Poly Bag and Iterator

Contents

Poly Bag and Iterator

Overview

In a nutshell a bag is a container that you stuff things into. You can then check if a thing is in the bag, iterate over the items in the bag, or use the index of the bag values to access other entries by index, such as items in an array or records on disk. For string data types you can specify whether the comparison of items will be case sensitive or case insensitive. *bag.inc* is a PowerBASIC source file that may be included in your projects to provide the ability to store most PowerBASIC variable types (`#Include Once "bag.inc"`). Copies of values are stored in the bag. For example if you add a string to a bag, it is a copy of the string value that is stored in the bag. This allows you to safely add values to a bag in any scope and still ensure your application will function normally. The bag is implemented using a hashtable and is suitable for implementing associative arrays, database indexing, caches, and sets.

Because the bag is designed to hold most PowerBASIC variable types, you must pass items to the bag ByRef. A notable exception is the `Bag_Item` which returns a reference to the stored value. For this you would normally pass a pointer variable of the appropriate type. The reference remains valid for the same duration as the bag and value ie. until the value reference is removed, the bag is released, the bag is cleared, or the bag is freed.

Before a bag can be used, you must initialize it. This is done by `Bag_Alloc` for allocated bags, or `Bag_Init` for `DIM`'ed bags. Once initialized you must call either Bag_Free to release an bag that was allocated, or `Bag_Release` to release a bag that as `DIM`'ed. You can call `Bag_Release` on allocated bags, but you can't call `Bag_Free` on `DIM`'ed bags.

Poly Bag and Iterator

Enums, Types, Collations

tBagTypes: `eString, eWString, eStringZ, eWStringZ, eLong, eDWord, eWord, eQuad, eInteger, eByte, eExtended, eDouble, eSingle, eCurrency, eCurrencyX, eType`

List of types a `tBag` can contain. These are set either when calling `Bag_Init` or `Bag_Alloc`.

tBag

The base bag type that is `DIM`'ed. The members of a `tBag` are:

```
table_ As Long Ptr          ' Array of Pointers to entries_
maxTable_ As Long           ' total # of slots in table_
modTable_ As Long           ' value to AND to get table index

entries_ As tBagEntry Ptr   ' Array of bagEntries
entryCount_ As Long         ' # of items in entries_
maxEntries_ As Long         ' total # of slots in entries_

freeCount_ As Long          ' # of removed items (count = high
free_ As Long               ' first free entries_ offset (+1)

collation_ As Byte Ptr      ' Pointer to binary collation table (String, WString, StringZ, WStringZ)
type_ As Long               ' See BagTypes
size_ As Long               ' Size of WStringZ, StringZ, Type
dwords_ As Long             ' Size of type in dwords
eSize_ As Long              ' Total size of each entry
```

Example1

```
Dim bag As tBag
Bag_Init b, %tBagTypes.eString, CodePtr( caseSensitiveAscii ), 0, %COUNT
```

Example2

```
Dim bag As tBag Ptr
Bag_Alloc a, %tBagTypes.eString, CodePtr( caseSensitiveAscii ), 0, %COUNT
```

tBagEach

A `tBag` iterator structure. This is defined in a separate include `bagEach.inc`. The members are:

`obj_ As tBag Ptr` : a pointer to a `tBag` to iterate over

`index_ As Long` : the current item

Collations

`caseInsensitiveAscii, caseSensitiveAscii`

Binary comparison collations for string types. `CodePtr`'s of these are passed to either `Bag_Alloc` or `Bag_Init`. You can create addition collations by defining the 256 comparison bytes using `AsmData` / `End AsmData`. These are used for comparing `eString`, `eWString`, `eStringZ`, and `eWStringZ` string data types.

```
Local b As tBag
Local a As tBag Ptr
Bag_Init b, %tBagTypes.eString, CodePtr( caseSensitiveAscii ), 0, %COUNT
```

Poly Bag and Iterator

```
Bag_Alloc a, %tBagTypes.eWString, CodePtr( caseInsensitiveAscii ), 0, %COUNT
```

Poly Bag and Iterator

Public Routines

Bag_Add

Function: Adds a new copy of a value to the bag if it doesn't already exist.

Returns

`Long`: Returns the index of the existing or added item.

