LC.zObjects

Developing with APL+Win, .Net and Objects

Eric Lescasse

Paris Lescasse Consulting

Copyright © 2016 by Eric Lescasse

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review or scholarly journal.

First Printing: 2016

ISBN <Enter your ISBN>

Publisher:
Lescasse Consulting
18 rue de la Belle Feuille
92100 Boulogne
France

www.lescasse.com

Ordering information:

Special discounts are available on quantity purchases by corporations, associations, educators, and others. For details, contact the publisher at the above listed address.

U.S. trade bookstores and wholesalers: Please contact Lescasse Consulting Tel: +33 6 47 98 13 61 or email eric@lescasse.com

Table of Contents

Copyright © 2016 Eric Lescasse

Copyright © 2016 Eric Lescasse

Copyright © 2016 Eric Lescasse

Copyright © 2016 Eric Lescasse

Copyright © 2016 Eric Lescasse

Copyright © 2016 Eric Lescasse

Preface

“Programming is an art”

Copyright © 2016 Eric Lescasse

Introduction

This book is intended to **APL+Win** Developers.

APL+Win is a version of the **APL** (A Programming Language) language developed and distributed by the APL2000 company[1].

This book is a Tutorial teaching how to use the **LC.zObjects** freeware product I[2] have developed to make it easy developing Windows desktop APL+Win applications.

The **LC.zObjects** freeware can be downloaded from http://www.lescasse.com/zObjects

To be able to reproduce the examples contained in this book on your computer, you need:

1. Windows 7+
2. The latest version of APL+Win
3. The Microsoft .Net Framework 4.5

To best benefit from this book, be sure to start from the beginning and retype all examples in your APL+Win session. That is the best way to experiment and to learn as well. You'll quickly see how easy it is to develop powerful and very nice looking APL applications using LC.zObjects.

LC.zObjects is made of:

1. A collection of APL+Win objects delivered in an APL+Win User Command file (**zObjects.sf**)
2. A collection of Microsoft .Net Framework objects, also delivered in the same APL+Win User Command file (**zObjects.sf**)
3. A unique C# ActiveX DLL (**LC.zObjects.dll**)

---

[1] APL2000
1 Research Court, Suite 325
Rockville, Maryland 20850 USA
Phone: +1 (301) 208-7150
http://www.apl2000.com

[2] Eric Lescasse
Lescasse Consulting
18 rue de la Belle Feuille
92100 Boulogne
France
Phone : +33 6 47 98 13 61
http://www.lescasse.com

Copyright © 2016 Eric Lescasse

Installation

Prerequisites

Please check that:

1. You are running Windows 7 or later
2. You have the latest version of APL+Win installed on your computer
3. You have the .Net Framework 4.5 installed on your computer

If you don't fulfill these requirements, you won't be able to run this Tutorial on your computer, but you can still read this book and learn from the techniques it exposes.

Download

Download **LC.zObjects.zip** from http://www.lescasse.com/zObjects.aspx

Installation

1. Unzip **LC.zObjects.zip**
2. Then run **zObjectsSetup.exe**

You'll be prompted to enter the folder in which you want to install **LC.zObjects**

What the installer does

The installer:

1. Creates the installation folder you specified (if it does not yet exist)
2. Copies the **zObjects.sf** file to this folder
3. Copies the **zzEmpty.w3** worskpace to this folder
4. Copies the **zzInit.w3** workspace to this folder
5. Copies the **LC.zObjects.dll** file to this folder
6. Copies the **RegisterzObjects.bat** file to this folder
7. Copies the **UnregisterzObjects.bat** file to this folder
8. Registers the **LC.zObjects.dll** C# ActiveX on your computer
9. Registers the **zObjects** path to the Registry

Copyright © 2016 Eric Lescasse

Initializing zObjects for first use

Assuming you have properly installed **zObjects** as described earlier, do the following:

1. Start APL+Win
2. Load the **zzInit** workspace

The **zzInit** workspace contains one single function called **zzInit** and is all you need to get started using **zObjects**.

You can then reload APL+Win and start using **zObjects** are described in the next lesson.

Notes

The **zzInit** function checks that some APL+Win .INI file parameters are set up correctly for **zObjects** to run fine. If not, these parameters will be automatically updated in your APL+Win **.INI** file and the following message(s) be printed in your APL session:

```
[Compatibility]BreakWaitOnError parameter changed to 1 in {your
.INI file name}

[Compatibility]MDIDropdown parameter changed to 1 in {your .INI
file name}
```

This function will also check that the APL+Win User Command Processor is up and running.

If the User Command Processor it not set up adequately in your APL+Win .INI file, your .INI file will be automatically updated and the following message be displayed in the APL session:

```
[Config]UcmdFile parameter set in {your .INI file name}
```

Finally, the **zzInit** function installs the **zObjects.sf** file as the second file in your UCMD file list (unless your UCMD file list contains only UCMDS? UCMDS2, UCMDS3 and/or UCMDSW in which case zObjects.sf is placed at the top of your UCMD file list).

IMPORTANT NOTE:
There is no **zObjects.w3** workspace. The **zObjects** are delivered as a **zObjects.sf UCMD file** from which you can load all the necessary functions you want to use with the **]zload** command.

Copyright © 2016 Eric Lescasse

The **]zload** command is similar to the standard **]uload** command except that it loads functions from the **zObjects** UCMD file whether or not it is at the top of your UCMD list.

Copyright © 2016 Eric Lescasse

Getting Started

Initializing zObjects in an APL session

Once you have run the **zzInit** workspace (this needs to be done only once ever), do the following to start using **zObjects**:

1. In **File Explorer**, navigate to the **zObjects** installation folder
2. Double click the **zzEmpty.w3**[3] workspace
3. Ensure that **zObjects** is in your User Command files list doing: **]ufile**
4. Then do:
   ```
   ]zload zzInit
   ```
   (or alternatively do: `]zload *` if you want to load all zObjects functions and objects)
5. Then run:
   ```
   zzInit
   ```

Alternatively:

1. Load APL+Win
2. Load any workspace (or stay in a CLEAR workspace)
3. If function **zzInit** is already in your workspace, run:
   ```
   zzInit
   ```
   Alternatively:
4. Ensure that **zObjects** is in your User Command files list doing: **]ufile**
5. Then do:
   ```
   ]zload zzInit
   ```
6. Then run:
   ```
   zzInit
   ```

Running the **zzInit** function results in:

- initializing the **zObjects** system
- creating an instance of the **zObject** object
- adding a number of fundamental utilities to the workspace (which names all start with **zz**)

Notes

1. Because all **zObjects** functions start with a lowercase **z**, there should likely be no conflict with your existing functions. Moreover **zObjects** will be listed at the end of your **)fns** functions list. It is recommended that you avoid creating functions starting with a **z** yourself.

[3] The **zzEmpty.w3** workspace is a totally empty workspace: it is just a convenience to let you double click on an APL+Win workspace in the zObjects installation folder. You can then start loading zObjects functions from the zObjects.sf User Command file which has been set at the top of your]ufile list

Copyright © 2016 Eric Lescasse

2. You need to run **zzInit** only once per APL session. However, you need to rerun **zzInit** again if you run: **'#'⎕wi'Reset'**

3. The **zzInit** function itself does run **'#'⎕wi'Reset'** so it leaves all your APL+Win objects (forms, etc.) alive. However, **zzInit** deletes any zObjects object instance that may still be running.

4. **zzInit** normally does not display any result or message to your APL session, unless part of the initialization done in Lesson 2 has been lost.

5. **zzInit** checks that the zObjects UCMD file is in your **]ufile** list and installs it there if it is not

Stop using zObjects in an APL session

Run:

```
      zzReset
```

zzReset performs the following actions:

1. deletes any instance of a **zObject** that may still be running
2. resets the system object **onNew** handler to what it was before **zzInit**
3. resets the system object **newclasses** property to what it was before **zzInit**
4. erases all the **zz** utility functions
5. erases all enums created by the zEnums function
6. does not run: '#'⎕wi'Reset'

Using more zObjects

As mentioned above, all you need to start using **zObjects** is to load the **zzInit** and **zObject** functions from the **zObjects.sf** UCMD file and to run **zzInit**.

Then, you should load the **zObjects** you want or need to use in your application from the **zObjects.sf** UCMD file.

Example:

```
      ]zload zForm zEdit zButton zGrid
4 objects loaded
```

Note:
zObjects does not use any other subroutine than the set of fundamental utilities created by **zzInit**.

Copyright © 2016 Eric Lescasse

So, when you want to use a new **zObject** in your application, simply load it from the **zObjects.sf** UCMD file: you will never have to load anything else in addition.

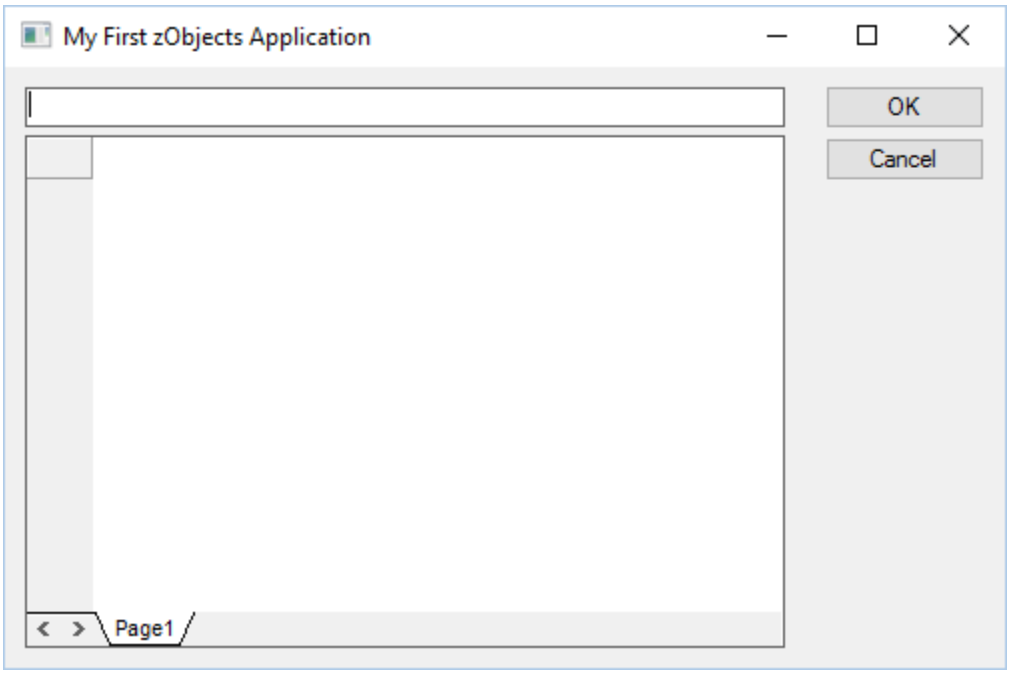
At this stage, you can start developing your application using these zObjects.

Here is an example:

Try it:

```
←'ff'□wi'*Create' 'zForm'('*caption' 'My First zObjects
      Application')('*size'300 500)
←'ff'□wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 380)('anchor' 'ltr')
←'ff'□wi'*.gr1.Create' 'zGrid'('*border'1)('wherelc' '>' '=' '>'
 '>>ed1')('anchor' 'ltrb')
←'ff'□wi'*.bn1.Create' 'zButton'('wherelc' '=ed1' '<' 0 0 ¯1)
      ('anchor' 'tr')('*caption' 'OK')
←'ff'□wi'*.bn2.Create' 'zButton'('wherelc' '>' '=' '=' '=')
      ('anchor' 'tr')('*caption' 'Cancel')
←'ff'□wi'minimumsize'250 450
←'ff'□wi'CenterScreen'
←'ff'□wi'*Show'
```

This creates the user interface of the following application:



This form looks good, is fully resizable, has a minimum size, and is centered on the screen.

Note:

Copyright © 2016 Eric Lescasse

Don’t try to understand the above function for now: it is just an initial example and things will be explained in much detail further on in this book.

Also, **zObjects** is not only a way to build perfect user interfaces easily.

It is much more than that. It is an extensive set of tools to help build APL+Win applications much more easily, much faster and in a much more secure way.

Conventions

Naming Conventions

All **zObjects** functions start with either a single **z** or by **zz**.

Functions starting by a single **z** or **znet** followed by an uppercase letter are objects (example: **zObject**).

Functions starting with **zz** are APL utilities or tutorial functions and are not objects.

Because of these conventions you should be able to use the **zObjects** system without any conflict with your own functions names in your own applications. It is however recommended that you do not create functions starting with a lowercase **z**.

zObjects Categories

There are 2 main categories of zObjects:

1. zObjects that derive from an **APL+Win** standard object
   (these zObjects are pure APL objects: example: **zForm**, **zEdit**, **zButton**, **zTimer**)
2. zObjects that derive from a **.Net Framework** object
   (these zObjects require the **LC.zObjects.dll**)

Note that **zObjects** that derive from a .Net Framework class start with **znet** followed by the name of the .Net Framework class from which they derive (example: **znetButton**, **znetTextBox**, **znetDictionary**, **znetRegex**, **znetFtpWebRequest**, etc.)

Note

Some **zObjects** may be pure data objects or objects that need not derive from an existing APL+Win class: however, the APL+Win System requires that a custom APL object do derive from an APL+Win existing class.

So, to be able to create such **zObjects**, we make them derive from the APL+Win **Menu** class because:

1. Creating a top level **Menu** object does not display any user interface
2. **Menu** is the APL+Win class that has the smaller memory footprint

Copyright © 2016 Eric Lescasse

zObjects Sub-Categories

Whether a zObject derives from an APL+Win standard object or from a .Net Framework object, it can be classified into 3 sub-categories:

1. **Class objects** (which do not have any User Interface; example: **zEmployee**, **znetRegex**, etc.)

2. **Control objects** (which have a User Interface and must be children of a container object; example: **zEdit**, **zButton**, **zGrid**, **znetButton**, **znetSplitter**, **zSplitContainer**, etc.)

3. **Form objects** (which have a User Interface and are top level objects, like **zForm**, **zMDIForm**, **znetForm**, etc.)

Copyright © 2016 Eric Lescasse

What are zObjects?

zObjects are APL+Win functions.

They also are APL+Win **custom objects**.

They all use the APL+Win System Object:

1. **onNew** event handler
2. **onAction** event handler
3. **newclasses** property

All this is done in the background and you basically use **zObjects** as you would use standard APL+Win objects like **Form**, **Edit**, **Button**, etc.

To use **zObjects**, you must use the APL+Win **⎕wi** system function.

How to create an instance of a zObject object?

To create an instance (called **ff**) of an APL+Win standard **Form** object you do:

```
      'ff'⎕wi'*Create' 'Form'
```

To create an instance of the **zForm** zObject you do:

```
      'ff'⎕wi'*Create' 'zForm'
```

Note: by default a zForm is not made visible when it is instantiated.

You'll learn later when to use the * prefix before properties, methods and events, but already note that you must **always** use the * prefix with the **Create** method.

Properties, Methods and Events

Just like APL+Win standard objects, **zObjects** have their own properties, methods and events.

Example:

```
      'ff'⎕wi'*Create' 'zForm' ('*color'255 0 0)
ff
      'ff'⎕wi'margins'8 8
      'ff'⎕wi'margins'
8 8
      'ff'⎕wi'DemoShow'.2 .2
```

Copyright © 2016 Eric Lescasse

More on this subject later.

Just note that:

1. You must use the * prefix if you call a property, method or event of the underlying APL+Win object
2. You must not use the * prefix when you call a custom property, method or event which is defined inside your object

Copyright © 2016 Eric Lescasse

The zObject object

One object has an essential and particular role in the **zObjects** library.

This object is called **zObject**.

All other **zObjects** inherit from **zObject**.

An instance of **zObject** called **zz** is created when you run **zzInit** and is available at all times while **zObjects** are up and running.

Most **zObject** properties and methods are called by other objects which inherit from **zObject**.

Example:

```
      'ff'⎕wi'*Create' 'zForm'
      'ff'⎕wi'DemoShow'.2 .2
```

The **DemoShow** method is not defined in the **zForm** object, but in the **zObject** object. It displays the form as a topmost form in the upper right corner of the screen, here with a height and width of 20% of the screen height and width.

However you can call it from the **zForm ff** instance (as done above) because **zForm** inherits from **zObject**.

The reason **DemoShow** is defined in **zObject** and not in **zForm** is that it is a common method that can also be called from a **zMDIForm** or **zDialogBox** object or from any object that derive from **zForm**, **zMDIForm** or **zDialogBox**. This way we do not have to duplicate code and defined the **DemoShow** method in the **zMDIForm**, **zDialogBox**, etc. objects.

You can also call some[4] properties and methods directly from the zObject zz instance.

Example:

```
      'zz'⎕wi'class'
zObject
      'zz'⎕wi'version'
2.3 (dll: 2.3)
      'zz'⎕wi'appdir'
C:\APLWIN\ZOBJECTS\
      'zz'⎕wi'A2R' 'A2:C5'
2 1 5 3
```

[4] Note that most **zObject** properties and methods are only useful when called by inheritance from other objects. They may fail if called directly from zObject i.e. from the zz instance.

Copyright © 2016 Eric Lescasse

Alias Names

Practically all zObjects use alias names (or links), i.e. for practically all zObjects an alias name is created at the time you create an instance of the zObject (except for top level objects).

This alias name is the last part of the object full name.

Example:

```
      ←'ff'⎕wi'*Create' 'zForm'
      ←'ff'⎕wi'*.lab.Create' 'zLabel'
      ←'ff'⎕wi'*.lab.lcc.Create' 'zLCChart'
```

After creating these 3 zObjects, you can query the system object **links** property to check that alias names have indeed been created:

```
      '#'⎕wi'links'
 lab ff.lab
 lcc ff.lab.lcc
```

So, in your code, instead of using the fully qualified object names which is often long and cumbersome to type, you can and should use the alias names: Example:

Instead of using:

```
      'ff.lab.lcc'⎕wi'*zMarginLeft'
36
```

rather, simply use:

```
      'lcc'⎕wi'*zMarginLeft'
36
```

Note

Using alias names implies that you always use unique object names throughout your whole application.

If you attempt to create an object that has the same terminal name as an existing one you'll get a warning printed to the APL session: example:

```
←'ff'⎕wi'*.fr1.Create' 'zFrame'
←'ff'⎕wi'*.fr1.lcc.Create' 'zEdit'
```

Copyright © 2016 Eric Lescasse

The <lcc ff.fr1.lcc> link overrides the existing <ff.lab.lcc> link!

You should always run the following command before creating an instance of your application main form (you can do it in the main form constructor):

```
←'#'⎕wi'links'(0 2⍴'')
```

to empty the alias or links list.

Note that aliases are extremely powerful and greatly simplify your application code in general.

Copyright © 2016 Eric Lescasse

Getting Information on Objects

Getting the list of available objects

```
      'zz'⎕wi'zobjects'
zButton           zList        zUButton           znetMenuItem
zCheck            zListview    zUEdit             znetNotifyIcon
zCombo            zManager     zULabel            znetNumericUpDown
zCommandBar       zMDIForm     znetAttachment     znetPath
zCommandButton    zMedia       znetButton         znetPanel
zDateTime         zMenu        znetButton16       znetPictureBox
…
```

Getting a list of properties, methods and events for an object

First create an instance of the object:

```
      'ff'⎕wi'*Create' 'zForm'
```

Then, query its **allprops** property:

```
      'ff'⎕wi'allprops'
Properties
----------
*bars             *events       *noredraw         *tooltiptime
*barwrap          *extent       *opened           *tooltipwidth
*border           *font         *place            *translate
*caption          *help         *pointer          *value
*children          help         *prompt           *visible
*class            *helpcontext  *properties       *where
 class            *hwnd         *scale            *ΔΔdialogcontrols
*color            *icon         *scrollaccel      *ΔΔdialogresult
*data             *instance     *scrollmargin     *ΔΔgaps
*ddeTopic         *keys         *self             *ΔΔmargins
*def              *limitwhere   *size             *ΔΔoldwhere
 dialogcontrols   *links        *state            *ΔΔontop
 dialogresult     *methods      *style            *ΔΔscreenmargins
 dialogvalues     *mode         *suppress         *ΔΔsysmenu
*edge             *modified     *targetformats    *ΔΔΔrefctrl
*enabled          *modifystop   *theme            *ΔΔΔsize
 esc              *name         *tooltipenabled

Methods
-------
*Close             *Draw    *Hide          *New     *Ref       *SetLinks
*ContextHelpMode   *Event   *Info           New      Refresh   *Show
*Create            *Exec    *Modify        *Open    *Resize    *Wait
*Defer             *Focus    MovePinkTip   *Paint   *Send
*Delete            *Help     MoveZInfo     *Popup   *Set

Events
------
*onAction          *onDragOver    *onModified      *onPaint
*onClose           *onDrop        *onMouseDouble   *onReopen
 onClose           *onDropDown    *onMouseDown     *onResize
```

Copyright © 2016 Eric Lescasse

```
*onContextMenu    *onExitError  *onMouseDrag     onResize
*onDdeConnect     *onFocus      *onMouseEnter   *onSend
*onDdeDisconnect   onFocus      *onMouseLeave   *onShow
*onDdeExecute     *onHelp       *onMouseMove     onShow
*onDelete         *onHide       *onMouseUp      *onUnfocus
*onDestroy        *onKeyDown    *onMove          onUnfocus
*onDragEnter      *onKeyPress    onMove         *onWait
*onDragLeave      *onKeyUp      *onOpen
```

Properties, methods and events marked by a * prefix are native properties of the underlying APL object. For example, a zForm object is based on a standard APL+Win Form object.

Properties, methods and events not marked by a * prefix are custom properties, methods or events defined inside the zObject itself and should be used without a prefix.

Note

Sometimes a custom property or method has the same name as a property or method of the underlying APL object.

For example:

```
    'ff'⎕wi'*class'
Form
    'ff'⎕wi'class'
zForm
```

So, when using the * prefix you are querying the **class** of the **zForm** underlying APL object which is a **Form**. When using the **class** property without a prefix, you're querying the current **zObject** class, here a **zForm**.

Filtering the list of properties, methods and events

The **allprops** property accepts an argument which allows you to filter the result. Just pass part of the properties, methods or events names you're interested in.

Example: in the following example we are interested in only those properties, methods and events containing tick in their names:

```
      'ff'⎕wi'*Create' 'zForm'
      'ff.lcc'⎕wi'*Create' 'zLCChart'

      'lcc'⎕wi'allprops' 'tick'
Properties
----------
*zITickStyle     *zXTickStyle  *zZTickStyle
*zTickMarkWidth  *zYTickStyle

Methods
```

```
-------
*zGetITickPositions  *zSetXTickLengths  *zSetYTickMarks2
*zGetXTickPositions  *zSetXTickMarks    *zSetZTickMarks
*zGetYTickPositions  *zSetXTickMarks2   *zSetZTickMarks2
*zGetZTickPositions  *zSetYTickLengths
*zSetITickMarks      *zSetYTickMarks

Events
------
```

Getting a list of custom properties, methods and events

The list of properties methods and events returned by allprops is sometimes overwhelming and you may be interested in seeing only those properties, methods and events which are defined within the zObject iself. For that, use the **props** property instead:

```
      'ff'□wi'props'
Properties
----------
class  dialogcontrols  dialogresult  dialogvalues  esc  help

Methods
-------
MovePinkTip  MoveZInfo  New  Refresh

Events
------
onClose  onFocus  onMove  onResize  onShow  onUnfocus
```

Getting Documentation

Once you know the names of the object properties, methods and events, you can get documentation on these properties, methods and events the normal way, using the ? prefix[5]

Getting documentation for APL objects

```
      ←'ff'⎕wi'*Create' 'zForm'
      ←'ff'⎕wi'*.ed1.Create' 'zEdit'

      'ff.ed1'⎕wi'properties'
 class font help value

      'ff.ed1'⎕wi'?value'
Get or Set the zEdit value
Syntax:  {value←}'obj'⎕wi'value'{value}
value: the text to be displayed in the zEdit control
Example:
      'zform.zedit'⎕wi'value' 'Test'
```

Getting documentation for .Net objects

The principle is the same but .Net objects may have overloads, i.e. several methods with the same name but differing by their arguments.

Example:

```
      ←'ff'⎕wi'*.lcc.Create' 'zLCChart'

      'lcc'⎕wi'?DrawBarChart'
Syntax: 'obj'⎕wi'*zDrawBarChart'(double[] data)
Summary: Draws a simple barchart from an array of values

Syntax: 'obj'⎕wi'*zDrawBarChart'(int[][] data)
Summary: Draws a grouped or stacked barchart from an integer array
      of arrays of values

Syntax: 'obj'⎕wi'*zDrawBarChart'(int[] data)
Summary: Draws a simple barchart from an array of integer values

Syntax: 'obj'⎕wi'*zDrawBarChart'(double[][] data)
Summary: Draws a grouped or stacked barchart from an array of
      arrays of values
```

In that case the documentation indicates the C# types of the arguments.

[5] Provided you have created an instance of the object first

Copyright © 2016 Eric Lescasse

int[]	means that you must pass an integer vector
double[]	means that you must pass a floating point vector
int[][]	indicates you must pass a nested vector of integer vectors

etc.

Note

Some properties, methods or events may not be documented yet or the documentation may not be available for those properties, methods or events.

Also note that for .Net objects (i.e. zLCChart and all the objects starting with znet):

you should not include a * prefix before ?
you should not include the z prefix

Example:

zLCChart has a zDrawBarChart method
to call this method you must use:
    'lcc'⎕wi'*zDrawBarChart' somedata
to query documentation about this method you must use:
    'lcc'⎕wi'?DrawBarChart'

Getting standard APL documentation

If you use a * prefix before the ? symbol, you'll get the standard APL documentation that simply gives you an idea of the parameters and sometimes of their types.

Example:

```
      'lcc'⎕wi'*?zDrawBarChart'
XzDrawBarChart method:
  ⎕WI 'XzDrawBarChart' data
```

Note

In that case, you must use the ***** prefix and the **z** prefix.

Copyright © 2016 Eric Lescasse

The zForm object

To start studying zObjects, let's look at the zForm object.

Create an instance of a zForm

```
      ←'ff'⎕wi'*Create' 'zForm'
```

As you can see, unlike an APL Form, a zForm does not show itself when you create it.

```
      'ff'⎕wi'*class'
Form
      'ff'⎕wi'class'
zForm
```

Why use a zForm rather than a simple APL+Win Form

First, as a zForm inherits from a Form it has all the characteristics, properties, methods and events of an APL+Win Form.

But on top of that, it has a lot of additional nice and useful features, which are presented in the next paragraphs.

That is the benefit of using zObjects.

Showing a zForm object

You can use one of the following methods to show a zForm object:

4. *Show
5. *Wait
6. Show
7. Wait
8. DemoShow
9. DemoWait

*Show and *Wait are the standard APL+Win Show and Wait methods.

Show and Wait are the same as *Show and *Wait except that they work for C# forms defined in LC.zObjects.dll.

DemoShow and DemoWait are powerful versions of *Show and *Wait.

Copyright © 2016 Eric Lescasse

The DemoShow and DemoWait methods

These 2 methods accept from 0 to 4 arguments. By default, if you don't use any argument, they display the form as a topmost form being half the screen in width and height, at the top right of your screen.

This is very handy when you need to test an application you are developing.

```
      'ff'⎕wi'?DemoShow'
Shows the current zForm or zMdiForm at the top right of the screen
Syntax:  'obj'⎕wi'DemoShow'{height}{width}{topmost}{focus}
height:  (optional) the height expressed in pixels or, if <1, in % of screen size;
if 0, the form keeps its default normal height
width:   (optional) the width expressed in pixels or, if <1, in % of screen size;
if 0, the form keeps its default normal width
topmost: (optional) 1=make the form topmost, 0(default)=don't
focus:   (optional) 1=return focus to the APL Session, 0(default)=don't
Example:
      ←'ff'⎕wi'*Create' 'zForm'('size'.3 .4)'DemoShow'
      ←'ff'⎕wi'*Create' 'zForm'('DemoShow'300 400)
      ←'ff'⎕wi'*Create' 'zForm'('DemoShow'.3 .4)
      ←'ff'⎕wi'*Create' 'zForm'('DemoShow'.3 .4 0)
      ←'ff'⎕wi'*Create' 'zForm'('DemoShow'.3 .4 0 0)
```

Try the above expressions.

Using Esc to close the form

When a **zForm** has the focus, you can press the **Escape** key to close it.

This is the default **zForm** behavior.

You can disable this feature and the escape key by setting the **esc** property to **0**.

```
      ←'ff'⎕wi'*Create' 'zForm' 'DemoShow'
```

Press Esc. The form will close.

```
      ←'ff'⎕wi'*Create' 'zForm' 'DemoShow'('esc'0)
```

Pres Esc. The form will not close.

Form resizing behavior

In many situations resizing an APL+Win Form that contains many controls results in a lot of flicker.

A zForm knows how to eliminate flicker by only displaying an outline of your form while you are resizing it.

Try resizing the form:

Copyright © 2016 Eric Lescasse

```
←'ff'⎕wi'*Create' 'zForm' 'DemoShow'
```

The zForm size property

The zForm size property lets you define the form size in pixels or if you provide values less than 1, in % of the screen workarea dimensions.

Try it:

```
←'ff'⎕wi'*Create' 'zForm'('size'300 400)'*Show'
←'ff'⎕wi'*Create' 'zForm'('size'.5 400)'*Show'
←'ff'⎕wi'*Create' 'zForm'('size'.3 .3)'*Show'
```

Note:
Be sure to not prefix size with a * otherwise you'll use the standard APL+Win size property which does not know how to treat arguments less than 1 as percent of the screen.

Centering a zForm on the screen

To center an instance of a zForm on the screen just use its CenterScreen method.
Try it:

```
←'ff'⎕wi'*Create' 'zForm'('size'.2 .3)'*Show'
←'ff'⎕wi'*Create' 'zForm'('size'.2 .3)'CenterScreen' '*Show'
```

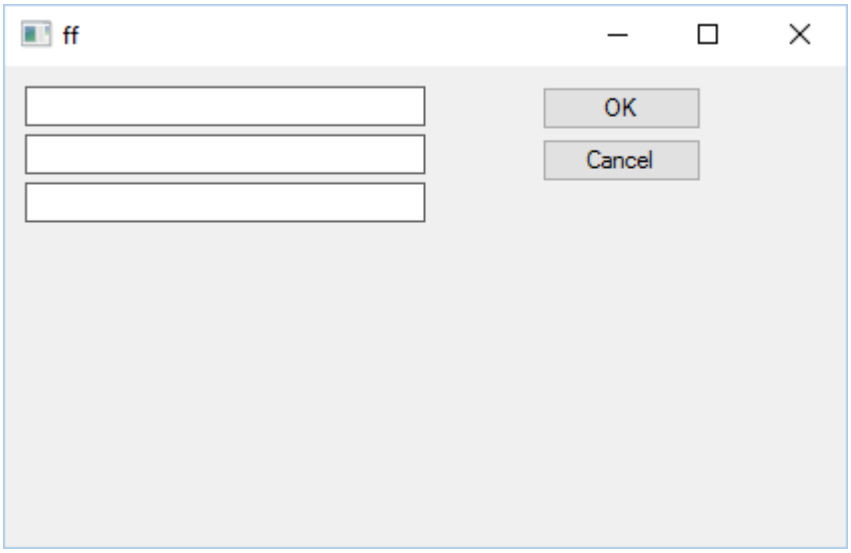
Copyright © 2016 Eric Lescasse

Autosizing a zForm

Let's create a zForm with a few controls. We'll explain later the fundamental wherelc property.

Try it:

```
←'ff'⎕wi'*Create' 'zForm'('size'240 420)
←'ff'⎕wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 200)
←'ff'⎕wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'⎕wi'*.ed3.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'⎕wi'*.bn1.Create' 'zButton'('wherelc' '=ed1' '>' 0 80 0 50)('*caption'
      'OK')
←'ff'⎕wi'*.bn2.Create' 'zButton'('wherelc' '>' '=' '=' '=')('*caption'
      'Cancel')
←'ff'⎕wi'*Show'
```

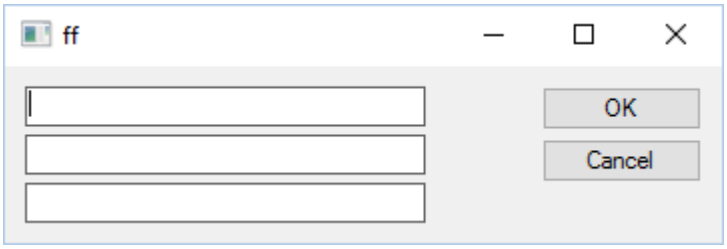


Controls are nicely laid out on the form thanks to the **wherelc** property, but the form size is too large and not adequate to the controls it contains.

Use the **AutoSize** method to force it to have the minimum size that still displays all controls.

Try it:

```
←'ff'⎕wi'AutoSize'
```



Copyright © 2016 Eric Lescasse

The zForm minimumsize property

The main form of an application is often a resizable form. But most often you want to prevent the form from being resized to a too small size.

You can do that easily using the **minimumsize** property.

Try it:

```
←'ff'⎕wi'*Create' 'zForm'('minimumsize'300 400)'Show'
```

Try resizing the form to a small size (smaller than 300 by 400 pixels). The form will not accept it.

Forcing a zForm to be topmost

At times you may want your **zForm** to stay topmost, i.e. remain visible on the screen above all other forms, even when it does not have the focus.

You can achieve that by using the **DemoShow** or **DemoWait** methods, but what if you have already displayed the form with *Show or Show.

Just use the **topmost** property.

Try it:

```
←'ff'⎕wi'*Create' 'zForm' 'Show'
```

Now click in the APL Session: the form disappears because it no longer has the focus and is not topmost.

Now type:

```
←'ff'⎕wi'topmost'1
```

The form reappears. Now click on it to give it the focus, then click in the APL Session. The form remains visible.

Just use:

```
←'ff'⎕wi'topmost'0
```

when you no longer want your **zForm** to be topmost.

The zForm spy

When developing an APL application it is often very useful to be able to know which events are occurring and in which order they occur.

You can easily do that using the zForm **spy** property.

Copyright © 2016 Eric Lescasse

```
      ←'ff'⎕wi'*Create' 'zForm'
      'ff'⎕wi'?spy'
Indicates if zForm or zMDIForm is being spied or not
Syntax: 'obj'⎕wi'spy'value
value: 0=dont spy  1=spy  2=spy eliminating some mouse events 3=also eliminate
Paint and Timer events
Example:
      'ff'⎕wi'spy'2
```

Try it:

```
      ←'ff'⎕wi'*Create' 'zForm'('spy'1)'*Show'
```

The form displays on the screen and the following events get displayed in the APL Session:

```
Show on ff
Action on ff (⎕warg= Refresh )
Focus on ff
Paint on ff
KeyUp on ff (⎕warg=28 1 0 0 0 1 13)
Action on ff (⎕warg= MoveZInfo )
Now, try moving your mouse above the form, clicking on the form, resizing the
form.  More events gets displayed in the APL Session:
Action on ff (⎕warg= MoveZInfo )
MouseEnter on ff (⎕warg=208 827 0 0 0)
MouseMove on ff (⎕warg=208 827 0 0 0)
MouseMove on ff (⎕warg=207 825 0 0 0)
MouseMove on ff (⎕warg=206 824 0 0 0)
MouseMove on ff (⎕warg=205 823 0 0 0)
MouseMove on ff (⎕warg=203 821 0 0 0)
MouseMove on ff (⎕warg=201 820 0 0 0)
MouseMove on ff (⎕warg=502 591 0 0 0)
MouseMove on ff (⎕warg=538 602 0 0 0)
MouseLeave on ff
Unfocus on ff
Focus on ff
Action on ff (⎕warg= MoveZInfo )
MouseEnter on ff (⎕warg=553 128 0 0 0)
MouseMove on ff (⎕warg=553 128 0 0 0)
MouseMove on ff (⎕warg=541 132 0 0 0)
MouseMove on ff (⎕warg=320 264 0 0 0)
MouseMove on ff (⎕warg=320 265 0 0 0)
MouseDown on ff (⎕warg=320 265 1 1 0)
MouseUp on ff (⎕warg=320 265 1 0 0)
MouseMove on ff (⎕warg=320 268 0 0 0)
MouseMove on ff (⎕warg=547 925 0 0 0)
MouseMove on ff (⎕warg=551 929 0 0 0)
MouseMove on ff (⎕warg=556 934 0 0 0)
MouseMove on ff (⎕warg=560 938 0 0 0)
MouseLeave on ff
Resize on ff (⎕warg=292 392 0)
Action on ff (⎕warg= ResizeChildren )
Action on ff (⎕warg= Refresh )
Paint on ff
Action on ff (⎕warg= MoveZInfo )
MouseEnter on ff (⎕warg=291 383 0 0 0)
MouseMove on ff (⎕warg=291 383 0 0 0)
MouseMove on ff (⎕warg=5 268 0 0 0)
```

Copyright © 2016 Eric Lescasse

```
MouseMove on ff (⎕warg=1 262 0 0 0)
MouseMove on ff (⎕warg=0 259 0 0 0)
MouseLeave on ff
Move on ff (⎕warg=28.25 97.25)
Action on ff (⎕warg= MovePinkTip )
Action on ff (⎕warg= MoveZInfo )
Unfocus on ff
```

If you are not interested in the (overwhelming) mouse events, you can use **'spy'2** instead of **'spy'1** and if you are not interested in Paint and Timer events either, you can use **'spy'3**.

Example:

```
      ←'ff'⎕wi'*Create' 'zForm'('spy'3)'*Show'
Hide on ff
Destroy on ff
Delete on ff
Delete on ff.Δesc
Delete on ff.Δaplsession
Show on ff
Focus on ff
KeyUp on ff (⎕warg=28 1 0 0 0 1 13)
MouseDown on ff (⎕warg=97 322 1 1 0)
MouseUp on ff (⎕warg=97 322 1 0 0)
Resize on ff (⎕warg=396 562 0)
Move on ff (⎕warg=21.3125 86.125)
MouseDown on ff (⎕warg=83 167 2 2 0)
MouseUp on ff (⎕warg=83 167 2 0 0)
ContextMenu on ff (⎕warg=83 167)
MouseDown on ff (⎕warg=77 184 1 1 0)
MouseUp on ff (⎕warg=77 184 1 0 0)
MouseDouble on ff (⎕warg=77 184 1 1 0)
MouseUp on ff (⎕warg=78 184 1 0 0)
Unfocus on ff
```

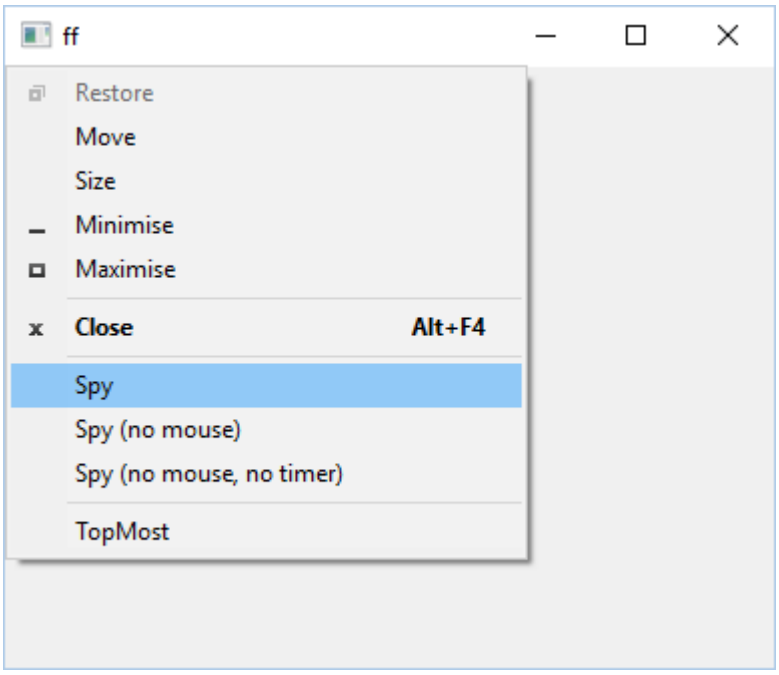
The zForm System Menu

By default, a **zForm** has a System Menu which you can use to make the form topmost or not and to spy the form.

Try it:

```
      ←'ff'⎕wi'*Create' 'zForm'('size'.25 .2)'*Show'
```

Now click the form top left caption button to open its System Menu and experiment with it.



You can prevent your **zForm** from having these 4 additional options in its System Menu it you set its **sysmenu** property to **0**.

Try it:

```
←'ff'□wi'*Create' 'zForm'('size'.25 .3)('sysmenu'0)'*Show'
```

Centering a form on another form

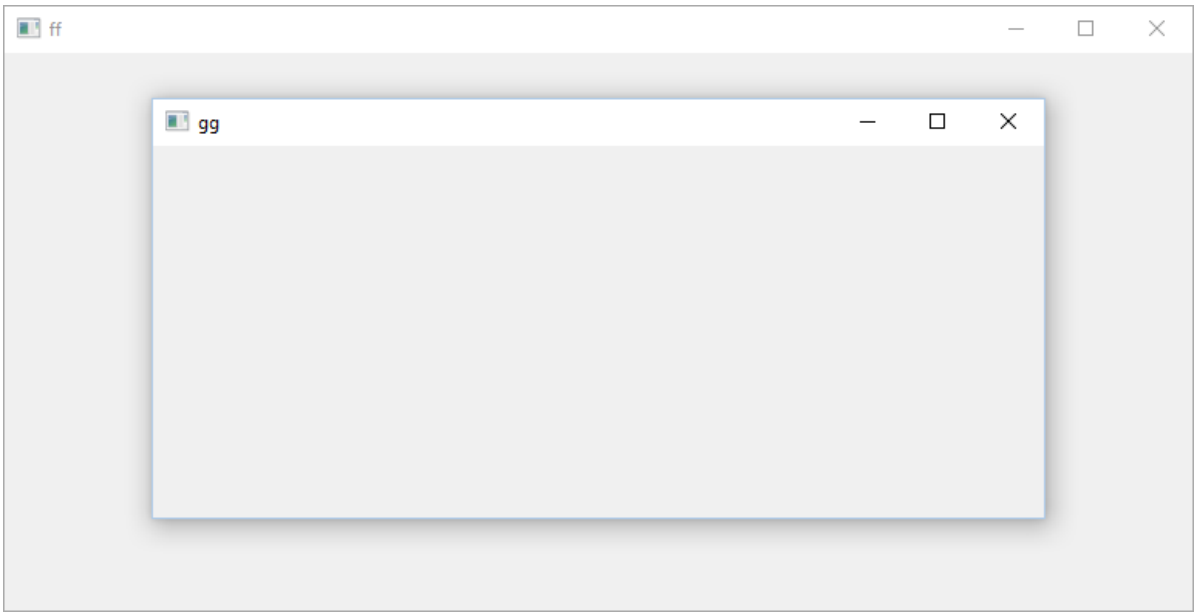
At times, when you display dialog boxes in your application you may want to center them on the parent form.

Just use the **CenterOn** method for that purpose.

Try it:

```
←'ff'□wi'*Create' 'zForm'('size'.3 .4)'*Show'
←'gg'□wi'*Create' 'zForm'('size'.2 .3)('CenterOn' 'ff')'*Show'
```

Copyright © 2016 Eric Lescasse



Note

A zForm has a good number of other capabilities, especially since it inherits from zObject and therefore can make use of any zObject property or method.

In particular, it has 4 extremely powerful properties which will be introduced soon:

- wherelc
- anchor
- gaps
- margins

In conclusion to this first introduction to a zObject, you can already probably guess the enormous development power that one can gain from using a zObject:

1. A **zForm** behaves like a standard APL+Win **Form** and inherits all its capabilities (properties, methods and events)
2. But a **zForm** has a number of additional frequently needed capabilities: it allows you to easily perform a number of common tasks that would otherwise require quite some programming
3. Moreover, these extra capabilities are built once and for all into **zForm**, so any time you create a new instance of a **zForm**, you can use all of these capabilities. It allows you to avoid reinventing the wheel all the time and maximizes reusability.

This is typical of objects in zObjects.

Copyright © 2016 Eric Lescasse

The wherelc property

The **wherelc** property, together with its companion **anchor** property, is the key to being able to easily build absolutely perfect forms in APL+Win without needing any visual editor.

The idea behind wherelc

After having built a number of forms and dialog boxes in APL, you sure know that the most cumbersome task is to compute the size and position of controls so that your form looks nice.

Why not leave that to the computer!

That is the task of the **wherelc** property!

The **wherelc** property knows how to install a control in your form by reference to a previous control already installed in the form.

With the **wherelc** property you should never have to compute the size or position of a control again.

Benefits of systematically using wherelc

1. You spare a real lot of time developing your forms and dialog boxes
2. Your forms and dialog boxes look pixel perfect
3. Developing your forms and dialog boxes UI becomes an easy task

Let's dig in wherelc!

The basics

The **wherelc** property may have from 4 to 8 arguments.

It is similar to the standard APL+Win **where** property, but:

1. Its first 4 arguments may be integers, ⍬ or characters like > or =
2. Its last 4 arguments are optional and are number of pixels used to adjust the position calculated

Here is a description of the arguments in detail:

1. Its 1st argument concerns the top position of the control
2. Its 2nd argument concerns the left position of the control
3. Its 3rd argument concerns the height of the control
4. Its 4th argument concerns the width of the control
5. Its 5th argument (optional) is a pixel adjustment for the control top position
6. Its 6th argument (optional) is a pixel adjustment for the control left position

Copyright © 2016 Eric Lescasse

7. Its 7th argument (optional) is a pixel adjustment for the control height
8. Its 8th argument (optional) is a pixel adjustment for the control width

An argument of 0 means “use the default”.

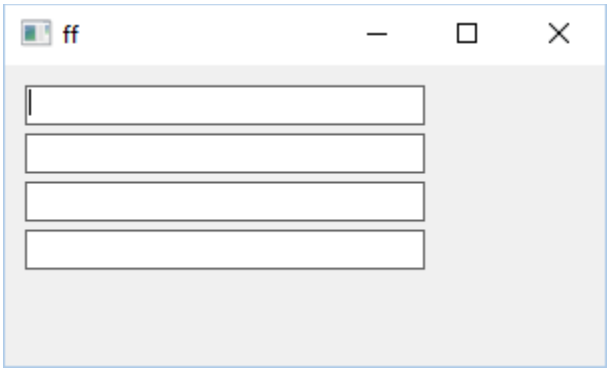
> **Note:**
> One of the rules to follow for **wherelc** to work fine is that ALL controls in your form must be zObjects. Only zObjects are aware of the **wherelc** property.

First example

Build a form with 4 zEdit controls, one below the other:

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'150 300)
←'ff'□wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 200)
←'ff'□wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed3.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed4.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*Show'
```



> **Note:**
> One important thing to do is to give the **zForm** a **size** at the time you create it.
>
> The reason is that the wherelc property may include instructions to automatically extend a control to an edge of the form and it needs to know the form size for that purpose.

Let’s explain the first wherelc: 'wherelc'0 0 0 200

1. The 1st 0 refers to the control top position and means: use its default top position (which is determined by the 1st element of the **zForm margins** property which by default is **10** pixels)

Copyright © 2016 Eric Lescasse

2. The 2nd 0 refers to the control left position and means: use its default left position (which is determined by the 2nd element of the **zForm margins** property which by default is **10** pixels)
3. The 3rd 0 refers to the control height and means: use its default height (which for a zZedit control is **20** pixels)
4. 200 means a width of 200 pixels

Now let's explain the other wherelc properties: 'wherelc' '>' '=' '=' '='

1. The > symbol means vertically display the control after the previous one (which vertically means below the previous one): the distance separating the control and the previous one is governed by the 1st element of the **gaps** property which by default is **4** pixels
2. The 1st = symbol refers to the control left position and means: horizontally display the control after the previous one (which horizontally means to the right of the previous one): the distance separating the control and the previous one is governed by the 2nd element of the **gaps** property with by default is **8** pixels
3. The 2nd = symbol refers to the control height and means: set its height equal to the previous control height
4. The 3rd = symbol refers to the control width and means: set its width equal to the previous control width

The power of wherelc

Here is where things become really nice and interesting.

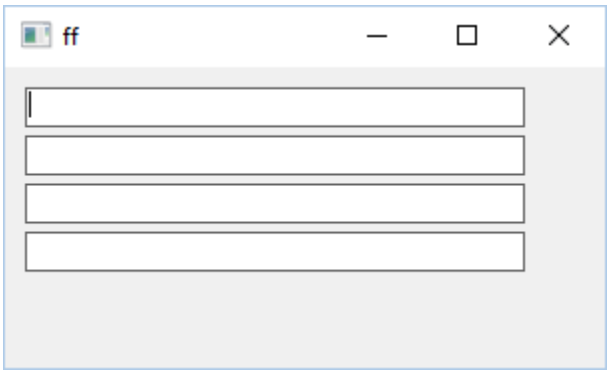
Assume we want to change the width of the 4 Edit controls to be 270 pixels instead of 200 pixels.

All you need to change is one number in your code.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'150 300)
←'ff'□wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 270)
←'ff'□wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed3.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed4.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*Show'
```

Copyright © 2016 Eric Lescasse

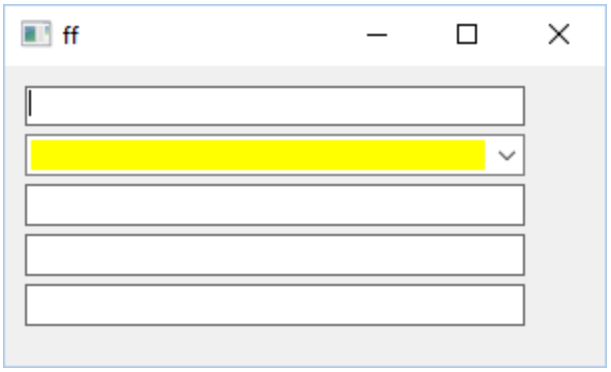


Because controls positions and sizes are computed at runtime by wherelc, the ed2, ed3 and ed4 controls auto adjust their sizes to match ed1's one.

Another example: assume we need to insert a zCombo control below the first zEdit control (ed1).

All we need to do is add a simple line:

```
←'ff'□wi'*Create' 'zForm'('*size'150 300)
←'ff'□wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 250)
←'ff'□wi'*.c1.Create' 'zCombo'('wherelc' '>' '=' '=' '=') ('*color'255 255 0)
←'ff'□wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed3.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed4.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*Show'
```



I have set the inserted control color to yellow so that you can clearly distinguish it.

Note that we have not had to recalculate the ed2, ed3 and ed4 control positions! Without wherelc we would have to recalculate their positions. Making changes in complex forms with the traditional where and size properties is often a nightmare. Making changes using wherelc is often as simple as inserting one line in a program.

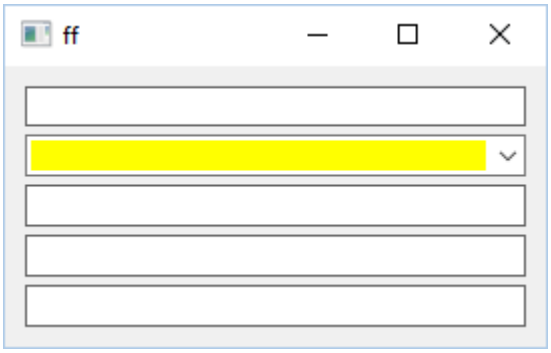
Autosizing the form

We have already see the AutoSize method in action, but just as a reminder you can force your form to AutoSize itself so that it fits its controls.

Copyright © 2016 Eric Lescasse

Try it:

```
      ←'ff'⎕wi'AutoSize'
```



Now the form is “perfect”: the margins are 10 pixels all around, the vertical distance between all controls is exactly 4 pixels everywhere.

The wherelc property has tons of other possibilities. We’ll review them after introducing the gaps and margins properties.

Copyright © 2016 Eric Lescasse

The gaps and margins properties

The gaps property

The gaps property is defined at the zForm level and has 2 elements:

1. The 1st element is the default vertical distance between controls in pixels
2. The 2nd element is the default horizontal distance between controls in pixels

The default is 4 8.

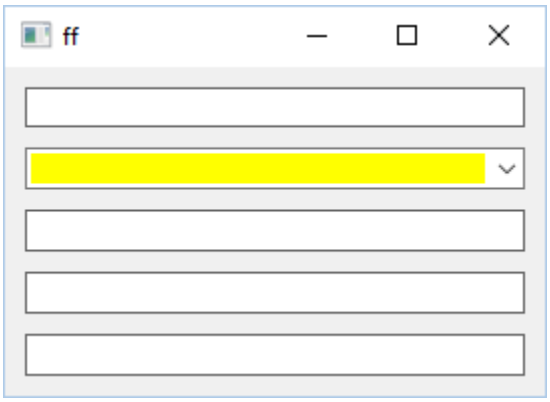
Try it:

```
      ←'ff'⎕wi'*Create' 'zForm'
      'ff'⎕wi'gaps'
4 8
```

Let's reuse the exact previous example but with a gaps property of **10 8** instead of **4 8**.

Try it:

```
      ←'ff'⎕wi'*Create' 'zForm'('*size'150 300)('gaps'10 8)
      ←'ff'⎕wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 250)
      ←'ff'⎕wi'*.c1.Create' 'zCombo'('wherelc' '>' '=' '=' '=')('*color'255 255 0)
      ←'ff'⎕wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
      ←'ff'⎕wi'*.ed3.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
      ←'ff'⎕wi'*.ed4.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
      ←'ff'⎕wi'AutoSize'
      ←'ff'⎕wi'*Show'
```



The vertical distance between controls is now precisely 10 pixels and our form is still "perfect" albeit different.

The margins property

The margins property refers to the distance separating the controls from the edges of the form. It also has 2 elements:

Copyright © 2016 Eric Lescasse

The 1st element is vertical margin in pixels

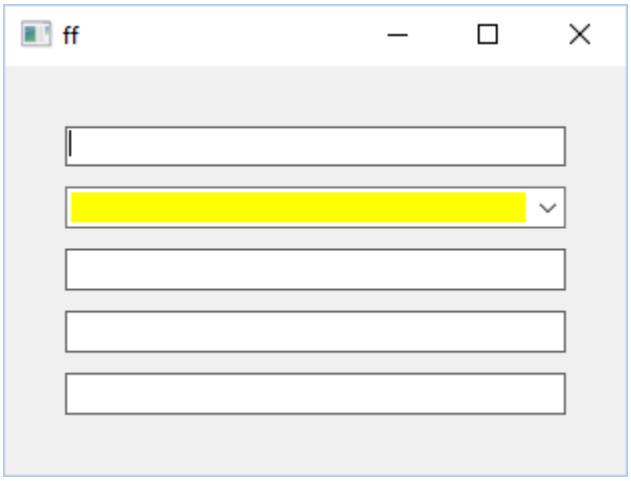
The 2nd element is the horizontal margin in pixels

The default is 10 10.

Let's change the margins property to 30 30 and see the effect.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'150 300)('gaps'10 8)('margins'30 30)
←'ff'□wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 250)
←'ff'□wi'*.c1.Create' 'zCombo'('wherelc' '>' '=' '=' '=')('*color'255 255 0)
←'ff'□wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed3.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed4.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'AutoSize'
←'ff'□wi'*Show'
```



The form is still "perfect" with a 30 pixels margin all around.

Copyright © 2016 Eric Lescasse

The caption property

zObjects controls have a **caption** property.

This property is special in that it automatically creates a Label associated with the control you are creating. Moreover it vertically aligns the Label caption with the text you may type in the control.

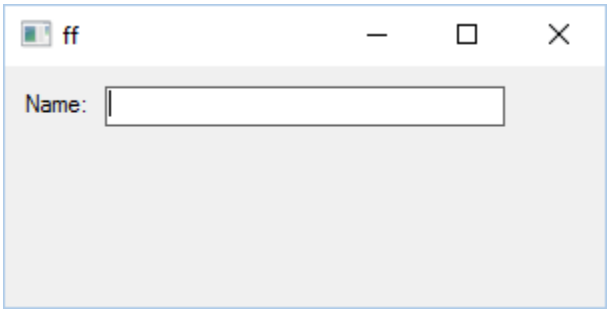
> **Note:**
> You must set the **caption** property AFTER the **wherelc** property

Simple use of the caption property

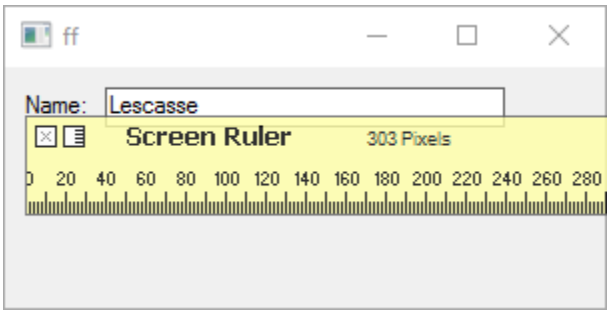
Let's create a zEdit control with its associated Label, using the caption property.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'120 300)
←'ff'□wi'*.ed1.Create' 'zEdit'('wherelc' 0 50 0 200)('caption' 'Name:')
←'ff'□wi'*Show'
```



Let's type my name in the zEdit control and check with my **Screen Ruler**[6] that the Label caption and my name are perfectly vertically aligned.



[6] A very useful simple Shareware that I recommend to any developer
Find it at: www.microfox.com

Copyright © 2016 Eric Lescasse

Caption alignment

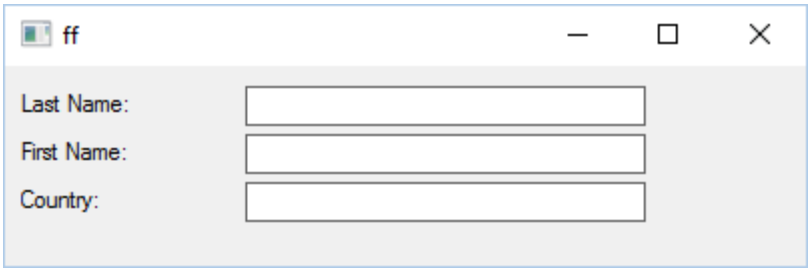
The caption alignment is governed by a number provided as the second element to the caption property.

Aligning captions at left margin

If you do not provide a 2nd element to the caption property, captions will be aligned at the left margin.

Try it:

```
←'ff'⎕wi'*Create' 'zForm'('*size'100 400)
←'ff'⎕wi'*.ed1.Create' 'zEdit'('wherelc' 0 120 0 200)('caption' 'Last
      Name:')
←'ff'⎕wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' 0 '=')('caption' 'First
      Name:')
←'ff'⎕wi'*.ed3.Create' 'zEdit'('wherelc' '>' '=' 0 '=')('caption' 'Coun
      try:')
←'ff'⎕wi'*Show'
```



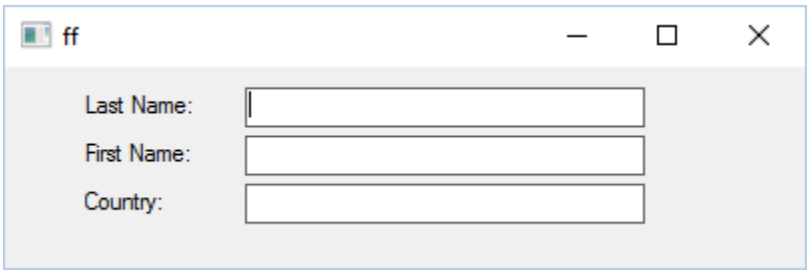
Aligning captions at a given fixed position

Use a positive numeric value as the 2nd caption element to align captions at this position.

Try it:

```
←'ff'⎕wi'*Create' 'zForm'('*size'100 400)
←'ff'⎕wi'*.ed1.Create' 'zEdit'('wherelc' 0 120 0 200)('caption' 'Last
      Name:' 40)
←'ff'⎕wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' 0 '=')('caption' 'First
      Name:' 40)
←'ff'⎕wi'*.ed3.Create' 'zEdit'('wherelc' '>' '=' 0 '=')('caption'
      'Country:' 40)
←'ff'⎕wi'*Show'
```

Copyright © 2016 Eric Lescasse

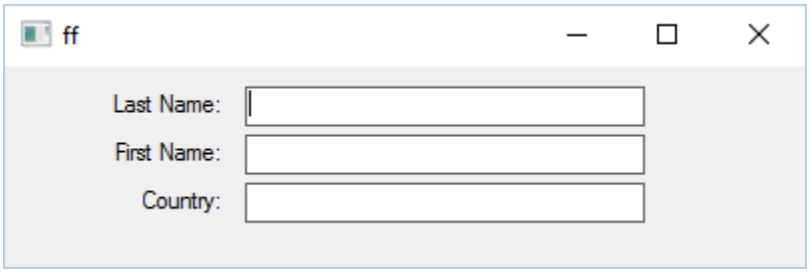


Right aligning the captions

Use a 0 numeric value as the 2nd caption element to right align captions at a "gap" distance from the Edit controls.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'100 400)
←'ff'□wi'*.ed1.Create' 'zEdit'('wherelc' 0 120 0 200)('caption' 'Last
      Name:' 0)
←'ff'□wi'*.ed2.Create' 'zEdit'('wherelc' '>' '='0'=')('caption' 'First
      Name:' 0)
←'ff'□wi'*.ed3.Create' 'zEdit'('wherelc' '>' '='0'=')('caption' 'Country:'
      0)
←'ff'□wi'*Show'
```



Note:
You don't need to use the **caption** property with the **zCheck** and **zOption** controls. Rather use their ***caption** property and set their **style** property to **1** to force their captions to be on the left

Copyright © 2016 Eric Lescasse

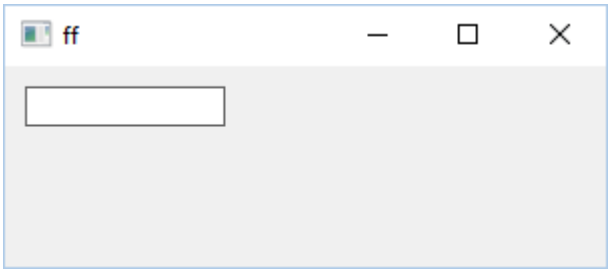
More wherelc goodies

Make a control extend itself to the right edge of the form

Use the > or >> symbols for the 4th wherelc argument element.

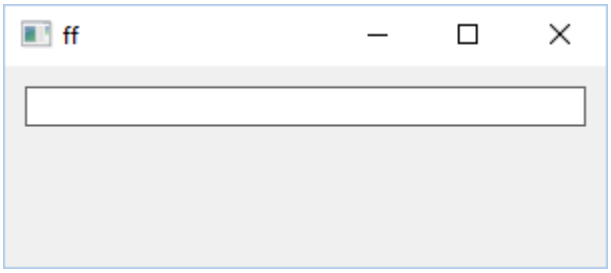
Try it:

```
←'ff'⎕wi'*Create' 'zForm'('*size'100 300)
←'ff'⎕wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 100)
←'ff'⎕wi'*Show'
```



Now let's use >:

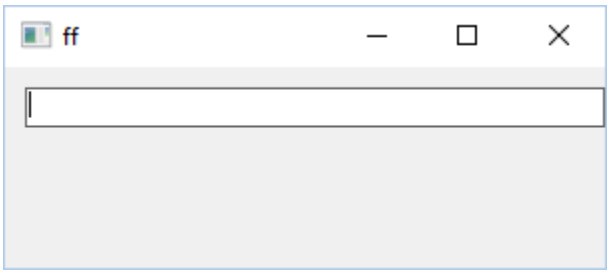
```
←'ff'⎕wi'*Create' 'zForm'('*size'100 300)
←'ff'⎕wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 '>')
←'ff'⎕wi'*Show'
```



When using > for the 4th argument element the control extends to the edge of the form minus the horizontal margin. If you want to the control to extend to the very edge of the form without any margin, use >> instead:

```
←'ff'⎕wi'*Create' 'zForm'('*size'100 300)
←'ff'⎕wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 '>>')
←'ff'⎕wi'*Show'
```

Copyright © 2016 Eric Lescasse

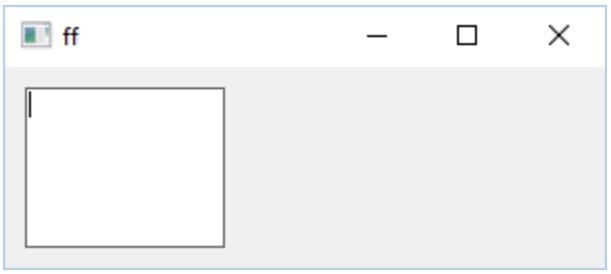


Make a control extend itself to the bottom of the form

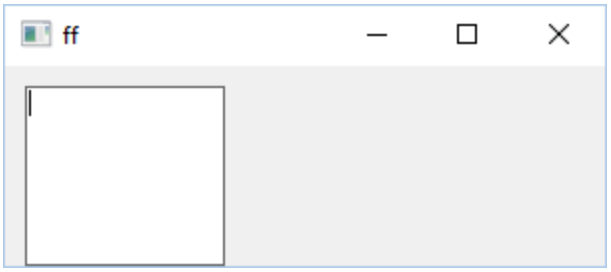
Similarly as before, just use > or >> for the 3rd element of the whereIc arguments.

Try it:

```
←'ff'⎕wi'*Create' 'zForm'('*size'100 300)
←'ff'⎕wi'*.ed1.Create' 'zEdit'('whereIc'0 0 '>' 100)
←'ff'⎕wi'*Show'
```




```
←'ff'⎕wi'*Create' 'zForm'('*size'100 300)
←'ff'⎕wi'*.ed1.Create' 'zEdit'('whereIc'0 0 '>>' 100)
←'ff'⎕wi'*Show'
```

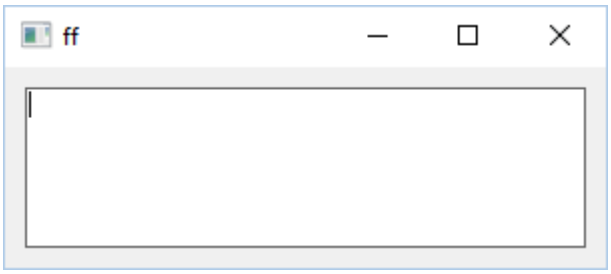


And of course you can combine > and/or >> for the 3rd and 4th elements.

Try it:

```
←'ff'⎕wi'*Create' 'zForm'('*size'100 300)
←'ff'⎕wi'*.ed1.Create' 'zEdit'('whereIc'0 0 '>' '>')
←'ff'⎕wi'*Show'
```

Copyright © 2016 Eric Lescasse



The control now extends to both the bottom edge of the form and to the right edge (minus the margin).

Copyright © 2016 Eric Lescasse

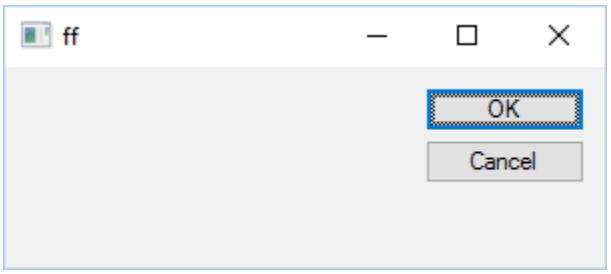
Installing a control at the right edge of a form

The next thing you often need to be doing is to install a control at the right edge of a form or at the bottom edge of a form.

Use < or << for the 2nd element of the wherelc arguments.

Try it:

```
←'ff'⍞wi'*Create' 'zForm'('*size'100 300)
←'ff'⍞wi'*.bnOK.Create'      'zButton'('wherelc'  0  '<'  0   0)
←'ff'⍞wi'*.bnCancel.Create'  'zButton'('wherelc' '>' '=' '=' '=')
←'ff'⍞wi'*.bnOK.caption'     'OK'
←'ff'⍞wi'*.bnCancel.caption' 'Cancel'
←'ff'⍞wi'*Show'
```

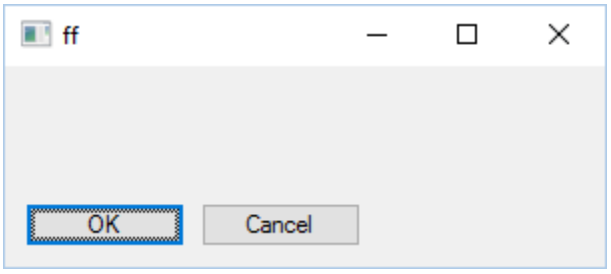


Installing a control at the bottom edge of a form

Similarly use < or << for the 1st element of the wherelc argument.

Try it:

```
←'ff'⍞wi'*Create' 'zForm'('*size'100 300)
←'ff'⍞wi'*.bnOK.Create'      'zButton'('wherelc' '<'  0   0   0)
←'ff'⍞wi'*.bnCancel.Create'  'zButton'('wherelc' '=' '>' '=' '=')
←'ff'⍞wi'*.bnOK.caption'     'OK'
←'ff'⍞wi'*.bnCancel.caption' 'Cancel'
←'ff'⍞wi'*Show'
```



Note that this time, we wanted the Cancel button to the right of the OK button, so we used 'wherelc' '=' '>' '=' '=' for the the bnCancel button.

The first = means: display bnCancel vertically at the same position as bnOK

Copyright © 2016 Eric Lescasse

The > means: display bnCancel horizontally after (i.e. to the right) of the bnOK button

Displaying controls at the bottom right of a form

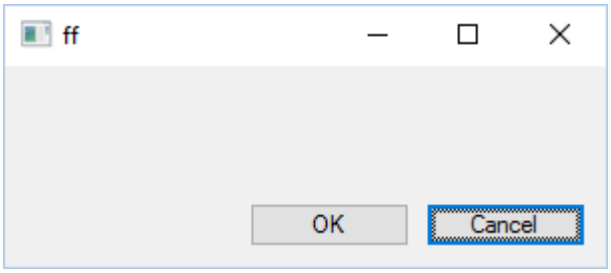
This is slightly more challenging, because we need to:

1. Create the Cancel button first and install it at the bottom right of the form
2. Then create the **bnOK** button and install it before the **bnCancel** button

Let's do it with what we learnt so far.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'100 300)
←'ff'□wi'*.bnCancel.Create'  'zButton'('wherelc' '<' '<'          0   0)
←'ff'□wi'*.bnOK.Create'      'zButton'('wherelc' '=' '<bnCancel' '=' '=')
←'ff'□wi'*.bnOK.caption'     'OK'
←'ff'□wi'*.bnCancel.caption' 'Cancel'
←'ff'□wi'*Show'
```

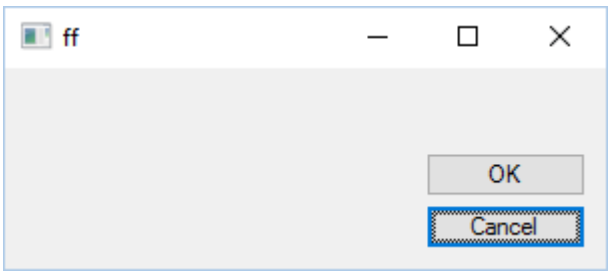


The bnCancel button is first installed at the bottom right of the form.

Then note the **<bnCancel** notation to indicate that the horizontal position of the bnOK button should be before (i.e. to the left of) the bnCancel button.

Exercise:

Use the proper **wherelc** properties to create the following user interface:



Copyright © 2016 Eric Lescasse

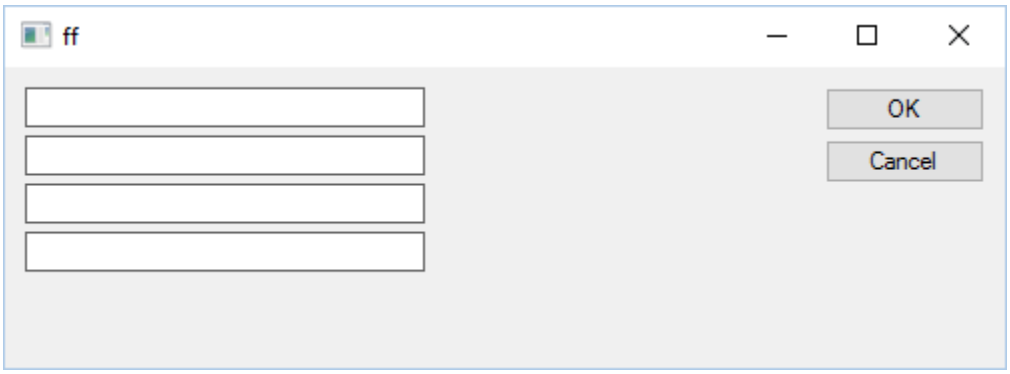
The reference control

When you use the wherelc property to position and size a control, wherelc calculates its position and size relative to the previous control paced on the form with wherelc.

This previous control is called the **Reference Control**.

At times it may be necessary to dynamically change the reference control.

Assume you want to create the following user interface:



and that you want to ensure that the OK button is at the very same vertical position as the 1st edit control (ed1).

You can achieve this as follows:

```
←'ff'□wi'*Create' 'zForm'('*size'150 500)
←'ff'□wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 200)
←'ff'□wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed3.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed4.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.bnOK.Create'      'zButton'('wherelc' '=ed1' '<'  0   0)
←'ff'□wi'*.bnCancel.Create'  'zButton'('wherelc' '>'    '=' '=' '=')
←'ff'□wi'*.bnOK.caption'     'OK'
←'ff'□wi'*.bnCancel.caption' 'Cancel'
←'ff'□wi'*Show'
```

By indicating **=ed1** for the bnOK vertical position you are:

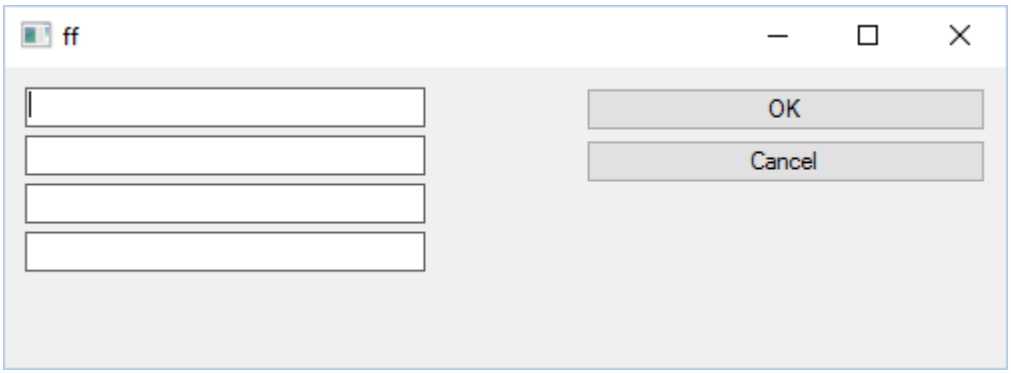
1. Temporarily changing the Reference Control to now be **ed1** (it was ed4 which was the previous control placed on the form)
2. Forcing the vertical position of the bnOK control to be the same as the ed1 one

It is important to note that ed1 has then become the Reference Control for the whole bnOK wherelc statement.

This may have implications.

Copyright © 2016 Eric Lescasse

Assume we change the 2nd 0 on this line by '='. Here is what we would get.



Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'150 500)
←'ff'□wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 200)
←'ff'□wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed3.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.ed4.Create' 'zEdit'('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.bnOK.Create'      'zButton'('wherelc' '=ed1' '<'  0  '=')
←'ff'□wi'*.bnCancel.Create'  'zButton'('wherelc' '>'    '=' '=' '=')
←'ff'□wi'*.bnOK.caption'     'OK'
←'ff'□wi'*.bnCancel.caption' 'Cancel'
←'ff'□wi'*Show'
```

Using an = symbol for the bnOK 4th element of the wherelc argument has now forced bnOK to have the same width as ed1 (because ed1 is now the Reference Control).

As a consequence the bnOK button has a much larger width.

Once the bnOK button is created it becomes the Reference Control.

And as a consequence of that, the bnCancel button which has a width equal to the width of the Reference Control is now also much larger!

Aligning Controls

Another frequent task to produce nice looking interfaces is to align some controls in the form.

Aligning controls by moving them

Assume we have a Grid and want to right align a Button below the Grid.

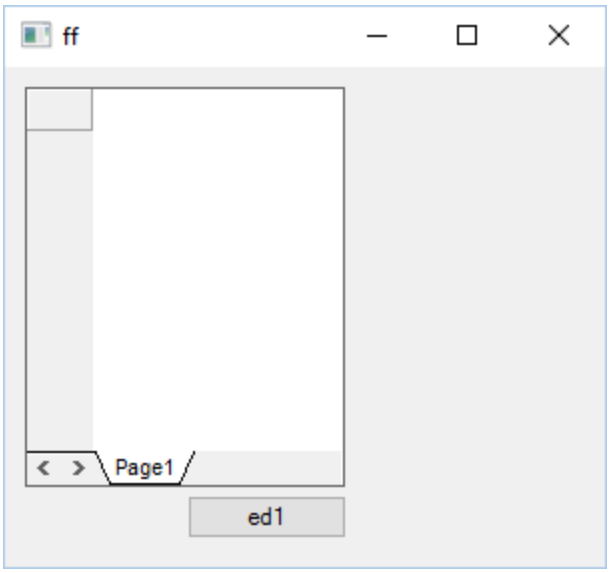
Use ⊣ to perform the alignment.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'250 300)
```

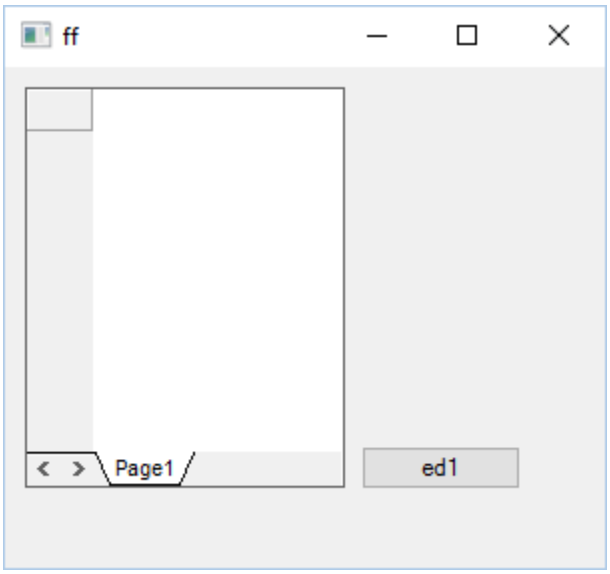
Copyright © 2016 Eric Lescasse

```
←'ff'⎕wi'*.gr1.Create' 'zGrid'('*border'1)('wherelc' 0 0 200 160)
←'ff'⎕wi'*.ed1.Create' 'zButton'('wherelc' '>' '⊣gr1' 0 0)
←'ff'⎕wi'*Show'
```



Similarly, to align the bottom of several controls, use the ⊥ symbol.

```
←'ff'⎕wi'*Create' 'zForm'('*size'250 300)
←'ff'⎕wi'*.gr1.Create' 'zGrid'('*border'1)('wherelc' 0 0 200 160)
←'ff'⎕wi'*.ed1.Create' 'zButton'('wherelc' '⊥gr1' '>' 0 0)
←'ff'⎕wi'*Show'
```



Aligning controls by extending them

In the previous example we used a symbol like ⊣ or ⊥ in the wherelc 2nd element because we did not want to change the object height or width to do the alignment.

Copyright © 2016 Eric Lescasse

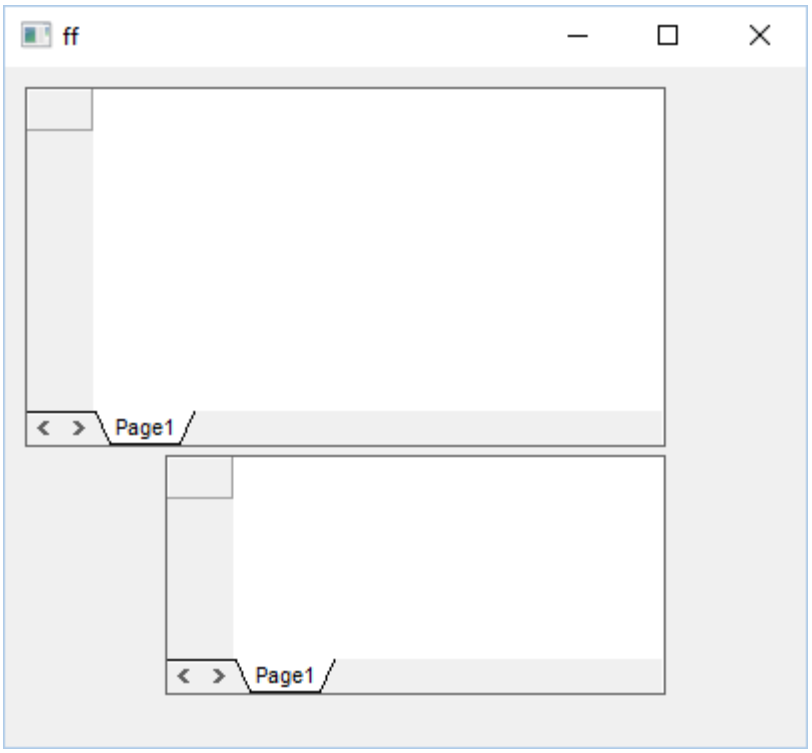
But sometimes, you need the opposite: you want to align a control to the right edge or bottom edge of another control, without changing the first control left corner position but by extending it.

Since this does not change the position but the height or width of the control we use a symbol in the 3rd or 4th wherelc element. The symbol to use is: >>control

Here are a couple of examples.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'340 400)
←'ff'□wi'*.gr1.Create' 'zGrid'('*border'1)('wherelc' 0 0 180 320)
←'ff'□wi'*.gr2.Create' 'zGrid'('*border'1)('wherelc' '>' 80 120 '>>gr1')
←'ff'□wi'*Show'
```

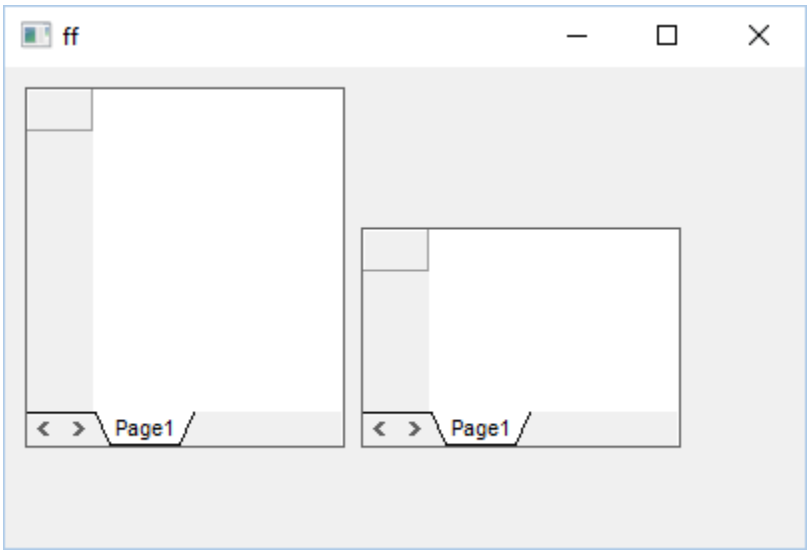


In the above example we wanted a second grid object to start at 80 pixels from the form left edge and to extend to align right with the first grid.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'240 400)
←'ff'□wi'*.gr1.Create' 'zGrid'('*border'1)('wherelc' 0 0 180 160)
←'ff'□wi'*.gr2.Create' 'zGrid'('*border'1)('wherelc' 80 '>' '>>gr1' '=')
←'ff'□wi'*Show'
```

Copyright © 2016 Eric Lescasse



In the above example we wanted a second grid object to start at 80 pixels from the top of the form and to extend to the bottom of the first grid.

