

Lists, Generics, Enumerators, Enumerations, Serialization



DIY: Closing the OO Gap



Consultingwerk Ltd.

- Independent IT consulting organization
- Focusing on **OpenEdge** and **related technology**
- Located in Cologne, Germany
- Customers in Europe, North America, Australia and South Africa
- Vendor of tools and consulting programs
- 25 years of Progress experience (V5 ... OE11)
- Specialized in GUI for .NET, OO, Software Architecture, Application Integration

Warning!

- If you believe include files should not be used with class files at all, you are probably in the wrong presentation
- You will see how include files can be used to enhance OO ABL usability and help focus on the real problem



Agenda

- **Introduction – OO ABL**
- OO ABL's missing features
- Lists of Objects
- Generic Lists of Objects
- List Enumerators
- Enumerations
- Object Serialization



Introduction - OO ABL

- OOABL is not a separate language, it's a feature of the ABL (aka 4GL available since 1982)
- OO ABL and procedural cooperate
- Procedures can
 - create Object instances
 - Invoke methods of Objects
 - Get/Set properties
 - Subscribe to events from classes/events
- Procedures can use classes as parameters

```
DEFINE INPUT PARAMETER poParameter AS Sample01.CustomerReportParameter NO-UNDO .
```

OO ABL Timeline

- **10.1A first implementation**, classes, objects, methods
- **10.1B Interfaces**, USING statement, properties
- **10.1C Static members, structured error-handling**, properties in Interfaces, DYNAMIC-NEW
- **10.2A GUI for .NET, garbage collection** for objects (anything reference by a WIDGET-HANDLE or COM-HANDLE is not an object)
- **10.2B Abstract classes**, abstract members, .NET generic type definition, **strong typed events**, reflection part I

OO ABL Timeline

- 11.0 **DYNAMIC-PROPERTY**
- 11.0 **JSON Object Model** as classes
- 11.2 REST Adapter can call into class directly, singleton-run
- 11.4 **Serialization** between ABL Client and AppServer
- 11.4 Ability to **THROW** errors from AppServer to client
- 11.6, expected later in 2015
https://community.progress.com/community_groups/openeledge_general/m/documents/1823.aspx
- Generally new language features are more often added as objects and not as new statements

OO ABL and AppServer

- The AppServer protocol only speaks procedural
- Every client needs to call into procedures (except the REST Adapter)
- Activate, Deactivate, Connect, Disconnect **procedures**
- AppServer may use objects from there on
- Can only pass an object as a parameter between AppServer and ABL Client from 11.4 on

OO ABL and AppServer

- Can't remotely call into an object like we can into a remote persistent procedure (not recommended anyway, but possible, unfortunately used a lot)
- OO ABL and AppServer limitation typically solved by OERA patterns:
 - Service Adapter on the client
 - Service Interface on the AppServer

Agenda

- Introduction – OO ABL
- **OO ABL's missing features**
- Lists of Objects
- Generic Lists of Objects
- List Enumerators
- Enumerations
- Object Serialization



OO ABL's missing features

- Lists, Dictionaries
- Generic Lists and Dictionaries
- LINQ
- Enums
- Reflection: Ability to query methods and properties of an object or a class
- Ability to query a classes, properties, methods annotations at runtime
- Serialization for non ABL clients
- Ability to store objects (structures) in DB

Risk with OO ABL's missing features

- Poor OO code ..., too many workarounds
- ABL may be seen as a legacy code only language
- Difficulty adopting patterns or sample code from other OO languages to ABL
- Acceptance problems of OO ABL at young developers
- Modernization decisions may be based on missing OO features, ignoring the strength of the ABL in so many other aspects

Agenda

- Introduction – OO ABL
- OO ABL's missing features
- **Lists of Objects**
- Generic Lists of Objects
- List Enumerators
- Enumerations
- Object Serialization



Lists of Objects

- ABL variable may reference a single object instance at a time
- ABL property may reference a single object at a time

```
/* ***** Definitions ***** */  
DEFINE VARIABLE oCustomer AS Samples.Customer.Customer NO-UNDO .  
/* ***** Main Block ***** */  
FIND FIRST Customer .  
oCustomer = NEW Samples.Customer.Customer (BUFFER Customer:HANDLE) .
```

```
USING Progress.Lang.* FROM PROPATH .  
USING Samples.Customer.* FROM PROPATH .  
USING Consultingwerk.Assertion.* FROM PROPATH.
```

```
CLASS Samples.Customer.Customer:
```

```
/*-----  
Purpose: References the address of the customer  
Notes:  
-----
```

```
DEFINE PUBLIC PROPERTY Address AS Address NO-UNDO  
GET.  
SET.
```

List of Objects

- What if we are successful and win a second customer? Or a third? Or more?
- What if a customer may have multiple addresses?
- We can use arrays (EXTENT's) of Objects

Referencing objects in an Array

```
/* ***** Definitions ***** */  
  
USING Samples.Customer.* FROM PROPATH.  
  
DEFINE VARIABLE oCustomers AS Samples.Customer.Customer NO-UNDO EXTENT .  
DEFINE VARIABLE i          AS INTEGER                      NO-UNDO .  
  
DEFINE QUERY qCustomer FOR Customer .  
  
/* ***** Main Block ***** */  
  
OPEN QUERY qCustomer  
  PRESELECT EACH Customer WHERE Customer.CustNum < 1000  
                                AND Customer.SalesRep = "HXM" .  
  
EXTENT (oCustomers) = QUERY qCustomer:NUM-RESULTS .  
  
DO WHILE QUERY qCustomer:GET-NEXT ():  
  i = i + 1 .  
  
  oCustomers[i] = NEW Customer (BUFFER Customer:Handle) .  
  
END.
```