Parameters

`ByRef obj As tBag`: The `tBag` variable to add a new value to.

`ByVal value As Dword`: Pass a `ByRef` variable to add. The type of the variable must exactly match the bagType value used to initialize the bag. No implicit conversions will be done.

Example

```
Dim bag As tBag
Dim value As Quad
Bag_Init bag, %tBagTypes.eQuad
value = 1: Bag_Add bag, ByRef value
value = 2: Bag_Add bag, ByRef value
```

Poly Bag and Iterator

Bag_Alloc

Subroutine: Allocates memory for a `tBag` using the provided parameters. When the bag is no longer needed you must call `Bag_Free` to release the allocated memory. Allocated bags persist until `Bag_Free` is called and may safely be passed up the scope tree. Allocated `tBag Ptr` variables are typically passed using `@bag`.

Parameters

`ByRef obj As Dword`(output): Pass a DWord sized variable that will point to the allocated tBag object. In the following example the bag variable will point to a tBag that was allocated:

```
Dim bag As tBag Ptr
Bag_Alloc bag, %tBagTypes.eQuad
```

`ByRef bagType As Long`: One of the tBagTypes. Determines the type of data that will be stored in the bag.

`Optional ByVal collation As Dword`: Used for `String`, `WString`, `StringZ`, and `WStringZ` to determine the binary collation used for comparing strings. Ex: `CodePtr( caseSensitiveAscii )`

`Optional ByVal memberSize As Long`: Determines the max size of `StringZ`, `WStringZ`, and `Type`s. For types use `sizeof( typeName )` where *typeName* is the name of a type you have defined.

`Optional ByVal requestedCapacity As Long` : Creates a bag with `requestedCapacity` slots. It will be able to hold at least .75 x requested capacity items. For example if you pass 100, the bag is guaranteed to have at least 100 slots and be able to hold at least 75 items without expanding. A `tBag` will automatically expand as needed, but it is slightly more efficient to pre-allocate space if you know it.

Example

```
Local a As tBag Ptr
Bag_Alloc a, %tBagTypes.eWString, CodePtr( caseInsensitiveAscii ), 0, %COUNT
```

Poly Bag and Iterator

Bag_Clear

Subroutine: Removes all items from a `tBag`. The items are freed by adding them to the internal free list. Free items will be reused as new items are added. If you want an empty `tBag` call `Bag_Release` instead. `Bag_Clear` is faster than `Bag_Release`.

Parameters

`ByRef obj As tBag` : The `tBag` variable to remove all items from.

Example

```
Local bag As tBag
Local value As Byte
Local bp As Byte Ptr
Bag_Init bag, %tBagTypes.eByte
value = 1: Bag_Add bag, ByRef value
value = 2: Bag_Add bag, ByRef value
value = 3: Bag_Add bag, ByRef value
Bag_Clear bag
```

Poly Bag and Iterator

Bag_Count

Function: Returns the # of active items in a `tBag`.

Returns

`Long`: Returns the # of active items in a `tBag`.

Parameters

`ByRef obj As tBag` : The `tBag` variable to get the count from.

Example

```
Local bag As tBag
Local value As Byte
Local bp As Byte Ptr
Bag_Init bag, %tBagTypes.eByte
value = 1: Bag_Add bag, ByRef value
value = 2: Bag_Add bag, ByRef value
value = 3: Bag_Add bag, ByRef value
Bag_RemoveAt bag, 0
? Format$( Bag_Count( bag ) ) ' display 2
```

Poly Bag and Iterator

Bag_Free

Subroutine: Free all memory for a `tBag` variable that was allocated with `Bag_Alloc`. Releases the `tBag` itself as well. i.e. after calling `Bag_Free` the `tBag` will no longer be valid. Attempting to use it will result in general protection faults. Failing to free an allocated bag when it is no longer needed will result in a memory leak.

Parameters

`ByRef obj As Dword` : The variable that was previously allocated using `Bag_Alloc`.

Example

```
Dim bag As tBag Ptr
Dim value As Quad
Bag_Alloc bag, %tBagTypes.eQuad
value = 1: Bag_Add @bag, ByRef value
Bag_Free bag
```

Poly Bag and Iterator

Bag_IndexOf

Function: Finds the index of a value in the bag. The returned index values are stable and may be used as array indexes or to retrieve records.