Computing the control width based on its caption

When you add Label (zLabel, Check boxes (zCheck) or Radio buttons (zRadio) objects to your user interface you often need to be able to calculate the width of your control based on its caption.

In C# Windows Forms, you set the AutoSize property to true.

In zObjects you use the '⊂' symbol for the 4th wherelc element.

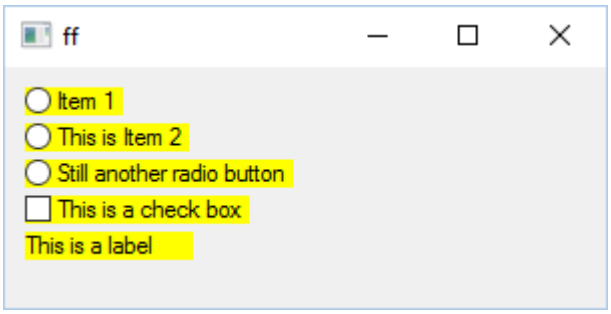
Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'120 300)
←'ff'□wi'*.op1.Create' 'zOption'('*caption' 'Item 1')('wherelc' 0 0 0 '⊂')
←'ff'□wi'*.op2.Create' 'zOption'('*caption' 'This is Item 2')('wherelc' '>'
      '=' '=' '⊂')
←'ff'□wi'*.op3.Create' 'zOption'('*caption' 'Still another radio
     button')('wherelc' '>' '=' '=' '⊂')
←'ff'□wi'*.ck1.Create' 'zCheck'('*caption' 'This is a check box')('wherelc'
      '>' '=' '=' '⊂')
←'ff'□wi'*.lb1.Create' 'zLabel'('*caption' 'This is a label')('wherelc' '>'
      '=' '=' '⊂')
←'ff'□wi'*.op1.color'255 255 0
←'ff'□wi'*.op2.color'255 255 0
←'ff'□wi'*.op3.color'255 255 0
←'ff'□wi'*.ck1.color'255 255 0
←'ff'□wi'*.lb1.color'255 255 0
←'ff'□wi'*Show'
```

Note that when you use ⊂ it is required that you set the *caption property before running wherelc.

Copyright © 2016 Eric Lescasse

I have forced the controls backcolor to yellow to allow show the control widths:



Performing Adjustments

As perfect as wherelc may be, there are times where you need or want to make adjustments to the location or size calculated by wherelc.

You use elements 5,6, 7 or 8 of the wherelc argument to perform these adjustments as follows:

3. The 5th wherelc element is the vertical position adjustment in pixel
4. The 6th wherelc element is th horizontal position adjustment in pixel
5. The 7th wherelc element is the height adjustment in pixels
6. The 8th wherelc element is the width adjustment in pixels

With these adjustments you can really achieve absolutely any user interface.

Assume you want to create a form with 4 groups of 2 Edit boxes and want to separate the groups by 20 pixels while the Edit boxes in the same group stay separated by the gap of 4 pixels. And assume you want 4 buttons, one per group, 40 pixels to the right of the first Edit box in each group.

Here is what you could come up with.

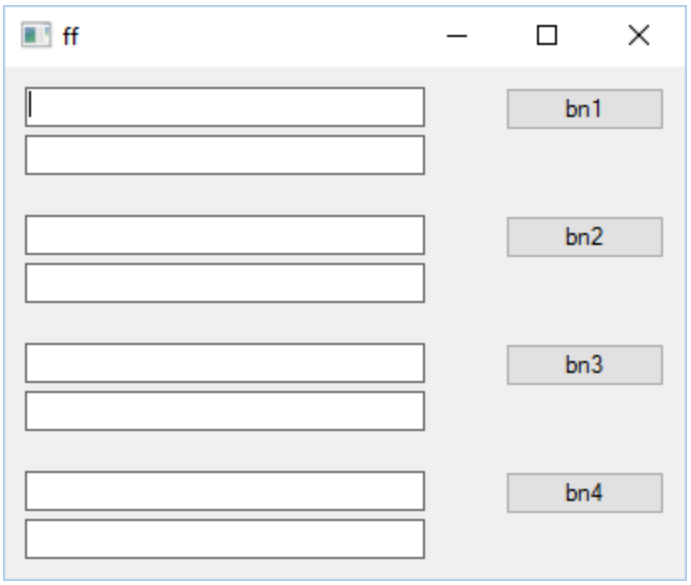
Try it:

```
←'ff'⎕wi'*Create' 'zForm'('*size'150 500)
←'ff'⎕wi'*.ed1.Create' 'zEdit'('wherelc'0 0 0 200)
←'ff'⎕wi'*.ed2.Create' 'zEdit'('wherelc' '>'  '=' '=' '=')
←'ff'⎕wi'*.ed3.Create' 'zEdit'('wherelc' '>>' '=' '=' '=20)
←'ff'⎕wi'*.ed4.Create' 'zEdit'('wherelc' '>'  '=' '=' '=')
←'ff'⎕wi'*.ed5.Create' 'zEdit'('wherelc' '>>' '=' '=' '=20)
←'ff'⎕wi'*.ed6.Create' 'zEdit'('wherelc' '>'  '=' '=' '=')
←'ff'⎕wi'*.ed7.Create' 'zEdit'('wherelc' '>>' '=' '=' '=20)
←'ff'⎕wi'*.ed8.Create' 'zEdit'('wherelc' '>'  '=' '=' '=')
←'ff'⎕wi'*.bn1.Create' 'zButton'('wherelc' '=ed1' '>>' 0 0 0 40)
←'ff'⎕wi'*.bn2.Create' 'zButton'('wherelc' '=ed3' '=bn1' '=' '=')
←'ff'⎕wi'*.bn3.Create' 'zButton'('wherelc' '=ed5' '=bn1' '=' '=')
←'ff'⎕wi'*.bn4.Create' 'zButton'('wherelc' '=ed7' '=bn1' '=' '=')
←'ff'⎕wi'AutoSize'
```

Copyright © 2016 Eric Lescasse

```
      ←'ff'□wi'*Show'
```

This creates the following form:



A few remarks:

1. First for ed3, ed5 and ed7, we use >> for the 1st wherelc element: this is because we want to create those Edit controls right below their Reference Controls with no gaps: the reason is that we want exactly 20 pixels between groups of Edit controls. If we had used >, ed3, ed5 and ed 7 would be created at a “gap” distance (i.e. 4 pixels) from the previous control pus the 20 pixels adjustment which would have resulted in a 24 pixels gap between the group of controls

2. Similarly, for bn1, we use >> to position it horizontally right after ed1, with no gap so that with the 40 pixels adjustment we get exactly 40 pixels between the controls

3. Also note that we don’t want to adjust the bn1 button vertical position, only its horizontal position, so we use 0 40 which means: adjust the vertical position by 0 pixels and the horizontal position by 40 pixels

4. Finally note that we change the Reference Control twice inside the wherelc specifications for buttons bn2, bn3 and bn4. The reason is that we use =ed1 to force the vertical position to be the same as the ed1 control, but we then want to refer to the bn1 button for the horizontal position, height and width.

All this may seem a little tricky, but with a little habit, it becomes natural and simple.

Copyright © 2016 Eric Lescasse

Changing only part of the position or size afterwards

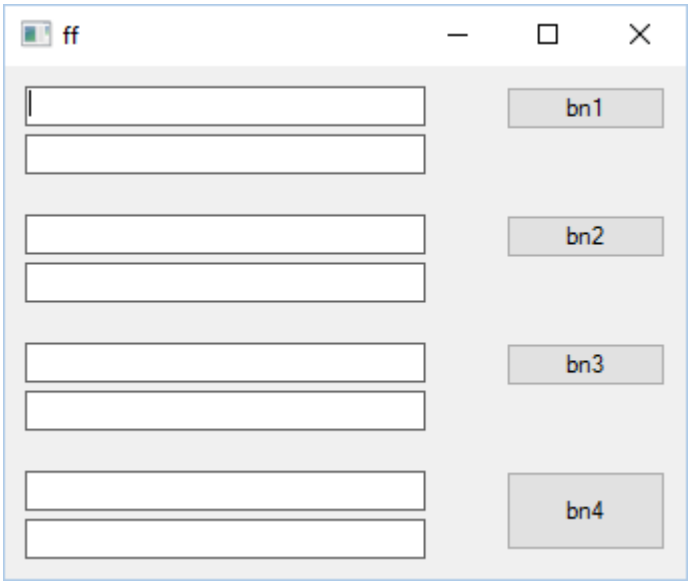
Assume you have already positioned and sized a control with wherelc in your form and that you later want to change just one of its wherelc elements without changing the others.

You can do that using the ∘ (alt+J) symbol.

Try it:

```
←'ff'⎕wi'.bn4.wherelc' '∘' '∘' 40 '∘'
```

This means: leave the position and the width of the bn4 button unchanged, but change its height to 40 pixels. This would result in the following change in the last form:



Centering a control horizontally

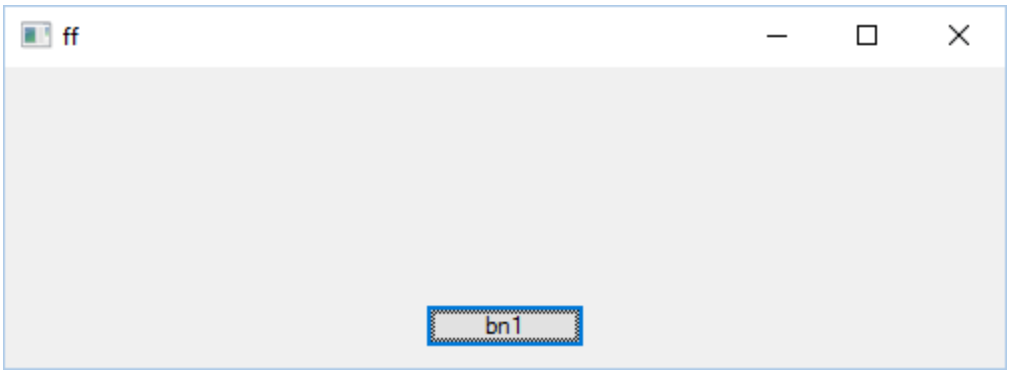
At times, you want to center a control horizontally in the control parent object.

Use the **|** symbol for this purpose.

Try it:

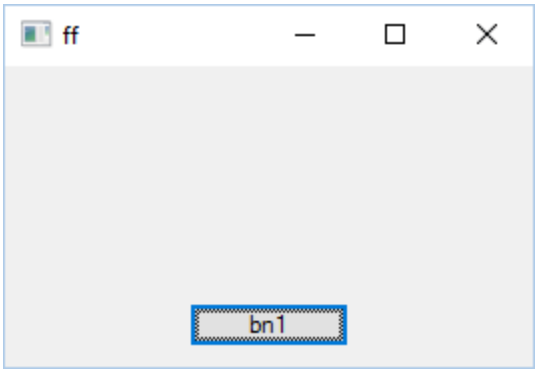
```
←'ff'⎕wi'*Create' 'zForm'('*size'150 500)
←'ff'⎕wi'*.bn1.Create' 'zButton'('wherelc' '<' '|' 0 0)
←'ff'⎕wi'*Show'
```

Copyright © 2016 Eric Lescasse



This creates a form with a button centered at the bottom.

Note that the button magically remains centered in the form when you resize the form:

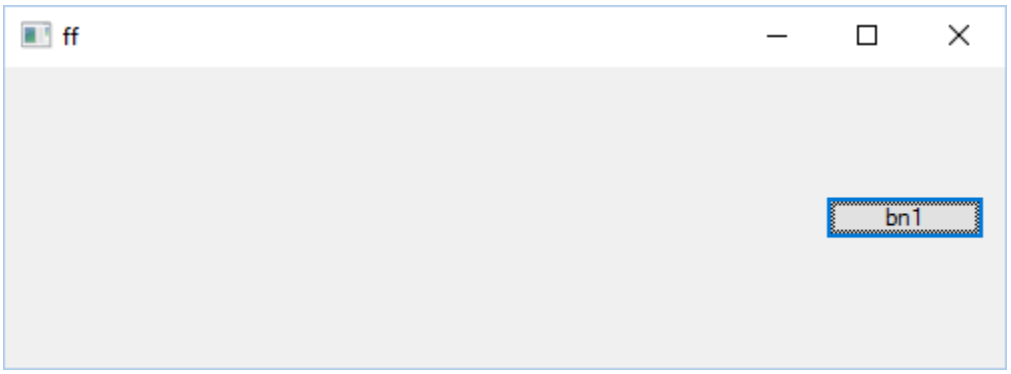


Centering a control vertically

To center a control vertically use the – symbol.

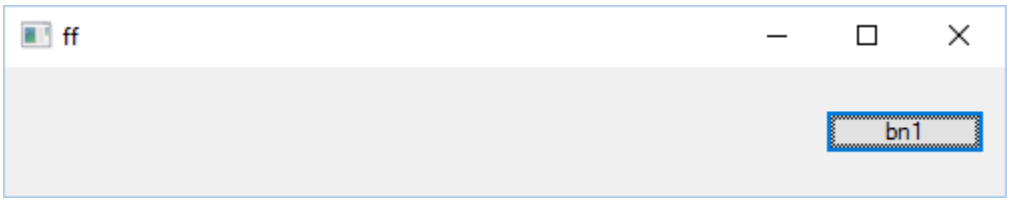
Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'150 500)
←'ff'□wi'*.bn1.Create' 'zButton'('wherelc' '-' '<' 0 0)
←'ff'□wi'*Show'
```



If you resize the form, the button remains vertically centered in the form:

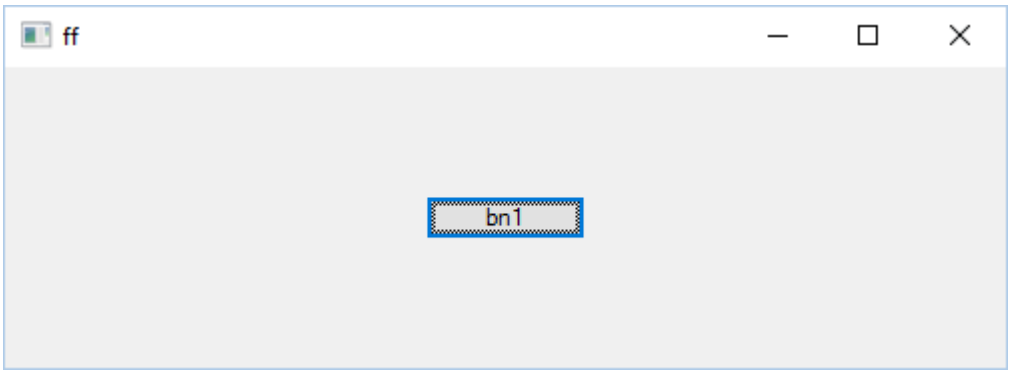
Copyright © 2016 Eric Lescasse



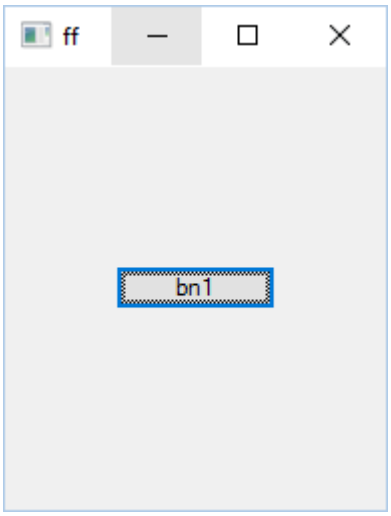
Centering a control bother vertically and horizontally

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'150 500)
←'ff'□wi'*.bn1.Create' 'zButton'('wherelc' '-' '|' 0 0)
←'ff'□wi'*Show'
```



The button remains centered when you resize the form.



Let's now raise the bar.

Centering multiple controls vertically

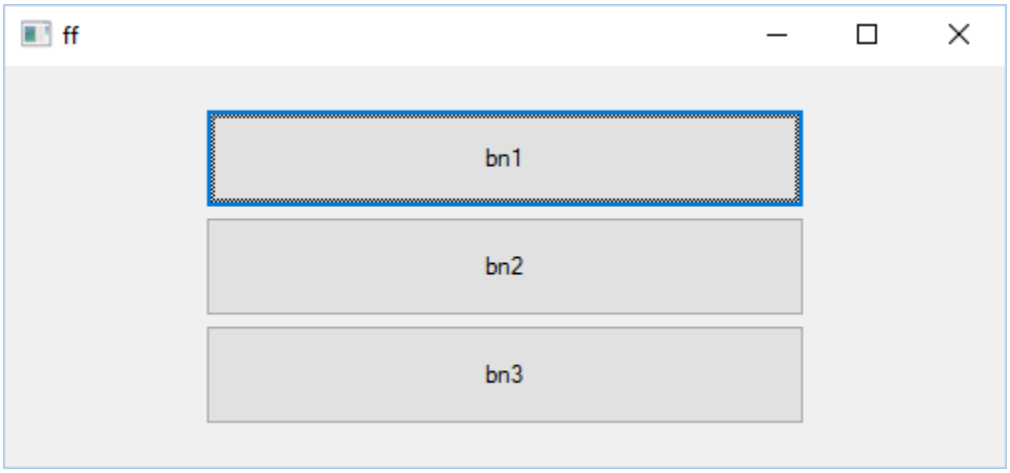
Centering multiple controls vertically or horizontally in the form is normally significantly more difficult.

Copyright © 2016 Eric Lescasse

However, with zObjects it is simple. Just use 2 – symbols instead of just one -.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'200 500)
←'ff'□wi'*.bn1.Create' 'zButton'('wherelc' '--' '|' 50 300)
←'ff'□wi'*.bn2.Create' 'zButton'('wherelc' '--' '|' '=' '=')
←'ff'□wi'*.bn3.Create' 'zButton'('wherelc' '--' '|' '=' '=')
←'ff'□wi'*Show'
```



When you resize the form the controls remain perfectly centered in the form. Or in their container.

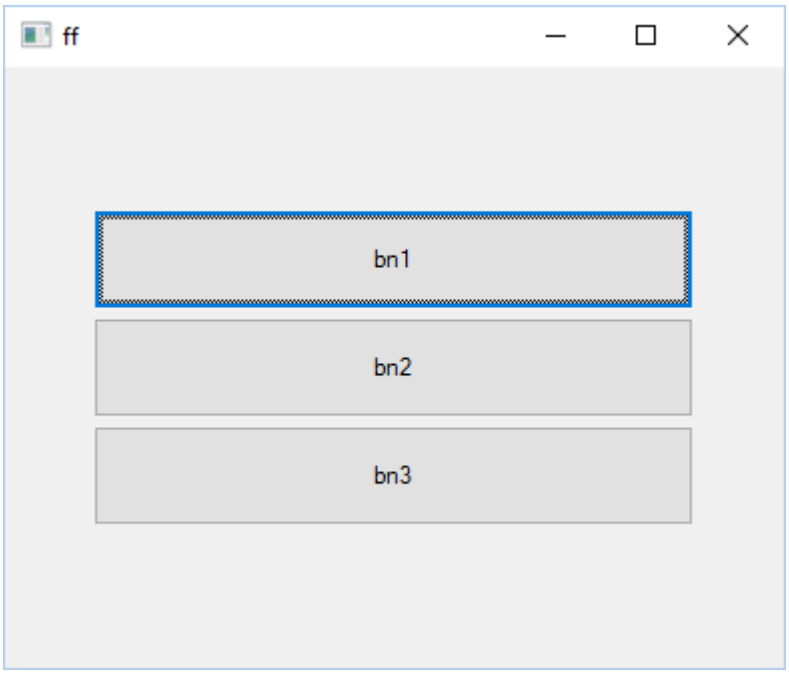
Note

It is important to note that zObjects support only one group of vertically centered controls in a given form or container.

All the controls that have – as their vertical position are considered part of the group of controls that need to be vertically centered in the form.

Note that we used only one | symbol in the wherelc elements, 2 – symbols thus giving priority to the vertical centering. Otherwise the form could not know if you want to center the controls horizontally or vertically.

Copyright © 2016 Eric Lescasse



You can use this technique of horizontally and vertically centering a set of large buttons in a form to build a starting menu for your application.

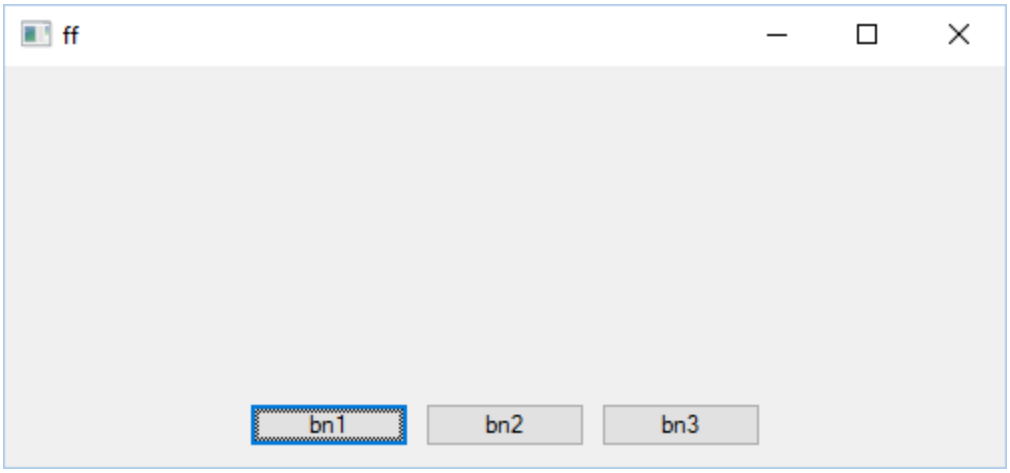
Centering multiple controls horizontally

The principle is the same. Just use || to indicate that you want to horizontally center all the controls that have a || position.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'200 500)
←'ff'□wi'*.bn1.Create' 'zButton'('wherelc' '<' '||'  0   0)
←'ff'□wi'*.bn2.Create' 'zButton'('wherelc' '=' '||' '=' '=')
←'ff'□wi'*.bn3.Create' 'zButton'('wherelc' '=' '||' '=' '=')
←'ff'□wi'*Show'
```

Copyright © 2016 Eric Lescasse



When you resize the form, the controls remain centered on the form or in their container.

> **Note**
>
> It is important to note that zObjects support only one group of horizontally centered controls in a given form or container.
>
> All the controls that have || as their horizontal position are considered part of the group of controls that need to be horizontally centered in the form.

Centering a control on another control

You may sometimes wish to center a control based on another control.

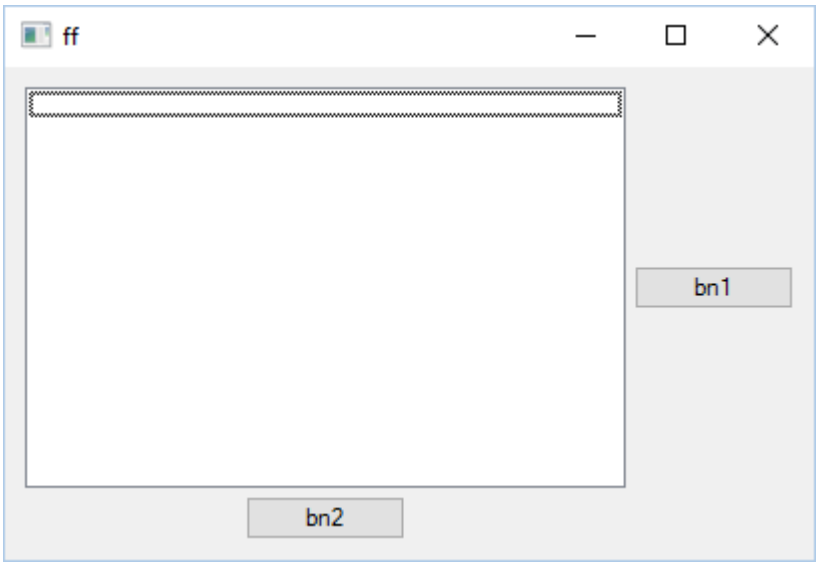
To center a control vertically based on another control use **-control** for your wherelc 1st element.

To center a control horizontally based on another control use **|control** for your wherelc 2nd element.

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'250 500)('gaps'4 4)
←'ff'□wi'*.ls1.Create' 'zList'('wherelc' 0 0 200 300)
←'ff'□wi'*.bn1.Create' 'zButton'('wherelc' '-ls1' '>'    0 0)
←'ff'□wi'*.bn2.Create' 'zButton'('wherelc' '>ls1' '|ls1' 0 0)
←'ff'□wi'AutoSize'
←'ff'□wi'*Show'
```

Copyright © 2016 Eric Lescasse



Copyright © 2016 Eric Lescasse

Automatic resizing with the anchor property

Introduction

One of the most cumbersome task with APL user interfaces is to properly resize all controls on your form when the form is resized.

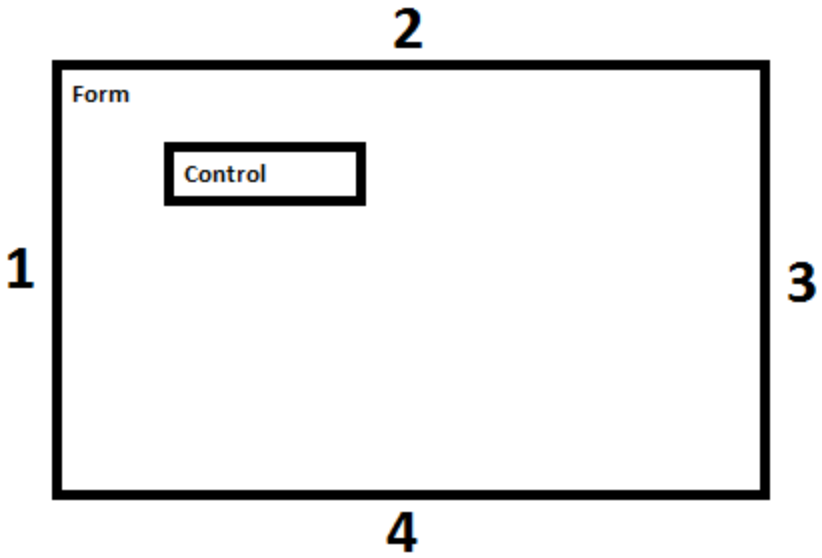
You can solve this problem very easily and to the perfection using the **anchor**[7] property in combination with the **whereIc** property.

The rule is: all controls which should resize when your form is resized must have an **anchor** property defined.

Basics

The anchor property must have a 4-element integer vector argument. The possible values are 1, 2, 3 and 4 and represent the edges of your form.

Consider one Form and one Control:



The left, top, right and bottom Form edges are numbered 1, 2, 3 and 4.

The anchor property tells to which form edge each of the control left, top, right and bottom edges should be attached.

By default, the anchor property value is: **1 2 1 2**

This means:

[7] The zObjects **anchor** property is very similar to the C# Windows Forms **Anchor** property

Copyright © 2016 Eric Lescasse

- The control left edge is attached to the form left edge (1)
- The control top edge is attached to the form top edge (2)
- The control right edge is attached to the form left edge (1)
- The control bottom edge is attached to the form top edge (2)

Being “attached to” means “remain at the same distance from” when the form is resized.

Note:

If using the 1, 2, 3 and 4 numbers seem confusing to you, you can alternatively use the following letters:

l	to indicate the control must be attached to the left edge of the form
t	to indicate the control must be attached to the top edge of the form
r	to indicate the control must be attached to the right edge of the form
b	to indicate the control must be attached to the right edge of the form

So, specifying the following expressions are equivalent:

(‘anchor’ 1 2 1 2)	is equivalent to:	(‘anchor’ ‘tl’)
(‘anchor’ 1 2 3 2)	is equivalent to:	(‘anchor’ ‘tlr’)
(‘anchor’ 3 2 3 4)	is equivalent to:	(‘anchor’ ‘trb’)
(‘anchor’ 1 4 3 4)	is equivalent to:	(‘anchor’ ‘lrb’)
(‘anchor’ 3 4 3 4)	is equivalent to:	(‘anchor’ ‘rb’)
(‘anchor’ 1 2 3 4)	is equivalent to:	(‘anchor’ ‘lrtb’)

Note that the letter order is irrelevant: **lrtb** is same as **btlr** or **lbtr**

Defining the anchor property using letters may be simpler for developers also using C#.

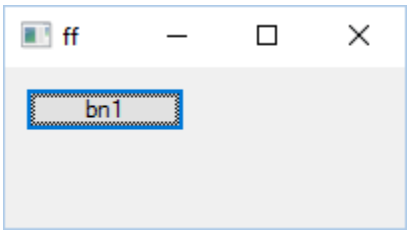
Examples

Create a control attached to the left and right edge of the form

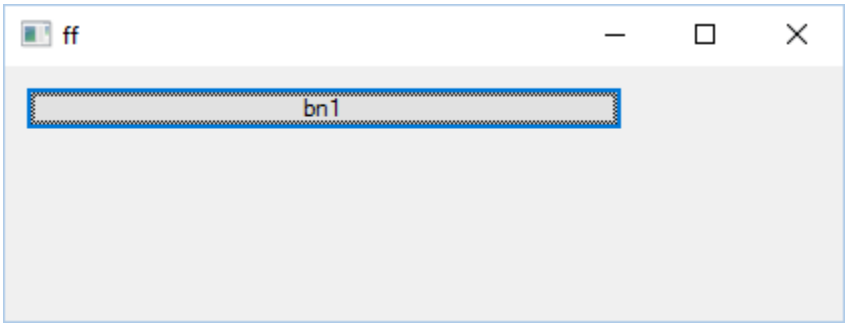
Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'80 200)
←'ff'□wi'*.bn1.Create' 'zButton'('wherelc' 0 0 0 80)('anchor'1 2 3 2)
←'ff'□wi'*Show'
```

Copyright © 2016 Eric Lescasse



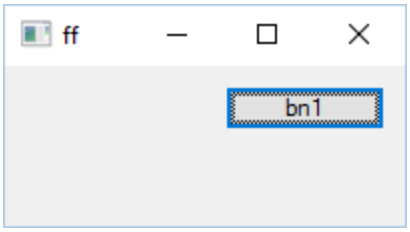
Resize the form:



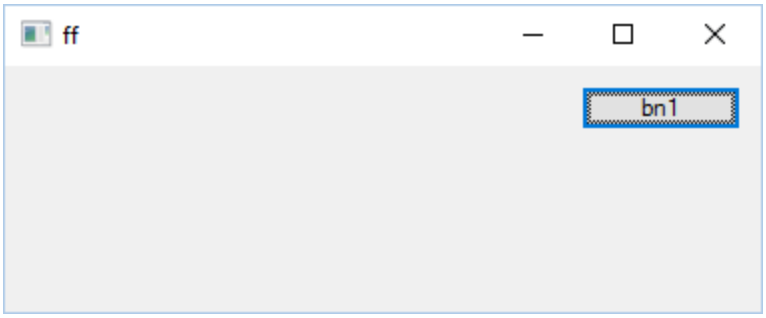
Create a button attached to the right edge of the form

Try it:

```
←'ff'□wi'*Create' 'zForm'('*size'80 200)
←'ff'□wi'*.bn1.Create' 'zButton'('wherelc' 0 0 0 80)('anchor'3 2 3 2)
←'ff'□wi'*Show'
```



Resize the form:



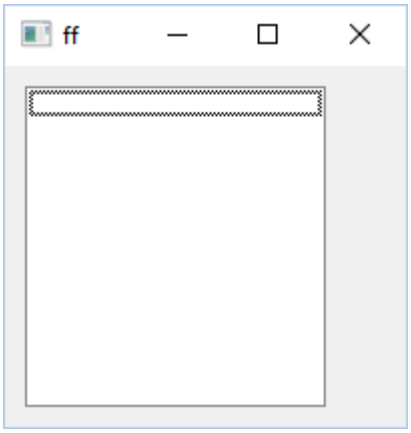
Create a control attached to the left, top and bottom of the form

Try it:

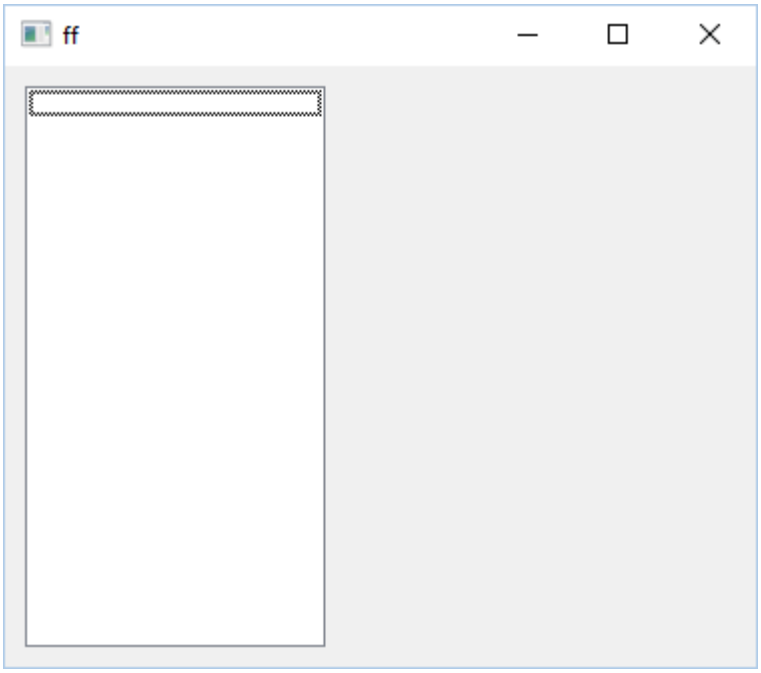
```
←'ff'□wi'*Create' 'zForm'('*size'180 200)
```

Copyright © 2016 Eric Lescasse

```
      ←'ff'⎕wi'*.bn1.Create' 'zList'('wherelc' 0 0 '>' 150)('anchor'1 2 1 4)
      ←'ff'⎕wi'*Show'
```



Resize the form:

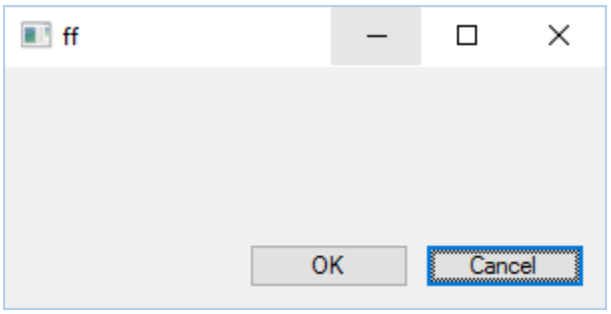


Create two buttons attached to the bottom and right edge of the form

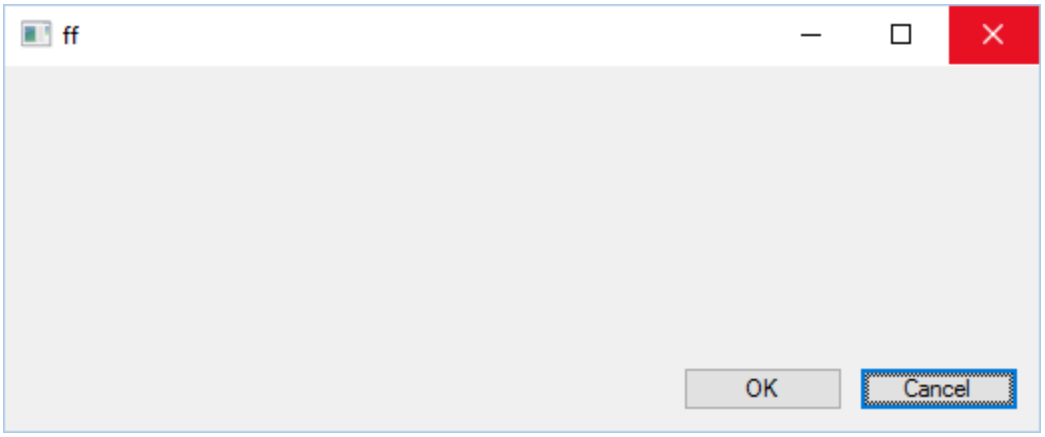
Try it:

```
      ←'ff'⎕wi'*Create' 'zForm'('*size'120 300)
      ←'ff'⎕wi'*.bnca.Create' 'zButton'('wherelc' '<' '<'0 0)('anchor' 'rb')
      ←'ff'⎕wi'*.bnok.Create' 'zButton'('wherelc' '=' '<bnca' '=' '=')('anchor' 'rb')
      ←'ff'⎕wi'*.bnok.caption' 'OK'
      ←'ff'⎕wi'*.bnca.caption' 'Cancel'
      ←'ff'⎕wi'*Show'
```

Copyright © 2016 Eric Lescasse



Resize the form:



Note that we have used the 'rb' notation this time to indicate that the buttons are attached to the **r**ight and to the **b**ottom of the form. We could have used: 3 4 3 4 instead of 'rb'

Note:

Let it be very clear that:

1. The **whereIc** property is used during the construction of your form to give your controls their initial positions and sizes

2. The **anchor** property is recorded during the construction of the form, but is only used during a form resize operation

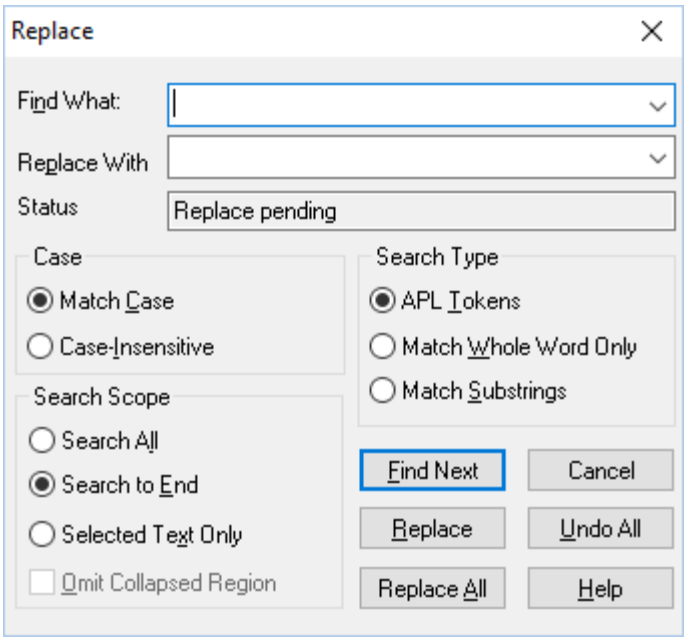
Conclusion

The **whereIc** and **anchor** properties, together with the **gaps** and **margins** properties are extremely powerful tools which not only allow you to really create any imaginable APL user interface easily, but also result in absolutely perfect looking APL forms.

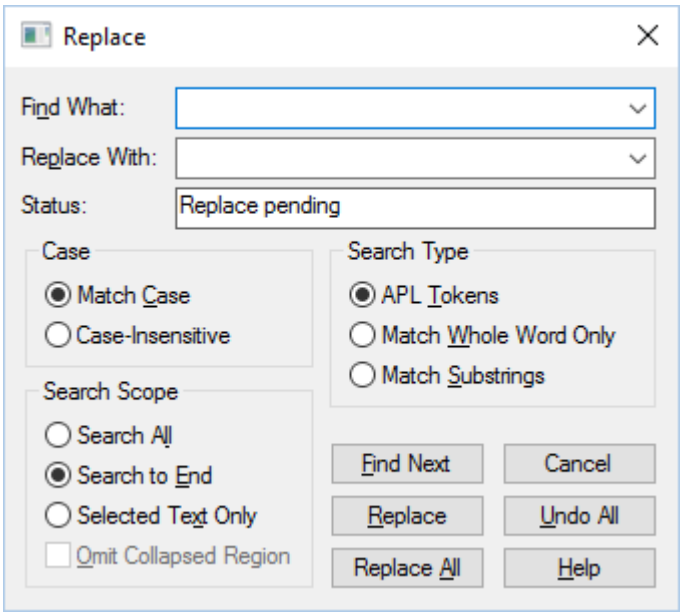
Copyright © 2016 Eric Lescasse

Case Study:
Reproduce the APL+Win Replace dialog

Let's use what we have learnt so far to create a resizable dialog box identical to the APL+Win Replace dialog (which you get by pressing Ctrl+H in APL).
This dialog looks as follows:

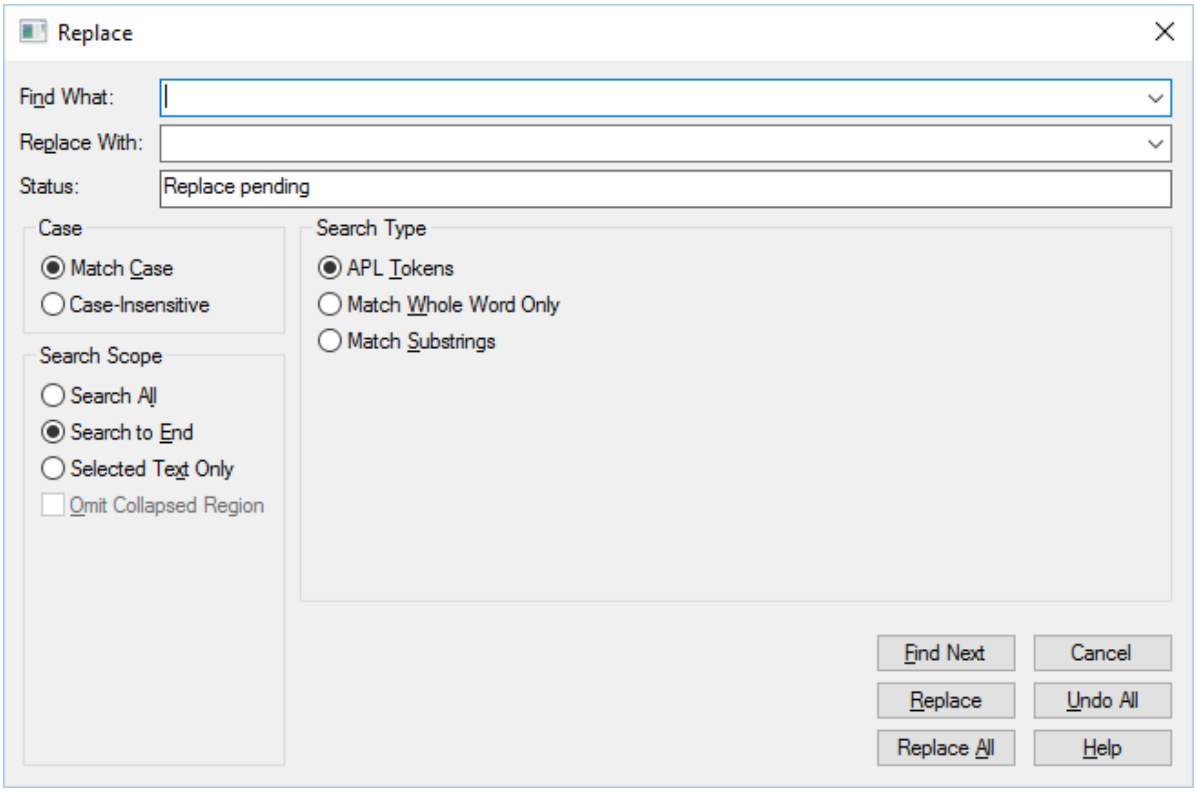


I'll show you the APL function that recreated the same dialog using **wherelc** and **anchor** soon, but here is what I could come up with in just a few minutes:



Copyright © 2016 Eric Lescasse

Moreover, even though there is no real need for that, the zObjects version is resizable. Let's resize the dialog:



The function that I wrote using **wherelc** and **anchor** to create this dialog is reproduced below:

Copyright © 2016 Eric Lescasse

```
    ∇ zzreplace
[1]   ⍝∇ Clone the APL+Win Replace dialog using the wherelc and anchor properties
[2]
[3]   ←'ff'⎕wi'*Create' 'zForm'('*border'2 16)('*caption' 'Replace')
[4]   ←'ff'⎕wi'*.cb1.Create' 'zCombo'
[5]   ←'ff'⎕wi'*.cb2.Create' 'zCombo'
[6]   ←'ff'⎕wi'*.ed1.Create' 'zEdit'('*text' 'Replace pending')
[7]   ←'ff'⎕wi'*.cb1.Set'('wherelc'0 85 0 240)('anchor' 'tlr')
[8]   ←'ff'⎕wi'*.cb1.Set'('caption' 'Fi&nd What:'0)
[9]   ←'ff'⎕wi'*.cb2.Set'('wherelc' '>' '=' '=' '=')('anchor' 'tlr')
[10]  ←'ff'⎕wi'*.cb2.Set'('caption' 'Re&place With:'0)
[11]  ←'ff'⎕wi'*.ed1.Set'('wherelc' '>' '=' '=' '=')('anchor' 'tlr')
[12]  ←'ff'⎕wi'*.ed1.Set'('caption' 'Status:'0)
[13]
[14]  ←'ff'⎕wi'*.fr1.Create' 'zFrame'('*caption' 'Case')('gaps'6 6)
[15]  ←'ff'⎕wi'*.fr1.op1.Create' 'zOption'('*caption' 'Match &Case')('*value'1)
[16]  ←'ff'⎕wi'*.fr1.op2.Create' 'zOption'('*caption' 'Case-Insensitive')
[17]  ←'ff'⎕wi'*.fr1.Set'('wherelc' '>'0 0 0)
[18]  ←'ff'⎕wi'*.fr1.op1.Set'('wherelc'22 0 0'<')
[19]  ←'ff'⎕wi'*.fr1.op2.Set'('wherelc' '>' '=' '=' '<')
[20]  ←'ff'⎕wi'.fr1.AutoSize'
[21]
[22]  ←'ff'⎕wi'*.fr2.Create' 'zFrame'('*caption' 'Search Scope')('gaps'6 6)
[23]  ←'ff'⎕wi'*.fr2.op3.Create' 'zOption'('*caption' 'Search A&ll')
[24]  ←'ff'⎕wi'*.fr2.op4.Create' 'zOption'('*caption' 'Search to &End')('*value'1)
[25]  ←'ff'⎕wi'*.fr2.op5.Create' 'zOption'('*caption' 'Selected Te&xt Only')
[26]  ←'ff'⎕wi'*.fr2.ck1.Create' 'zCheck'('*caption' '&Omit Collapsed
            Region')('*enabled'0)
[27]  ←'ff'⎕wi'*.fr2.Set'('wherelc' '>fr1'0 0 '=')('anchor' 'ltb')
[28]  ←'ff'⎕wi'*.fr2.op3.Set'('wherelc'22 0 0'<')
[29]  ←'ff'⎕wi'*.fr2.op4.Set'('wherelc' '>' '=' '=' '<')
[30]  ←'ff'⎕wi'*.fr2.op5.Set'('wherelc' '>' '=' '=' '<')
[31]  ←'ff'⎕wi'*.fr2.ck1.Set'('wherelc' '>' '=' '=' '<')
[32]  ←'ff'⎕wi'.fr2.AutoSize'
[33]  ←'ff'⎕wi'.fr1.wherelc' '∘' '∘' '∘'(4⊃'fr2'⎕wi'*where')
[34]
[35]  ←'ff'⎕wi'*.fr3.Create' 'zFrame'('*caption' 'Search Type')('gaps'6 6)
[36]  ←'ff'⎕wi'*.fr3.op6.Create' 'zOption'('*value'1)('*caption' 'APL &Tokens')
[37]  ←'ff'⎕wi'*.fr3.op7.Create' 'zOption'('*caption' 'Match &Whole Word Only')
[38]  ←'ff'⎕wi'*.fr3.op8.Create' 'zOption'('*caption' 'Match &Substrings')
[39]  ←'ff'⎕wi'*.fr3.Set'('wherelc' '=fr1' '>' 92 '>>cb1')('anchor' 'ltrb')
[40]  ←'ff'⎕wi'*.fr3.op6.Set'('wherelc'22 0 0'<')
[41]  ←'ff'⎕wi'*.fr3.op7.Set'('wherelc' '>' '=' '=' '<')
[42]  ←'ff'⎕wi'*.fr3.op8.Set'('wherelc' '>' '=' '=' '<')
[43]  ←'ff'⎕wi'.fr3.AutoSize'
[44]  ←'ff'⎕wi'.fr3.wherelc' '∘' '∘' '∘' '>>cb1'
[45]
[46]  ←'ff'⎕wi'*.bn1.Create' 'zButton'('*caption' '&Find Next')
[47]  ←'ff'⎕wi'*.bn2.Create' 'zButton'('*caption' '&Replace')
[48]  ←'ff'⎕wi'*.bn3.Create' 'zButton'('*caption' 'Replace &All')
[49]  ←'ff'⎕wi'*.bn4.Create' 'zButton'('*caption' 'Cancel')
[50]  ←'ff'⎕wi'*.bn5.Create' 'zButton'('*caption' '&Undo All')
[51]  ←'ff'⎕wi'*.bn6.Create' 'zButton'('*caption' '&Help')
[52]  ←'ff'⎕wi'*.bn1.Set'('wherelc' '>fr3' '=' 0 77 12)('anchor' 'rb')
[53]  ←'ff'⎕wi'*.bn2.Set'('wherelc' '>' '=' '=' '=')('anchor' 'rb')
[54]  ←'ff'⎕wi'*.bn3.Set'('wherelc' '>' '=' '=' '=')('anchor' 'rb')
[55]  ←'ff'⎕wi'*.bn4.Set'('wherelc' '=bn1' '>' '=' '=')('anchor' 'rb')
[56]  ←'ff'⎕wi'*.bn5.Set'('wherelc' '>' '=' '=' '=')('anchor' 'rb')
[57]  ←'ff'⎕wi'*.bn6.Set'('wherelc' '>' '=' '=' '=')('anchor' 'rb')
```

Copyright © 2016 Eric Lescasse

```
[58]
[59]  ←'ff'⎕wi'AutoSize'
[60]  ←'ff'⎕wi'minimumsize'((2⍴'#'⎕wi'units')×2↓'ff'⎕wi'*where')
[61]  ←'ff'⎕wi'*Show'
[62]
    ∇
```

I would normally have created one line per control, but for editing/reading purposes and to avoid function lines spanning 2 lines in this document I have cut lines in pieces, using the APL+Win *Set method.

The interesting parts of this function, at this stage, concerns only the use of the **wherelc** and **anchor** properties, but before I extract only the lines for an easier study, let's make a few remarks.

1. First, in a real world application I would have given meaningful names to all controls
2. Second, note the use of the 'caption' property (not '*caption') for the first 3 controls: it automatically creates the ComboBox Label or the Edit control Label and vertically aligns its caption with the text typed in the Combo Box or Edit control
3. Third, note line 33 where I have to recompute the width of the first frame based on the width of the second frame
4. Then, similarly, note line 44 where I recompute the fr3 Frame width so that it right aligns perfectly with the top controls in the form
5. On lines 18, 19, 28, 29 , 30, 31, 40, 41 and 42 note the use of '⊂' which sets the width of the control based on the width of its caption
6. Also note the use of the 12 pixels vertical adjustment I made on line 52: this was to force the bottom buttons to vertically align with the bottom of the fr2 Frame.
7. I have had to try a couple of button widths before finding 77 was the right one on line 52. Once found, all other buttons automatically get the same widths thanks to their '=' width spec
8. On line 60 I have had to convert the current form where property values to pixels in order to properly set the minimumsize property which must be set in pixels

You may find that the above function is complicated. However, anyone who has ever tried to develop such a dialog box with raw APL instructions to achieve the same result as the above dialog, knows it is a real pain and takes a lot of time and efforts. Moreover, if one wants his dialog written with raw APL instructions to resize nicely, then its even more complicated.

Here is the list of instructions making use of wherelc and anchor in the above function:

```
←'ff'⎕wi'*.cb1.Set'     ('wherelc'  0   85  0  240)        ('anchor' 'tlr')
←'ff'⎕wi'*.cb2.Set'     ('wherelc' '>' '=' '=' '=')        ('anchor' 'tlr')
←'ff'⎕wi'*.ed1.Set'     ('wherelc' '>' '=' '=' '=')        ('anchor' 'tlr')
←'ff'⎕wi'*.fr1.Set'     ('wherelc' '>'  0   0   0 )
←'ff'⎕wi'*.fr1.op1.Set' ('wherelc'  22  0   0  '>')
←'ff'⎕wi'*.fr1.op2.Set' ('wherelc' '>' '=' '=' '=')
←'ff'⎕wi'*.fr2.Set'     ('wherelc' '>fr1' 0  0 '=')        ('anchor' 'ltb')
```

Copyright © 2016 Eric Lescasse

```
←'ff'□wi'*.fr2.op3.Set' ('wherelc'  22  0   0  '>')
←'ff'□wi'*.fr2.op4.Set' ('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.fr2.op5.Set' ('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.fr2.ck1.Set' ('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.fr3.Set'     ('wherelc' '=fr1'  '>' 0 '>>cb1')  ('anchor' 'ltrb')
←'ff'□wi'*.fr3.op6.Set' ('wherelc'  22  0   0  '>')
←'ff'□wi'*.fr3.op7.Set' ('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.fr3.op8.Set' ('wherelc' '>' '=' '=' '=')
←'ff'□wi'*.bn1.Set'     ('wherelc' '>fr3' '=' 0 77 12)     ('anchor' 'rb')
←'ff'□wi'*.bn2.Set'     ('wherelc' '>' '=' '=' '=')        ('anchor' 'rb')
←'ff'□wi'*.bn3.Set'     ('wherelc' '>' '=' '=' '=')        ('anchor' 'rb')
←'ff'□wi'*.bn4.Set'     ('wherelc' '=bn1' '>' '=' '=')     ('anchor' 'rb')
←'ff'□wi'*.bn5.Set'     ('wherelc' '>' '=' '=' '=')        ('anchor' 'rb')
←'ff'□wi'*.bn6.Set'     ('wherelc' '>' '=' '=' '=')        ('anchor' 'rb')
```

When you look at all your wherelc instructions, the number of numeric values you find gives you an indication of what you've had to find by yourself. All the rest was automatically calculated by wherelc.

Copyright © 2016 Eric Lescasse

Creating your own Objects

Anatomy of a zObject

First let's study the anatomy of a **zObject.** It has the following structure:

```
    ∇ a zTemplate b;c;d;e;f;g;h;i;j;k;l;m;n;o;p;q;r;s;t;u;v;w;x;y;z;⎕io
;arg1;arg2;arg3;result
[1]   ⍝∇ a zTemplate b -- The zTemplate Object is REPLACEME
[2]   ⍝∇ a ↔ object name (may be omitted if ⎕wself is set)
[3]   ⍝∇ b ↔ 'property'
[4]   ⍝∇   or 'property'value
[5]   ⍝∇   or 'Method'
[6]   ⍝∇   or 'Method'argument1 ... argumentN
[7]   ⍝∇ Requires: (F) zObject
[8]   ⍝∇ zObjects v1.83
[9]   ⍝∇ Copyright(c)2013 Eric Lescasse
[10]  ⍝∇ ELE10nov13
[11]
[12]  ⎕io←1
[13]  :if ⎕monadic ⋄ a←⎕wself ⋄ :endif
[14]  :if b≡'Action' ⋄ →('?'=↑b←↑⎕warg)⍴⍙⍙Help ⋄ :endif
[15]  →(8≠''⍴⎕idloc'⍙',b)⍴⍙⍙Inherit ⋄ →⍎'⍙',b
[16]  ⍝∇-------------------------------------------------New
[17]  ⍙New:
[18]      ⍝∇ Create a new instance of zTemplate
[19]      ⍝∇ Example:
[20]      ⍝∇       'zt'⎕wi'*Create' 'zTemplate'
[21]      z←a⎕wi'*Create' 'Menu'
[22]      z←a⎕wi'*onAction'(⍎'zTemplate' 'zObject',¨⊂'"Action"',⎕tcnl)
[23]      z←a⎕wi'*onClick' 'zTemplate"onClick"'
[24]      :if(c←a⎕wi'*name')≠d←a⎕wi'*self' ⋄ z←'#'⎕wi'SetLinks'c d ⋄ :endif
[25]      :return
[26]  ⍝∇-------------------------------------------------class
[27]  ⍙class:
[28]      ⎕wres←'zTemplate'
[29]      :return
[30]  ⍝∇-------------------------------------------------help
[31]  ⍙help:
[32]      ⍝∇ 0 0⍴'zt'⎕wi'*Create' 'zTemplate'
[33]      ⎕wres←⎕wi'GetHelp' 'help'
[34]      :return
[35]  ⍝∇-------------------------------------------------onClick
[36]  ⍙onClick:
[37]      ⎕←'The onClick event fired!'
[38]      :return
[39]  ⍝∇-------------------------------------------------property
[40]  ⍙property:
[41]      ⍝∇ Comment describing the property
[42]      ⍝∇ Syntax: {value←}'obj'⎕wi'property'{value}
[43]      ⍝∇ value: describe the property value
[44]      ⍝∇ Example:
[45]      ⍝∇       'myobj'⎕wi'property'1234
[46]      :if 1=⍴⎕warg
[47]          ⎕wres←⎕wi'*⍙⍙property'
[48]      :else
```

Copyright © 2016 Eric Lescasse

```
[49]          z←⎕wi'*ΔΔproperty'(1↓⎕warg)
[50]      :endif
[51]      :return
[52]  ⍝∇-----------------------------------------------Method
[53]  Δ̲Method:
[54]      ⍝∇ Comment describing the Method
[55]      ⍝∇ Syntax: {result←}'obj'⎕wi'Method'{arg1}{arg2}...{argN}
[56]      ⍝∇ arg1: describe the method 1st argument
[57]      ⍝∇ arg2: describe the method 2nd argument
[58]      ⍝∇ ...
[59]      ⍝∇ argN: describe the method Nth argument
[60]      ⍝∇ result: describe the method result if any
[61]      ⍝∇ Example:
[62]      ⍝∇       'myobj'⎕wi'Method'1234'Test'(2 2⍴⍳4)
[63]      (arg1 arg2 arg3)←1↓⎕warg
[64]      ⍝ do some processing
[65]      ⍝ optionally calculate the result (if any) and return it:
[66]      ⎕wres←result←arg1 arg2 arg3
[67]      :return
[68]
[69]
[70]  ⍝-------------------------------------------------
[71]  ΔΔHelp:
[72]      ⎕wres←⎕wi'GetHelp'(('?'=⍭b)↓b)
[73]      :return
[74]  ⍝-------------------------------------------------
[75]  ΔΔInherit:
[76]      ⎕wres←⎕wi'*'
    ∇
```

Note that:

A typical **zObject** like **zTemplate** is self-contained: it encapsulates all its **Properties**, **Methods** and **Events**.

The **New** method is the **Constructor** and is the code which is executed when you create an instance of the object:

```
'zt'⎕wi'*Create' 'zTemplate'
```

For a Custom Object to work in APL+Win, you must create an instance of a standard APL+Win object in the constructor. When your custom Object is not a visual Object directly based on a standard APL+Win object (like **Form**, **Button**, etc.) we use the APL+Win **Menu** object (as done on line **21**) because it is a top level object, not directly visible, and because it is the standard APL+Win object that has the smallest memory footprint.

Line **22** is essential: this is where you define inheritance and this is the one that allows your custom **Properties** and **Methods** to be called. In **zTemplate** we indicate that **zTemplate** inherits from **zObject**. If we needed **zTemplate** to inherit from **zObject1**, **zObject2** and **zObject**, line **22** would read:

```
[22]      z←a⎕wi'*onAction'(∈'zTemplate' 'zObject1' 'zObject2' 'zObject',¨⊂'"Action"', ⎕tcnl)
```

Copyright © 2016 Eric Lescasse

The remaining of our template object contains properties (**class**, **help**, **property**), an event (**onClick**) and a sample method (**Method**).

To use a custom object, you must first have executed **zzInit** once in your APL Session to initialize the **zObjects** System: you can then create an instance of your object and start using its properties, methods and events:

```
      zzInit

      'zt'⎕wi'*Create' 'zTemplate'
zt

      'zt'⎕wi'property'(⍳10)

      'zt'⎕wi'property'
1 2 3 4 5 6 7 8 9 10

      'zt'⎕wi'Method'10 20 30
10 20 30
```

So, all this is very simple.

If you try to call a property which is not defined in your object itself, the system does not find a label corresponding to the property and falls through to execute line **76**. This results in starting searching for a label corresponding to the property you're calling in the next function in the inheritance tree (in our case, in **zObject**).

Example:

```
      'zt'⎕wi'version'
1.83
```

The **version** property does not exist in **zTemplate**, but is defined in **zObject** which **zTemplate** inherits from, so we can use it.

If you try using a property or method which is neither defined in your object nor in any of the objects it inherits from, you get an error:

```
      'zt'⎕wi'NonExistingMethod'
Unknown Method: NonExistingMethod. Did you intend to use: *NonExistingMethod  instead?
```

The reason you get this message is that there is still the possibility that you intended to use a **property** or **Method** defined in the underlying APL+Win standard object (here: **Menu**) your custom object is based on. If this is the case, you must prefix the property or method you're calling with a * to indicate to APL+Win your intention. This bypasses searching your hierarchy of custom objects and directly calls the **property** or **Method** in the underlying APL+Win standard object.

Example:

Copyright © 2016 Eric Lescasse

```
      'zt'⎕wi'*Click'
The onClick event fired!
```

Note that the **Menu** object standard **Click** method is indeed called and we can see that, because it does result in firing the **onClick** event which in turns branches to **line 36** in **zTemplate** and executes line **37**.

This custom object mechanism which APL2000 has built into APL+Win is very solid, is working extremely well and fast and I have developed many APL applications (some large containing hundreds of very complex Dialog Boxes and Forms of all kinds) where everything is a Custom Object using the above described technology.

There are many advantages of using custom objects written the way **zTemplate** is written:

Those objects are self-contained, i.e. are easily reusable.

All you need to do is copying the object (and the **zzInit** function) into another workspace and start using it: you don't have to copy myriads of subroutines or event handlers, which generally make things very hard to reuse

They are very maintainable.

The structure is very clear, each event, property and method being clearly isolated and encapsulated inside the object. Additionally, I pay a lot of attention in respecting the following rules:

- all my Event handlers are defined first
- then all my properties are defined
- then all my Methods are defined
- and I get sure to sort them alphabetically. Sometimes custom objects may be quite huge if they have tons of events, properties and methods and following the above rules systematically, makes them very manageable even when very large.

Documentation

Another very nice aspect of custom objects defined on the **zTemplate** model is that you can easily document all your properties, methods and events. Moreover the documentation is inserted inside the properties, methods and events themselves which makes maintenance and comprehension of the software easy.

To retrieve the documentation of a given property, method or event, all you have to do is prefix the property, method or event with a question mark:

```
      'zt'⎕wi'?property'
Comment describing the property
```

Copyright © 2016 Eric Lescasse

```
Syntax: {value←}'obj'⍁wi'property'{value}
value: describe the property value
Example:
      'myobj'⍁wi'property'1234

      'zt'⍁wi'?Method'
Comment describing the Method
Syntax: {result←}'obj'⍁wi'Method'{arg1}{arg2}...{argN}
arg1: describe the method 1st argument
arg2: describe the method 2nd argument
...
argN: describe the method Nth argument
result: describe the method result if any
Example:
      'myobj'⍁wi'Method'1234'Test'(2 2⍴⍳4)
```

It is important to include a description of the syntax, of each argument, of the results and to include one or more examples showing how to use the property, method or event you're documenting.

It takes time to write that documentation, because soon, you may end up with hundreds of objects which means thousands of properties and methods to document.

But if you make the effort of writing the documentation at the time you develop each property or method, you could give your workspace full of objects to another developer without telling him or her anything about your system and he should be able to dig into it easily all the information he needs to use it, provided you teach him the following basics:

First, run **zzInit** to initialize the custom objects system

Then, do:

```
      80 zzTELPRINT'#'⍁wi'newclasses'
zButton           zList              zUEdit             znetNumericUpDown
zCheck            zListview          zULabel            znetPath
zCombo            zManager           znetAttachment     znetPanel
zCommandBar       zMDIForm           znetButton         znetPictureBox
zCommandButton    zMedia             znetButton16       znetProcess
zDateTime         zMenu              znetButton32       znetRegex
zDialogBox        zObject            znetCheckBox       znetRegexTest
zDoc              zOption            znetCheckedListBox znetRichTextBox
zDomainUpDown     zPage              znetComboBox       znetSelectIconForm
zEdit             zPrinter           znetContextMenu    znetSharpPlot
zEditNum          zProgress          znetDateTime       znetSortedDictionary
zEditNumSpinner   zPicture           znetDictionary     znetSplitContainer
zEmployee         zRegistry          znetDirectory      znetSplitterPanel
zEnums            zRichEdit          znetDirectoryInfo  znetSmtpClient
zExcel            zScroll            znetDomainUpDown   znetTabControl
zForm             zSelector          znetEnvironment    znetTabPage
zFrame            zSimpleEdit        znetFile           znetTextBox
zGrid             zSpinner           znetFileInfo       znetTimeSpan
zHLine            zSplitter          znetFtpWebRequest  znetWebBrowser
zImagelist        zStatus            znetHashSet        znetWebClient
zInfo             zTemplate          znetLinkLabel      znetWebRequest
zLCChart          zTimer             znetMailAddress    znetXmlDocument
```

Copyright © 2016 Eric Lescasse

```
zLCChart3DUC      zToolbox              znetMailMessage     znetXmlWriter
zLCChartExamples  zTrackbar             znetMaskedTextBox
zLCChartUC        zTree                 znetMenuItem
zLabel            zUButton              znetNotifyIcon
zMDIForm          znetDictionary        znetWebClient
zMedia            znetDirectory         znetWebRequest
zMenu             znetDirectoryInfo     znetXmlDocument
```

to get a list of all the objects in your system

Then, to create an instance of an given object:

```
      'ff'⎕wi'*Create' 'zForm'
ff
```

Then, once you get an instance of the object, you can query its properties, methods and events:

```
      80 zzTELPRINT'ff'⎕wi'properties'
class  dialogcontrols  dialogresult  dialogvalues  esc  help

      80 zzTELPRINT'ff'⎕wi'*properties'
bars       events       mode         scrollmargin    value
barwrap    extent       modified     self            visible
border     font         modifystop   size            where
caption    help         name         state           ΔΔdialogcontrols
children   helpcontext   noredraw     style           ΔΔdialogresult
class      hwnd         opened       suppress        ΔΔgaps
color      icon         place        targetformats   ΔΔmargins
data       instance     pointer      theme           ΔΔscreenmargins
ddeTopic   keys         prompt       tooltipenabled  ΔΔsysmenu
def        limitwhere   properties   tooltiptime     ΔΔΔsize
edge       links        scale        tooltipwidth
enabled    methods      scrollaccel  translate

      80 zzTELPRINT'ff'⎕wi'methods'
MovePinkTip  MoveZInfo  New  Refresh

      80 zzTELPRINT'ff'⎕wi'*methods'
Close            Defer   Event  Help  Modify  Paint  Resize  SetLinks
ContextHelpMode  Delete  Exec   Hide  New     Popup  Send    Show
Create           Draw    Focus  Info  Open    Ref    Set     Wait

      80 zzTELPRINT'ff'⎕wi'events'
onClose  onFocus  onMove  onResize  onShow  onUnfocus

      80 zzTELPRINT'ff'⎕wi'*events'
Action         Delete     DropDown   KeyPress     MouseEnter  Paint    Wait
Close          Destroy    ExitError  KeyUp        MouseLeave  Reopen
ContextMenu    DragEnter  Focus      Modified     MouseMove   Resize
DdeConnect     DragLeave  Help       MouseDouble  MouseUp     Send
DdeDisconnect  DragOver   Hide       MouseDown    Move        Show
DdeExecute     Drop       KeyDown    MouseDrag    Open        Unfocus
```

Note that, when you do not prefix properties, methods or events with a *, you only get the custom properties, methods or events that you have defined inside your custom objects.

Then, to know how to use a property, method or event, get its documentation using the ? prefix:

```
      'ff'⎕wi'?margins'
Gets or sets the margins between controls and container edges
Syntax:  {margins←}'obj'⎕wi'margins'{margins}
margins: a 2-element integer vector
      [0]= the vertical margin (default: 10)
      [1]= the horizontal margin (default: 10)
```

Example:

```
      'ff'⎕wi'margins'8 12
```

Then use the property or method:

```
      'ff'⎕wi'margins'20 20
```

Creating your own objects

zObjects makes it very easy to create your own objects.

Object registration

First, you need to know that an object must be registered with the APL+Win system (i.e. be a member of the `'#'⎕wi'newclasses'` list) before it can be used as an object.

But you'll soon see that this is done automatically by the **Derive** method.

Note that objects are registered by only 2 functions in zObjects:

- **zzInit**
- **zzCustomClasses**

Registration in **zzInit** is exclusively reserved to my own **zObjects** objects (objects starting with a **z**).

Registration in **zzCustomClasses** is reserved for your own objects (objects not starting with a **z**).

The zzCustomClasses function

The **zzCustomClasses** function is created by **zzInit**, if it is not present in the workspace when you run zzInit.

If **zzCustomClasses** is in the workspace at the time you run zzInit, it is not overwritten nor modified.

Copyright © 2016 Eric Lescasse

Also, **zzCustomClasses** is not erased when you run **zzReset**.

Thus **zzCustomClasses**, which is the function where your own objects are registered with APL+Win, is never changed by zObjects.

The Derive method

You create an object using the zObject **Derive** method.

Example:

```
'zz'⎕wi'Derive' 'cMyObject' 'Menu' 'zAPLClassTemplate'
```

The Derive method requires 3 arguments:

1. The first argument is the name of the object you want to create
   It is recommended that you choose a lowercase letter prefix like **c** or **u** for all the objects you create. It is highly recommended that you NOT start your object name with a **z**.

2. The second argument concerns inheritance and is the name of an existing objects from which you want to inherit
   You should decide to inherit from **Menu** if you create a non visual object.
   You can derive from any APL+Win object or any existing **zObject** (except those starting with znet)

3. The third argument is an existing object function model
   It can be:
   zAPLClassTemplate (and must be if the second argument was **Menu**)
   zAPLControlTemplate (if you inherit from an APL+Win or zObject control object)
   zAPLFormTemplate (if you inherit from an APL+Win Form, MDIForm or from zFom or zMDIForm)
   zNetClassTemplate (if you inherit from a .Net non visual class object)
   zNetControlTemplate (if you inherit from a .Net Windows Forms control object)
   zNetFormTemplate (if you inherit from a .Net Windows Forms top level Form object)

Notes:

1. The only proper way to create your own objects is to use the **Derive** method.
2. Be sure to use **zAPLClassTemplate** if you derive from Menu
3. Do not choose any of the **zNetXXXXXTemplate** models

Example: create your first object

Let's start with a clean **zzCustomClasses** function.

Copyright © 2016 Eric Lescasse

```
    ∇ r←zzCustomClasses
[1]   r←''
    ∇
```

Create a new **uFile** object to help manipulating APL+Win component files

```
      'zz'⎕wi'Derive' 'uFile' 'Menu' 'zAPLClassTemplate'
Class <uFile> successfully created and registered!
```

The **zzCustomClasses** function has been automatically updated and now reads:

```
    ∇ r←zzCustomClasses
[1]   r←''
[2]   r,←⊂'uFile'
    ∇
```

The following **uFile** object has also been created in the workspace:

```
    ∇ a uFile b;c;d;⎕io
[1]   ⍝∇ a uFile b -- The uFile object is a non-visual class derived from a Menu
object
[2]   ⍝∇ a ←→ object name (may be omitted if ⎕wself is set)
[3]   ⍝∇ b ←→ 'property'
[4]   ⍝∇   or 'property'value
[5]   ⍝∇   or 'Method'
[6]   ⍝∇   or 'Method'argument1 ... argumentN
[7]   ⍝∇ Requires: (F) zObject
[8]   ⍝∇ zObjects v2.3
[9]   ⍝∇ Copyright(c)Lescasse Consulting 2013-2016
[10]  ⍝∇ ELE17jan16
[11]
[12]  :region-------------------------------------------Action
[13]  ⎕io←1
[14]  :if ⎕monadic ⋄ a←⎕wself ⋄ :endif
[15]  :if b≡'Action' ⋄ →('?'=↑b←↑⎕warg)ρ⍙⍙Help ⋄ :endif
[16]  →(8≠''ρ⎕idloc'⍙',b)ρ⍙⍙Inherit ⋄ →⍙'⍙',b
[17]  :endregion
[18]
[19]  :region-------------------------------------------Constructor
[20]  ⍙New:
[21]      ⍝∇ Create a new instance of uFile
[22]      ⍝∇ Example:
[23]      ⍝∇       'zobj'⎕wi'*Create' 'uFile'
[24]      ←a⎕wi'*Create' 'Menu'
[25]      ←a⎕wi'*onAction'(∈'uFile' 'zObject',¨⊂'"Action"',⎕tcnl)
[26]  ⍝    ←zzAddlink a
[27]      :return
[28]  :endregion
[29]
[30]  :region-----------------------------------------class
[31]  ⍙class:
[32]      ⎕wres←'uFile'
[33]      :return
[34]  :endregion
[35]
[36]  :region-------------------------------------------help
```

```
[37]  ∆help:
[38]      ⍝∇ ←'zobj'⎕wi'*Create' 'uFile'
[39]      ⎕wres←⎕wi'GetHelp' 'help'
[40]      :return
[41]  :endregion
[42]
[43]  :region-------------------------------------------Internal
[44]  ∆∆Help:
[45]      ⎕wres←⎕wi'GetHelp'(('?'=↑b)↓b)
[46]      :return
[47]  ∆∆Inherit:
[48]      ⎕wres←⎕wi'*'
[49]  :endregion
[50]
    ∇
```

Additionally, it has automatically been registered and can be used right away:

```
      'file'⎕wi'*Create' 'uFile'
file
      'file'⎕wi'properties'
 class help
      'file'⎕wi'class'
uFile
```

You can now start adding properties and methods to **uFile**.

Let's start by adding an **Fstie** method that opens a file or creates and opens it if it does not yet exist and returns the tie number.

Paste the following code inside **uFile**, just before :region----------Internal

```
:region-------------------------------------------Fstie
∆Fstie:
    ⍝∇ The Open method opens the specified file or creates it if it does not exist
    ⍝∇ Syntax:  tieno←'obj'⎕wi'Open'fullFileName
    ⍝∇ fullFileName:  the full path name of the file to create or open
    ⍝∇ tieno:  the file tie number
    ⍝∇ Example:
    ⍝∇       'file'⎕wi'Open' 'c:\temp\test.sf'
    ⍝∇ 1002
    f←2⊃⎕warg
    t←1+⌈/⎕xfnums,⎕fnums,0
    :try *
        f⎕xfcreate t
    :catchif 1∈⎕em⎕ss'The file exists'
        :try
            f⎕xfstie t
        :catchall
            ⎕error⎕em
        :endtry
    :catchall
        ⎕error⎕em
    :endtry
```

```
    ←⎕wi'*ΔΔtieno't
    ⎕wres←t
    :return
    :endregion
```

Press **Ctrl+E** to save the **uFile** function.

> **Note**:
> Note the `←⎕wi'*ΔΔtieno't` instruction which saves the file tie number as a user defined property in the **uFile** object

You can start using **uFile** and its **Fstie** method right away:

```
      ←'file'⎕wi'*Create' 'uFile'

      'file'⎕wi'Fstie' 'c:\temp\test.sf'
1002


      ]ties
Tie#   File Name                Size    #Comps
----   -------------------    -------    ------
1001   C:\APLWIN15.1\UCMDS    270,768       228

Tie#   Extended File Name        Size    #Comps
----   ------------------       -----    ------
1002   c:\temp\test.sf          1,056         0

      'file'⎕wi'Fstie' 'c:\temp\NonExistingFile.sf'
1003

      ]ties
Tie#   File Name                       Size    #Comps
----   -------------------           -------    ------
1001   C:\APLWIN15.1\UCMDS           270,768       228

Tie#   Extended File Name              Size    #Comps
----   -------------------------      -----    ------
1002   c:\temp\test.sf                1,056         0
1003   c:\temp\NonExistingFile.sf     1,056         0

      'file'⎕wi'Fstie' 'c:\NonExistingFolder\test.sf'
XFHOST ERROR _sopen 1010 2 3 The system cannot find the path specified.
>[file;Action] uFile"Action"
               ^
```

If yo don't remember how to use the **Fstie** method, at least remember that you can easily get documentation about it:

```
      'file'⎕wi'?Fstie'
The Open method opens the specified file or creates it if it does not exist
Syntax:  tieno←'obj'⎕wi'Fstie'fullFileName
fullFileName:  the full path name of the file to create or open
```

Copyright © 2016 Eric Lescasse

```
tieno:  the file tie number
Example:
      'file'⎕wi'Fstie' 'c:\temp\test.sf'
1002
```

It is important to note that you now have created an object called **uFile** and a useful documented **Fstie** method which you'll be able to reuse forever.

But, the interest of creating an object is to encapsulate quite a number of functionalities in the same object.

So, let's now add a **lastcomp** read-only property that returns the number of the last component existing in the file.

Try it:

Paste the following code inside **uFile**, just before :region---------- Fstie

```
:region----------------------------------------lastcomp
∆lastcomp:
    ⍝∇ Returns the number of the last component existing in the file
    ⍝∇ Syntax:  lastcomp←'obj'⎕wi'lastcomp'
    ⍝∇ lastcomp:  the number of the last component in the file
    ⍝∇ Example:
    ⍝∇       'file'⎕wi'lastcomp'
    ⍝∇ 1002
    t←⎕wi'*ΔΔtieno'                              ⍝ retrieve the tie number
    :if~t∈⎕xfnums,⎕fnums                         ⍝ ensure the file is tied
        ⎕error'FILE TIE ERROR'
    :else
        ⎕wres←¯1+2⊃⎕fsize t
    :endif
    :return
    :endregion
```

Use the new **lastcomp** property:

```
      'file'⎕wi'Fstie' 'c:\temp\test.sf'
1002

      'file'⎕wi'lastcomp'
0

      ←(⍳10)⎕fappend 1002
      ←(⍳5)⎕fappend 1002

      'file'⎕wi'lastcomp'
2
```

Copyright © 2016 Eric Lescasse

It's probably not a good idea to store the tie number in a user defined property and, in real life, I would rather have created a **LastComp** method accepting a file tie number as its argument, like this:

```
:region-----------------------------------------LastComp
∆LastComp:
    ⍝∇ Returns the number of the last component existing in the file
    ⍝∇ Syntax:  lastcomp←'obj'⎕wi'LastComp'tieno
    ⍝∇ tieno:  the file tie number
    ⍝∇ lastcomp:  the number of the last component in the file
    ⍝∇ Example:
    ⍝∇       'file'⎕wi'LastComp'1003
    ⍝∇ 2
    t←2⊃⎕warg                                    ⍝ the tie number
    :if~t∈⎕xfnums,⎕fnums                         ⍝ ensure the file is tied
        ⎕error'FILE TIE ERROR'
    :else
        ⎕wres←¯1+2⊃⎕fsize t
    :endif
    :return
    :endregion
```

However, I chose to create **lastcomp** as a property and to use a **ΔΔtieno** user defined variable to illustrate how you can save any kind of information inside your object.

Then, I suggest you imagine a couple more methods or properties that you think would be useful to add to the uFile object and that you add them yourself.

Then try using those additional properties and methods to **uFile**.

You probably start seeing the immense benefit that you can gain from creating your own objects.

If you do things right, your objects should:

1. Be stand alone functions (i.e. not use ANY global variable or subroutine, except the ones created by zzInit)
2. Encapsulate all the properties and methods that can make your developer life easier
3. Include documentation for all properties, methods or events you define in your object

Note:
It is extremely important to write stand alone objects.

This is the way to maximize the chances you'll reuse the object in other workspaces.

How would you reuse it if it called 3 subroutines which themselves called 5 subroutines

Copyright © 2016 Eric Lescasse

which themselves called 10 subroutines? You would never have the patience to find out all the dependencies and copy them to the new workspace.

On the other hand, if your object is stand alone and if you have saved it to a UCMD file where you store all your objects, all you have to do to reuse your object in another workspace is:

```
]uload uFile /f=myObjectsFile
```

And remember that you can however make use of any of the **zz** utilities created by **zzInit**, which cover a lot of common programming needs, since **zzInit** needs to be run before you can use any object and therefore this ensures those utilities will be available in the workspace.

Also note that your properties and methods can call other properties and methods in the same object. Since everything is encapsulated in the same object, you can consider your properties and methods as **APL functions localized** in your object.

Copyright © 2016 Eric Lescasse

Tutorial:
Create a real life application using zObjects

This chapter will show in details how to create an application using **zObjects**.

Introduction

The application will be relatively simple, but what's important is to understand how to proceed.

This application will display information about a bike club members bike rides.

Starting from scratch

For this new application, we will start from a clear workspace.