Demo

- Populating Array of Customers

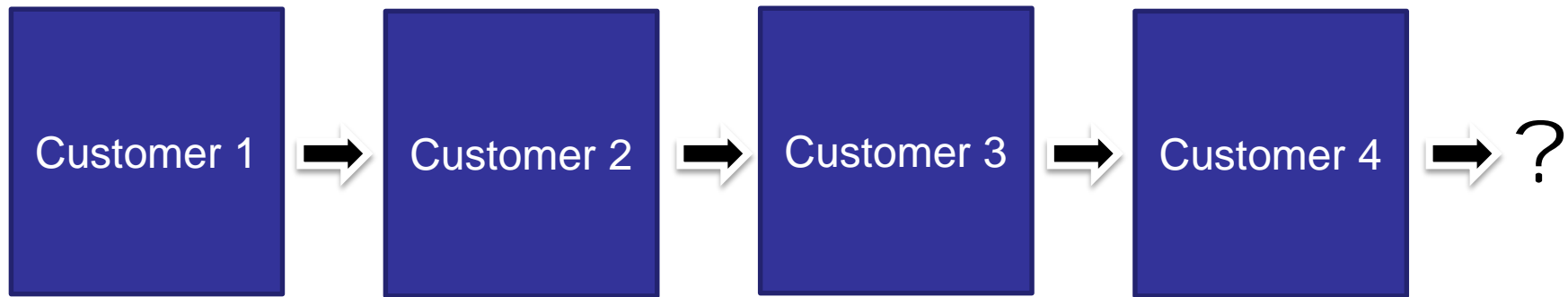
Array drawbacks

- Arrays are fixed in extent size once they contain data
- To re-size an Array (add another item or remove an item) you have to re-initialize the array causing the Array to loose all data
- An Array is considered a single variable – so all object references (pointers) are required to be within 32k
 - A rather theoretical limitation, I believe

Alternative variable length Lists

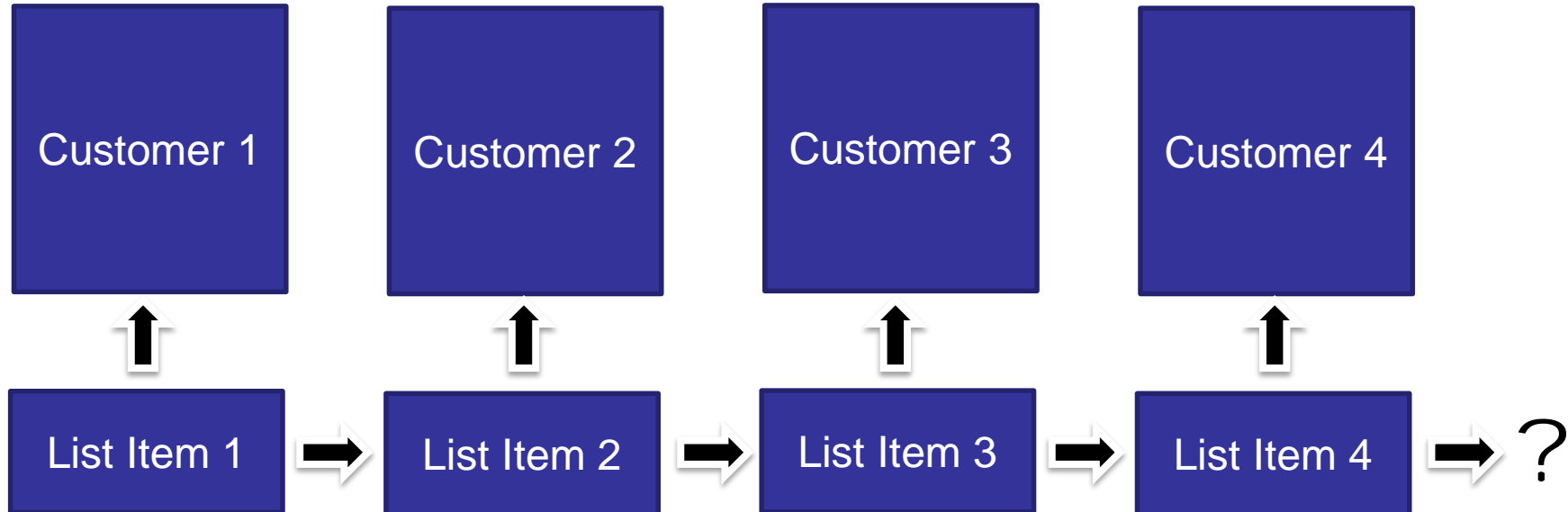
- Linked lists
- Temp-Table with Progress.Lang.Object field

Linked Lists



- Customer object needs additional property to reference next item in the list
- Disadvantage: Customer Object needs to manage list as well, no separation of concern

Linked lists



- Customer object kept as it
- Specialized “List Item” object instances that reference “their” customer instance and the next list item

Linked Lists

- Allows all kind of List manipulations
- Add instance (at the end)
- Insert instance (anywhere in the list)
- Delete reference

- Requires additional object for list item
- Relatively complex implementation
 - not very ABL'ish – we don't use ABL because we are keen to manipulate pointer values

List based on Temp-Table

- Temp-Tables may contain fields of type „Progress.Lang.Object“ to reference objects
- Temp-Tables may contain any number of records (0 .. n)
- Temp-Tables provide the “ABL”-ishst way of managing variable a set of object references

```
DEFINE PRIVATE TEMP-TABLE ttList NO-UNDO  
  FIELD ListItem      AS Progress.Lang.Object  
  INDEX ListItem ListItem
```

▪

Typical List class methods

- Add (Progress.Lang.Object)
- Add (Progress.Lang.Object[])
- Clear ()
- LOGICAL Contains (Progress.Lang.Object)
- Progress.Lang.Object GetItem (INTEGER)
- Remove (Progress.Lang.Object)
- RemoveAt (INTEGER)
- Object[] ToArray ()