Returns

`Long`: Returns the zero based index of value passed in. If the value is not found -1 is returned.

Parameters

`ByRef obj As tBag`: The `tBag` variable to search

`ByVal value As Dword`: A `ByRef` variable to find the index of the value. The type of the variable must exactly match the `bagType` value used to initialize the bag. No implicit conversions will be done.

Example

```
Dim bag As tBag
Dim value As Quad
Dim j As Long
Bag_Init bag, %tBagTypes.eQuad
value = 1: Bag_Add bag, ByRef value
value = 2: Bag_Add bag, ByRef value
value = 3: Bag_Add bag, ByRef value
value = 2
j = Bag_IndexOf( bag, ByRef value )   ' Returns 1
```

Poly Bag and Iterator

Bag_Init

Subroutine: Initializes a bag variable using the provided parameters. Typically called on a bag that was DIM'ed with Dim, Local, Static, Global, or Threaded.

```
Dim bag As tBag
Bag_Init b, %tBagTypes.eCurrency
```

Parameters

ByRef obj As tBag: Pass a tBag variable to initialize. In the following example the bag variable will point to a tBag that was allocated:

```
Dim bag As tBag
Bag_Init bag, %tBagTypes.eQuad
```

ByRef bagType As Long: One of the tBagTypes. Determines the type of data that will be stored in the bag.

Optional ByVal collation As Dword: Used for String, WString, StringZ, and WStringZ to determine the binary collation used for comparing strings. Ex: CodePtr(caseSensitiveAscii)

Optional ByVal memberSize As Long: Determines the max size of StringZ, WStringZ, and Types. For types use sizeof(*typeName*) where *typeName* is the name of a type you have defined.

Optional ByVal requestedCapacity As Long : Creates a bag initially large enough to hold .75 x requestedCapacity items. For example if you pass 100, the bag is guaranteed to hold at least 75 items without expanding. A tBag will automatically expand as needed, but it's more efficient to pre-allocate space if you know how many items you will store.

Example

```
Dim bag As tBag
Bag_Init b, %tBagTypes.eString, CodePtr( caseSensitiveAscii ), 0, %COUNT
```

Poly Bag and Iterator

Bag_Item

Subroutine: Returns a reference to an item from a `tBag` by index. If you attempt to retrieve an item that has been removed, the reference will be 0. Generally trying to do anything with an item that has a 0 reference will result in a GPF.

Parameters

`ByRef obj As tBag`: The `tBag` variable to retrieve an item from.

`ByVal index As Long`: The index of the item to retrieve.

`ByRef item As Dword` (output): A pointer to a variable of the bags item type that will be returned.

Example

```
Local bag As tBag
Local value As Byte
Local bp As Byte Ptr
Bag_Init bag, %tBagTypes.eByte
value = 1: Bag_Add bag, ByRef value
value = 2: Bag_Add bag, ByRef value
value = 3: Bag_Add bag, ByRef value
Bag_Item tBag, 1, bp ' Retrieve a reference to 2
If bp Then ? @bp     ' After ensuring the item was not deleted, display the item
```

Poly Bag and Iterator

Bag_MaxEntry

Function: Returns the number of entries in use in the tBag. This may be higher than Bag_Count if you have removed items from the bag. This is the value you use to iterate over all items in the bag.

Returns

`Long`: Returns the number of items in use in the bag.

Parameters

`ByRef obj As tBag` : The tBag variable to retrieve the number of in use entries from.

Example

```
Local sp As String Ptr
For i = 0 To Bag_MaxEntry( bag )-1
  Bag_Item bag, i, ByRef sp
  If sp Then
    ' Valid string pointer
  End If
Next
```

Poly Bag and Iterator

Bag_Release

Subroutine: Release any internal memory allocated. The end result is a completely empty bag. It can optionally be expanded hold a certain number of items. This does not free the `tBag` itself, just memory internal to the `tBag`. This must be called when a `tBag` is no longer needed or before the `tBag` variable goes out of scope, such as at the end of a procedure. Failing to release a bag before it goes out of scope will result in memory leaks.

Parameters

`ByRef obj As tBag` : The `tBag` variable to clear.