```
      )clear
CLEAR WS
```

First ensure that you start from a clean APL+Win state:

```
      '#'⎕wi'Reset'
```

To be able to develop a **zObjects** application we need to bring in **zzInit** and **zObject** from the **zObjects.sf** UCMD file:

```
      ]zload zzInit
1 object loaded
```

Finally, to initialize the **zObjects** system, we must run **zzInit**:

```
      zzInit
```

At this stage note that, in addition to **zzInit** and **zObject**, the workspace contains a number of utilities created by **zzInit**:

```
      80 zzTELPRINT ⎕nl 3
zAPLFormTemplate  zzClear          zzLOWERCASE  zzROWFIND    zzVersion
zForm             zzCustomClasses  zzLogError   zzReset      zzWORDREPL
zObject           zzDEB            zzMATtoSS    zzSStoMAT    zzWhere
zzANSI2AV         zzDev            zzOK         zzSize       zzWrap
zzAV2ANSI         zzError          zzOKCaption  zzTELPRINT   zzWtree
zzAddLF           zzErrorIcon      zzOVER       zzTEXTREPL   zzYesNo
zzAddLink         zzInit           zzOnNew      zzUPPERCASE  zzYesNoCancel
```

The last initial important step is to give a name to your new workspace and to save it:

```
      )wsid c:\aplwin\zobjects\bikeclub
```

Copyright © 2016 Eric Lescasse

```
WAS CLEAR WS

      )psave
C:\APLWIN\ZOBJECTS\BIKECLUB SAVED 18 January, 2016 12:32:14 PM
```

Developing the User Interface

The first step in developing our application is to develop its User Interface.

And for that, the first step is to do a careful analysis of our needs and draw sketches of our User Interface, as complete as possible, on paper.

I'll assume this has been done.

Create a new object deriving from zForm

The first step is to create the main form for our application.

For that, we will create a new object that derives from **zForm**.

> **Note:**
> This point is essential for 2 reasons:
> • First, we want to inherit from **zForm** in order to benefit from all its properties, methods, and events, for free
> • Second, as a reminder, all the objects you use in your application must be **zObjects**, i.e. be **whereIc** and **anchor** aware.

Since the form we are creating will derive from **zForm** (and therefore is a form) we will use the **zAPLFormTemplate** model, so let's bring in these 2 functions:

```
      ]zload zForm zAPLFormTemplate
2 objects loaded
```

And let's create our new **uBikeClub** object:

```
      'zz'⎕wi'Derive' 'uBikeClub' 'zForm' 'zAPLFormTemplate'
Class <uBikeClub> successfully created and registered!
```

After this step, note that the **zzCustomClasses** function reads:

```
    ∇ r←zzCustomClasses
[1]   r←''
[2]   r,←⊂'uBikeClub'
    ∇
```

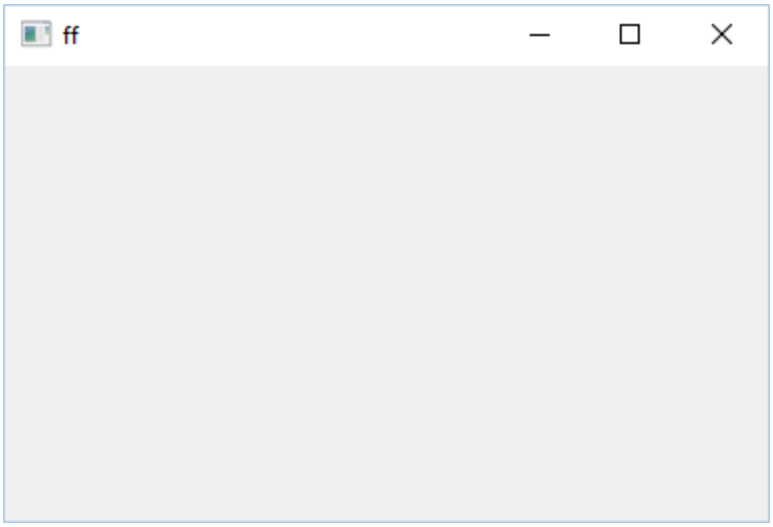
And we now have a **uBikeClub** object in our workspace reading:

```
    ∇ a uBikeClub b;c;d;⎕io
[1]   ⍝∇ a uBikeClub b -- The uBikeClub object is a Form (or MDIForm) derived from
a zForm object
[2]   ⍝∇ a ←→ object name (may be omitted if ⎕wself is set)
[3]   ⍝∇ b ←→ 'property'
[4]   ⍝∇   or 'property'value
[5]   ⍝∇   or 'Method'
[6]   ⍝∇   or 'Method'argument1 ... argumentN
[7]   ⍝∇ Requires: (F) zObject
[8]   ⍝∇ zObjects v2.3
[9]   ⍝∇ Copyright(c)Lescasse Consulting 2013-2016
[10]  ⍝∇ ELE18jan16
[11]
[12]  :region----------------------------------------------Action
[13]  ⎕io←1
[14]  :if ⎕monadic ⋄ a←⎕wself ⋄ :endif
[15]  :if b≡'Action' ⋄ →('?'=↑b←↑⎕warg)ρ⍙⍙Help ⋄ :endif
[16]  →(8≠''ρ⎕idloc'⍙',b)ρ⍙⍙Inherit ⋄ →⍎'⍙',b
[17]  :endregion
[18]
[19]  :region----------------------------------------------Constructor
[20]  ⍙New:
[21]      ⍝∇ Create a new instance of uBikeClub
[22]      ⍝∇ Example:
[23]      ⍝∇       'zt'⎕wi'*Create' 'uBikeClub'
[24]      ←a⎕wi'*Create' 'zForm'('*scale'5)
[25]      ←a⎕wi'*onAction'(∈'uBikeClub' 'zForm' 'zObject',¨⊂'"Action"',⎕tcnl)
[26]      ←zzAddLink a
[27]      :return
[28]  :endregion
[29]
[30]  :region----------------------------------------------class
[31]  ⍙class:
[32]      ⎕wres←'uBikeClub'
[33]      :return
[34]  :endregion
[35]
[36]  :region----------------------------------------------help
[37]  ⍙help:
[38]      ⍝∇ ←'zt'⎕wi'*Create' 'uBikeClub'
[39]      ⎕wres←⎕wi'GetHelp' 'help'
[40]      :return
[41]  :endregion
[42]
[43]  :region----------------------------------------------Internal
[44]  ⍙⍙Help:
[45]      ⎕wres←⎕wi'GetHelp'(('?'=↑b)↓b)
[46]      :return
[47]  ⍙⍙Inherit:
[48]      ⎕wres←⎕wi'*'
[49]  :endregion
[50]
    ∇
```

Trying our new object

At this stage, we can already try our new object, but of course it does not contain any User Interface yet, so it will display as an empty form.

```
      ←'ff'⎕wi'*Create' 'uBikeClub' 'Show'
```



Note that, since **uBikeClub** inherits from **zForm**, we may want to rather use **DemoShow**:

```
      ←'ff'⎕wi'*Create' 'uBikeClub' ('DemoShow'.4 .4 1 1)
```

This will display the **uBikeClub** at the top right of the screen as a topmost form using 40% of the screen workarea height and width.

Displaying our application form topmost at the top right of the screen is handy when we are testing it. We can see both our APL Session and the form and we can try things from the APL Session.

Save the workspace.

Creating a go function

At this stage, we will create a **go** function to avoid having to retype the previous instruction, every time we test our app.

So let's create the following function:

```
    ∇ go
[1]   ←'ff'⎕wi'*Create' 'uBikeClub' ('DemoShow'.4 .4 1)
    ∇
```

Copyright © 2016 Eric Lescasse

Test it:

```
go
```

Since a **zForm** knows how to close itself when we press **Esc**, just press **Esc** to close our **uBikeClub** form.

Save the workspace.

Start developing the User Interface

It's time to start creating our User Interface.

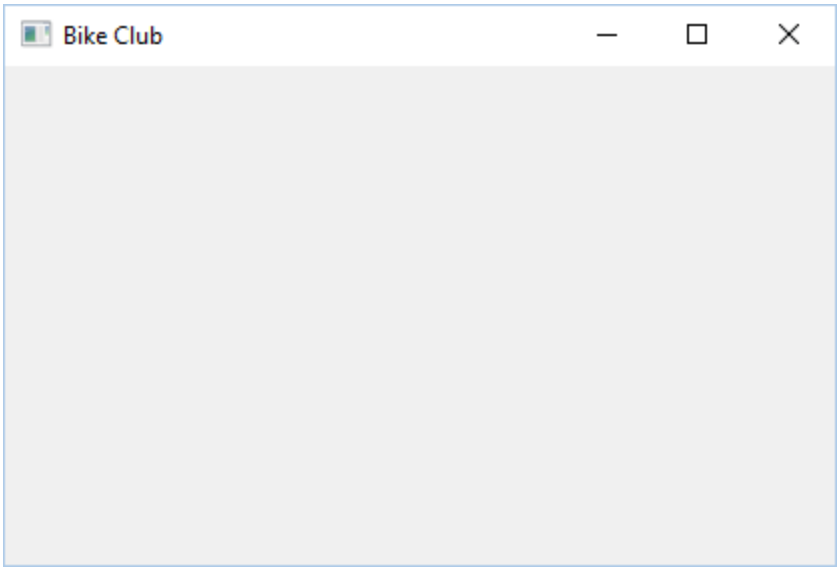
Obviously, this needs to be done in the form Constructor, i.e. in the **New** region in **uBikeClub**.

First, let's change the form caption. Change the **New** region to read:

```
:region----------------------------------------Constructor
∆New:
    ⍝∇ Create a new instance of uBikeClub
    ⍝∇ Example:
    ⍝∇       'zt'⎕wi'*Create' 'uBikeClub'
    ←a⎕wi'*Create' 'zForm'('*scale'5)('*caption' 'Bike Club')
    ←a⎕wi'*onAction'(∈'uBikeClub' 'zForm' 'zObject',¨⊂'"Action"',⎕tcnl)
    ←zzAddLink a
    :return
    :endregion
```

Let's test our change:

```
go
```



Installing a vertical zSplitter

Load the **zSpitter** object as well as **zLabel** object.

```
      ]zload zSplitter zLabel
2 object loaded
```

We will vertically separate our form in 2 parts:

- The left part will later contain a **zCombo** control and a **zGrid**
- The right part will later contain a **zLCChart** object and maybe some **znetButton32** objects

We will also keep room at the top of the form for future buttons. So we will reserve 40 pixels of vertical space for that.

Change the constructor to read:

```
:region----------------------------------------Constructor
∆New:
    ⍝∇ Create a new instance of uBikeClub
    ⍝∇ Example:
    ⍝∇       'zt'⎕wi'*Create' 'uBikeClub'
    ←a⎕wi'*Create' 'zForm'('*scale'5)('*caption' 'Bike Club')
    ←a⎕wi'*onAction'(∈'uBikeClub' 'zForm' 'zObject',¨⊂'"Action"',⎕tcnl)
    ←zzAddLink a
    ←a⎕wi'.l1.Create' 'zLabel'('*border'1)('*caption'
''')('wherelc'400'>'200)('anchor' 'ltb')
    ←a⎕wi'.l2.Create' 'zLabel'('*border'1)('*caption' '')('wherelc' '=' '>' '='
'>')('anchor' 'ltbr')
    ←a⎕wi'.sp1.Create' 'zSplitter'('Split' 'l1' 'l2')
    :return
    :endregion
```

Let's save the function and test our application:

```
      go
```

Try resizing the form.
Try using the splitter to change the relative width of the 2 labels
Here is a picture of the **uBikeClub** form with the splitter being moved to the left:

Copyright © 2016 Eric Lescasse

Populating the left zLabel

Let's create a **zCombo** control and a **zGrid** control as children of the **l1 zLabel** control.

```
      ]zload zCombo zGrid
2 objects loaded
```

Change the constructor to read:

```
:region-----------------------------------------Constructor
∆New:
    ⍝∇ Create a new instance of uBikeClub
    ⍝∇ Example:
    ⍝∇       'zt'□wi'*Create' 'uBikeClub'
    ←a□wi'*Create' 'zForm'('*scale'5)('*caption' 'Bike Club')('margins'5 5)
('gaps'5 5)
    ←a□wi'*onAction'(∈'uBikeClub' 'zForm' 'zObject',¨⊂'"Action"',□tcnl)
    ←zzAddLink a
    ←a□wi'*.l1.Create' 'zLabel'('*border'1)('*caption' '')
('wherelc'400'>'200)('anchor' 'ltb')
    ←a□wi'*.l2.Create' 'zLabel'('*border'1)('*caption' '')('wherelc' '=' '>' '='
'>') ('anchor' 'ltbr')
    ←a□wi'*.sp1.Create' 'zSplitter'('Split' 'l1' 'l2')
    ←a□wi'*.l1.cbMembers.Create' 'zCombo'('wherelc'050'>')('anchor' 'ltr')
('caption' 'Member')
    ←a□wi'*.l1.grid.Create' 'zGrid'('*border'1)('wherelc' '>'0'>>' '>>')
('anchor' 'ltrb')
    :return
    :endregion
```

Note:
You may notice that I am not using spaces to separate things in my code when it is not necessary. The advantage of doing that is that you spare a lot of horizontal room in your code and room in the workspace as well.

So I am writing:

… 'zCombo'('wherelc'ϴ50ϴ'>')('anchor' 'ltr')('caption' 'Member')

rather than:

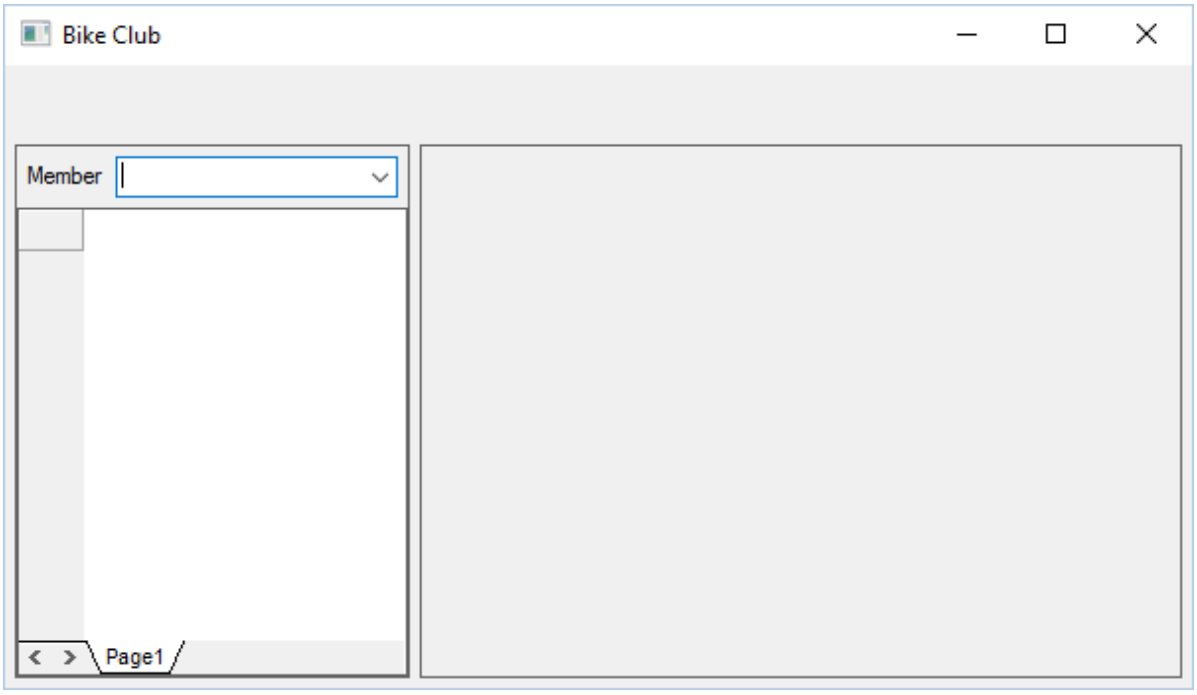
… 'zCombo' ('wherelc' ϴ 50 ϴ '>') ('anchor' 'ltr') ('caption' 'Member')

If you want to adopt this habit, the rule is very simple:

You do not need a space to separate a symbol and text or numbers.

Thus: ϴ 50 is the same as: ϴ50 and 'wherelc' ϴ is the same as 'wherelc'ϴ because ' is a symbol.

The user interface now looks like this:



You may notice that, because we are using a border around the **zGrid** object, the left, bottom and right **zGrid** borders look thick because they display just against the **zLabel** border.

Copyright © 2016 Eric Lescasse

We can solve this problem by **wherelc** adjustments.

Change the **zGrid wherelc** statement from:

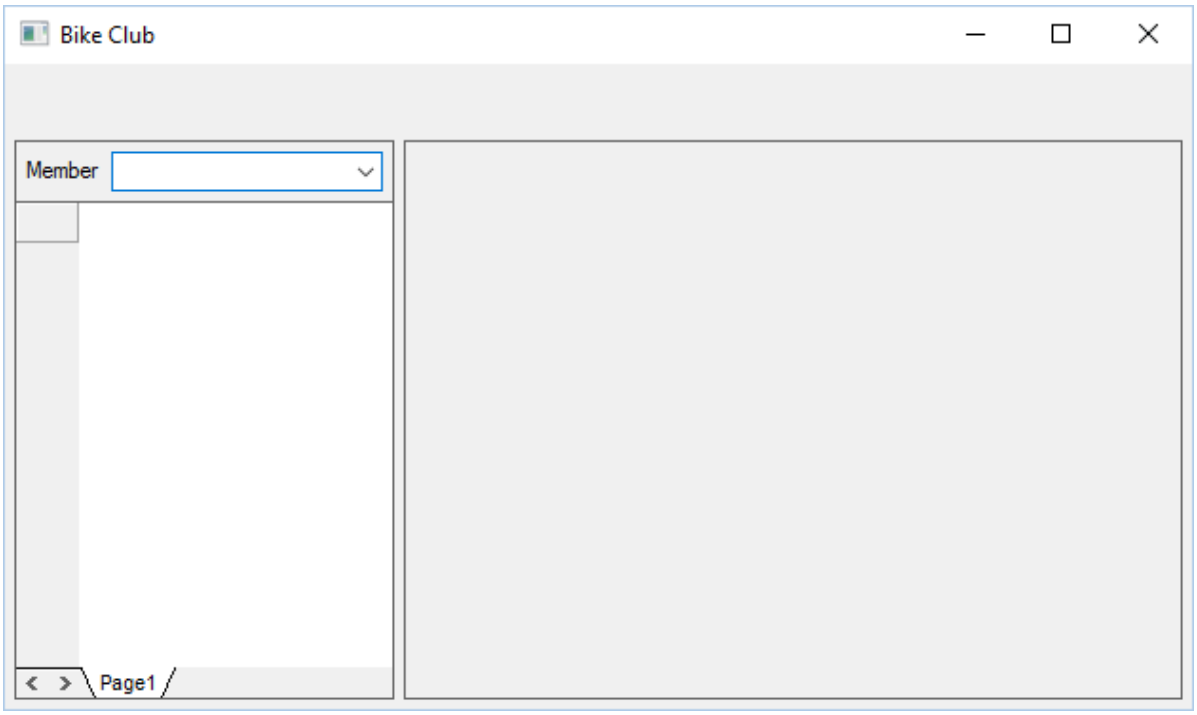
```
('wherelc' '>'0'>>' '>>')
```

to:

```
('wherelc' '>'¯1'>>' '>>'0 0 2 2)
```

This forces the **zGrid** to start at a **¯1** horizontal position, thus making the **zGrid** left border coincide with the **zLabel** border and forces the **zGrid** to have an additional **2** pixels for its height and width thus making its bottom and right borders also coincide with the **zLabel** bottom and right borders.

The user interface becomes:



Try resizing the form or moving the splitter. We have a quasi perfect user interface so far.

Adding a status bar

Bring in the **zStatus** and **zTimer** object:

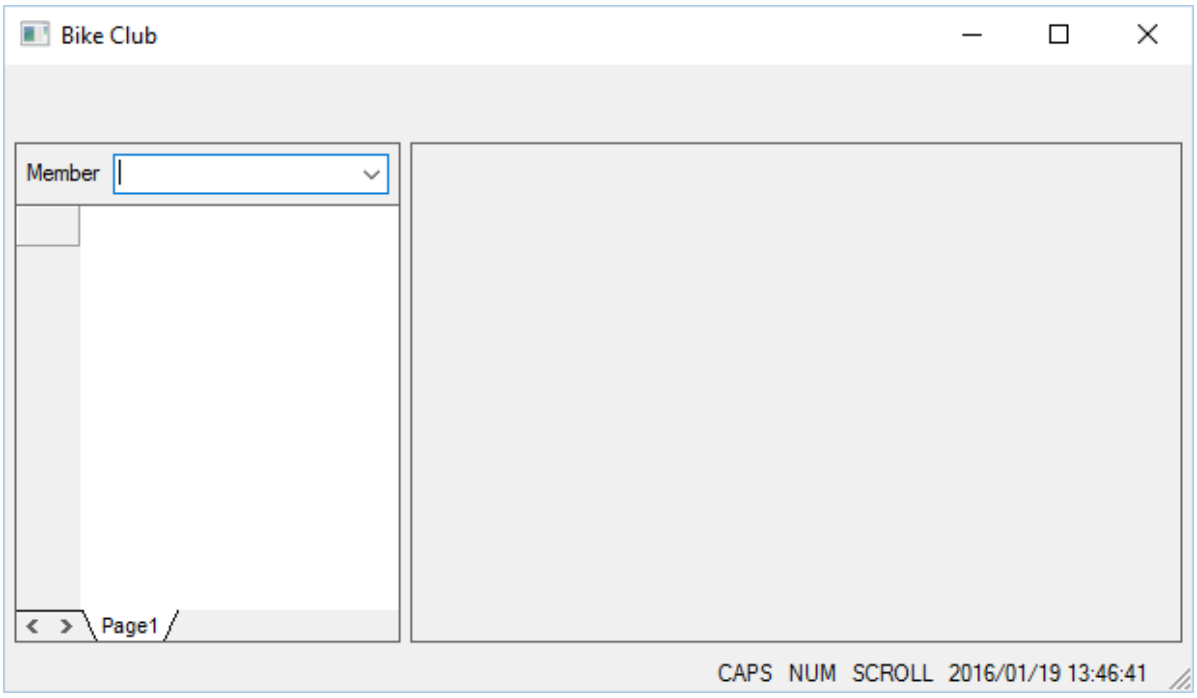
```
      ]zload zStatus zTimer
2 objects loaded
```

Then add the following line at the bottom of the constructor, just before **:return**:

```
      ←a⎕wi'*.st.Create' 'zStatus'('panes'1 2 3'caps' 'num' 'scroll' '⎕ts'¯12)
```

This creates a status bar called **st** with 3 custom fields, 3 fields reserved to display the **NumLock**, **CapsLock** and **ScrollLock** keyboard states, 1 field to display the current date and time and an overlay field.

Launch the app. Assuming the the **NumLock**, **CapsLock** and **ScrollLock** have been pressed, it should look like this:



Adding a zLCChart object

We will now add a **zLCChart** object as a child of the second label, i.e. to the right of the splitter.

Bring in the **zLCChart** object:

```
      ]zload zLCChart
1 object loaded
```

Add the following code to the bottom of the constructor, just before the **:return** statement:

```
      ←a⎕wi'*.l2.lcc.Create' 'zLCChart'('whereIc'32 0'>>' '>>')('anchor' 'lrtb')
```

Try to run the application:

```
      go
VALUE ERROR
>[ff.l2.lcc;Action] zEnums"Action"
                    ^
```

Copyright © 2016 Eric Lescasse

We get an error. That's because the **zLCChart** object requires the **zEnums** object. We can easily solve the problem. Bring in the **zEnums** object and continue execution:

```
      ]zload zEnums
1 object loaded

      →⎕lc
```

Notice that the **zLCChart** object is invisible for now: the reason is that its background color is white by default, that it does not have a border by default and that it does not display any chart by default either.

However, the **zLCChart** object has a **ShowDefaultChart** method that we can use just to make it display a dummy chart.

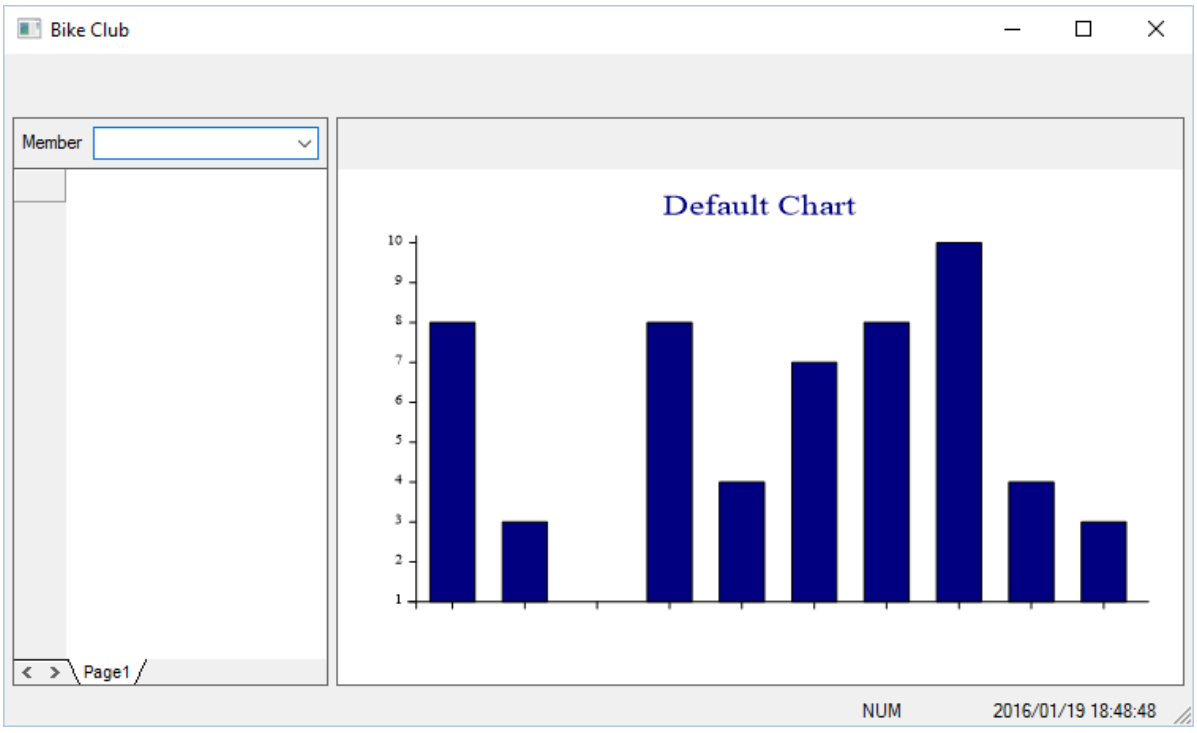
So, change the line creating the **zLCChart** instance in the constructor from:

```
    ←a⎕wi'*.12.1cc.Create' 'zLCChart'('wherelc'32 0'>>' '>>')('anchor' 'lrtb')
```

to:

```
    ←a⎕wi'*.12.1cc.Create' 'zLCChart'('wherelc'32 0'>>' '>>')('anchor' 'lrtb') '
            ShowDefaultChart'
```

Start the application: it now displays something like this:



Our application will need some toolbar buttons that will be displayed above the current bordered labels and also some buttons that will be displayed above the chart.

Copyright © 2016 Eric Lescasse

However, at this stage, we don't know yet exactly what buttons we will need, so let's keep that for later.

Our application User Interface is almost done.

Our application constructor should look like this:

```
:region----------------------------------------------Constructor
∆New:
    ⍝∇ Create a new instance of uBikeClub
    ⍝∇ Example:
    ⍝∇       'zt'⎕wi'*Create' 'uBikeClub'
    ←a⎕wi'*Create' 'zForm'('*scale'5)('*caption' 'Bike Club')('margins'5 5)
             ('gaps'5 5)
    ←a⎕wi'*onAction'(∈'uBikeClub' 'zForm' 'zObject',¨⊂'"Action"',⎕tcnl)
    ←zzAddLink a
    ←a⎕wi'*.l1.Create' 'zLabel'('*border'1)('*caption' '')('wherelc'400'>'200)
             ('anchor' 'ltb')
    ←a⎕wi'*.l2.Create' 'zLabel'('*border'1)('*caption' '')('wherelc' '=' '>' '='
             '>') ('anchor' 'ltbr')
    ←a⎕wi'*.sp1.Create' 'zSplitter'('Split' 'l1' 'l2')
    ←a⎕wi'*.l1.cbMembers.Create' 'zCombo'('wherelc'0500'>')('anchor' 'ltr')
             ('caption' 'Member')
    ←a⎕wi'*.l1.grid.Create' 'zGrid'('*border'1)('wherelc' '>'¯1'>>' '>>'0 0 2 2)
             ('anchor' 'ltrb')
    ←a⎕wi'*.st.Create' 'zStatus'('panes'1 2 3'caps' 'num' 'scroll' '⎕ts'¯12)
    ←a⎕wi'*.l2.lcc.Create' 'zLCChart'('wherelc'32 0'>>' '>>')('anchor' 'lrtb')
             'ShowDefaultChart'
    :return
    :endregion
```

We now need to think about displaying data.

Displaying data in our application

The next step is to display data in our application.

Separation of concerns

We should observe good programming practices and therefore apply the **separation of concerns** principle everywhere in our application.

According to this principle, we should not mix code that

- deals with the UI
- deals with the data
- deals with the computation and business rules

First, we will separate the event handler definitions from the UI building code by moving all event handler definitions to the bottom of the constructor.

Second, we will handle the **onShow** event on the form and will use this event to populate the form with data.

However, to observe our above mentioned rules we will not write raw APL code that deals with data directrly inside the **onShow** event handler.

Creating an onShow event handler

Add the following code at the bottom of the constructor, just before the **:return** statement:

```
⍝ Events
←a⎕wi'AddHandler' '*onShow' 'uBikeClub"onShow"'
```

Note that the event handler is defined as: **'uBikeClub"onShow"'** , so the event handler will be encapsulated in the **uBikeClub** object itself (to keep it self contained).

Note: Multicast Events

Note the use of the **AddHandler** method here.

Whenever we inherit from a **zObject** (remember **uBikeClub** is inheriting from **zForm**), we should use **AddHandler** to define our event handlers.

So, instead of using:

```
←a⎕wi'*onShow' 'uBikeClub"onShow"'
```

use:

```
←a⎕wi'AddHandler' '*onShow' 'uBikeClub"onShow"'
```

Copyright © 2016 Eric Lescasse

This is important because the object we are inheriting from (here zForm) may already have an **onShow** event defined internally and we don't want to override this event.

Instead we want to add more **onShow** behavior to whatever **onShow** behavior may already be defined in the object we inherit from.

This is called **multicast events** in other languages like C#. **zObjects** fully support multicasts events.

The **AddHandler** method supports an optional additional argument which is a **boolean scalar** value. By default it is **1** and means that the event we define using **AddHandler** fires **after** the event already defined in the underlying object.

Use **0** to force the event handler to fire **before** the one (if any) defined in the underlying object we are inheriting from.

Finally, note that we can define as many event handlers for a given event, using **AddHandler**, as you want.

Now paste the following event handler code at the bottom of the **uBikeClub** object:

```
:region----------------------------------------onShow
∆onShow:
    ←⍬wi'OpenDatabase'
    ←⍬wi'PopulateMembersCombo'
    :return
    :endregion
```

As we mentioned, in order to properly separate concerns, we must avoid by all means writing raw APL code coping with data in an **onShow** event handler. **onShow** concerns the User Interface and data concerns data!

Note:
Why is it so important to separate concerns?

In this application we will dynamically create the data the application uses. Imagine we later want to instead use an **SQL Server** database to hold the data. By using separate methods like **CreateData** or **ReadData** to cope with data, etc. we protect the **onShow** event from having to be rewritten or even changed at all.

So, the idea is that, if the application is updated to now use another data repository like an **SQL Server** database, the only places that will need to be updated in the application will be

the methods in the data layer. None of the properties, methods or event handlers coping with the User Interface should need to be changed at all.

This may seem a bit artificial in a simple application as the one we are writing, but becomes all the more important that our application gets bigger.

Finally let's add an **OpenDatabase** method and a **PopulateMembersCombo** method.

Paste the following code, just above the **onShow** event handler:

```
:region----------------------------------------OpenDatabase
∆OpenDatabase:
    ⍝∇ Opens the database or creates it if it does not exist
    ⍝∇ Syntax:  'obj'⎕wi'OpenDatabase'
    :return
    :endregion

:region----------------------------------------PopulateMembersCombo
∆PopulateMembersCombo:
    ⍝⍝∇ Populates the Members combo box
    ⍝⍝∇ Syntax:  'obj'⎕wi'PopulateMembersCombo'
    :return
    :endregion
```

Note that we don't need to write the code for these methods right now, but it's a good habit to already document them as we have done above.

All this is clean and we still should be able to run the application with no problem. Give it a try.

Developing the database

Let's develop the **OpenDatabase** method.

We want to create a very simple database containing random data and the following pieces of information:

1. Member Name (string)
2. Date (yyyymmdd integer)
3. Bike Ride Length (in miles)
4. Bike Ride Time (hhmmss integer)
5. Bike Ride Average Speed (in miles/hour)

We also want this information to be saved in an APL+Win component file called **BikeClub.sf** which will automatically be created, if it does not exist, in the application folder.

This database structure will be:

Component 1: a nested vector of Member names

Copyright © 2016 Eric Lescasse

Component 2-10: (reserved)
Component 11+: one Nx4 nested array per component

So the first Member will have its Nx4 data matrix stored in component 11, the second Member will have its Nx4 data matrix saved in component 12, etc.

Change the **OpenDatabase** method to read:

```
:region-----------------------------------------OpenDatabase
∆OpenDatabase:
    ⍝∇ Opens the database or creates it if it does not exist
    ⍝∇ Syntax:  'obj'⎕wi'OpenDatabase'
    f←⎕wsid
    :try
        f⎕fstie t←1+⌈/⎕xfnums,⎕fnums,0
    :catchall
        c←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        m←↑⍵n←'Anne' 'Eric' 'Frederic' 'John' 'Robert'
        m←↑⍵n←↑,/(⊂¨n)∘.,¨(?mρ4)↑¨⎕split c[m 4ρ26?26] ⍝ names
        p←50+?mρ100                                  ⍝ nb of rides per member
        d←⎕wi'DateRep'((⎕wi'DateBase'20150101)+0,⍳364)  ⍝ all days in 2015
        e←(⊂¨p?¨365)⌷¨⊂d                             ⍝ nested vec of ride dates (all members)
        g←(∈p↑¨1)⎕penclose.1×100+?(+/p)ρ900          ⍝ nested vec of distances in miles
        s←(∈p↑¨1)⎕penclose.1×150+?(+/p)ρ120          ⍝ nested vec of plausible speeds in m/h
        u←100⊥¨(⊂24 60 60)⊤¨⌊3600×g÷s                ⍝ nested vec of ride times (hhmmss)
        m←⊃¨e,¨¨g,¨¨u,¨¨s                            ⍝ nested vec of matrices
        f⎕fcreate t←1+⌈/⎕xfnums,⎕fnums,0              ⍝ create the database
        ⍵n⎕fappend t                                 ⍝ save names in component 1
        ⍵(⊂'')⎕fappend¨9ρt                           ⍝ init comps 2-10
        ⍵m⎕fappend¨t                                 ⍝ save all member matrices
    :endtry
    ⍵←⎕wi'*ΔΔtieno't                                 ⍝ save tie number
    :return
    :endregion
```

Here one can really see the power of the APL language in action.

We are generating a random number of member names in the **n** variable.

We are then generating a random number of rides per member in the **p** variable.

Then we are using the **DateBase** and **DateRep** methods (inherited from **zObject**) to generate all 2015 day dates in the **d** variable.

The next line randomly selects 2015 dates for the correct number of rides for each member, in the **e** variable.

We then generate random bike ride distances ranging from 10 miles to 100 miles for each member in the **g** variable.

We do the same thing with speeds ranging from 15 m/h to 27 m/h.

Copyright © 2016 Eric Lescasse

This allows us to calculate the plausible bike ride times in variable **u**.

We finally create a nested vector of matrices in variable **m**.

We now have all the data ready to be saved to the database.

The next 4 instructions:

1. create the APL component file database (**BikeClub.sf**) in the application folder,
2. save the names as a nested vector in the first component,
3. create 9 empty components
4. save all the member matrices in the subsequent components

Finally note the last instruction:

```
←⎕wi'*ΔΔtieno't                    ⍝ save tie number
```

which allows us to save the database tie number in a User Defined variable as we will need it within other methods in this application.

Note: User Defined Properties vs Global Variables

It is extremely important to use **User Defined properties** to save information that you need to persist during the application life time.

DO NOT use **global variables** for that purpose.

Global variables are the worst thing that can be done to an APL application (and as a matter of fact to any application). When applications become large, global variables make applications prone to bugs and errors and moreover make applications very much harder to maintain.

NEVER use any global variable in an APL application if you can avoid it.

So, now, the first time you run the application, the database will be sliently and automatically created.

The next time you run the application, it will simply be open.

Populating the Members combo box

We are now ready to populate the Members combo box with member names.

So change the **PopulateMembersCombo** method to read:

```
:region-----------------------------------------PopulateMembersCombo
∆PopulateMembersCombo:
    ⍝∇ Populates the Members combo box
```

```
    ⍝∇ Syntax:  'obj'⎕wi'PopulateMembersCombo'
    t←⎕wi'*ΔΔtieno'                              ⍝ retrieve the tie number
    c←⎕fread t 1
    c←c[⎕av⍋⊃c]
    ←'cbMembers'⎕wi'list'c
    :return
    :endregion
```

Simple isn't it?

Yes, but this is not right. We said we must not mix up UI code and data related code!

The first 3 instructions are data related code.

The last one is UI code.

So, we'd better write things like that:

```
:region-------------------------------------------GetMemberNames
ΔGetMemberNames:
    ⍝∇ Retrieves a sorted list of member names
    ⍝∇ Syntax:  names←'obj'⎕wi'GetMemberNames'
    ⍝∇ names: sorted nested vector of all member names
    t←⎕wi'*ΔΔtieno'                              ⍝ retrieve the tie number
    c←⎕fread t 1
    ⎕wres←c[⎕av⍋⊃c]
    :return
    :endregion

:region-------------------------------------------PopulateMembersCombo
ΔPopulateMembersCombo:
    ⍝∇ Populates the Members combo box
    ⍝∇ Syntax:  'obj'⎕wi'PopulateMembersCombo'
    c←⎕wi'GetMemberNames'
    ←'cbMembers'⎕wi'list'c
    :return
    :endregion
```

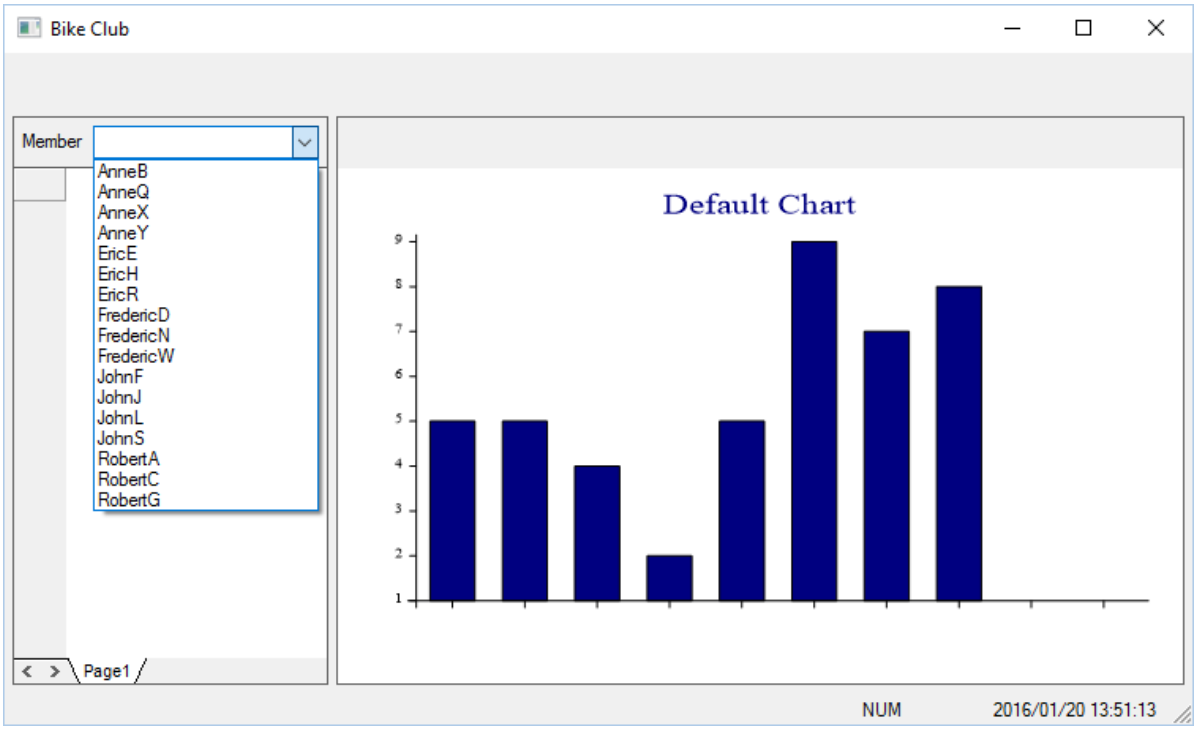
Paste the **GetMemberNames** method just above the **OpenDatabase** method to keep our methods in alphabetical order within the **uBikeName** object.

Not only splitting the **PopulateMembersCombo** into the above 2 methods allows us to respect the separation of concerns principle, but it also allows us to have a new method called **GetMemberNames** that we may be able to reuse later in the application.

Note that I am using the **zCombo list** property (not ***list**): this is important for the **zCombo** to support some additional functionalities that we'll use later on.

Try the application and open the combo box:

Copyright © 2016 Eric Lescasse



Great.

Copyright © 2016 Eric Lescasse

At this stage, the **uBikeClub** object should look like this:

```
    ∇ a uBikeClub b;c;d;e;f;g;m;n;p;s;t;u;⎕io
[1]   ⍝∇ a uBikeClub b -- The uBikeClub object is a Form (or MDIForm) derived from a
       zForm object
[2]   ⍝∇ a ←→ object name (may be omitted if ⎕wself is set)
[3]   ⍝∇ b ←→ 'property'
[4]   ⍝∇   or 'property'value
[5]   ⍝∇   or 'Method'
[6]   ⍝∇   or 'Method'argument1 ... argumentN
[7]   ⍝∇ Requires: (F) zObject
[8]   ⍝∇ zObjects v2.3
[9]   ⍝∇ Copyright(c)Lescasse Consulting 2013-2016
[10]  ⍝∇ ELE18jan16
[11]
[12]  :region-----------------------------------------Action
[13]  ⎕io←1
[14]  :if ⎕monadic ⋄ a←⎕wself ⋄ :endif
[15]  :if b≡'Action' ⋄ →('?'=↑b←↑⎕warg)ρ⍙⍙Help ⋄ :endif
[16]  →(8≠''ρ⎕idloc'⍙',b)ρ⍙⍙Inherit ⋄ →⍙'⍙',b
[17]  :endregion
[18]
[19]  :region-------------------------------------Constructor
[20]  ⍙New:
[21]      ⍝∇ Create a new instance of uBikeClub
[22]      ⍝∇ Example:
[23]      ⍝∇       'zt'⎕wi'*Create' 'uBikeClub'
[24]      ←a⎕wi'*Create' 'zForm'('*scale'5)('*caption' 'Bike Club')('margins'5 5)
            ('gaps'5 5)
[25]      ←a⎕wi'*onAction'(∈'uBikeClub' 'zForm' 'zObject',¨⊂'"Action"',⎕tcnl)
[26]      ←zzAddLink a
[27]      ←a⎕wi'*.l1.Create' 'zLabel'('*border'1)('*caption' '')('wherelc'40⌀'>'200)
            ('anchor' 'ltb')
[28]      ←a⎕wi'*.l2.Create' 'zLabel'('*border'1)('*caption' '')
            ('wherelc' '=' '>' '=' '>')('anchor' 'ltbr')
[29]      ←a⎕wi'*.sp1.Create' 'zSplitter'('Split' 'l1' 'l2')
[30]      ←a⎕wi'*.l1.cbMembers.Create' 'zCombo'('wherelc'⌀50⌀'>')('anchor' 'ltr')
            ('caption' 'Member')
[31]      ←a⎕wi'*.l1.grid.Create' 'zGrid'('*border'1)('wherelc' '>'¯1'>>' '>>'0 0 2 2)
            ('anchor' 'ltrb')
[32]      ←a⎕wi'*.st.Create' 'zStatus'('panes'1 2 3'caps' 'num' 'scroll' '⎕ts'¯12)
[33]      ←a⎕wi'*.l2.lcc.Create' 'zLCChart'('wherelc'32 0'>>' '>>')('anchor' 'lrtb')
            'ShowDefaultChart'
[34]      ⍝ Events
[35]      ←a⎕wi'AddHandler' '*onShow' 'uBikeClub"onShow"'
[36]      :return
[37]      :endregion
[38]
[39]  :region-------------------------------------------class
[40]  ⍙class:
[41]      ⎕wres←'uBikeClub'
[42]      :return
[43]      :endregion
[44]
[45]  :region--------------------------------------------help
[46]  ⍙help:
[47]      ⍝∇ ←'zt'⎕wi'*Create' 'uBikeClub'
[48]      ⎕wres←⎕wi'GetHelp' 'help'
```

```
[49]      :return
[50]      :endregion
[51]
[52]  :region----------------------------------------GetMemberNames
[53]  ∆GetMemberNames:
[54]      ⍝∇ Retrieves a sorted list of member names
[55]      ⍝∇ Syntax:  names←'obj'⎕wi'GetMemberNames'
[56]      ⍝∇ names: sorted nested vector of all member names
[57]      t←⎕wi'*ΔΔtieno'                              ⍝ retrieve the tie number
[58]      c←⎕fread t 1
[59]      ⎕wres←c[⎕av⍋⊃c]
[60]      :return
[61]      :endregion
[62]
[63]
[64]  :region----------------------------------------OpenDatabase
[65]  ∆OpenDatabase:
[66]      ⍝∇ Opens the database or creates it if it does not exist
[67]      ⍝∇ Syntax:  'obj'⎕wi'OpenDatabase'
[68]      f←⎕wsid
[69]      :try
[70]          f⎕fstie t←1+⌈/⎕xfnums,⎕fnums,0
[71]      :catchall
[72]          c←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
[73]          m←↑⍬n←'Anne' 'Eric' 'Frederic' 'John' 'Robert'
[74]          m←↑⍬n←↑,/(⊂¨n)∘.,¨(?m⍴4)↑¨⎕split c[m 4⍴26?26] ⍝ names
[75]          p←50+?m⍴100                              ⍝ nb of rides per member
[76]          d←⎕wi'DateRep'((⎕wi'DateBase'20150101)+0,⍳364)  ⍝ all days in 2015
[77]          e←(⊂¨p?¨365)[]¨⊂d                        ⍝ nested vec of ride dates (all members)
[78]          g←(∈p↑¨1)⎕penclose.1×100+?(+/p)⍴900      ⍝ nested vec of distances in miles
[79]          s←(∈p↑¨1)⎕penclose.1×150+?(+/p)⍴120      ⍝ nested vec of plausible speeds in m/hr
[80]          u←100⊥¨(⊂24 60 60)⊤¨⌊3600×g÷s            ⍝ nested vec of ride times (hhmmss)
[81]          m←⊃¨e,⍤⍤1 g,⍤⍤1 u                        ⍝ nested vec of matrices
[82]          f⎕fcreate t←1+⌈/⎕xfnums,⎕fnums,0          ⍝ create the database
[83]          ←n⎕fappend t                             ⍝ save names in component 1
[84]          ←(⊂'')⎕fappend¨9⍴t                       ⍝ init comps 2-10
[85]          ←m⎕fappend¨t                             ⍝ save all member matrices
[86]      :endtry
[87]      ←⎕wi'*ΔΔtieno't                              ⍝ save tie number
[88]      :return
[89]      :endregion
[90]
[91]  :region----------------------------------------PopulateMembersCombo
[92]  ∆PopulateMembersCombo:
[93]      ⍝∇ Populates the Members combo box
[94]      ⍝∇ Syntax:  'obj'⎕wi'PopulateMembersCombo'
[95]      c←⎕wi'GetMemberNames'
[96]      ←'cbMembers'⎕wi'list'c
[97]      :return
[98]      :endregion
[99]
[100] :region----------------------------------------onShow
[101] ∆onShow:
[102]     ←⎕wi'OpenDatabase'
[103]     ←⎕wi'PopulateMembersCombo'
[104]     :return
[105]     :endregion
[106]
```

Copyright © 2016 Eric Lescasse

```
[107] :region----------------------------------------------Internal
[108] ∆∆Help:
[109]     ⎕wres←⎕wi'GetHelp'(('?'=↑b)↓b)
[110]     :return
[111] ∆∆Inherit:
[112]     ⎕wres←⎕wi'*'
[113]     :endregion
[114]
    ∇
```

Note that we have achieved all this in a self contained function of only 114 lines so far.

Populating the grid

The next step is to populate the grid with member data when the user selects a name in the combo box.

Add the following handler at the bottom of the constructor, just before the **:return** statement:

```
    ←'cbMembers'⎕wi'AddHandler' '*onClick' 'uBikeClub"cbMembers_onClick"'
```

and paste the following event handler just after the **onShow** event handler in **uBikeClub**:

```
:region----------------------------------------cbMembers_onClick
∆cbMembers_onClick:
    m←zzDEB⎕wi'*text'
    d←⎕wi'GetMemberData'm
    ←⎕wi'PopulateGrid'm
    :return
    :endregion
```

Paste the following new **GetMemberData** method in **uBikeClub**, just before the **GetMemberNames** method so that methods stay in alphabetical order in **uBikeClub**:

```
:region----------------------------------------GetMemberData
∆GetMemberData:
    ⍝∇ Retrieves the bike rides data for the specified member
    ⍝∇ Syntax:  data←'obj'⎕wi'GetMemberData'member
    ⍝∇ member: the name of a bike club member
    ⍝∇ data:  the Nx4 bike rides data matrix for this member
    ⍝∇        [;1]= dates in number of days since 19000101
    ⍝∇        [;2]= bike ride distance (in miles)
    ⍝∇        [;3]= bike ride time in % of 24 hours
    ⍝∇        [;4]= bike ride average speed (in miles/hour)
    ⍝∇ Example:
    ⍝∇       'bk'⎕wi'GetMemberData' 'EricE'
    ⍝∇ 42008 90.1 0.2062615741  18.2
    ⍝∇ 42010 32.3 0.07962962963 16.9
    ⍝∇ 42011 29   0.0491087963  24.6
    ⍝∇ ...    t←⎕wi'*∆∆tieno'                    ⍝ retrieve the tie number
    c←⎕fread t 1
    :if(↑⍴c)<e←⊂⎕warg[2]
```

```
        ⎕wres←0 4⍴''
    :else
        m←24 60 60
        d←⎕fread t(10+e)
        d←d[⍋d[;1];]                    ⍝ sort matrix by increasing dates
        d[;1]←2+⎕wi'DateBase'(d[;1])    ⍝ convert yyyymmdd to grid dates
        d[;3]←(m⊥100 100 100⍉d[;3])÷×/m ⍝ convert hhmmss times to grid times
        ⎕wres←d
    :endif
    :return
    :endregion
```

The **GetMemberData** methods not only retrieves the proper member data from the database but also converts the dates (column 1) and times (column 3) to values that the APL+Win grid can understand as date and times.

Now paste the following **PopulateGrid** method in **uBikeClub**:

```
:region-----------------------------------------PopulateGrid
∆PopulateGrid:
    ⍝∇ Populates the Grid
    ⍝∇ Syntax:  'obj'⎕wi'PopulateGrid'matrix
    ⍝∇ matrix: the Nx4 member bike rides matrix
    (r c)←⍴m←2⊃⎕warg
    p←'Date' 'Miles' 'Time' 'Speed'
    f←'dd-MMM-yy' '#.#' 'H:mm:ss' '#.#'
    ←'grid'⎕wi'*xRows'0                          ⍝ first empty the grid
    ←'grid'⎕wi'*XRedraw'
    ←'grid'⎕wi'*xHeadCols'0
    ←'grid'⎕wi'*xRows'r
    ←'grid'⎕wi'*xCols'c
    ←'grid'⎕wi'*xText'¯1(⍳c)p
    ←'grid'⎕wi'*xValue'(⍳r)(2 4)(m[;2 4])
    ←'grid'⎕wi'*xDate'(⍳r)(1 3)(m[;1 3])
    ←'grid'⎕wi'*xFormat'(⍳r)(⍳c)(r c⍴f)
    ←'grid'⎕wi'*xRowSize'(⍳r)17
    ←'grid'⎕wi'*xFitCol'(⍳c)
    :return
    :endregion
```

Note:

When developing objects, it is recommended that we keep our methods as simple and as short as possible, for the task they have to perform.

Also, each method should have **one purpose** and only one.

For example, we also want to update the status bar displaying the following information:

- The total number of rides ridden by the member during the year
- The total number of miles he rode
- The average speed he could achieve

Copyright © 2016 Eric Lescasse

We could easily add instructions that update the status bar in the **PopulateGrid** method, but that would break the "one purpose" principle. PopulateGrid should only deal with the Grid, not with the Status Bar!

We will write an **UpdateStatus** method instead.

Updating the Status Bar

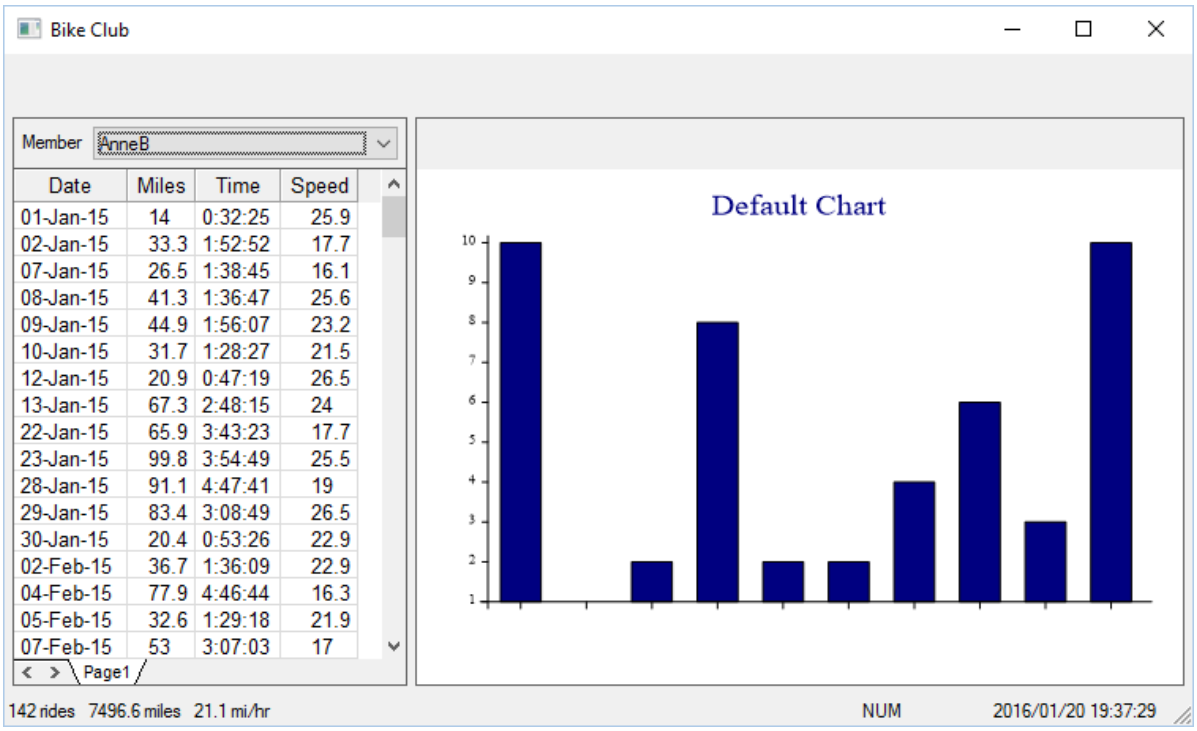
Let's add a call to an **UpdateStatus** method in the **cbMembers_onClick** event handler:

```
:region-----------------------------------------cbMembers_onClick
∆cbMembers_onClick:
    m←zzDEB⎕wi'*text'
    d←⎕wi':GetMemberData'm
    ←⎕wi':PopulateGrid'd
    ←⎕wi':UpdateStatus'd
    :return
    :endregion
```

Now paste the following **UpdateStatus** method in the **uBikeClub** object:

```
:region-----------------------------------------UpdateStatus
∆UpdateStatus:
    ⍝∇ Updates the status bar with bike ride statistics
    ⍝∇ Syntax:  'obj'⎕wi'UpdateStatus'matrix
    ⍝∇ matrix: the Nx4 member bike rides matrix
    m←2⊃⎕warg
    n←↑⍴m                                    ⍝ total number of rides
    p←+/m[;2]                                ⍝ total number of miles
    s←p÷⍨+/×/m[;2 4]                         ⍝ annual average speed
    s←.1×⌊.5+10×s                            ⍝ round avg speed to 1 decimal
    ←'st'⎕wi'*SetStatus'1((⍕n),' rides')
    ←'st'⎕wi'*SetStatus'2((⍕p),' miles')
    ←'st'⎕wi'*SetStatus'3((⍕s),' mi/hr')
    :return
    :endregion
```

Let's launch the application and select a member in the combo box. We should see something like:



Our application starts looking good!

Hooking up the chart to the grid

OK: it's now time to get rid of our default chart and to replace it by a chart hooked up to the grid data.

First, add a call to a **DrawMilesPerMonthSimpleBarChart** method in **cbMembers_onClick** event handler so that it becomes:

```
:region-----------------------------------------------cbMembers_onClick
∆cbMembers_onClick:
    m←zzDEB□wi'*text'
    d←□wi':GetMemberData'm
    ←□wi':PopulateGrid'd
    ←□wi':UpdateStatus'd
    ←□wi':DrawMilesPerMonthSimpleBarChart'd
    :return
    :endregion
```

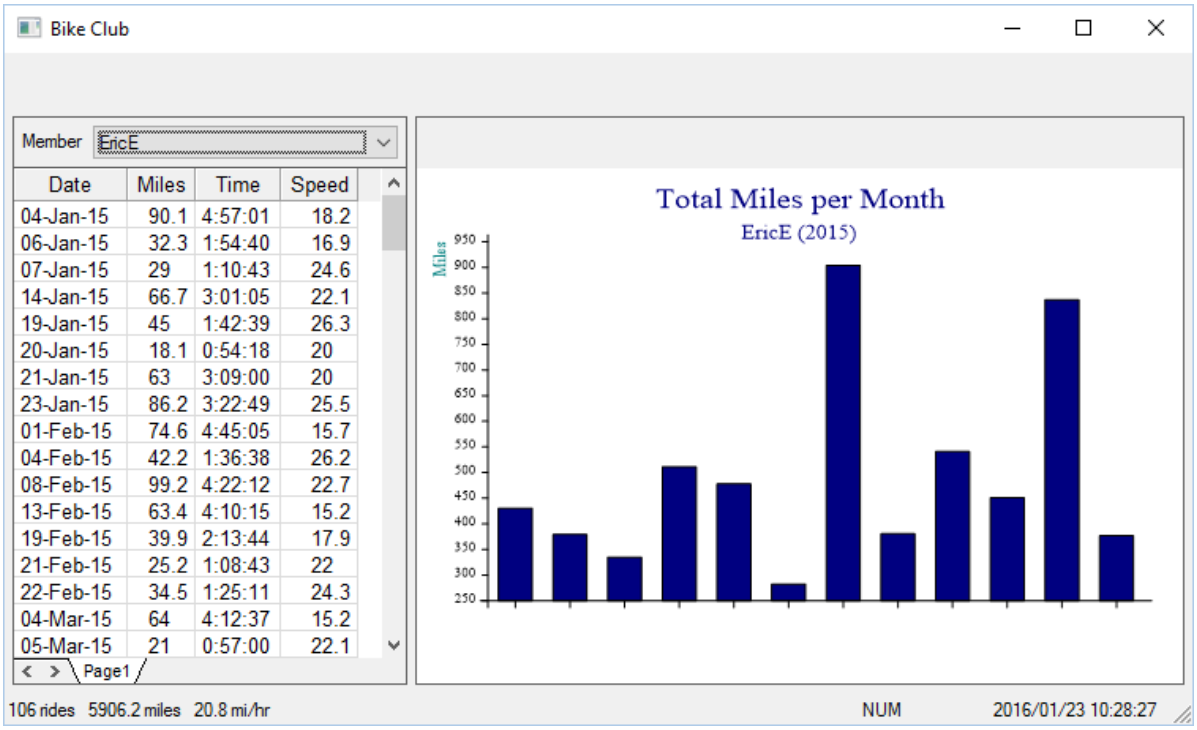
Now paste the following **DrawMilesPerMonthSimpleBarChart** method at the right place in **uBikeClub**:

```
:region-----------------------------------------DrawMilesPerMonthSimpleBarChart
∆DrawMilesPerMonthSimpleBarChart:
    ⍝∇ Draws a miles per month bar chart
    ⍝∇ Syntax:  data←'obj'□wi'DrawMilesPerMonthSimpleBarChart'matrix
    ⍝∇ matrix: the Nx4 member bike rides matrix
    ⍝ Prepare data for the chart
    m←2⊃□warg
```

Copyright © 2016 Eric Lescasse

```
d←⎕wi'DateRep'(m[;1]-2)                    ⍝ convert dates back to yyyymmdd format
g←∪n←⌊d÷100                                ⍝ yyyymm months (sorted)
f←+/¨n⊂m[;2]                               ⍝ total distance per month
h←'Jan' 'Feb' 'Mar' 'Apr' 'May' 'Jun'
h,←'Jul' 'Aug' 'Sep' 'Oct' 'Nov' 'Dec'
h←h[1+100|g-1]                             ⍝ month names
p←'cbMembers'⎕wi'*text'                    ⍝ member name
⍝ Draw the chart
←'lcc'⎕wi'*zReset'432 324
←'lcc'⎕wi'*zHeading' 'Total Miles per Month'
←'lcc'⎕wi'*zSubHeading'(p,' (2015)')
←'lcc'⎕wi'*zYCaption' 'Miles'
←'lcc'⎕wi'*zDrawBarChart'f
←'lcc'⎕wi'*zDrawChart'
:return
:endregion
```

Let's launch the application and select a bike club member: we should see something like:



Looks good, but we'd like to very much enhance the chart.

The **zLCChart** object has many properties and methods which will allow us to do that.

However, we want to retain the **DrawMilesPerMonthSimpleBarChart** method and keep this simple bar chart in our application as our default chart.

So, we will add another chart method, but first we need a way for the user to select the chart he wants to display.

For that we will add buttons above the chart.

Copyright © 2016 Eric Lescasse

Selecting icons for buttons

zObjects include 2 types of .Net buttons:

- znetButton16
- znetButton32

The first one is a small button that can host a 16x16 pixels icon.

The second one is a larger button that can host a 32x32 pixels icon.

The **zObjects.dll** also contains a very large collection of 16x16 and 32x32 icons.

It also contains a C# helper application that you can use to easily select the icons you want to use.

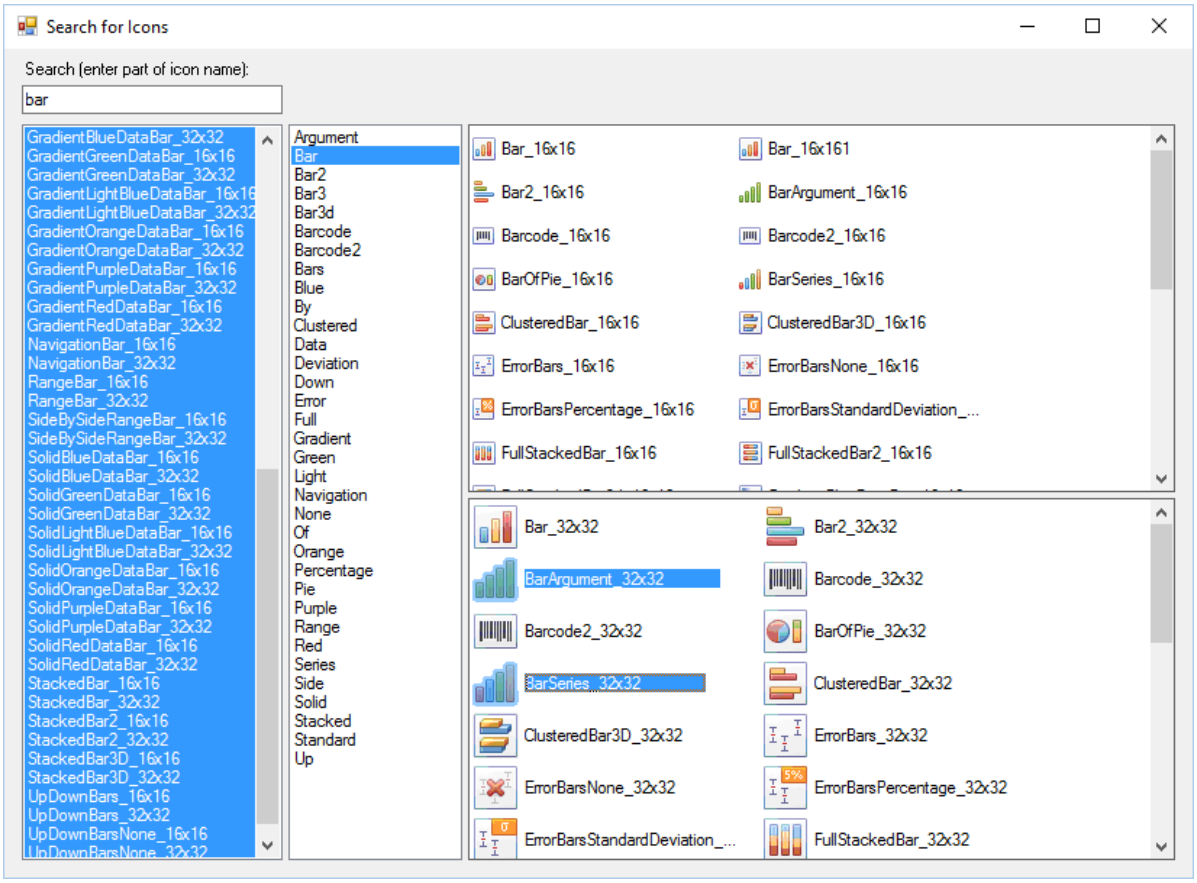
Load the following functions:

```
     ]zload znetSelectIconForm zznetselecticonform
2 objects loaded
```

Launch the application by typing:

```
      zznetselecticonform
```

Copyright © 2016 Eric Lescasse



Enter: **bar** in the top text box.

This will populate the leftmost ListBox with all icon names containing **bar** in their names.

Those names will be split into their constituent names and the unique list of constituent names will appear in the second ListBox.

The second ListBox first item will get automatically selected and as a consequence, elements in the first ListBox that contain this text (Argument) will get selected and the corresponding icons will get displayed in the right Listviews.

Select **Bar** in the second ListBox. Since **Bar** exist within all names in the first ListBox, all elements will get selected in the first ListBox and you'll see all the corresponding icons.

To choose an icon, double click it and it will be printed in the APL Session.

Choose the following icons:

```
BarArgument_32x32
BarSeries_32x32
```

Then close the form.

Copyright © 2016 Eric Lescasse

These are the names we want to use in our **uBikeClub** program.

> **Note:**
> A C# ActiveX DLL such as **zObjects.dll** can contain:
>
> • Pure C# classes which are non Visual Objects (like a **zFile** class or a **zRegex** class)
> • C# User Controls that you can embed in your APL+Win forms (**znetButton32** is an example)
> • complete operational C# forms or even applications (**znetSelectIconForm** is an example)

Adding .Net buttons to our application

First bring in the necessary zObjects:

```
      ]zload znetButton znetButton32
2 objects loaded
```

Then, add the following code to the **uBikeClub** constructor:

```
:region-----------------------------------------Constructor
∆New:
    ⍝∇ Create a new instance of uBikeClub
    ⍝∇ Example:
    ⍝∇       'zt'⎕wi'*Create' 'uBikeClub'
    ←a⎕wi'*Create' 'zForm'('*scale'5)('*caption' 'Bike Club')('margins'5 5)('gaps'5 5)
    ←a⎕wi'*onAction'(∈'uBikeClub' 'zForm' 'zObject',¨⊂'"Action"',⎕tcnl)
    ←zzAddLink a
    ←a⎕wi'*.l1.Create' 'zLabel'('*border'1)('*caption' '')('wherelc'40θ'>'250)
            ('anchor' 'ltb')
    ←a⎕wi'*.l2.Create' 'zLabel'('*border'1)('*caption' '')('wherelc' '=' '>' '=' '>')
            ('anchor' 'ltbr')
    ←a⎕wi'*.sp1.Create' 'zSplitter'('Split' 'l1' 'l2')
    ←a⎕wi'*.l1.cbMembers.Create' 'zCombo'('*style'2 4 8 16 128 256)('wherelc'θ50θ'>')
            ('anchor' 'ltr')('caption' 'Member')
    ←a⎕wi'*.l1.grid.Create' 'zGrid'('*border'1)('wherelc' '>'¯1'>>' '>>'0 0 2 2)
            ('anchor' 'ltrb')
    ←a⎕wi'*.st.Create' 'zStatus'('panes'1 2 3'caps' 'num' 'scroll' '⎕ts'¯12)
    ←a⎕wi'*.l2.fr.Create' 'zFrame'('*style'7)('*caption' '')('wherelc'0 0 42'>>')
            ('anchor' 'ltr')
    ←a⎕wi'*.l2.fr.bn21.Create' 'znetButton32'('wherelc'0 0'∘' '∘'0 ¯1)
            ('*zImage' 'BarArgument_32x32')
    ←a⎕wi'*.l2.fr.bn22.Create' 'znetButton32'('wherelc' '=' '>>' '=' '='0 ¯1)
            ('*zImage' 'BarSeries_32x32')
    ←a⎕wi'*.l2.lcc.Create' 'zLCChart'('wherelc'42 0'>>' '>>')('anchor' 'lrtb')
    ⍝ Events
    ←a⎕wi'AddHandler' '*onShow' 'uBikeClub"onShow"'
    ←'cbMembers'⎕wi'AddHandler' '*onClick' 'uBikeClub"cbMembers_onClick"'
    ←'bn21'⎕wi'AddHandler' '*onXClick' 'uBikeClub"bn21_onXClick"'
    ←'bn22'⎕wi'AddHandler' '*onXClick' 'uBikeClub"bn22_onXClick"'
    :return
    :endregion
```

Note that I have also removed the call to the **ShowDefaultChart** method on the line creating the **zLCChart** instance and increased the top position from **32** to **42**.

Now paste 2 the following 2 event handlers toward the bottom of the **uBikeForm** object:

```
:region----------------------------------------bn21_onXClick
∆bn21_onXClick:
    ⎕←⎕wself⎕wevent⎕warg
    :return
    :endregion

:region----------------------------------------bn22_onXClick
∆bn22_onXClick:
    ⎕←⎕wself⎕wevent⎕warg
    :return
    :endregion
```

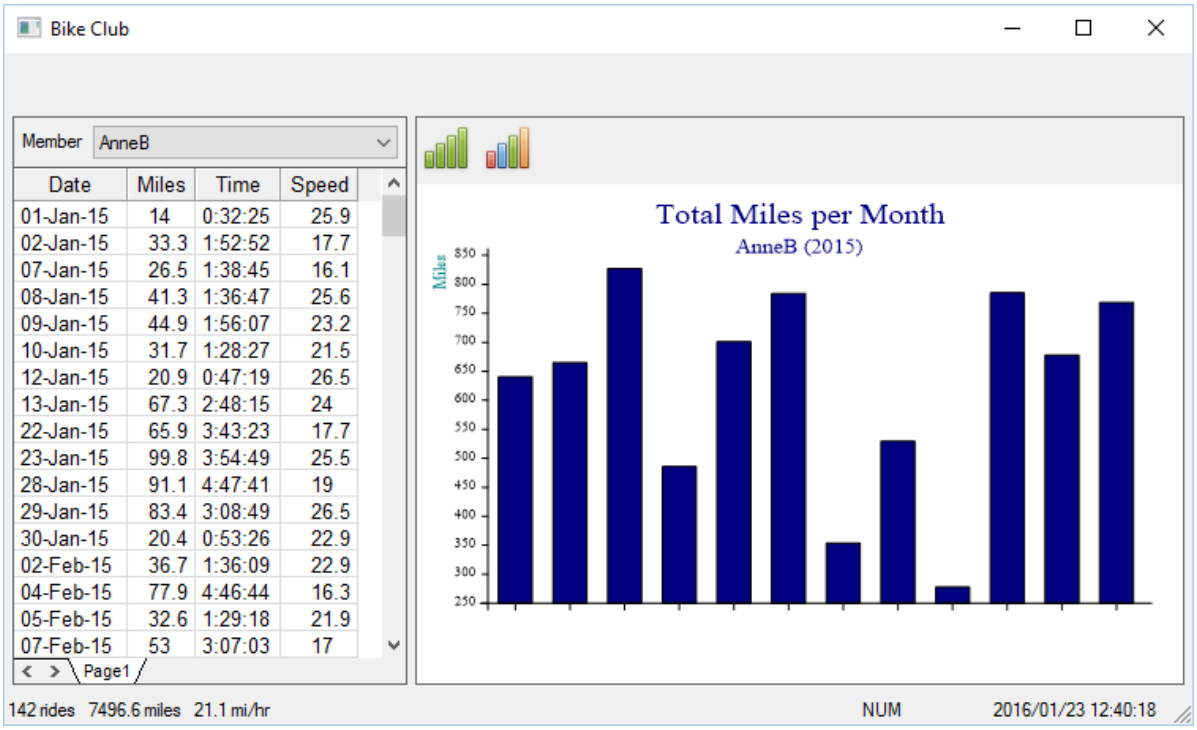
Note:
When adding new event handlers to an object I often start by defining them as running the following instruction:

⎕←⎕wself⎕wevent⎕warg

This allows to immediately run the application and check that the events handlers have been correctly registered and fire ok: if they fire ok, ⎕wself, ⎕wevent and ⎕warg should get printed to the APL Session.

Run the application and select a member. You should now see something like:

Copyright © 2016 Eric Lescasse



We now have 2 nice .Net buttons displayed above the chart in our form.

Click on each of these buttons: you should see the following being printed to the APL Session:

```
ff.l2.fr.bn21 XClick
ff.l2.fr.bn22 XClick
```

Note that .Net objects are much richer objects than the APL+Win standard objects and generally have a large list of properties, methods and events. Normally, only a relatively small percentage of these properties, methods and events should be available to be used from APL because .Net uses many different types (more than 10000), while APL only knows about 4 types: booleans, integers, floating points and strings.

However, **zObjects.dll** includes a large number of automatic **Converters**, which convert .net types which are unknown by APL to structures that APL understands. As a result, a much higher proportion of all the .Net properties, methods and events can be used from APL.

Just as an example, here are all the properties, methods and events available for a **znetButton32** object:

```
      'bn21'⎕wi'allprops'
Properties
----------
*apldata                          *zBackgroundImageLayout
*barwrap                          *zBorderColor
```

```
*border
*children
*class
 class
*clsid
*content
*data
*def
*deferexit
*description
*edge
*enabled
*errorcode
*errormessage
*events
*extent
*help
 help
*helpcontext
*instance
*interface
*links
*methods
*modified
*modifystop
*name
*noredraw
*obj
*opened
*order
*pending
*place
*pointer
*progid
*prompt
*properties
*savecontent
*scale
*self
*size
*state
*suppress
*tabgroup
*tabparent
*tabstop
*theme
*tooltipenabled
*tooltipstyle
*tooltiptime
*tooltipwidth
*unicodebstr
*version
*visible
*where
*xMainProps
*zAccessibleDefaultActionDescription
*zAccessibleDescription
*zAccessibleName
*zAccessibleRole
*zBorderSize
*zBottom
*zBounds
*zCanFocus
*zCanSelect
*zCapture
*zCausesValidation
*zClientRectangle
*zClientSize
*zCompanyName
*zContainsFocus
*zContextMenu
*zCreated
*zDialogResult
*zDisplayRectangle
*zDisposing
*zDock
*zEnabled
*zFlatStyle
*zFocused
*zFont
*zForeColor
*zHandle
*zHasChildren
*zHeight
*zImage
*zImageAlign
*zImageIndex
*zImageKey
*zImageSize
*zInvokeRequired
*zIsAccessible
*zIsDisposed
*zIsHandleCreated
*zIsMirrored
*zLeft
*zLocation
*zMargin
*zMaximumSize
*zMinimumSize
*zName
*zPadding
*zPreferredSize
*zProductName
*zProductVersion
*zRecreatingHandle
*zRight
*zRightToLeft
*zSize
*zTabIndex
*zTabStop
*zTag
*zText
*zTextAlign
*zTextImageRelation
*zTop
*zUseCompatibleTextRendering
*zUseMnemonic
*zUseVisualStyleBackColor
```

```
*zAllowDrop                          *zUseWaitCursor
*zAnchor                             *zVisible
*zAutoEllipsis                       *zWhere
*zAutoScrollOffset                   *zWidth
*zAutoSize                           *ΔΔisnetclass
*zAutoSizeMode                       *ΔΔisusercontrol
*zBackColor                          *ΔΔwhereIc
*zBackgroundImage                    *ΔΔΔsize

Methods
-------
*Close   *Resize              *zInitializeLifetimeService  *zResetForeColor
*Create  *Send                *zInvalidate                 *zResetImeMode
*Defer   *Set                 *zInvalidate_2               *zResetRightToLeft
*Delete  *SetLinks            *zInvalidate_3               *zResetText
*Event   *Show                *zInvalidate_4               *zResumeLayout
*Exec    *zAddControl         *zNotifyDefault              *zResumeLayout_2
*Focus   *zBringToFront       *zPerformClick               *zScale
*Help    *zCreateControl      *zPerformLayout              *zSelect
*Hide    *zDispose            *zPointToClient              *zSendToBack
*Info    *zDoDragDrop         *zPointToScreen              *zSetBounds
*Modify  *zDoc                *zRectangleToClient          *zSetBounds_2
*New     *zEquals             *zRectangleToScreen          *zShow
 New     *zFocus              *zRefresh                    *zSuspendLayout
*Open    *zGetHashCode        *zResetBackColor             *zToString
*Paint   *zGetLifetimeService *zResetBindings              *zUpdate
*Popup   *zGetPreferredSize   *zResetCursor
*Ref     *zHide               *zResetFont

Events
------
*onAction                          *onXForeColorChanged
*onClose                           *onXGiveFeedback
*onContextMenu                     *onXGotFocus
*onDelete                          *onXHandleCreated
*onDestroy                         *onXHandleDestroyed
*onExit                            *onXHelpRequested
*onExitError                       *onXInvalidated
*onFocus                           *onXKeyDown
*onHelp                            *onXKeyPress
*onHide                            *onXKeyUp
*onModified                        *onXLayout
*onMove                            *onXLeave
*onOpen                            *onXLocationChanged
*onPaint                           *onXLostFocus
*onReopen                          *onXMarginChanged
*onResize                          *onXMouseCaptureChanged
*onSend                            *onXMouseClick
*onShow                            *onXMouseDoubleClick
*onUnfocus                         *onXMouseDown
*onXAutoSizeChanged                *onXMouseEnter
*onXBackColorChanged               *onXMouseHover
*onXBackgroundImageChanged         *onXMouseLeave
*onXBackgroundImageLayoutChanged   *onXMouseMove
*onXBindingContextChanged          *onXMouseUp
*onXCausesValidationChanged        *onXMouseWheel
*onXChangeUICues                   *onXMove
*onXClick                          *onXPaddingChanged
*onXClientSizeChanged              *onXPaint
```

```
*onXContextMenuChanged            *onXParentChanged
*onXContextMenuStripChanged       *onXQueryAccessibilityHelp
*onXControlAdded                  *onXQueryContinueDrag
*onXControlRemoved                *onXRegionChanged
*onXCursorChanged                 *onXResize
*onXDisposed                      *onXRightToLeftChanged
*onXDockChanged                   *onXSizeChanged
*onXDoubleClick                   *onXStyleChanged
*onXDragDrop                      *onXSystemColorsChanged
*onXDragEnter                     *onXTabIndexChanged
*onXDragLeave                     *onXTabStopChanged
*onXDragOver                      *onXTextChanged
*onXEnabledChanged                *onXValidated
*onXEnter                         *onXValidating
*onXFontChanged                   *onXVisibleChanged
```

You can use many of those properties, methods and events, but not all.

Copyright © 2016 Eric Lescasse

Developing a better bar chart

Paste the following method at the right location within uBikeClub:

```
:region----------------------------------------DrawMilesPerMonthBarChart
∆DrawMilesPerMonthBarChart:
    ⍝∇ Draws a miles per month bar chart
    ⍝∇ Syntax:  data←'obj'⎕wi'DrawMilesPerMonthBarChart'matrix
    ⍝∇ matrix: the Nx4 member bike rides matrix
    ⍝ Prepare data for the chart
    m←2⊃⎕warg
    d←⎕wi'DateRep'(m[;1]-2)                  ⍝ convert dates back to yyyymmdd format
    g←Un←⌊d÷100                              ⍝ yyyymm months (sorted)
    f←+/¨n⊂m[;2]                             ⍝ total distance per month
    h←'Jan' 'Feb' 'Mar' 'Apr' 'May' 'Jun'
    h,←'Jul' 'Aug' 'Sep' 'Oct' 'Nov' 'Dec'
    h←h[1+100|g-1]                           ⍝ months names
    p←'cbMembers'⎕wi'*text'                  ⍝ member name
    ⍝ Draw the chart
    ←'lcc'⎕wi'*zReset'432 324
    ←'lcc'⎕wi'*zHeading' 'Total Miles per Month'
    ←'lcc'⎕wi'*zSubHeading'(p,' (2015)')
    ←'lcc'⎕wi'*zYCaption' 'Miles'
    ←'lcc'⎕wi'*zBarChartStyle' (BarChartStyles.ValueTags)
    ←'lcc'⎕wi'*zValueTagStyle' (ValueTagStyles.Vertical+ValueTagStyles.Inside+
          ValueTagStyles.RecolorOutside)
    ←'lcc'⎕wi'*zSetValueFont' 'Arial' 12 FontStyle.Bold Color.Black
    ←'lcc'⎕wi'*zSetColors' Color.Red
    ←'lcc'⎕wi'*zSetFillStyles' FillStyle.GradientBottom
    ←'lcc'⎕wi'*zXAxisStyle' (XAxisStyles.GridLines)
    ←'lcc'⎕wi'*zSetHeadingFont' 'Helvetica' 24 FontStyle.Bold
    ←'lcc'⎕wi'*zSetSubheadingFont' 'Helvetica' 16 FontStyle.Bold
    ←'lcc'⎕wi'*zSetXLabelFont' 'Calibri' 10
    ←'lcc'⎕wi'*zSetYLabelFont' 'Calibri' 10
    ←'lcc'⎕wi'*zSetYCaptionFont' 'Calibri' 14
    ←'lcc'⎕wi'*zMarginLeft'50
    ←'lcc'⎕wi'*zMarginTop'60
    ←'lcc'⎕wi'*zSetXLabels'(⍕¨h)
    →'lcc'⎕wi'*zSetYRange'0 1200
    ←'lcc'⎕wi'*zDrawBarChart'f
    ←'lcc'⎕wi'*zDrawChart'
    :return
    :endregion
```

This method is similar to the **DrawMilesPerMonthSimpleBarChart** method, but uses many more **zLCChart** properties and methods.

It shows how we can customize a chart to obtain the exact look and feel that we want.

Hooking up the button click events

Change the 2 button click events to read:

Copyright © 2016 Eric Lescasse