- PROPERTY: Count (INTEGER)

Reducing Temp-Table overhead

- OO may need lots of lists ...
- Temp-Tables with small amount of records disproportionate overhead (dbi file growth)
- Issue relaxed for empty temp-tables in OE11
- Solution: break encapsulation – use shared temp-table

```
DEFINE PRIVATE STATIC TEMP-TABLE ttList NO-UNDO  
  FIELD RecordOwner AS CHARACTER  
  FIELD ListItem AS Progress.Lang.Object  
  INDEX RecordOwner RecordOwner ListItem
```


Reducing Temp-Table overhead

- OO may need lots of lists ...
- Temp-Tables with small amount of records

```
/*-----  
Purpose: Adds an Item to the List  
Notes:  
@param poItem The Item to add to the List  
@return The item that was added to the List  
-----*/  
METHOD PUBLIC Progress.Lang.Object Add (poItem AS Progress.Lang.Object):  
  
    DEFINE BUFFER ttList FOR ttList .  
  
    CREATE ttList.  
    ASSIGN ttList.RecordOwner = cInternalId  
           ttList.ListItem    = poItem .  
  
    THIS-OBJECT:OnListChanged (NEW ListChangedEventArgs (ListChangedTypeEnum:ListItemAdded)) .  
  
    RETURN poItem .  
  
END METHOD.
```

Adding Customers to List class

```
USING Consultingwerk.Framework.Base.* FROM PROPATH .
USING Samples.CustomerWithList.*    FROM PROPATH.

DEFINE VARIABLE oCustomers AS List    NO-UNDO .

DEFINE QUERY qCustomer FOR Customer .

/* ***** Main Block ***** */

oCustomers = NEW List () .

OPEN QUERY qCustomer
    PRESELECT EACH Customer WHERE Customer.CustNum < 1000
                                AND Customer.SalesRep = "HXM" .

DO WHILE QUERY qCustomer:GET-NEXT ():

    oCustomers:Add (NEW Customer (BUFFER Customer:Handle)) .

END.

MESSAGE "Count" oCustomers:Count
    VIEW-AS ALERT-BOX.
```

Customer class with List of Addresses

```

CONSTRUCTOR PUBLIC Customer (phBuffer AS HANDLE):
  DEFINE VARIABLE oAddress AS Address NO-UNDO .

  SUPER ().

  BufferAssert:IsAvailable (phBuffer) .

  ASSIGN THIS-OBJECT:Addresses = NEW List () .

  ASSIGN THIS-OBJECT:CustNum      = phBuffer::CustNum
  THIS-OBJECT:Name                = phBuffer::Name
  THIS-OBJECT:Contact            = phBuffer::Contact
  THIS-OBJECT:Phone              = phBuffer::Phone
  THIS-OBJECT:SalesRep           = phBuffer::SalesRep
  THIS-OBJECT:CreditLimit        = phBuffer::CreditLimit
  THIS-OBJECT:Balance            = phBuffer::Balance
  THIS-OBJECT:Terms              = phBuffer::Terms
  THIS-OBJECT:Discount           = phBuffer::Discount
  THIS-OBJECT:Comments           = phBuffer::Comments
  THIS-OBJECT:Fax                = phBuffer::Fax
  THIS-OBJECT:EmailAddress       = phBuffer::EmailAddress .

  oAddress = NEW Address () .

  THIS-OBJECT:Addresses:Add (oAddress) .

  ASSIGN oAddress:Country        = phBuffer::Country
  oAddress:Address               = phBuffer::Address
  oAddress:Address2              = phBuffer::Address2
  oAddress:City                  = phBuffer::City
  oAddress:State                 = phBuffer::State
  oAddress:PostalCode            = phBuffer::PostalCode .

END CONSTRUCTOR.

```

Sample

- Customer class with list of Addresses
- Loop through List of Customers
- Review List class methods

Agenda

- Introduction – OO ABL
- OO ABL's missing features
- Lists of Objects
- **Generic Lists of Objects**
- List Enumerators
- Enumerations
- Object Serialization



Generic Lists of Objects

- Standard List (of `Progress.Lang.Object`) two problems
 - You can't enforce item type during `Add`
 - You have to cast to item type after `GetItem()`

Standard List can't enforce item type

- Can add Address to oCustomers and add Customer to Addresses

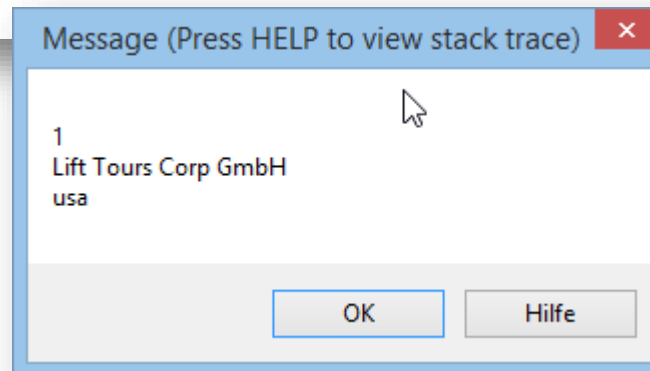
```
CAST (oCustomers:GetItem(1), Customer):Addresses:Add ((NEW Customer (42))) .
```

Standard List requires CAST on GetItem()

- Need to CAST GetItem(1) of oCustomers to Customer
- Need to CAST GetItem(1) of oCustomers:Addresses:GetItem(1) to Address

```
MESSAGE CAST (oCustomers:GetItem(1), Customer):CustNum SKIP
CAST (oCustomers:GetItem(1), Customer):Name SKIP

CAST (CAST (oCustomers:GetItem(1), Customer):Addresses:GetItem(1), Address):Country
VIEW-AS ALERT-BOX .
```



Need for ListCustomer and ListAddress

- Need specific List's for Customer and Address:
- ListCustomer
 - Add (Customer)
 - Customer GetItem (INTEGER)
- ListAddress
 - Add (Address)
 - Address GetItem (INTEGER)

Generic Types in C#

- List<T>

```
public class List<T> : IList<T>, ICollection<T>,
    IList, ICollection, IReadOnlyList<T>, IReadOnlyCollection<T>, IEnumerable<T>,
    IEnumerable
```

```
public void Add(
    T item
)
```

Parameters

item

Type: T

The object to be added to the end of the `List<T>`. The value can be **null** for reference types.

```
public T this[
    int index
] { get; set; }
```

Parameters

index

Type: `System.Int32`

The zero-based index of the element to get or set.