`Optional ByVal requestedCapacity As Long` : Ensure the empty `tBag` will have at least `requestedCapacity` slots and be able to hold .75 x `requestedCapacity` items without expanding. A `tBag` will automatically expand as needed, but it's more efficient to pre-allocate space if you know how many items you will be adding.

Example

```
Local a As tBag Ptr
Bag_Alloc a, %tBagTypes.eWString, CodePtr( caseInsensitiveAscii ), 0, %COUNT
Bag_Release a
```

Poly Bag and Iterator

Bag_Remove

Function: Remove an item from a tBag by value.

Returns

`Long`: returns -1 (TRUE) if an item was removed. Returns 0 (FALSE) if an item was not removed.

Parameters

`ByRef obj As tBag` : A reference to the `tBag` to remove an item from.

`ByVal value As Dword` : A `ByRef` variable to find the item with a value to be removed.

Example

```
Local bag As tBag
Local value As CurrencyX
Bag_Init bag, %tBagTypes.eCurrencyX
value = 12.31@@: Bag_Add bag, ByRef value
value = 19.52@@: Bag_Add bag, ByRef value
value = 11.18@@: Bag_Add bag, ByRef value
value = 19.52@@: Bag_Remove bag, ByRef value: 'Remove the entry with 19.52 as a value
```

Poly Bag and Iterator

Bag_RemoveAt

Function: Remove an item from a `tBag` by index. Indexes are 0 based.

Returns

`Long`: Returns -1 (TRUE) if an item as removed. Returns 0 (FALSE) if an item was not removed.

Parameters

`ByRef obj As tBag` : A reference to the tBag to remove an item from.

`ByVal index As Long` : The index of the item to remove. It should be in the range of `0` to `Bag_MaxEntries-1`. You can only remove non-deleted entries.

Example

```
Local bag As tBag
Local value As CurrencyX
Bag_Init bag, %tBagTypes.eCurrencyX
value = 12.31@@: Bag_Add bag, ByRef value
value = 19.52@@: Bag_Add bag, ByRef value
value = 11.18@@: Bag_Add bag, ByRef value
Bag_RemoveAt bag, 1: 'Remove the second entry (19.52)
```

Poly Bag and Iterator

Internal Routines

BagGetNextCollectionSize

Function: Return a value that is greater than the number passed. For example `BagGetNextCollectionSize(0)` will return the next size a bag would be if it had to have at least one slot. A bag with x number of slots will hold at most .75 x values.

Returns

`Long`: The size the bag should be.

Parameters

`ByVal tableCount As Long` : The current size of the bag. This value will be incremented and then the next bag size found.

Poly Bag and Iterator

Bag_Expand

Subroutine: Expand a bag. Optionally you can request it to have at least requestedCapacity slots.

Parameters

`ByRef obj As tBag` : The `tBag` variable to expand.

`Optional ByVal requestedCapacity As Long` : The optional number of slots, minus one, that should be in the bag.

Poly Bag and Iterator

Bag_HashCode

Function: Calculates the hash code for a value.

Returns

`Long`: Returns the hash code computed for a value.

Parameters

`ByRef obj As tBag` : The `tBag` variable to compute a hash code for.

`ByRef value As Dword` : The `ByRef` variable to compute the hash code of its value.

Poly Bag and Iterator

Bag Iterator

A `tBag` can be iterated over easily by retrieving each item and skipping over any that have a `hashcode_` of `0`. The optional `bageach.inc` include file provides a few routines that make this easier.

BagEach_Reset

Subroutine: `BagEach_Reset` sets the `tBag` to iterate over and resets the record to `-1`, ready for the first call to `BagEach_Next`.

Parameters

`ByRef iterator As tBagEach` : The iterator to reset.

`ByRef obj As tBag` : The bag to iterate over.

Example

```
Local bag As tBag                               ' The bag we'll be filling
Local it As tBagEach                            ' The bag iterator
Bag_Init bag, %tBagTypes.eStringZ, _            ' Initialize the bag
         CodePtr( caseInsensitiveAscii ), 6
BagEach_Reset it, bag                           ' Reset the iterator
```

Poly Bag and Iterator

BagEach_Next

Function: `BagEach_Next` moves the current pointer to the next item in the `tBag` if there is one.

Returns

`Long` : Return -1 (TRUE) if there was a next item or 0 (FALSE) if there was not a next item.