```
:region----------------------------------------------bn21_onXClick
∆bn21_onXClick:
    :if~0∈⍴m←zzDEB'cbMembers'⎕wi'*text'
        d←⎕wi':GetMemberData'm
        ←⎕wi':DrawMilesPerMonthSimpleBarChart'd
        ←⎕wi'*:ΔΔcharttype'1
    :endif
    :return
    :endregion

:region----------------------------------------------bn22_onXClick
∆bn22_onXClick:
    :if~0∈⍴m←zzDEB'cbMembers'⎕wi'*text'
        d←⎕wi':GetMemberData'm
        ←⎕wi':DrawMilesPerMonthBarChart'd
        ←⎕wi'*:ΔΔcharttype'2
    :endif
    :return
    :endregion
```

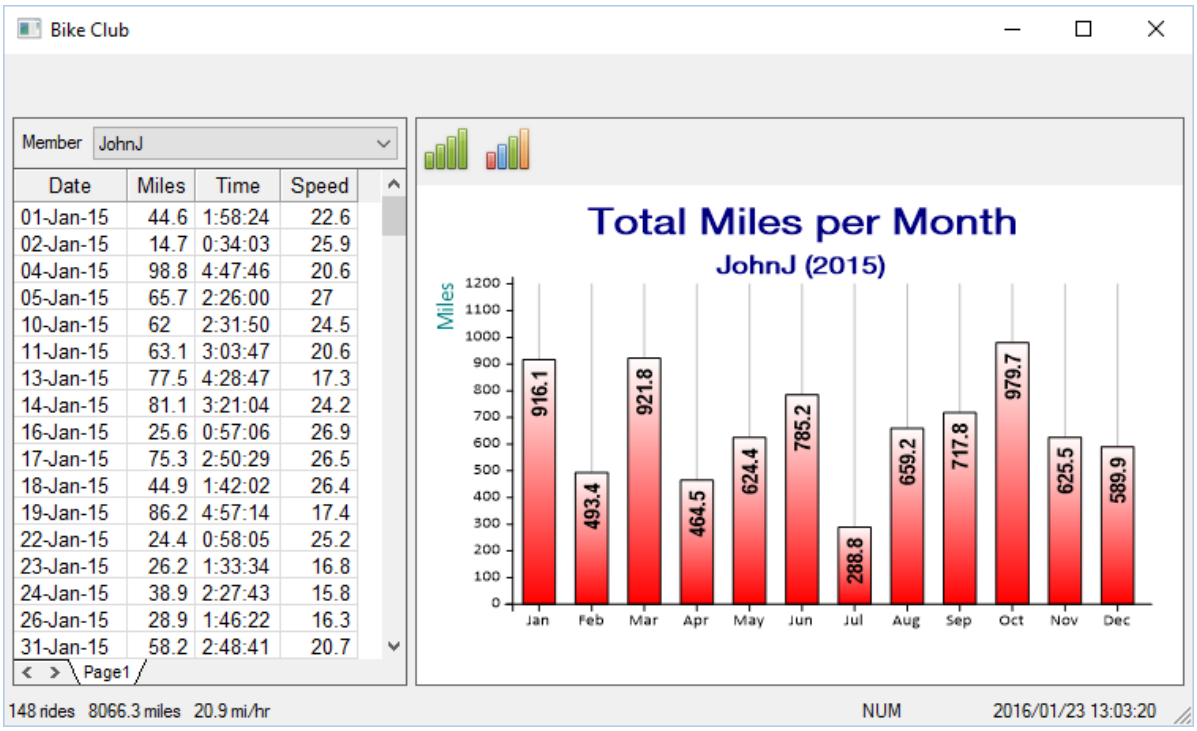
Note that we are saving the chart type in a User Defined variable called **ΔΔcharttype**.

We should update the **cbMembers_onClick** to read:

```
:region----------------------------------------------cbMembers_onClick
∆cbMembers_onClick:
    m←zzDEB⎕wi'*text'
    d←⎕wi':GetMemberData'm
    ←⎕wi':PopulateGrid'd
    ←⎕wi':UpdateStatus'd
    :select 1⌈⎕wi'*:ΔΔcharttype'
    :case 1 ⋄ ←⎕wi':DrawMilesPerMonthSimpleBarChart'd
    :case 2 ⋄ ←⎕wi':DrawMilesPerMonthBarChart'd
    :endselect
    :return
    :endregion
```

Let's start the application and select a member. Then click on the second button. We should see something like:

Copyright © 2016 Eric Lescasse



Let's finish up this sample little application by adding 2 new buttons at the top of the form to allow adding or removing data lines in the grid.

Adding new .net buttons

Here is the code we need to paste in the Constructor of the **uBikeClub** object:

```
    ←a⎕wi'*.bn1.Create' 'znetButton32'('wherelc'0 0'∘' '∘'0 ¯1)
             ('*zImage' 'AddDatabaseRecord32np')
    ←a⎕wi'*.bn2.Create' 'znetButton32'('wherelc' '=' '>>' '=' '='0 ¯1)
             ('*zImage' 'DeleteDatabaseRecord32np')
…
    ←'bn1'⎕wi'AddHandler' '*onXClick' 'uBikeClub"bn1_onXClick"'
    ←'bn2'⎕wi'AddHandler' '*onXClick' 'uBikeClub"bn2_onXClick"'
```

and toward the bottom of the **uBikeClub** object:

```
:region----------------------------------------------bn1_onXClick
∆bn1_onXClick:
    :if~0∈⍴m←zzDEB'cbMembers'⎕wi'*text'
        d←⎕wi':GetMemberData'm
        ←⎕wi':AddGridRow'
    :endif
    :return
    :endregion

:region----------------------------------------------bn2_onXClick
∆bn2_onXClick:
    :if~0∈⍴m←zzDEB'cbMembers'⎕wi'*text'
        d←⎕wi':GetMemberData'm
        ←⎕wi':DeleteGridRow'
```

Copyright © 2016 Eric Lescasse

```
    :endif
    :return
    :endregion
```

Note:
Again, in a real life application I would have used more meaningful names for the User Interface objects, like **bnAddGridRow** and **bnDeleteGridRow** maybe. I am using short names here like **bn1**, **bn2** to spare space in this document.

Deleting rows in the grid

Let's start by the simplest.

Deleting grid rows is relatively easy. When the user clicks the second top button, we want to delete the currently active grid row.

Paste the following method in **uBikeClub**:

```
:region-----------------------------------------DeleteGridRow
∆DeleteGridRow:
    ⍝∇ Deletes the currently active grid row
    ⍝∇ Syntax:  'obj'⎕wi'DeleteGridRow'
    (r c)←(⊂'grid')⎕wi¨'*xRows' '*xCols'      ⍝ get the new grid nb of rows and cols
    d←'grid'⎕wi'*xRow'                        ⍝ currently active grid row
    ←'grid'⎕wi'*XDeleteRows'd                 ⍝ delete current row
    ←'grid'⎕wi'*XRedraw'                      ⍝ redraw the grid
    v←'grid'⎕wi'*xValue'(⍳r)(⍳c)              ⍝ read the entire grid content
    m←zzDEB'cbMembers'⎕wi'*text'
    ←⎕wi':SetMemberData'm v
    :return
    :endregion
```

Please carefully note that we are paying attention to not mix User Interface raw APL code and Data Layer raw APL code here either, so we are calling another method called **SetMemberData** to save the new data to the database.

Note:
The choice of property, method and event names is very important!

I would even say that the choice of names in a computer application is one of the most important things, that needs to be carefully done and thought about.

We call the method that writes data back to the database **SetMemberData**, because we have named the method that reads data from the database **GetMemberData**.

Copyright © 2016 Eric Lescasse

Moreover, if you remember, the **GetMemberData** method was not only reading data from the database but was also transforming it to data that the grid could understand.

So it seems clean to design the **SetMemberData** method to do the exact reverse operation. It will first convert the grid data to database data format and will then save the data to the database.

So, let's paste the following method in **uBikeClub**:

```
:region-----------------------------------------SetMemberData
∆SetMemberData:
    ⍝∇ Saves the bike rides data for the specified member to the database
    ⍝∇ Syntax:  'obj'⎕wi'SetMemberData'member matrix
    ⍝∇ member: the name of a bike club member
    ⍝∇ matrix:  the Nx4 bike rides data matrix for this member (in grid format)
    t←⎕wi'*ΔΔtieno'                          ⍝ retrieve the tie number
    c←⎕fread t 1
    :if(↑⍴c)<e←c⎕warg[2]
        ⍝ Member not found: do nothing
    :else
        m←24 60 60
        d←3⊃⎕warg
        d[;1]←¯2+⎕wi'DateRep'(d[;1])         ⍝ convert nb of days to yyyymmdd
        d[;3]←100⊥⍉Td[;3]××/m                ⍝ convert decimal times to hhmmss
        d⎕freplace t(10+e)
    :endif
    :return
    :endregion
```

Note that the fact we have designed **SetMemberData** to be doing the exact reverse work that **GetMemberData** does, has made writing **SetMemberData** much easier.

I did it by just copy/pasting the **GetMemberData** method, then changing its name **to SetMemberData** and adapting a few things in the method.

You can compare **SetMemberData** with **GetMemberData** (reproduced here):

```
:region-----------------------------------------GetMemberData
∆GetMemberData:
    ⍝∇ Retrieves the bike rides data for the specified member
    ⍝∇ Syntax:  data←'obj'⎕wi'GetMemberData'member
    ⍝∇ member: the name of a bike club member
    ⍝∇ data:  the Nx4 bike rides data matrix for this member
    t←⎕wi'*ΔΔtieno'                          ⍝ retrieve the tie number
    c←⎕fread t 1
    :if(↑⍴c)<e←c⎕warg[2]
        ⎕wres←0 4⍴''
    :else
        m←24 60 60
        d←⎕fread t(10+e)
```

```
        d←d[⍋d[;1];]                          ⍝ sort matrix by increasing dates
        d[;1]←2+⎕wi'DateBase'(d[;1])          ⍝ convert yyyymmdd to grid dates
        d[;3]←(m⊥100 100 100⊤d[;3])÷×/m       ⍝ convert hhmmss times to grid times
        ⎕wres←d
    :endif
    :return
    :endregion
```

One can see how close and symmetrical these 2 functions are: having adopted some good development naming conventions and practices has made developing **SetMemberData** really easy.

Confirming deletion

It's aslo a good practice to request the user to confirm a deletion operation.

For that, we can use the **zzYesNo** or **zzYesNoCancel** utility. These functions (just like many functions starting with **zz**) have been created by **zzInit**.

Since calling **zzYesNo** is User Interface stuff, we can add it directly to our **DeleteGridRow** method:

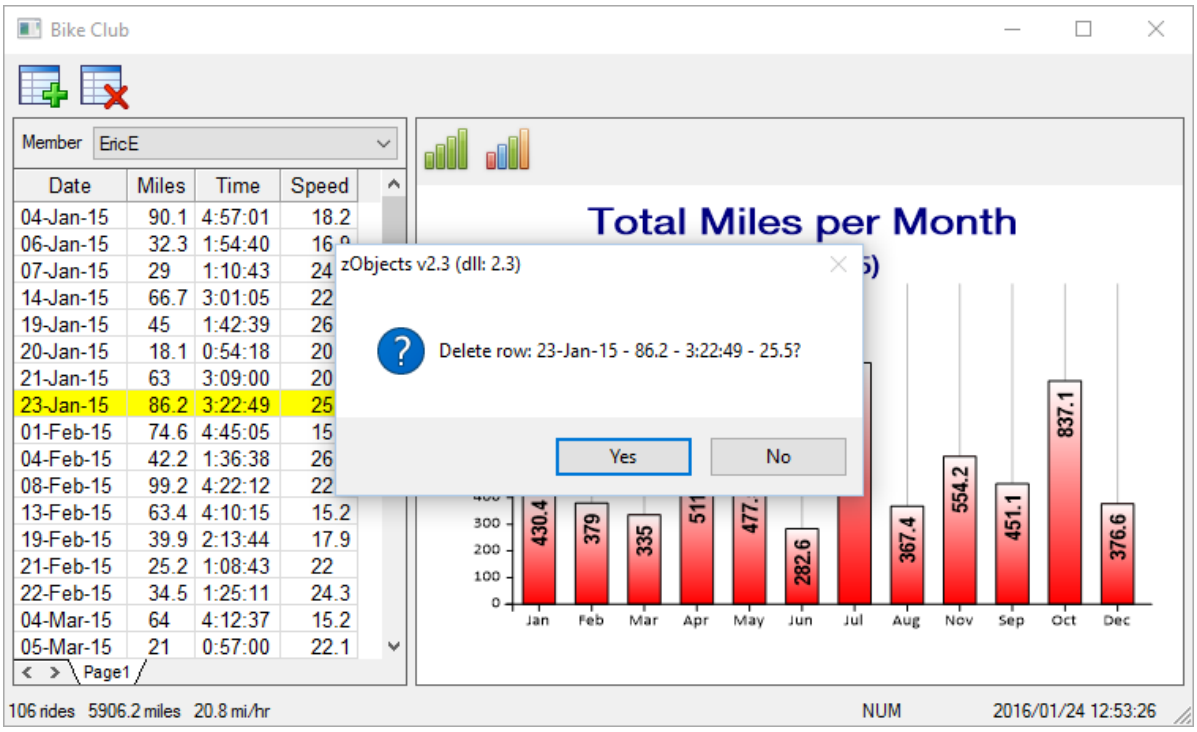
```
:region----------------------------------------DeleteGridRow
∆DeleteGridRow:
    ⍝∇ Deletes the currently active grid row
    ⍝∇ Syntax:  'obj'⎕wi'DeleteGridRow'
    (r c)←(⊂'grid')⎕wi¨'*xRows' '*xCols'      ⍝ get the new grid nb of rows and cols
    d←'grid'⎕wi'*xRow'                        ⍝ currently active grid row
    e←3↓∈,(⊂' - '),¨'grid'⎕wi'*xText'd(⍳c)
    ←'grid'⎕wi'*xColorBack'd(⍳c)(256⊥0 255 255)
    ←'grid'⎕wi'*XRedraw'd 1 1 4
    f←zzYesNo('Delete row: ',(⍕e),'?')zzOKCaption(⎕wi'*:hwnd')
    :if 1=f
        ←'grid'⎕wi'*XDeleteRows'd             ⍝ delete current row
        ←'grid'⎕wi'*XRedraw'                  ⍝ redraw the grid
        v←'grid'⎕wi'*xValue'(⍳r)(⍳c)          ⍝ read the entire grid content
        m←zzDEB'cbMembers'⎕wi'*text'
        ←⎕wi':SetMemberData'm v
    :else
        ←'grid'⎕wi'*xColorBack'd(⍳c)(256⊥255 255 255)
        ←'grid'⎕wi'*XRedraw'd 1 1 4
    :endif
    :return
    :endregion
```

I have highlighted the changes to the **DeleteGridRow** method in yellow.

Let's start the application, load a member data, select a row and click the **Delete Row** button.

We should see something like this:

Copyright © 2016 Eric Lescasse



You may notice that the Yes/No message box is perfectly centered on the form.

Adding rows to the grid

We could easily add a row to the grid when the user clicks the first button at the top left of the form and let him enter data directly in the grid, but as an exercise, we will instead create a secondary form to collect his input.

This form will be a modal form or dialog box, so let's bring in the **zDialogBox** object:

```
      ]zload zDialogBox
1 object loaded
```

Now, let's create a new form in our application that derives from **zDialogBox**:

```
      'zz'□wi'Derive' 'uGridRowForm' 'zDialogBox' 'zAPLFormTemplate'
Class <uGridRowForm> successfully created and registered!
```

We create a new **uGridFormRow** object that inherits from **zDialogBox** (which itself inherits from **zForm**, which inherits from an APL+Win standard **Form** object).

Note:
Developing with **zObjects** consists almost exclusively of creating objects using the **zObject Derive** method (as done above).

Your own application object names should never start with a **z**.

Copyright © 2016 Eric Lescasse

You should also never make any change to a **z** function otherwise your changes would be overridden by a **zObjects** update.

Instead, you want to create objects that inherits from existing **zObjects** using the **Derive** method. This way your objects are "**yours**", but inherit all the properties, methods and events of existing zObjects, i.e. all their power.

You may also create objects that derive from other objects you have already created yourself.

Your work as an application developer is to add your own **properties**, your own **methods** and your own **event handlers** to **your objects**, therefore making them even more powerful than the objects they derive from!

zObjects is really a product made for developers.

Why does **zObjects** contain a **zDialogBox** object in addition to a **zForm** object?

The reason is that a **zDialogBox** object is a more sophisticated object than a **zForm** object.

First, it contains an **OK** button and a **Cancel** button which are properly placed at the bottom right of the dialog box and are attached to it.

Second, a **zDialogBox** object contains several additional properties, like **changed**, **datacontrols, datavalues** and **dialogresult** which we will use soon.

Let's create an instance of a **zDialogBox** object, just to see what it looks like:

```
'db'⎕wi'*Create' 'zDialogBox' 'Show'
```

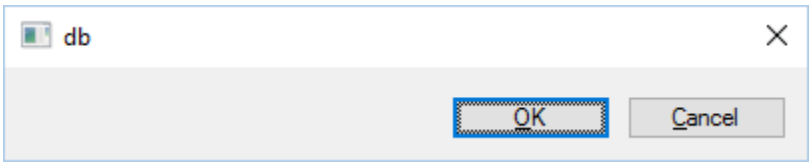
Nothing happens! The reason is that a **zDialogBox** is by default displayed outside of the screen (at location ¯5000 ¯5000). This is to avoid any flicker while the **zDialogBox** is populated with data.

If you want to see the **zDialogBox**, you should call a method like **CenterScreen** or set its **where** property.

```
'db'⎕wi'*Create' 'zDialogBox'('*size'300 400)'CenterScreen' 'Show'
```

Also a **zDialogBox** always gets "AutoSized" after the **OK** and **Cancel** button have been added to it, so here is how it looks like when it does not contain any controls (other than the **OK** and **Cancel** buttons):

Copyright © 2016 Eric Lescasse



A **zDialogBox** is typically displayed using the **Wait** method, as a modal form.

When the user closes the dialog box, the **Wait** method terminates and the following **zDialogBox** properties are available:

dialogresult	contains **'ok'** if user clicked the **OK** button or **'cancel'** if he clicked the **Cancel** button
dialogvalues	a nested vector containing values for each control in the form
dialogcontrols	a nested vector of all controls contained in the form

All the above properties are automatically set by the **zDialogBox** object and you don't need to care about them except just use them after the **zDialogBox** closes.

That makes it extremely simple to develop and use a dialog box.

Paste the following code for the **AddGridRow** method in the **uBikeClub** object:

```
:region----------------------------------------AddGridRow
∆AddGridRow:
    ⍝∇ Displays a form allowing to enter data for a new row added to the grid
    ⍝∇ Syntax:  'obj'⎕wi'AddGridRow'

    ⍝ Display the dialog box and wait on it
    ←'grf'⎕wi'*Create' 'uGridRowForm' 'Show'('topmost'1)
            ('CenterOn'(⎕wi'*:self'))'*Wait'

    :if'ok'≡'grf'⎕wi'dialogresult'            ⍝ if user clicked the OK button
        ⍝ Get data entered in the dialog box
        (d e f g)←'grf'⎕wi'dialogvalues'      ⍝ d=date, e=dist, f=time, g=speed
        m←24 60 60
        d←⍎2↑⎕wi'DateBase'(100⊥3↑d)
        f←(m⊥3↑3↓f)÷×/m
        h←'dd-MMM-yy' '#.#' 'H:mm:ss' '#.#'
        ⍝ Update the grid
        (r c)←(⊂'grid')⎕wi¨'*xRows' '*xCols'  ⍝ get the new grid nb of rows and cols
        ←'grid'⎕wi'*XInsertRows'(r+.5)        ⍝ append a new row to the grid
        ←'grid'⎕wi'*xValueType'(r+1)(⍳c)(1 4⍴3 1 3 1) ⍝ 3(=date) 1(=number)
        ←'grid'⎕wi'*xRow'(r+1)                ⍝ move to the new row
        ←'grid'⎕wi'*xValue'(r+1)(⍳c)(d e f g) ⍝ populate the new row
        ←'grid'⎕wi'*xFormat'(r+1)(⍳c)h        ⍝ format cells
        ←'grid'⎕wi'*xRowSize'(r+1)17          ⍝ set the new row height
        ←'grid'⎕wi'*XRedraw'                        ⍝ refresh the grid
        ⍝ Save the grid data to the database
        v←'grid'⎕wi'*xValue'(⍳r)(⍳c)                ⍝ read the entire grid content
        m←zzDEB'cbMembers'⎕wi'*text'
```

```
      ←⍞wi':SetMemberData'm v
    :endif
    :return
    :endregion
```

Here are a few comments about this method:

- The **uGridRowForm** is first displayed and centered on its parent form using the **CenterOn** method

- Note that if you displayed the main form using the **DemoShow** method and made it **topmost**, you need to also make the dialog box topmost with **'topmost'1** otherwise it will be displayed behind its parent form.

- Also note that we must call the **Show** method before calling **CenterOn** in the case of a **zDialogBox**: the reason is that it is in its **onShow** event handler that the **OK** and **Cancel** button are automatically added to the form. This makes sense since these buttons need be added at the bottom of the form, below all other controls and **onShow** is a good place to do that since we know the form has already been populated with all its controls when its **onShow** event occurs.

- As soon as the **Wait** method terminates, we check how the user closed the dialog box and we proceed to add a row to the grid only if he clicked **OK**.

- In that case, we get all the values he entered in the dialog box by querying the **dialogvalues** property. In our case, there are only 4 controls in the dialogbox so it's easy. But, if there were a lot of controls in the form, we would use the **dialogcontrols** property as follows:

  ```
  (dialogcontrols⍳⊂'dist')⊃dialogvalues
  ```

  to retrieve the value for the **dist** control, for example.

The **uGridRowForm** object will make use of a number of a few new **zObjects**, so let's bring them in from the zObjects UCMD file:

```
      ]zload zDateTime zEditNum zSimpleEdit zEdit zButton /r
5 objects loaded
```

Here is the code for the **uGridRowForm** object:

```
    ∇ a uGridRowForm b;s;t;v;⎕io
[1]   ⍝∇ a uGridRowForm b -- The uGridRowForm object is a Form (or MDIForm)
            derived from a zDialogBox object
[2]   ⍝∇ a ←→ object name (may be omitted if ⎕wself is set)
[3]   ⍝∇ b ←→ 'property'
[4]   ⍝∇   or 'property'value
[5]   ⍝∇   or 'Method'
[6]   ⍝∇   or 'Method'argument1 ... argumentN
[7]   ⍝∇ Requires: (F) zObject
[8]   ⍝∇ zObjects v2.3
[9]   ⍝∇ Copyright(c)Lescasse Consulting 2013-2016
[10]  ⍝∇ ELE23jan16
[11]
[12]  :region-----------------------------------------Action
[13]  ⎕io←1
[14]  :if ⎕monadic ⋄ a←⎕wself ⋄ :endif
[15]  :if b≡'Action' ⋄ →('?'=↑b←↑⎕warg)ρ⍙⍙Help ⋄ :endif
[16]  →(8≠''ρ⎕idloc'⍙',b)ρ⍙⍙Inherit ⋄ →⍎'⍙',b
[17]  :endregion
[18]
[19]  :region-----------------------------------------Constructor
[20]  ⍙New:
[21]      ⍝∇ Create a new instance of uGridRowForm
[22]      ⍝∇ Example:
[23]      ⍝∇       'zt'⎕wi'*Create' 'uGridRowForm'
[24]      ←a⎕wi'*Create' 'zDialogBox'('*scale'5)('*caption' 'Enter Bike Ride Info')
[25]      ←a⎕wi'*onAction'(∈'uGridRowForm' 'zDialogBox' 'zForm' 'zObject',¨⊂'"Ac-
tion"',⎕tcnl)
[26]      ←zzAddLink a
[27]      ←a⎕wi'*.date.Create' 'zDateTime'('*style'1)('wherelc'⍬100 21 140)
                ('caption' 'Date:')('*format' 'MM/dd/yy ddd')
[28]      ←a⎕wi'*.dist.Create' 'zEditNum'('wherelc' '>' '='⍬ 50)
                ('caption' 'Distance (mi):')
[29]      ←a⎕wi'*.time.Create' 'zDateTime'('*style'3)('wherelc' '>' '=' '=date' '=')
                ('caption' 'Time (hh:mm:ss):')('*format' 'HH:mm:ss')
[30]      ←a⎕wi'*.speed.Create' 'zEditNum'('wherelc' '>' '=' '=dist' '=')
                ('caption' 'Speed (mi/hr):')('enabled'0)
[31]      ⍝ Events
[32]      ←'dist'⎕wi'AddHandler' '*onChange' 'uGridRowForm"disttime_onChange"'
[33]      ←'time'⎕wi'AddHandler' '*onChange' 'uGridRowForm"disttime_onChange"'
[34]      :return
[35]  :endregion
[36]
[37]  :region-----------------------------------------class
[38]  ⍙class:
[39]      ⎕wres←'uGridRowForm'
[40]      :return
[41]  :endregion
[42]
[43]  :region-----------------------------------------help
[44]  ⍙help:
[45]      ⍝∇ ←'zt'⎕wi'*Create' 'uGridRowForm'
[46]      ⎕wres←⎕wi'GetHelp' 'help'
[47]      :return
[48]  :endregion
[49]
```

```
[50]  :region----------------------------------------------disttime_onChange
[51]  ∆disttime_onChange:
[52]      v←'dist'⎕wi'value'                                ⍝ distance
[53]      t←24 60 60⊥3↑3↓'time'⎕wi'value'                   ⍝ time in seconds
[54]      :try
[55]          s←.1×⌊.5+10×3600×v÷t                          ⍝ average speed
[56]          ←'speed'⎕wi'value's                           ⍝ speed
[57]      :catchall
[58]      :endtry
[59]      :return
[60]  :endregion
[61]
[62]
[63]  :region-------------------------------------------Internal
[64]  ∆∆Help:
[65]      ⎕wres←⎕wi'GetHelp'(('?'=↑b)↓b)
[66]      :return
[67]  ∆∆Inherit:
[68]      ⎕wres←⎕wi'*'
[69]  :endregion
[70]
    ∇
```

Here are a couple of notes about this object:

- The User Interface is all done in just 4 lines (27-30)
- Only 2 events are handled: the **onChange** event on the **dist** Edit box and the **on-Change** event on the **time** DateTime control. Note that, since the event handlers for these events are the same, we are using only one event handler (**distime_on-Change**) for both events to avoid duplicate code in our object.
- Note that we don't set the **dialogresult**, **dialogvalues**, **dialogcontrols** properties in **uGridRowForm**: all this is inherited from **zDialogBox** and **zForm**.
- Looking at the **disttime_onChange** event handler notice that we are using the **value** property of the **dist**, **time** and **speed** controls. In zObjects, all controls have a **value** property which returns the most appropriate control "value":

  For example, a **zEdit** control returns the control text as a string in its **value** property,

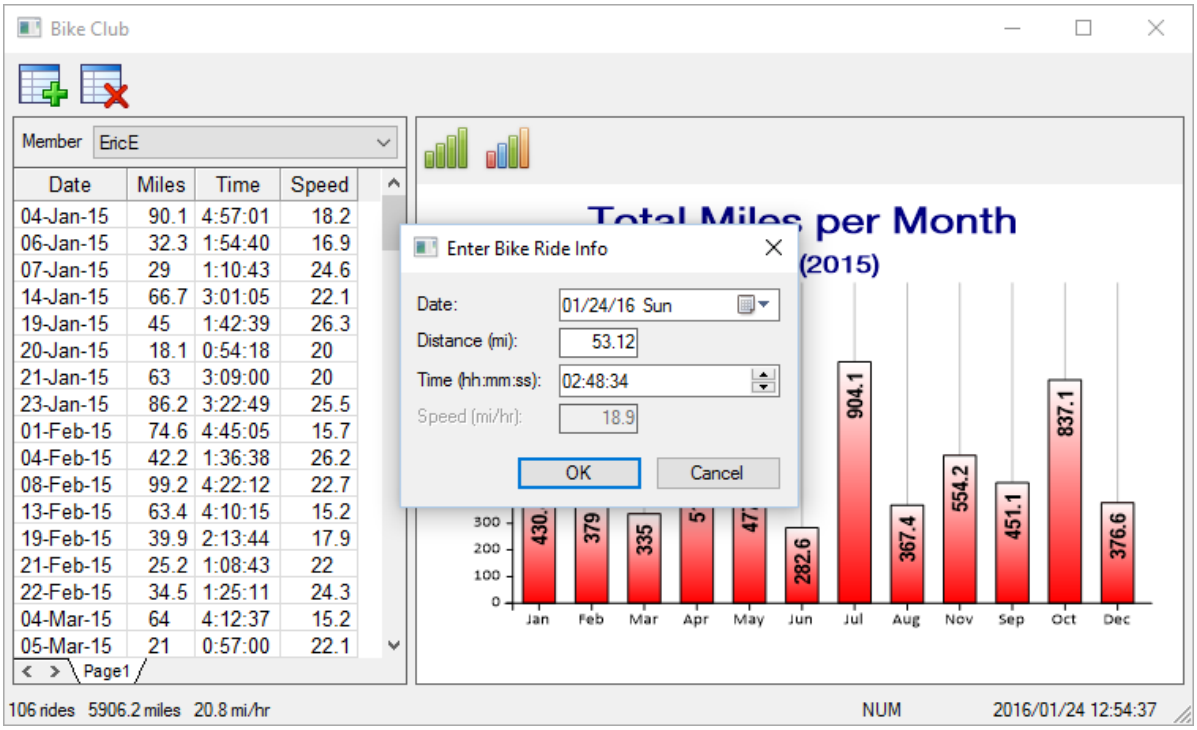
  A **zEditNum** control returns the numeric value displayed in the control in the **value** property,

  Etc.
- The fact that each **zObject** has a **value** property of the right type is very handy, es-pecially in event handlers such as the **disttime_onChange** event handler which are therefore generally very simple and easy to write.

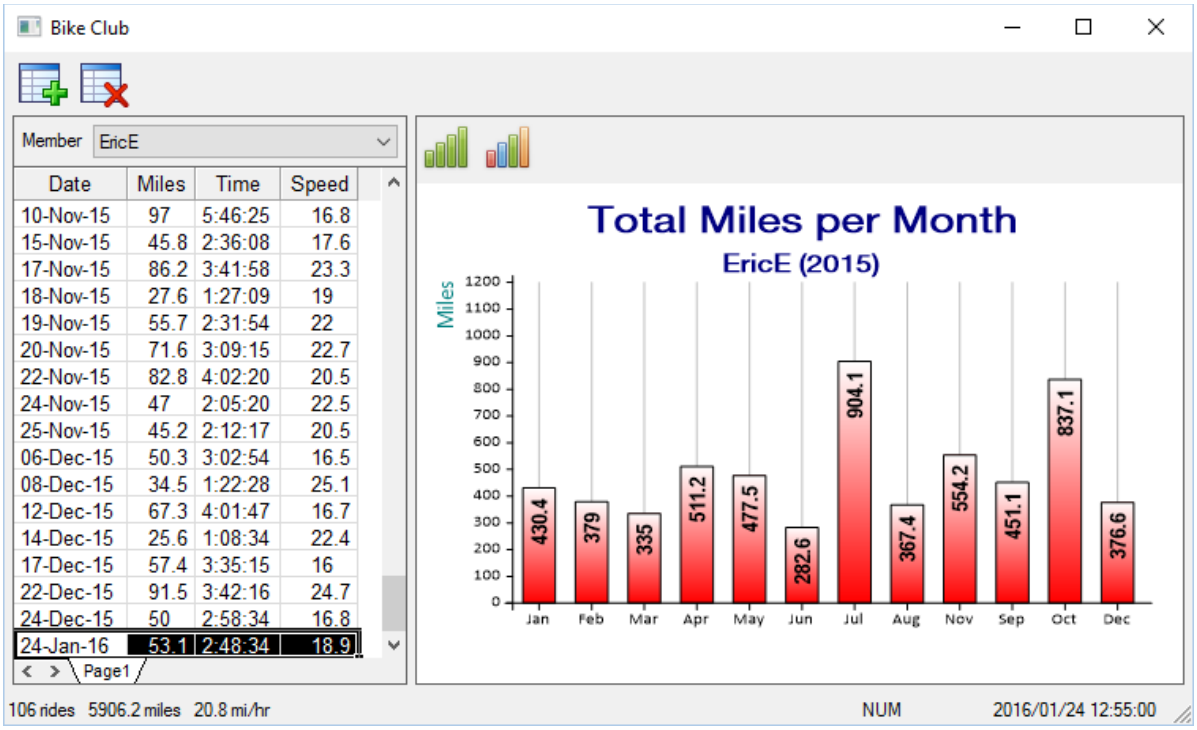
Copyright © 2016 Eric Lescasse

We are ready to start our application and test it.

Let's choose a bike club member, then click the top left button and enter some data in the dialog box:



When we click the **OK** button, a new row is added to the grid and saved to the database:



Copyright © 2016 Eric Lescasse

Our sample application

Every developer knows that an application, as small as it may be, is never finished.

To make our small application more complete and a good Windows citizen application, there are many other things we should do:

- Add accelerator keys for the various buttons
- Add tooltips to the various buttons
- Force the focus to be on the **cbMembers** combo box when the application starts
- Show the buttons glyphs as disabled when a Member has not yet been selected in the combo box
- Prompt for confirmation when closing the application
- Regroup and sort methods belonging to the Data Layer in a DataLayer region (using the :region control structure)
- Regroup and sort methods belonging to the User Interface Layer in a UserInterface-Layer region (using the :region control structure)
- Observing that we've had to use the same kind of code several times to convert data dates to grid dates, and data times to grid times, we should have created a Business Layer and we should have created separate methods in the Business Layer for just those conversions
- Etc.

Our goal was not to write a perfect and complete application but rather to show how to develop an application using **zObjects**.

Additional benefits you get from using objects

The fact we have written our application in an Object Oriented way has additional benefits.

You must always remember that an object of yours, here **uBikeClub**, is very similar to a **zObject** and behaves exactly like a **zObject**. Additionally, it also inherits from a real lot of behaviors built in the zObjects system itself (just because your objects inherit from zObjects).

For example, we can create an instance of **uBikeClub**:

```
'bk'□wi'*Create' 'uBikeClub' 'Show'
```

Then query the **bk** object properties, methods, events:

Copyright © 2016 Eric Lescasse

```
      80 zzTELPRINT'bk'⎕wi'properties'
class  help

      80 zzTELPRINT'bk'⎕wi'methods'
AddGridRow                      GetMemberData    PopulateGrid
DeleteGridRow                   GetMemberNames   PopulateMembersCombo
DrawMilesPerMonthBarChart       New              SetMemberData
DrawMilesPerMonthSimpleBarChart OpenDatabase     UpdateStatus

      80 zzTELPRINT'bk'⎕wi'events'
bn1_onXClick   bn22_onXClick  cbMembers_onClick
bn21_onXClick  bn2_onXClick   onShow
```

We can get documentation about any of these properties, methods or events (as long as we have done our job correctly and documented these properties and methods inside our object):

```
      'bk'⎕wi'?GetMemberNames'
Retrieves a sorted list of member names
Syntax:  names←'obj'⎕wi'GetMemberNames'
names: sorted nested vector of all member names

      'bk'⎕wi'?GetMemberData'
Retrieves the bike rides data for the specified member
Syntax:  data←'obj'⎕wi'GetMemberData'member
member: the name of a bike club member
data:  the Nx4 bike rides data matrix for this member
       [;1]= dates in number of days since 19000101
       [;2]= bike ride distance (in miles)
       [;3]= bike ride time in % of 24 hours
       [;4]= bike ride average speed (in miles/hour)
Example:
      'bk'⎕wi'GetMemberData' 'EricE'
42008 90.1 0.2062615741  18.2
42010 32.3 0.07962962963 16.9
42011 29   0.0491087963  24.6
...
```

Detailed properties and methods documentation is very important: it is the unique thing which will make our methods really reusable in the future.

Then we can call our properties and methods from other programs if needed.

Example:

```
      80 zzTELPRINT'bk'⎕wi'GetMemberNames'
AnneB  AnneX  EricE  EricR      FredericN  JohnF  JohnL  RobertA  RobertG
AnneQ  AnneY  EricH  FredericD  FredericW  JohnJ  JohnS  RobertC

      'bk'⎕wi'GetMemberData' 'JohnF'
 42007 88.8 0.2114236111  17.5
 42012 55.4 0.147025463   15.7
 42015 41.6 0.1063310185  16.3
 42018 11.2 0.01994212963 23.4
 42028 27   0.05113425926 22
 42030 71.5 0.1163657407  25.6
```

Copyright © 2016 Eric Lescasse

```
 42031 33.5 0.06678240741 20.9
 42036 16.8 0.04635416667 15.1
 42038 63.6 0.1172453704  22.6
…
```

This is why it is extremely important to very much think ahead of time about our properties, methods and event names and syntax (when I say method hereafter, I mean: property or method).

In particular, we should think out each method syntax asking us the following questions:

- What arguments should I need for my method, so that I could usefully use it from outside my object
- What is the **minimum set** of arguments I need to make this method usable as an API from outside my object
- What is the best order for my method arguments?
- Should my method return a result?
- If my method returns a result, what should the result be?
- Have I designed my method as doing one task and only one task? If not, I should surely split my method into several simpler methods.
- Does my method deal with more than one of these stuff: **User Interface** stuff, **Data Layer** stuff, **Business Rules** stuff? If yes, I should split my method into as many methods, each one dealing with only one of these topics: User Interface, Data Access, Business Rules

In answering the above questions we should always try to favor “simplicity”. When we hesitate between two options, the simpler option is often the better.

So, when we develop application with zObjects, not only are we writing Windows applications, but also we are developing a framework API that can be used from outside of your objects.

Nothing should even prevent us from adding properties and methods in our object which are never called from inside our object itself, but that we can call as just API properties and methods.

For example, we can recognize that the **GetMemberData** method return dates and times in a format which is not what we would expect for an API. It would be better to get dates in **yyyymmdd** format and times in **hhmmss** format.

Again, in a real life application, I would have created a Business Layer with date and times conversion methods and would have used these methods wherever appropriate.

Copyright © 2016 Eric Lescasse

But since we did not do that, we can at least add a new method called **GetMember-RawData** that returns what we want:

```
:region-----------------------------------------GetMemberRawData
∆GetMemberRawData:
    ⍝∇ Retrieves the bike rides data for the specified member
    ⍝∇ Syntax:  data←'obj'⎕wi'GetMemberRawData'member
    ⍝∇ member: the name of a bike club member
    ⍝∇ data:  the Nx4 bike rides data matrix for this member
    ⍝∇        [;1]= dates in yyyymmdd format
    ⍝∇        [;2]= bike ride distance (in miles)
    ⍝∇        [;3]= bike ride time in hhmmss format
    ⍝∇        [;4]= bike ride average speed (in miles/hour)
    ⍝∇ Example:
    ⍝∇       'bk'⎕wi'GetMemberRawData' 'EricE'
    ⍝∇ 20150104 90.1 45701 18.2
    ⍝∇ 20150106 32.3 15440 16.9
    ⍝∇ 20150107 29   11043 24.6
    ⍝∇ ...
    t←⎕wi'*ΔΔtieno'                          ⍝ retrieve the tie number
    c←⎕fread t 1
    :if(↑⍴c)<e←c⍳⎕warg[2]
        ⎕wres←0 4⍴''
    :else
        m←24 60 60
        d←⎕fread t(10+e)
        d←d[⍋d[;1];]                         ⍝ sort matrix by increasing dates
        ⎕wres←d
    :endif
    :return
    :endregion
```

This **GetMemberRawData** method is never called from within the **uBikeClub** object, but can certainly be useful as an API we can call from outside **uBikeClub**:

```
      'bk'⎕wi'GetMemberRawData' 'EricE'
 20150104 90.1 45701 18.2
 20150106 32.3 15440 16.9
 20150107 29   11043 24.6
 20150114 66.7 30105 22.1
 20150119 45   14239 26.3
 …
```

Copyright © 2016 Eric Lescasse

The uBikeClub "finished" application code

Here is the code for our **uBikeClub** object:

```
    ∇ a uBikeClub b;c;d;e;f;g;h;m;n;p;r;s;t;u;v;⎕io
[1]   ⍝∇ a uBikeClub b -- The uBikeClub object is a Form (or MDIForm) derived from
       a zForm object
[2]   ⍝∇ a ←→ object name (may be omitted if ⎕wself is set)
[3]   ⍝∇ b ←→ 'property'
[4]   ⍝∇    or 'property'value
[5]   ⍝∇    or 'Method'
[6]   ⍝∇    or 'Method'argument1 ... argumentN
[7]   ⍝∇ Requires: (F) zObject
[8]   ⍝∇ zObjects v2.3
[9]   ⍝∇ Copyright(c)Lescasse Consulting 2013-2016
[10]  ⍝∇ ELE18jan16
[11]
[12]  :region-----------------------------------------Action
[13]  ⎕io←1
[14]  :if ⎕monadic ⋄ a←⎕wself ⋄ :endif
[15]  :if b≡'Action' ⋄ →('?'=↑b←↑⎕warg)ρ⍙⍙Help ⋄ :endif
[16]  →(8≠''ρ⎕idloc'⍙',b)ρ⍙⍙Inherit ⋄ →⍎'⍙',b
[17]  :endregion
[18]
[19]  :region-------------------------------------Constructor
[20]  ⍙New:
[21]      ⍝∇ Create a new instance of uBikeClub
[22]      ⍝∇ Example:
[23]      ⍝∇       'zt'⎕wi'*Create' 'uBikeClub'
[24]      ←a⎕wi'*Create' 'zForm'('*scale'5)('*caption' 'Bike Club')('margins'5 5)
             ('gaps'5 5)
[25]      ←a⎕wi'*onAction'(∈'uBikeClub' 'zForm' 'zObject',¨⊂'"Action"',⎕tcnl)
[26]      ←zzAddLink a
[27]      ←a⎕wi'*.bn1.Create' 'znetButton32'('wherelc'0 0'⋄' '⋄'0 ¯1)
                  ('*zImage' 'AddDatabaseRecord32np')
[28]      ←a⎕wi'*.bn2.Create' 'znetButton32'('wherelc' '=' '>>' '=' '='0 ¯1)
                  ('*zImage' 'DeleteDatabaseRecord32np')
[29]      ←a⎕wi'*.l1.Create' 'zLabel'('*border'1)('*caption' '')('wherelc'40'>'250)
                  ('anchor' 'ltb')
[30]      ←a⎕wi'*.l2.Create' 'zLabel'('*border'1)('*caption' '')
                  ('wherelc' '=' '>' '=' '>')('anchor' 'ltbr')
[31]      ←a⎕wi'*.sp1.Create' 'zSplitter'('Split' 'l1' 'l2')
[32]      ←a⎕wi'*.l1.cbMembers.Create' 'zCombo'('*style'2 4 8 16 128 256)
                  ('wherelc'0500'>')('anchor' 'ltr')('caption' 'Member')
[33]      ←a⎕wi'*.l1.grid.Create' 'zGrid'('*border'1)('wherelc' '>'¯1'>>' '>>'0 0 2 2)
                  ('anchor' 'ltrb')
[34]      ←a⎕wi'*.st.Create' 'zStatus'('panes'1 2 3'caps' 'num' 'scroll' '⎕ts'¯12)
[35]      ←a⎕wi'*.l2.fr.Create' 'zFrame'('*style'7)('*caption' '')
                  ('wherelc'0 0 42'>>')('anchor' 'ltr')
[36]      ←a⎕wi'*.l2.fr.bn21.Create' 'znetButton32'('wherelc'0 0'⋄' '⋄'0 ¯1)
                  ('*zImage' 'BarArgument_32x32')
[37]      ←a⎕wi'*.l2.fr.bn22.Create' 'znetButton32'('wherelc' '=' '>>' '=' '='0 ¯1)
                  ('*zImage' 'BarSeries_32x32')
[38]      ←a⎕wi'*.l2.lcc.Create' 'zLCChart'('wherelc'42 0'>>' '>>')('anchor' 'lrtb')
[39]      ⍝ Events
[40]      ←a⎕wi'AddHandler' '*onShow' 'uBikeClub"onShow"'
[41]      ←'cbMembers'⎕wi'AddHandler' '*onClick' 'uBikeClub"cbMembers_onClick"'
[42]      ←'bn1'⎕wi'AddHandler' '*onXClick' 'uBikeClub"bn1_onXClick"'
```

```
[43]      ←'bn2'⎕wi'AddHandler' '*onXClick' 'uBikeClub"bn2_onXClick"'
[44]      ←'bn21'⎕wi'AddHandler' '*onXClick' 'uBikeClub"bn21_onXClick"'
[45]      ←'bn22'⎕wi'AddHandler' '*onXClick' 'uBikeClub"bn22_onXClick"'
[46]      :return
[47]      :endregion
[48]
[49]  :region----------------------------------------------class
[50]  ∆class:
[51]      ⎕wres←'uBikeClub'
[52]      :return
[53]      :endregion
[54]
[55]  :region----------------------------------------------help
[56]  ∆help:
[57]      ⍝∇ ←'zt'⎕wi'*Create' 'uBikeClub'
[58]      ⎕wres←⎕wi'GetHelp' 'help'
[59]      :return
[60]      :endregion
[61]
[62]  :region----------------------------------------------AddGridRow
[63]  ∆AddGridRow:
[64]      ⍝∇ Displays a form allowing to enter data for a new row added to the grid
[65]      ⍝∇ Syntax:  'obj'⎕wi'AddGridRow'
[66]
[67]      ⍝ Display the dialog box and wait on it
[68]      ←'grf'⎕wi'*Create' 'uGridRowForm' 'Show'('topmost'1)('Center-
On'(⎕wi'*:self'))'*Wait'
[69]
[70]      :if'ok'≡'grf'⎕wi'dialogresult'            ⍝ if user clicked the OK button
[71]          ⍝ Get data entered in the dialog box
[72]          (d e f g)←'grf'⎕wi'dialogvalues'      ⍝ d=date, e=dist, f=time, g=speed
[73]          m←24 60 60
[74]          d←⍎2+⎕wi'DateBase'(100⊥3↑d)
[75]          f←(m⊥3↑3↓f)÷×/m
[76]          h←'dd-MMM-yy' '#.#' 'H:mm:ss' '#.#'
[77]          ⍝ Update the grid
[78]          (r c)←(⊂'grid')⎕wi¨'*xRows' '*xCols'  ⍝ get the new grid nb of rows and
cols
[79]          ←'grid'⎕wi'*XInsertRows'(r+.5)        ⍝ append a new row to the grid
[80]          ←'grid'⎕wi'*xValueType'(r+1)(⍳c)(1 4⍴3 1 3 1) ⍝ 3(=date) 1(=number)
[81]          ←'grid'⎕wi'*xRow'(r+1)                ⍝ move to the new row
[82]          ←'grid'⎕wi'*xValue'(r+1)(⍳c)(d e f g) ⍝ populate the new row
[83]          ←'grid'⎕wi'*xFormat'(r+1)(⍳c)h        ⍝ format cells
[84]          ←'grid'⎕wi'*xRowSize'(r+1)17          ⍝ set the new row height
[85]          ←'grid'⎕wi'*XRedraw'                  ⍝ refresh the grid
[86]          ⍝ Save the grid data to the database
[87]          v←'grid'⎕wi'*xValue'(⍳r)(⍳c)          ⍝ read the entire grid content
[88]          m←zzDEB'cbMembers'⎕wi'*text'
[89]          ←⎕wi':SetMemberData'm v
[90]      :endif
[91]      :return
[92]      :endregion
[93]
[94]  :region----------------------------------------------DeleteGridRow
[95]  ∆DeleteGridRow:
[96]      ⍝∇ Deletes the currently active grid row
[97]      ⍝∇ Syntax:  'obj'⎕wi'DeleteGridRow'
[98]      (r c)←(⊂'grid')⎕wi¨'*xRows' '*xCols'      ⍝ get the new grid nb of rows and
cols
```

Copyright © 2016 Eric Lescasse

```
[99]      d←'grid'⎕wi'*xRow'                          ⍝ currently active grid row
[100]     e←3↓∈,(⊂' - '),¨'grid'⎕wi'*xText'd(⍳c)
[101]     ←'grid'⎕wi'*xColorBack'd(⍳c)(256⊥0 255 255)
[102]     ←'grid'⎕wi'*XRedraw'd 1 1 4
[103]     f←zzYesNo('Delete row: ',(⍕e),'?')zzOKCaption(⎕wi'*:hwnd')
[104]     :if 1=f
[105]         ←'grid'⎕wi'*XDeleteRows'd               ⍝ delete current row
[106]         ←'grid'⎕wi'*XRedraw'                    ⍝ redraw the grid
[107]         v←'grid'⎕wi'*xValue'(⍳r)(⍳c)            ⍝ read the entire grid content
[108]         m←zzDEB'cbMembers'⎕wi'*text'
[109]         ←⎕wi':SetMemberData'm v
[110]     :else
[111]         ←'grid'⎕wi'*xColorBack'd(⍳c)(256⊥255 255 255)
[112]         ←'grid'⎕wi'*XRedraw'd 1 1 4
[113]     :endif
[114]     :return
[115]     :endregion
[116]
[117] :region-----------------------------------------DrawMilesPerMonthBarChart
[118] ∆DrawMilesPerMonthBarChart:
[119]     ⍝∇ Draws a miles per month bar chart
[120]     ⍝∇ Syntax:  data←'obj'⎕wi'DrawMilesPerMonthBarChart'matrix
[121]     ⍝∇ matrix: the Nx4 member bike rides matrix
[122]     ⍝ Prepare data for the chart
[123]     m←2⊃⎕warg
[124]     d←⎕wi'DateRep'(m[;1]-2)                     ⍝ convert dates to yyyymmdd
[125]     g←∪n←⌊d÷100                                 ⍝ yyyymm months (sorted)
[126]     f←+/¨n⊂m[;2]                                ⍝ total distance per month
[127]     h←'Jan' 'Feb' 'Mar' 'Apr' 'May' 'Jun'
[128]     h,←'Jul' 'Aug' 'Sep' 'Oct' 'Nov' 'Dec'
[129]     h←h[1+100|g-1]                              ⍝ months names
[130]     p←'cbMembers'⎕wi'*text'                     ⍝ member name
[131]     ⍝ Draw the chart
[132]     ←'lcc'⎕wi'*zReset'432 324
[133]     ←'lcc'⎕wi'*zHeading' 'Total Miles per Month'
[134]     ←'lcc'⎕wi'*zSubHeading'(p,' (2015)')
[135]     ←'lcc'⎕wi'*zYCaption' 'Miles'
[136]     ←'lcc'⎕wi'*zBarChartStyle' (BarChartStyles.ValueTags)
[137]     ←'lcc'⎕wi'*zValueTagStyle' (ValueTagStyles.Vertical+
                ValueTagStyles.Inside+ValueTagStyles.RecolorOutside)
[138]     ←'lcc'⎕wi'*zSetValueFont' 'Arial' 12 FontStyle.Bold Color.Black
[139]     ←'lcc'⎕wi'*zSetColors' Color.Red
[140]     ←'lcc'⎕wi'*zSetFillStyles' FillStyle.GradientBottom
[141]     ←'lcc'⎕wi'*zXAxisStyle' (XAxisStyles.GridLines)
[142]     ←'lcc'⎕wi'*zSetHeadingFont' 'Helvetica' 24 FontStyle.Bold
[143]     ←'lcc'⎕wi'*zSetSubheadingFont' 'Helvetica' 16 FontStyle.Bold
[144]     ←'lcc'⎕wi'*zSetXLabelFont' 'Calibri' 10
[145]     ←'lcc'⎕wi'*zSetYLabelFont' 'Calibri' 10
[146]     ←'lcc'⎕wi'*zSetYCaptionFont' 'Calibri' 14
[147]     ←'lcc'⎕wi'*zMarginLeft'50
[148]     ←'lcc'⎕wi'*zMarginTop'60
[149]     ←'lcc'⎕wi'*zSetXLabels'(⍕¨h)
[150]     ←'lcc'⎕wi'*zSetYRange'0 1200
[151]     ←'lcc'⎕wi'*zDrawBarChart'f
[152]     ←'lcc'⎕wi'*zDrawChart'
[153]     :return
[154]     :endregion
[155]
[156] :region-----------------------------------------DrawMilesPerMonthSimpleBarChart
```

Copyright © 2016 Eric Lescasse

```
[157] ∆DrawMilesPerMonthSimpleBarChart:
[158]     ⍝∇ Draws a miles per month bar chart
[159]     ⍝∇ Syntax:  data←'obj'⎕wi'DrawMilesPerMonthSimpleBarChart'matrix
[160]     ⍝∇ matrix: the Nx4 member bike rides matrix
[161]     ⍝ Prepare data for the chart
[162]     m←2⊃⎕warg
[163]     d←⎕wi'DateRep'(m[;1]-2)                    ⍝ convert dates to yyyymmdd
[164]     g←∪n←⌊d÷100                                ⍝ yyyymm months (sorted)
[165]     f←+/¨n⊂m[;2]                               ⍝ total distance per month
[166]     h←'Jan' 'Feb' 'Mar' 'Apr' 'May' 'Jun'
[167]     h,←'Jul' 'Aug' 'Sep' 'Oct' 'Nov' 'Dec'
[168]     h←h[1+100|g-1]                             ⍝ month names
[169]     p←'cbMembers'⎕wi'*text'                    ⍝ member name
[170]     ⍝ Draw the chart
[171]     ←'lcc'⎕wi'*zReset'432 324
[172]     ←'lcc'⎕wi'*zHeading' 'Total Miles per Month'
[173]     ←'lcc'⎕wi'*zSubHeading'(p,' (2015)')
[174]     ←'lcc'⎕wi'*zYCaption' 'Miles'
[175]     ←'lcc'⎕wi'*zDrawBarChart'f
[176]     ←'lcc'⎕wi'*zDrawChart'
[177]     :return
[178]     :endregion
[179]
[180] :region-------------------------------------------GetMemberData
[181] ∆GetMemberData:
[182]     ⍝∇ Retrieves the bike rides data for the specified member
[183]     ⍝∇ Syntax:  data←'obj'⎕wi'GetMemberData'member
[184]     ⍝∇ member: the name of a bike club member
[185]     ⍝∇ data:  the Nx4 bike rides data matrix for this member
[186]     ⍝∇        [;1]= dates in number of days since 19000101
[187]     ⍝∇        [;2]= bike ride distance (in miles)
[188]     ⍝∇        [;3]= bike ride time in % of 24 hours
[189]     ⍝∇        [;4]= bike ride average speed (in miles/hour)
[190]     ⍝∇ Example:
[191]     ⍝∇       'bk'⎕wi'GetMemberData' 'EricE'
[192]     ⍝∇ 42008 90.1 0.2062615741  18.2
[193]     ⍝∇ 42010 32.3 0.07962962963 16.9
[194]     ⍝∇ 42011 29   0.0491087963  24.6
[195]     ⍝∇ ...
[196]     t←⎕wi'*ΔΔtieno'                            ⍝ retrieve the tie number
[197]     c←⎕fread t 1
[198]     :if(↑⍴c)<e←c⍳⎕warg[2]
[199]         ⎕wres←0 4⍴''
[200]     :else
[201]         m←24 60 60
[202]         d←⎕fread t(10+e)
[203]         d←d[⍋d[;1];]                           ⍝ sort matrix by increasing dates
[204]         d[;1]←2+⎕wi'DateBase'(d[;1])           ⍝ convert yyyymmdd to grid dates
[205]         d[;3]←(m⊥100 100 100⊤d[;3])÷×/m        ⍝ convert hhmmss times to grid
times
[206]         ⎕wres←d
[207]     :endif
[208]     :return
[209]     :endregion
[210]
[211] :region-------------------------------------------GetMemberNames
[212] ∆GetMemberNames:
[213]     ⍝∇ Retrieves a sorted list of member names
[214]     ⍝∇ Syntax:  names←'obj'⎕wi'GetMemberNames'
```

Copyright © 2016 Eric Lescasse

```
[215]     ⍝∇ names: sorted nested vector of all member names
[216]     t←⎕wi'*ΔΔtieno'                          ⍝ retrieve the tie number
[217]     c←⎕fread t 1
[218]     ⎕wres←c[⎕av⍋⊃c]
[219]     :return
[220]     :endregion
[221]
[222] :region-----------------------------------------GetMemberRawData
[223] ∆GetMemberRawData:
[224]     ⍝∇ Retrieves the bike rides data for the specified member
[225]     ⍝∇ Syntax:  data←'obj'⎕wi'GetMemberRawData'member
[226]     ⍝∇ member: the name of a bike club member
[227]     ⍝∇ data:  the Nx4 bike rides data matrix for this member
[228]     ⍝∇         [;1]= dates in yyyymmdd format
[229]     ⍝∇         [;2]= bike ride distance (in miles)
[230]     ⍝∇         [;3]= bike ride time in hhmmss format
[231]     ⍝∇         [;4]= bike ride average speed (in miles/hour)
[232]     ⍝∇ Example:
[233]     ⍝∇        'bk'⎕wi'GetMemberRawData' 'EricE'
[234]     ⍝∇ ...
[235]     t←⎕wi'*ΔΔtieno'                          ⍝ retrieve the tie number
[236]     c←⎕fread t 1
[237]     :if(↑⍴c)<e←c⍳⎕warg[2]
[238]         ⎕wres←0 4⍴''
[239]     :else
[240]         m←24 60 60
[241]         d←⎕fread t(10+e)
[242]         d←d[⍋d[;1];]                         ⍝ sort matrix by increasing dates
[243]         ⎕wres←d
[244]     :endif
[245]     :return
[246]     :endregion
[247]
[248] :region-----------------------------------------OpenDatabase
[249] ∆OpenDatabase:
[250]     ⍝∇ Opens the database or creates it if it does not exist
[251]     ⍝∇ Syntax:  'obj'⎕wi'OpenDatabase'
[252]     f←⎕wsid
[253]     :try
[254]         f⎕fstie t←1+⌈/⎕xfnums,⎕fnums,0
[255]     :catchall
[256]         ⍝ Database does not exist: generate one
[257]         c←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
[258]         m←↑⍴n←'Anne' 'Eric' 'Frederic' 'John' 'Robert'
[259]         m←↑⍴n←↑,/(⊂¨n)∘.,¨(?m⍴4)↑¨⎕split c[m 4⍴26?26] ⍝ names
[260]         p←50+?m⍴100                          ⍝ nb of rides per member
[261]         d←⎕wi'DateRep'((⎕wi'DateBase'20150101)+0,⍳364)  ⍝ all days in 2015
[262]         e←(⊂¨p?¨365)⌷¨⊂d                     ⍝ nested vec of ride dates
[263]         g←(∈p↑¨1)⎕penclose.1×100+?(+/p)⍴900   ⍝ nested vec of distances in mi
[264]         s←(∈p↑¨1)⎕penclose.1×150+?(+/p)⍴120   ⍝ nested vec of speeds in mi/hour
[265]         u←100⊥¨(⊂24 60 60)⊤¨⌊3600×g÷s         ⍝ nested vec of times (hhmmss)
[266]         m←⊃¨e,⊂¨g,⊂¨u,⊂¨s                     ⍝ nested vec of matrices
[267]         f⎕fcreate t←1+⌈/⎕xfnums,⎕fnums,0      ⍝ create the database
[268]         ←n⎕fappend t                         ⍝ save names in component 1
[269]         ←(⊂'')⎕fappend¨9⍴t                    ⍝ init comps 2-10
[270]         ←m⎕fappend¨t                         ⍝ save all member matrices
[271]     :endtry
[272]     ←⎕wi'*ΔΔtieno't                          ⍝ save tie number
```

Copyright © 2016 Eric Lescasse

```
[273]     :return
[274]     :endregion
[275]
[276] :region----------------------------------------PopulateGrid
[277] ∆PopulateGrid:
[278]     ⍝∇ Populates the Grid
[279]     ⍝∇ Syntax:  'obj'⎕wi'PopulateGrid'matrix
[280]     ⍝∇ matrix: the Nx4 member bike rides matrix
[281]     (r c)←⍴m←2⊃⎕warg
[282]     p←'Date' 'Miles' 'Time' 'Speed'
[283]     f←'dd-MMM-yy' '#.#' 'H:mm:ss' '#.#'
[284]     ←'grid'⎕wi'*xRows'0                          ⍝ first empty the grid
[285]     ←'grid'⎕wi'*XRedraw'
[286]     ←'grid'⎕wi'*xHeadCols'0
[287]     ←'grid'⎕wi'*xRows'r
[288]     ←'grid'⎕wi'*xCols'c
[289]     ←'grid'⎕wi'*xText'¯1(⍳c)p
[290]     ←'grid'⎕wi'*xValue'(⍳r)(2 4)(m[;2 4])
[291]     ←'grid'⎕wi'*xDate'(⍳r)(1 3)(m[;1 3])
[292]     ←'grid'⎕wi'*xFormat'(⍳r)(⍳c)(r c⍴f)
[293]     ←'grid'⎕wi'*xRowSize'(⍳r)17
[294]     ←'grid'⎕wi'*xFitCol'(⍳c)
[295]     :return
[296]     :endregion
[297]
[298] :region----------------------------------------PopulateMembersCombo
[299] ∆PopulateMembersCombo:
[300]     ⍝∇ Populates the Members combo box
[301]     ⍝∇ Syntax:  'obj'⎕wi'PopulateMembersCombo'
[302]     c←⎕wi'GetMemberNames'
[303]     ←'cbMembers'⎕wi'list'c
[304]     :return
[305]     :endregion
[306]
[307] :region----------------------------------------SetMemberData
[308] ∆SetMemberData:
[309]     ⍝∇ Saves the bike rides data for the specified member to the database
[310]     ⍝∇ Syntax:  'obj'⎕wi'SetMemberData'member matrix
[311]     ⍝∇ member: the name of a bike club member
[312]     ⍝∇ matrix:  the Nx4 bike rides data matrix for this member (in grid format)
[313]     t←⎕wi'*ΔΔtieno'                          ⍝ retrieve the tie number
[314]     c←⎕fread t 1
[315]     :if(↑⍴c)<e←c⍳⎕warg[2]
[316]         ⍝ Member not found: do nothing
[317]     :else
[318]         m←24 60 60
[319]         d←3⊃⎕warg
[320]         d[;1]←¯2+⎕wi'DateRep'(d[;1])          ⍝ convert nb of days to yyyymmdd
[321]         d[;3]←100⊥⍉Td[;3]××/m                 ⍝ convert decimal times to hhmmss
[322]         d⎕freplace t(10+e)
[323]     :endif
[324]     :return
[325]     :endregion
[326]
[327] :region----------------------------------------UpdateStatus
[328] ∆UpdateStatus:
[329]     ⍝∇ Updates the status bar with bike ride statistics
[330]     ⍝∇ Syntax:  'obj'⎕wi'UpdateStatus'matrix
[331]     ⍝∇ matrix: the Nx4 member bike rides matrix
```

```
[332]     m←2⊃⎕warg
[333]     n←↑⍴m                                    ⍝ total number of rides
[334]     p←+/m[;2]                                ⍝ total number of miles
[335]     s←p÷⍨+/×/m[;2 4]                         ⍝ annual average speed
[336]     s←.1×⌊.5+10×s                            ⍝ round avg speed to 1 decimal
[337]     ←'st'⎕wi'*SetStatus'1((⍕n),' rides')
[338]     ←'st'⎕wi'*SetStatus'2((⍕p),' miles')
[339]     ←'st'⎕wi'*SetStatus'3((⍕s),' mi/hr')
[340]     :return
[341]     :endregion
[342]
[343]
[344] :region----------------------------------------onShow
[345] ∆onShow:
[346]     ←⎕wi'OpenDatabase'
[347]     ←⎕wi'PopulateMembersCombo'
[348]     :return
[349]     :endregion
[350] :region----------------------------------------bn1_onXClick
[351] ∆bn1_onXClick:
[352]     :if~0∈⍴m←zzDEB'cbMembers'⎕wi'*text'
[353]         d←⎕wi':GetMemberData'm
[354]         ←⎕wi':AddGridRow'
[355]     :endif
[356]     :return
[357]     :endregion
[358] :region----------------------------------------bn2_onXClick
[359] ∆bn2_onXClick:
[360]     :if~0∈⍴m←zzDEB'cbMembers'⎕wi'*text'
[361]         d←⎕wi':GetMemberData'm
[362]         ←⎕wi':DeleteGridRow'
[363]     :endif
[364]     :return
[365]     :endregion
[366] :region----------------------------------------bn21_onXClick
[367] ∆bn21_onXClick:
[368]     :if~0∈⍴m←zzDEB'cbMembers'⎕wi'*text'
[369]         d←⎕wi':GetMemberData'm
[370]         ←⎕wi':DrawMilesPerMonthSimpleBarChart'd
[371]         ←⎕wi'*:ΔΔcharttype'1
[372]     :endif
[373]     :return
[374]     :endregion
[375] :region----------------------------------------bn22_onXClick
[376] ∆bn22_onXClick:
[377]     :if~0∈⍴m←zzDEB'cbMembers'⎕wi'*text'
[378]         d←⎕wi':GetMemberData'm
[379]         ←⎕wi':DrawMilesPerMonthBarChart'd
[380]         ←⎕wi'*:ΔΔcharttype'2
[381]     :endif
[382]     :return
[383]     :endregion
[384] :region----------------------------------------cbMembers_onClick
[385] ∆cbMembers_onClick:
[386]     m←zzDEB⎕wi'*text'
[387]     d←⎕wi':GetMemberData'm
[388]     ←⎕wi':PopulateGrid'd
[389]     ←⎕wi':UpdateStatus'd
[390]     :select 1⌈⎕wi'*:ΔΔcharttype'
```

Copyright © 2016 Eric Lescasse

```
[391]     :case 1 ⋄ ←⎕wi':DrawMilesPerMonthSimpleBarChart'd
[392]     :case 2 ⋄ ←⎕wi':DrawMilesPerMonthBarChart'd
[393]     :endselect
[394]     :return
[395]     :endregion
[396]
[397] :region-----------------------------------------Internal
[398] ∆∆Help:
[399]     ⎕wres←⎕wi'GetHelp'(('?'=↑b)↓b)
[400]     :return
[401] ∆∆Inherit:
[402]     ⎕wres←⎕wi'*'
[403]     :endregion
[404]
    ∇
```

The zEncrypt object

Introduction

The **zEncrypt** object is a .Net Framework based object that allows you to encrypt or decrypt files or texts.

The zEncrypt API

The **zEncrypt** object contains 1 property:

```
zEncryptionKey
```

and 4 methods:

```
zDecryptFile
zEncryptFile
zDecryptText
zEncryptText
```

How to use the zEncrypt object

First, create an instance of the **LC.zObjects.zEncrypt** class:

```
      ←'enc'⎕wi'*Create' 'LC.zObjects.zEncrypt'
```

zEncryptText and zDecryptText

Then, you can use the **zEncryptText** method to encrypt any text:

```
      aaa←'enc'⎕wi'*zEncryptText' 'This is some text to encrypt'
```

The result is a string containing the encrypted text:

```
      aaa
kTLsyOhXrrNWdrF-
BRiFTFvFzm1HNB9rPD8KNh0lVJxbE4fWgk+4oh1A3y/XPTBlSe+0UrOTbcUXmqzAQe3tv7g==
```

Use the **zDecryptText** method to decrypt it:

```
      'enc'⎕wi'*zDecryptText'aaa
This is some text to encrypt
```

Note that you can encrypt and decrypt APL characters:

```
      ⎕vr'zzOnNew'
    ∇ zzOnNew
[1]   :if(⊂⎕warg)∈'#'⎕wi'newclasses'
[2]       ⍎⎕warg,'"New"'
```

Copyright © 2016 Eric Lescasse

```
[3]    :end
    ∇

      aaa←'enc'⎕wi'*zEncryptText'(⎕vr'zzOnNew')

      aaa
ZIoY+B1ZhdRM1ZrYjN-
NuVNSkPJ/Hk7X10h0zdH/RaX2RcHvzz5nNZ/mPj6pTuRhJ1z0+xudj33+bvPhOthRgjidlkBYbDGISSY7E
7mPxVGNi-
qiY/kn4DLB/opGpLLv4Chji/w5U8izROk+zlYioSaFlb+mIdFHjkXMflZHxnpbcmd6EJP7F67GOsBsg
      1WtS/9JUv+CZzg3dji1jC3GeqqLkWjtM7M3HypMdDelmsghflnJhvc0juu8HBLswYwdW/

      'enc'⎕wi'*zDecryptText'aaa
    ∇ zzOnNew
[1]    :if(⊂⎕warg)∈'#'⎕wi'newclasses'
[2]        ⍎⎕warg,'"New"'
[3]    :end
    ∇
```

zEncryptionKey

Note that we have not had to set the encryption key: the zEncrypt class has a default encryption key.

However, it is recommended that you use your own encryption key: you can do so by setting the zEncryptionKey property prior to use the zEncrypt methods:

```
      'enc'⎕wi'*zEncryptionKey' 'MyOwnKey'
```

Note:
Be aware that attempting to decrypt text encrypted with another encryption key returns in an error:

```
      'enc'⎕wi'*zDecryptText'aaa
⎕WI ERROR: System.Core exception -2146233296 (0x80131430) Padding is invalid and cannot be removed.
      'enc'⎕wi'*zDecryptText'aaa
           ^
```

You should use your own encryption key and hide it as well as you can.

zEncryptFile and zDecryptFile

Finally, the **zEncryptFile** and **zDecryptFile** methods allow you to encrypt/decrypt entire text files.

Examples:

```
      'enc'⎕wi'*zEncryptFile' 'c:\temp\test.xml' 'c:\temp\testencrypted.xml'
      'enc'⎕wi'*zDecryptFile' 'c:\temp\testencrypted.xml' 'c:\temp\test.xml'
```

Copyright © 2016 Eric Lescasse

The zHtml object

Introduction

The **zHtml** object is a pure APL object (no C# involved here) that allows you to programmatically create well formed HTML documents.

You can then display those documents in a **znetWebBrowser** object in your APL+Win forms.

The zHtml object contains an API somewhat similar to the **znetXmlWriter** object, which allows you to create XML documents.

The zHtml API

The zHtml API is very simple:

```
      'hh'⎕wi'props'
Properties
----------
class  html           indentlevel  loremipsum
help   indentdefault  linelength   selfclosingtags

Methods
-------
EditWebPage  GetWebPage  Publish  StartEnd  Write
End          New         Start    Validate  WriteLine

Events
------
```

Among these properties and methods, you will mostly use **Start**, **StartEnd** and **End**.

How to use the zHtml object

Loading the zHtml object

```
      ]zload zHtml
```

Creating an instance of the zHtml object

```
      ←'hh'⎕wi'*Create' 'zHtml'
```

Understanding HTML

As we all know, an HTML Web page is made of a hierarchical series of tags and contents.

This means that, if you open a tag, then add content inside this tag, you must not forget to close the tag afterwords.

Copyright © 2016 Eric Lescasse

Additionally, a given tag, may include other tags which are children tags.

All these children tags must also be closed, but the parent tag must be closed after all the children tags are closed.

This is the way the HTML document tag hierarchy is built.

The zHtml object works exactly the same way.

You use the **Start** method to create a tag and add attributes and content to it.

You must use the **End** method to close this tag.

To simplify and reduce the amount of code, there is also another method called **StartEnd** which does everything all at once: create the tag, add its (optional) attributes, add its (optional) content and close the tag.

Creating an HTML document

```
      ←'hh'⎕wi'Start' 'html'
      ←'hh'⎕wi'End' 'html'
```

Checking the HTML code produced

At any time, you can check the HTML code produced by querying the **html** property:

```
      'hh'⎕wi'html'
<!DOCTYPE html>
<html>
</html>
```

Adding a head section

Since the html closing tag has already been generated, you cannot add the head section now. You have to restart all over again. For this reason, it is easier to progressively build a function.

So let's create a **zzhtml12** function as follows:

```
    ∇ r←zzhtml12
[1]
[2]   ⎕wself←'hh'⎕wi'*Create' 'zHtml'
[3]   ←⎕wi'Start' 'html'
[4]   ←⎕wi'Start' 'head'
[5]   ←⎕wi'End' 'head'
[6]   ←⎕wi'End' 'html'
[7]
[8]   r←⎕wi'html'
    ∇
```

The function returns the HTML produced. Let's try it:

```
      zzhtml12
<!DOCTYPE html>
```

```
<html>
<head>
</head>
</html>
```

Adding a title tag to the head section

We use the same principle to add the **title** tag:

```
    ∇ r←zzhtml13
[1]
[2]   ⎕wself←'hh'⎕wi'*Create' 'zHtml'
[3]   ←⎕wi'Start' 'html'
[4]   ←⎕wi'Start' 'head'
[5]   ←⎕wi'Start' 'title' '' 'My Web Page Title'
[6]   ←⎕wi'End' 'title'
[7]   ←⎕wi'End' 'head'
[8]   ←⎕wi'End' 'html'
[9]
[10]  r←⎕wi'html'
    ∇
```

Let's run function **zzhtml13**:

```
      zzhtml13
<!DOCTYPE html>
<html>
<head>
   <title>My Web Page Title
   </title>
</head>
</html>
```

Note that we could have used the StartEnd method instead since the title tag does not contain children tags. The zzhtml13 function would have read:

```
    ∇ r←zzhtml13
[1]
[2]   ⎕wself←'hh'⎕wi'*Create' 'zHtml'
[3]   ←⎕wi'Start' 'html'
[4]   ←⎕wi'Start' 'head'
[5]   ←⎕wi'StartEnd' 'title' '' 'My Web Page Title'
[6]   ←⎕wi'End' 'head'
[7]   ←⎕wi'End' 'html'
[8]
[9]   r←⎕wi'html'
    ∇
```

Let's run function **zzhtml13**:

```
      zzhtml13
<!DOCTYPE html>
<html>
<head>
   <title>My Web Page Title</title>
```

Copyright © 2016 Eric Lescasse

```
</head>
</html>
```

Note:
Note the subtle difference in the output: the </title> closing tag is now on the same line as the title text.

The Start and StartEnd methods

The Start and StartEnd methods accept 3 arguments:

[1] = the tag name (i.e. 'html', 'body', 'p', 'img', etc.)
[2] = the tag attributes or '' if there are no attributes
[3] = the tag content (generally some text)

The attributes must be specified as a nested vector of attribute-value pairs.

Example:

```
      ←⎕wi'StartEnd' 'img'(('src' 'http://www.lescasse.com/images/eric2011.png')
('height'240)('width'240)('alt' 'Eric Lescasse'))''
```

Fixing the head section

Knowing how to add attributes we can improve function zzhtml13 by adding a lang="en" attribute to the <head> section and adding a <meta charset="UTF8"/> tag as well:

```
    ∇ r←zzhtml13
[1]
[2]   ⎕wself←'hh'⎕wi'*Create' 'zHtml'
[3]   ←⎕wi'Start' 'html'
[4]   ←⎕wi'Start' 'head'('lang' 'en')
[5]   ←⎕wi'StartEnd' 'meta'('charset' 'UTF-8')
[6]   ←⎕wi'StartEnd' 'title' '' 'My Web Page Title'
[7]   ←⎕wi'End' 'head'
[8]   ←⎕wi'End' 'html'
[9]
[10]  r←⎕wi'html'
    ∇
```

Let's run function **zzhtml13** again:

```
      zzhtml13
<!DOCTYPE html>
<html>
<head lang="en">
   <meta charset="UTF-8"/>
   <title>My Web Page Title</title>
</head>
</html>
```

Copyright © 2016 Eric Lescasse

Setting the linelength property

You can decide how long you want your HTML output lines to be. The default is 100.

Let's change the linelength to 70:

```
      'hh'⎕wi'linelength'
100
      'hh'⎕wi'linelength'70
      'hh'⎕wi'linelength'
70
```

Adding a body tag and starting to populate it

Know what we learnt so far, let's start populating the body tag with a few simple tags.

```
    ∇ r←zzhtml14
[1]
[2]   ⎕wself←'hh'⎕wi'*Create' 'zHtml'('linelength'70)
[3]   ←⎕wi'Start' 'html'
[4]   ←⎕wi'Start' 'head'('lang' 'en')
[5]   ←⎕wi'StartEnd' 'meta'('charset' 'UTF-8')
[6]   ←⎕wi'StartEnd' 'title' '' 'My Web Page Title'
[7]   ←⎕wi'End' 'head'
[8]   ←⎕wi'Start' 'body'
[9]
[10]  ←⎕wi'StartEnd' 'h1' '' 'Title'
[11]  ←⎕wi'StartEnd' 'h2' '' 'Paragraph 1'
[12]  ←⎕wi'Start' 'div'
[13]  ←⎕wi'StartEnd' 'img'(('src' 'http://www.lescasse.com/im-
ages/eric2011.png')('height'240)('width'240)('alt' 'Eric Lescasse'))''
[14]  ←⎕wi'StartEnd' 'p' ''(200⍴⎕wi'loremipsum')
[15]  ←⎕wi'End' 'div'
[16]
[17]  ←⎕wi'End' 'body'
[18]  ←⎕wi'End' 'html'
[19]
[20]  r←⎕wi'html'
    ∇
```

Let's run function **zzhtml14**:

```
      zzhtml14
<!DOCTYPE html>
<html>
<head lang="en">
   <meta charset="UTF-8"/>
   <title>My Web Page Title</title>
</head>
<body>
   <h1>Title</h1>
   <h2>Paragraph 1</h2>
   <div>
```

Copyright © 2016 Eric Lescasse

```
      <img src="http://www.lescasse.com/images/eric2011.png" height="240"
width="240" alt="Eric Lescasse"/>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
         eiusmod tempor incididunt ut labore et dolore magna aliqua.
         Ut enim ad minim veniam, quis nostrud exercitation ullamco
         laboris nisi ut
      </p>
   </div>
</body>
</html>
```

As you can see the **img** tag has been automatically closed and the HTML is automatically indented.

Also, because the tag content spans several lines, the closing tag is placed on a new line to improved the HTML readability.

> **Note:**
> Note the **loremipsum** property which returns the well known **Lorem ipsum …** text used by the industry to display dummy text in sample web pages.

Setting the indentation default

The default indentation is set to 3 characters.

But you can change it by using the **indentdefault** property.

If you don't want any indentation, just set the **indentdefault** property to **0**.

Displaying the HTML code in a Web Browser

At this stage, I guess you become eager to see what the HTML code you generated looks like when displayed in a Web Browser.

You can use the **znetWebBrowser** object and its **zDocumentText** property to do so.

Let's write a small, easy to type, APL function called **zzweb** that accepts an HTML document text as its argument and displays it in an APL+Win form containing a C# znetWebBrowser object:

```
    ∇ zzweb h
[1]   ←'ff'⎕wi'*Create' 'zForm'('size'.7 .4)('*caption' 'zHtml Demo')
[2]   ←'ff'⎕wi'*.wbr.Create' 'znetWebBrowser'('whereIc'0 0'>>' '>>')('anchor' 'lrtb')
[3]   ←'wbr'⎕wi'*zScriptErrorsSuppressed'1
[4]   ←'wbr'⎕wi'*zBorderStyle'0
[5]   ←'wbr'⎕wi'*zDocumentText'h
[6]   ←'ff'⎕wi'CenterScreen'
[7]   ←'ff'⎕wi'*Show'
[8]   ←'ff'⎕wi'*onClose' '"wbr"⎕wi"*zDispose"'
    ∇
```

Let's display the result of zzhtml14:

```
zzweb zzhtml14
```

Adding a list

Since we are using APL functions to create the HTML, we can obviously use loops.

Let's create a list using the <ul> tag, with a :for loop:

```
    ∇ r←zzhtml15;i
[1]
[2]   ⎕wself←'hh'⎕wi'*Create' 'zHtml'('linelength'70)
[3]   ←⎕wi'Start' 'html'
[4]   ←⎕wi'Start' 'head'('lang' 'en')
[5]   ←⎕wi'StartEnd' 'meta'('charset' 'UTF-8')
[6]   ←⎕wi'StartEnd' 'title' '' 'My Web Page Title'
[7]   ←⎕wi'End' 'head'
[8]   ←⎕wi'Start' 'body'
[9]
[10]  ←⎕wi'StartEnd' 'h1' '' 'Title'
[11]  ←⎕wi'StartEnd' 'h2' '' 'Paragraph 1'
[12]  ←⎕wi'Start' 'div'
[13]  ←⎕wi'StartEnd' 'img'(('src' 'http://www.lescasse.com/im-
ages/eric2011.png')('height'240)('width'240)('alt' 'Eric Lescasse'))''
[14]  ←⎕wi'StartEnd' 'p' ''(200⍴⎕wi'loremipsum')
[15]  ←⎕wi'End' 'div'
[16]
[17]  ←⎕wi'StartEnd' 'h2' '' 'Paragraph 2'
[18]  ←⎕wi'Start' 'ul'
[19]  :for i :in⍳8
```

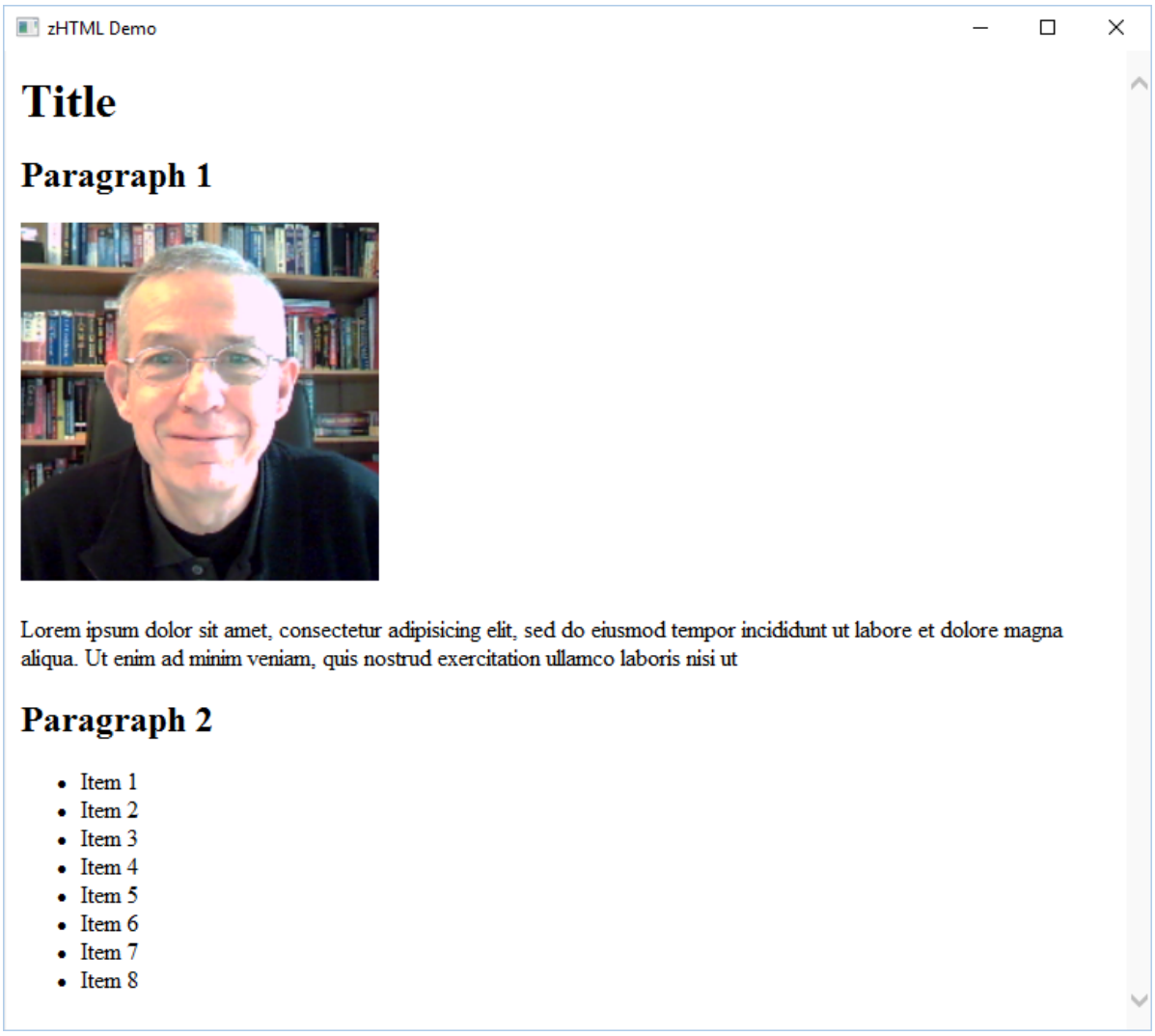
```
[20]      ←⎕wi'StartEnd' 'li' '' ('Item ',⍕i)
[21]    :endfor
[22]    ←⎕wi'End' 'ul'
[23]
[24]    ←⎕wi'End' 'body'
[25]    ←⎕wi'End' 'html'
[26]
[27]    r←⎕wi'html'
      ∇
```

Let's run function **zzhtml15**:

```
      zzhtml15
<!DOCTYPE html>
<html>
<head lang="en">
   <meta charset="UTF-8"/>
   <title>My Web Page Title</title>
</head>
<body>
   <h1>Title</h1>
   <h2>Paragraph 1</h2>
   <div>
      <img src="http://www.lescasse.com/images/eric2011.png" height="240"
width="240" alt="Eric Lescasse"/>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
         eiusmod tempor incididunt ut labore et dolore magna aliqua.
         Ut enim ad minim veniam, quis nostrud exercitation ullamco
         laboris nisi ut
      </p>
   </div>
   <h2>Paragraph 2</h2>
   <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
      <li>Item 4</li>
      <li>Item 5</li>
      <li>Item 6</li>
      <li>Item 7</li>
      <li>Item 8</li>
   </ul>
</body>
</html>
```

Let's look at the result in the **znetWebBrowser** object:

Copyright © 2016 Eric Lescasse



Things look good!

Adding a table

Very often you will want to display some APL matrix result in the znetWebBrowser object to create reports.

For that you need to create an HTML table, filling it with your APL matrix.