Property Value

Type: T

The element at the specified index.

Generic Types in C# (or GUI for .NET)

- DEFINE VARIABLE oList AS “List<Customer>” .
- On the fly defined ...
- oList:Add (Customer)
- oList[0]:Name -> no CAST required

- Add enforces list type
- GetItem does not require CAST to List Type

- **ABL lacks capabilities for ABL Generic Types**

Generic List in the ABL

- Generic List implementation using Include File ...

```
USING Consultingwerk.Framework.Base.* FROM PROPATH .
USING Samples.GenericLists.*         FROM PROPATH .
USING Progress.Lang.*                FROM PROPATH .

CLASS Samples.GenericLists.ListCustomer
    INHERITS GenericList:

    {Consultingwerk/Framework/Base/GenericList.i Customer}

END CLASS.
```

```
USING Consultingwerk.Framework.Base.* FROM PROPATH .
USING Samples.GenericLists.*         FROM PROPATH .
USING Samples.Customer.*             FROM PROPATH .
USING Progress.Lang.*                FROM PROPATH .

CLASS Samples.GenericLists.ListAddress
    INHERITS GenericList:

    {Consultingwerk/Framework/Base/GenericList.i Address}

END CLASS.
```

Preprocessor Listing

```

/*-----
Purpose: Adds an item to the generic List
Notes:
@param poItem And item of the Lists member type
@return The new Item added to the List
-----*/
METHOD PUBLIC Customer Add (poItem AS Customer):

    SUPER:InternalAdd (poItem).

    RETURN poItem .

END METHOD.

```

```

/*-----
Purpose: Retrieves an item from the generic List
Notes:  CAST's the element from the underlying Progress.Lang.Object based
        list
@param piIndex The 1 based index of the item to retrieve
@return The item of the Lists member type
-----*/
METHOD PUBLIC Customer GetItem (INPUT piIndex AS INTEGER ):

    RETURN CAST (SUPER:InternalGetItem (piIndex), Customer) .

END METHOD.

```

```

/*-----
Purpose: Adds an item to the generic List
Notes:
@param poItem And item of the Lists member
@return The new Item added to the List
-----*/
METHOD PUBLIC {1} Add (poItem AS {1}):

    SUPER:InternalAdd (poItem).

    RETURN poItem .

END METHOD.

```


Access Customer and Address

```
CLASS Samples.GenericLists.Customer:
```

```

/*-----
  Purpose: References the address of the customer
  Notes:
-----*/
DEFINE PUBLIC PROPERTY Addresses AS ListAddress NO-UNDO
GET.
SET.

```

```
oCustomers = NEW ListCustomer () .
```

```
OPEN QUERY qCustomer
```

```
  PRESELECT EACH Customer WHERE Customer.CustNum < 1000
    AND Customer.SalesRep = "HXM" .
```

```
DO WHILE QUERY qCustomer:GET-NEXT ():
```

```
  oCustomers:Add (NEW Customer (BUFFER Customer:Handle)) .
```

```
END.
```

```
MESSAGE oCustomers:GetItem(1):CustNum SKIP
```

```
  oCustomers:GetItem(1):Name      SKIP
```

```
  oCustomers:GetItem(1):Addresses:GetItem(1):Country
```

```
VIEW-AS ALERT-BOX .
```

Agenda

- Introduction – OO ABL
- OO ABL's missing features
- Lists of Objects
- Generic Lists of Objects
- **List Enumerators**
- Enumerations
- Object Serialization



Agenda

- Introduction – OO ABL
- OO ABL's missing features
- Lists of Objects
- Generic Lists of Objects
- **List Enumerators**
- Enumerations
- Object Serialization



List Enumerators

- Typical requirement to process all or some elements of the list in sequence
- One option to loop from 1 to oList:Count with a counter

```

DEFINE VARIABLE oCustomer AS Customer NO-UNDO .
DEFINE VARIABLE i AS INTEGER NO-UNDO .

DEFINE VARIABLE oAddress AS Address NO-UNDO .
DEFINE VARIABLE j AS INTEGER NO-UNDO .
    
```

```

DO i = 1 TO oCustomers:Count:

    oCustomer = oCustomers:GetItem(1) .

    MESSAGE oCustomer:CustNum SKIP
            oCustomer:Name
            VIEW-AS ALERT-BOX .

    DO j = 1 TO oCustomer:Addresses:Count:

        oAddress = oCustomer:Addresses:GetItem (j) .

        MESSAGE oAddress:Address SKIP
                oAddress:Address2 SKIP
                oAddress:City
                VIEW-AS ALERT-BOX.

    END.
END.
    
```

Enumerator in C#

- C# allows to "foreach" a list or other sets that are IEnumerable

```
foreach (Control oControl in this.Controls)
{
    Console.WriteLine(oControl.Name);
}
```

```
1 foreach (Customer oCustomer in oCustomers)
  {
    Console.WriteLine(oCustomer.CustNum);
    Console.WriteLine(oCustomer.Name);

    foreach (Address oAddress in oCustomer.Addresses)
    {
        Console.WriteLine(oAddress.Address);
        Console.WriteLine(oAddress.Address2);
        Console.WriteLine(oAddress.City);
    }
  }
```