Parameters

`ByRef iterator As tBagEach` : The iterator to move to the next item.

`ByRef iterator As tBagEach` : The iterator to reset.

`ByRef obj As tBag` : The bag to iterate over.

Example

```
Local bag As tBag                               ' The bag we'll be filling
Local it As tBagEach                            ' The bag iterator
Bag_Init bag, %tBagTypes.eStringZ, _            ' Initialize the bag
          CodePtr( caseInsensitiveAscii ), 6
BagEach_Reset it, bag                           ' Reset the iterator
While BagEach_Next( it )                        ' Iterate over each item that's left
Wend
```

Poly Bag and Iterator

BagEach_Index

Function: `BagEach_Index` returns the index (0-n) of the current item or -1 if there is no current item.

Returns

`Long` : Returns the index (0-n) of the current item or -1 if there is no item.

Parameters

ByRef iterator As tBagEach : The iterator to retrieve the index from.

Example

```
Local bag As tBag                               ' The bag we'll be filling
Local it As tBagEach                            ' The bag iterator
Local index As Long

Bag_Init bag, %tBagTypes.eStringZ, _            ' Initialize the bag
          CodePtr( caseInsensitiveAscii ), 6

BagEach_Reset it, bag                           ' Reset the iterator

While BagEach_Next( it )                        ' Iterate over each item that's left
  index = BagEach_Index( it )                   ' Bag index (0 to n-1), Record # (1 to n)
Wend
```

Poly Bag and Iterator

BagEach_Item

Subroutine: `BagEach_Item` Returns a pointer to the current item if there is one or `0` if there is no current item.

Parameters

`ByRef iterator As tBagEach` : The iterator to retrieve the item from.

`ByRef valueRef As Dword` : The value to set the address to.

Example

```
Local bag As tBag                           ' The bag we'll be filling
Local it As tBagEach                        ' The bag iterator
Local index As Long
Local szp As StringZ Ptr

Bag_Init bag, %tBagTypes.eStringZ, _        ' Initialize the bag
          CodePtr( caseInsensitiveAscii ), 6

BagEach_Reset it, bag                       ' Reset the iterator

While BagEach_Next( it )                    ' Iterate over each item that's left
  BagEach_Item( it, szp )                   ' Get a pointer to the current item
  If szp Then ? @szp                        ' Display the value if it exists
Wend
```

Poly Bag and Iterator

Examples

The following examples are from the samples directory.

Associative Array

An associate array is an array you retrieve values from by key. i.e. instead of accessing item by index such as you would with an array, you access them by some key such as last name. Sometimes an associative array is called a map, because you map one value to another.

```
#Compiler PBCC 6, PBWin 10
#Compile Exe
#Dim All

#Include Once "bag.inc"

Global values() As String
Global bag As tBag

Function PBMain () As Long
  Bag_Init bag, %tBagTypes.eString, CodePtr( caseInsensitiveAscii )
  Map_Add "Smith", "A gunslinger in the old west."
  Map_Add "Jones", "The partner of Smith."
  Map_Add "Einstien", "A physicist of some reknown."
  ? Map_Item( "Jones" )
End Function

Sub Map_Add( key As String, value As String )
  Local index As Long
  index = Bag_Add( bag, ByRef key )
  If bag.maxEntries_ > UBound(values())+1 Then
    ReDim Preserve values( bag.maxEntries_-1 )
  End If
  values( index ) = value
End Sub

Function Map_Item( key As String ) As String
  Local index As Long
  index = Bag_IndexOf( bag, ByRef key )
  If index>-1 Then Function = values(index)
End Function
```