```
    ∇ r←zzhtml16;i;t;j
[1]
[2]   ⎕wself←'hh'⎕wi'*Create' 'zHtml'('linelength'70)
[3]   ←⎕wi'Start' 'html'
[4]   ←⎕wi'Start' 'head'('lang' 'en')
[5]   ←⎕wi'StartEnd' 'meta'('charset' 'UTF-8')
[6]   ←⎕wi'StartEnd' 'title' '' 'My Web Page Title'
[7]   ←⎕wi'End' 'head'
[8]   ←⎕wi'Start' 'body'
[9]
[10]  ←⎕wi'StartEnd' 'h1' '' 'Title'
[11]  ←⎕wi'StartEnd' 'h2' '' 'Paragraph 1'
```

Copyright © 2016 Eric Lescasse

```
[12]  ←⎕wi'Start' 'div'
[13]  ←⎕wi'StartEnd' 'img'(('src' 'http://www.lescasse.com/im-
ages/eric2011.png')('height'240)('width'240)('alt' 'Eric Lescasse'))''
[14]  ←⎕wi'StartEnd' 'p' ''(200⍴⎕wi'loremipsum')
[15]  ←⎕wi'End' 'div'
[16]
[17]  ←⎕wi'StartEnd' 'h2' '' 'Paragraph 2'
[18]  ←⎕wi'Start' 'ul'
[19]  :for i :in⍳8
[20]      ←⎕wi'StartEnd' 'li' '' ('Item ',⍕i)
[21]  :endfor
[22]  ←⎕wi'End' 'ul'
[23]
[24]  ←⎕wi'StartEnd' 'h2' '' 'Paragraph 3'
[25]  t←zzDEB¨,¨(⊂'CI12')⎕fmt¨?5 6⍴1000000
[26]  ←⎕wi'Start' 'table'
[27]  :for i :in⍳⍴t
[28]      ←⎕wi'Start' 'tr'
[29]      :for j :in⍳¯1↓⍴t
[30]          ←⎕wi'StartEnd' 'td' ''(j⊃t[i;])
[31]      :endfor
[32]      ←⎕wi'End' 'tr'
[33]  :endfor
[34]  ←⎕wi'End' 'table'
[35]
[36]  ←⎕wi'End' 'body'
[37]  ←⎕wi'End' 'html'
[38]
[39]  r←⎕wi'html'
    ∇
```

Let's run function **zzhtml16**:

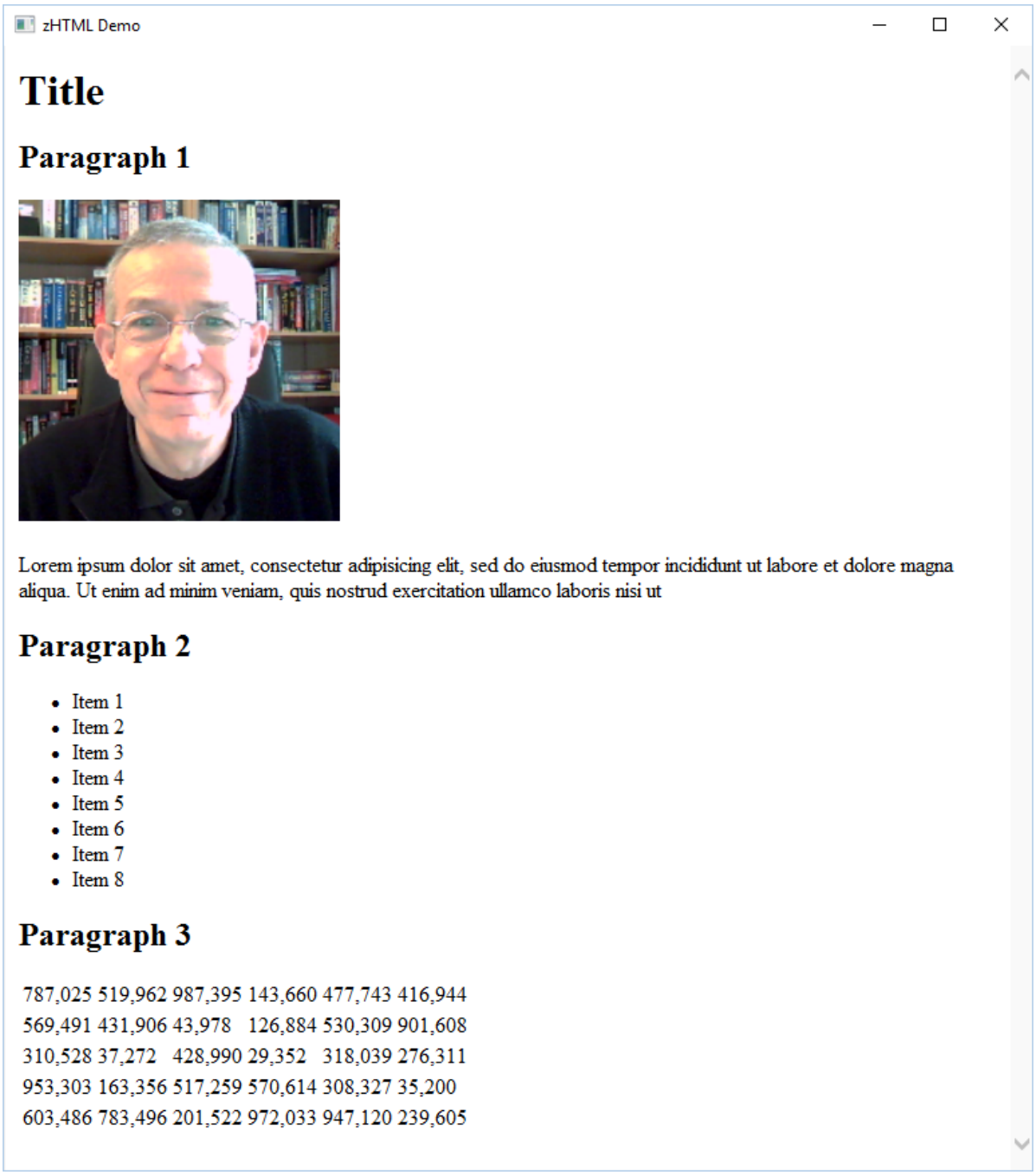
```
      zzhtml16
<!DOCTYPE html>
<html>
<head lang="en">
   <meta charset="UTF-8"/>
   <title>My Web Page Title</title>
</head>
<body>
   <h1>Title</h1>
   <h2>Paragraph 1</h2>
   <div>
      <img src="http://www.lescasse.com/images/eric2011.png" height="240"
width="240" alt="Eric Lescasse"/>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
         eiusmod tempor incididunt ut labore et dolore magna aliqua.
         Ut enim ad minim veniam, quis nostrud exercitation ullamco
         laboris nisi ut
      </p>
   </div>
   <h2>Paragraph 2</h2>
   <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
      <li>Item 4</li>
```

Copyright © 2016 Eric Lescasse

```
      <li>Item 5</li>
      <li>Item 6</li>
      <li>Item 7</li>
      <li>Item 8</li>
   </ul>
   <h2>Paragraph 3</h2>
   <table>
      <tr>
         <td>515,010</td>
         <td>762,732</td>
         <td>220,617</td>
         <td>909,765</td>
         <td>410,736</td>
         <td>231,686</td>
      </tr>
      <tr>
         <td>938,555</td>
         <td>287,552</td>
         <td>870,337</td>
         <td>745,508</td>
         <td>744,013</td>
         <td>622,252</td>
      </tr>
      <tr>
         <td>186,083</td>
         <td>485,238</td>
         <td>389,229</td>
         <td>759,834</td>
         <td>514,108</td>
         <td>612,102</td>
      </tr>
      <tr>
         <td>582,667</td>
         <td>880,426</td>
         <td>304,753</td>
         <td>971,745</td>
         <td>111,457</td>
         <td>249,520</td>
      </tr>
      <tr>
         <td>681,376</td>
         <td>874,048</td>
         <td>109,564</td>
         <td>426,556</td>
         <td>118,713</td>
         <td>205,759</td>
      </tr>
   </table>
</body>
</html>
```

Displaying the result in a **znetWebBrowser** object, we get:

Copyright © 2016 Eric Lescasse



Well! The table worked ok, but does not look so nice.

The whole page is ok but, as a matter of fact could be enhanced with some CSS.

Enhancing our Web page with CSS

Let's add some CSS in the **<head>** section of our Web page to make it prettier to look at.

Copyright © 2016 Eric Lescasse

```
    ∇ r←zzhtml17;i;j;t
[1]
[2]   ⎕wself←'hh'⎕wi'*Create' 'zHtml'('linelength'70)
[3]   ←⎕wi'Start' 'html'
[4]   ←⎕wi'Start' 'head'('lang' 'en')
[5]   ←⎕wi'StartEnd' 'meta'('charset' 'UTF-8')
[6]   ←⎕wi'StartEnd' 'title' '' 'My Web Page Title'
[7]   ←⎕wi'WriteLine' '<style type="text/css">'
[8]   ←⎕wi'WriteLine' '   body{font-family:Calibri;font-size:14px;}'
[9]   ←⎕wi'WriteLine' '   h1{font-family:Arial;font-size:32px;color:navy;}'
[10]  ←⎕wi'WriteLine' '   h2{font-family:Arial;font-size:24px;color:navy;}'
[11]  ←⎕wi'WriteLine' '   img{border:1px solid #333;padding:3px;float:right;}'
[12]  ←⎕wi'WriteLine' '   table{border-collapse:collapse;}'
[13]  ←⎕wi'WriteLine' '   tr.odd{background-color:#FFCBA9;}'
[14]  ←⎕wi'WriteLine' '   th,td{border:1px solid #aaa;padding:1px 5px 1px
      25px;text-align:right;}'
[15]  ←⎕wi'WriteLine' '   .imagecaption{float:right;margin:0 30px 10px 30px;dis
      play:inline;}'
[16]  ←⎕wi'WriteLine' '</style>'
[17]  ←⎕wi'End' 'head'
[18]  ←⎕wi'Start' 'body'
[19]
[20]  ←⎕wi'StartEnd' 'h1' '' 'Title'
[21]  ←⎕wi'StartEnd' 'h2' '' 'Paragraph 1'
[22]  ←⎕wi'Start' 'div'
[23]  ←⎕wi'StartEnd' 'img'(('src' 'http://www.lescasse.com/im-
ages/eric2011.png')('height'240)('width'240)('alt' 'Eric Lescasse'))''
[24]  ←⎕wi'StartEnd' 'p' ''(2000⍴⎕wi'loremipsum')
[25]  ←⎕wi'End' 'div'
[26]
[27]  ←⎕wi'StartEnd' 'h2' '' 'Paragraph 2'
[28]  ←⎕wi'Start' 'ul'
[29]  :for i :in⍳8
[30]      ←⎕wi'StartEnd' 'li' '' ('Item ',⍕i)
[31]  :endfor
[32]  ←⎕wi'End' 'ul'
[33]
[34]  ←⎕wi'StartEnd' 'h2' '' 'Paragraph 3'
[35]  t←zzDEB¨,¨(⊂'CI12')⎕fmt¨?5 10⍴1000000
[36]  ←⎕wi'Start' 'table'
[37]  :for i :in⍳⍴t
[38]      ←⎕wi'Start' 'tr',(0≠2|i)/⊂('class' 'odd')
[39]      :for j :in¯1↓⍳⍴t
[40]          ←⎕wi'StartEnd' 'td' ''(j⊃t[i;])
[41]      :endfor
[42]      ←⎕wi'End' 'tr'
[43]  :endfor
[44]  ←⎕wi'End' 'table'
[45]
[46]  ←⎕wi'End' 'body'
[47]  ←⎕wi'End' 'html'
[48]
[49]  r←⎕wi'html'
    ∇
```

This time we have used the WriteLine method to directly append lines at the end of the html code.

Copyright © 2016 Eric Lescasse

Note that we have increased the length of the **lorem ipsum** text we display this time to **2000** characters and increased the number of columns of the table to **10**.

Let's run function **zzhtml17**:

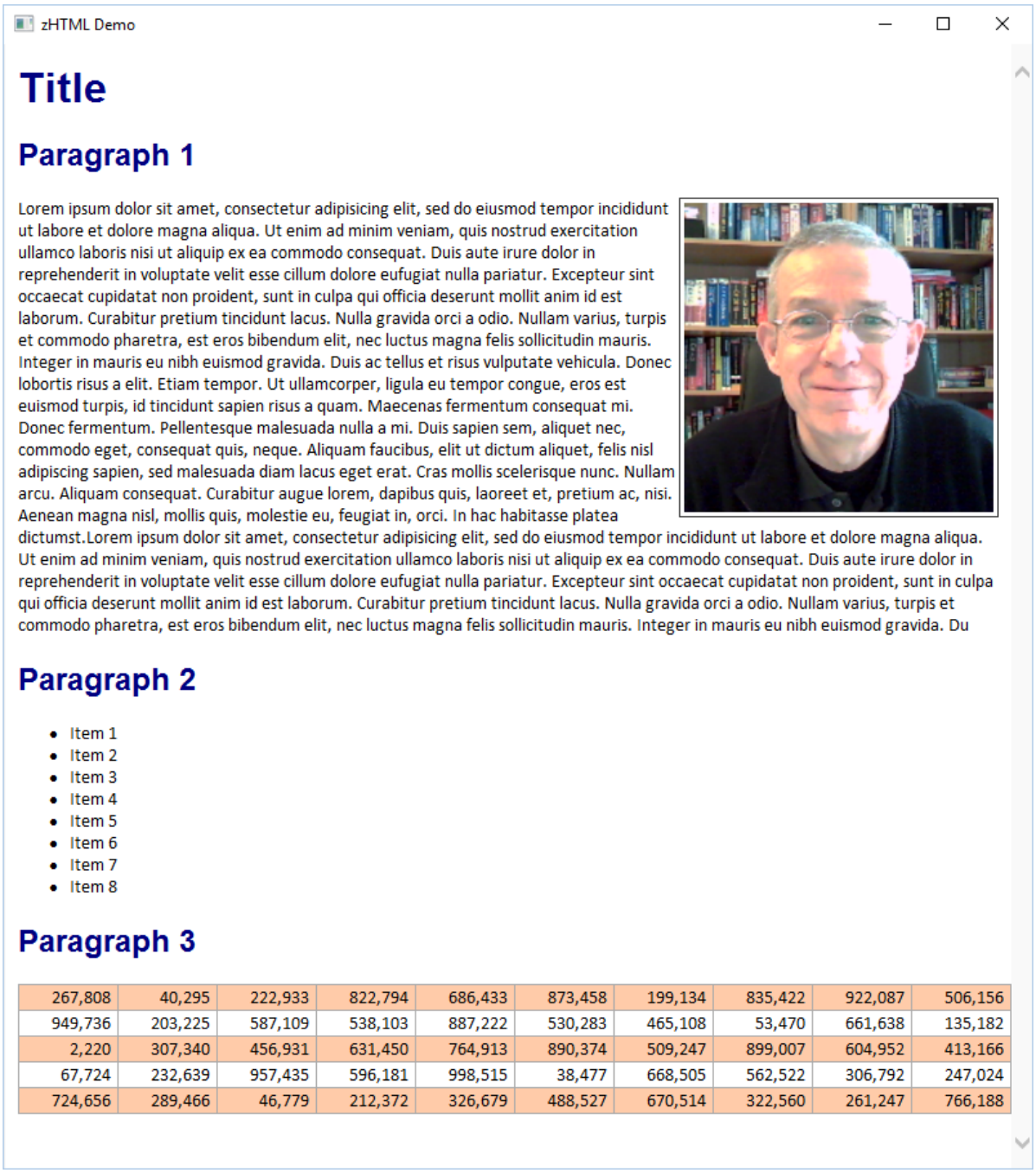
```
      zzhtml17
<!DOCTYPE html>
<html>
<head lang="en">
   <meta charset="UTF-8"/>
   <title>My Web Page Title</title>
   <style type="text/css">
      body{font-family:Calibri;font-size:14px;}
      h1{font-family:Arial;font-size:32px;color:navy;}
      h2{font-family:Arial;font-size:24px;color:navy;}
      img{border:1px solid #333;padding:3px;float:right;}
      table{border-collapse:collapse;}
      tr.odd{background-color:#FFCBA9;}
      th,td{border:1px solid #aaa;padding:1px 5px 1px 25px;text-align:right;}
      .imagecaption{float:right;margin:0 30px 10px 30px;display:inline;}
   </style>
</head>
<body>
   <h1>Title</h1>
   <h2>Paragraph 1</h2>
   <div>
      <img src="http://www.lescasse.com/images/eric2011.png" height="240"
         width="240" alt="Eric Lescasse"/>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
         eiusmod tempor incididunt ut labore et dolore magna aliqua.
         Ut enim ad minim veniam, quis nostrud exercitation ullamco
         laboris nisi ut aliquip ex ea commodo consequat. Duis aute
         irure dolor in reprehenderit in voluptate velit esse cillum
         dolore eufugiat nulla pariatur. Excepteur sint occaecat
         cupidatat non proident, sunt in  culpa qui officia deserunt
         mollit anim id est laborum. Curabitur pretium tincidunt
         lacus. Nulla gravida orci a odio. Nullam varius, turpis et
         commodo pharetra, est eros bibendum elit, nec luctus magna
         felis sollicitudin mauris. Integer in mauris eu nibh
         euismod gravida. Duis ac tellus et risus vulputate
         vehicula. Donec lobortis risus a elit. Etiam tempor. Ut
         ullamcorper, ligula eu tempor congue, eros est euismod
         turpis, id tincidunt sapien risus a quam. Maecenas
         fermentum consequat mi. Donec fermentum. Pellentesque
         malesuada nulla a mi. Duis sapien sem, aliquet nec, commodo
         eget, consequat quis, neque. Aliquam faucibus, elit ut
         dictum aliquet, felis nisl adipiscing sapien, sed malesuada
         diam lacus eget erat. Cras mollis scelerisque nunc. Nullam
         arcu. Aliquam consequat. Curabitur augue lorem, dapibus
         quis, laoreet et, pretium ac, nisi. Aenean magna nisl,
         mollis quis, molestie eu, feugiat in, orci. In hac
         habitasse platea dictumst.Lorem ipsum dolor sit amet,
         consectetur adipisicing elit, sed do eiusmod tempor
         incididunt ut labore et dolore magna aliqua. Ut enim ad
         minim veniam, quis nostrud exercitation ullamco laboris
         nisi ut aliquip ex ea commodo consequat. Duis aute irure
         dolor in reprehenderit in voluptate velit esse cillum
         dolore eufugiat nulla pariatur. Excepteur sint occaecat
```

Copyright © 2016 Eric Lescasse

```
      cupidatat non proident, sunt in  culpa qui officia deserunt
      mollit anim id est laborum. Curabitur pretium tincidunt
      lacus. Nulla gravida orci a odio. Nullam varius, turpis et
      commodo pharetra, est eros bibendum elit, nec luctus magna
      felis sollicitudin mauris. Integer in mauris eu nibh
      euismod gravida. Du
   </p>
</div>
<h2>Paragraph 2</h2>
<ul>
   <li>Item 1</li>
   <li>Item 2</li>
   <li>Item 3</li>
   <li>Item 4</li>
   <li>Item 5</li>
   <li>Item 6</li>
   <li>Item 7</li>
   <li>Item 8</li>
</ul>
<h2>Paragraph 3</h2>
<table>
   <tr class="odd">
      <td>176,497</td>
      <td>374,402</td>
      <td>573,042</td>
      <td>103,832</td>
      <td>95,080</td>
      <td>993,406</td>
      <td>160,306</td>
      <td>253,007</td>
      <td>276,870</td>
      <td>349,306</td>
   </tr>
   <tr>
      <td>775,676</td>
      <td>776,058</td>
      <td>197,046</td>
      <td>744,572</td>
      <td>19,061</td>
      <td>346,795</td>
      <td>571,385</td>
      <td>256,624</td>
      <td>71,105</td>
      <td>50,765</td>
   </tr>
   <tr class="odd">
      <td>201,006</td>
      <td>293,622</td>
      <td>903,057</td>
      <td>677,193</td>
      <td>576,104</td>
      <td>572,744</td>
      <td>101,984</td>
      <td>31,602</td>
      <td>120,780</td>
      <td>943,693</td>
   </tr>
   <tr>
      <td>646,042</td>
```

```
            <td>22,970</td>
            <td>40,994</td>
            <td>977,396</td>
            <td>84,495</td>
            <td>97,864</td>
            <td>799,844</td>
            <td>976,839</td>
            <td>732,164</td>
            <td>466,403</td>
         </tr>
         <tr class="odd">
            <td>818,744</td>
            <td>618,144</td>
            <td>137,103</td>
            <td>274,802</td>
            <td>584,394</td>
            <td>895,375</td>
            <td>555,945</td>
            <td>762,089</td>
            <td>422,341</td>
            <td>277,330</td>
         </tr>
      </table>
   </body>
   </html>
```

Let's look at the result.



Retrieving the content of a Web page with GetWebPage

You can retrieve the content of any Web page using the **GetWebPage** method:

```
      'hh'□wi'GetWebPage' 'http://www.webstorm.com'
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
```

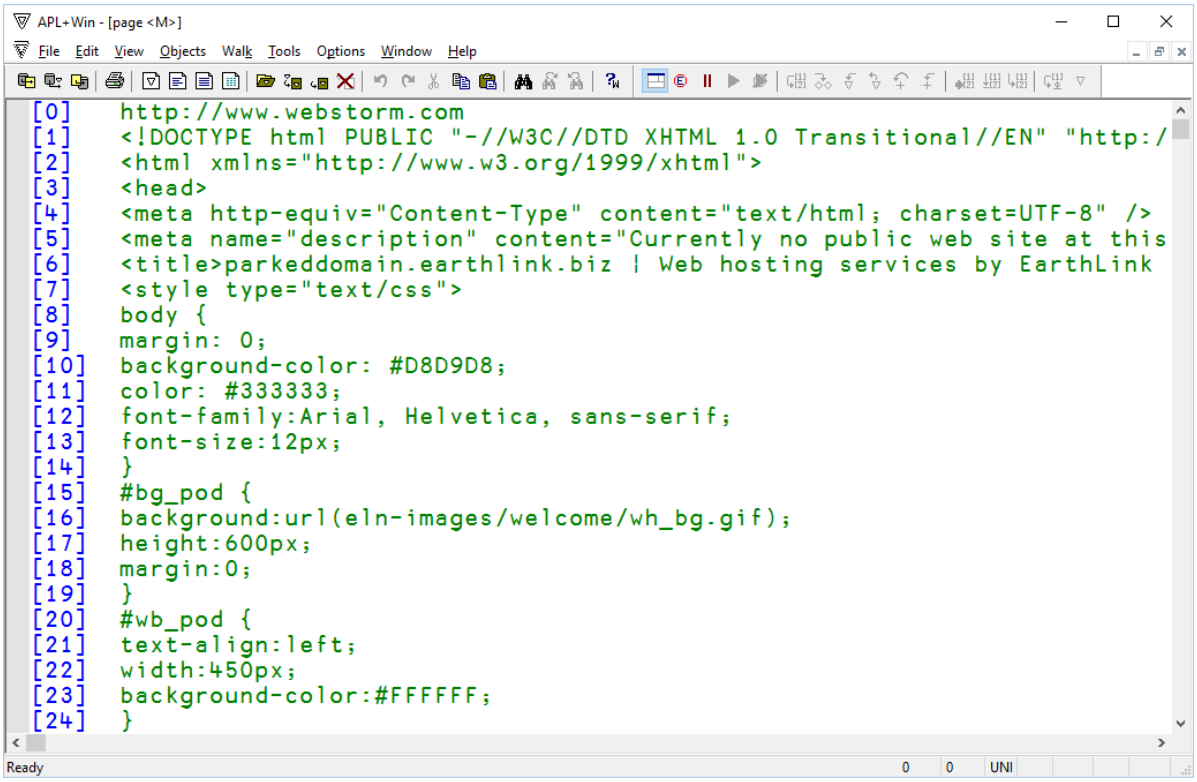
Copyright © 2016 Eric Lescasse

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<meta name="description" content="Currently no public web site at this web ad-
dress." />

<title>parkeddomain.earthlink.biz | Web hosting services by EarthLink Web Host-
ing</title>

<style type="text/css">
…
```

Loading the content of any Web page in the APL+Win Editor with EditWebPage

Alternatively, you can automatically load the content of any Web page in the APL+Win Editor (lines will be truncated at 1000 characters), using **EditWebPage**:

```
'hh'⎕wi'EditWebPage' 'http://www.webstorm.com'
```



Validating the HTML produced

The zHtml object includes a **Validate** method which can help you validate the HTML code produced by **zHtml**, or indeed validate any string HTML code or any Web page.

The Validate method result is a Nx4 nested matrix with:

[;1] = error level
[;2] = line number where error occurred

Copyright © 2016 Eric Lescasse

[;3] = column number where error occurred
[;4] = error message

Example:

```
      aaa←zzhtml15
      'hh'⎕wi'Validate'
 0 31 1 No warnings or errors were found
```

If used with no arguments, the **Validate** method validates the content of the **html** property contained in the **zHtml** instance.

The above is equivalent to:

```
      'hh'⎕wi'Validate'aaa
 0 31 1 No warnings or errors were found
```

You can also use the **Validate** method to validate any HTML string:

```
      'hh'⎕wi'Validate' '<html><body><h1>Test<buddy>'
 1 1  7 Inserting missing 'title' element
 2 1 22 <buddy> is not recognized!
 1 1 22 Discarding unexpected <buddy>
 1 1 28 Missing </h1>
 0 1 29 Document content looks like HTML 2.0
 0 1 29 3 warnings, 1 error was found!
 0 1 29 This document has errors that must be fixed before using HTML Tidy to gen-
erate a
```

By default, the messages are truncated at **100** characters but you can use an additional argument to truncate them at a different place:

```
      'hh'⎕wi'Validate' '<html><body><h1>Test<buddy>' 50
 1 1  7 Inserting missing 'title' element
 2 1 22 <buddy> is not recognized!
 1 1 22 Discarding unexpected <buddy>
 1 1 28 Missing </h1>
 0 1 29 Document content looks like HTML 2.0
 0 1 29 3 warnings, 1 error was found!
 0 1 29 This document has errors that must be fixed before
```

Finally, you can use still another argument which is a Boolean (the default is 0) to see all messages in full, but wrapped at the character you specify:

```
      'hh'⎕wi'Validate' '<html><body><h1>Test<buddy>' 50 1
 1 1  7  Inserting missing 'title' element

 2 1 22  <buddy> is not recognized!

 1 1 22  Discarding unexpected <buddy>

 1 1 28  Missing </h1>
```

Copyright © 2016 Eric Lescasse

```
0 1 29  Document content looks like HTML 2.0

0 1 29  3 warnings, 1 error was found!

0 1 29  This document has errors that must be fixed
        before using HTML Tidy to generate a tidied up
        version.
```

You can even use the **Validate** method to validate any Web page by passing it an argument starting with http://

When you do so, not only do you get a result in the APL Session, but the Web page is retrieved and automatically displayed in the APL editor. This allows you to toggle between the APL Session and the APL Editor (using **Ctrl+Tab**) to check the errors found by the **Validate** method.

Example:

```
      'hh'□wi'Validate' 'http://www.lescasse.com'
1     7  10 Adjacent hyphens within comment
1    47   1 Too many title elements in <head>
1   109 286 Unknown attribute "onfocus"
1   478 115 Unescaped & or unknown entity "&body"
1   509  13 img proprietary attribute value "absmiddle"
1   509  13 img lacks "alt" attribute
1   787 101 Unescaped & which should be written as &amp;
1   847  55 Replacing element <br> with <br>
1   847  55 Inserting implicit <br>
1   949  16 Unescaped & which should be written as &amp;
1   950  19 Unescaped & which should be written as &amp;
0  1081   8 11 warnings were found!
0  1081   8 The table summary attribute should be used to describe the table
             structure. It i
0  1081   8 The alt attribute should be used to give a short description of an im
            age; longer
0  1081   8 For further advice on how to make your pages accessible see
            "http://www.w3.org/W
```

Yes, my Home page does not validate! Sorry, I need to fix those errors.

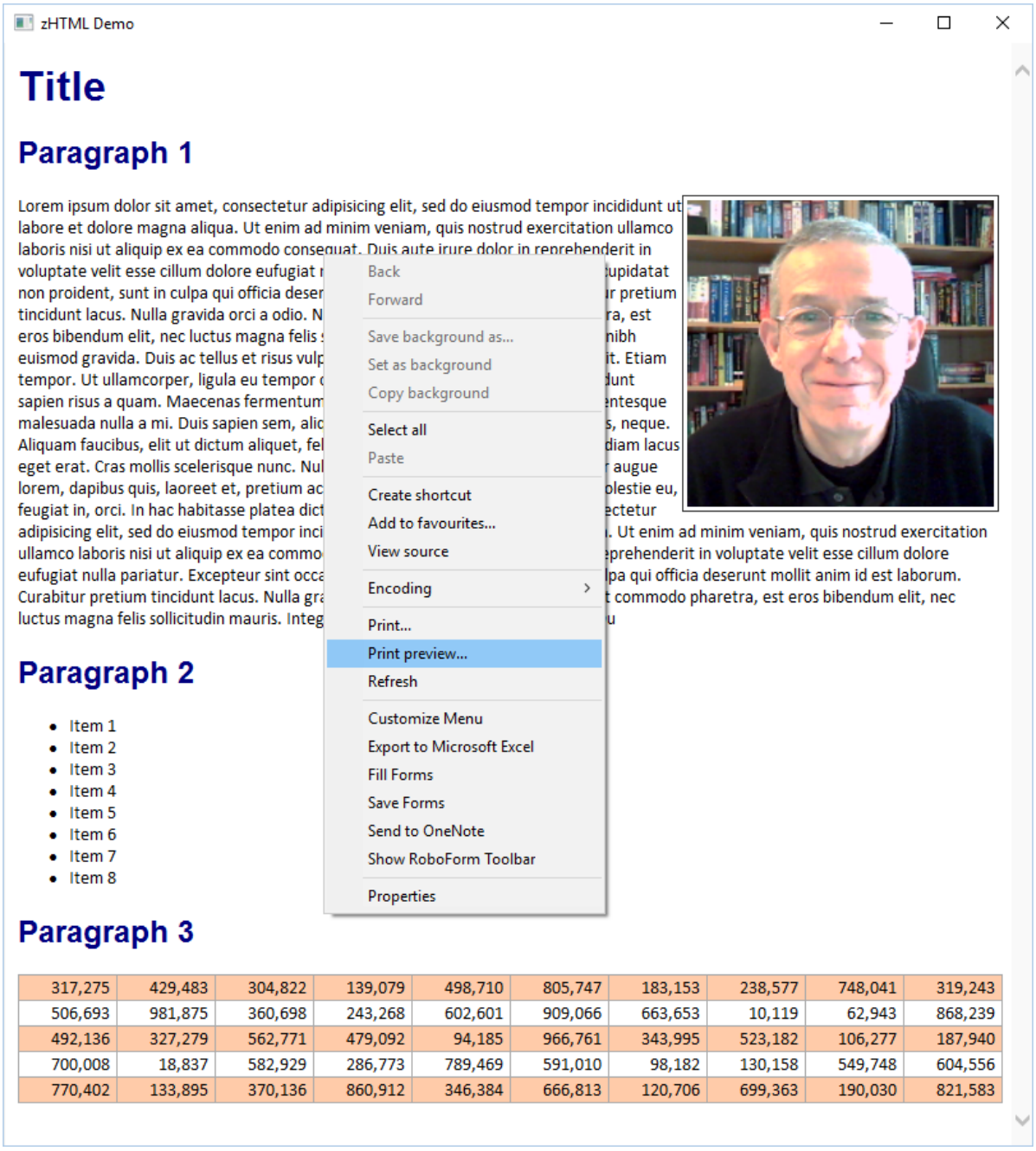
But try to validate **http://www.microsoft.com**!

Print Preview and Printing

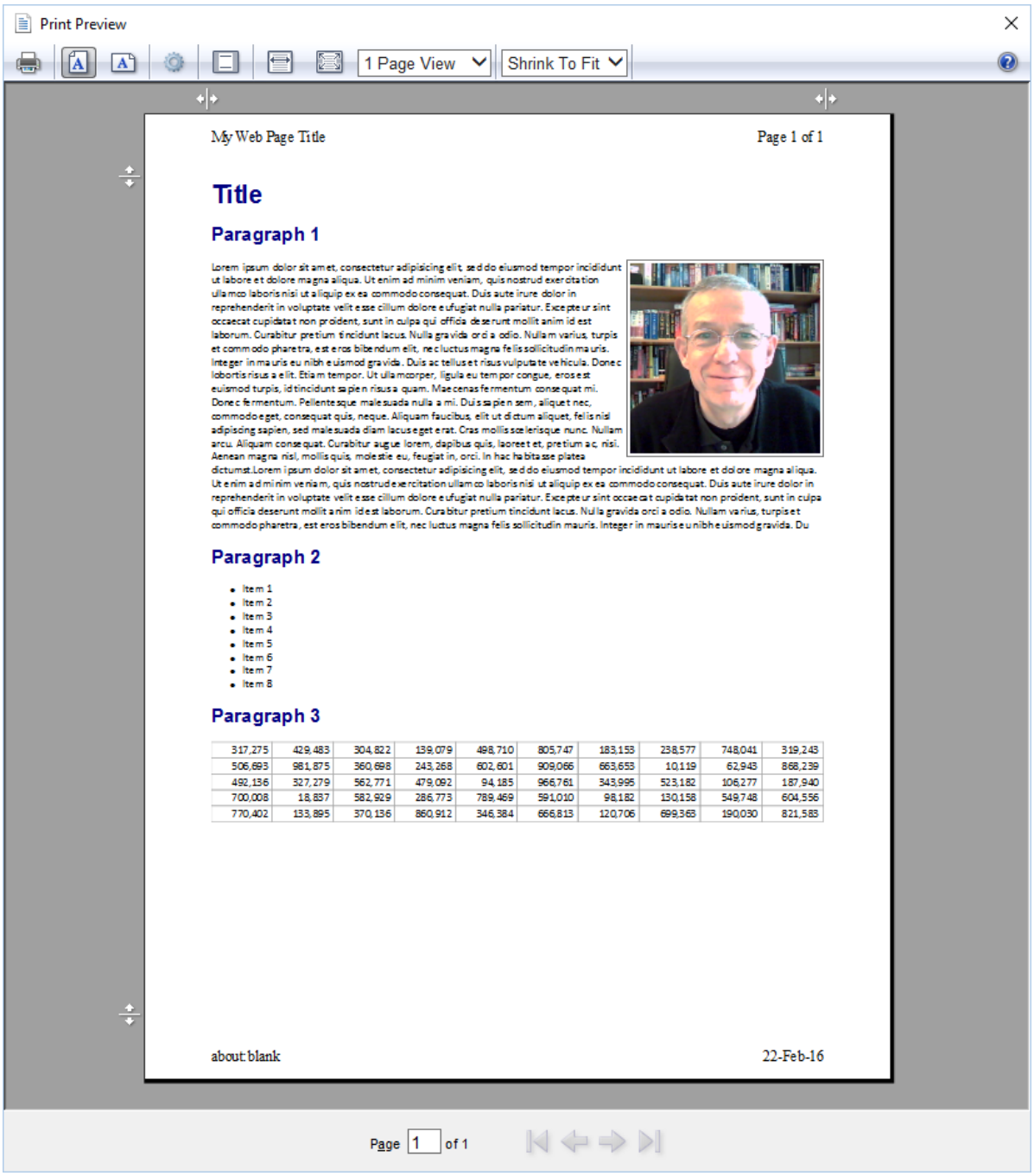
Using the **znetWebBrowser** object to display reports in an APL+Win application has another important advantage: thanks to its context menu, the **znetWebBrowser** object allows to **Print Preview** your report and to **Print** it.

So, you benefit from the Internet Explorer Print Engine for free and that can spare you a lot of work.

Copyright © 2016 Eric Lescasse

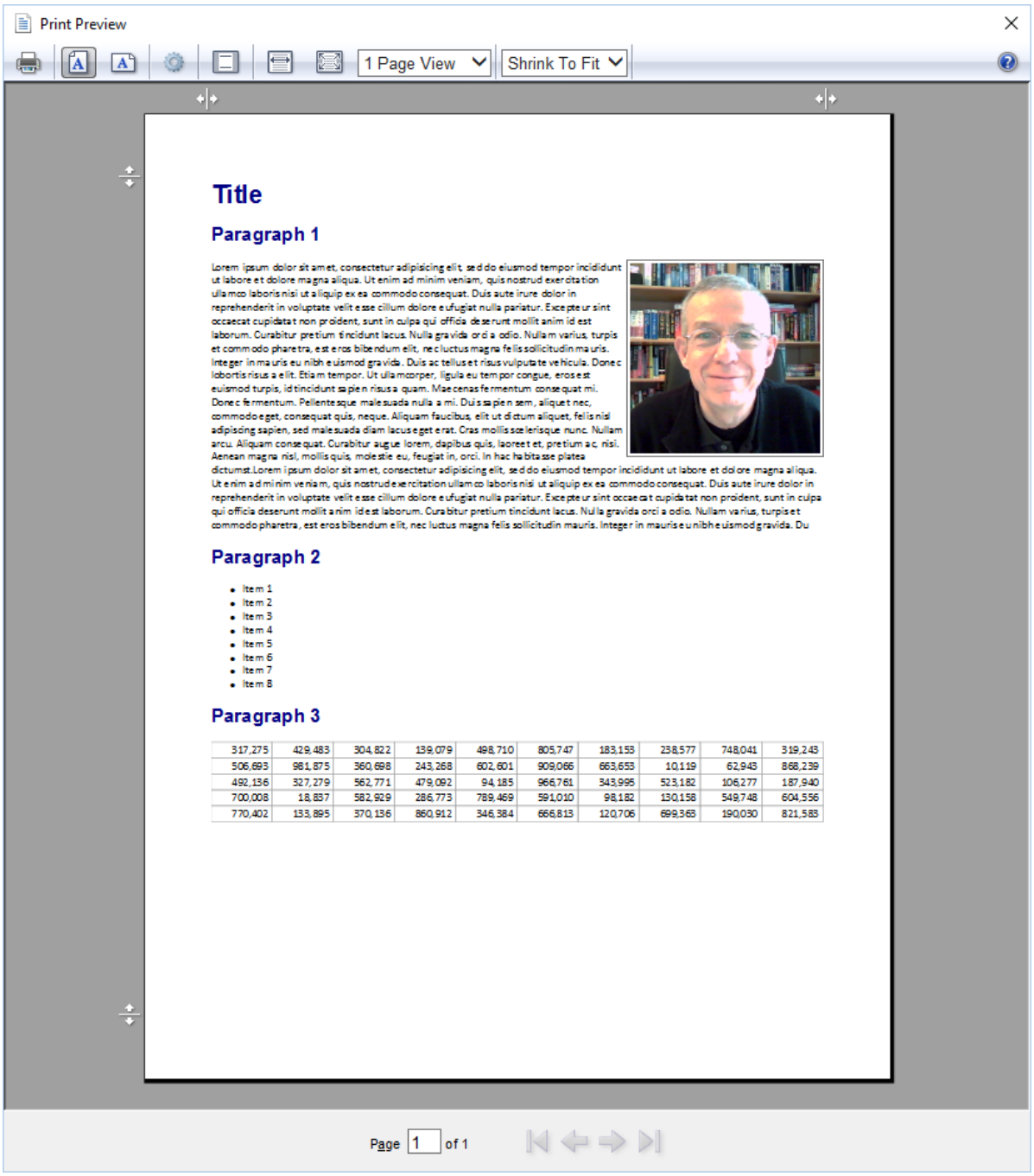


Copyright © 2016 Eric Lescasse



You can click on the toolbar button to remove the header and footer and get a cleaner printout:

Copyright © 2016 Eric Lescasse



Conclusion

The example shown above were simple, the goal being to teach how to use the zHtml object.

With this object you can create well-formed HTML code and display really nice-looking reports in your APL applications.

Copyright © 2016 Eric Lescasse

The zLCChart Object

Introduction

zLCChart is a general charting object that allows you to produce and include 2D and 3D charts in your APL+Win forms.

Simple Chart

The **zzchart** function contains a minimal example of a simple chart embedded in an APL+Win form:

```
    ∇ zzchart;data
[1]
[2]   ⍝ Step 1: Create the UI
[3]   ←'ff'⎕wi'*Create' 'zForm'('*caption' 'zLCChart')
[4]   ←'ff'⎕wi'*.dummy.Create' 'zEdit'('*where'¯5000 ¯5000)
[5]   ←'ff'⎕wi'*.lab.Create' 'zLabel'('*caption' '')('wherelc'0 0 '>>' '>>')('an-
chor'1 2 3 4)
[6]   ←'ff'⎕wi'*.lab.lcc.Create' 'zLCChart'('wherelc'0 0'>>' '>>')('anchor'1 2 3
4)
[7]
[8]   ⍝ Step 2: Build the chart
[9]   data←?8⍴10
[10]  ←'lcc'⎕wi'*zDrawBarChart'data
[11]
[12]  ⍝ Step 3: Display the chart
[13]  ←'lcc'⎕wi'*zDrawChart'
[14]  ⍝ Step 4: Display the form
[15]  ←'ff'⎕wi'DemoShow' .3 .3 1 1
    ∇
```

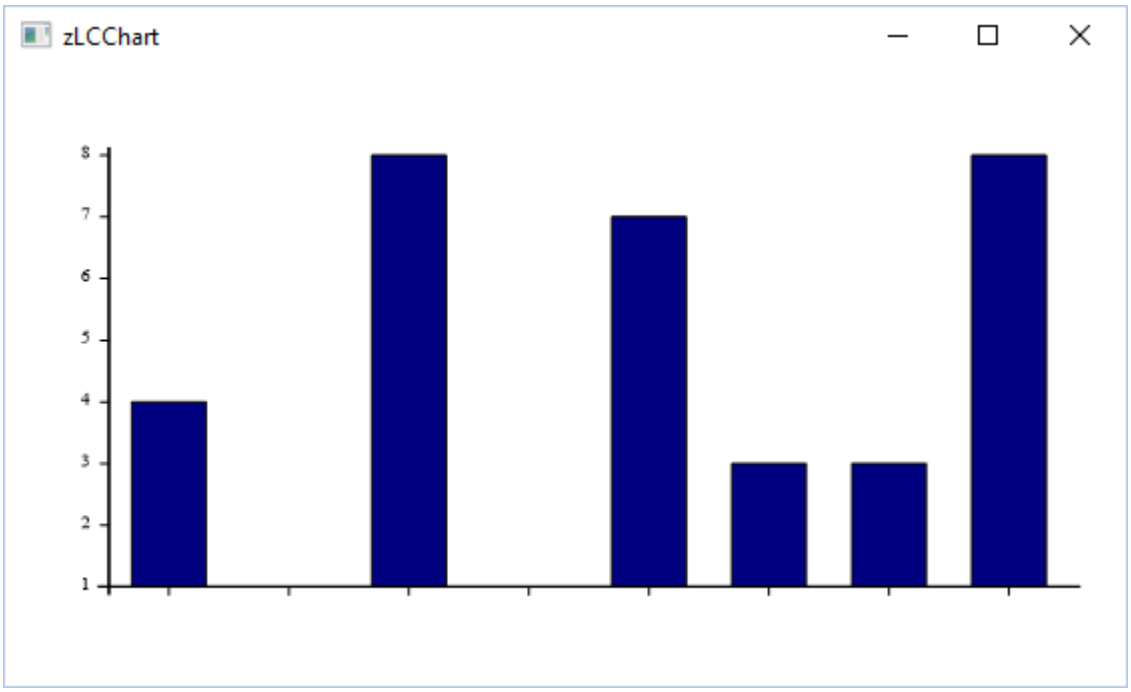
Step 2 is where you build your chart, using all the zLCChart properties and methods.

Note that, there must be at least one other control that can receive the focus (hence the dummy invisible zEdit control).

Also, note that the zLCChart object must be made a child of an APL zObject control (for example a zLabel or zFrame).

The above function produces the following chart.

Copyright © 2016 Eric Lescasse



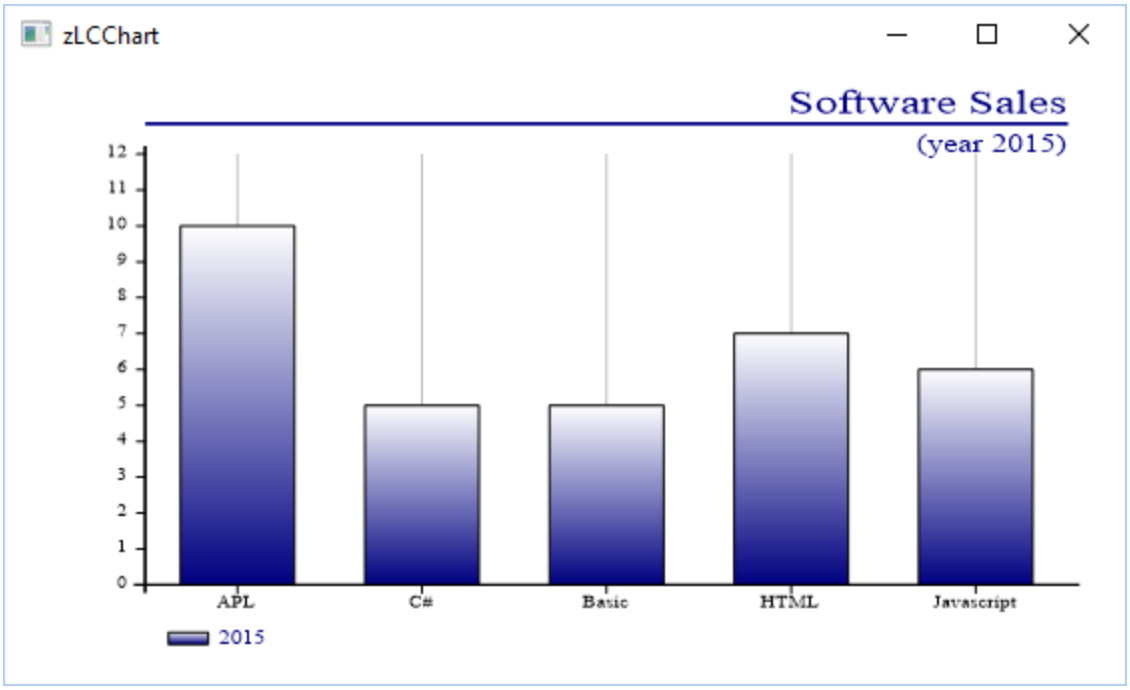
Customizing Your Chart

You can use **zLCChart** properties, before using the draw method that builds your chart (here **zDrawBarChart**) to customize the chart:

```
    ∇ zzchart;data
[1]
[2]   ⍝ Step 1: Create the UI
[3]   ←'ff'⎕wi'*Create' 'zForm'('*caption' 'zLCChart')
[4]   ←'ff'⎕wi'*.dummy.Create' 'zEdit'('*where'¯5000 ¯5000)
[5]   ←'ff'⎕wi'*.lab.Create' 'zLabel'('*caption' '')('wherelc'0 0 '>>' '>>')('an-
chor'1 2 3 4)
[6]   ←'ff'⎕wi'*.lab.lcc.Create' 'zLCChart'('wherelc'0 0'>>' '>>')('anchor'1 2 3
4)
[7]
[8]   ⍝ Step 2: Build the chart
[9]   data←?5⍴10
[10]  ←'lcc'⎕wi'*zHeading' 'Software Sales'
[11]  ←'lcc'⎕wi'*zSubheading' '(year 2015)'
[12]  ←'lcc'⎕wi'*zHeadingStyle'(HeadingStyles.Right+HeadingStyles.RuledBelow)
[13]  ←'lcc'⎕wi'*zMarginLeft' 50
[14]  ←'lcc'⎕wi'*zBarChartStyle' BarChartStyles.ForceZero
[15]  ←'lcc'⎕wi'*zXAxisStyle' XAxisStyles.GridLines
[16]  ←'lcc'⎕wi'*zSetFillStyles' FillStyle.GradientBottom
[17]  ←'lcc'⎕wi'*zSetXLabels' 'APL' 'C#' 'Basic' 'HTML' 'Javascript'
[18]  ←'lcc'⎕wi'*zSetYRange'0 12
[19]  ←'lcc'⎕wi'*zSetKeyText' '2015'
[20]  ←'lcc'⎕wi'*zDrawBarChart'data
[21]
[22]  ⍝ Step 3: Display the chart
[23]  ←'lcc'⎕wi'*zDrawChart'
```

Copyright © 2016 Eric Lescasse

```
[24]  ⍝ Step 4: Display the form
[25]  ←'ff'⎕wi'DemoShow' .3 .3 1 0
      ∇
```



The zLCChartExamples object

zLCChart has so many properties and methods that it may be sometimes difficult to find out the right ones to use.

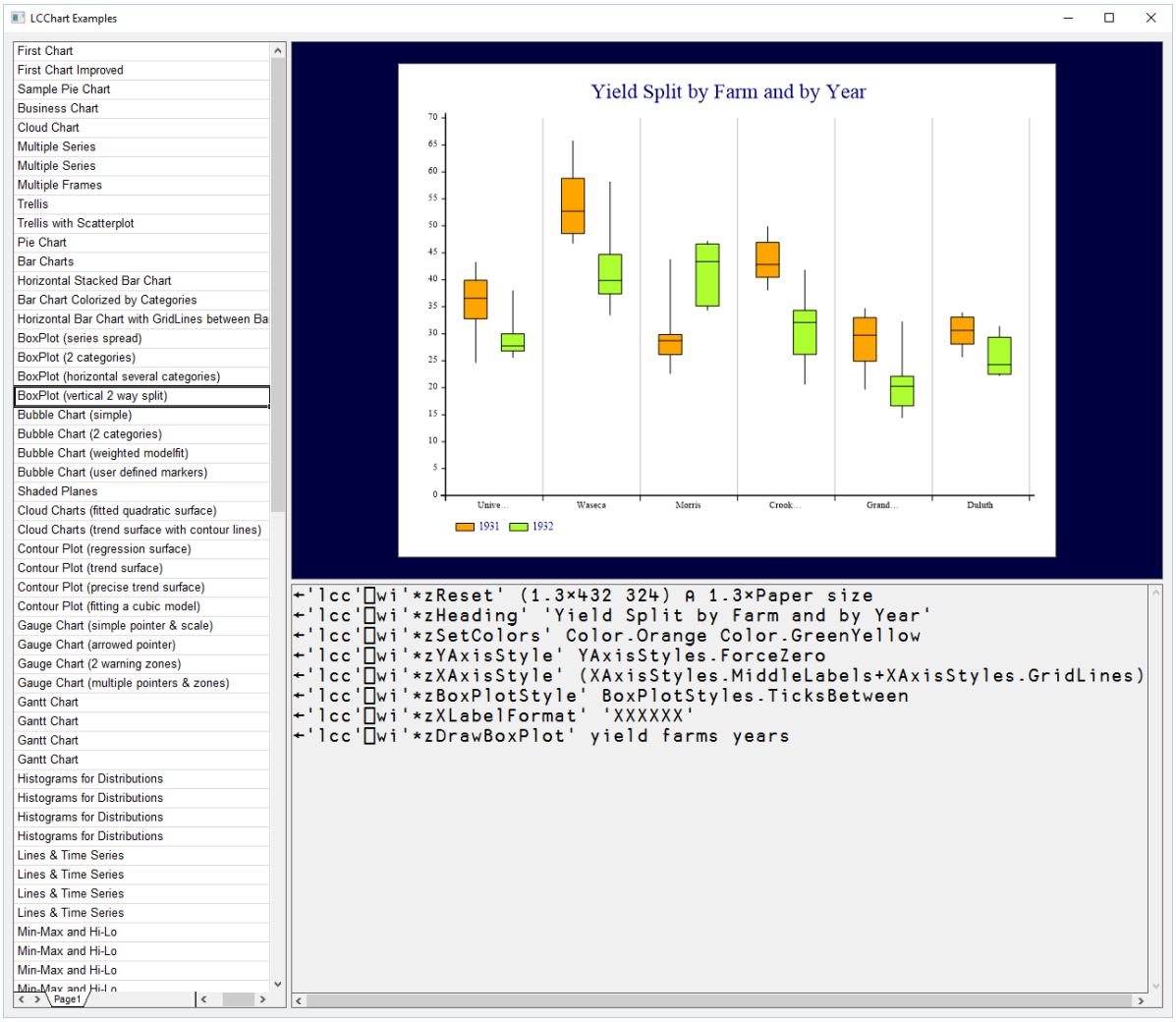
One of the simplest way to find out which ones to use is use the **zLCChartExamples** application.

Start the **zLCChartExamples** application as follows:

```
zzlcchartexamples
```

This displays a form allowing you to browse through 99 **zLCChart** examples and to see the code that has been used to produce the charts.

Copyright © 2016 Eric Lescasse



Enums and Styles

As you'll quickly see browsing the **zLCChartExamples**, **zLCChart** uses a lot of different styles.

All these styles as well as colors are defined as "enums" in **zObjects**. An "enum" is a simple ⎕**mom** APL object containing variables that you can access using the dot notation.

Enums appear as "variables" in your workspace:

```
BarChartStyles      GroupByFunction     TableStyles
BoxPlotStyles       HeadingStyles       TowerChartStyles
BubbleChartStyles   HistogramStyles     TraceChartStyles
CaptionStyles       IAxisStyles         TreeMapStyles
ChartType           KeyStyles           TrellisStyles
CloudChartStyles    LabelStyles         TriangleStyles
Color               LineGraphStyles     ValueTagStyles
ColorTranslator     LineStyle           VectorStyles
ContourPlotStyles   Marker              VennDiagramStyles
DataStyles          MinMaxChartStyles   XAxisStyles
DatumTagStyles      NoteStyles          XBarChartStyles
DialChartStyles     ParityStyles        XTickStyles
```

Copyright © 2016 Eric Lescasse

```
EquationStyles     PieChartStyles      YAxisStyles
FillStyle          PolarChartStyles    YTickStyles
FontStyle          ResponsePlotStyles  ZAxisStyles
FootnoteStyles     ScalebarStyles      ZTickStyles
FrameStyles        ScatterPlotStyles   ZoneStyles
GanttChartStyles   StepChartStyles
```

You can query the variables contained in any of these enums using .⎕nl 2 as follows:

```
      HeadingStyles.⎕nl 2
Bottom
Left
NoWrap
Opaque
Right
RuledBelow
```

You use a particular style by specifying its fully qualified name. Example:

```
      HeadingStyles.Right
 2
```

If you need to use more than one style at a time for a styles property, add the styles together. Example:

```
←'lcc'⎕wi'*zHeadingStyle'(HeadingStyles.Right+HeadingStyles.RuledBelow)
```

You can use the **zzTELPRINT** utility to display the styles across the screen rather than as a long vertical list. Example:

```
      zzTELPRINT BarChartStyles.⎕nl 2
CroppedAxes   ForceZero  Horizontal   NoAxes       TicksBetween
ExplodeAxes   FrameAxes  Indexed      OverlayGrid  ValueTags
FloatingBars  GridLines  Logarithmic  StackedBars
```

The Draw Functions Arguments

In general the Draw methods and most other zLCChart methods can accept a variety of arguments.

For example, if you define data in the last zzchart function as:

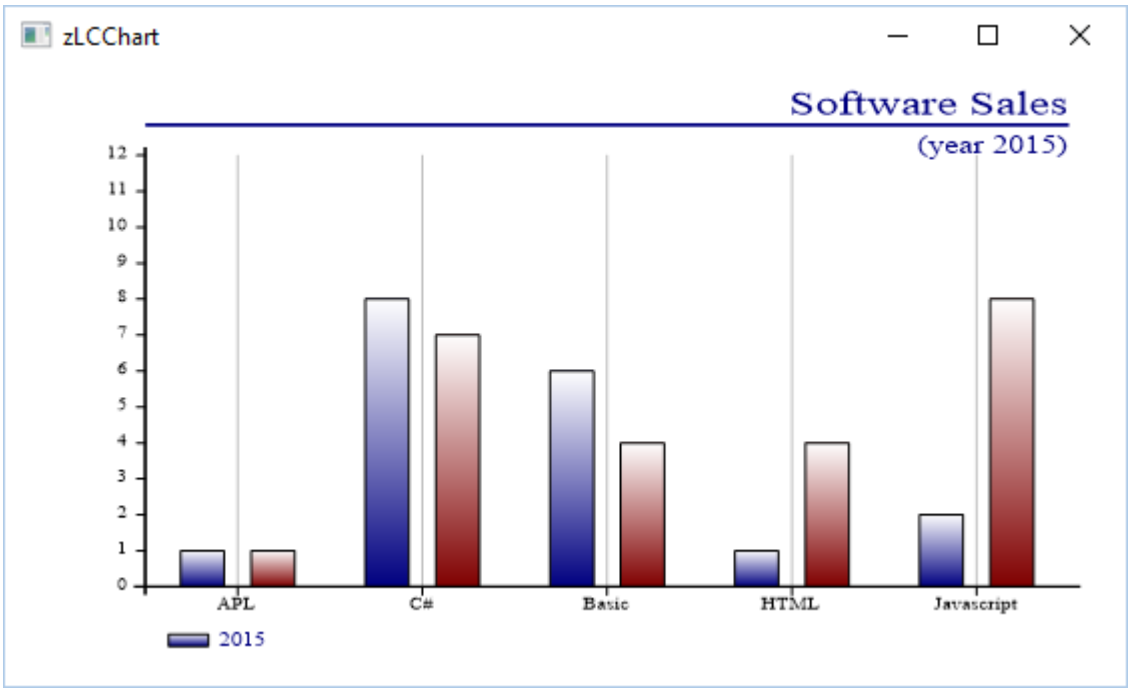
```
data←(?5⍴10)(?5⍴10)
```

Instead of:

```
data←(?5⍴10)
```

you'll get a side by side bar chart:

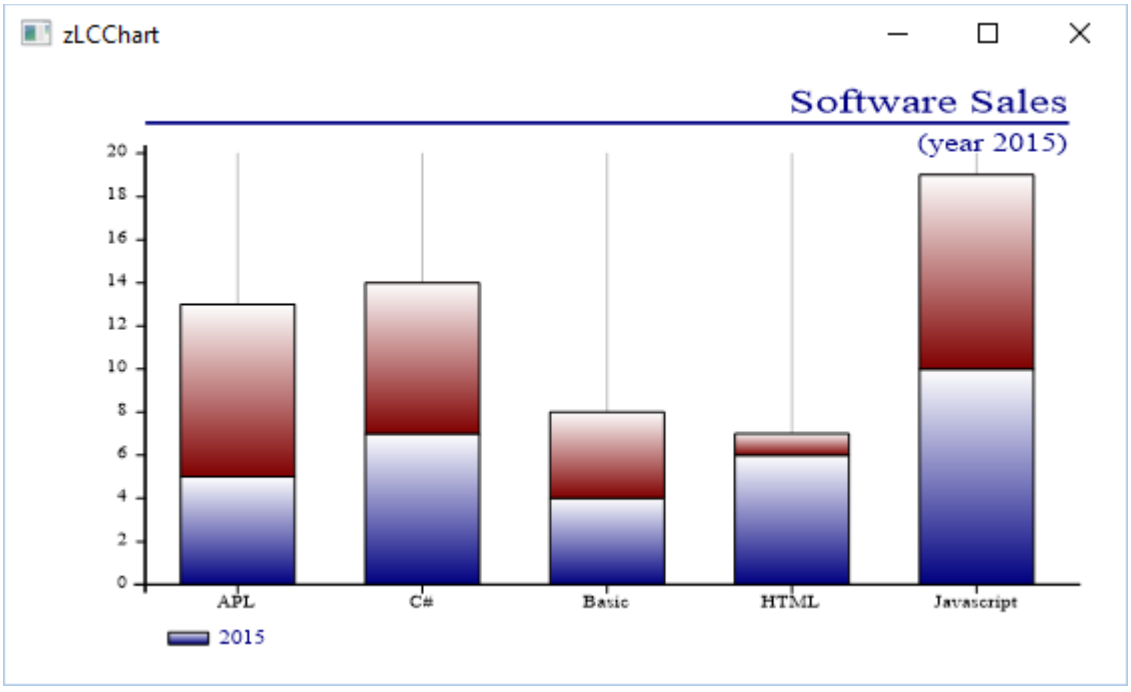
Copyright © 2016 Eric Lescasse



If you want to get a stacked bar chart instead, you should simply add the StackedBars style:

```
←'lcc'⎕wi'*zBarChartStyle' (BarChartStyles.ForceZero+BarChartStyles.StackedBars)
```

and you get:



Using zLCChart in more complex Forms

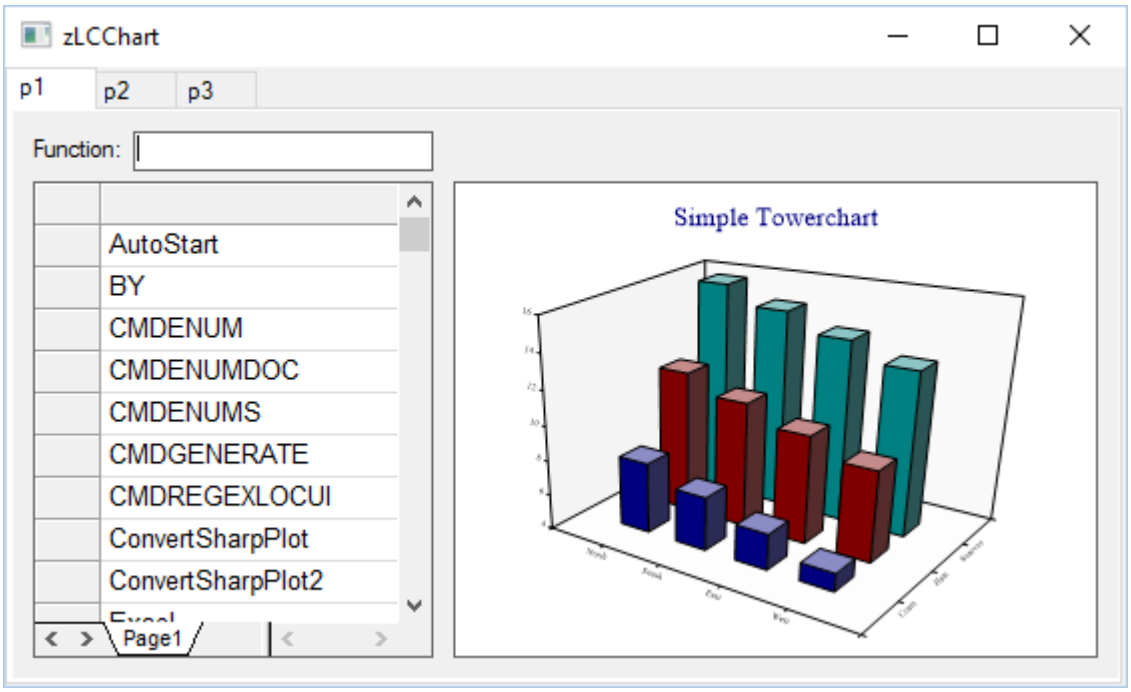
The zLCChart does not have to be the only object on your form: it can be one of multiple controls on your form. Here is an example:

Copyright © 2016 Eric Lescasse

```
    ∇ zzchart2;f;r;towers
[1]
[2]   f←(r←⍴⍴f)⍴f←⎕cn⎕nl 3
[3]   ⍝ Step 1: Create the UI
[4]   ←'ff'⎕wi'*Create' 'zForm'('*caption' 'zLCChart')
[5]   ←'ff'⎕wi'*.se.Create' 'zSelector'('wherelc'0 0'>>' '>>')('anchor'1 2 3 4)
[6]   ←'ff'⎕wi'*.se.p1.Create' 'zPage'
[7]   ←'ff'⎕wi'*.se.p2.Create' 'zPage'
[8]   ←'ff'⎕wi'*.se.p3.Create' 'zPage'
[9]   ←'p1'⎕wi'*.e1.Create' 'zEdit'('wherelc'⍬ 60 ⍬ 150)('caption' 'Func-
tion:'¯50)( 'anchor'1 2 3 2)
[10]  ←'p1'⎕wi'*.gr.Create' 'zGrid'('*border'1)('wherelc' '>' ⍬ '>'200)('anchor'1
2 1 4)
[11]  ←'gr'⎕wi'*Set'('*xRows'r)('*xCols'1)('*xValue'(⍳r)1 f)('*xColSize'1 200)
[12]  ←'p1'⎕wi'*.lab.Create' 'zLabel'('*border'1)('*caption' '')('wherelc' '=' '>'
'>' '>')('anchor'1 2 3 4)
[13]  ←'p1'⎕wi'*.lab.lcc.Create' 'zLCChart'('wherelc'0 0'>>' '>>')('anchor'1 2 3
4)
[14]
[15]  ⍝ Step 2: Build the chart
[16]  towers←(8 7 6 5) (12 11 10 9) (16 15 14 13)
[17]  ←'lcc'⎕wi'*zReset'432 324  ⍝⍝
[18]  ←'lcc'⎕wi'*zSetMargins' 48 12 34 0  ⍝ top bot left right
[19]  ←'lcc'⎕wi'*zHeading' 'Simple Towerchart'
[20]  ←'lcc'⎕wi'*zSetXLabels' 'North' 'South' 'East' 'West'
[21]  ←'lcc'⎕wi'*zSetYLabels' 'Coats' 'Hats' 'Scarves'
[22]  ←'lcc'⎕wi'*zTowerChartStyle' TowerChartStyles.WallShading
[23]  ←'lcc'⎕wi'*zDrawTowerChart' towers
[24]
[25]  ⍝ Step 3: Display the chart
[26]  ←'lcc'⎕wi'*zDrawChart'
[27]  ⍝ Step 4: Display the form
[28]  ←'ff'⎕wi'DemoShow' 450 700 1 0
    ∇
```

Note that the UI is clean and fully resizable: all objects resize nicely when the form is resized.

Copyright © 2016 Eric Lescasse



Replacing a chart by another on the fly

After having created the above form and displayed it, suppose we want to display another chart in place of the Tower chart.

Here are the steps:

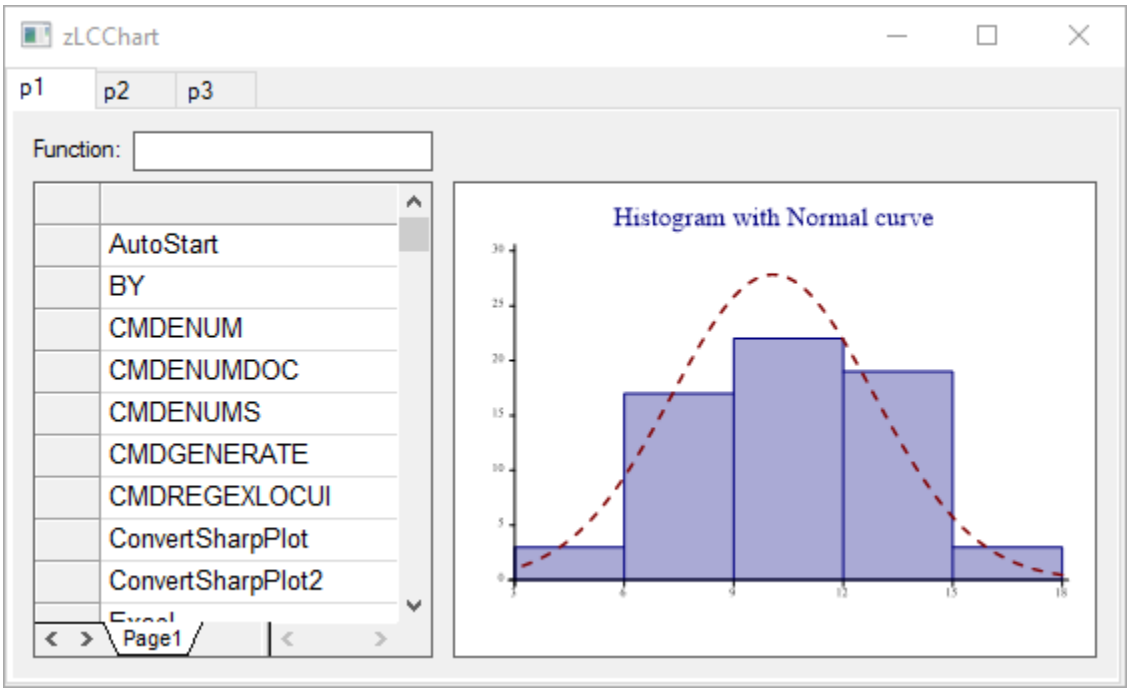
1. Call the zReset method
2. Build the new chart using the same zLCChart instance
3. Run the zDrawChart method

Example:

```
threedice←9 8 6 11 9 14 11 8 9 14 12 7 6 11 15 14 11 7 5 8 7 9 6 7
threedice,←12 7 13 8 9 13 7 7 8 12 11 16 12 9 11 12 12 12 9 14 8 9
threedice,← 13 11 10 13 11 13 9 10 10 4 14 8 9 11 12 5 12 15
←'lcc'⎕wi'*zReset'432 324
←'lcc'⎕wi'*zHeading' 'Histogram with Normal curve'
←'lcc'⎕wi'*zHistogramStyle' (HistogramStyles.SurfaceShading+HistogramStyles.Ris-
ers+HistogramStyles.NormalCurve)
←'lcc'⎕wi'*zSetFillStyles' FillStyle.Halftone
←'lcc'⎕wi'*zSetPenWidths' 2
←'lcc'⎕wi'*zSetXRange' 3 18
←'lcc'⎕wi'*zClassInterval' 3
←'lcc'⎕wi'*zSetXTickMarks' 3
←'lcc'⎕wi'*zDrawHistogram' threedice
←'lcc'⎕wi'*zDrawChart'
```

And the chart is immediately updated in the form at the time you run zDrawChart:

Copyright © 2016 Eric Lescasse



Overlaying Charts

With zLCChart you can overlay as many charts as you want on the same chart.

It is as simple as continuing drawing on the same canvas using normal zLCChart properties and methods.

All the charts you have created will be rendered in the same chart as you run the zDrawChart method.

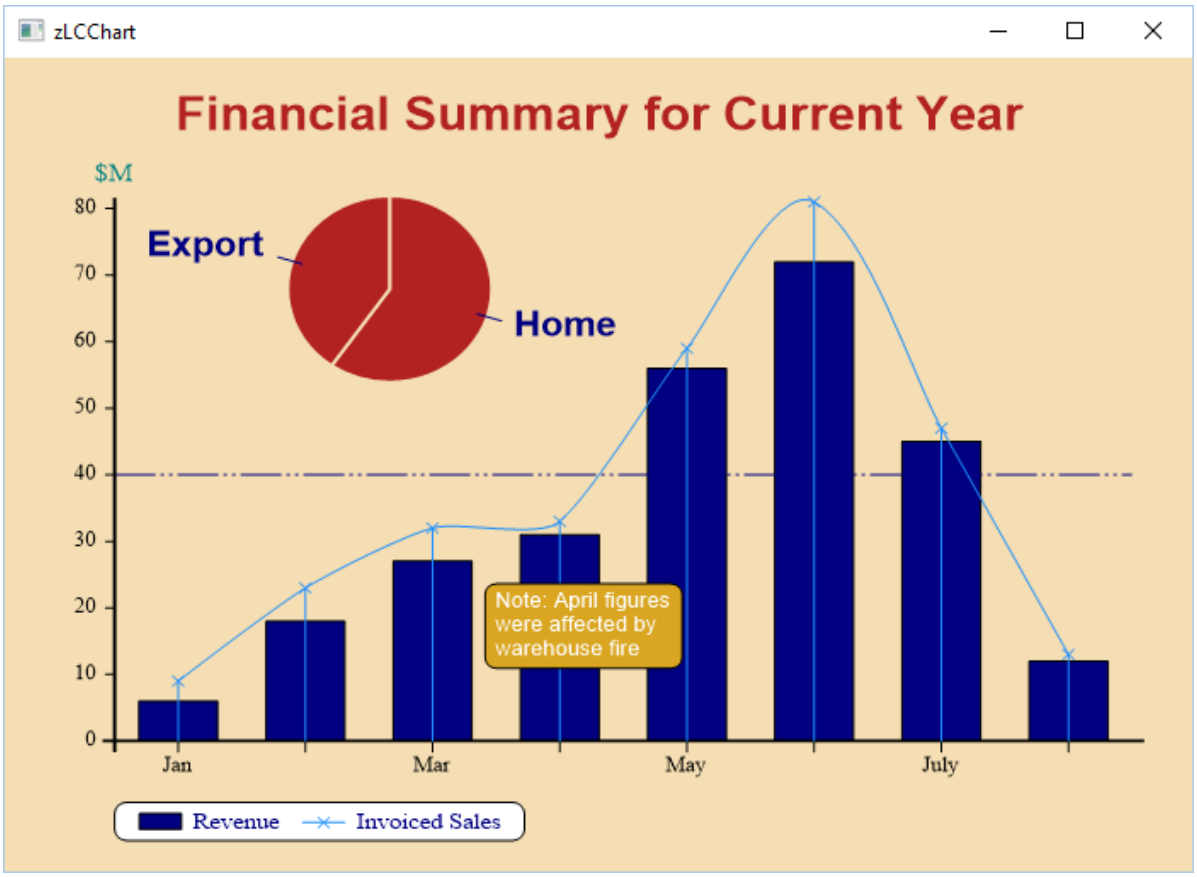
Here is a more complex example that demonstrates it:

```
    ∇ zzchart3;data1;data2;xlabs
[1]
[2]   ⍝ Step 1: Create the UI
[3]   ←'ff'⎕wi'*Create' 'zForm'('*caption' 'zLCChart')
[4]   ←'ff'⎕wi'*.dummy.Create' 'zEdit'('*where'¯5000 ¯5000)
[5]   ←'ff'⎕wi'*.lab.Create' 'zLabel'('*caption' '')('wherelc'0 0 '>>' '>>')('an-
chor'1 2 3 4)
[6]   ←'ff'⎕wi'*.lab.lcc.Create' 'zLCChart'('wherelc'0 0'>>' '>>')('anchor'1 2 3
4)
[7]
[8]   ⍝ Step 2: Build the chart
[9]   data1←6 18 27 31 56 72 45 12
[10]  data2←9 23 32 33 59 81 47 13
[11]  xlabs←'Jan' 'Feb' 'Mar' 'Apr' 'May' 'June' 'July' 'August\n(part)'
[12]  ⍝ First example with a barchart
[13]  ←'lcc'⎕wi'*zReset'432 324  ⍝⍝
[14]  ←'lcc'⎕wi'*zHeading' 'Financial Summary for Current Year'
[15]  ←'lcc'⎕wi'*zSetHeadingFont' 'Arial' 18 FontStyle.Bold Color.FireBrick
[16]  ⍝ Use a simple shaded paper as background
[17]  ←'lcc'⎕wi'*zSetBackground' Color.Wheat FillStyle.Solid
```

Copyright © 2016 Eric Lescasse

```
[18]  ←'lcc'⎕wi'*zMarginTop' 56 ⍝ Leave a lttle more headspace
[19]  ←'lcc'⎕wi'*zYCaption' '$M'
[20]  ←'lcc'⎕wi'*zYAxisStyle' (YAxisStyles.ForceZero+YAxisStyles.AtEndCaption)
[21]  ←'lcc'⎕wi'*zSetXLabels' xlabs
[22]  ←'lcc'⎕wi'*zSetDatumLineStyle' Color.Navy (24 3 2 3 2 3)
[23]  ←'lcc'⎕wi'*zSetYDatumLines' 40
[24]  ←'lcc'⎕wi'*zDrawBarChart' data1
[25]  ⍝ Overlay a linegraph on the same axis set
[26]  ←'lcc'⎕wi'*zLineGraphStyle' (LineGraphStyles.Markers+LineGraphStyles.Ris-
ers+LineGraphStyles.Curves)
[27]  ←'lcc'⎕wi'*zSetLineStyles' LineStyle.Dash
[28]  ←'lcc'⎕wi'*zSetColors' Color.DodgerBlue
[29]  ←'lcc'⎕wi'*zSetPenWidths' 0.8 ⍝ For line and risers
[30]  ←'lcc'⎕wi'*zFlexibility' 5 ⍝ For curved line
[31]  ←'lcc'⎕wi'*zDrawLineGraph' data2
[32]  ⍝ Key picks up series info from both charts
[33]  ←'lcc'⎕wi'*zSetKeyText' ('Revenue' 'Invoiced Sales')
[34]  ←'lcc'⎕wi'*zKeyStyle' (KeyStyles.Boxed+KeyStyles.Rounded)
[35]  ⍝ Overlay a Pie chart
[36]  ←'lcc'⎕wi'*zSetXLabelFont' 'Arial' 13.4 FontStyle.Bold Color.Navy
[37]  ←'lcc'⎕wi'*zSetXLabels' ('Home' 'Export')
[38]  ←'lcc'⎕wi'*zSetPieCenter' 100 180
[39]  ←'lcc'⎕wi'*zPieRadius' 50
[40]  ←'lcc'⎕wi'*zSetEdgeStyle' Color.Wheat LineStyle.Solid 1.2
[41]  ←'lcc'⎕wi'*zSetColors' Color.FireBrick
[42]  ←'lcc'⎕wi'*zDrawPieChart' (34 23)
[43]  ⍝ Add a note
[44]  ←'lcc'⎕wi'*zNoteStyle' (NoteStyles.Boxed+NoteStyles.Rounded)
[45]  ←'lcc'⎕wi'*zSetNoteBackground' Color.Goldenrod
[46]  ←'lcc'⎕wi'*zSetNoteFont' Color.White
[47]  ←'lcc'⎕wi'*zDrawNote' 'Note: April figures were affected by warehouse fire'
3.5 20 0 1.6
[48]
[49]  ⍝ Step 3: Display the chart
[50]  ←'lcc'⎕wi'*zDrawChart'
[51]  ⍝ Step 4: Display the form
[52]  ←'ff'⎕wi'DemoShow' .3 .3 1 0
    ∇
```

Copyright © 2016 Eric Lescasse



Making it easier to work with zLCChart

The]enum and]enums user commands

It is pretty hard to remember all the possible styles you can use with zLCChart.

zObjects includes 2 User Commands to help you work with Enums:

-]enums
-]enum

The first one: **]enums** can be used as follows:

]enums {string}	Lists all enums currently defined in the workspace If {string} is used, lists only those enusm containing string in their names
]enums /build]enums /b	Creates all enums in the workspace (it required the zEnums object)

Copyright © 2016 Eric Lescasse

]enums /erase]enums /e	Erases all enums from the workspace

The second one: **]enum** can be used as follows:

]enum enumname {/names} {/nums} {/contains=string}

The]enum command lists all the names, nums or members for a given enum

If /na{mes} is used,]enum returns all names contained in the specified enum

If /nu{ms} is used,]enum returns all numeric values contained in the specified enum

If /c{ontains}=string is used,]enum displays only the enum members containing <sting> in their names

Examples:

```
      ]enum TableStyles
Spanned=1     Merged=2      UseHeaders=32  Absolute=512  FitHeight=8192
Rounded=128   Center=2048   FitWidth=4     Boxed=64
GridLines=16  Shadowed=256  Right=4096     Opaque=8

      ]enum TableStyles /na
Spanned  GridLines  Center    UseHeaders  Right     Boxed   FitHeight
Rounded  Merged     Shadowed  FitWidth    Absolute  Opaque

      ]enum TableStyles /nu
1  2  4  8  16  32  64  128  256  512  2048  4096  8192

      ]enum Color /c=red
DarkRed=139 0 0             IndianRed=205 92 92        Red=255 0 0
MediumVioletRed=199 21 133  PaleVioletRed=219 112 147  OrangeRed=255 69 0
```

Getting zLCChart documentation:]lccprops and]lccdoc

Two user commands are also available to help find the names of **zLCChart** properties and methods you want to use as well as to get documentation about these properties and commands, without having to create an instance of zLCChart first.