- Loops through all Controls in this.Controls (List)
- [http://msdn.microsoft.com/en-us/library/ttw7t8t6\(v=vs.71\).aspx](http://msdn.microsoft.com/en-us/library/ttw7t8t6(v=vs.71).aspx)

.NET Enumerator from ABL (GUI for .NET)

- This is the ABL code similar to the C# foreach

```
DEFINE VARIABLE oControl AS System.Windows.Forms.Control NO-UNDO .  
DEFINE VARIABLE oControlEnumerator AS System.Collections.IEnumerator NO-UNDO .  
  
ASSIGN oControlEnumerator = CAST(oForm:Controls, System.Collections.IEnumerable):GetEnumerator() .  
oControlEnumerator:Reset() .  
  
DO WHILE oControlEnumerator:MoveNext() ON ERROR UNDO, THROW:  
    ASSIGN oControl = CAST(oControlEnumerator:Current, System.Windows.Forms.Control) .  
  
    MESSAGE oControl:Name VIEW-AS ALERT-BOX .  
END.
```

- First, we get the „Enumerator“ for the List
- That is an object, that provides a reference (Current) to an item and iterates over the items in the List
- Similar, to the ABL FOR EACH on a BUFFER

.NET Enumerator from ABL (GUI for .NET)

- Let's write Consultingwerk/foreach.i (Include)

```

DEFINE VARIABLE {2} AS {1} NO-UNDO .
DEFINE VARIABLE {2}Enumerator AS System.Collections.IEnumerator NO-UNDO .

ASSIGN {2}Enumerator = CAST({4}, System.Collections.IEnumerable):GetEnumerator() .

{2}Enumerator:Reset() .

DO WHILE {2}Enumerator:MoveNext() ON ERROR UNDO, THROW:
    ASSIGN {2} = CAST({2}Enumerator:Current, {1}) .
    
```

{3} not used, only used to fill up syntax and match c#

```

{Consultingwerk/foreach.i Control oControl in oForm:Controls}

MESSAGE oControl:Name VIEW-AS ALERT-BOX .

END.
    
```

Code Completion on properties of oControl works in recent Versions of PDSOE, did not work in 10.2B

Enumerator implementation for ABL List

- IEnumerable Interface with GetEnumerator() method
- Enumerator instance needs to provide method to
 - Reset()
 - MoveNext()
- Property
 - Current
- As List is implemented using ABL Temp-Table, we can create BUFFER and QUERY

Lists GetEnumerator() method

```
/*-----  
Purpose: Returns a new IEnumerator instance for this object instance  
Notes:  
@return The IEnumerator instance for this object  
-----*/  
METHOD PUBLIC IEnumerator GetEnumerator ():  
  
    DEFINE VARIABLE hBuffer AS HANDLE NO-UNDO .  
    DEFINE VARIABLE hQuery AS HANDLE NO-UNDO .  
  
    CREATE BUFFER hBuffer FOR TABLE TEMP-TABLE ttList:HANDLE .  
    CREATE QUERY hQuery .  
  
    hQuery:SET-BUFFERS (hBuffer) .  
    hQuery:QUERY-PREPARE (SUBSTITUTE ("FOR EACH ttList WHERE ttList.RecordOwner = &1":U,  
        QUOTER (cInternalId))) .  
  
    RETURN NEW ListEnumerator (THIS-OBJECT,  
        hQuery,  
        hBuffer) .  
END METHOD.
```

Enumerators Reset() method

```
/*-----  
    Purpose: Resets the Enumerator (starts enumerating from the first item on)  
    Notes:  
-----*/  
METHOD PUBLIC VOID Reset ():  
  
    hQuery:QUERY-OPEN () .  
  
    THIS-OBJECT:ListChanged = FALSE .  
  
END METHOD.
```

Enumerators MoveNext() method

```

/*-----
  Purpose: Moves the enumerator to the next item
  Notes:
  @return True when the next item is available, false when not.
-----*/
METHOD PUBLIC LOGICAL MoveNext ():

  IF THIS-OBJECT:ListChanged THEN
    UNDO, THROW NEW Consultingwerk.Framework.Exceptions.CannotMoveNextOnChangedList () .

  hQuery:GET-NEXT () .

  IF hQuery:QUERY-OFF-END THEN
    RETURN FALSE .
  ELSE
    RETURN TRUE .

END METHOD.

```

```

/*-----
  Purpose: Returns the current item in the List
  Notes:
-----*/
DEFINE PUBLIC PROPERTY Current AS Progress.Lang.Object NO-UNDO
GET:
  Consultingwerk.Assertion.HandleAssert:ValidHandle (hBuffer, "Enumeration":U) .
  Consultingwerk.Assertion.BufferAssert:IsAvailable (hBuffer) .

  RETURN hBuffer::ListItem .

END GET .

```

Object Reference
from List Temp-
Table

foreachABL.i

- We need a special version of foreach.i – simply because we cannot use the same IEnumerator interface for pure ABL and ABL with GUI for .NET

```
DEFINE VARIABLE {2} AS {1} NO-UNDO .
DEFINE VARIABLE {2}Enumerator AS Consultingwerk.Framework.Base.IEnumerator NO-UNDO .

ASSIGN {2}Enumerator = CAST({4}, Consultingwerk.Framework.Base.IEnumerable):GetEnumerator() .

{2}Enumerator:Reset() .

DO WHILE {2}Enumerator:MoveNext() ON ERROR UNDO, THROW:
    ASSIGN {2} = CAST({2}Enumerator:Current, {1}) .
```