Poly Bag and Iterator

Memory Handles

This sample allocates 32 bytes of memory using PowerBASIC's `GlobalMem Alloc` and uses it to store a string value.

```
#Compiler PBWin 10, PBCC 6
#Compile Exe
#Dim All

#Include Once "..\bag.inc"

Function PBMain () As Long
  Local bag As tBag                   ' The bag we'll be filling
  Local values() As StringZ*32        ' The array of values to add to the bag
  ReDim values(9)

  Bag_Init bag, %tBagTypes.eDWord     ' Initialize the bag to hold DWords (memory handles)

  Call BuildValues( values(), 0 )     ' Fill the array with 0-9
  GetMem bag, values()                ' Add them to the bag

  Call BuildValues( values(), 100 )   ' Fill the array with 100-109
  GetMem bag, values()                ' Add them to the bag

  Call BuildValues( values(), 1000 )  ' Fill the array with 1000-1009
  GetMem bag, values()                ' Add them to the bag

  ShowMem bag                         ' Display the values

  FreeMem bag                         ' Free the memory in the bag
  Bag_Release bag                     ' Free the bag memory
End Function

Sub BuildValues( values() As StringZ*32, ByVal start As Long )
  Local i As Long
  For i=0 To 9
    values(i) = Format$( i+start )    ' Stick a number in the array
  Next
End Sub

Sub GetMem( bag As tBag, values() As StringZ*32 )
  Local vHndl As Dword
  Local i, u As Long
  Local sp As StringZ Ptr

  ' Allocate some memory
  For i=0 To 9
    GlobalMem Alloc 32 To vHndl       ' Allocate memory
    GlobalMem Lock vHndl To sp        ' Lock allocated memory returning a stringZ pointer
    @sp = values(i)                   ' Assign the StringZ variable from the array
    GlobalMem UnLock vHndl To u       ' Unlock the memory
    Bag_Add bag, ByRef vHndl          ' Add the hangle to the bag
  Next
End Sub

Sub FreeMem( bag As tBag )
  Local i As Long
  Local vHndl As Dword Ptr
  Local v As Dword

  For i=0 To Bag_MaxEntry( bag )-1    ' Loop through all the possible entries (even deleted)
    Bag_Item bag, i, vHndl            ' Grab the item
    If vHndl Then                     ' If the item wasn't deleted
      Bag_RemoveAt bag, i             ' Remove the item from the bag
      GlobalMem Free @vHndl To v      ' Free the memory
    End If
  Next
End Sub

Sub ShowMem( bag As tBag )
  Local i, u As Long
```

Poly Bag and Iterator

```
  Local vHndl As Dword Ptr
  Local sp As StringZ Ptr
  Local s As String
  Local msg As IStringBuilderA

  msg = Class "StringBuilderA"

  For i=0 To Bag_MaxEntry( bag )-1     ' Loop through all the possible entries (even deleted)
    Bag_Item bag, i, vHndl             ' Grab the item
    If vHndl Then                      ' If it was't deleted
      GlobalMem Lock @vHndl To sp      ' Lock the memory
      s = @sp
      msg.Add s                        ' Add the string to our string builder
      msg.Add $CrLf                    ' Along with a CR/LF
      GlobalMem UnLock @vHndl To u     ' Unlock the memory
    End If
  Next
  ? msg.String                         ' Show what's in the bag
End Sub
```

Poly Bag and Iterator

Dynamic Strings

This example demonstrates using a dynamically allocated `tBag`. Internally the bag stores handles to `GlobalMem Alloc` to allocated memory locations that are just large enough to hold a string passed to it.

```
#Compiler PBWin 10, PBCC 6
#Compile Exe
#Dim All

#Include Once "..\bag.inc"

Function PBMain () As Long
  Local bag As tBag Ptr                    ' The allocated bag we'll be filling

  Bag_Alloc bag, %tBagTypes.eDWord         ' Initialize the allocated bag to hold DWords ("dynamic string"
handles)

  AddValues @bag, 0                        ' Fill the array with 0-9
  AddValues @bag, 100                      ' Fill the array with 100-109
  AddValues @bag, 1000                     ' Fill the array with 1000-1009

  ShowValues @bag                          ' Display the values

  RemoveAllValues @bag                     ' Remove all the values from the bag
  Bag_Free bag                             ' Free the allocated bag
End Function

'================================================
' "high-level" value functions
'================================================
Sub AddValues( bag As tBag, ByVal start As Long )
  Local i As Long
  For i=0 To 9
    Bag_Add bag, _                         ' Create a "dynamic" StringZ
            MakeString( Format$( i+start ) )
  Next
End Sub

Sub RemoveAllValues( bag As tBag )
  Local i As Long
  Local vHndl As Dword Ptr
  For i=0 To Bag_MaxEntry( bag )-1         ' Loop through all the possible entries (even deleted)
    Bag_Item bag, i, vHndl                 ' Grab the item
    If vHndl Then                          ' If the item wasn't deleted
      FreeString @vHndl                    ' Free the string
      Bag_RemoveAt bag, i                  ' Remove the entry from the bag
    End If
  Next
End Sub

Sub ShowValues( bag As tBag )
  Local i As Long
  Local vHndl As Dword Ptr
  Local msg As IStringBuilderA

  msg = Class "StringBuilderA"             ' Create the string builder

  For i=0 To Bag_MaxEntry( bag )-1         ' Loop through all the possible entries (even deleted)
    Bag_Item bag, i, vHndl                 ' Grab the item
    If vHndl Then                          ' If it wasn't deleted
      msg.Add GetString( @vHndl )          ' Get the string and add it to the string builder
      msg.Add $CrLf                        ' Add a Cr/Lf
    End If
  Next
  ? msg.String                             ' Show what's in the bag
End Sub

'================================================
' "Low level" Dynamic String functions
```