The **]lccprops** user command allows you to list all properties, methods and events available in zLCChart:

```
      ]lccprops
Properties
----------
*apldata              *zFrameStyle          *zTickMarkWidth
*children             *zGanttChartStyle     *zTitleBar
*class                *zGap                 *zTowerAspect
 class                *zGroupGap            *zTowerChartStyle
```

```
*clsid                  *zGutter              *zTowerLimit
*data                   *zHandle              *zTraceChartStyle
*def                    *zHeading             *zTreeMapStyle
*description            *zHeadingLineSpacing  *zTrellisStyle
*errorcode              *zHeadingStyle        *zTrendLineKey
…
```

The list is pretty long, so often you'll prefer to filter it by specifying the subject you are interested in.

For example, if you are searching for a property or method relative to value tags, you can enter:

```
      ]lccprops value
Properties
----------
*zMissingValue      *zValueTagAngle   *zValueTagStyle
*zValueLineSpacing  *zValueTagFormat

Methods
-------
*zGetFittedValues  *zSetValueFont   *zSetValueTags
*zGetValueFont     *zSetValueNudge

Events
------
```

The **]lccprops** argument is **case insensitive**.

Once you have identified the method you want to use, you can get documentation all its possible syntaxes using the **]lccdoc** user command:

```
      ]lccdoc SetValueTags
Syntax: 'obj'□wi'*zSetValueTags'(string label)
Summary: Set Value tags with a comma-separated string

Syntax: 'obj'□wi'*zSetValueTags'(string[][] labels)
Summary: Set multiple rows of labels

Syntax: 'obj'□wi'*zSetValueTags'(int[] values)
Summary: Format Value tags from an integer array with current formatter

Syntax: 'obj'□wi'*zSetValueTags'(double[] values)
Summary: Format Value tags from a numeric array with current formatter

Syntax: 'obj'□wi'*zSetValueTags'(string[] labels)
Summary: Set Value tags from an array of strings
```

Note that most **zLCChart** method have a good number of overlays, i.e. may be used with various types of arguments.

At least the **]lccdoc** user command list all the possible syntaxes, indicating the C# type of argument which is expected.

Copyright © 2016 Eric Lescasse

An array in C# is noted as [], so int[] means a vector of integers.

An array of array in C# is noted as [][], so string[][] means an array of string arrays. This means that you should pass a nested vector of nested vector of strings, such as:

('string1' 'string2')('string3' 'string4')

The **]lccdoc** argument is **case sensitive**.

As per this version, the **]lccdoc** user command retrieves documentation for methods only.

Also, for **]lccdoc** to work, the **sharpplot.xml** file must be in the same folder as the **zObjects.dll** file.

Warning:
The **]lccdoc** user command does not yet work for a good number of **zLCChart** methods.

Errors such as:

```
   ]lccdoc SetYLabelFont
WI ERROR: LC.zObjects exception -2147467262 (0x80004002) Unable to cast
object of type 'Color[]' to type 'System.Int32[]'.
```

may occur, or the result may be empty.

This should be fixed in a later release.

Copyright © 2016 Eric Lescasse

Using .Net zObjects

Introduction

zObjects includes a number of .Net objects that you can use in your APL applications.

These objects will be reviewed in this chapter.

All these objects have a name starting with **znet,** most often followed by the .Net class name they represent (example: **znetButton**, **znetRegex**, **znetSmtpClient**, etc.)

Just like all other zObjects, you use the .Net objects using □wi.

Mostly everything you learnt so far still applies to the .Net zObjects.

Getting Help

The bad news is that, working with the .Net Framework may be a difficult task and may require quite some knowledge and experience.

Happily, a number of factor makes this task easier that what you may think:

- First, whenever possible zObjects tries to simplify things and adapt them to an APL+Win user.
- Second, you can use the .Net zObjects controls in the exact same manner as you use APL controls (more about this later)
- Third, you can get a real lot of help on Internet about any .Net topic

As APL developers, we are not used to search information on Internet for a good reason: there is very few information to find there and we are left alone with the APL documentation and the APL+Win Help Files!

But, as far as .Net and Windows Forms is concerned, there are literally millions of Web Sites and Web Pages that can virtually answer any question you may have.

You can find:

- Tutorials about any topic
- Videos about any topic
- Web Site that answer developer's questions
- Etc.

Copyright © 2016 Eric Lescasse

Among resources you'll want to use, here are a few important ones:

Google (www.google.com) to search on any .Net subject

C# Corner (http://www.c-sharpcorner.com/) includes tutorials on mostly all .Net Windows Forms controls

http://www.dotnetperls.com also includes turtorials on many .Net Windows Forms controls and classes

On http://www.youtube.com you can find video tutorials about almost any possible .Net subjects (although they not all have the same quality).

If you search on a topic in Google and find answers that involve http://www.stackoverflow.com, go read those pages as they often give you the solution to your problem. StackOverflow is the top site for answering .Net developers questions.

The various .Net zObjects

There are several categories of .Net zObjects you may want to use:

- Non visual .Net classes
  These include:

znetEnvironment	znetPath	znetTimeSpan	
znetDateTime	znetFile	znetProcess	znetWebClient
znetDictionary	znetFileInfo	znetRegex	znetWebRequest
znetDirectory	znetFtpWebRequest	znetSmtpClient	znetXmlDocument
znetDirectoryInfo	znetHashSet	znetSortedDictionary	znetXmlWriter

- .Net Windows Forms controls
  These include:

znetButton	znetContextMenu	znetNumericUpDown	znetTabControl
znetButton16	znetDomainUpDown	znetPanel	znetTabPage
znetButton32	znetLinkLabel	znetPictureBox	znetTextBox
znetCheckBox	znetMaskedTextBox	znetRichTextBox	znetWebBrowser
znetCheckedListBox	znetMenuItem	znetSplitContainer	
znetComboBox	znetNotifyIcon	znetSplitterPanel	

- .Net Forms (which are complete .Net applications developed to help with various topics)
  These include:

znetRegexTest	znetSelectIconForm

These lists are exhaustive for the current zObjects release (2.4.0.0), but many more .Net zObjects will be added in the future.

Copyright © 2016 Eric Lescasse

Basic Principles

.Net zObjects property and method names

All the .Net zObjects provide you with the same set of properties, methods and events as the original .Net class from which they derive, may contain.

All the .Net zObjects properties, methods have the exact same name as the real .Net object properties or methods except that they have a **z** prefix and that you must also use the ***** prefix.

For example, a **File** .Net object has a **ReadAllLines** method which reads an entire file and returns its content as a nested vector of strings (one per line in the file): if you create an instance of the **znetFile** object, you will have to use ***zReadAllLines** if you want to call this method.

Example:

```
      ←'file'⎕wi'*Create' 'znetFile'
      aaa←'file'⎕wi'*zReadAllLines' 'c:\aplwin15.1\HelpStrings.txt'
      ⍴aaa
9441
```

If you try the above example, you will be amazed by its speed. The **HelpStrings.txt** is a **436 Kb** file with **9441** lines, but you get the result absolutely instantaneously, so much that you may ask yourself if the file has really been read.

That speed is typical of the .Net Framework.

> **Note:**
> The fact that the .Net zObjects properties and methods always have the same names as the .Net object properties and methods (except for the *z prefix), makes it easy to learn how to use them: just search for a good Tutorial on Google and try the properties and methods you see there, prefixing them with *z. It is that simple.

.Net zObjects event names

You don't use the ***z** prefix for events, but the ***onX** prefix.

Here is an example:

```
←'ff'⎕wi'*Create' 'zForm'('DemoShow'.4 .4 1 1)
←'ff'⎕wi'*.zb.Create' 'znetButton'
←'ff'⎕wi'*.zb.onXClick' '⎕←"Clicked"'
```

Now click on the .Net button in the form: you should see the following text printed in the APL Session, proving that the **onXClick** event handler is correctly firing:

```
Clicked
```

Dynamic help

Of course, remember that you can always use the object **allprops** and **mainprops** properties to learn about the available properties, methods and events for the .Net zObjects.

You can also always query the following properties: properties, methods, events, hproperties, hmethods, hevents (without the * prefix).

When you call a method or property you must use the *z prefix for .Net zObjects.

However, when you want to get documentation about a property, method or event, you must not use the ***z** prefix; just use the **?** prefix. For example, to get the documentation about the ***zFlatStyle** property, do:

```
      'bn1'⎕wi'?FlatStyle'
Syntax: {flatStyle←}'obj'⎕wi'*zFlatStyle'{flatStyle}
Summary: Gets or sets the flat style appearance of the button control
flatStyle: One of the FlatStyle values. The default value is Standard
           0 = FlatStyle.Flat
           1 = FlatStyle.Popup
           2 = FlatStyle.Standard
           3 = FlatStyle.System
```

Note that, if the argument is a .Net **enum**, you get the complete enum list with its values.

To use, the ***zFlatStyle** property you would write:

```
      'bn1'⎕wi'*zFlatStyle'3          ⍝ set FlatStyle to FlatStyle.System
```

For an event, do not omit the ***onX** prefix. For example, to get the documentation converning the ***onXMouseMove** event, do:

```
      'bn1'⎕wi'?MouseMove'
Syntax: 'obj'⎕wi'*onXMouseMove' eventHandler
Summary: Occurs when the mouse pointer is moved over the control
   ⎕warg[1]= (int) Button:
             Gets which mouse button was pressed. (MouseButtons)
   ⎕warg[2]= (int) Clicks:
             Gets the number of times the mouse button was pressed and
             released. (int)
   ⎕warg[3]= (int) X:
             Gets the x-coordinate of the mouse during the generating mouse
             event. (int)
   ⎕warg[4]= (int) Y:
             Gets the y-coordinate of the mouse during the generating mouse
             event. (int)
   ⎕warg[5]= (int) Delta:
             Gets a signed count of the number of detents the mouse wheel has
             rotated, multiplied by the WHEEL_DELTA constant. A detent is one
             notch of the mouse wheel. (int)
```

Copyright © 2016 Eric Lescasse

```
□warg[6]= (object) Location:
          Gets the location of the mouse during the generating mouse
          event. (int[])
```

Note that you get a complete description of the □warg arguments.

Limitations

APL and .Net data types

The .Net zObjects have some limitations: no all the properties, methods or events of the original .Net class are available to APL.

The reason which you must very well understand is the following.

APL+Win knows about 4 data types:

- Booleans (1 bit)
- Strings
- Integers (32-bit)
- Floating Points (64-bit)

The .Net Framework knows and uses more than 10000 types!

The bad news is that APL cannot understand any of these 10000 types, except the 4 ones who correspond to the above 4 APL types (i.e. bool, string, int, double).

The good news is that, in many cases, I would even say in very many cases, it is possible to convert a .Net type that APL does not understand to a structure that APL understands, so this is exactly what the **zObjects.dll** is trying to do whenever possible.

Automatic converters: the Rectangle example

Let's take an example: there is a .Net Framework type called **Rectangle**[8]

A .Net Button class has a **ClientRectangle** property that returns a **Rectangle** object.

APL would not be able to use this **ClientRectangle** property because APL does not know what a **Rectangle** is.

But a rectangle is not much more than 4 integers describing its **Left** and **Top** position and its **Width** and **Height**.

So it is possible to write an automatic converter that converts a **Rectangle** to a 4-element integer array and that is something that APL knows how to understand pretty well!!

[8] You can find a description at: https://msdn.microsoft.com/en-us/library/system.drawing.rectangle(v=vs.110).aspx

Copyright © 2016 Eric Lescasse

For this reason, the **zObjects.dll** contains a **Converters.cs** class with tons of conversion methods, that automatically convert many .Net types to structures that APL can understand.

Thanks to that **Converters** class, we can now query the **ClientRectangle** property as follows:

```
      ←'ff'⎕wi'*Create' 'zForm' ('DemoShow'.4 .4 1 1)
      ←'ff'⎕wi'*.zb.Create' 'znetButton'
      'ff'⎕wi'*.zb.zClientRectangle'
0 0 160 162
```

Note:
It is important to note that in the .Net world Left always comes before Top and Width always comes before Height.

So a **Size** object in .Net is initialized by specifying its **Width** first and then its **Height**. In APL a **size** property returns the **Height** first and then the **Width**.

zObjects converter methods always return results in an order familiar to APL developers.

So, unlike the .Net **ClientRectangle** property which returns: **Left Top Width Height**, the **zObject zClientRectangle** property returns: **Top Left Height Width**.

This again, makes it easier for an APL developer to work with .Net zObjects.

Limitations

This automatic type conversion scheme is working pretty well to allow a lot of .Net properties or methods to be used from APL, but there are limitations:

- Converters have been written in **zObjects.dll** only for the most often encountered .Net data types
- For some .Net data types it is not possible or practical to write an automatic Converter

As a consequence, zObjects does not allow to use absolutely all .Net object properties and methods and events.

However, you'll see that in most cases you can use a vast majority of the existing .Net properties, methods and events, probably many more than you really need in your applications.

Copyright © 2016 Eric Lescasse

Using a non visual .Net zObject from APL+Win

In this paragraph we will explain how you may use a non visual .Net zObject. Let's take the **znetFile** and **znetFileInfo** objects as examples.

Your first step, before trying to use znetFile and znetFileInfo, should be to go to google and search for a good tutorial on the .Net File and FileInfo class.

So, go to google and type: **C# File class tutorial**

You'll get 5 430 000 results! No panic. You generally find your happiness within the first result page.

For example, visit the following link:
http://www.c-sharpcorner.com/UploadFile/84c85b/file-io-using-C-Sharp/

Let's try to use the examples on this page, but using znetFile and znetFileInfo:

We first need to create an instance of the **znetFile** object:

```
      ←'file'⎕wi'*Create' 'znetFile'
```

Before we start using the **znetFile** object, it's a good idea to query its **allprops** property to check what is available:

```
      'file'⎕wi'allprops'
Properties
----------
*apldata    *def           *instance    *name        *self         *ΔΔisnetclass
*children   *description   *interface   *obj         *state
*class      *errorcode     *links       *opened      *suppress
 class      *errormessage  *methods     *pending     *unicodebstr
*clsid      *events        *modified    *progid      *version
*data        help          *modifystop  *properties  *xMainProps

Methods
-------
*Close      *Open           *zExists                *zReplace_2
*Create     *Ref            *zGetAttributes         *zSetAttributes
*Defer      *Send           *zGetCreationTime       *zSetCreationTime
*Delete     *Set            *zGetCreationTimeUtc    *zSetCreationTimeUtc
*EnumEnd    *SetLinks       *zGetLastAccessTime     *zSetLastAccessTime
*EnumNext   *zAppendAllText *zGetLastAccessTimeUtc  *zSetLastAccessTimeUtc
*EnumStart  *zCopy          *zGetLastWriteTime      *zSetLastWriteTime
*Event      *zCopy_2        *zGetLastWriteTimeUtc   *zSetLastWriteTimeUtc
*Exec       *zDecrypt       *zMove                  *zWriteAllBytes
*Info       *zDelete        *zReadAllBytes          *zWriteAllLines
*Modify     *zDoc           *zReadAllLines          *zWriteAllText
*New        *zEncrypt       *zReadAllText
 New        *zEquals        *zReplace

Events
------
```

```
*onAction  *onClose  *onDelete  *onModified  *onOpen  *onReopen  *onSend
```

Then we can start using the **znetFile** object:

```
      filePath←"C:\Temp\TestFile.txt"
      data←"C# Corner MVP & Microsoft MVP;"
      'file'⎕wi'WriteAllText'filePath data
Unknown Method: WriteAllText. Did you intend to use: *WriteAllText  instead?
```

Oops! We forgot to prefix the method with ***z**. Let's try again:

```
      'file'⎕wi'*zWriteAllText'filePath data
```

Check that a file named **TestFile.txt** has been created in your **c:\temp** folder.

Let's read the entire content of the file:

```
      'file'⎕wi'*zReadAllText'filePath
C# Corner MVP & Microsoft MVP;
```

Let's write a nested vector of strings to the file (one string per line):

```
      data←"MCT" "MCPD" "MCTS" "MCSD.NET" "MCAD.NET" "CSM"
      'file'⎕wi'*zWriteAllLines'filePath data
```

And let's read back all lines from the file:

```
      'file'⎕wi'*zReadAllText'filePath
MCT

MCPD

MCTS

MCSD.NET

MCAD.NET

CSM
```

Note that each result line ends with ⎕tcnl,⎕tclf.

If we want to get rid of the ⎕tclf's:

```
      ⎕tclf~⍨'file'⎕wi'*zReadAllText'filePath
MCT
MCPD
MCTS
MCSD.NET
MCAD.NET
CSM

      data←"Also Certified from IIT Kharagpur"
      'file'⎕wi'*zAppendAllText'filePath data
      ⎕tclf~⍨'file'⎕wi'*zReadAllText'filePath
```

Copyright © 2016 Eric Lescasse

```
MCT
MCPD
MCTS
MCSD.NET
MCAD.NET
CSM
Also Certified from IIT Kharagpur

      otherData←"Worked with Microsoft" "Lived in USA"
      'file'⎕wi'*zAppendAllLines'filePath otherData
⎕WI ACTION ERROR: Action "zAppendAllLines" not found
      'file'⎕wi'*zAppendAllLines'filePath otherData
            ^
```

Unfortunately, **zAppendAllLines** is one of the methods we can't use from APL.

Let's copy the file:

```
      'file'⎕wi'*zCopy'filePath'c:\temp\data.txt'0
LENGTH ERROR: Wrong number of arguments
      'file'⎕wi'*zCopy'filePath'c:\temp\data.txt'0
            ^
```

OK: this time the problem is that the .Net Framework often includes several overlays for a given method. In C#, it is possible to have several methods (equivalent to an APL functions) that have the same names but differ by the number and/or types of arguments they use.

In that case, the methods are named:

```
Copy
Copy_2
Copy_3
…
Copy_N
```

Remember that we can get documentation about anything at any time:

```
      'file'⎕wi'?Copy'
Syntax: 'obj'⎕wi'*zCopy'sourceFileName destFileName
Summary: Copies an existing file to a new file. Overwriting a file of the same
name is not allowed
sourceFileName: The file to copy
destFileName: The name of the destination file. This cannot be a directory or an
existing file

Syntax: 'obj'⎕wi'*zCopy_2'sourceFileName destFileName overwrite
Summary: Copies an existing file to a new file. Overwriting a file of the same
name is allowed
sourceFileName: The file to copy
destFileName: The name of the destination file. This cannot be a directory
overwrite: true if the destination file can be overwritten; otherwise, false
```

So let's try again copying the file, using the right method:

```
      'file'⎕wi'*zCopy_2'filePath'c:\temp\data.txt'0
```

Copyright © 2016 Eric Lescasse

```
      'file'⎕wi'*zReadAllLines' 'c:\temp\data.txt'
 MCT MCPD MCTS MCSD.NET MCAD.NET CSM Also Certified from IIT Kharagpur
```

Great the file has been copied ok.

Let's delete the copied file:

```
      'file'⎕wi'*zDelete' 'c:\temp\data.txt'
      'file'⎕wi'*zReadAllLines' 'c:\temp\data.txt'
⎕WI ERROR: mscorlib exception -2147024894 (0x80070002) Could not find file
'c:\temp\data.txt'.
      'file'⎕wi'*zReadAllLines' 'c:\temp\data.txt'
            ^
```

The file has been correctly deleted: we can't read it any more.

Let's check for sure:

```
      'file'⎕wi'*zExists' 'c:\temp\data.txt'
0
      'file'⎕wi'*zExists' 'c:\temp\testFile.txt'
1
```

When did we create the testFile.txt file?

```
      'file'⎕wi'*zGetCreationTime' 'c:\temp\testFile.txt'
20160125.2011209
```

The **GetCreationTime** .Net method returns a **DateTime** object but the zObject DateTime converter converts it to **yyyymmdd.hhmmssx** where x is 10^{th} of a second.

Let's get more information on the testFile.txt file:

```
      ←'info'⎕wi'*Create' 'znetFileInfo'('*zInit' 'c:\temp\testFile.txt')
```

Note that we have had to use the zInit method to specify the file for which we want to get information. The problem is that the .Net FileInfo class require a constructor parameter: the file name. In APL the Create method cannot accept a parameter: this kind of problem is solved in zObjects.dll by calling an additional method (here zInit), often called zCreate with the required parameter.

Let's see what we can do with a **znetFileInfo** object:

```
      'info'⎕wi'allprops'
Properties
----------
*apldata       *events      *opened       *zAttributes       *zLastAccessTimeUtc
*children       help        *pending      *zCreationTime     *zLastWriteTime
*class         *instance    *progid       *zCreationTimeUtc  *zLastWriteTimeUtc
 class         *interface   *properties   *zDirectory        *zLength
*clsid         *links       *self         *zDirectoryName    *zName
*data          *methods     *state        *zExists           *ΔΔisnetclass
*def           *modified    *suppress     *zExtension
*description   *modifystop  *unicodebstr  *zFullName
```

```
*errorcode     *name        *version      *zIsReadOnly
*errormessage  *obj         *xMainProps   *zLastAccessTime

Methods
-------
*Close      *Info      *zCopyTo              *zInit
*Create     *Modify    *zCopyTo_2            *zInitializeLifetimeService
*Defer      *New       *zDecrypt             *zMoveTo
*Delete      New       *zDelete              *zRefresh
*EnumEnd    *Open      *zDoc                 *zReplace
*EnumNext   *Ref       *zEncrypt             *zReplace_2
*EnumStart  *Send      *zEquals              *zToString
*Event      *Set       *zGetHashCode
*Exec       *SetLinks  *zGetLifetimeService

Events
------
*onAction  *onClose  *onDelete  *onModified  *onOpen  *onReopen  *onSend
```

Let's query a lot of information about the **testFile.text** file:

```
      'info'⍁wi'*zCreationTime'
20160125.2011209
      'info'⍁wi'*zCreationTimeUtc'
20160125.1911209
      'info'⍁wi'*zDirectory'
 20141213.213941 20141213.203941 1  c:\temp 20160125.2211343 20160125.2111343 20
      160125.2211343 20160125.2111343 temp c:\ c:\
      'info'⍁wi'*zDirectoryName'
c:\temp
      'info'⍁wi'*zExists'
1
      'info'⍁wi'*zExtension'
.txt
      'info'⍁wi'*zFullName'
c:\temp\testFile.txt
      'info'⍁wi'*zIsReadOnly'
0
      'info'⍁wi'*zLastAccessTime'
20160125.2011209
      'info'⍁wi'*zLastAccessTimeUtc'
20160125.1911209
      'info'⍁wi'*zLength'
75
      'info'⍁wi'*zName'
testFile.txt
```

Note that the **zDirectory** property returns a nested vector: in fact, C# returns a **Directory** object, but the **zObjects.dll** converts it to a nested array containing information about the Directory. Let's get the documentation:

```
      'info'⍁wi'?Directory'
Syntax: directory←'obj'⍁wi'*zDirectory'
Summary: Gets an instance of the parent directory
directory: A System.IO.DirectoryInfo object representing the parent directory
      of this file
The result is a 12-elements nested vector containing:
```

```
[1]= creation time (yyyymmdd.hhmmsst)
[2]= creation time UTC (yyyymmdd.hhmmsst)
[3]= exists (0 or 1)
[4]= extension (i.e. .ext)
[5]= full name
[6]= last access time (yyyymmdd.hhmmsst)
[7]= last access time UTC (yyyymmdd.hhmmsst)
[8]= last write time (yyyymmdd.hhmmsst)
[9]= last write time UTC (yyyymmdd.hhmmsst)
[10]= name
[11]= parent full name
[12]= root name
```

As you can see, it is pretty simple to use .Net zObjects.

One might say that he could achieve the same results using only APL expressions.

Yes, for some methods or properties: for example, the equivalent of:

```
'file'⎕wi'*zReadAllLines' 'c:\temp\testFile.txt'
```

would be:

```
'c:\temp\testFile.txt'⎕xntie t←¯1+⌈/⎕xnnums,⎕nnums,0
data←⎕tclf~⍨⎕nread t 82,2↑⎕nsize t
⎕nuntie t
1↓¨(data=⎕tcnl)⊂penclose data←⎕tcnl,data
```

Not only using the APL solution is much more complex, but it forces you to create variables and is prone to errors. For example, you might forget to close the file and leave it opened! Or you could make a programming error.

Additionally, there are a good number of things that the znetFile or znetFileInfo do, which would be difficult to do in APL, like retrieving the file creation time or its UTC creation time or last access time, or encrypting or decrypting the file.

Finally, the .Net Framework is:

- A Framework that has been very carefully designed
- Is extremely efficient
- Is most of the time easy to use
- Is consistent

Very simple things like the DirectoryName property we have seen earlier is something I have seen APL developers write over and over again: some include a terminal backslash, others don't. If you are an APL developer, you probably have written a utility (probably even several times in your programmer's life) that extracts the directory name from a full path file name. Are you sure that all the versions you have written are consistent and all included a terminal backslash or do not?

Copyright © 2016 Eric Lescasse

Also, why write something over and over again when we can use a .Net Framework property or method that is guaranteed to do the job perfectly, more efficiently and more consistently.

That is just a very simple example, but it is true for a lot of things we do as an APL developer. Learning to use the .Net Framework can make our life much simpler and result in faster and easier to maintain applications.

Using .Net Windows Forms Controls in your APL+Win forms

As mentioned earlier, zObjects include a good number of Windows Forms controls that you can use in your APL+Win zObjects forms.

Use a zForm as a host

First, it is important to remember that you must use a **zForm** (or **zMDIForm)** if you want to host one or more .Net zObject Windows Forms controls.

Moreover, you must only use zObjects in your zForm: this guarantees that your .Net zObject will behave like all other zObject controls and notably will support the **wherelc** and **anchor** properties which are essential for easy design of your forms.

Use the standard APL+Win dot notation

When you add a control to a Windows Forms application, the .Net Framework standard behavior is that:

1. You create an instance of your control (without any parent)
2. You initialize control properties
3. You then add the instance of your control to its parent **Controls** collection

The standard APL+Win way of adding a control to a form is to simply create it as a child of its parent, using the dot notation.

Example:

```
←'ff'⎕wi'*Create' 'zForm'('DemoShow'.4 .4 1 1)
←'ff.ed1'⎕wi'*Create' 'zEdit'
```

zObjects simplifies the use of .Net Windows Forms controls into an APL+Win Form by allowing you to use the same APL syntax you've been using for years to build your APL+Win User Interfaces.

Example:

```
←'ff'⎕wi'*Create' 'zForm' ('DemoShow'.4 .4 1 1)
←'ff.ed1'⎕wi'*Create' 'znetTextBox'
```

Copyright © 2016 Eric Lescasse

Use whereIc and anchor with .Net zObjects

Yes, you can use the **whereIc** and **anchor** properties at any time with the .Net zObjects you use in your zForms. This is very handy since you basically use the same zObject "frame-work" all over the place, whether you use APL+Win zObjects or .Net zObjects in your form and you can mix the two.

Example:

```
      ←'ff'⎕wi'*Create' 'zForm'('DemoShow'.4 .4 1 1)
      ←'ff.wb1'⎕wi'*Create' 'znetWebBrowser'('whereIc'0 0'>' '>')('anchor' 'tlrb')
      ←'ff.wb1'⎕wi'*zNavigate' 'http://www.microsoft.com'
```

Now try resizing the form: the znetWebBrowser resizes perfectly.

Rules

There is a rule that you must abide to, however, when you add .Net zObjects to your APL+Win forms.

Once you have added a .Net zObject to your zForm, its children can only be .Net zObjects.

In other words, you cannot create an APL+Win zObject as a child of a .Net zObject.

More complex example

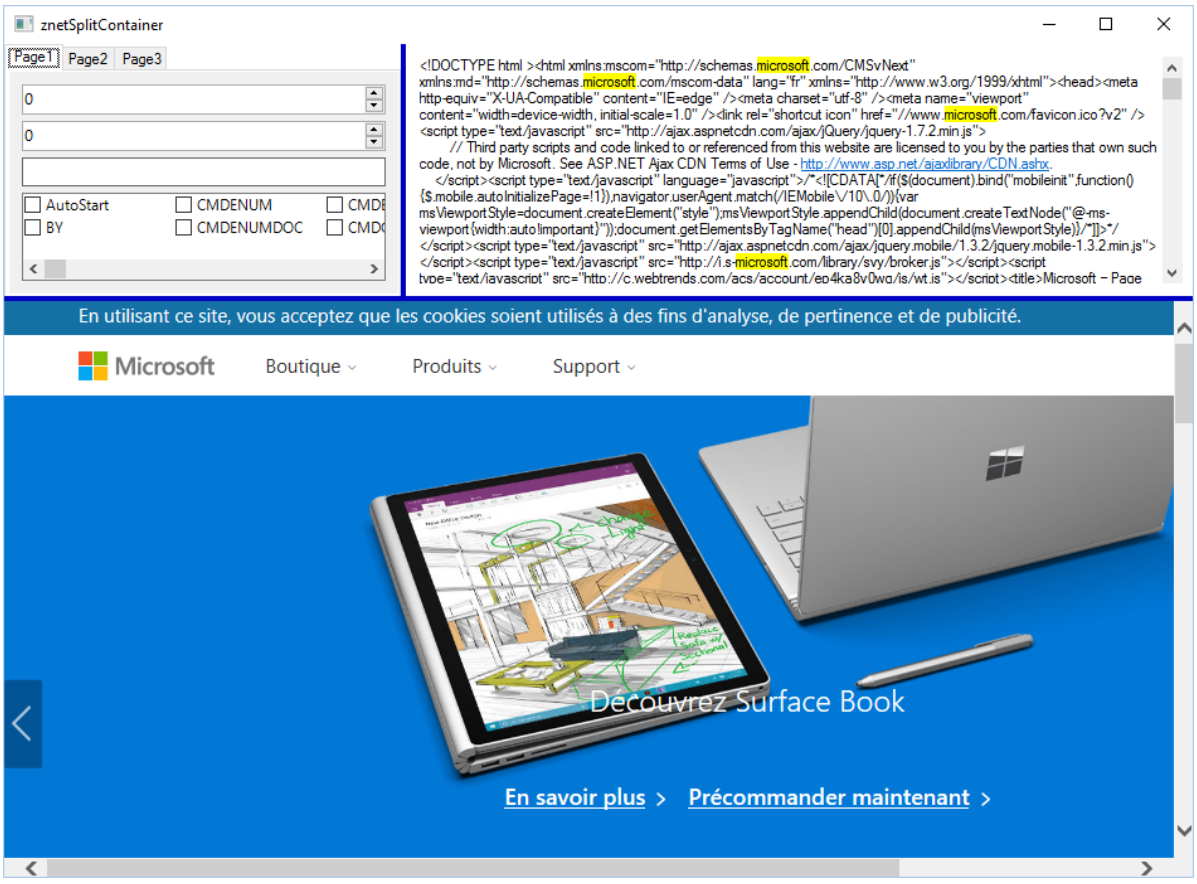
The zznetsplitcontainer function demonstrates how you can use .Net zObjects in your user interface.

Carefully study the following function:

```
    ∇ zznetsplitcontainer
[1]   ⍝∇ zznetsplitcontainer -- Demonstrates how to use the znetSplitContainer object
[2]
[3]
[4]
[5]   ←'ff'⎕wi'*Create' 'zForm'('*caption' 'znetSplitContainer')
[6]
[7]   ←'ff.sc1'⎕wi'*Create' 'znetSplitContainer'('whereIc'0 0'>>' '>>')
            ('anchor'1 2 3 4)('*zborderstyle'0)('*zorientation'0)
            ('*zsplitterwidth'4)('*zbackcolor'0 0 192)
[8]   ←'ff.sc1'⎕wi'*zfixedpanel'1
[9]   ←'ff.sc1'⎕wi'*zsplitterdistance'200
[10]  ←'ff.sc1.sc1p1'⎕wi'*Create' 'znetSplitterPanel'('whereIc'0 0'>>' '>>')
            ('anchor'1 2 3 4)('*zbackcolor'255 255 255)
[11]  ←'ff.sc1.sc1p1.sc2'⎕wi'*Create' 'znetSplitContainer'('whereIc'0 0'>>' '>>')
            ('anchor'1 2 3 4)('*zborderstyle'0)('*zdock'5)
            ('*zorientation'1)('*zsplitterwidth'4)('*zbackcolor'0 0 192)
[12]  ←'ff.sc1.sc1p1.sc2.sc2p1'⎕wi'*Create' 'znetSplitterPanel'('whereIc'0 0'>>' '>>')
            ('anchor'1 2 3 4)('*zbackcolor'255 255 255)
[13]  ←'ff.sc1.sc1p1.sc2.sc2p1.tb1'⎕wi'*Create' 'znetTabControl'
            ('whereIc'0 0'>>' '>>'0 0 2 2)('anchor'1 2 3 4)
[14]  ←'ff.sc1.sc1p1.sc2.sc2p1.tb1.tb1p1'⎕wi'*Create' 'znetTabPage'('*ztext' 'Page1')
[15]  ←'ff.sc1.sc1p1.sc2.sc2p1.tb1.tb1p2'⎕wi'*Create' 'znetTabPage'('*ztext' 'Page2')
```

```
[16]  ←'ff.sc1.sc1p1.sc2.sc2p1.tb1.tb1p3'⍞wi'*Create' 'znetTabPage'('*ztext' 'Page3')
[17]  ←'ff.sc1.sc1p1.sc2.sc2p1.tb1.tb1p1.nud1'⍞wi'*Create' 'znetNumericUpDown'
            ('wherelc'000'>')('anchor'1 2 3 2)
[18]  ←'ff.sc1.sc1p1.sc2.sc2p1.tb1.tb1p1.nud2'⍞wi'*Create' 'znetNumericUpDown'
            ('wherelc' '>' '=' '=' '=')('anchor'1 2 3 2)
[19]  ←'ff.sc1.sc1p1.sc2.sc2p1.tb1.tb1p1.mtb1'⍞wi'*Create' 'znetMaskedTextBox'
            ('wherelc' '>' '=' '=' '=')('anchor'1 2 3 2)
[20]  ←'ff.sc1.sc1p1.sc2.sc2p1.tb1.tb1p1.clb1'⍞wi'*Create' 'znetCheckedListBox'
            ('wherelc' '>' '=' '>' '>')('anchor'1 2 3 4)('*zitems'(⍞cn ⍞nl 3))
            ('*zmulticolumn'1)('*zcheckonclick'1)
[21]  ←'ff.sc1.sc1p1.sc2.sc2p2'⍞wi'*Create' 'znetSplitterPanel'
            ('wherelc'0 0'>>' '>>')('anchor'1 2 3 4)('*zbackcolor'255 255 255)
[22]  ←'ff.sc1.sc1p1.sc2.sc2p2.rtb'⍞wi'*Create' 'znetRichTextBox'
            ('wherelc'10 10'>>' '>>'0 0 ¯10 ¯10)('anchor'1 2 3 4)
[23]  ←'ff.sc1.sc1p2'⍞wi'*Create' 'znetSplitterPanel'('wherelc'0 0'>>' '>>')
            ('anchor'1 2 3 4)('*zbackcolor'255 255 255)
[24]  ←'ff.sc1.sc1p2.wb'⍞wi'*Create' 'znetWebBrowser'('wherelc'0 0'>>' '>>')
            ('anchor'1 2 3 4)
[25]  ←'ff.sc1.sc1p2.wb'⍞wi'*onXDocumentCompleted' '←"rtb"⍞wi"*ztext"
            ("wb"⍞wi"*zdocumenttext")⋄←"rtb"⍞wi"*zHighlightText" "microsoft"(255 255
            0)'
[26]  ←'ff.sc1.sc1p2.wb'⍞wi'*znavigate' 'http://www.microsoft.com'
[27]
[28]  ←'ff'⍞wi'DemoShow'.5 .5 1 1
[29]  ⍞wself←'ff.sc1'
    ∇
```

It creates the following User Interface, using many .Net zOjects:



This simple application (less than 30 lines of code) creates a form with:

1. A **znetSplitContainer** object (sc1)
2. Which itself contains 2 **znetSplitterPanel** objects (sc1p1 and sc1p2)
3. The first **znetSplitterPanel** (sc1p1) itself contains a **znetSplitContainer** object (sc2)
4. Which itself contains 2 **znetSplitterPanel** objects (sc2p1 and sc2p2)
5. The sc2p1 **znetSplitterPanel** contains a **znetTabControl** object (tb1)
6. The **znetTabControl** object contains 3 **znetTabPage** objects (tb1p1, tb1p2 and tb1p3)
7. The first **znetTabPage** (tb1p1) contains:
    - 2 **znetNumericUpDown** controls (nud1 and nud2)
    - 1 **znetMaskedTextBox** control (mtb1)
    - 1 **znetCheckedListBox** control (clb1)
8. The sc2p1 **znetSplitterPanel** contains a **znetRichTextBox** control (rtb)
9. Finally, the sc1p2 **znetSplitterPanel** contains a **znetWebBrowser** control (wb)

So, this **zForm** contains the following .Net zObject controls:

- 2 znetSplitContainer
- 4 znetSplitterPanel
- 1 znetTabControl
- 3 znetTabPage
- 2 znetNumericUpDown
- 1 znetMaskedTextBox
- 1 znetCheckedListBox
- 1 znetRichTextBox
- 1 znetWebBrowser

All these controls are perfectly positioned and dimensioned in the form.

Try resizing the form: you'll see that all the control it contains automatically resize perfectly.

Note only has it been possible to do that with only 30 lines of APL code, but this application even has some behavior built in:

1. When it starts, it opens and displays the Microsoft Home Page in the **znetWebBrowser** object

2. As soon as the Microsoft Home page is displayed the **znetRichTextBox** displays the source code for the Microsoft Home page

3. Not only that, but the word Microsoft is automatically highlighted in yellow everywhere in the **znetRichTextBox**

Copyright © 2016 Eric Lescasse

Try to imagine how much code, time and effort it would have required to write the same application with just raw APL+Win code! The result would sure be a several hundred lines function.

In fact, it would not even have been possible or at least would have been very difficult to write this application in raw APL+Win since APL+Win does not have a **MaskedTextBox** object and does not have a **CheckedListBox** object.

Let's now review all the available .Net zObjects and their most important properties and methods.

Copyright © 2016 Eric Lescasse

Sample function using a znetButton

Here is a function that creates a **zForm** with a **znetButton** and sets many of the most important znetButton properties:

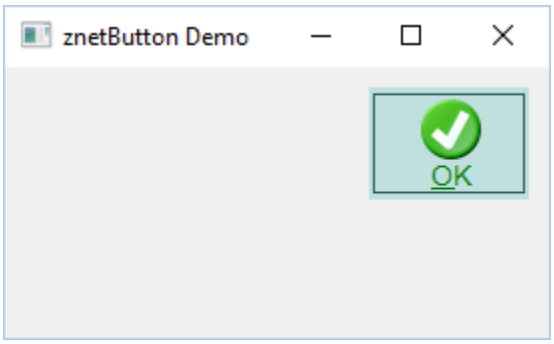
```
    ∇ zznetbutton
[1]   ⍝∇ This function demonstrates how to use the znetButton class
[2]
[3]   ←'ff'⎕wi'*Create' 'zForm'('*caption' 'znetButton Demo')('DemoShow'.15 .15 1)
[4]   ←'ff'⎕wi'*.bn1.Create' 'znetButton'('wherelc'0'<'56 80)('anchor' 'tr')
[5]   ⍝ Events
[6]   ←'bn1'⎕wi'*onXClick' '⎕←⎕wself⎕wevent⎕warg'
[7]   ←'bn1'⎕wi'*onXGotFocus' '⎕←⎕wself⎕wevent⎕warg'
[8]   ←'bn1'⎕wi'*onXLostFocus' '⎕←⎕wself⎕wevent⎕warg'
[9]   ←'bn1'⎕wi'*onXMouseEnter' '⎕←⎕wself⎕wevent⎕warg'
[10]  ←'bn1'⎕wi'*onXMouseMove' '⎕←⎕wself⎕wevent⎕warg'
[11]  ←'bn1'⎕wi'*onXMouseLeave' '⎕←⎕wself⎕wevent⎕warg'
[12]  ←'bn1'⎕wi'*onXTextChanged' '⎕←⎕wself⎕wevent⎕warg'
[13]  ←'bn1'⎕wi'*onXBackColorChanged' '⎕←⎕wself⎕wevent⎕warg'
[14]  ⍝ Properties
[15]  ←'bn1'⎕wi'*zText' '&OK'
[16]  ←'bn1'⎕wi'*zUseMnemonic'1                ⍝ use O as a mnemonic
[17]  ←'bn1'⎕wi'*zBackColor'192 224 224
[18]  ←'bn1'⎕wi'*zForeColor'0 128 0
[19]  ←'bn1'⎕wi'*zTextAlign'2                  ⍝ 1=TopLeft 2=TopCenter 4=TopRight ...
[20]  ←'bn1'⎕wi'*zTextImageRelation'1          ⍝ 0=overlay 1=imageabove
            2=textabove 4=imagebefore 8=textbefore
[21]  ←'bn1'⎕wi'*zImageAlign'2                 ⍝ 1=TopLeft 2=TopCenter 4=TopRight ...
[22]  ←'bn1'⎕wi'*zImage' 'OK32np'
[23]  ←'bn1'⎕wi'*zPadding'0
[24]  ←'bn1'⎕wi'*zBorderSize'0
[25]  ←'bn1'⎕wi'*zFont' 'Arial'14 0
[26]  ←'bn1'⎕wi'*zEnabled'1
    ∇
```

Let's run this function: the APL Session displays:

```
ff.bn1 XTextChanged
ff.bn1 XBackColorChanged
```

and the following form gets displayed:

Copyright © 2016 Eric Lescasse



Click on the button: the APL session should display something like:

```
ff.bn1 XMouseEnter
ff.bn1 XMouseMove  0 0 2 47 0  47 2
ff.bn1 XMouseMove  0 0 7 41 0  41 7
ff.bn1 XMouseMove  0 0 12 36 0  36 12
ff.bn1 XMouseMove  0 0 13 35 0  35 13
ff.bn1 XMouseMove  0 0 16 31 0  31 16
ff.bn1 XMouseMove  0 0 18 28 0  28 18
ff.bn1 XMouseMove  0 0 19 27 0  27 19
ff.bn1 XMouseMove  0 0 19 26 0  26 19
ff.bn1 XMouseMove  0 0 19 25 0  25 19
ff.bn1 XMouseMove  0 0 20 25 0  25 20
ff.bn1 XGotFocus
ff.bn1 XMouseMove  1048576 0 20 25 0  25 20
ff.bn1 XClick
ff.bn1 XMouseMove  0 0 20 25 0  25 20
```

You may also call one of the button methods, for example **zPerformClick**:

```
      'bn1'□wi'*zPerformClick'
ff.bn1 XClick
```

Note that this automatically fires the **onXClick** event for the button.

Images

Use the **zznetselecticonform** function to select the **16x16** or **32x32** image you want to display in your **znetButton**.

Main properties, methods and events

```
      'bn1'□wi'mainprops'
Main Properties
---------------
*zAutoEllipsis  *zFlatStyle   *zImageKey           *zUseCompatibleTextRendering
*zBorderColor   *zImage       *zImageSize          *zUseMnemonic
*zBorderSize    *zImageAlign  *zTextAlign          *zUseVisualStyleBackColor
*zDialogResult  *zImageIndex  *zTextImageRelation

Main Methods
------------
*zDoc  *zNotifyDefault  *zPerformClick
```

Copyright © 2016 Eric Lescasse

Main Events

The znetButton16 and znetButton32 objects

Description

The **znetButton16** and **znetButton32** zObjects do not directly derive from .Net objects. They instead derive from **znetButton**.

A **znetButton16** is a **znetButton** made to display a **16x16** icon, with no text.

A **znetButton32** is a **znetButton** made to display a **32x32** icon, with no text.

Both of these objects have the following characteristics:

1. They don't have a border (**BorderSize**=0)
2. The have an empty **Text** property
3. They have a **size** automatically set to best fit the image they host
4. They have a **FlatStyle** of 0 (=FlatStyle.Flat)
5. They have a proper **Padding** of 0
6. They have an **ImageAlign** property set to best center the image in the button

This makes it very easy to add a series of such buttons directly on a **zForm** or some other container (**zLabel**, **zFrame**, …) to create nice toolbars in an application.

Sample function using znetButton16 and znetButton32

Here is a function demonstrating how to use **znetButton16** and **znetButton32** buttons:

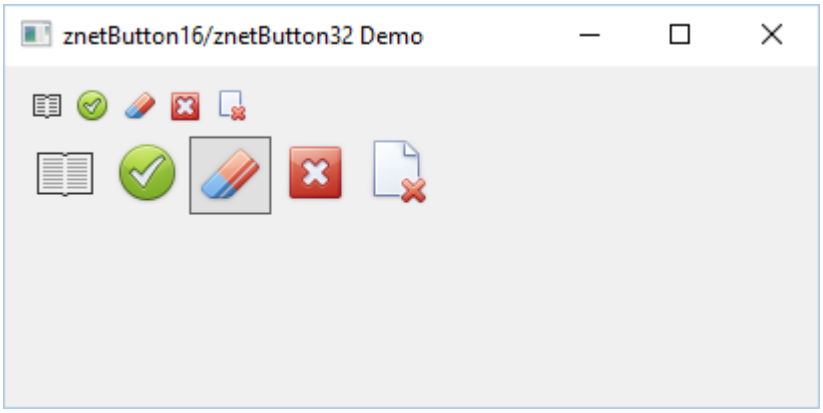
```
    ∇ zznetbutton1632
[1]   ⍝∇ This function demonstrates how to use the znetButton16 and znetButton32
objects
[2]
[3]   ←'ff'⎕wi'*Create' 'zForm'('*where'¯5000 ¯5000)
            ('*caption' 'znetButton16/znetButton32 Demo')
[4]
[5]   ←'ff'⎕wi'*.bn1.Create' 'znetButton16'('wherelc'0000)
            ('*zImage' 'Reading_16x16')
[6]   ←'ff'⎕wi'*.bn2.Create' 'znetButton16'('wherelc' '=' '>>' '=' '='0 2)
            ('*zImage' 'Apply_16x16')
[7]   ←'ff'⎕wi'*.bn3.Create' 'znetButton16'('wherelc' '=' '>>' '=' '='0 2)
            ('*zImage' 'Clear_16x16')('toggle'1)
[8]   ←'ff'⎕wi'*.bn4.Create' 'znetButton16'('wherelc' '=' '>>' '=' '='0 2)
            ('*zImage' 'Close_16x16')
[9]   ←'ff'⎕wi'*.bn5.Create' 'znetButton16'('wherelc' '=' '>>' '=' '='0 2)
            ('*zImage' 'DeleteList_16x16')('toggle'1)
[10]
[11]  ←'ff'⎕wi'*.bn11.Create' 'znetButton32'('wherelc' '>bn1' '='00)
            ('*zImage' 'Reading_32x32')
[12]  ←'ff'⎕wi'*.bn12.Create' 'znetButton32'('wherelc' '=' '>>' '=' '='0 2)
            ('*zImage' 'Apply_32x32')
[13]  ←'ff'⎕wi'*.bn13.Create' 'znetButton32'('wherelc' '=' '>>' '=' '='0 2)
            ('*zImage' 'Clear_32x32')('toggle'1)('value'1)
```

Copyright © 2016 Eric Lescasse

```
[14]  ←'ff'⎕wi'*.bn14.Create' 'znetButton32'('whereIc' '=' '>>' '=' '=0 2)
            ('*zImage' 'Close_32x32')
[15]  ←'ff'⎕wi'*.bn15.Create' 'znetButton32'('whereIc' '=' '>>' '=' '=0 2)
            ('*zImage' 'DeleteList_32x32')
[16]
[17]  ((⊂'bn'),¨⍕¨,0 10∘.+⍳5)⎕wi¨⊂'AddHandler' '*onXClick' '⎕←" clicked",
            ⍕¨⎕wi"*name"'
[18]
[19]  ←'ff'⎕wi'DemoShow'.18 .22 1 0
    ▽
```

Running the above function creates the following user interface:

```
      zznetbutton1632
```



Click some buttons: you should see something like the following being printed to the APL session:

```
bn2 clicked
bn4 clicked
bn13 clicked
bn14 clicked
```

Note that a **znetButton16** or **znetButton32** object gets a darker gray background when you hover your mouse over it.

Images

Use the **zznetselecticonform** function to select the **16x16** or **32x32** images you want to display in your **znetButton16** and **znetButton32** objects.

Main properties, methods and events

Since **znetButton16** and **znetButton32** objects derive from **znetButton**, they inherit all the **znetButton** properties, methods and events (see **znetButton**).

They have the following additional properties:

toggle	boolean scalar; if 1 the button is a toggle button (example: a Bold button)

Copyright © 2016 Eric Lescasse

value	boolean scalar: if 1, the button is checked, if 0 it is unchecked A checked button has a border and a slightly darker background

Copyright © 2016 Eric Lescasse

The zznetCheckBox object

Description

The **zznetCheckBox** zObject allows you to use a Windows Forms **CheckBox** object in your APL applications.

The zznetCheckBox is similar to an APL+Win Check control, but:

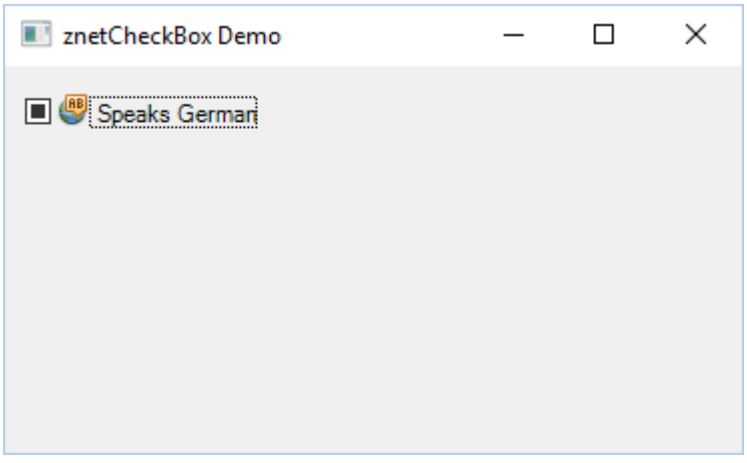
- Allows to display an image
- Has more properties, methods and events
- Can be programmatically transformed into a button

Sample function using a zznetCheckBox

The following function demonstrates how to use a zznetCheckBox:

```
    ∇ zznetcheckbox
[1]   ⍝∇ This function demonstrates how to use the znetCheckBox object
[2]
[3]   ←'ff'⎕wi'*Create' 'zForm'('*caption' 'znetCheckBox Demo')
[4]   ←'ff'⎕wi'*.ck.Create' 'znetCheckBox'('wherelc'0 0 0 120)
            ('*zText' 'Speaks German')
[5]   ←'ff'⎕wi'*.ck.Set'('*zThreeState'1)('*zImage' 'Language_16x16')
            ('*zTextImageRelation'4)
[6]   ←'ff'⎕wi'*.ck.onXAppearanceChanged' '⎕←"onXAppearanceChanged:
            ⎕warg=",⍕⎕warg'
[7]   ←'ff'⎕wi'*.ck.onXCheckStateChanged' '⎕←"onXCheckStateChanged:
            ⎕warg=",⍕⎕warg'
[8]   ←'ff'⎕wi'*.ck.onXCheckedChanged'  '⎕←"onXCheckedChanged: ⎕warg=",⍕⎕warg'
[9]   ←'ff'⎕wi'DemoShow'.2 .2 1 0
    ∇
```

Running this functions displays the following user interface:



Clicking several times on the **zznetCheckBox**, displays the following in the APL session:

Copyright © 2016 Eric Lescasse

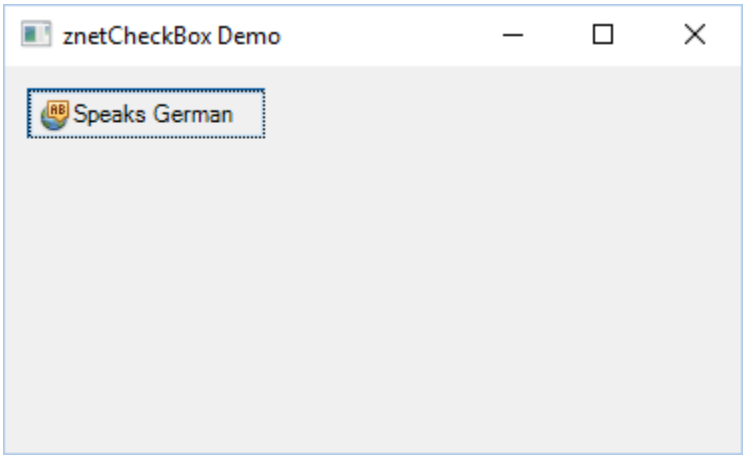
```
onXCheckedChanged: ⎕warg=
onXCheckStateChanged: ⎕warg=
onXCheckStateChanged: ⎕warg=
onXCheckedChanged: ⎕warg=
onXCheckStateChanged: ⎕warg=
```

To transform the **zznetCheckBox** into a button, change its ***zAppearance** property to **1**:

```
      'ck'⎕wi'?Appearance'
Syntax: {appearance←}'obj'⎕wi'*zAppearance'{appearance}
Summary: Gets or sets the value that determines the appearance of a CheckBox
      control
appearance: One of the Appearance values. The default value is
      Appearance.Normal
           0 = Appearance.Normal
           1 = Appearance.Button

      'ck'⎕wi'*zAppearance'1
onXAppearanceChanged: ⎕warg=
```

Note that change the zAppearance property fires the onXAppearanceChanged event. Also, we need to change the button height while leaving the button position and width un-changed: this can be easily done as follows:

```
      'ck'⎕wi'wherelc' '∘' '∘'26'∘'
```



Main properties, methods and events

The main zznetCheckBox properties methods and events are:

```
Main Properties
---------------
*zAppearance     *zFlatStyle    *zTextImageRelation
*zAutoCheck      *zImage        *zThreeState
*zAutoEllipsis   *zImageAlign   *zUseCompatibleTextRendering
*zCheckAlign     *zImageIndex   *zUseMnemonic
*zCheckState     *zImageKey     *zUseVisualStyleBackColor
*zChecked        *zTextAlign
```

Copyright © 2016 Eric Lescasse

```
Main Methods
------------
*zDoc

Main Events
-----------
*onXAppearanceChanged  *onXCheckStateChanged  *onXCheckedChanged
```

The znetCheckedListBox object

Description

The **znetCheckedListBox** object lets you use a multi-column list box where each item has a check box attached to it.

Sample function using a znetCheckedListBox

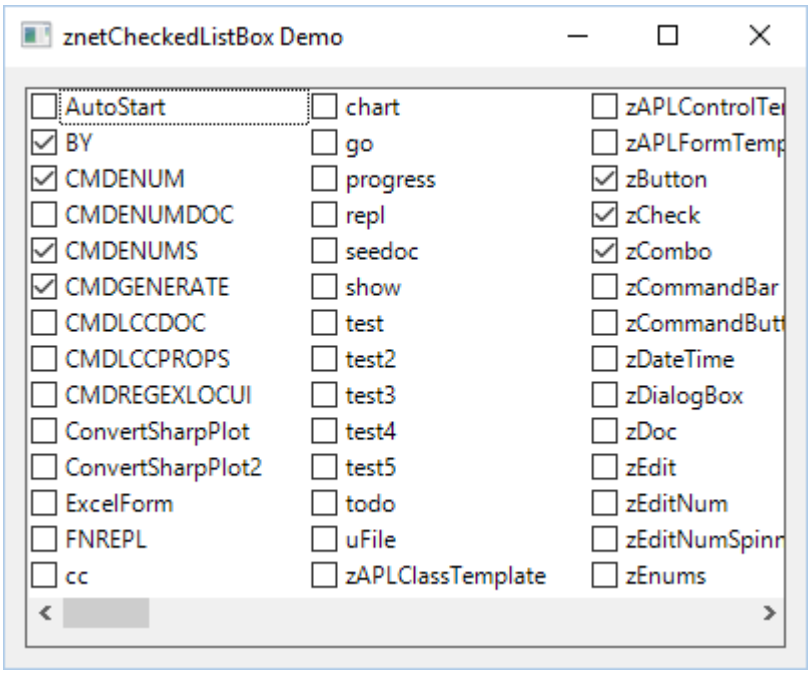
The following functions shows the most important properties and events you want to use for a znetCheckedLisBox:

```
    ∇ zznetcheckedlistbox
[1]   ⍝∇ This function demonstrates how to use the znetCheckedListBox object
[2]
[3]   ←'ff'⎕wi'*Create' 'zForm'('*caption' 'znetCheckedListBox Demo')
[4]   ←'ff'⎕wi'*Set'('*where'¯5000 ¯5000)('*size'300 400)
[5]   ←'ff'⎕wi'*.ck.Create' 'znetCheckedListBox'('wherelc'0 0'>' '>')
            ('anchor' 'lrtb')
[6]   ⍝ ←'ck'⎕wi'*zBorderSize'0       ⍝ no border
[7]   ←'ck'⎕wi'*zitems'(⎕cn ⎕nl 3)    ⍝ populate the control
[8]   ←'ck'⎕wi'*zmulticolumn'1        ⍝ display items in multiple columns
[9]   ←'ck'⎕wi'*zcheckonclick'1       ⍝ otherwise double click required to check
[10]  ←'ck'⎕wi'*zIntegralHeight'1     ⍝ let control resize to avoid partial items
[11]  ←'ck'⎕wi'*zSelectedIndices'1 2 4 5 30 31 32          ⍝ select some items
[12]  ←'ck'⎕wi'*zColumnWidth'140
[13]  ⎕←⊃(⊂'ck')⎕wi¨(⊂'*zSelected'),¨'Indices' 'Items'    ⍝ display selected
            indices & items
[14]  ⍝ Events
[15]  ←'ck'⎕wi'*onXItemCheck' '⎕←⎕wevent⎕warg'
[16]  ←'ck'⎕wi'*onXSelectedIndexChanged' '⎕←⎕wevent⎕warg'
[17]  ←'ck'⎕wi'*onXSelectedValueChanged'  '⎕←⎕wevent⎕warg'
[18]  ←'ck'⎕wi'*onXValueMemberChanged'  '⎕←⎕wevent⎕warg'
[19]  ←'ff'⎕wi'DemoShow'0 0 1 1
[20]  ⎕wself←'ck'
    ∇
```

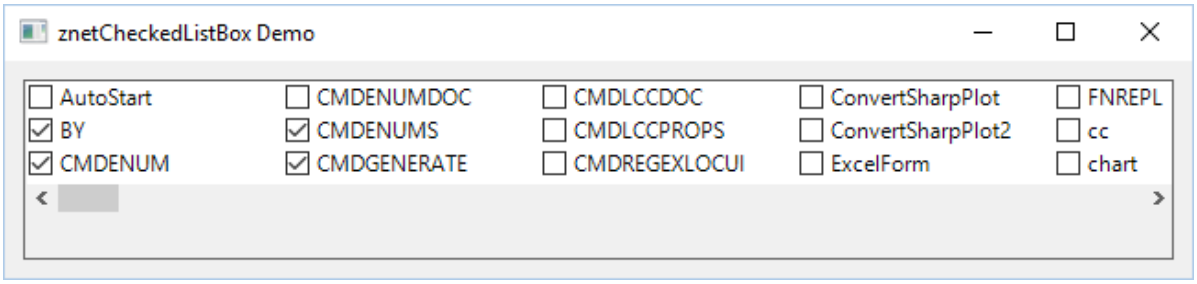
Running this function displays the following form and the following information in the APL+Win Session:

```
      zznetcheckedlistbox
  1       2        4           5      30     31     32
 BY CMDENUM CMDENUMS CMDGENERATE zButton zCheck zCombo
```

Copyright © 2016 Eric Lescasse



Note that the **znetCheckedLisBox** automatically readjusts its content when it gets resized:



Main properties, methods and events

The znetCheckedListBox main properties, methods and events are:

```
      'ck'□wi'mainprops'
Main Properties
---------------
*zCheckOnClick          *zItemHeight            *zSelectionMode
*zColumnWidth           *zItems                 *zSorted
*zDataSource            *zMultiColumn           *zThreeDCheckBoxes
*zDisplayMember         *zPreferredHeight       *zTopIndex
*zDrawMode              *zScrollAlwaysVisible   *zUseCompatibleTextRendering
*zFormatString          *zSelectedIndex         *zUseCustomTabOffsets
*zFormattingEnabled     *zSelectedIndices       *zUseTabStops
*zHorizontalExtent      *zSelectedItem          *zValueMember
*zHorizontalScrollbar   *zSelectedItems
*zIntegralHeight        *zSelectedValue

Main Methods
------------
*zBeginUpdate    *zFindStringExact    *zGetItemRectangle  *zSetItemCheckState
*zDoc            *zFindStringExact_2  *zGetItemText       *zSetItemChecked
```

Copyright © 2016 Eric Lescasse

```
*zEndUpdate      *zGetItemCheckState  *zGetSelected       *zSetSelected
*zFindString     *zGetItemChecked     *zIndexFromPoint
*zFindString_2   *zGetItemHeight      *zIndexFromPoint_2

Main Events
-----------
*onXDataSourceChanged      *onXFormatInfoChanged          *onXMeasureItem
*onXDisplayMemberChanged   *onXFormatStringChanged        *onXSelectedIndexChanged
*onXDrawItem               *onXFormattingEnabledChanged   *onXSelectedValueChanged
*onXFormat                 *onXItemCheck                  *onXValueMemberChanged
```

You can for example find the index of the first item starting with a given string:

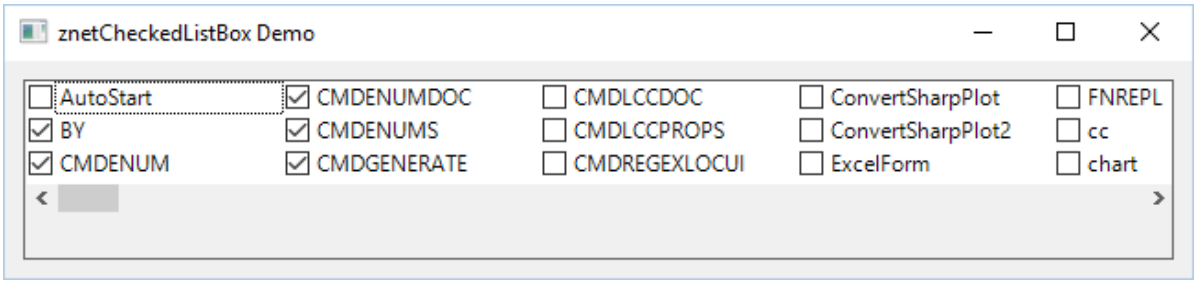
```
      'ck'□wi'*zFindString' 'cmdlcc'
6
```

The search is case insensitive. The **zFindString** returns **-1** if no item starts with the specified string.

Or you can programmatically check an item without changing all the other items check states with:

```
      'ck'□wi'?SetItemChecked'
Syntax: 'obj'□wi'*zSetItemChecked'index value
Summary: Sets CheckState for the item at the specified index to Checked
index: The index of the item to set the check state for
value: true to set the item as checked; otherwise, false

      'ck'□wi'*zSetItemChecked'3 1
 XItemCheck  3 1 0
```

Note that programmatically checking the item has fired the **onXItemCheck** event.



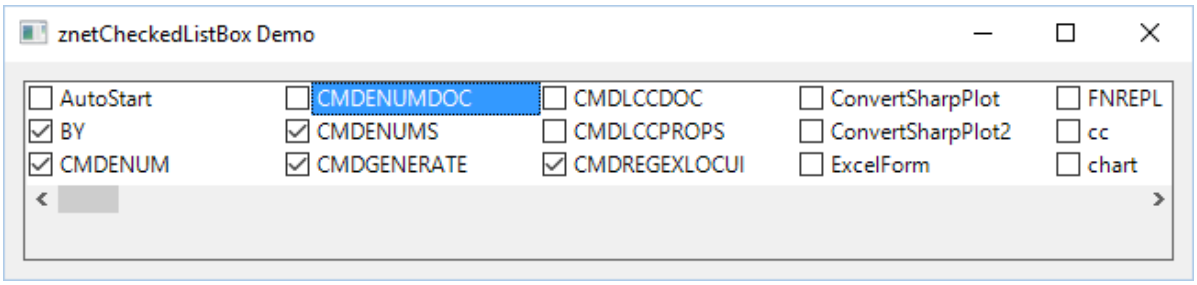
The 4th item: **CMDENUMDOC** (remember indices in .Net are 0-based) has been checked.

You may also query an item to see if it is checked with the zGetItemChecked method:

```
      'ck'□wi'*zGetItemChecked'3
1
```

Or know if a particular item is selected (which is different than being checked):

Copyright © 2016 Eric Lescasse



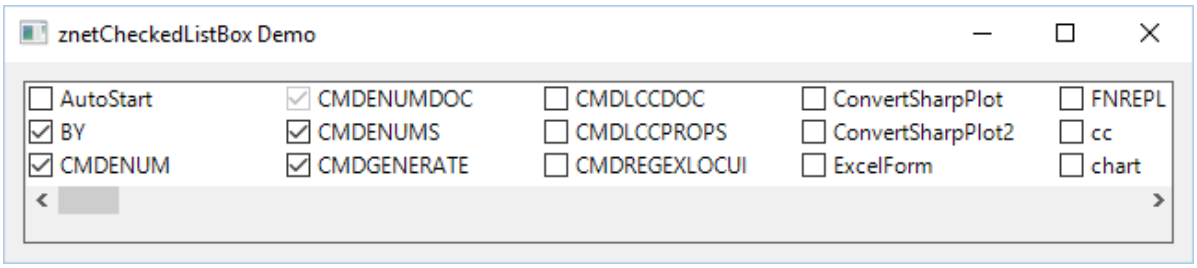

```
      'ck'⍞wi'*zGetSelected'3
1
      'ck'⍞wi'*zGetItemChecked'3
0
```

Always remember that there are many more properties, methods and events available. Use the **allprops** property to know them all.

Checkboxes may have 3 states. The 3rd state is call **indeterminate**.

You can set it or retrieve the state value using the zSetItemCheckState and zGetItemCheck-State methods:

```
      ⍞wi'?SetItemCheckState'
Syntax: 'obj'⍞wi'*zSetItemCheckState'index value
Summary: Sets the check state of the item at the specified index
index: The index of the item to set the state for
value: One of the CheckState values
           0 = CheckState.Unchecked
           1 = CheckState.Checked
           2 = CheckState.Indeterminate

      ⍞wi'*zSetItemCheckState'3 2
 XItemCheck  3 2 0

      ⍞wi'*zGetItemCheckState'3
2
```



A check box in indeterminate state is grayed out.

Copyright © 2016 Eric Lescasse

The znetContextMenu object

Description

The znetContextMenu lets you extremely easily define context menus for your .Net zObjects.

Note that you cannot use znetContextMenu to define a context menu for an APL zObject.

Sample function demonstrating znetContextMenu

The following function show 6 different example of context menus for a znetButton object:

```
    ∇ zznetcontextmenu a
[1]   ⍝∇ This function demonstrates how to use context menus with .Net controls
and forms
[2]
[3]   ←'ff'⎕wi'*Create' 'zForm'
[4]   ←'ff'⎕wi'*.bn.Create' 'znetButton'('*zText' 'Test Context Menu')
            ('wherelc'000140)
[5]   ←'cm'⎕wi'*Create' 'znetContextMenu'('*onXItemClicked'
            '⎕←⎕ucmd"]displayt⎕warg"')
[6]   :select a
[7]   :case 1 ⋄ ←'cm'⎕wi'*zItems' 'Cut'
[8]   :case 2 ⋄ ←'cm'⎕wi'*zItems' 'Cut' 'Copy' 'Paste'
[9]   :case 3 ⋄ ←'cm'⎕wi'*zItems' 'Cut' 'Copy'('Paste'('Paste' 'Paste Special'))
            '-' '/Exit/Alt+F4'
[10]  :case 4 ⋄ ←'cm'⎕wi'*zItems' '/Cut/Ctrl+X'
[11]  :case 5 ⋄ ←'cm'⎕wi'*zItems' '/Cut/Ctrl+X' '/Copy/Ctrl+C' '/Paste/Ctrl+V'
            '-' '/Exit/Alt+F4'
[12]  :case 6 ⋄ ←'cm'⎕wi'*zItems' '/Cut/Ctrl+X' '/Copy/Ctrl+C'('/Paste/Ctrl+V'
            ('/Paste/Ctrl+Shift+V' '/Paste Special/Alt+Shift+V'))'-'
            '/Exit/Alt+F4'
[13]  :endselect
[14]  ←'bn'⎕wi'*zContextMenu'('cm'⎕wi'*zPointer')
[15]  ←'ff'⎕wi'DemoShow'.2 .3 1 0
    ∇
```

The 4 steps for creating a context menu for a .Net zObject are the following:

1. Create your .Net zObject (see line 4)

2. Create a separate **znetContextMenu** object and define its **onXItemClicked** event handler (see line 5)

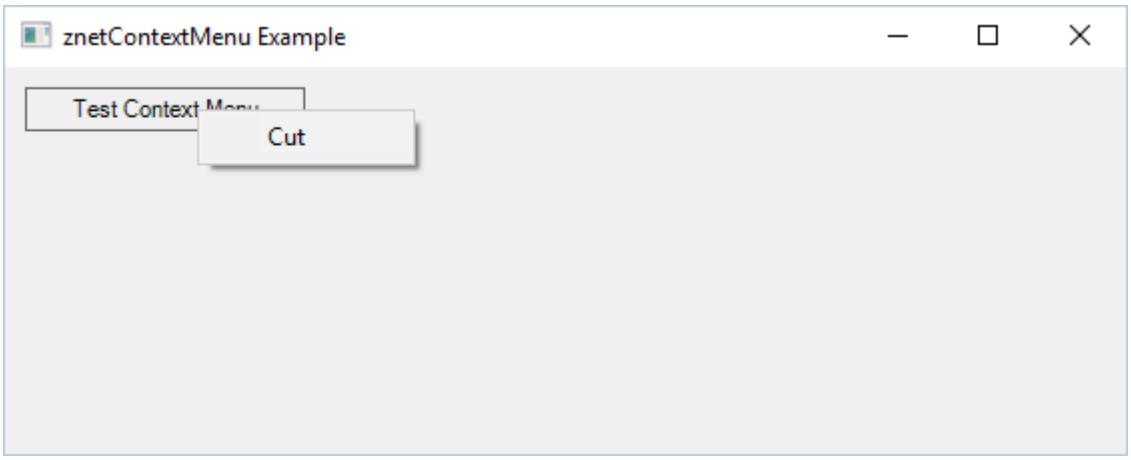
3. Add the menu items to the context menu using the **zItems** property (lines 7, 8, 9 10, 11 or 12)

4. Set the **zContextMenu** property of your .Net object to the **zPointer** property value of the **znetContextMenu** object (see line 14)

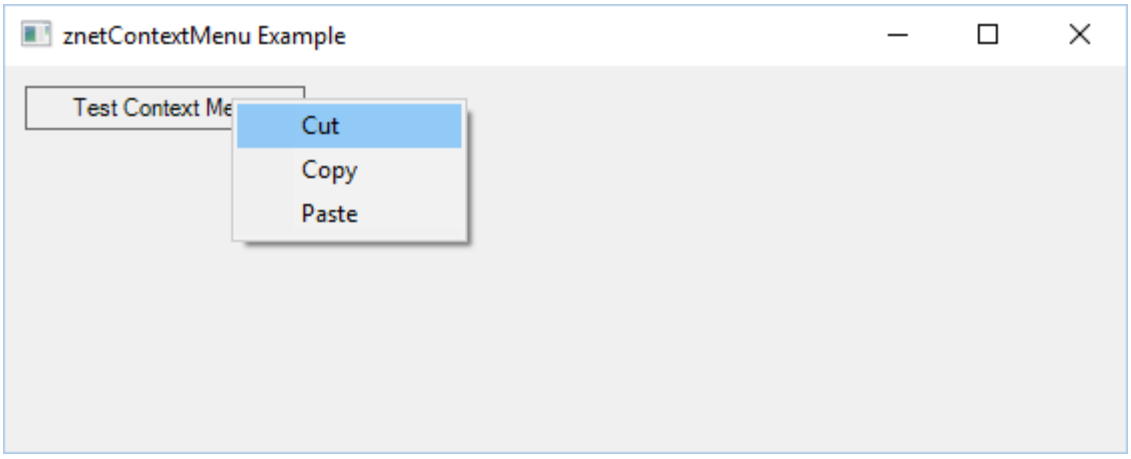
Copyright © 2016 Eric Lescasse

Calling the zznetContextMenu function with a 1 to 6 argument and then right clicking the button display the following context menus:

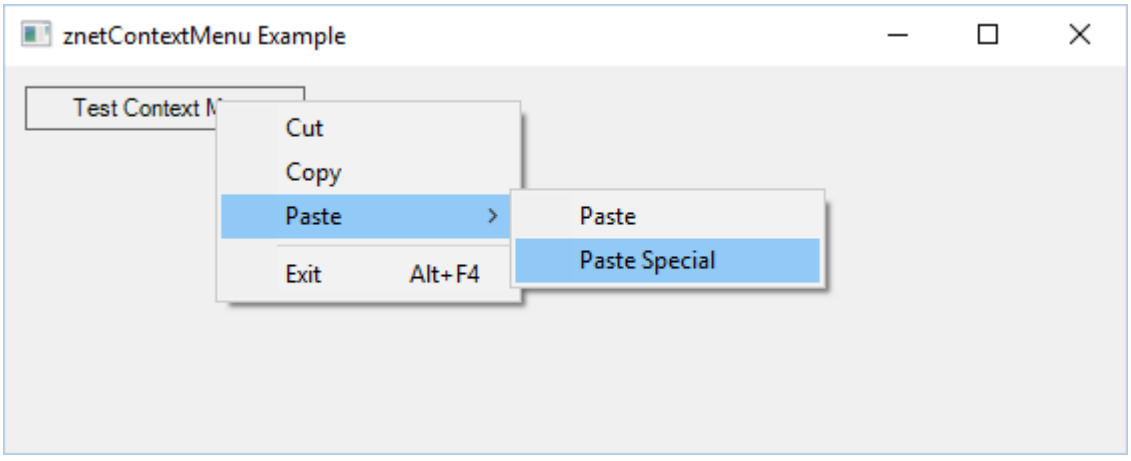
```
zznetcontextmenu 1
```




```
zznetcontextmenu 2
```

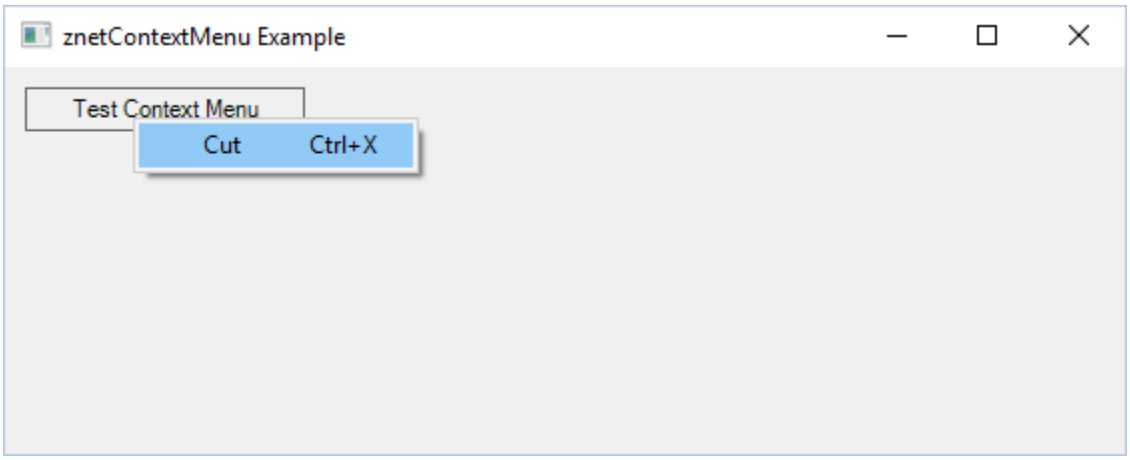



```
zznetcontextmenu 3
```

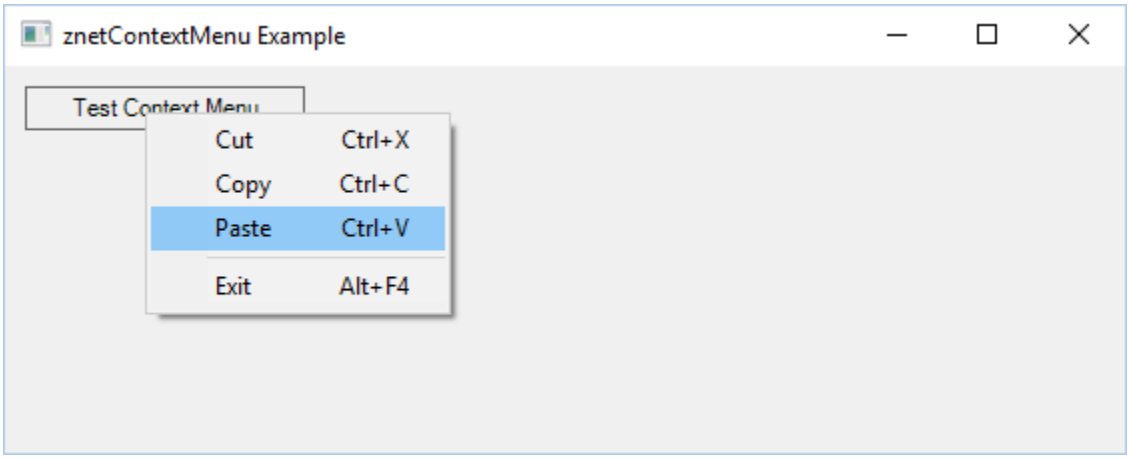


Copyright © 2016 Eric Lescasse

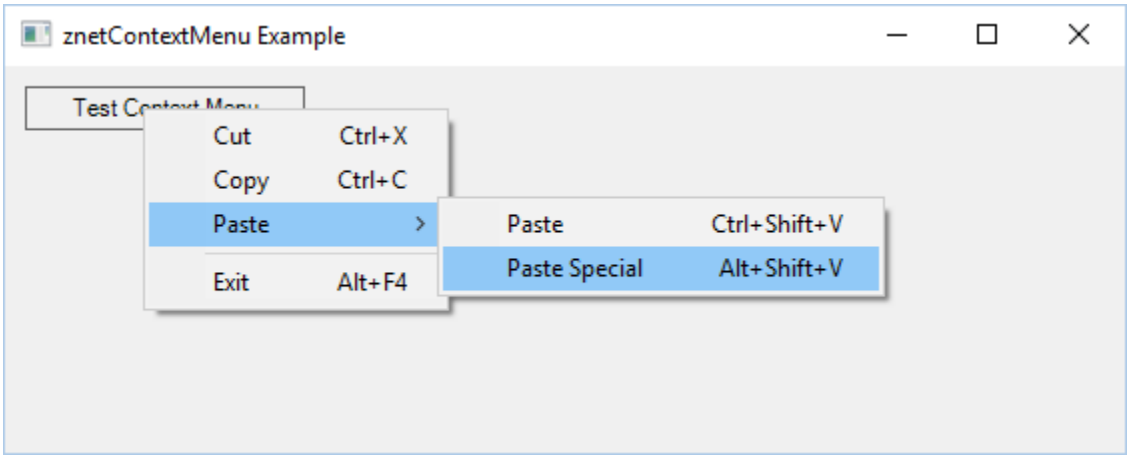
```
zznetcontextmenu 4
```




```
zznetcontextmenu 5
```




```
zznetcontextmenu 6
```



The menu items are defined using the **zItems** property.

Copyright © 2016 Eric Lescasse

- For simple menu items, just pass a nested vector of strings:
  Example:

```
←'cm'⎕wi'*zItems' 'Cut' 'Copy' 'Paste'
```

- For a context menu having sub-menus, pass a depth 3 nested vector of strings
  Example:

```
←'cm'⎕wi'*zItems' 'Cut' 'Copy'('Paste'('Paste' 'Paste Special'))
```

  In this menu the 3rd option (Paste) will have a sub-menu with 2 options: Paste and Paste Special

- To specify accelerator keys, define each menu option as a segmented with:
  - segment 1 being the menu option text
  - segment 2 being the accelerator keys (use Ctrl, Shift, Alt and +)
  Example:

```
←'cm'⎕wi'*zItems' '/Cut/Ctrl+X' '/Copy/Ctrl+C' '/Exit/Alt+F4'
```

  The separator may be any character which is + and is not part of the menu option text

- Finally, use '-' to specify separation lines
  Example:

```
←'cm'⎕wi'*zItems' 'Cut' 'Copy' '-' '/Exit/Alt+F4'
```

When you click on menu options in the above sample forms, you see the following be pringed in the APL Session, proving that the onXItemClick event fires ok:

```
.→1---.
|.→3-.|
||Cut||
|'---'|
'∈----'
.→2------------.
|.→5---..→5---.|
||Paste||Paste||
|'-----''-----'|
'∈-------------'
.→2--------------------.
|.→5---..→13----------.|
||Paste||Paste Special||
|'-----''-------------'|
'∈----------------------'
.→1----.
|.→4--.|
||Exit||
|'----'|
```

```
'∈-----'
```

Your event handler function may have code similar to the following for handling the **onX-ItemClick** events:

```
:case'cm_onXItemClick'
    :select∈'⊢',¨⍞warg
    :case'⊢Cut' ⋄ ⍞←'Cut was clicked!'
    :case'⊢Copy' ⋄ ⍞←'Copy was clicked!'
    :case'⊢Paste' ⋄ ⍞←'Paste was clicked!'
    :case'⊢Paste⊢Paste' ⋄ ⍞←'Paste was clicked in the Paste submenu!'
    :case'⊢Paste⊢Paste Special' ⋄ ⍞←'Paste Special was clicked in the Paste submenu!'
    :case'⊢Exit' ⋄  ⍞←'Exit was clicked!'
    :endselect
```

Note that I use the ⊢ separator because it is unlikely that a menu option text contains this character.