- But as we mimic .NET Enumerators, the code looks very similar

Enumerating Customers and

Include File in
PROPATH, no .i
extention

```
{Cons  {foreachABL Customer oCustomer in oCustomers}

      MESSAGE oCustomer:CustNum SKIP
            oCustomer:Name
            VIEW-AS ALERT-BOX .

      {  {foreachABL Address oAddress in oCustomer:Addresses} sses}

            MESSAGE oAddress:Address SKIP
                  oAddress:Address2 SKIP
                  oAddress:City
                  VIEW-AS ALERT-BOX.

      E
END.      END.
          END.
```

- No need to remember that ABL starts counting with 1 and .NET starts counting with 0

Querying while iterating List

- ABL does not provide ability to Query objects
- Progress.Lang.Object field in Temp-Table can only be queried on object reference (same pointer)
- We could extend List implementation to include Filter criteria
- Probably would need multiple Filter criteria, would require to sync Filter criteria in List implementation with referenced objects
- Ultimately leads to redundancy of data in List temp-table, questioning the Object at all

LINQ in C#

- „Language **I**Ntegrated **Q**uery“
- Set of object + language (compiler features to provide syntax)

```
List<Customer> oCustomers = new List<Customer> ();

var queryLondonCustomers = from cust in oCustomers
                            where cust.City == "London" || cust.City == "Paris"
                            select cust;

foreach (Customer cust in queryLondonCustomers)
{
    Console.WriteLine(cust.Name);
}
```

```
foreach (Customer cust in (from cust in oCustomers
                            where cust.City == "London" || cust.City == "Paris"
                            select cust))
{
    Console.WriteLine(cust.Name);
}
```

ABL version of LINQ?

- As a matter of fact most lists will be rather small
- All data is in memory anyway (Objects not stored in DBI file as Temp-Tables are)
- It won't cause significant overhead if we iterate the List and just NEXT those records that don't match the selection criteria (negative filtering)

```
{foreachABL Customer oCustomer in oCustomers}  
  
    IF oCustomer:Discount <> 5 THEN  
        NEXT .  
  
    MESSAGE oCustomer:CustNum SKIP  
            oCustomer:Name SKIP  
            oCustomer:Terms SKIP  
            oCustomer:Discount  
            VIEW-AS ALERT-BOX .  
  
END.
```

linqABL.i

- Combines the benefits of foreachABL.i with filtering using positive expressions


```
{linqABL Customer oCustomer in oCustomers  
  where Discount = 5 or Discount = 20}
```

```
MESSAGE oCustomer:CustNum SKIP  
oCustomer:Name SKIP  
oCustomer:Terms SKIP  
oCustomer:Discount  
VIEW-AS ALERT-BOX .
```

```
END.
```

Preprocessor view

```
DO WHILE oCustomerEnumerator:MoveNext() ON ERROR UNDO, THROW:  
    ASSIGN oCustomer = CAST(oCustomerEnumerator:Current, Customer) .  
  
    IF NOT (oCustomer:Discount = 5  
  
        or oCustomer:Discount = 20  
  
    ) THEN NEXT .  
  
    MESSAGE oCustomer:CustNum SKIP  
            oCustomer:Name SKIP  
            oCustomer:Terms SKIP  
            oCustomer:Discount  
            VIEW-AS ALERT-BOX .  
  
END.
```



Filter criteria added

Agenda

- Introduction – OO ABL
- OO ABL's missing features
- Lists of Objects
- Generic Lists of Objects
- List Enumerators
- **Enumerations**
- Object Serialization



Enumeration

- Often named „Enum“ in other languages
- Set of related values of the same type
 - Weekdays
 - Months
 - Gender
 - AddressType
- Each entry is an object instance itself, it's a member of a set of "values"
- Enumeration may be Enumerable ... do we start to exaggerate?

Enumeration

- Typically representing a Name (Text representation) and a number (for ordering and comparison)
- Enumeration itself typically set of static references to member instances
- Much safer than Weekday based on INTEGER or OrderStatus based on CHARACTER
- Compiler detects typos, no need to runtime test
- Represents a type of it's own: strong typing of object properties or method parameters!

Enum in C#

- Enum represents a value type, each entry stands for a value

```
enum Weekday
{
    Monday = 1,
    Tuesday = 2,
    Wednesday = 3,
    Thursday = 4,
    Friday = 5,
    Saturday = 6,
    Sunday = 7
}
```

```
var currentDay = Weekday.Monday;

if (currentDay == Weekday.Monday)
{
    Console.WriteLine("it's Monday!");
}
```

- We can define variables of type Weekday
- Those can hold one of the Weekdays or null

Enums in the ABL

- ABL *currently* does not have support for Enums
- Enum can be build using single class
 - Static portion representing the Enumeration
 - Instance for each member
 - A single instance created for each member (singleton style) accessed via Properties of Enum

```

DEFINE PUBLIC STATIC PROPERTY Monday AS WeekDayEnum NO-UNDO
GET:
    IF NOT VALID-OBJECT (WeekDayEnum:Monday) THEN
        WeekDayEnum:Monday = NEW WeekDayEnum (1, "Monday":U) .

    RETURN WeekDayEnum:Monday .
END GET .
PRIVATE SET.
    
```

STATIC

```

DEFINE PUBLIC STATIC PROPERTY Tuesday AS WeekDayEnum NO-UNDO
GET:
    IF NOT VALID-OBJECT (WeekDayEnum:Tuesday) THEN
        WeekDayEnum:Tuesday = NEW WeekDayEnum (2, "Tuesday":U) .

    RETURN WeekDayEnum:Tuesday .
END GET .
PRIVATE SET.
    
```

