type.

- Static Class
  - Add Static Method
  - The first parameter specifies which type the method operates on, and the parameter is preceded by the SELF modifier.

Generics

- Generics
  - Usage
    - List<String>
  - Definition
    - MyArray<T>
  - Constraints

Struct	Only Value Type
Class	Only Reference Type
New()	parameter-less constructor

  - Let's go for a Demo !

LINQ

- LINQ
  - Language-Integrated Query
    - Definition

  - The full LINQ feature set will be supported by X#:
    - FROM
    - LET
    - WHERE
    - JOIN
    - ORDER BY
    - EQUALS
    - INTO
  - A sample is better than a long talk….

Async / Await

Async

- Async
  - Potentially blocking operation ?
    - Web Access (HttpClient, …)
    - File Access (StreamWriter, XMLReader, …)
    - Media manipulation (BitmapEncoder, MediaCapture, …)
  - How to ?
    - Threads !
    - BackgroundWorkers

Async

- Async
  - How to ?
    - ASYNC keyword in the method signature
      - By convention, ends with an "Async" suffix.
    - The return type is a Task, or Void
- Await
  - The ASYNC method can't continue past that point until the process is complete.
  - Control returns immediately to the caller of the async method.

THANKS FOR YOUR ATTENTION…☺

Fabrice Foray
fabrice@xsharp.eu