Handling context menus in an APL+Win application has never been easier.

Copyright © 2016 Eric Lescasse

The znetDateTime object

Description

The **znetDateTime** object is not a User Control that you can embed in your APL+Win forms, but is rather an object allowing you to process dates and times in your application.

Due to APL+Win maximum precision, znetDateTime dates are saved or returned as:

yyyymmdd.hhmmssx

where x is 10th of a second.

If a date is returned as: **20160130.2012** this just means that the number of seconds **ss** is 0 and that the number of 10th of a second **x** is **0**. In other words, trailing 0's are not printed to the APL Session.

A lot of the .Net DateTime methods return **DateTime** objects: **znetDateTime** converts these objects to **yyyymmdd.hhmmssx** APL floating point numbers.

Sample function using a znetDateTime object

The following function shows a number of **znetDateTime** properties and methods in action:

```
    ∇ zznetdatetime;⎕pp;t
[1]   ⍝∇ This function shows how to use the znetDateTime object
[2]
[3]   ⎕pp←17
[4]   ⎕wself←'dd'⎕wi'*Create' 'znetDateTime'('*zInit'⎕ts)  ⍝ create a datetime
      made of today's date & time
[5]   ⎕←⎕wi¨'*zYear' '*zMonth' '*zDay' '*zHour' '*zMinute' '*zSecond'
[6]   ⎕←'Date:',⎕wi'*zDate'
[7]   ⎕←'Kind:',⎕wi'*zKind'                  ⍝ 0=Unspecified, 1=Utc, 2=Local
[8]   ⎕←'MilliSeconds:',⎕wi'*zMilliSecond'
[9]   ⎕←'Ticks:',⎕wi'*zTicks'                ⍝ nb. of 100-nanoseconds since 1/1/1
[10]  ⎕←'DayOfWeek:',⎕wi'*zDayOfWeek'        ⍝ 0=Sun, 1=Mon, ... 6=Sat
[11]  ⎕←'DayOfYear:',⎕wi'*zDayOfYear'
[12]  ⎕←'TimeOfDay:',⎕wi'*zTimeOfDay'
[13]  ⎕←'UtcNow:',⎕wi'*zUtcNow'
[14]  ⎕←'AddMonths 30:',⎕wi'*zAddMonths'30
[15]  ⎕←'AddDays 30:',⎕wi'*zAddDays'30
[16]  ⎕←'AddHours 148:',⎕wi'*zAddHours'148
[17]  ⎕←'AddMinutes -112:',⎕wi'*zAddMinutes'¯112
[18]  ⎕←'AddSeconds 23234:',⎕wi'*zAddSeconds'23234
[19]  ⎕←'AddTicks 512000:',⎕wi'*zAddTicks'512000
[20]  ⎕←'FromFileTime 131275375752387677:',⎕wi'*zFromFileTime'131275375752387677
[21]  ⎕←'IsLeapYear 1600:',⎕wi'*zIsLeapYear'1600
[22]  ⎕←'DaysInMonth 2016 2:',⎕wi'*zDaysInMonth'2016 2
[23]  ⎕←'IsDaylightSavingTime:',⎕wi'*zIsDaylightSavingTime'
[24]  ⎕←'Subtract 20151225.0901512:',⎕wi'*zSubtract'20151225.0901512
[25]  ⎕←'ToLongDateString:',⎕wi'*zToLongDateString'
```

Copyright © 2016 Eric Lescasse

```
[26]  ⎕←'ToShortDateString:',⎕wi'*zToShortDateString'
[27]  ⎕←'ToLongTimeString:',⎕wi'*zToLongTimeString'
[28]  ⎕←'ToShortTimeString:',⎕wi'*zToShortTimeString'
[29]  ⍝ Parsing date strings with zParse
[30]  ⎕←'Parse 1/1/2000:',⎕wi'*zParse' '1/1/2000'
[31]  ⎕←'Parse Fri, 27 Feb 2009 03:11:21 GMT:',⎕wi'*zParse' 'Fri, 27 Feb 2009
            03:11:21 GMT'
[32]  ⎕←'Parse 2009/02/26 18:37:58:',⎕wi'*zParse' '2009/02/26 18:37:58'
[33]  ⎕←'Parse Thursday, February 26, 2009:',⎕wi'*zParse' 'Thursday, February 26,
            2009'
[34]  ⎕←'Parse February 26, 2009:',⎕wi'*zParse' 'February 26, 2009'
[35]  ⎕←'Parse 2002-02-10:',⎕wi'*zParse' '2002-02-10'
[36]  ⍝ Note: Parse is culture aware: since I am running
[37]  ⍝ these examples from France, Parse expects a
[38]  ⍝ dd/mm/yyyy format, not a mm/dd/yyyy format
[39]  ⎕←'Parse 21/2/2009 10:35 PM:',⎕wi'*zParse' '21/2/2009 10:35 PM'
[40]  ⎕←'Parse 8:04:00 PM:',⎕wi'*zParse' '8:04:00 PM'
[41]  ⍝ Parsing date strings with zParseExact
[42]  ⎕←'ParseExact 20160130 yyyyMMdd:',⎕wi'*zParseExact' '20160130' 'yyyyMMdd'
[43]  ⎕←'ParseExact 1 jan 16 8:54 PM d MMM yy h:mm tt:',⎕wi'*zParseExact' '1 jan 16
            8:54 PM' 'd MMM yy h:mm tt'
[44]  ⎕←'ParseExact 18/08/2015 06:30:15.006542 dd/MM/yyyy hh:mm:ss.ffffff:',
            ⎕wi'*zParseExact' '18/08/2015 06:30:15.006542' 'dd/MM/yyyy hh:mm:ss.
           ffffff'
    ∇
```

If we run this function, we get the following output:

```
2016 1 31 0 21 56
Date: 20160131
Kind: 0
MilliSeconds: 539
Ticks:  6.3589796516538992E17
DayOfWeek: 0
DayOfYear: 31
TimeOfDay: 2156.5390000000000
UtcNow: 20160130.2321566
AddMonths 30: 20180731.0021565
AddDays 30: 20160301.0021565
AddHours 148: 20160206.0421565
AddMinutes -112: 20160130.2229565
AddSeconds 23234: 20160131.0649105
AddTicks 512000: 20160131.0021565
FromFileTime 131275375752387677: 20161230.0312552
IsLeapYear 1600: 1
DaysInMonth 2016 2: 29
IsDaylightSavingTime: 0
Subtract 20151225.0901512: 36152005.339
ToLongDateString:31 January, 2016
ToShortDateString:31-Jan-16
ToLongTimeString:12:21:56 AM
ToShortTimeString:12:21 AM
Parse 1/1/2000: 20000101
Parse Fri, 27 Feb 2009 03:11:21 GMT: 20090227.041121
Parse 2009/02/26 18:37:58: 20090226.183758
Parse Thursday, February 26, 2009: 20090226
Parse February 26, 2009: 20090226
Parse 2002-02-10: 20020210
```

Copyright © 2016 Eric Lescasse

```
Parse 21/2/2009 10:35 PM: 20090221.2235
Parse 8:04:00 PM: 20160131.2004
ParseExact 20160130 yyyyMMdd: 20160130
ParseExact 1 jan 16 8:54 PM d MMM yy h:mm tt: 20160101.2054
ParseExact 18/08/2015 06:30:15.006542 dd/MM/yyyy hh:mm:ss.ffffff: 20150818.063015
```

Efficiency

A znetDateTime object represents a single date. What if we have to process a 100 dates or maybe 100000 dates.

The .Net Framework is extremely efficient.

Here is a little function:

```
    ∇ v←a zztest n;d;e;i;t
[1]   ←'dd'⎕wi'*Create' 'znetDateTime'('*zInit'(d←3↑⎕ts))
[2]   e←100⊥d
[3]   t←⎕ts
[4]   :select a
[5]   :case 1
[6]       v←nρ0
[7]       :for i :inιn
[8]           v[i]←'dd'⎕wi'*zAddDays'i
[9]       :endfor
[10]  :case 2
[11]      v←'dd'⎕wi'*zAddDaysv'(ιn)
[12]  :case 3
[13]      v←100⊥⍉DATEREP(ιn)+DATEBASE d
[14]  :endselect
[15]  ⎕←-/(⊂24 60 60 1000)⊥¨3↓¨⎕ts t
    ∇
```

This function adds ιn days to the current date, using 3 different algorithms.

- The 1st one is looping n times calling the **zAddDays** method once per loop
- The 2nd one does the same loop calling the **DateTime** method once per loop, but it does it in C# in the **LC.zObjects.DLL**
- The 3rd one uses the very efficient **DATEREP** and **DATEBASE** APL functions delivered in the **TOOLS\DATES** workspace.

Let's add ι100000 days to the current date and compare the milliseconds times consumled by each algorithm:

```
      aaa←1 zztest 100000
893
      bbb←2 zztest 100000
23
      ccc←3 zztest 100000
120
      aaa≡bbb
1
```

```
      aaa≡ccc
1
```

As we can see, using the **zAddDaysv** C# method is more than **5** times faster than the very fast **DATEREP/DATEBASE** APL algorithm!

This is all the more impressive when you consider that, in the case of algorithm #2, on top of doing the 100000 date additions, we are sending 100000 integers from APL to C# (⍳100000) and C# sends us back 100000 computed dates!!!

The 893 milliseconds consumed by the first algorithm are mostly consumed by running an APL loop 100000 times!

Note that in this version of zObjects, the **zAddDaysv** method is the only one in the **znet-DateTime** object that accepts a vector argument and returns a vector of dates. All other methods are scalar methods (i.e. methods which require loops to work on APL vectors).

Copyright © 2016 Eric Lescasse

The znetDictionary object

Description

The .Net Framework includes a large nmber of objects which are Collections.

One of them is the Dictionary object.

The znetDictionary object implements a Dictionary<string, object> generic object which you can use in your APL applications.

Basically, from an APL point of view, you can see such a dictionary just as a vector of APL objects (of any rank, size, depth, structure) where each element has a name.

You can then retrieve elements from the dictionary by their names, retrieve all the names contained in the dictionary, retrieve all the values, etc.

Using a **znetDictionary** is extremely easy.

How to use a znetDictionary

First, create an instance of a znetDictionary:

```
      ⎕wself←'zdic'⎕wi'*Create' 'znetDictionary'
```

Use the *zAdd method to add key/value pairs to the dictionary:

```
      ⎕wi'*zAdd' 'months'(201500+⍳12)
      ⎕wi'*zAdd' 'sales'(?10 12⍴1000)
```

Retrieve the names of all elements saved in the dictionary:

```
      ⎕wi'*zKeys'
 months sales
```

Retrieve all the values contained in the dictionary:

```
      ⎕wi'*zValues'
201501 201502 201503 201504 201505 201506 201507 201508 201509 201510 201511
201512
```

Copyright © 2016 Eric Lescasse

```
220 683 407 622  70 894 543 605 745 446 177 816
 15 863 575 889 637 631 443 802 115 355 144 858
351  21 247 596 787 500 299  96 461 700 838 770
863 425 582 239 322 811 699 466  44 795 164 376
152 110 660  80 779 902 984 804  56 978 776 398
777 819 929 418 181 838 872 411 556 131 862 742
299 879 221 734 957 387 631 582 460 679 632 186
949 118 801 352 434 502 951   8 361 985 217 655
811 135 466 465 157 351   4 827 145 839 747 629
101 715 641  53 978 855 689 329 865 751 733 306
```

Find how many elements the dictionary contains using the **zCount** method:

```
      ⎕wi'*zCount'
2
```

Use the ***zTryGetValue** method to retrieve an element of the dictionary by its key:

```
      ⎕wi'*zTryGetValue' 'sales'
 1    691  59 407 797 589 285 440 715  13 139 975 878
       75 638 307 467 469 174 586 226 343  30 927 712
      289 341 934 708 650 756 981 985 926 652 797 722
      530 974 375 916 188 135  77 502 332 298 673 686
      948 332 212 122 495 750  63 163 914 637  49 744
      435 239  63 186 702   4 680 287 708 144 970  89
      203  45 411 720   8 707 963 171 357 909 278 589
       91 981 235 108 152 130 162 840 557 600 895 890
       30 302 723 173 615 722 323 842 205 492 101 172
      921 535 580 667 270 195 486 463  19 613 346 622

      ⍴¨⎕wi'*zTryGetValue' 'sales'
   10 12

      ⎕wi'*zTryGetValue' 'months'
201501 201502 201503 201504 201505 201506 201507 201508 201509 201510 201511
201512
```

If the operation succeeds the first element of the result is 1, otherwise it is 0:

```
      ⎕wi'*zTryGetValue' 'NonExistenKey'
 0
      ]display ⎕wi'*zTryGetValue' 'NonExistenKey'
.→2-------------------------.
|   .⊖0--------------------.|
| 0 |.-------------------.||
|   ||.→5----------------.|||
|   |||.⊖0--.            ||||
|   ||||.→2.| V  T  0  0 ||||
|   |||||VT|| -  -       ||||
|   ||||'--'|             ||||
|   |||'∈---'             ||||
|   ||'∈-----------------'|||
|   |'∈-------------------'||
|   '∈---------------------'|
'∈--------------------------'
```

Note that, in that case, the result second element is identical to '#'⎕wi'empty':

```
      ('#'⎕wi'empty')≡2⊃⎕wi'*zTryGetValue' 'NonExistentKey'
1
```

You can also more simply use the ***zGet** method to retrieve the value stored for any key:

```
      ⎕wi'*zGet' 'months'
201501 201502 201503 201504 201505 201506 201507 201508 201509 201510 201511
201512
```

Here again, if the specified key does not exist in the znetDictionary; the result is '#'⎕wi'empty':

```
      ('#'⎕wi'empty')≡⎕wi'*zGet' 'NonExistentKey'
1
```

You can check if the dictionary contains a given key:

```
      ⎕wi'*zContainsKey' 'sales'
1
      ⎕wi'*zContainsKey' 'NonExistenKey'
0
```

And finally you can remove an item from the dictionary by its key:

```
      ⎕wi'*zCount'
2
      ⎕wi'*zKeys'
 months sales
      ⎕wi'*zRemove' 'months'
1
      ⎕wi'*zKeys'
 sales
      ⎕wi'*zCount'
1
```

At any time, you can clear the dictionary, using its **zClear** method:

```
      'zdic'⎕wi'*zClear'
      'zdic'⎕wi'*zCount'
0
```

Efficiency

Using a **znetDictionary** is very efficient.

Try to add 1000 key/value pairs in the dictionary as follows, for example:

```
      aaa←(⊂'key'),¨⍕¨⍳1000
      bbb←⍳¨⍳1000
```

```
      t←⎕ts ⋄ ⎕wi¨(⊂⊂'*zadd'),¨(⊂¨aaa),¨⊂¨bbb ⋄ ⎕←-/(⊂24 60 60 1000)⊥¨3↓¨⎕ts t
34

      ⎕size'bbb'
2030028
      ⎕wi'*zCount'
1000
      10↑⎕wi'*zKeys'
 key1 key2 key3 key4 key5 key6 key7 key8 key9 key10
      6↑⎕wi'*zValues'
 1  1 2  1 2 3  1 2 3 4  1 2 3 4 5  1 2 3 4 5 6
```

It has taken **34** milliseconds to create a dictionary with **1000** key/value pairs, the total size of the values being more than **2 million bytes**!

Conclusion

znetDictionary objects could be used in many places in APL applications to refer to data by their names, which makes applications clearer to work with and easier to maintain.

Basically, instead of writing things like:

```
      sales←myData[2]
```

you would write:

```
      sales←⎕wi'*zGet' 'sales'
```

It's a little more verbose, but you won't have to remember that the Sales data is in item **2** of your myData variable: your data is now in the **sales** slot of your dictionary. An index holds no meaning and using indices everywhere in an application makes it much harder to maintain.

Finally, remember that **znetDictionary** is a memory object: hence, the keys and values you store in the **znetDictionary** are not persisted from one application session to the other. So, you often load the **znetDictionary** from a permanent source as the application starts, use it while the application runs and save it back to the permanent source as the application ends (or whenever you want while the application runs).

Copyright © 2016 Eric Lescasse

The znetEnvironment object

Introduction

The **znetEnvironment** object allows you to programmatically:

- retrieve all kinds of information about your system,
- retrieve information about the current user,
- retrieve information about the current process,
- to read or change environment variables

and more.

How to use znetEnvironment

Loading the znetEnvironment object

```
]zload znetEnvironment
```

Creating an instance of znetEnvironment

```
←'env'⍁wi'*Create' 'znetEnvironment'
```

The znetEnvironment API

The **znetEnvironment** object includes the following properties and methods:

```
      'env'⍁wi'mainprops'
Main Properties
---------------
*zCommandLine              *zMachineName       *zTickCount
*zCurrentDirectory         *zNewLine           *zUserDomainName
*zCurrentManagedThreadId   *zOSVersion         *zUserInteractive
*zExitCode                 *zProcessorCount    *zUserName
*zHasShutdownStarted       *zStackTrace        *zVersion
*zIs64BitOperatingSystem   *zSystemDirectory   *zWorkingSet
*zIs64BitProcess           *zSystemPageSize

Main Methods
------------
*zDoc                          *zGetEnvironmentVariable_2
*zExit                         *zGetFolderPath
*zExpandEnvironmentVariables   *zGetFolderPath_2
*zFailFast                     *zGetLogicalDrives
*zGetCommandLineArgs           *zSetEnvironmentVariable
*zGetEnvironmentVariable       *zSetEnvironmentVariable_2

Main Events
-----------
```

Copyright © 2016 Eric Lescasse

Testing the znetEnvironment API

Getting information about the current process

Retrieving the command line that started this APL session:

```
      'env'⎕wi'*zCommandLine'
"C:\APLWIN15.1\aplw.exe" c:\aplwin\zobjects\zobjects.w3
```

Getting the current directory:

```
      'env'⎕wi'*zCurrentDirectory'
c:\aplwin15.1
```

Getting the amount of physical memory used by the current process:

```
      'env'⎕wi'*zWorkingSet'
61829120
      'CI10'⎕fmt'env'⎕wi'*zWorkingSet'
61,849,600
```

Getting information about the current computer:

```
      'env'⎕wi'*zIs64BitOperatingSystem'
1

      'env'⎕wi'*zMachineName'
SPRO3

      'env'⎕wi'*zProcessorCount'
4

      'env'⎕wi'*zSystemDirectory'
C:\WINDOWS\system32

      'env'⎕wi'*zSystemPageSize'
4096

      'env'⎕wi'*zOSVersion'
 2 4.0.30319.42000  Microsoft Windows NT 10.0.10586.0

      ]display 'env'⎕wi'*zOSVersion'
.→4-------------------------------------------------------.
|   .→15-----------..⍟..→33-------------------------------.|
| 2 |4.0.30319.42000|| ||Microsoft Windows NT 10.0.10586.0||
|   '---------------''-''---------------------------------'|
'⍷-------------------------------------------------------'

      'env'⎕wi'?zOSVersion'
Syntax: OSVersion←'obj'⎕wi'*zOSVersion'
OSVersion: a nested vector containing information about the OS version
      [1]= platform id (int)
           0 = PlatformID.Win32S
           1 = PlatformID.Win32Windows
           2 = PlatformID.Win32NT
           3 = PlatformID.WinCE
           4 = PlatformID.Unix
```

Copyright © 2016 Eric Lescasse

```
           5 = PlatformID.Xbox
           6 = PlatformID.MacOSX
      [2]= version number (string)
      [3]= service pack (string)
      [4]= version string (string) [this is the one containing the Windows version
number, i.e. 10]


      'env'⎕wi'*zVersion'
4.0.30319.42000

      'env'⎕wi'*zGetLogicalDrives'
 C:\ D:\ E:\
      ]display 'env'⎕wi'*zGetLogicalDrives'
.→3-------------.
|.→3-..→3-..→3-.|
||C:\||D:\||E:\||
|'---''---''---'|
'ϵ--------------'
```

Getting information about the current user

```
      'env'⎕wi'*zUserName'
Eric Lescasse

      'env'⎕wi'*zUserDomainName'
SPRO3

      'env'⎕wi'*zUserInteractive'
1
```

Measuring execution times

You can use the **zTickCount** property to get the time in milliseconds since the last time the computer was started:

```
      'env'⎕wi'*zTickCount'
598471031
      'env'⎕wi'*zTickCount'
598471953
```

This gives you a way to measure the time an APL expression takes to execute, in milliseconds.

Example:

```
      aaa←'env'⎕wi'*zTickCount' ⋄ bbb←?1000000⍴1000000 ⋄ aaa-⍨'env'⎕wi'*zTickCount'
63
```

If you repeat the same set of instructions you will not always get the same result, but it will give you a pretty good idea of the time it takes for the expression to get executed:

```
      aaa←'env'⎕wi'*zTickCount' ⋄ bbb←?1000000⍴1000000 ⋄ aaa-⍨'env'⎕wi'*zTickCount'
63
```

Copyright © 2016 Eric Lescasse

```
      aaa←'env'⎕wi'*zTickCount' ⋄ bbb←?1000000⍴1000000 ⋄ aaa-⍨'env'⎕wi'*zTickCount'
62
      aaa←'env'⎕wi'*zTickCount' ⋄ bbb←?1000000⍴1000000 ⋄ aaa-⍨'env'⎕wi'*zTickCount'
63
      aaa←'env'⎕wi'*zTickCount' ⋄ bbb←?1000000⍴1000000 ⋄ aaa-⍨'env'⎕wi'*zTickCount'
47
      aaa←'env'⎕wi'*zTickCount' ⋄ bbb←?1000000⍴1000000 ⋄ aaa-⍨'env'⎕wi'*zTickCount'
47
      aaa←'env'⎕wi'*zTickCount' ⋄ bbb←?1000000⍴1000000 ⋄ aaa-⍨'env'⎕wi'*zTickCount'
63
      aaa←'env'⎕wi'*zTickCount' ⋄ bbb←?1000000⍴1000000 ⋄ aaa-⍨'env'⎕wi'*zTickCount'
47
```

Using Environment Variables

Getting a list of all environment variables

The **zGetEnvironmentVariables** returns a nested matrix with all environment variables on this machine:

```
      'env'⎕wi'*zGetEnvironmentVariables'
 PROCESSOR_ARCHITEW6432          AMD64
 COMPUTERNAME                    SPRO3
 CommonProgramFiles(x86)         C:\Program Files (x86)\Common Files
 TMP                             C:\Users\Eric\AppData\Local\Temp
 HOMEPATH                        \Users\Eric
 MOZ_PLUGIN_PATH                 C:\Program Files (x86)\Nuance\Power PDF\Bin
 PROCESSOR_REVISION              4501
 PATHEXT                         .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
 USERDOMAIN_ROAMINGPROFILE       SPRO3
 TEMP                            C:\Users\Eric\AppData\Local\Temp
 LOCALAPPDATA                    C:\Users\Eric\AppData\Local
 PUBLIC                          C:\Users\Public
 USERDOMAIN                      SPRO3
 ProgramFiles(x86)               C:\Program Files (x86)
. . .
```

Reading one particular environment variable

Use the **zGetEnvironment** variable method:

```
      'env'⎕wi'*zGetEnvironmentVariable' 'PROCESSOR_REVISION'
4501
```

Note: this query is not case sensitive:

```
      'env'⎕wi'*zGetEnvironmentVariable' 'processor_revision'
4501
```

Changing an environment variable

Use the **zSetEnvironmentVariable**:

```
      'env'⎕wi'*zGetEnvironmentVariable' 'tmp'
C:\Users\Eric\AppData\Local\Temp

      'env'⎕wi'*zSetEnvironmentVariable' 'TMP' 'c:\temp'

      'env'⎕wi'*zGetEnvironmentVariable' 'tmp'
c:\temp

      'env'⎕wi'*zSetEnvironmentVariable' 'tmp' 'C:\Users\Eric\AppData\Local\Temp'

      'env'⎕wi'*zGetEnvironmentVariable' 'tmp'
C:\Users\Eric\AppData\Local\Temp
```

Retrieving the path of important folders

Use the **zGetFolderPath** to get the path of most important folders on your computer.

The documentation is the following:

```
      'env'⎕wi'?GetFolderPath'
Syntax: result←'obj'⎕wi'*zGetFolderPath'folder
Summary: Gets the path to the system special folder that is
      identified by the specified enumeration
folder: An enumerated constant that identifies a system special
      folder
           0 = SpecialFolder.Desktop
           2 = SpecialFolder.Programs
           5 = SpecialFolder.MyDocuments
           5 = SpecialFolder.Personal
           6 = SpecialFolder.Favorites
           7 = SpecialFolder.Startup
           8 = SpecialFolder.Recent
           9 = SpecialFolder.SendTo
          11 = SpecialFolder.StartMenu
          13 = SpecialFolder.MyMusic
          14 = SpecialFolder.MyVideos
          16 = SpecialFolder.DesktopDirectory
          17 = SpecialFolder.MyComputer
          19 = SpecialFolder.NetworkShortcuts
          20 = SpecialFolder.Fonts
          21 = SpecialFolder.Templates
          22 = SpecialFolder.CommonStartMenu
          23 = SpecialFolder.CommonPrograms
          24 = SpecialFolder.CommonStartup
          25 = SpecialFolder.CommonDesktopDirectory
          26 = SpecialFolder.ApplicationData
          27 = SpecialFolder.PrinterShortcuts
          28 = SpecialFolder.LocalApplicationData
          32 = SpecialFolder.InternetCache
          33 = SpecialFolder.Cookies
          34 = SpecialFolder.History
          35 = SpecialFolder.CommonApplicationData
          36 = SpecialFolder.Windows
          37 = SpecialFolder.System
          38 = SpecialFolder.ProgramFiles
          39 = SpecialFolder.MyPictures
```

```
            40 = SpecialFolder.UserProfile
            41 = SpecialFolder.SystemX86
            42 = SpecialFolder.ProgramFilesX86
            43 = SpecialFolder.CommonProgramFiles
            44 = SpecialFolder.CommonProgramFilesX86
            45 = SpecialFolder.CommonTemplates
            46 = SpecialFolder.CommonDocuments
            47 = SpecialFolder.CommonAdminTools
            48 = SpecialFolder.AdminTools
            53 = SpecialFolder.CommonMusic
            54 = SpecialFolder.CommonPictures
            55 = SpecialFolder.CommonVideos
            56 = SpecialFolder.Resources
            57 = SpecialFolder.LocalizedResources
            58 = SpecialFolder.CommonOemLinks
            59 = SpecialFolder.CDBurning

result: (String) The path to the specified system special folder,
      if that folder physically exists on your computer; otherwise,
      an empty string (""). A folder will not physically exist if
      the operating system did not create it, the existing folder
      was deleted, or the folder is a virtual directory, such as My
      Computer, which does not correspond to a physical path
```

Here are a few examples:

```
      'env'□wi'*zGetFolderPath' 0              Desktop
C:\Users\Eric\Desktop
      'env'□wi'*zGetFolderPath' 5              My Documents
C:\Users\Eric\Documents
      'env'□wi'*zGetFolderPath' 36             Windows
C:\WINDOWS
      'env'□wi'*zGetFolderPath' 11             Start Menu
C:\Users\Eric\AppData\Roaming\Microsoft\Windows\Start Menu
      'env'□wi'*zGetFolderPath' 32             Internet Cache
C:\Users\Eric\AppData\Local\Microsoft\Windows\INetCache
      'env'□wi'*zGetFolderPath' 20             Fonts
C:\WINDOWS\Fonts
```

Copyright © 2016 Eric Lescasse

The znetFtpWebRequest object

Introduction

The **znetFtpWebRequest** allows you to use FTP to communicate with a remote server in your application.

With **znetFtpWebRequest** you can fully use FTP from an APL application. Basically, you can:

- List files in an FTP site remote folder
- Get information about files in an FTP site remote folder
- Upload files to an FTP site
- Download files from an FTP site
- Create and delete folders in an FTP site
- Rename remote files in an FTP Site
- Delete files in an FTP site remote folder
- Rename files in an FTP site remote folder

How to use znetFtpWebRequest

Loading the necessary functions

First, let's load the necessary function from zObjects.sf:

```
]zload znetFtpWebRequest zznetwebrequest
```

Creating an instance of znetWebRequest

The first step is, of course, to create an instance of znetWebRequest:

```
←'req'⎕wi'*Create' 'znetFtpWebRequest'
```

The znetFtpWebRequest API

```
      'req'⎕wi'mainprops'
Main Properties
---------------
*zAuthenticationLevel   *zEnableSsl           *zRenameTo
*zConnectionGroupName   *zImpersonationLevel  *zRequestUri
*zContentLength         *zKeepAlive           *zTimeout
*zContentOffset         *zMethod              *zUseBinary
*zContentType           *zPreAuthenticate     *zUseDefaultCredentials
*zCredentials           *zReadWriteTimeout    *zUsePassive

Main Methods
------------
*zAbort            *zDoc              *zListDirectory         *zUploadFile
*zCreate           *zDownloadFile     *zListDirectoryDetails
```

Copyright © 2016 Eric Lescasse

```
*zCreateDirectory   *zGetFileCreatedTime   *zRemoveDirectory
*zDeleteFile        *zGetFileSize         *zRename

Main Events
-----------
```

Connecting to an FTP Site

The second step must be to connect to an FTP Site.

In all the following examples we will use the O'Reilly ftp.albahari.com FTP public site[9].

You must use the zCreate method and provide it with:

1. The full FTP site name
2. The FTP User Name
3. The FTP User Password

Example:

```
      ←'req'□wi'*zCreate' 'ftp://ftp.albahari.com' 'nutshell' 'oreilly'
```

Note that the **zCreate** method does not yield any error in case you provide wrong credentials.

Once you have successfully connected to the FTP Site, you can use any of the znetFtpRequest other methods to communicate with the FTP Site.

Listing remote folders

Use the **zListDirectory**[10] method to list the content of a folder at the FTP Site:

```
     'req'□wi'*zListDirectory' ''
 code.htm guestbook.txt New Folder tempfile.txt

      ]display 'req'□wi'*zListDirectory' ''
.→4-------------------------------------------------.
|.→8------..→13----------..→10-------..→12---------.|
||code.htm||guestbook.txt||New Folder||tempfile.txt||
|'--------''-------------''----------''------------'|
'ϵ-------------------------------------------------'
```

An empty argument to zListDirectory (as done above) means you want to list files in the FTP Site **root folder**.

[9] If you follow the examples below and reproduce them on your machine, you will be creating folders and files on the ftp.albahari.com FTP site; please be sure to remove them at the end of your tries to leve the site as celan as you found it when starting.

[10] Be warned that FTP operations may sometimes be very slow: be patient and wait for the methods to return their result. That mostly depends on the FTP Site. What you may observe is that it is often the first request that takes long: if you immediately send other FTP requests to the server you generally get answers pretty fast.

Copyright © 2016 Eric Lescasse

The **zListDirectory** method returns both file names and folder names.

Creating a remote folder

Use the **zCreateDirectory** method to create a remote folder in the FTP Site:

```
      ←'req'⎕wi'*zCreateDirectory' 'Test'
```

Let's check the **Test** folder has been created:

```
      ]display 'req'⎕wi'*zListDirectory' ''
.→5-------------------------------------------------------.
|.→8------..→13----------..→10-------..→12---------..→4--.|
||code.htm||guestbook.txt||New Folder||tempfile.txt||Test||
|'--------''-------------''----------''------------''----'|
'∈-------------------------------------------------------'
```

You could then create another directory inside the Test folder:

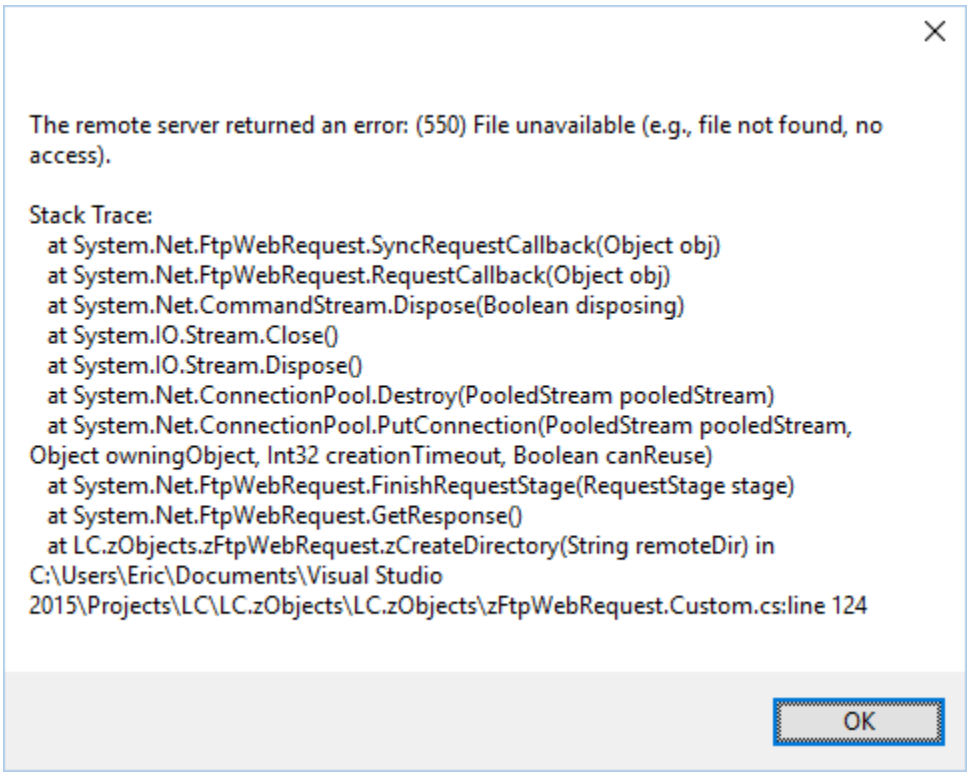
```
      ←'req'⎕wi'*zCreateDirectory' 'Test\Test2'

      ]display 'req'⎕wi'*zListDirectory' 'Test'
.→1-----.
|.→5---.|
||Test2||
|'-----'|
'∈------'
```

If you try to create adirectory that already exists:

```
      ←'req'⎕wi'*zCreateDirectory' 'Test\Test2'
```

you'll get the following error:

Copyright © 2016 Eric Lescasse



Uploading a file to a remote folder

Use the **zUploadFile** method to upload any file to an existing remote folder.

First, let's create a dummy file:

```
      'c:\temp\dummyfile.txt'□xncreate ¯1
      (20000⍴'blah blah blah ')□nappend ¯1
      □nuntie ¯1
```

Now, let's upload it to the **Test\Test2** folder:

```
      ←'req'□wi'*zUploadFile' 'Test/Test2/dummyfile.txt' 'c:\temp\dummyfile.txt'
```

And let's check that it is there:

```
      'req'□wi'*zListDirectory' 'Test/Test2'
 dummyfile.txt
```

Getting the size of a remote file

At any time, you can request the size of a remote file using the **zGetFileSize** method. That may be quite useful before attempting to download a file as it would give you an idea of the time the download maight take:

```
      'req'□wi'*zGetFileSize' 'Test/Test2/dummyfile.txt'
20000
```

Copyright © 2016 Eric Lescasse

Getting more details about a remote folder

Sometimes, it may be useful to get more details than just the file names about a remote folder: you can do that using the **zListDirectoryDetails** method:

```
      ]display ⊃'req'⍞wi'*zListDirectoryDetails' 'Test\Test2'
.→3 68------------------------------------------------------------.
↓drwxrwxrwx   1 user     group           0 Feb 15 06:25 .          |
|drwxrwxrwx   1 user     group           0 Feb 15 06:15 ..         |
|-rw-rw-rw-   1 user     group       20000 Feb 15 06:25 dummyfile.txt|
'------------------------------------------------------------------'
```

Note that each line of the result is a character string.

Renaming a remote file

Use the **zRename** method to rename a remote file:

```
      'req'⍞wi'*zRename' 'Test/Test2/dummyfile.txt' 'dummy.txt'
1

      'req'⍞wi'*zListDirectory' 'Test/Test2'
 dummy.txt
```

Downloading a remote file

To download a remote file, use the **zDownloadFile** method and pass it 2 arguments:

1. The full relative path name of the remote file to download
2. The full path name of the file to create locally

Example:

```
      FileExist'c:\temp\dummy.txt'
0
      'req'⍞wi'*zDownloadFile' 'Test\Test2\dummy.txt' 'c:\temp\dummy.txt'
1
      FileExist'c:\temp\dummy.txt'
1
      'c:\temp\dummy.txt'⍞xntie ¯1
      ⍞nsize ¯1
20000
      ⍞nuntie ¯1
```

Deleting a remote file

To delete a remote file, use the **zDeleteFile** method and pass it the full relative path file name:

```
      'req'⍞wi'*zDeleteFile' 'Test\Test2\dummy.txt'
1
      ]display ⊃'req'⍞wi'*zListDirectoryDetails' 'Test\Test2'
.→2 57-----------------------------------------------------.
↓drwxrwxrwx   1 user     group           0 Feb 15 06:35 . |
|drwxrwxrwx   1 user     group           0 Feb 15 06:15 ..|
```

Copyright © 2016 Eric Lescasse

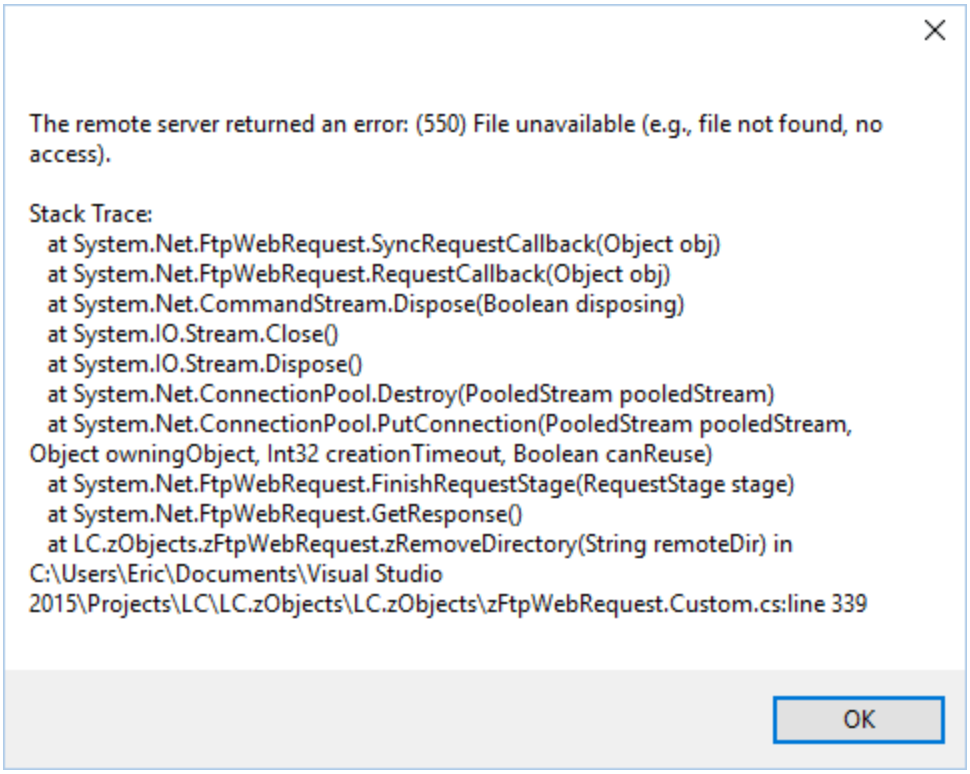
```
'------------------------------------------------------'
```

Deleting a remote folder

Use the **zRemoveDirectory** method to delete a remote folder:

```
      'req'⍞wi'*zRemoveDirectory' 'Test\Test2'
1
      'req'⍞wi'*zRemoveDirectory' 'Test'
1
      ]display ⊃'req'⍞wi'*zListDirectoryDetails' ''
.→4 68-------------------------------------------------------------.
↓-rw-rw-rw-   1 user     group       14822 Jan 26 05:19 code.htm     |
|-rw-rw-rw-   1 user     group           0 Jan 25 12:18 guestbook.txt|
|drwxrwxrwx   1 user     group           0 Nov 19  2014 New Folder   |
|-rw-rw-rw-   1 user     group           6 Jan 08 13:23 tempfile.txt |
'-------------------------------------------------------------------'
```

If the directory was not empty when you attempt to delete it, you'd get the following error:



Copyright © 2016 Eric Lescasse

The znetLinkLabel object

Introduction

The znetLinkLabel object allows you to include Internet links in your APL+WinUser Interfaces.

The link appears as a link would in a Web Browser, has a hand cursor when your mouse hovers over it.

How to use znetLinkLabel

Loading the znetLinkLabel object

First, load the **znetLinkLabel** object and the **zznetlinklabel** demo function:

```
]zload znetLinkLabel znetlinklabel
```

Creating an instance of znetLinkLabel

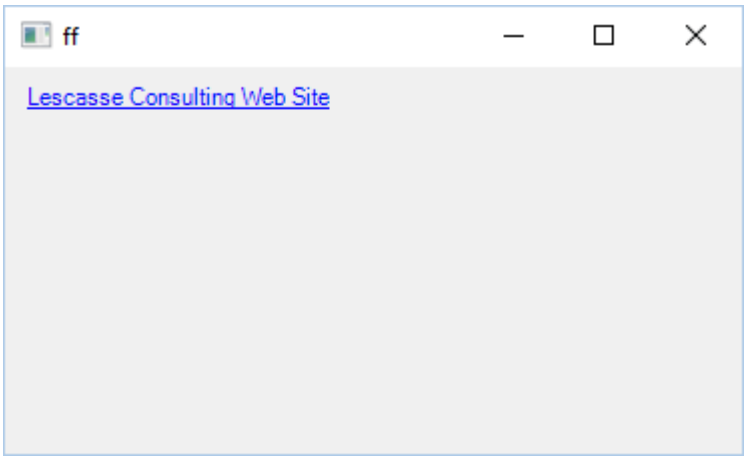
Since **znetLinkLabel** is a control, it must be created as a child of a container like a **zForm** or **zFrame**, **zPage**, etc.

```
←'ff'⎕wi'*Create' 'zForm'('DemoShow'.2 .2 1 0)
←'ff'⎕wi'*.ll1.Create' 'znetLinkLabel'('wherelc'0 0 16 300)
```

At this stage the znetLinkLabel is not visible because it has no text, so, at a minimum, you should also set the zText property:

```
←'ff'⎕wi'*.ll1.zText' 'Lescasse Consulting Web Site'
```

The form now reads:



Copyright © 2016 Eric Lescasse

Setting the link

The next step is to associate the **znetLinkLabel** object with a link.

The link can be:

- An Internet URL
- An application (.exe file)
- A document

You set the link using the zLink property as follows:

```
←'ff'□wi'*.ll1.zLink' 'http://www.lescasse.com/'
```

If you now click on the **znetLinkLabel** object, your default Internet browser will open with my home page displayed in it.

Handling clicks on znetLinkLabel objects

By default, the **znetLinkLabel** object knows what to do when the user clicks on it, provided you have set the **zLink** property. You don't have to handle a Click event on the **znetLinkLabel**.

For example, if the **zLink** property is set as follows:

```
←'ff'□wi'*.ll1.zLink' 'notepad.exe'
```

clicking on the ll1 znetLinkLabel control will open Notepad.

However, in some cases, you may want to override this default behavior and provide your own click event handler.

In those cases, you should do 2 things:

- set the **zDisableDefault** property to **1**
- subscribe to the **onXLinkClicked** event handler.

The zznetlinklabel sample function

The following function demonstrates various uses of the **znetLinkLabel** object:

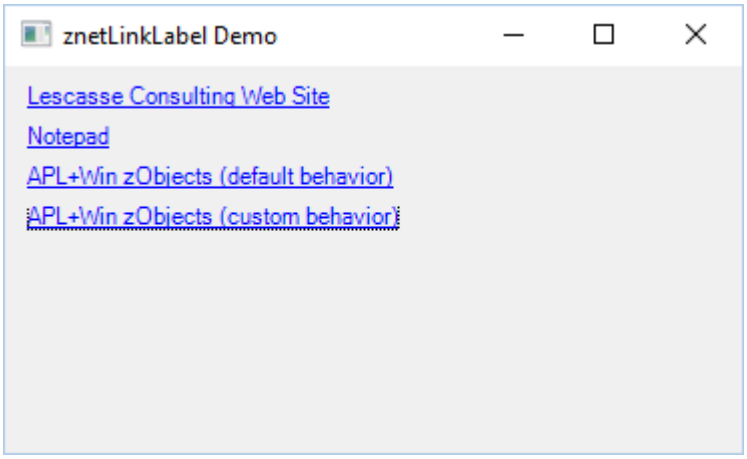
Copyright © 2016 Eric Lescasse

```
    ∇ zznetlinklabel a
[1]   ⍝∇ This function demonstrates how to use the znetLinkLabel object
[2]   ⍝∇ You only need 2 properties: zText and zLink
[3]   ⍝∇ The control knows what to do when the link label is clicked
[4]   ⍝∇ But you can override that default behavior by setting <zDisableDefault>
to 1
[5]   ⍝∇ and by defining your own onXLinkClicked event handler
[6]
[7]   :select a
[8]   :case''
[9]       ←'ff'⎕wi'*Create' 'zForm'('*caption' 'znetLinkLabel Demo')
[10]      ←'ff'⎕wi'*.lll.Create' 'znetLinkLabel'('wherelc'0 0 16 300)
[11]      ←'ff'⎕wi'*.lll.zText' 'Lescasse Consulting Web Site'
[12]      ←'ff'⎕wi'*.lll.zLink' 'http://www.lescasse.com/'
[13]
[14]      ←'ff'⎕wi'*.llz.Create' 'znetLinkLabel'('wherelc' '>' '=' '=' '=')
[15]      ←'ff'⎕wi'*.llz.zText' 'Notepad'
[16]      ←'ff'⎕wi'*.llz.zLink' 'notepad.exe'
[17]
[18]      ←'ff'⎕wi'*.llx.Create' 'znetLinkLabel'('wherelc' '>' '=' '=' '=')
[19]      ←'ff'⎕wi'*.llx.zText' 'APL+Win zObjects (default behavior)'
[20]      ←'ff'⎕wi'*.llx.zLink' 'c:\aplwin\zobjects\zobjects.w3'
[21]
[22]      ←'ff'⎕wi'*.lll4.Create' 'znetLinkLabel'('wherelc' '>' '=' '=' '=')
[23]      ←'ff'⎕wi'*.lll4.zText' 'APL+Win zObjects (custom behavior)'
[24]      ←'ff'⎕wi'*.lll4.zLink' 'c:\aplwin\zobjects\zobjects.w3'
[25]      ←'ff'⎕wi'*.lll4.zDisableDefault'1
[26]      ←'ff'⎕wi'*.lll4.onXLinkClicked' 'zznetlinklabel"lll4_onXLinkClicked"'
[27]
[28]      ←'ff'⎕wi'DemoShow'.2 .2 1 0
[29]  :case'lll4_onXLinkClicked'
[30]      ⎕←"The zobjects3 workspace was clicked!"
[31]  :endselect
    ∇
```

Try it:

```
      zznetlinklabel''
```

This should display the following form:



Copyright © 2016 Eric Lescasse

If you click on the 1st link, your default browser opens and displays my Web Site.

If you click on the 2nd link, **Notepad** opens.

If you click on the 3rd link, APL+Win gets loaded and the **zObjects** workspace loaded (provided you have a **c:\aplwin\zobjects\zobjects.w3** workspace on your machine).

If you click on the 4th link however, you should see the following being printed to the APL session:

```
The zobjects3 workspace was clicked!
```

even though the **zLink** property was set to the same value as the **zLink** for the 3rd **znetLinkLabel** object.

The znetMaskedTextBox object

Introduction

The **znetMaskedTextBox** object allows you to use a TextBox object that uses a mask to distinguish between proper and improper user input.

A **znetMaskedTextBox** control is very nice as it can help prevent users from entering wrong data.

The znetMaskedTextBox API

The **znetMaskedTexBox** contains the following main properties, methods and events:

```
Main Properties
---------------
*zAcceptsTab           *zMask                        *zResetOnSpace
*zAllowPromptAsInput   *zMaskCompleted               *zSelectedText
*zAsciiOnly            *zMaskFull                    *zSelectionLength
*zBeepOnError          *zMaskedTextProvider          *zSelectionStart
*zCanUndo              *zMaxLength                   *zShortcutsEnabled
*zCulture              *zModified                    *zSkipLiterals
*zCutCopyMaskFormat    *zMultiline                   *zTextAlign
*zFormatProvider       *zPasswordChar                *zTextLength
*zHidePromptOnLeave    *zPreferredHeight             *zTextMaskFormat
*zHideSelection        *zPromptChar                  *zUseSystemPasswordChar
*zInsertKeyMode        *zReadOnly                    *zValidatingType
*zIsOverwriteMode      *zRejectInputOnFirstFailure   *zWordWrap
*zLines                *zResetOnPrompt

Main Methods
------------
*zAppendText    *zDoc                              *zGetPositionFromCharIndex
*zClear         *zGetCharFromPosition              *zPaste
*zClearUndo     *zGetCharIndexFromPosition         *zScrollToCaret
*zCopy          *zGetFirstCharIndexFromLine        *zSelectAll
*zCut           *zGetFirstCharIndexOfCurrentLine   *zUndo
*zDeselectAll   *zGetLineFromCharIndex             *zValidateText

Main Events
-----------
*onXAcceptsTabChanged        *onXMaskChanged          *onXReadOnlyChanged
*onXBorderStyleChanged       *onXMaskInputRejected    *onXTextAlignChanged
*onXHideSelectionChanged     *onXModifiedChanged      *onXTypeValidationCompleted
*onXIsOverwriteModeChanged   *onXMultilineChanged
```

How to use znetMaskedTextBox

Load the znetMaskedTextBox object

```
]zload znetMaskedTextBox
```

Copyright © 2016 Eric Lescasse

Create an instance of the znetMaskedTextBox object

Since this object is a control, it must be hosted by a zForm or a container object which can be either an APL or a .Net object.

```
      ←'ff'⎕wi'*Create' 'zForm'('DemoShow'200 300 1 1)
      ←'ff'⎕wi'*.mtb1.Create' 'znetMaskedTextBox'('wherelc'00 20 200)('anchor'
'lrt')'*Paint'
```

Note that, because we showed the form before adding the **znetMaskedTextBox** to it, we need to use ***Paint** to force the **znetMaskedTextBox** to paint itself correctly.

Alternatively, we could write:

```
      ←'ff'⎕wi'*Create' 'zForm'('*size'200 300)
      ←'ff'⎕wi'*.mtb1.Create' 'znetMaskedTextBox'('wherelc'00 20 200)('anchor' 'lrt')
      ←'ff'⎕wi'DemoShow'0 0 1 1
```

Setting the zMask property

The next step is to set the **zMask** property to guide the user in entering a correct input into the control.

The **zMask** property is a string that describe what the user input should be.

For example, a **zMask** for a US phone number may be: **(000) 000-0000**

You can use the following characters in your mask

Masking element	Description
0	Digit, required. This element will accept any single digit between 0 and 9.
9	Digit or space, optional.
#	Digit or space, optional. If this position is blank in the mask, it will be rendered as a space in the **zText** property. Plus (+) and minus (-) signs are allowed.
L	Letter, required. Restricts input to the ASCII letters a-z and A-Z. This mask element is equivalent to [a-zA-Z] in regular expressions.
?	Letter, optional. Restricts input to the ASCII letters a-z and A-Z. This mask element is equivalent to [a-zA-Z]? in regular expressions.
&	Character, required. If the **zAsciiOnly** property is set to true, this element behaves like the "L" element.

Copyright © 2016 Eric Lescasse

C	Character, optional. Any non-control character. If the **zAsciiOnly** property is set to **true**, this element behaves like the "?" element.
A	Alphanumeric, required. If the **zAsciiOnly** property is set to **true**, the only characters it will accept are the ASCII letters a-z and A-Z. This mask element behaves like the "a" element.
a	Alphanumeric, optional. If the **zAsciiOnly** property is set to **true**, the only characters it will accept are the ASCII letters a-z and A-Z. This mask element behaves like the "A" element.
.	Decimal placeholder. The actual display character used will be the decimal symbol appropriate to the format provider, as determined by the control's **zFormatProvider** property.
,	Thousands placeholder. The actual display character used will be the thousands placeholder appropriate to the format provider, as determined by the control's **zFormatProvider** property.
:	Time separator. The actual display character used will be the time symbol appropriate to the format provider, as determined by the control's **zFormatProvider** property.
/	Date separator. The actual display character used will be the date symbol appropriate to the format provider, as determined by the control's **zFormatProvider** property.
$	Currency symbol. The actual character displayed will be the currency symbol appropriate to the format provider, as determined by the control's **zFormatProvider** property.
<	Shift down. Converts all characters that follow to lowercase.
>	Shift up. Converts all characters that follow to uppercase.
|	Disable a previous shift up or shift down.
\	Escape. Escapes a mask character, turning it into a literal. "\\" is the escape sequence for a backslash.
All other characters	Literals. All non-mask elements will appear as themselves within **znetMaskedTextBox**. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user.

Copyright © 2016 Eric Lescasse

Using the onXMaskInputRejected method

The **onXMaskInputRejected** event fires as soon as a user enters an invalid character in a **znetMaskedTextBox.**

You can use this event to display hints about the error the user has made in a status bar and/or to beep a sound.

When the **onXMaskInputRejected** event fires, ⎕warg is a 2-element integer vector and contains:

⎕warg[1] = the position at which an error occurred
⎕warg[2] = the reason for this error

The second element corresponds to the values of the following .Net enum:
System.ComponentModel.MaskedTextResultHint

Note:
You can use the **EnumDoc** method to programmatically retrieve the names and values of any .Net enum as a 2-column nested matrix.

Example:

```
      'ff'⎕wi'EnumDoc' 'System.ComponentModel.MaskedTextResultHint'
 ¯55 PositionOutOfRange
 ¯54 NonEditPosition
 ¯53 UnavailableEditPosition
 ¯52 PromptCharNotAllowed
 ¯51 InvalidInput
  ¯5 SignedDigitExpected
  ¯4 LetterExpected
  ¯3 DigitExpected
  ¯2 AlphanumericCharacterExpected
  ¯1 AsciiCharacterExpected
   0 Unknown
   1 CharacterEscaped
   2 NoEffect
   3 SideEffect
   4 Success
```

Copyright © 2016 Eric Lescasse

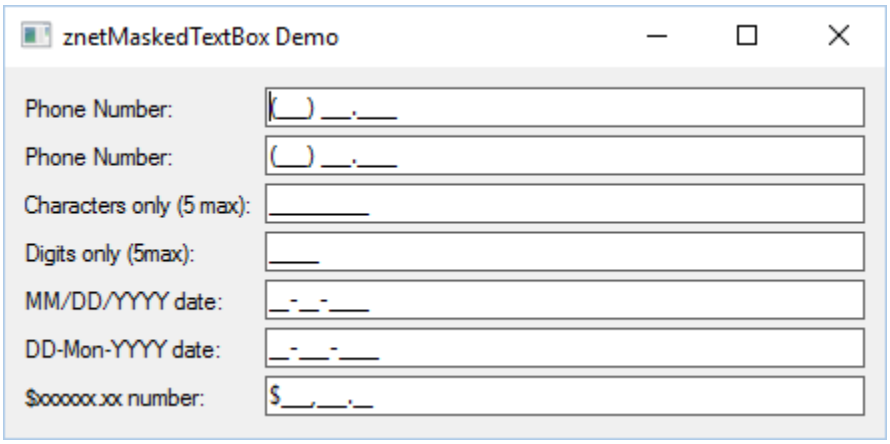
The sample zznetmaskedtextbox function

The following function demonstrates how to use the **znetMaskedTextBox**:

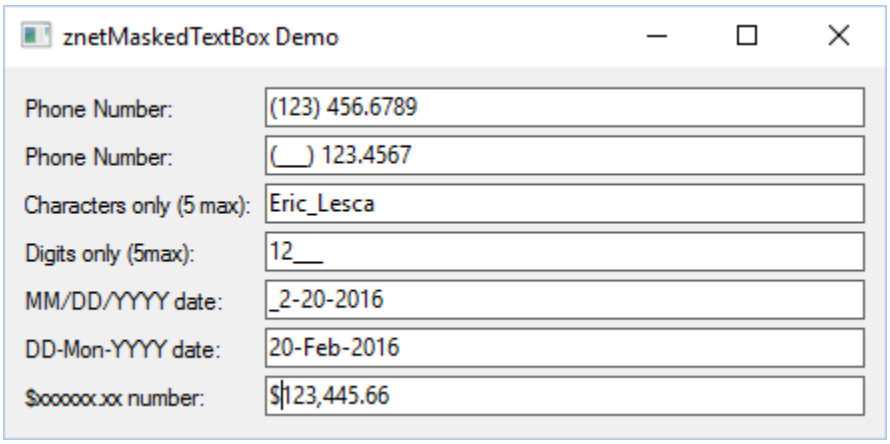
```
    ∇ zznetmaskedtextbox a;b;c;d
[1]   ⍝∇ Sample function demonstrating how to use the znetMaskedTextBox
[2]
[3]   :select a
[4]   :case''
[5]       ←'ff'⎕wi'*Create' 'zForm'('*size'250 500)('*caption' 'znetMaskedTextBox Demo')
[6]       ←'ff'⎕wi'*.mtb1.Create' 'znetMaskedTextBox'('whereIc'0 130 20 300)
              ('anchor' 'lrt')
[7]       ←'ff'⎕wi'*.mtb2.Create' 'znetMaskedTextBox'('whereIc' '>' '=' '=' '=')
              ('anchor' 'lrt')
[8]       ←'ff'⎕wi'*.mtb3.Create' 'znetMaskedTextBox'('whereIc' '>' '=' '=' '=')
              ('anchor' 'lrt')
[9]       ←'ff'⎕wi'*.mtb4.Create' 'znetMaskedTextBox'('whereIc' '>' '=' '=' '=')
              ('anchor' 'lrt')
[10]      ←'ff'⎕wi'*.mtb5.Create' 'znetMaskedTextBox'('whereIc' '>' '=' '=' '=')
              ('anchor' 'lrt')
[11]      ←'ff'⎕wi'*.mtb6.Create' 'znetMaskedTextBox'('whereIc' '>' '=' '=' '=')
              ('anchor' 'lrt')
[12]      ←'ff'⎕wi'*.mtb7.Create' 'znetMaskedTextBox'('whereIc' '>' '=' '=' '=')
              ('anchor' 'lrt')
[13]
[14]      ←'mtb1'⎕wi'caption' 'Phone Number:'
[15]      ←'mtb2'⎕wi'caption' 'Phone Number:'
[16]      ←'mtb3'⎕wi'caption' 'Characters only (5 max):'
[17]      ←'mtb4'⎕wi'caption' 'Digits only (5max):'
[18]      ←'mtb5'⎕wi'caption' 'MM/DD/YYYY date:'
[19]      ←'mtb6'⎕wi'caption' 'DD-Mon-YYYY date:'
[20]      ←'mtb7'⎕wi'caption' '$xxxxxx.xx number:'
[21]
[22]      ←'mtb1'⎕wi'*zMask' '(000) 000.0000'
[23]      ←'mtb2'⎕wi'*zMask' '(999) 000.0000'
[24]      ←'mtb3'⎕wi'*zMask' (5⍴'?')
[25]      ←'mtb4'⎕wi'*zMask' (5⍴'0')
[26]      ←'mtb5'⎕wi'*zMask' '00/00/0000'
[27]      ←'mtb6'⎕wi'*zMask' '00->L<LL-0000'
[28]      ←'mtb7'⎕wi'*zMask' '$999,999.00'
[29]
[30]      ←'ff'⎕wi'AutoSize'
[31]      ←'ff'⎕wi'CenterScreen'
[32]      ←'ff'⎕wi'Show'
[33]
[34]      ⍝ Events
[35]      ←((⊂'mtb'),¨⍕¨⍳7)⎕wi¨⊂'*onXMaskInputRejected' 'zznetmaskedtextbox
              "onXMaskInputRejected"'
[36]
[37]  :case'onXMaskInputRejected'
[38]      (b c)←⎕warg
[39]      d←⎕wi'EnumDoc' 'System.ComponentModel.MaskedTextResultHint'
[40]      ⎕←((d[;1]⍳c)⊃d[;2],⊂''),' at position ',(⍕b),' in ',⎕wi'*name'
[41]
[42]  :endselect
    ∇
```

Let's use it:

```
      zznetmaskedtextbox''
```



After entering input in the various fields, we may see:



If you type illegal characters you would get the following warnings printed in the APL Session:

```
      zznetmaskedtextbox''
DigitExpected at position 1 in mtb1
DigitExpected at position 1 in mtb1
DigitExpected at position 1 in mtb1
DigitExpected at position 1 in mtb1
DigitExpected at position 1 in mtb1
UnavailableEditPosition at position 14 in mtb1
UnavailableEditPosition at position 14 in mtb1
LetterExpected at position 2 in mtb3
LetterExpected at position 2 in mtb3
LetterExpected at position 2 in mtb3
DigitExpected at position 3 in mtb4
DigitExpected at position 3 in mtb4
DigitExpected at position 3 in mtb4
DigitExpected at position 3 in mtb4
UnavailableEditPosition at position 11 in mtb7
DigitExpected at position 1 in mtb7
UnavailableEditPosition at position 11 in mtb7
```

Copyright © 2016 Eric Lescasse

In a real life application, you could display similar messages in the status bar of your application with a small delay.

Copyright © 2016 Eric Lescasse

The znetNotifyIcon object

Introduction

The znetNotifyIcon object allows you to programmatically display Windows-like Notification messages at the bottom right of your screen.

How to use znetNotifyIcon

Loading the znetNotificationIcon object

```
]zload znetNotifyIcon zznetnotifyicon
```

The **zznetnotifyicon** function is a sample function showing how to use znetNotifyIcon.

Creating an instance of the znetNotifyIcon object

```
←'not'⎕wi'*Create' 'znetNotifyIcon'
```

The znetNotifyIcon API

The znetNotifyIcon object has a pretty simple API:

```
      'not'⎕wi'mainprops'
Main Properties
---------------
*zBalloonTipIcon  *zBalloonTipText  *zBalloonTipTitle

Main Methods
------------
*zDoc  *zShowBalloonTip  *zShowBalloonTip_2

Main Events
-----------
*onXBalloonTipClicked  *onXBalloonTipClosed  *onXBalloonTipShown
```

Showing a notification message

There are 5 steps to displaying a notification icon:

1. First always set the ***zIcon** property to **'notify'**
2. Choose the icon you want to display by setting the ***zBalloonTipIcon** property
3. Set the Title you want to display using the ***zBalloonTipTitle** property
4. Set the Text you want to display using the ***zBalloonTipText** property
5. Call the ***zShowBalloonTip** to display the notification message

Examples:

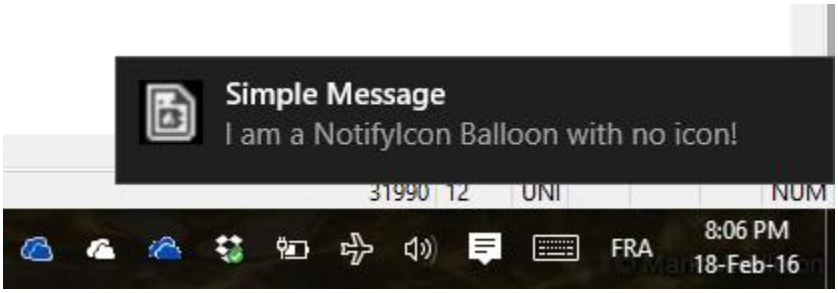
```
←'ni'⎕wi'*zIcon' 'notify'
```

Copyright © 2016 Eric Lescasse

Show a notification with no icons

```
←'ni'□wi'*zBalloonTipIcon'0                    ⍝ 0=ToolTipIcon.None;
←'ni'□wi'*zBalloonTipText' 'I am a NotifyIcon Balloon with no icon!'
←'ni'□wi'*zBalloonTipTitle' 'Simple Message'
←'ni'□wi'*zShowBalloonTip'200
```

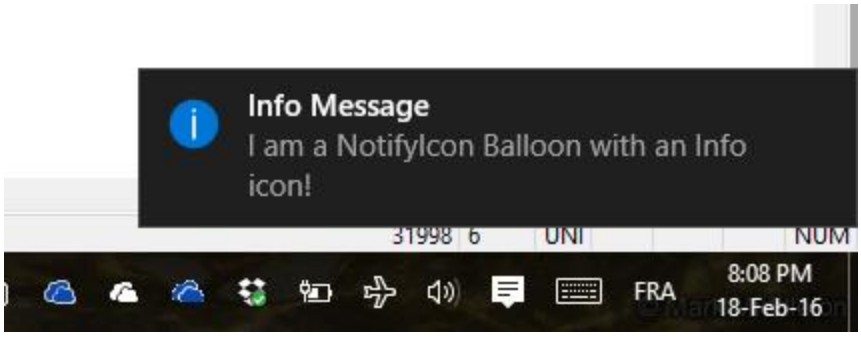
The following is displayed at the bottom right of the screen on my Windows 10 machine:



Show a notification with an Info icon

```
←'ni'□wi'*zBalloonTipIcon'1                    ⍝ 1=ToolTipIcon.Info;
←'ni'□wi'*zBalloonTipText' 'I am a NotifyIcon Balloon with an Info icon!'
←'ni'□wi'*zBalloonTipTitle' 'Info Message'
←'ni'□wi'*zShowBalloonTip'200
```

The following is displayed at the bottom right of the screen on my Windows 10 machine:

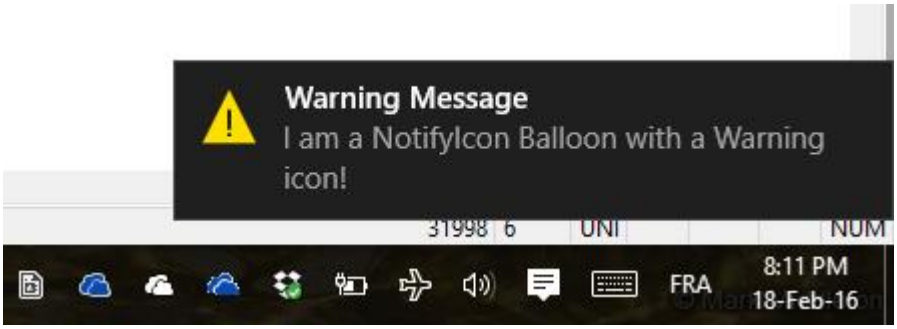


Show a notification with a Warning icon

```
←'ni'□wi'*zBalloonTipIcon'2                    ⍝ 2=ToolTipIcon.Warning;
←'ni'□wi'*zBalloonTipText' 'I am a NotifyIcon Balloon with a Warning icon!'
←'ni'□wi'*zBalloonTipTitle' 'Warning Message'
←'ni'□wi'*zShowBalloonTip'200
```

The following is displayed at the bottom right of the screen on my Windows 10 machine:

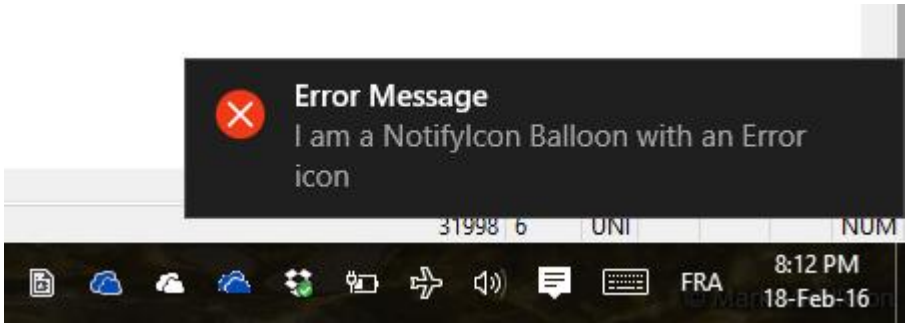
Copyright © 2016 Eric Lescasse



Show a notification with an Error icon

```
      ←'ni'⍞wi'*zBalloonTipIcon'3                    ⍝ 3=ToolTipIcon.Error;
      ←'ni'⍞wi'*zBalloonTipText' 'I am a NotifyIcon Balloon with an Error icon'
      ←'ni'⍞wi'*zBalloonTipTitle' 'Error Message'
      ←'ni'⍞wi'*zShowBalloonTip'200
```

The following is displayed at the bottom right of the screen on my Windows 10 machine:



Events

The **zznetnotifyicon** sample function reads:

```
      ∇ zznetnotifyicon a
[1]     ⍝∇ This function demonstrates how to use the znetNotifyIcon control
[2]
[3]     :select a
[4]     :caselist 0 1 2 3
[5]         ←'ni'⍞wi'*Create' 'znetNotifyIcon'
[6]         ⍝ Note: you MUST set the zIcon property before using the other properties
[7]         ←'ni'⍞wi'*zIcon' 'notify'
[8]         ←'ni'⍞wi'*onXBalloonTipClicked' 'zznetnotifyicon"onXBalloonTipClicked"'
[9]         ←'ni'⍞wi'*onXBalloonTipClosed' 'zznetnotifyicon"onXBalloonTipClosed"'
[10]        ←'ni'⍞wi'*onXBalloonTipShown' 'zznetnotifyicon"onXBalloonTipShown"'
[11]        :select a
[12]        :case 0
[13]            ←'ni'⍞wi'*zBalloonTipIcon'0              ⍝ 0=ToolTipIcon.None
[14]            ←'ni'⍞wi'*zBalloonTipText' 'I am a NotifyIcon Balloon with no icon!'
[15]            ←'ni'⍞wi'*zBalloonTipTitle' 'Simple Message'
[16]            ←'ni'⍞wi'*zShowBalloonTip'200
[17]        :case 1
[18]            ←'ni'⍞wi'*zBalloonTipIcon'1              ⍝ 1=ToolTipIcon.Info
```

```
[19]          ←'ni'⎕wi'*zBalloonTipText' 'I am a NotifyIcon Balloon with an Info
icon!'
[20]          ←'ni'⎕wi'*zBalloonTipTitle' 'Info Message'
[21]          ←'ni'⎕wi'*zShowBalloonTip'200
[22]      :case 2
[23]          ←'ni'⎕wi'*zBalloonTipIcon'2              ⍝ 2=ToolTipIcon.Warning
[24]          ←'ni'⎕wi'*zBalloonTipText' 'I am a NotifyIcon Balloon with a Warning
icon!'
[25]          ←'ni'⎕wi'*zBalloonTipTitle' 'Warning Message'
[26]          ←'ni'⎕wi'*zShowBalloonTip'200
[27]      :case 3
[28]          ←'ni'⎕wi'*zBalloonTipIcon'3              ⍝ 3=ToolTipIcon.Error
[29]          ←'ni'⎕wi'*zBalloonTipText' 'I am a NotifyIcon Balloon with an Error
icon'
[30]          ←'ni'⎕wi'*zBalloonTipTitle' 'Error Message'
[31]          ←'ni'⎕wi'*zShowBalloonTip'200
[32]      :else
[33]          ⎕←'Please use an integer argument of 0 to 3!'
[34]      :endselect
[35]  :case"onXBalloonTipClicked"
[36]      ⎕←⎕wevent⎕warg
[37]  :case"onXBalloonTipClosed"
[38]      ⎕←⎕wevent⎕warg
[39]  :case"onXBalloonTipShown"
[40]      ⎕←⎕wevent⎕warg
[41]  :endselect
    ∇
```

Try:

```
      zznetnotifyicon 2
```

and leave it live its life!

You should see the following printed in the APL Session proving that the events fire ok:

```
XBalloonTipShown
XBalloonTipClosed
```

If you instead click in the middle of the notify icon while it is displayed, you get the following events (in that case the XBalloonTipClosed event does not fire):

```
XBalloonTipShown
XBalloonTipClicked
```

If instead you click the notify close button while it is displayed, you get the following events (in that case the XBalloonTipClicked does not fire):

```
XBalloonTipShown
XBalloonTipClosed
```

Copyright © 2016 Eric Lescasse

The znetProcess object

Introduction

The znetProcess object allows you to easily start any process programmatically and possibly capture its result.

The znetProcess API

The main **znetProcess** properties, methods and events are:

```
      'pp'wi'*Create' 'znetProcess'
      'pp'wi'mainprops'
Main Properties
---------------
*zBasePriority                *zPeakVirtualMemorySize
*zEnableRaisingEvents         *zPeakVirtualMemorySize64
*zExitCode                    *zPeakWorkingSet
*zExitTime                    *zPeakWorkingSet64
*zHandleCount                 *zPriorityBoostEnabled
*zHasExited                   *zPriorityClass
*zId                          *zPrivateMemorySize
*zMachineName                 *zPrivateMemorySize64
*zMainWindowHandle            *zPrivilegedProcessorTime
*zMainWindowTitle             *zProcessName
*zMaxWorkingSet               *zProcessorAffinity
*zMinWorkingSet               *zResponding
*zNonpagedSystemMemorySize    *zSessionId
*zNonpagedSystemMemorySize64  *zStartTime
*zPagedMemorySize             *zTotalProcessorTime
*zPagedMemorySize64           *zUserProcessorTime
*zPagedSystemMemorySize       *zVirtualMemorySize
*zPagedSystemMemorySize64     *zVirtualMemorySize64
*zPeakPagedMemorySize         *zWorkingSet
*zPeakPagedMemorySize64       *zWorkingSet64

Main Methods
------------
*zBeginErrorReadLine   *zCloseMainWindow  *zStart        *zWaitForExit_2
*zBeginOutputReadLine  *zDoc              *zStart_2      *zWaitForInputIdle
*zCancelErrorRead      *zEnterDebugMode   *zStart_3      *zWaitForInputIdle_2
*zCancelOutputRead     *zKill             *zStart_4
*zClose                *zLeaveDebugMode   *zWaitForExit

Main Events
-----------
*onXErrorDataReceived  *onXExited  *onXOutputDataReceived
```

The most useful method is the **zStart** method and its variants.

Copyright © 2016 Eric Lescasse

How to use the znetProcess object

Loading the znetProcess object

```
]zload znetProcess
```

Creating an instance of znetProcess

```
←'pp'⎕wi'*Create' 'znetProcess'
```

Various ways to use the zStart method

The **znetProcess zStart** method (or its variants) is used to programmatically start a process or application.

The **zStart** method can be used in 2 main ways:

1. By directly starting the process
2. By first creating a **zStartInfo** object and then passing it to the **zStart** method

The various **zStart** methods syntaxes are the following:

```
      'pp'⎕wi'allsyntax' 'start'
Result ← ⎕wi'*zStart'
⎕wi'*zStart_2'filename@String
⎕wi'*zStart_3'filename@String arguments@String
```

The 2nd and 3rd syntax are used for a direct **zStart** call.
The 1st syntax requires the use of a **zStartInfo** property
There's also a 4th syntax (**zStart_4**) but it can't be used from APL.

Directly using zStart or one of its variants

Programmatically starting an application

```
←'pp'⎕wi'*zStart_2' 'c:\windows\notepad.exe'
```

Since c:\windows is in the Path variable in any PC, the following works fine as well:

```
←'pp'⎕wi'*zStart_2' 'notepad.exe'
```

and since **znetProcess** assumes a **.exe** extension for an application, you can even simplify the call to:

```
←'pp'⎕wi'*zStart_2' 'notepad'
```

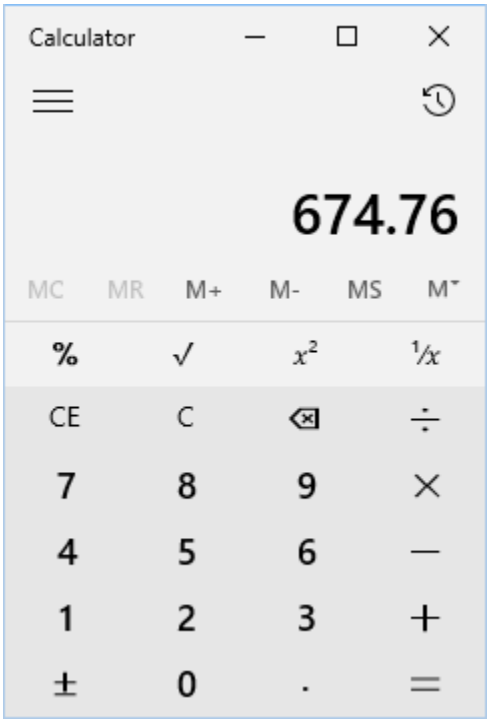
Note that you can start any **.exe** process this way: try:

```
←'pp'⎕wi'*zStart_2' 'c:\aplwin15.1\aplw'
←'pp'⎕wi'*zStart_2' 'aplw'
```

Copyright © 2016 Eric Lescasse

Or you can start the Windows Calculator:

```
      'pp'⎕wi'*zStart_2' 'calc'
```



Note:
Note that you can create an instance of the znetProcess object and start an application in just one instruction:

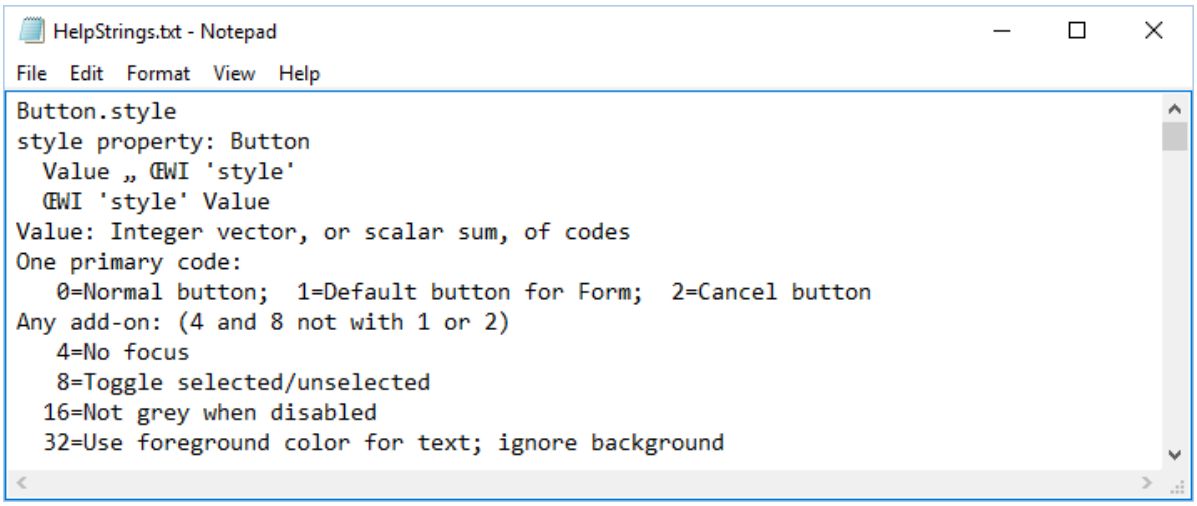
```
      ←'pp'⎕wi'*Create' 'znetProcess'('*zStart_2' 'notepad')
      ←'pp'⎕wi'*Create' 'znetProcess'('*zStart_2' 'calc')
```

Starting an application and loading a document

Use the **zStart_3** method to start an application and load a document in the application;

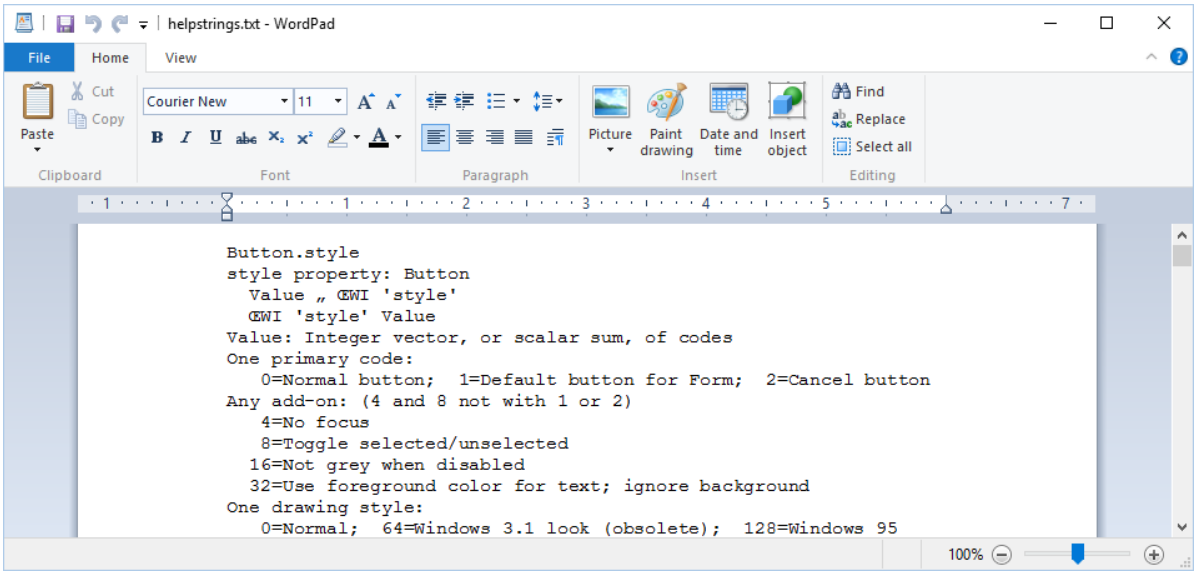
```
      ←'pp'⎕wi'*zStart_3' 'notepad' 'c:\aplwin15.1\helpstrings.txt'
```

Copyright © 2016 Eric Lescasse



When using the **zStart_3** method, you have the choice of the application into which you want to load the document:

```
←'pp'⎕wi'*zStart_3' 'wordpad' 'c:\aplwin15.1\helpstrings.txt'
```



Starting an application by file association

If a file extension has been registered with an application association in Windows, you can start the application and load the document by only specifying the document.

Be careful to use the **zStart_2** method (not **zStart_3**) in this case since there will be only one argument.

Try these various examples:

```
←'pp'⎕wi'*zStart_2' 'c:\aplwin15.1\helpstrings.txt'
←'pp'⎕wi'*zStart_2' 'c:\aplwin15.1\aplgui.chm'
←'pp'⎕wi'*zStart_2' 'c:\aplwin15.1\readme.pdf'
```

Copyright © 2016 Eric Lescasse

Each time, the application that is associated with the extension is automatically started and the document loaded in it.

Using zStart with a zStartInfo object

If you want to have more control about the process you want to start, you can use the **zStart** method.