Instance members

```

/*-----*/
Purpose: Constructor for the WeekDayEnum members
Notes:
@param piValue The internal (numeric) representation of the Enumeration member
@param pcLabel The text label of the Enumeration member
/*-----*/
CONSTRUCTOR PRIVATE WeekDayEnum (piValue AS INTEGER, pcLabel AS CHARACTER):
    SUPER ().

    ASSIGN THIS-OBJECT:Value = piValue
           THIS-OBJECT:Label = pcLabel .

END CONSTRUCTOR.
    
```

```

/*-----*/
Purpose: Returns a CHARACTER representation (human readable) of the
Enum member
Notes:
@return The CHARACTER representation of the enum member, identically to the Label property
/*-----*/
METHOD OVERRIDE PUBLIC CHARACTER ToString ():

    RETURN THIS-OBJECT:Label .

END METHOD.
    
```

A task for another Include file 😊

```
CLASS Consultingwerk.WeekDayEnum INHERITS Enum:

  {Consultingwerk/EnumMember.i Monday 1 WeekDayEnum}
  {Consultingwerk/EnumMember.i Tuesday 2 WeekDayEnum}
  {Consultingwerk/EnumMember.i Wednesday 3 WeekDayEnum}
  {Consultingwerk/EnumMember.i Thursday 4 WeekDayEnum}
  {Consultingwerk/EnumMember.i Friday 5 WeekDayEnum}
  {Consultingwerk/EnumMember.i Saturday 6 WeekDayEnum}
  {Consultingwerk/EnumMember.i Sunday 7 WeekDayEnum}

  /*-----
    Purpose: Constructor for the WeekDayEnum members
    Notes:
    @param piValue The internal (numeric) representation of the Enumeration member
    @param pcLabel The text label of the Enumeration member
  -----*/
  CONSTRUCTOR PRIVATE WeekDayEnum (piValue AS INTEGER, pcLabel AS CHARACTER):
    SUPER ().

    ASSIGN THIS-OBJECT:Value = piValue
           THIS-OBJECT:Label = pcLabel .

  END CONSTRUCTOR.
```

Demo

- Create a new Enum using Consultingwerk new Class Template in PDSOE
- Review TermsEnum in Customer
- Filter oCustomers on TermsEnum

```
DEFINE PUBLIC PROPERTY Terms AS TermsEnum NO-UNDO  
GET.  
SET.
```

PDSOE New Class Macro

New ABL Class

Create a user-defined class
 Optionally specify another class as a super class from which the class inherits state and behavior.

Package root: \ListEnumsSerializationSamples

Package: Samples.Enums

Class name: MyDemoEnum

Modifiers: Final Abstract Widget pool Serializable

Inherits: Consultingwerk.Enum

Implements:

Specify the code elements to be generated:

Method stubs:

Default constructor Destructor Super

Custom Class Template triggered by base class name

```

USING Consultingwerk.Enum.

CLASS Samples.Enums.MyDemoEnum INHERITS Enum:

  {Consultingwerk/EnumMember.i FirstValue 0 MyDemoEnum}

  /*-----
  Purpose: Constructor for the MyDemoEnum members
  Notes:
  @param piValue The internal (numeric) representation of the Enumeration
  @param pcLabel The text label of the Enumeration member
  -----*/

  CONSTRUCTOR PRIVATE MyDemoEnum (piValue AS INTEGER, pcLabel AS CHARACTER)
    SUPER ().

  ASSIGN THIS-OBJECT:Value = piValue
    THIS-OBJECT:Label = pcLabel .
  
```

Agenda

- Introduction – OO ABL
- OO ABL's missing features
- Lists of Objects
- Generic Lists of Objects
- List Enumerators
- Enumerations
- **Object Serialization**



Object Serialization

- Transforming an object instance (or a set of objects) into a form that can be persisted (disk, database, etc.) or be send to another system (aka marshalling)
- Deserialization is the process of converting this form back into an object – typically a new object instance, eventually on a different system or a different time (aka unmarshalling)
- Systems involved may be AppServer and Client
- Serialization is about Data in an object, not the implementation

Serialization formats

- Need to be understood by sender and receiver
- Binary form
- Text based formats
 - XML
 - JSON (from OpenEdge 11 on)
 - CSV
 - ...
- Morse code

OpenEdge Serialization in 11.4

- **Only supported between ABL Client and AppServer**
- Very well suited for parameter objects or throwing errors from the AppServer to the client
- Does not support serialization of objects to other clients types
 - XML serialization for .NET
 - JSON serialization for REST/Kendo UI/etc.
- So we are using Progress' serialization when it fits and our own when it does not

Walkthrough JSON Serializable object

- OpenEdge 11 provides JSON Object Model, flexible way of parsing and generating JSON Strings
- JSON is a LONGCHAR String, so it can be stored and send to another system

Walkthrough JSON Serializable object

- We typically want to serialize properties of an object and when we can send them to another system, it's a fair assumption that those properties are PUBLIC – transport cannot hide privates
- Serializing other members (e.g. temp-table would be possible as well, but not required by us)
- OpenEdge 11 has DYNAMIC-PROPERTY – so we can query and assign properties dynamically
- But we don't know what properties are available
 - No reflection in ABL (*yet*)

Serialization, again with an include file

- We maintain our own property specs – in a simple comma delimited list
- We use include file to consistently define property and property specs

```
DEFINE PUBLIC PROPERTY {1} AS {2} NO-UNDO {3}
GET.
SET.

&IF "{&SerializableProperties}":U NE "":U &THEN
&GLOBAL-DEFINE SerializableProperties {&SerializableProperties},{1},{2}
&ELSE
&GLOBAL-DEFINE SerializableProperties {1},{2}
&ENDIF
```