Poly Bag and Iterator

```
'================================================
Function MakeString( value As String ) As Dword
  Local vHndl, u As Dword
  Local sp As StringZ Ptr

  GlobalMem Alloc Len(value)+1 To vHndl    ' Allocate memory
  GlobalMem Lock vHndl To sp               ' Lock allocated memory returning a stringZ pointer
  @sp = value                              ' Assign the StringZ variable from the array
  GlobalMem UnLock vHndl To u              ' Unlock the memory
  Function = vHndl                         ' Return memory pointer
End Function

Sub FreeString( ByVal vHndl As Dword )
  Local v As Dword
  If vHndl Then                            ' if it wasn't deleted
    GlobalMem Free vHndl To v              ' Free the memory
  End If
End Sub

Function GetString( ByVal vHndl As Dword ) As String
  Local u As Dword
  Local sp As StringZ Ptr
  Local v As String
  GlobalMem Lock vHndl To sp               ' Lock allocated memory returning a stringZ pointer
  v = @sp                                  ' Capture the string
  GlobalMem UnLock vHndl To u              ' Unlock the memory
  Function = v                             ' Return the value
End Function
```

Poly Bag and Iterator

Record Access

This example builds a key "index" from a fixed length file of records using a `tBag`. It then retrieves records by key.

```
#Compiler PBWin 10, PBCC 6
#Compile Exe
#Dim All

#Include Once "..\bag.inc"

Type Customers
  Code As StringZ*6                              ' Key to access the customer
  Name As StringZ*40                             ' Customers name
End Type

Function PBMain () As Long
  Local bag As tBag
  Local hCust, recId As Long

  Bag_Init bag, %tBagTypes.eStringZ, _           ' Initialize the bag
            CodePtr( caseInsensitiveAscii ), 6

  hCust = FreeFile                               ' Get a file handle
  Open "recs.dat" _                              ' Open the file
    For Random    _
    As #hCust     _
    Len = SizeOf( Customers )

  BuildLookup hCust, bag                         ' Build the lookup

  ? GetRec( bag, hCust, "tomsp" )                ' Retrieve by key
  ? GetRec( bag, hCust, "Chops" )                ' Retrieve by key
  ? GetRec( bag, hCust, "LoNeP" )                ' Retrieve by key

  Bag_Release bag                                ' Release the memory allocated by the bag
End Function

Function GetRec( bag As tBag, ByVal fileNumber As Long, ByVal customerCode As String ) As String
  Local rec As Customers
  Local recId As Long
  Local key As StringZ*6

  key = customerCode                             ' Make the key a StringZ
  recId = Bag_IndexOf( bag, ByRef key ) + 1      ' Get the record id of the key (indexes are 0-n-1, records are
1 to n)
  If recId>0 Then                                ' If the record was found
    Get #fileNumber, recId, rec                  ' Get the record
    Function = rec.Code + ": " + rec.Name        ' Return it's contents
  Else
    Function = "(not found)"                     ' Indicate record wasn't found
  End If
End Function

Sub BuildLookup( ByVal ff As Long, bag As tBag )
  Local rec As Customers
  Local i As Long

  For i=1 To Lof(#ff)\SizeOf(Customers)          ' Loop through all records
    Get #ff, i, rec                              ' Get a record
    Bag_Add bag, ByRef rec.Code                  ' Add the code to the bag
  Next
End Sub
```