Consultingwerk.JsonSerializable Customer

```
CLASS Samples.Serialization.Customer
  INHERITS JsonSerializable:

  {Consultingwerk/JsonSerializableProperty.i Addresses ListAddress} .
  {Consultingwerk/JsonSerializableProperty.i CustNum INTEGER} .
  {Consultingwerk/JsonSerializableProperty.i Name CHARACTER} .
  {Consultingwerk/JsonSerializableProperty.i Contact CHARACTER} .
  {Consultingwerk/JsonSerializableProperty.i Phone CHARACTER} .
  {Consultingwerk/JsonSerializableProperty.i SalesRep CHARACTER} .
  {Consultingwerk/JsonSerializableProperty.i CreditLimit DECIMAL} .
  {Consultingwerk/JsonSerializableProperty.i Balance DECIMAL} .
  {Consultingwerk/JsonSerializableProperty.i Discount INTEGER} .
  {Consultingwerk/JsonSerializableProperty.i Comments CHARACTER} .
  {Consultingwerk/JsonSerializableProperty.i Fax CHARACTER} .
  {Consultingwerk/JsonSerializableProperty.i EmailAddress CHARACTER} .
  {Consultingwerk/JsonSerializableProperty.i Terms TermsEnum} .

  /*-----
    Purpose: Constructor for the Customer class
    Notes:
  -----*/

  CONSTRUCTOR PUBLIC Customer ():
    SUPER ().
    THIS-OBJECT:AddSerializableProperties ('{&SerializableProperties}':U) .
    THIS-OBJECT:Addresses = NEW ListAddress () .

  END CONSTRUCTOR.
```


Serializing Customer

```
USING Consultingwerk.Framework.Base.* FROM PROPATH .
USING Samples.Serialization.*        FROM PROPATH .

DEFINE VARIABLE oCustomer          AS Customer NO-UNDO .
DEFINE VARIABLE oInvoiceAddress AS Address NO-UNDO .

DEFINE VARIABLE lcSerialization AS LONGCHAR NO-UNDO .

/* ***** Main Block ***** */

FIND FIRST Customer NO-LOCK .

oCustomer = NEW Customer (BUFFER Customer:HANDLE) .

/* Add another address to Customer */
oInvoiceAddress = NEW Address () .
oInvoiceAddress:AddressType = AddressTypeEnum:Invoice .
oInvoiceAddress:Address     = "219 Littleton Road" .
oInvoiceAddress:City       = "Westford" .
oInvoiceAddress:State      = "MA" .
oInvoiceAddress:PostalCode = "01886" .

oCustomer:Addresses:Add (oInvoiceAddress) .

lcSerialization = oCustomer:Serialize() .

MESSAGE STRING (lcSerialization)
VIEW-AS ALERT-BOX.
```

```
● customer.json C:\Temp
1 {
2   "SerializedType": "Samples.Serialization.Customer",
3   "Addresses": [
4     {
5       "SerializedType": "Samples.Serialization.Address",
6       "Country": "USA",
7       "Address": "test",
8       "Address2": "poipoi",
9       "City": "Burlington",
10      "State": "MA",
11      "PostalCode": "01730",
12      "AddressType": "Unknown"
13    },
14    {
15      "SerializedType": "Samples.Serialization.Address",
16      "Address": "219 Littleton Road",
17      "City": "Westford",
18      "State": "MA",
19      "PostalCode": "01886",
20      "AddressType": "Invoice"
21    }
22  ],
23  "CustNum": 1,
24  "Name": "Lift Tours Corp GmbH",
25  "Contact": "Gloria Shepley",
26  "Phone": "(617) 450-0086",
27  "SalesRep": "HXM",
28  "CreditLimit": 66700.0,
29  "Balance": 903.64,
30  "Discount": 35,
31  "Comments": "This customer is on credit hold.",
32  "EmailAddress": "info@lift-tours.com",
33  "Terms": "Net30"
34 }
```

Demo

- Code Review Consultingwerk.JsonSerializable

Deserializing Customer

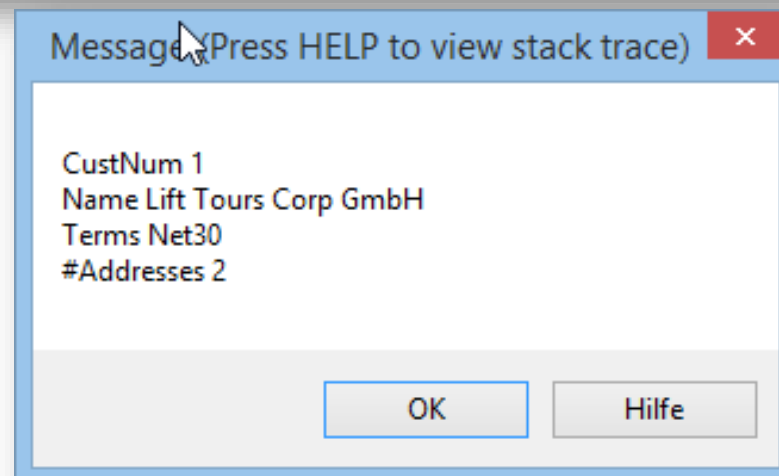
```

FIX-CODEPAGE (lcSerialization) = "utf-8" .

COPY-LOB FROM FILE "Samples\Serialization\customer.json" TO lcSerialization .

oCustomer = CAST (Consultingwerk.Serializable:DeserializeInstance (lcSerialization),
                  Customer) .

MESSAGE "CustNum"      oCustomer:CustNum SKIP
        "Name"         oCustomer:Name  SKIP
        "Terms"        oCustomer:Terms SKIP
        "#Addresses"   oCustomer:Addresses:Count
VIEW-AS ALERT-BOX.
    
```



Questions



Don't miss my other presentations

- Monday 11.00: **Telerik .NET for Infragistics Users**
- Monday 16.45: **DIY: Lists, Enumerators, Enumerations, Serialization**
- Tuesday 11.00: **Modernization – the SmartComponent Library**
- Tuesday 14.15: **Structured Error Handling**
- Wednesday 11.00: **Telerik Kendo UI with WebSpeed**

PUG
CHALLENGE
EXCHANGE
AMERICAS