It requires that you create **zStartInfo** object: you can do that using the **znetProcess zStart-Info** property, passing it a 3 to 16-element nested vector having the following structure:

[1]= filename (string)
[2]= arguments (string)
[3]= redirect standard output (bool)
[4]= redirect standard input (bool)
[5]= working directory (string)
[6]= use shell execute (bool)
[7]= create no window (bool)
[8]= domain (string)
[9]= error dialog (bool)
[10]= error dialog parent handler (int)
[11]= load user profile (bool)
[12]= (not used in this version) password (string)
[13]= redirect standard error
[14]= verb (string)
[15]= (not used)
[16]= window style (int [enum])

You can use 0 or '' for the elements you don't want to define.

Note that the **zStart** method returns a result so you can capture this result in your application.

Here is an example

```
c←7⍴⊂''
c[1]←⊂'cmd.exe'
c[2]←⊂'/C dir'       ⍝ note: /C is compulsory to terminate the process
c[3]←1
c[6]←0
c[7]←1
←'pp'⎕wi'*zStartInfo'c
aaa←⎕tclf~⍨'pp'⎕wi'*zStart'
```

By running this code we can run the DOS dir command in the current folder and capture its result instantaneously. Let's check the result:

```
      aaa
```

```
 Volume in drive C is Windows
 Volume Serial Number is 60B7-FE87

 Directory of C:\APLWIN15.1

18-Feb-16  01:28 PM    <DIR>          .
18-Feb-16  01:28 PM    <DIR>          ..
11-Dec-15  03:59 PM         1,085,082 )OFF YES.w3
02-Mar-15  04:10 PM           347,432 APL+Win EULA - 20150302.pdf
18-Feb-16  01:28 PM        12,378,605 apl.log
17-Aug-15  04:10 PM    <DIR>          APLDraw
14-Aug-15  02:07 PM           598,976 apldraw.ocx
14-Aug-15  02:07 PM           251,840 AplFileShExt.dll
...
31-Jan-16  12:55 AM    <DIR>          Tools
14-Feb-16  07:17 PM           272,656 UCMDS.SF
11-Aug-14  12:11 PM           127,868 UCMDS2.SF
11-Aug-14  12:11 PM           548,100 UCMDS3.SF
11-Aug-14  12:11 PM           447,252 UCMDSW.SF
14-Aug-15  02:07 PM           178,112 unzip32static.dll
14-Aug-15  12:12 PM           884,382 Version_History.pdf
14-Aug-15  02:07 PM           157,632 zip32.dll
              60 File(s)     53,933,431 bytes
               9 Dir(s)  296,263,008,256 bytes free
```

To help you run any DOS command and capture its result, the znetProcess object contains an APL **GetDos** method[11].

Let's use it to programmatically capture some important information:

```
      aaa←'pp'⎕wi'GetDos' 'ipconfig /all'
      ⍴aaa
3399
      aaa

Windows IP Configuration

   Host Name . . . . . . . . . . . . : SPro3
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : home

Ethernet adapter Ethernet 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Surface Ethernet Adapter

. . .

Tunnel adapter isatap.home:
```

[11] Since it is a method defined in APL in the znetProcess object itself and not a method defined in C#, it should be used without the *z prefix.

```
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : home
   Description . . . . . . . . . . . : Microsoft ISATAP Adapter #3
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
```

The **GetDos** command reads:

```
:region-----------------------------------------GetDos
∆GetDos:
    ⍝∇ Runs a DOS command and returns its result
    ⍝∇ Syntax: result←'obj'⎕wi'GetDos'command
    ⍝∇ command:  the DOS command to run
    ⍝∇ result: the DOS command result (string)
    ⍝∇ Example:
    ⍝∇       'process'⎕wi'*Create' 'znetProcess'
    ⍝∇       'process'⎕wi'GetDos' 'dir'
    ⍝∇       'process'⎕wi'GetDos' 'ipconfig /all'
    c←7⍴⊂''
    c[1]←⊂'cmd.exe'
    c[2]←⊂'/C ','⊢/⊂⊢'zzTEXTREPL 2⊃⎕warg  ⍝ /C is compulsory to terminate the process
    c[3]←1
    c[6]←0
    c[7]←1
    ←⎕wi'*zStartInfo'c
    ⎕wres←⎕tclf~⍨⎕wi'*zStart'
    :return
    :endregion
```

As you can see you set the **zStartInfo** property and once it is set, you call the **zStart** method without any argument.

Other uses of the znetProcess object

You can use the znetProcess object for several other tasks. Here are a few examples:

Opening File Explorer at a given folder

Try:

```
'pp'⎕wi'*zStart_2' 'c:\aplwin15.1'
```

Using znetProcess to load a Web Site

Try:

```
'pp'⎕wi'*zStart_2' 'http://www.google.com'
```

This starts your default browser and loads the Google page into it.

You can even do the same thing and do a search at the same time.

Try:

```
'pp'□wi'*zStart_2' 'http://www.google.com/search?q=lescasse'
```

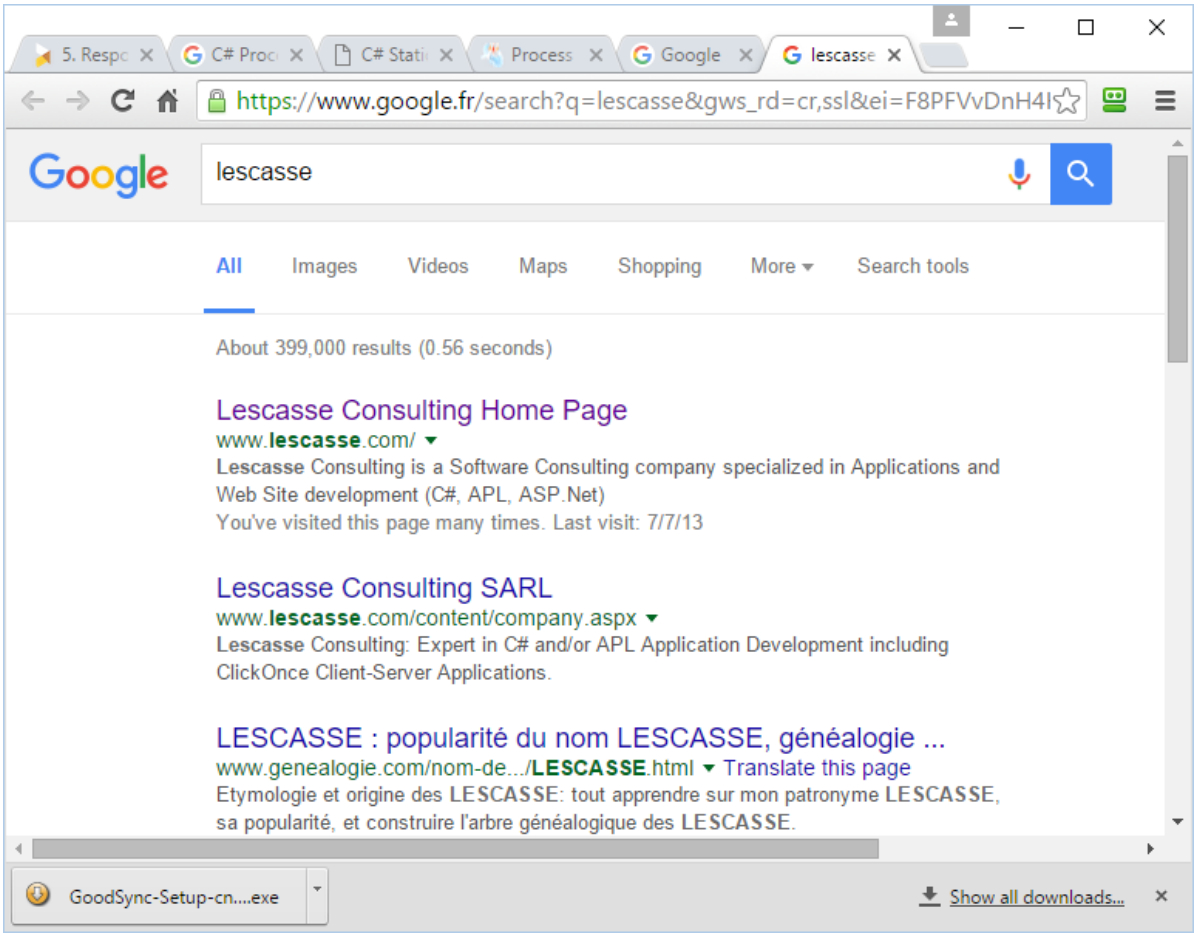
Hopefully, my Home page comes first!

To make this process of loading Google programmatically or doing a Google search programmatically, the **znetProcess** class provides a **Google** method.

Try:

```
'pp'□wi'Google'
'pp'□wi'Google' 'lescasse'
```

Note that methods do not need to be complex in zObjects: you may want to write a method just to reduce typing and/or make the call intention clearer (as the **Google** method does).



Waiting for the process to terminate

In some cases, you would like to have your application wait for the process you started with **znetProcess** to terminate, before your application goes on.

Using the zWaitForExit method

Her eis a function that demonstrates how to use the ***zWaitForExit** method.

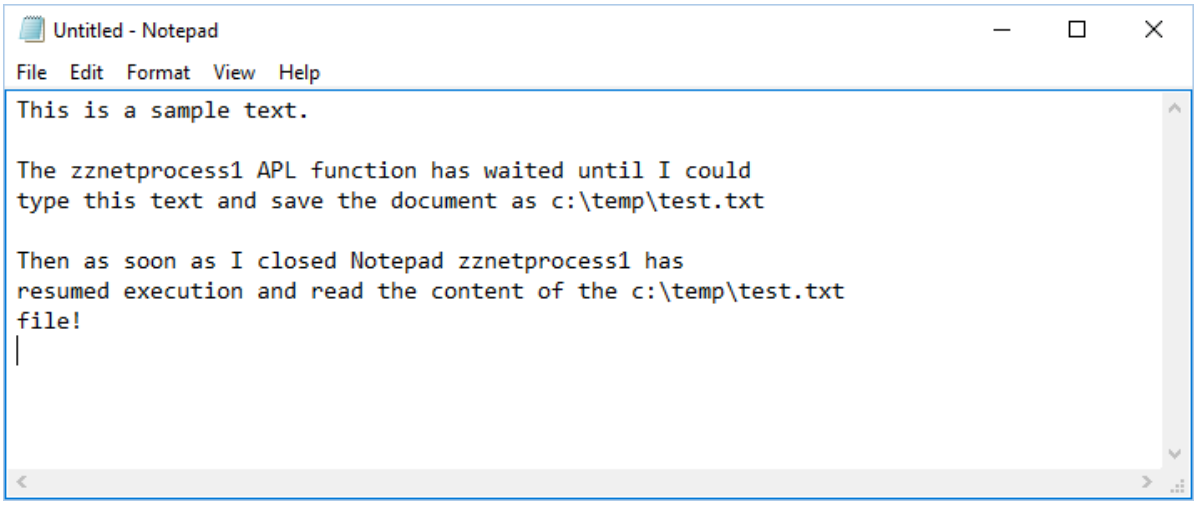
Example:

```
    ∇ zznetprocess1
[1]   ⍝ Demonstrating the zWaitForExit method
[2]
[3]   ←'pp'⎕wi'*Create' 'znetProcess'
[4]   ←'pp'⎕wi'*zStart_2' 'notepad'
[5]   ←'pp'⎕wi'*zWaitForExit'
[6]
[7]   ⍝ Assuming we have saved the file as c:\temp\test.txt
[8]   ←'io'⎕wi'*Create' 'znetFile'
[9]   ⎕←⎕tclf~⍨'io'⎕wi'*zReadAlltext' 'c:\temp\test.txt'
    ∇
```

Let's test this function.

Type the following text (or any text) once Notepad has started, then save the document as **c:\temp\test.txt**.

```
      zznetprocess1
```



Now close Notepad. As soon as you do this, you should see the following text printed to your APL session:

```
This is a sample text.

The zznetprocess1 APL function has waited until I could
type this text and save the document as c:\temp\test.txt

Then as soon as I closed Notepad zznetprocess1 has
resumed execution and read the content of the c:\temp\test.txt
file!
```

Copyright © 2016 Eric Lescasse

The znetRegexTest object

Introduction

Regular Expressions exist in all languages on the planet … except in APL+Win! This may be a slight exaggeration but is mostly true.

Regular Expressions are extremely important for a developer: they are used throughout applications for various purposes such as validating user input, searching text, replacing strings by other strings in text, extract information from a document based on a pattern match, etc.

zObjects include a **znetRegex** object, but before we look at this object, let's describe the **znetRegexTest** object which allows you to visually test Regular Expressions.

The sample text

The sample text we will use to explore Regular Expressions and **znetRegexTest** is extracted from the following Web page:

https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Dates_and_numbers

How to use znetRegexTest

First let's load the **znetRegexTest** object from the zObjects file:

```
]zload znetRegexTest
```

Now, let's create an instance of **znetRegexTest**:

```
'ff'□wi'*Create' 'znetRegexTest' ('DemoShow' 700 1140 1 1)
```

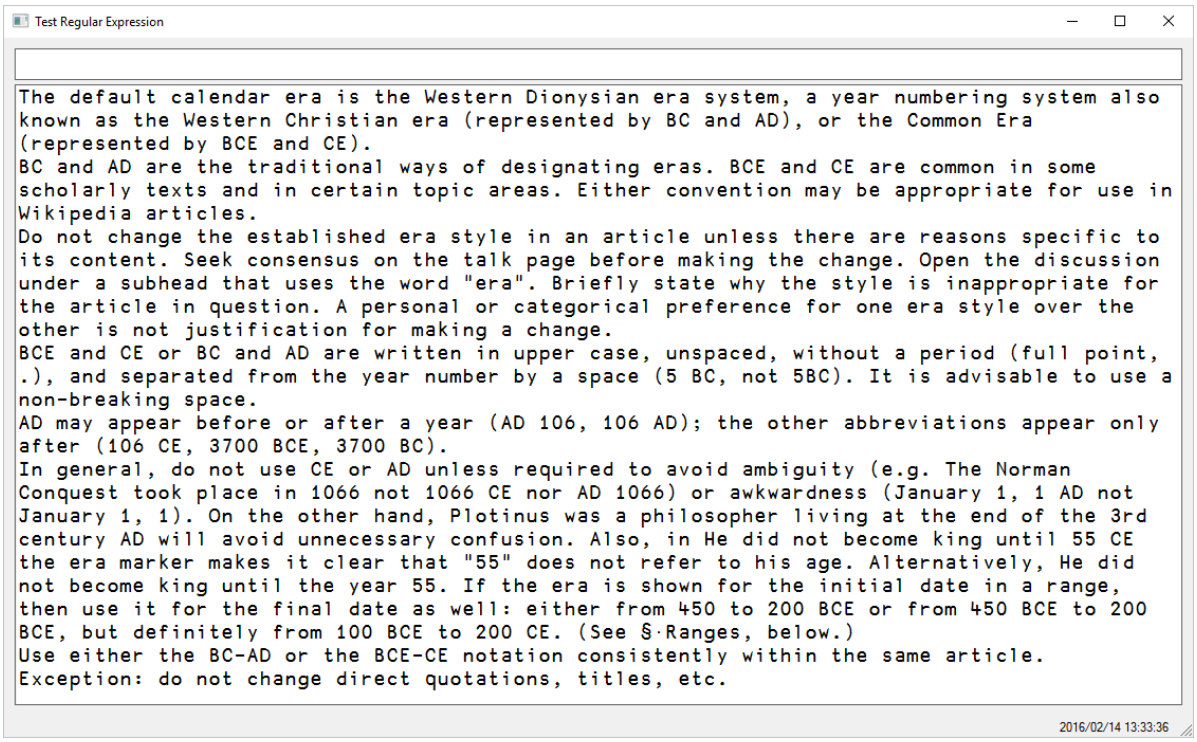
You use the **znetRegexTest** application by pasting the text[12] you want to search with a Regular Expression in the **znetRichTextBox** (the large control in the **znetRegexTest** object).

Let's copy the **Era style** paragraph from the above Web page and paste it in the **znetRegextTest znetRichTextBox** control:

We should see:

[12] The text you paste in the **znetRichTextBox** can be APL+Win code as the default font in **znetRegexTest** is the **APL385 Unicode** font.

Copyright © 2016 Eric Lescasse



Now you can enter any Regular Expression in the top Edit control and all elements in the text that match the regular expression will be highlighted.

For example, type **3700** or **king**. A simple text is a basic Regular Expression!

But assume we would like to find and extract all the numbers contained in the text.

In Regular Expression language, **\d** matches a single digit. If you type **\d** in the top Edit control, you'll see that all numbers are selected. This is because each individual digit of each number is selected.

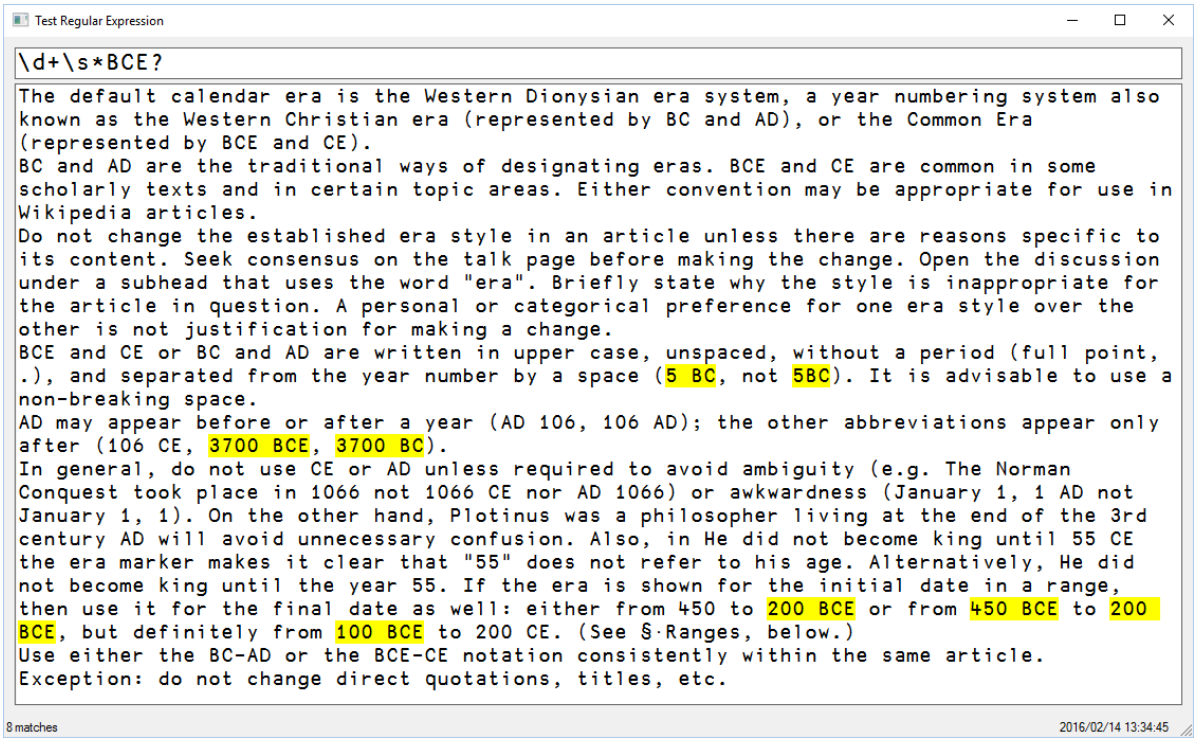
In fact, the status bar indicates that 60 matches were found!

Now let's find all the numbers which represents dates before Christ, i.e. number followed by BC or BCE with or without spaces separating the number from BC or BCE.

Type the following Regular Expression:

Copyright © 2016 Eric Lescasse

\d+\s*BCE?



Let's explain this regular expression:

- **\d** means a digit
- **+** means **one or more**, so **\d+** means one or more digits
- **\s** means a space and ***** means **0 or more** so **\s*** means **0 or more spaces**
- **BC** matches **BC**
- **?** means **0 or 1**, so **E?** matches **0 or one E**

Finally, our Regular Expression means:

Match a number (i.e. at least one digit), followed by 0 or more spaces followed by the BC letters followed by 0 or 1 E

Another example involving APL code

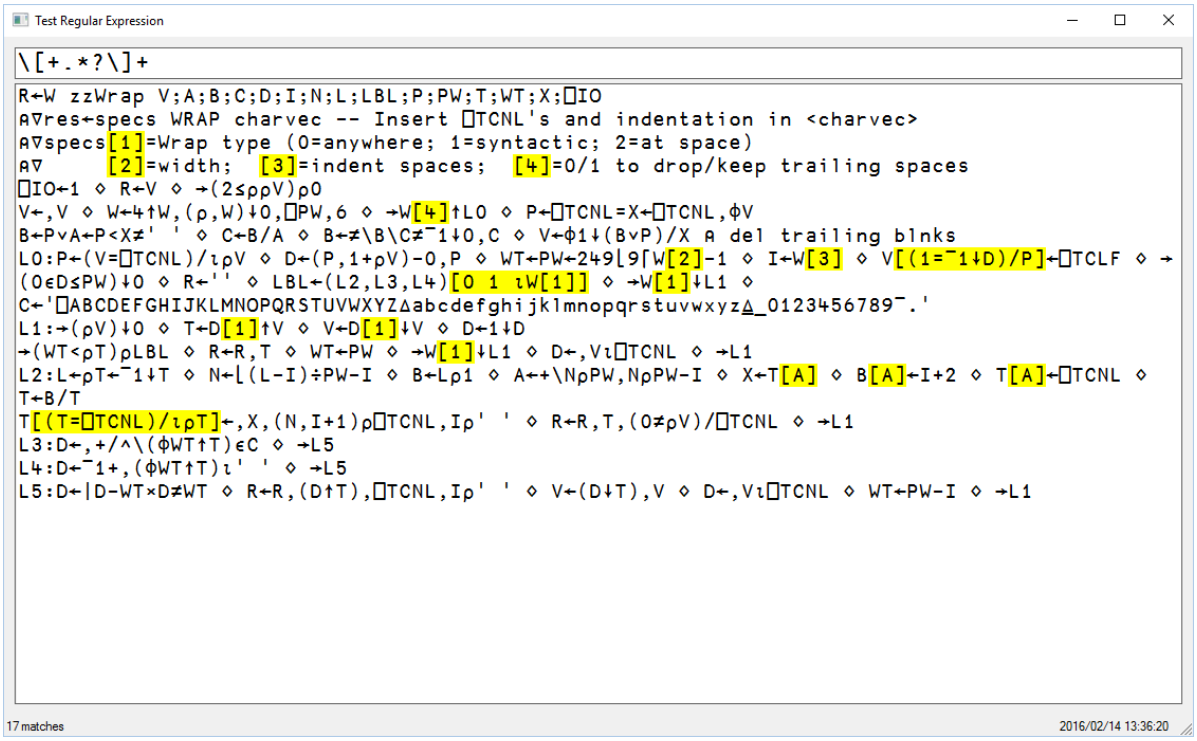
Copy/paste the ☐cr of the **zzWrap** function into the **znetRichTextBox** control.

We now want to find all expressions within brackets.

Type the following Regular Expression: **\[+.*?\]+**

Copyright © 2016 Eric Lescasse

Here is what you get:




```
R←W zzWrap V;A;B;C;D;I;N;L;LBL;P;PW;T;WT;X;⎕IO
∇res←specs WRAP charvec -- Insert ⎕TCNL's and indentation in <charvec>
∇specs[1]=Wrap type (0=anywhere; 1=syntactic; 2=at space)
∇      [2]=width;  [3]=indent spaces;  [4]=0/1 to drop/keep trailing spaces
⎕IO←1 ⋄ R←V ⋄ →(2≤ρρV)ρ0
V←,V ⋄ W←4↑W,(ρ,W)↓0,⎕PW,6 ⋄ →W[4]↑L0 ⋄ P←⎕TCNL=X←⎕TCNL,⍵V
B←PvA←P<X≠' ' ⋄ C←B/A ⋄ B←≠\B\C≠¯1↓0,C ⋄ V←⍵1↓(BvP)/X ⍝ del trailing blnks
L0:P←(V=⎕TCNL)/⍳ρV ⋄ D←(P,1+ρV)-0,P ⋄ WT←PW←249⌊9⌈W[2]-1 ⋄ I←W[3] ⋄ V[(1=¯1↓D)/P]←⎕TCLF ⋄ →
(0∈D≤PW)↓0 ⋄ R←'' ⋄ LBL←(L2,L3,L4)[0 1 ⍳W[1]] ⋄ →W[1]↓L1 ⋄
C←'⎕ABCDEFGHIJKLMNOPQRSTUVWXYZΔabcdefghijklmnopqrstuvwxyzΔ_0123456789¯.'
L1:→(ρV)↓0 ⋄ T←D[1]↑V ⋄ V←D[1]↓V ⋄ D←1↓D
→(WT<ρT)ρLBL ⋄ R←R,T ⋄ WT←PW ⋄ →W[1]↓L1 ⋄ D←,V⍳⎕TCNL ⋄ →L1
L2:L←ρT←¯1↓T ⋄ N←⌊(L-I)÷PW-I ⋄ B←Lρ1 ⋄ A←+\NρPW,NρPW-I ⋄ X←T[A] ⋄ B[A]←I+2 ⋄ T[A]←⎕TCNL ⋄
T←B/T
T[(T=⎕TCNL)/⍳ρT]←,X,(N,I+1)ρ⎕TCNL,Iρ' ' ⋄ R←R,T,(0≠ρV)/⎕TCNL ⋄ →L1
L3:D←,+/^\(⍵WT↑T)∈C ⋄ →L5
L4:D←¯1+,(⍵WT↑T)⍳' ' ⋄ →L5
L5:D←|D-WT×D≠WT ⋄ R←R,(D↑T),⎕TCNL,Iρ' ' ⋄ V←(D↓T),V ⋄ D←,V⍳⎕TCNL ⋄ WT←PW-I ⋄ →L1
```

Explanations:

- The [character is a metacharacter having a particular meaning in Regular Expression so it you want to match a [in your text you have to escape the [with a \.
- Therefore \[matches a [and \] matches a]
- \[+ matches one or more [
- . matches any single character, so .* matches 0 or more characters
- **\[+.*\]+** would mean match one or more [followed by any number of characters followed by one or more]
- If you try this Regular Expression you'll see that you are matching too many characters: the reason is that .* is greedy and matches as many characters it can until the last] it finds in any given line.
- So, it is necessary to use the ? lazy operator to mean the opposite, i.e. .*?\] means match all characters until the next (not the **last** occurrence) occurrence of a] character.

Copyright © 2016 Eric Lescasse

Regular expressions can sometimes become tricky or quite complex, but most often, with a little habit and testing using the znetRegexTest application you can find the exact Regular Expression you need in your application.

How to use a Regular Expression

Once you've found the right Regular Expression to use in your application, just copy/paste it from the **znetRegexTest** application and use the **znetRegex** object using this Regular Expression.

How to get help with Regular Expressions

There are hundreds of Web sites teaching Regular Expressions, Regular Expressions Tutorials and sites allowing you to test Regular Expressions and analyze them on line.

There are even sites containing databases of already built Regular Expressions for the most common things you may have to do with Regular Expressions.

Here are a few sites to learn more about Regular Expressions:

http://regexr.com/
http://www.regexe.com/
https://regex101.com/
http://regexone.com/
http://www.regexlib.com
http://www.regular-expressions.info/examples.html

Copyright © 2016 Eric Lescasse

The znetRegex object

Introduction

Now that we have seen how to find the right Regular Expression to match some text in a document using the **znetRegexTest** object, we can use this Regular Expression in our code with the **znetRegex** object.

The **znetRegex** object is a good example of a non visual zObject.

We can use **znetRegex** to:

1. Validate user input
2. Find matches in a document
3. Replace matched text by other strings
4. Extract matched texts from a document
5. Etc.

Let's review a few of these possibilities, but first let's have a look at the znetRegex properties and methods.

The znetRegex API

The **znetRegex** object includes the following main properties and methods:

```
      ←'regex'□wi'*Create' 'znetRegex'
      'rr'□wi'mainprops'
Main Properties
---------------
*zCacheSize  *zMatchTimeout  *zOptions

Main Methods
------------
*zDoc            *zGetGroupNumbers      *zIsMatch  *zNextMatch  *zUnescape
*zEscape         *zGroupNameFromNumber  *zMatch    *zReplace
*zGetGroupNames  *zGroupNumberFromName  *zMatches  *zSplit

Main Events
-----------
```

You most often will want to use one of the following methods:

zIsMatch, zMatch, zMatches, zReplace, zSplit

Using **znetRegex** is really easy: the only potentially difficult part is to find the right Regular Expression to use.

Let's show simple examples.

Copyright © 2016 Eric Lescasse

Finding if there is at least a match

Use the **zIsMatch** method:

```
      'rr'⎕wi'*zIsMatch' 'The cat is out and the cats are gray' 'cats*'
1
```

The **cats*** Regular Expression match **cat** and **cats** because it means: match **cat followed by 0 or more s**

So we have at least one match and **zIsMatch** returns 1.

```
      'rr'⎕wi'*zIsMatch' 'The cat is out and the cats are gray' 'Cats*'
0
```

Here we don't have any match because Regular Expression are case sensitive by default.

However, you may use an additional argument specifying options. The documentation tells us:

```
      'rr'⎕wi'?IsMatch'
Syntax:  bool←'obj'⎕wi'*zIsMatch'searchString pattern {options {timeout}}
searchString:  the string to be searched
pattern:  the Regular Expression pattern to use
options:  (optional) a combination of the following RegexOptions:
          RegexOptions.None = 0
          RegexOptions.IgnoreCase = 1
          RegexOptions.Multiline = 2
          RegexOptions.ExplicitCapture = 4
          RegexOptions.Compiled = 8
          RegexOptions.Singleline = 16
          RegexOptions.IgnorePatternWhitespace = 32
          RegexOptions.RightToLeft = 64
          RegexOptions.ECMAScript = 256
          RegexOptions.CultureInvariant = 512
timeout:  (optional) the timeout in seconds
bool:     a boolean scalar indicating if there is at least one match
Example:
       'rr'⎕wi'*Create' 'znetRegex'
       'rr'⎕wi'*zIsMatch' 'The cat is out and the cats are gray' 'cat'
 1
       'rr'⎕wi'*zIsMatch' 'The cat is out and the cats are gray' 'Cat'
 0
       'rr'⎕wi'*zIsMatch' 'The cat is out and the cats are gray' 'Cat' 1
 1
```

So, if we use the **1** option (**RegexOptions.IgnoreCase**) we now get a match again:

```
      'rr'⎕wi'*zIsMatch' 'The cat is out and the cats are gray' 'Cats*' 1
1
```

All the other methods we review below also accept an additional **options** argument.

Copyright © 2016 Eric Lescasse

Finding the first match

Use the **zMatch** method to find the first match if any.

Example:

```
      'rr'⎕wi'*zMatch' 'The cat is out and the cats are gray' 'cats*'
1 4 3 cat
```

The result is a 4-element nested vector as follows:

[1]= success (1 or 0)
[2]= match index
[3]= match length
[4]= match value

So, we found the first occurrence of **cats*** at position **4** (in 0-origin); the matched string was **3** characters long and the matched value was **cat**.

If there is no match, the result is: **0 0 0''**

```
      'rr'⎕wi'*zMatch' 'The cat is out and the cats are gray' 'dogs*'
0 0 0
```

Finding the next match

Finding the next match is too easy: just use the **zNextMatch** method with no argument:

```
      'rr'⎕wi'*zMatch' 'The cat is out and the cats are gray' 'cats*'
1 4 3 cat
      'rr'⎕wi'*zNextMatch'
1 23 4 cats
      'rr'⎕wi'*zNextMatch'
0 0 0
```

The third result is **0 0 0''** because there were only 2 occurences of **cat** or **cats** in the phrase.

Finding all matches

Use the **zMatches** method to find all matches at once: the result is an **Nx4** nested matrix with each row representing one match (see **zMatch**).

```
      'rr'⎕wi'*zMatches' 'The cat is out and the cats are gray' 'cats*'
1  4 3 cat
1 23 4 cats
```

This is very powerful: both the **zNextMatch** and the **zMatches** method allow you to loop over the various matches and do whatever you want for each.

Copyright © 2016 Eric Lescasse

Replacing matches

Replacing matches by another string is also extremely easy with Regular Expressions: just use the **zReplace** method and add one argument being the replacement string.

Example:

```
      'rr'⎕wi'*zReplace' 'The cat is out and the cats are gray' 'cats*' 'dog'
The dog is out and the dog are gray
```

As you can see, we should better use the simple cat Regular Expression to replace **cat** by **dog**:

```
      'rr'⎕wi'*zReplace' 'The cat is out and the cats are gray' 'cat' 'dog'
The dog is out and the dogs are gray
```

Splitting the search string based on the Regular Expression pattern

The final method you may sometimes want to use is the zSplit method which allows you to split the search string in to a nested vector of strings.

For example, assume you wanted to get all the words in the following phrase as a nested vector:

```
      ]display 'rr'⎕wi'*zSplit' 'The cat is out, and you know: the cats are
gray!' '[\s:,!]'
.→14-------------------------------------------------------------.
|.→3-..→3-..→2..→3-..⊖..→3-..→3-..→4--..⊖..→3-..→4--..→3-..→4--..⊖.|
||The||cat||is||out|| ||and||you||know|| ||the||cats||are||gray|| ||
|'---''---''--''---''-''---''---''----''-''---''----''---''----''-'|
'∈-------------------------------------------------------------'
```

The **[\s:.!]** Regular Expression means: match any of the **space**, **colon**, **dot** or **exclamation point** characters.

Note that you get empty vectors where there were several consecutive separators. You would have to remove the empty elements with a little APL code:

```
      ]display ('rr'⎕wi'*zSplit' 'The cat is out, and you know: the cats are
gray!' '[\s:,!]' 32)~⊂''
.→11-----------------------------------------------------.
|.→3-..→3-..→2..→3-..→3-..→3-..→4--..→3-..→4--..→3-..→4--.|
||The||cat||is||out||and||you||know||the||cats||are||gray||
|'---''---''--''---''---''---''----''---''----''---''----'|
'∈-----------------------------------------------------'
```

In all these examples, the Regular Expressions were rather trivial.

The whole interest of Regular Expressions is that they allow you to find absolutely anything you want in a document, most often easily, but of course Regular Expressions patterns are sometimes more complex than in the above examples.

Copyright © 2016 Eric Lescasse

Validating User Input

A good example of the **znetRegex** use, is input validation.

The **zEdit** object has a built-in **regexvalidator** property which you can use to validate User Input in your forms. This **regexvalidator** property uses **znetRegex**.

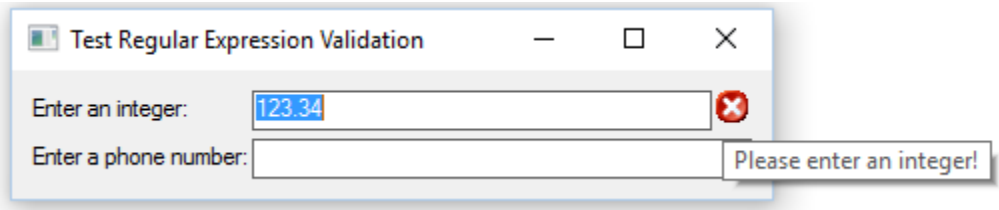
The **zzedit2** APL function is an example showing how to use **b**:

```
      ]zload zzedit2

    ∇ zzedit2
[1]   ⍝∇ Demonstrates how to use Regex Validators
[2]   ←'ff'⎕wi'*Create' 'zForm'('*size'300 400)
[3]   ←'ff'⎕wi'*caption' 'Test Regular Expression Validation'
[4]   ←'ff'⎕wi'*.ed1.Create' 'zEdit'('wherelc'0 120 0 250)('anchor' 'lrt')
[5]   ←'ff'⎕wi'.ed1.caption' 'Enter an integer:'
[6]   ←'ff'⎕wi'.ed1.regexvalidator' '^\d+$' 'Please enter an integer!'
[7]   ←'ff'⎕wi'*.ed2.Create' 'zEdit'('wherelc' '>' '=' '=' '=')('anchor' 'lrt')
[8]   ←'ff'⎕wi'.ed2.caption' 'Enter a phone number:'
[9]   ←'ff'⎕wi'.ed2.regexvalidator' '\(\d{3}\)\s\d{3}-\d{4}' 'Please enter a phone
            number in (xxx) xxx-xxxx format!'
[10]  ←'ff'⎕wi'AutoSize'
[11]  ←'ff'⎕wi'DemoShow' '∘' '∘'1 0
    ∇
```

You use **regexvalidator** by passing it a 2-element nested string vector (see lines 6 and 9):

[1] = the Regular Expression validator pattern to use
[2] = the error message to display if there is no match

Let's use it:

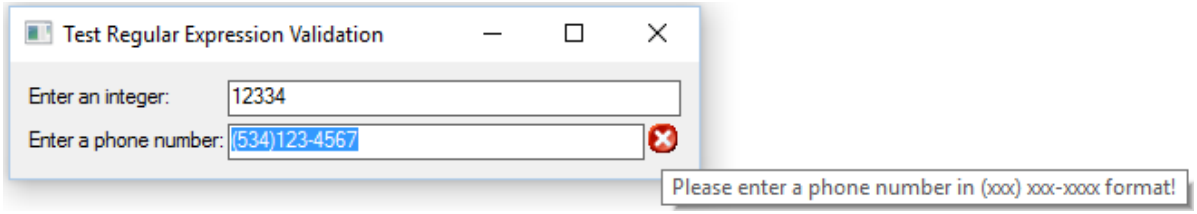


If the user enters anything that does not match the Regular Expression validator, a little blinking red icon gets displayed and if he hovers his mouse over it, the error message you defined in the **regexvalidator** is displayed!

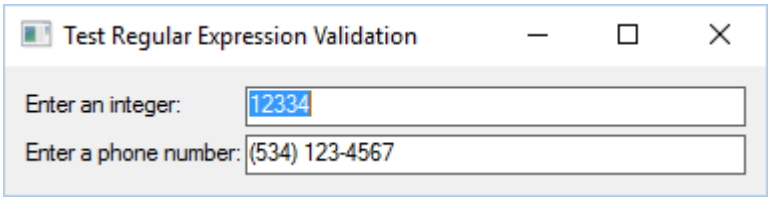
The user also can't exit the field until he enters valid input (unless he clicks the form **Cancel** button, if any, of course).

Let's remove the dot in **123.34,** tab to the next field and enter **(534)123-4567** in the phone field then try moving back to the first field:

Copyright © 2016 Eric Lescasse



Type a space after the closing paren and press Tab: this time the **regexvalidator** is happy and let us exit the field!



Finally, you may have noticed that the **zEdit** object knows how toautomatically reduce its width to leave room for the red icon to get displayed (otherwise it would have been at least partially displayed outside of our form).

Finding matches in a document

The **znetRegex** is handy to find all matches of a Regular Expression in a document, as we have seen.

We can exploit that to, for example, highlight all occurences of these matches in a **znetRichTextBox** control.

This is exactly the technique which is used in the **znetRegexTest** application to show and highlight all the Regular Expression matches.

The **zMatches** method result is perfect for this job as it returns the position and length of each match. A very simple loop[13] is then all what it takes to highlight the various matches.

Knowing that the **zMatches** result is stored in variable **m**, the loop is as simple as:

```
:for i :in⍳↑⍴m
      ←'ed2'⎕wi'*zSelectionStart'(m[i;2])
      ←'ed2'⎕wi'*zSelectionLength'(m[i;3])
      ←'ed2'⎕wi'*zSelectionBackColor'255 255 0
:endfor
```

[13] See the ed1_onChange event handler in znetRegexTest

Copyright © 2016 Eric Lescasse

Extracting matches from a document

Another useful use of **znetRegex** is to help extracting interesting data from a document.

Assume with would like to extract all tags contained in an HTML page.

Here is a simple HTML page (extracted from http://www.w3schools.com/html/default.asp)

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>This is a Heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Let's put this text in an APL variable called **html** and let's use **znetRegex** to extract all HTML tags.

```
      'rr'⎕wi'*zMatches' html '<(?![!/]?[ABIU][>\s])[^>]*>'
 1   0 15 <!DOCTYPE html>
 1  16  6 <html>
 1  23  6 <head>
 1  30  7 <title>
 1  47  8 </title>
 1  56  7 </head>
 1  64  6 <body>
 1  72  4 <h1>
 1  93  5 </h1>
 1  99  3 <p>
 1 122  4 </p>
 1 128  7 </body>
 1 136  7 </html>
```

Hence, the result will simply be:

```
      ('rr'⎕wi'*zMatches' html '<(?![!/]?[ABIU][>\s])[^>]*>')[;4]
 <!DOCTYPE html> <html> <head> <title> </title> </head> <body> <h1> </h1> <p> </p>
</body> </html>
```

Don't ask me how I found this Regular Expression, but if you ask, I simply went to www.regexlib.com, entered **HTML tags** in the search string and browsed the selected Regular Expressions the site returned until I found the one I wanted to use.

Easy enough for those who don't want to scratch their head or spend time finding the right regular expression.

Copyright © 2016 Eric Lescasse

However, I often try to find the right Regular Expression by myself, as I find it is fun and very rewarding when you have finally found it.

Copyright © 2016 Eric Lescasse

The znetRibbon object

Description

The **znetRibbon** object allows you to add an Office-like ribbon to your APL+Win zForm forms.

A ribbon may seem relatively simple when you look at it in an application like Word or Excel, but it is a pretty complex object indeed.

However, the **znetRibbon** object is an as simple as possible version of a ribbon, though it includes most of the features found in the official Microsoft Office ribbons.

The **znetRibbon** has some limitations, but should prove complete enough to be practically used in almost any APL application.

Th **znetRibbon** object is interesting because if is one of the are objects that can drastically change the look and feel of an existing APL application without requiring an overwhelming amount of work.

Basic Principles

Naming conventions

A ribbon object contains other objects in a hierarchy.

For example:

```
Ribbon
      RibbonTab
            RibbonPanel
                  RibbonGroupItem
                        RibbonButton
```

Each object has a name. To keep track of the hierarchy as well as to avoid name conflicts (you may want to have two buttons with the same name in two different parent objects), an object name is always prefixed by its parents name hierarchy, names being separated by an underscore (_) character.

Example: if the RibbonButton in the above hierarchy and its parents had the following names:

```
Rib
      Home
            Font
                  Styles
                        Bold
```

Copyright © 2016 Eric Lescasse

The full **Bold** button name would be: **Home_Font_Styles_Bold**[14]

Note that spaces, letters, numbers and other characters are allowed in names. The only forbidden character is the underscore character (_).

Thus an object can have the following name which is perfectly valid:

File_Save As_Find add-ins for other file formats

Object names are extremely important in **znetRibbon** as it is these names that you receive in APL in ⎕warg when the ribbon objects are clicked. It is these full names that let you know exactly what has been clicked in the ribbon.

In general, the full object name you want to create is the first argument of the various **znetRibbon** methods.

Shortcut Keys

The **znetRibbon** object does not support key tips (those little framed latters that show up in Office ribbons when you press the **alt** key).

But it does support shortcut keys.

You must define shortcut keys in the object full name, by embedding them in parenthesis.

Example:

```
←'rib'⎕wi'*zAddQATButton' 'Help (F1)' 'HelpBlue16np' 'Help (F1)' 'Get help!' ''
←'rib'⎕wi'*zAddButton' 'Home_Clipboard_>Cut (Ctrl+X)' 'Cut16np' '' '' ''
←'rib'⎕wi'*zAddButton' 'Home_Clipboard_>Copy (Ctrl+C)' 'Copy16np' '' '' ''
```

The shortcut keys will be stripped out of the object captions, so the button captions will be Help, Cut and Copy, but shortcut keys of F1n Ctrl+X and Ctrl+C will be created and associated with the corresponding objects.

When defining shortcut keys you can use the following key words: Ctrl, Alt, Shift, + and up-percase letters or numbers.

Object Captions

To simplify things and reduce the number of arguments to be passed to the ribbon meth-ods, the final part of an object name is used for its caption as well as for its name.

For example, a ribbon button with the following name:

[14] The Ribbon name itself (here **rib**) is never included in object names.

Copyright © 2016 Eric Lescasse

Home_Clipboard_Paste_Paste Special...

will appear in the Ribbon with the following caption: **Paste Special...**

and a menu item in the **File** menu with the following name:

File_Save As_Find add-ins for other file formats

will be displayed with the following caption: **Find add-ins for other file formats**

Images

All images you use for the Ribbon are contained in the LC.zObjects.dll. There are more than a thousand images available in both 16x16 and 32x32 pixels size.

You specify images by their names (example: **Cut16np**) exactly as they are stored in the LC.zObjects.dll.

To select the image you want to use for a ribbon object, use the **zznetselecticonform** function that has been introduced earlier in this document.

How to use the znetRibbon

In this section we will show how you can add a znetRibbon to an APL application.

Loading the znetRibbon and zznetribbon functions

Let's start by loading the **znetRibbon** and **zznetribbon** functions from the zObjects UCMD file (as well as the **zForm** and **zGrid** objects used by **zznetribbon**):

```
]zload znetRibbon zznetribbon zForm zGrid /r
```

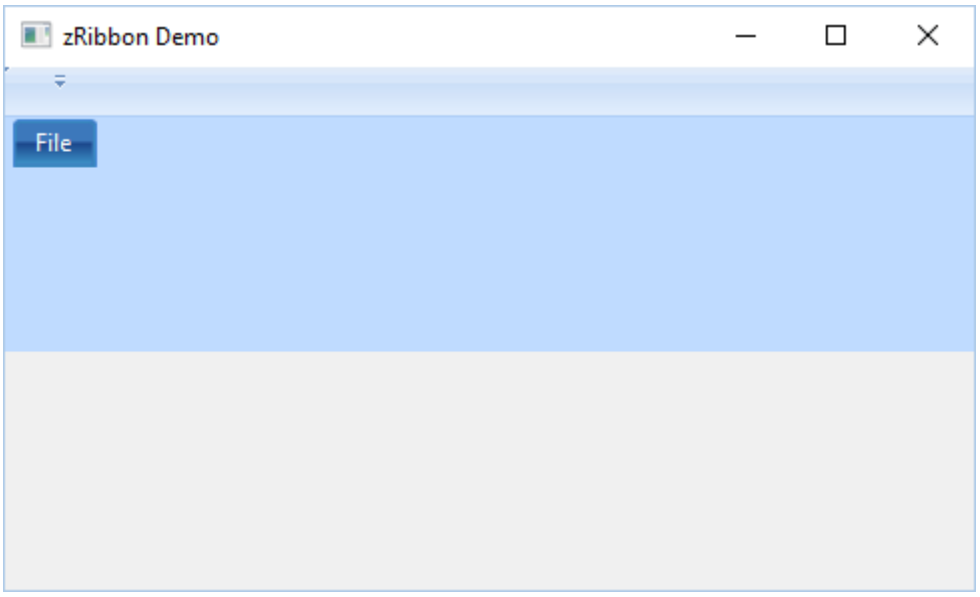
Adding a znetRibbon to a zForm

The first step is to a znetRibbon to a zForm:

```
←'ff'⎕wi'*Create' 'zForm'('*caption' 'zRibbon Demo')
←'ff'⎕wi'*.rib.Create' 'znetRibbon'('wherelc'0 0 142'>>')('anchor' 'ltr')
←'ff'⎕wi'*Show'
```

This displays the following form[15]:

[15] Note that the ribbon may seem very large, but the form has been reduced here due to limited space.

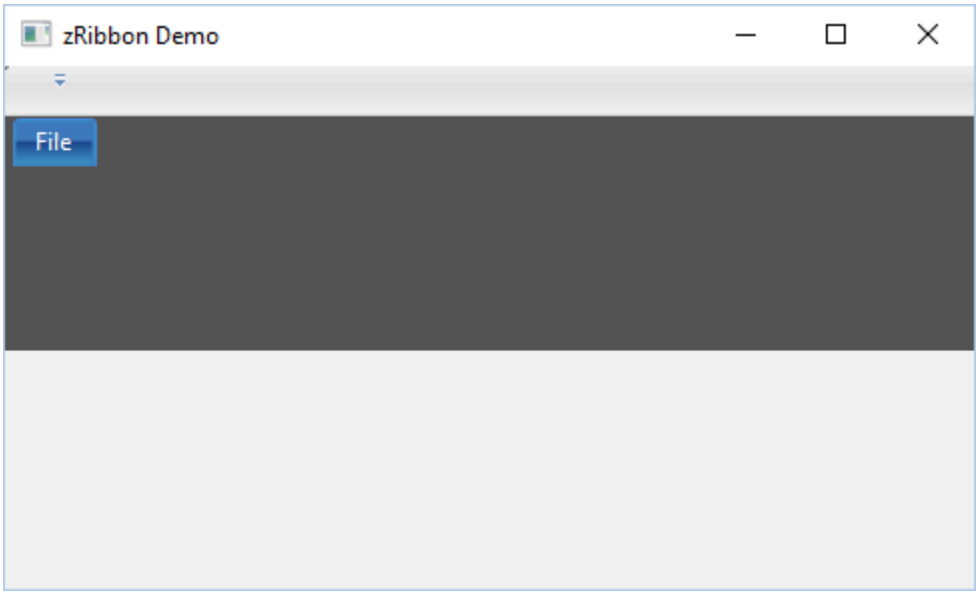


Changing the Theme

The znetRibbon object supports 2 themes called blue and black.

Let's change the theme to black:

```
←'rib'□wi'*zUseTheme' 'black'
```



In the following examples we will use the **blue** theme. To spare space in this document we will also only display the ribbon, not the entire form.

Adding a Quick Access Toolbar (QAT) with zAddQATButton

The Quick Access Toolbar or QAT is the little toolbar which is displayed at the top left, above the ribbon.

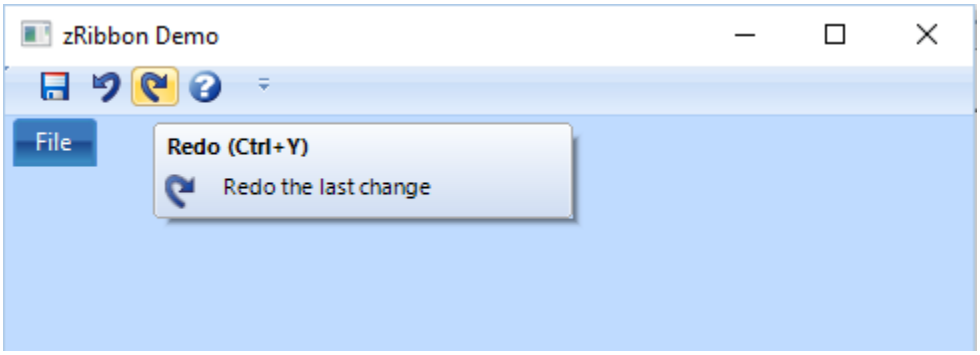
Copyright © 2016 Eric Lescasse

You add icon buttons to the QAT using the **zAddQATButton** method.
Its arguments are:

1. The button name (with an optional shortcut key within parenthesis)
2. The image name
3. The tooltip title
4. The tooltip text
5. The tooltip icon

Example:

```
←'rib'□wi'*zAddQATButton' 'Save (Ctrl+S)' 'SaveBlue16np' 'Save (Ctrl+S)'
          'Save the current document' ''
←'rib'□wi'*zAddQATButton' 'Undo (Ctrl+Z)' 'Undo16np' 'Undo (Ctrl+Z)'
          'Undo the last change' ''
←'rib'□wi'*zAddQATButton' 'Redo (Ctrl+Y)' 'Redo16np' 'Redo (Ctrl+Y)'
          'Redo the last change' ''
←'rib'□wi'*zAddQATButton' 'Help (F1)' 'HelpBlue16np' 'Help (F1)' 'Get help!'
          ''
```



The above ribbon shows the QAT and the Redo button tooltip when the mouse hovers over the Redo button.

Adding an Application Menu using zAddOrbMenuItem

When you create a **znetRibbon**, it displays with an empty application menu called File.

You can use the **zAddOrbMenu** method to change the Application Menu (or **Orb Menu** in this ribbon terminology) name. Most often you want to keep the name **File** which is quite standard for the Application Menu.

You then add menu items to the Orb Menu, using the **zAddOrbMenuItem** method which accepts 3 arguments:

1. The menu item full name
2. The image name

Copyright © 2016 Eric Lescasse

3. The menu item style (one of: '', 'split' or 'dropdown')

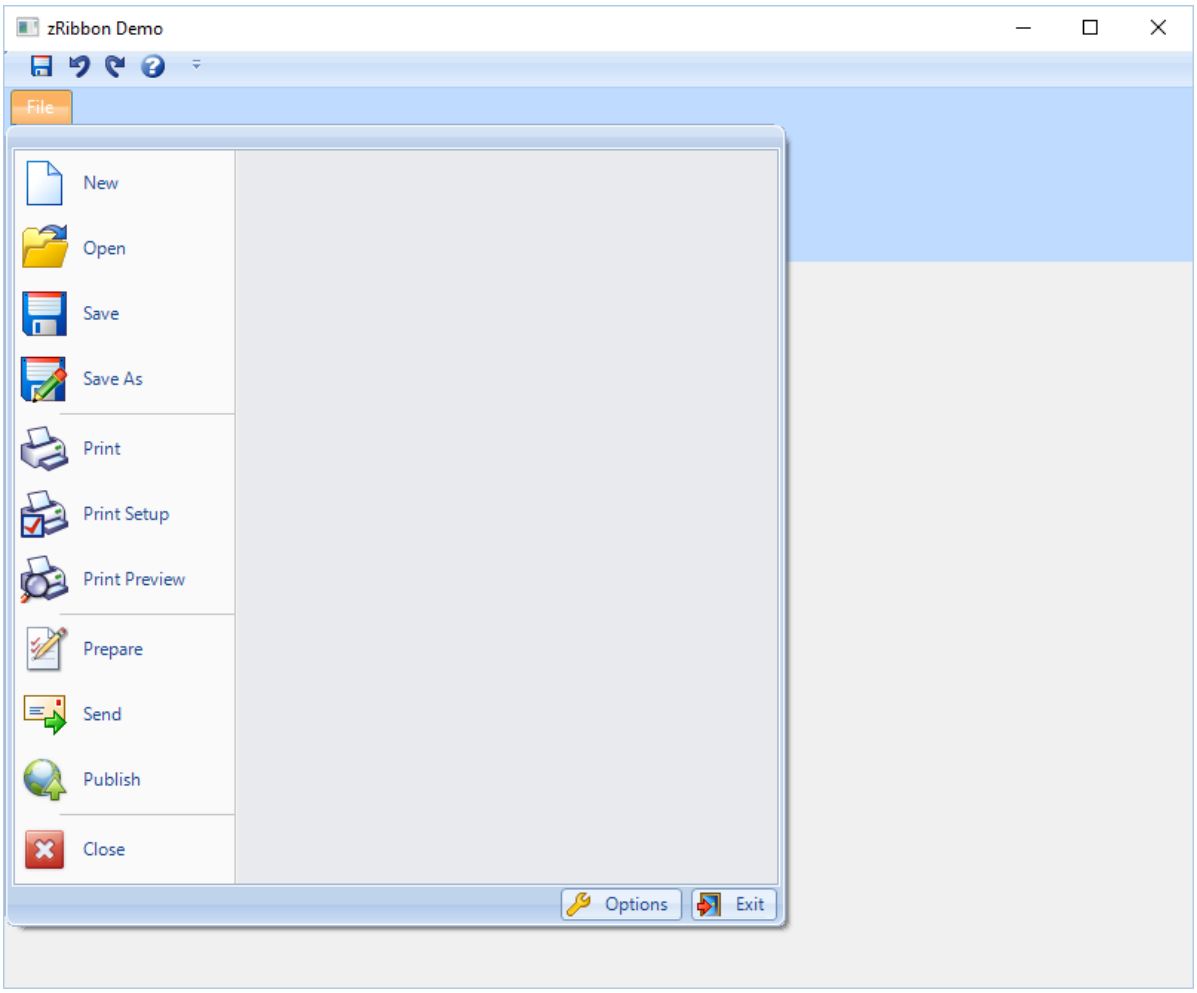
You can insert separator lines in the menu using the **zAddOrbMenuItemSeparator** method which requires a single argument: the separator object full name.

You can also add buttons at the bottom of the Orb Menu using the **zAddOrbOptionButton** method

Example:

```
←'rib'⎕wi'*zAddOrbMenu' 'File'
←'rib'⎕wi'*zAddOrbMenuItem' 'File_New' 'NewBlankDocument32np' ''
←'rib'⎕wi'*zAddOrbMenuItem' 'File_Open' 'OpenFileorFolder32np' ''
←'rib'⎕wi'*zAddOrbMenuItem' 'File_Save' 'SaveBlue32np' ''
←'rib'⎕wi'*zAddOrbMenuItem' 'File_Save As' 'SaveAs32np' ''
←'rib'⎕wi'*zAddOrbMenuItemSeparator' 'File_Sep1'
←'rib'⎕wi'*zAddOrbMenuItem' 'File_Print' 'Printer32np' ''
←'rib'⎕wi'*zAddOrbMenuItem' 'File_Print Setup' 'PrinterSetup32np' ''
←'rib'⎕wi'*zAddOrbMenuItem' 'File_Print Preview' 'PrintPreview32np' ''
←'rib'⎕wi'*zAddOrbMenuItemSeparator' 'File_Sep2'
←'rib'⎕wi'*zAddOrbMenuItem' 'File_Prepare' 'prepare32' ''
←'rib'⎕wi'*zAddOrbMenuItem' 'File_Send' 'SendMail32np' ''
←'rib'⎕wi'*zAddOrbMenuItem' 'File_Publish' 'Publish_32x32' ''
←'rib'⎕wi'*zAddOrbMenuItemSeparator' 'File_Sep3'
←'rib'⎕wi'*zAddOrbMenuItem' 'File_Close' 'Close_32x32' ''

←'rib'⎕wi'*zAddOrbOptionButton' 'File_Exit' 'ExitDoor216np'
←'rib'⎕wi'*zAddOrbOptionButton' 'File_Options' 'Options216np'
```

Copyright © 2016 Eric Lescasse



The panel which is displayed and includes the Orb Menu is called the BackstageView.

You often want to display most recently used files or elements in the BackstageView. You can do that using the **zAddOrbRecentItem** method which accepts the following arguments:

1. The object name (which ends with your most recently used elements)
2. The image name

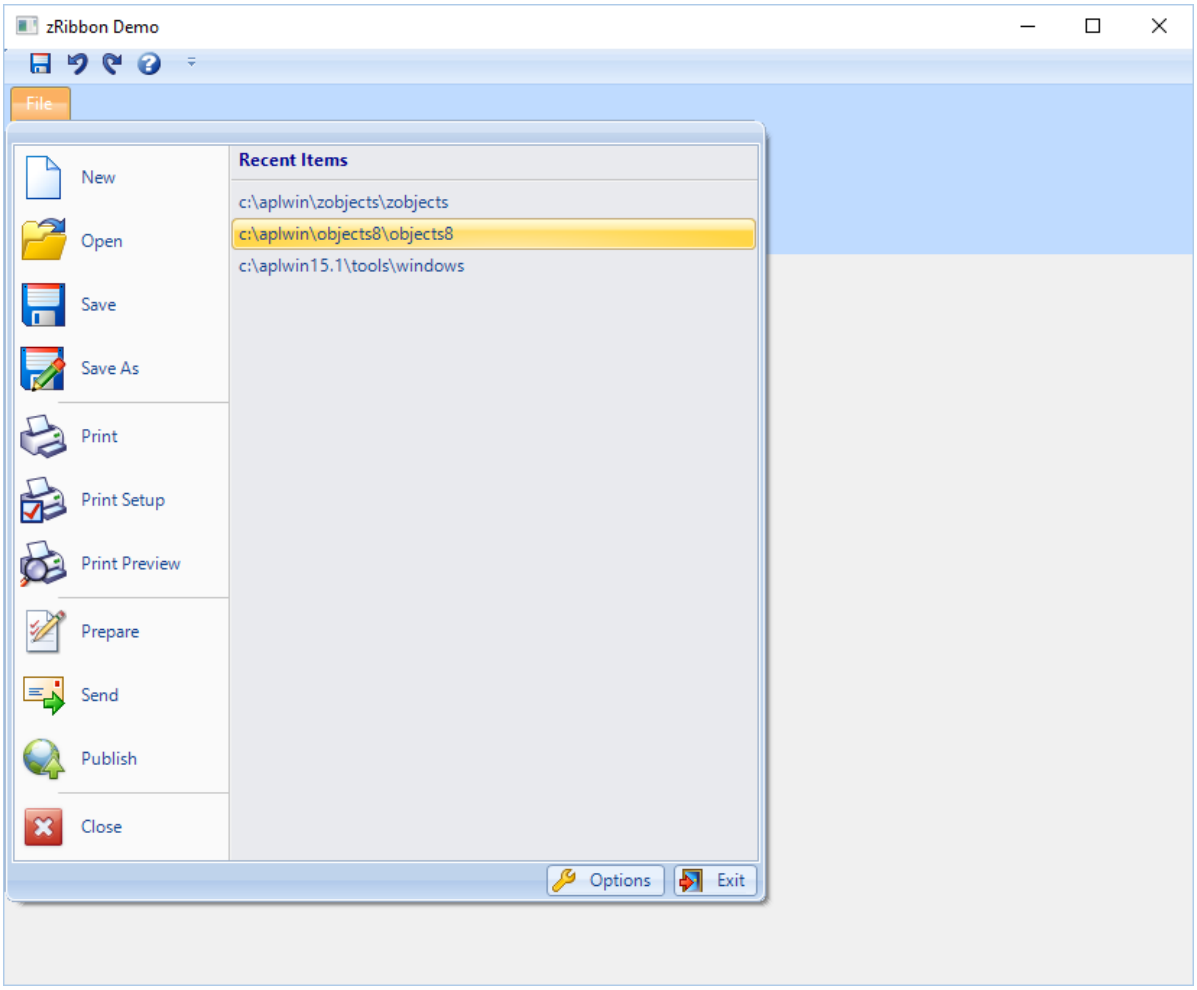
Example:

```
←'rib'□wi'*zRecentItemsCaption' 'Recent Items'
←'rib'□wi'*zAddOrbRecentItem' 'File_c:\aplwin\zobjects\zobjects'
        'RecentlyUse_32x32'
←'rib'□wi'*zAddOrbRecentItem' 'File_c:\aplwin\objects8\objects8'
        'RecentlyUse_32x32'
←'rib'□wi'*zAddOrbRecentItem' 'File_c:\aplwin15.1\tools\windows'
        'RecentlyUse_32x32'
```

Note how the button names in the most recently used files list end with the full file name itself (since this will be used as the button caption).

Copyright © 2016 Eric Lescasse

Also note that you often want to use the same icon for all these items.



Note that the Recent Items if you define any are displayed in the BackstageView regardless of the menu option that your mouse hovers over.

Adding Description Objects

To display specific menus in the **BackstageView** you must create Orb Menu Items with the **dropdown** style using the **zAddOrbMenuItem** method.

Let's replace the following line of code we used before:

```
←'rib'□wi'*zAddOrbMenuItem' 'File_Save As' 'SaveAs32np' ''
```

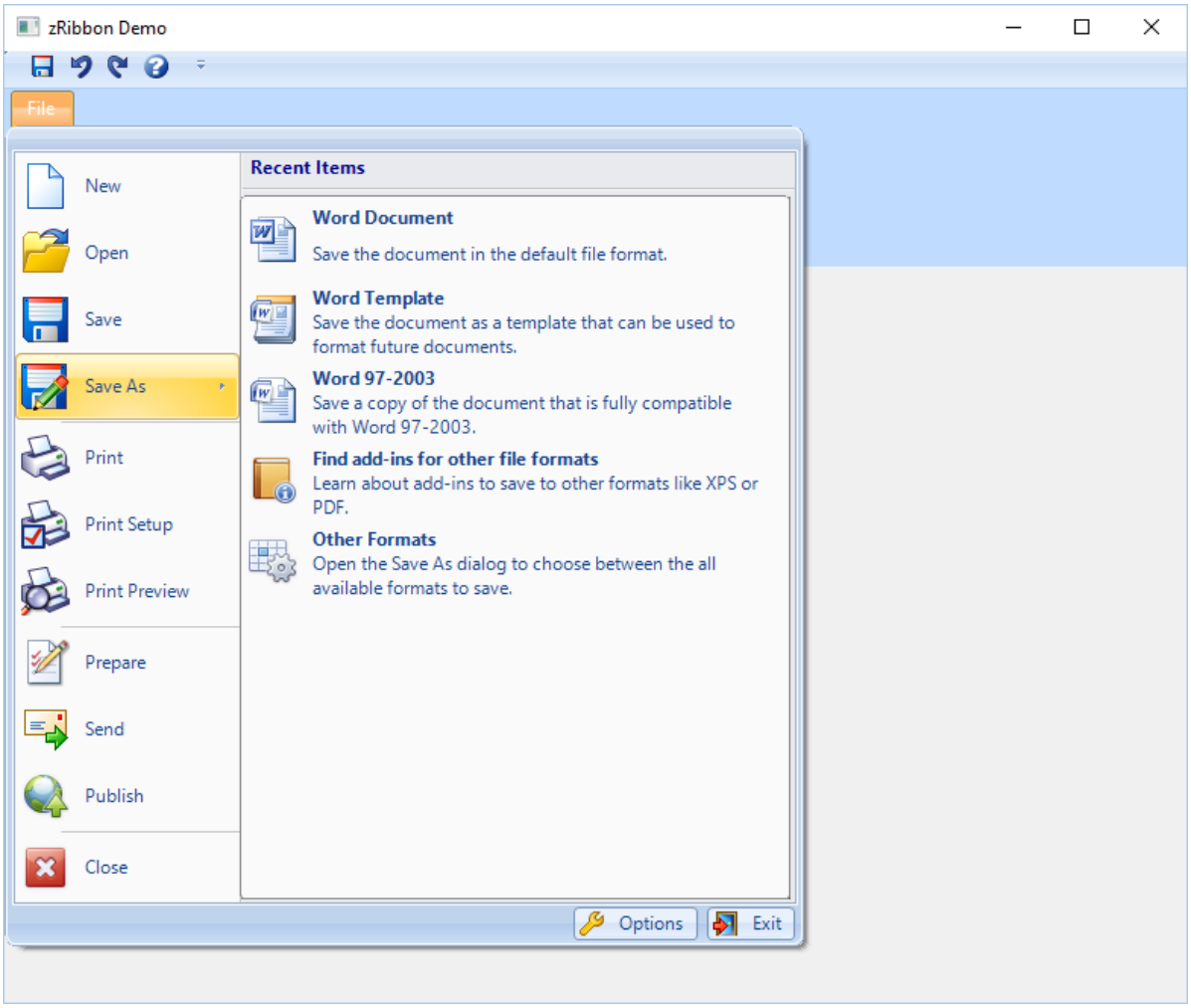
by the following set of lines:

```
←'rib'□wi'*zAddOrbMenuItem' 'File_Save As' 'SaveAs32np' 'dropdown'
```

Copyright © 2016 Eric Lescasse

```
←'rib'□wi'*zAddOrbMenuItemDescription' 'File_Save As_Word Document'
        'word2003doc32' 'Save the document in the default file format.'
←'rib'□wi'*zAddOrbMenuItemDescription' 'File_Save As_Word Template'
        'wordtemplate32' 'Save the document as a template that can be used to
        format future documents.'
←'rib'□wi'*zAddOrbMenuItemDescription' 'File_Save As_Word 97-2003'
        'worddocument32' 'Save a copy of the document that is fully
        compatible with Word 97-2003.'
←'rib'□wi'*zAddOrbMenuItemDescription' 'File_Save As_Find add-ins for other
        file formats' 'Information_32x32' 'Learn about add-ins to save to
        other formats like XPS or PDF.'
←'rib'□wi'*zAddOrbMenuItemDescription' 'File_Save As_Other Formats'
        'Format_32x32' 'Open the Save As dialog to choose between the all
        available formats to save.'
```

Now, when the mouse hovers over the **Save As** menu item, here is what we see:



Adding Ribbon Tabs with zAddTab

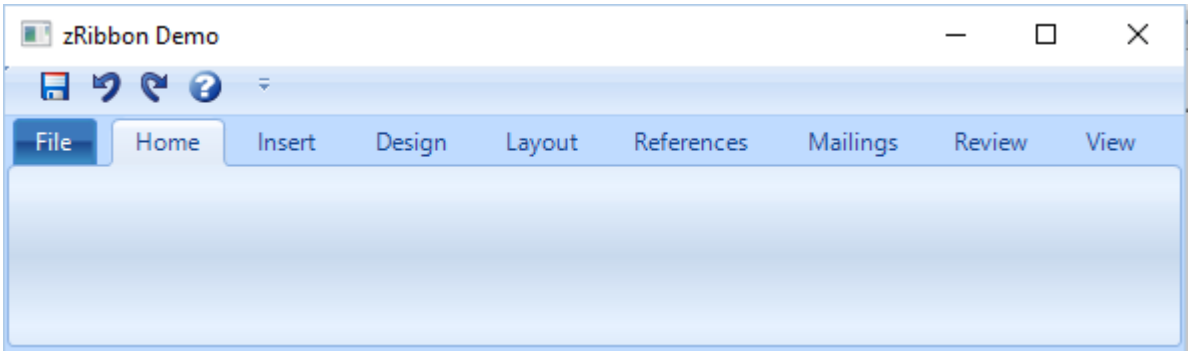
You use the zAddTab method to add tabs to the ribbon. Its single arguments is:

1. The Tab name (which also serves as the Tab caption)

Example:

```
←'rib'□wi'*zAddTab' 'Home'
←'rib'□wi'*zAddTab' 'Insert'
←'rib'□wi'*zAddTab' 'Design'
←'rib'□wi'*zAddTab' 'Layout'
←'rib'□wi'*zAddTab' 'References'
←'rib'□wi'*zAddTab' 'Mailings'
←'rib'□wi'*zAddTab' 'Review'
←'rib'□wi'*zAddTab' 'View'
```

The ribbon now reads:



Adding Panels to a Ribbon Tab with zAddPanel

Use the zAddPanel method to add a Panel to a Ribbon Tab. Its arguments are:

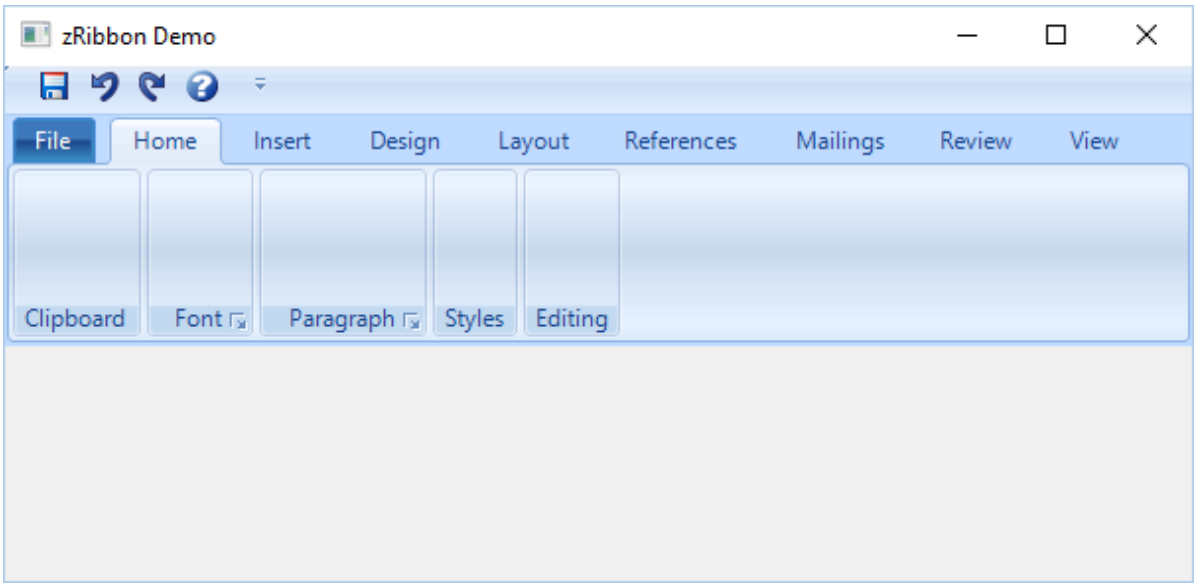
The **Panel** name (which also serves as its caption)
A boolean indicating if the panel Dialog button should be displayed or not

Example:

```
←'rib'□wi'*zAddPanel' 'Home_Clipboard'0
←'rib'□wi'*zAddPanel' 'Home_Font'1
←'rib'□wi'*zAddPanel' 'Home_Paragraph'1
←'rib'□wi'*zAddPanel' 'Home_Styles'0
←'rib'□wi'*zAddPanel' 'Home_Editing'0
```

The ribbon now reads:

Copyright © 2016 Eric Lescasse



Note the little buttons at the bottom right of the **Font** and **Paragraph** panels. There are called Dialog buttons as you generally click these buttons to display a more specific dialog box with more options.

Adding buttons to a Panel using zAddButton

Once you have created panels in your ribbon tabs, it's time to populate them with buttons.

For that you use the zAddButton method. Its arguments are:

1. The Button name (with its optional shortcut key with parenthesis)
   You may use an optional prefix:
   > means: display the button caption to the right of the button
   ~ means: do not display the button caption
2. The image name
3. The tooltip title
4. The tooltip text
5. The tooltip icon

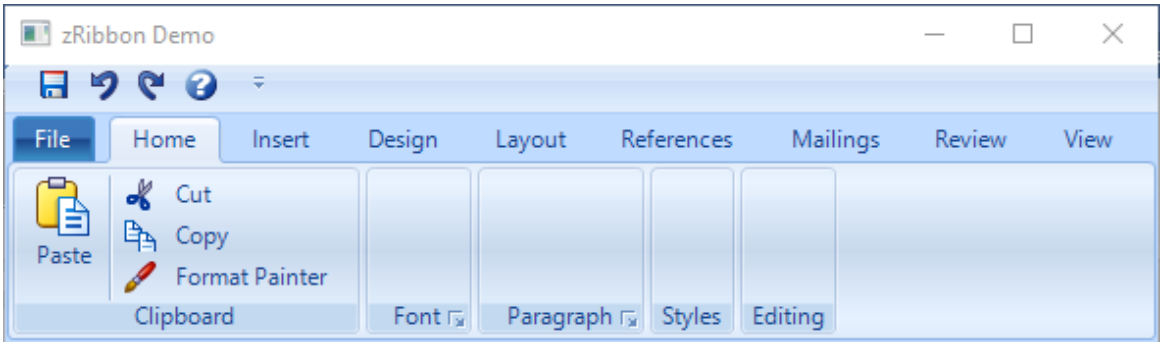
Example:

```
←'rib'□wi'*zAddButton' 'Home_Clipboard_Paste (Ctrl+V)' 'Paste32np'
        'Paste (Ctrl+V)' 'Paste the content of the clipboard into the current
        document' 'Paste32np'
←'rib'□wi'*zAddSeparator' 'Home_Clipboard_Sep1'
←'rib'□wi'*zAddButton' 'Home_Clipboard_>Cut (Ctrl+X)' 'Cut16np' '' '' ''
←'rib'□wi'*zAddButton' 'Home_Clipboard_>Copy (Ctrl+C)' 'Copy16np' '' '' ''
←'rib'□wi'*zAddButton' 'Home_Clipboard_>Format Painter' 'PaintbrushRed16np' ''
        '' ''
```

You can use the **zAddSeparator** to separate group of buttons by a vertical line.

Copyright © 2016 Eric Lescasse

The ribbon now reads:

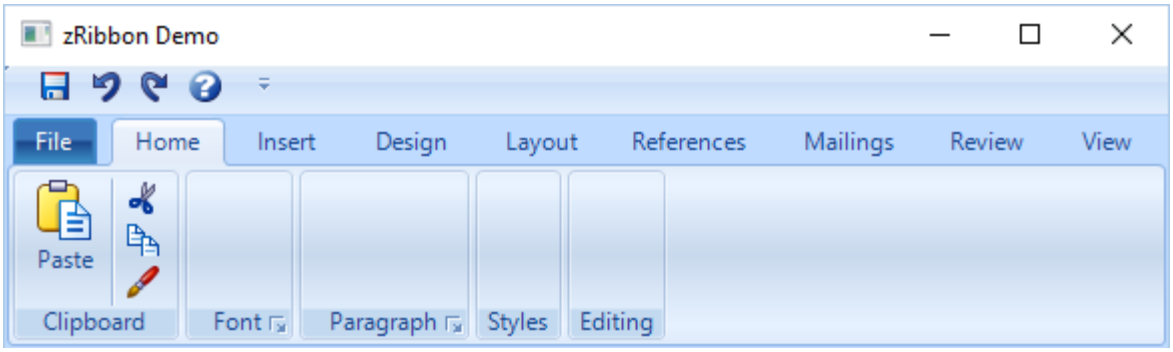


You may have noted the > prefix before some button names.

Without this prefix, the ribbon would display as follows:

(probably not really what you wanted)

With the ~prefix instead of the > prefix, the ribbon would read:



Adding Split Buttons to a Panel using zAddSplitButton

Just like in Word, we could have made the Paste button a split button using zAddSplitButton and zAddSplitButtonItem instead of zAddButton.

Copyright © 2016 Eric Lescasse

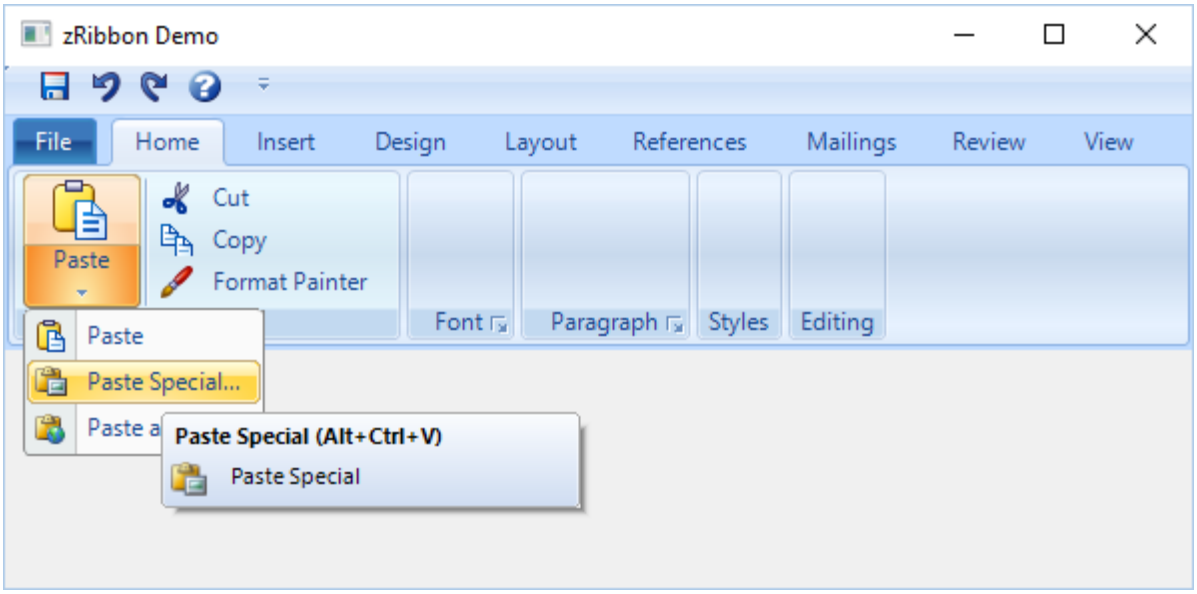
Just replace the following line of code we've used:

```
←'rib'□wi'*zAddButton' 'Home_Clipboard_Paste (Ctrl+V)' 'Paste32np'
        'Paste (Ctrl+V)' 'Paste the content of the clipboard into the current
        document' 'Paste32np'
```

by the following lines:

```
←'rib'□wi'*zAddSplitButton' 'Home_Clipboard_Paste (Ctrl+V)' 'Paste32np'
        'Paste (Ctrl+V)' 'Paste the content of the clipboard into the current
        document' 'Paste32np'
←'rib'□wi'*zAddSplitButtonItem' 'Home_Clipboard_Paste_Paste' 'Paste16np'
        'Paste' 'Paste the content of the clipboard into the current
        document' 'Paste16np'
←'rib'□wi'*zAddSplitButtonItem' 'Home_Clipboard_Paste_Paste Special...'
        'pastespecial16' 'Paste Special (Alt+Ctrl+V)' 'Paste Special'
        'Paste16np'
←'rib'□wi'*zAddSplitButtonItem' 'Home_Clipboard_Paste_Paste as Link'
        'pastelink16' 'Paste as Link' 'Paste the content of the clipboard a
        link in the current document' 'pastelink16'
```

The ribbon now looks like this:



Things start to look good!

Adding Groups of Buttons with zAddGroup and zAddGroupButton

Very often you need a lot of buttons in a Panel and to spare space in your ribbon, you want them grouped in a container where they are displayed joint to each other.

For that you must add a **Group** to the **Panel**, using the **zAddGroup** method.

Copyright © 2016 Eric Lescasse

The **zAddGroup** method accepts only one argument which is the full Group name.

Then you add **Buttons** to the **Group** with the **zAddGroupButton** method.

The **zAddGroupButton** accepts the following arguments:

1. The Button name (with its optional shortcut key with parenthesis)
2. The image name
3. The tooltip title
4. The tooltip text
5. The tooltip icon

Example:

```
←'rib'□wi'*zAddGroup' 'Home_Font_Styles'
←'rib'□wi'*zAddGroupButton' 'Home_Font_Styles_Bold (Ctrl+B)' 'Bold16np' '' ''
        ''
←'rib'□wi'*zAddGroupButton' 'Home_Font_Styles_Italic (Ctrl+I)' 'Italic16np' ''
        '' ''
←'rib'□wi'*zAddGroupButton' 'Home_Font_Styles_Underline (Ctrl+U)'
        'Underline16np' '' '' ''
←'rib'□wi'*zAddGroupButton' 'Home_Font_Styles_Strikeout' 'Strikeout16np' '' ''
        ''
←'rib'□wi'*zAddGroupLabel' 'Home_Font_Styles_~Sep1' '' '' '' ''
←'rib'□wi'*zAddGroupButton' 'Home_Font_Styles_FontLarger' 'FontLarger16np' ''
        '' ''
←'rib'□wi'*zAddGroupButton' 'Home_Font_Styles_FontSmaller' 'FontSmaller16np'
        '' '' ''
←'rib'□wi'*zAddGroup' 'Home_Font_Formatting'
←'rib'□wi'*zAddGroupButton' 'Home_Font_Formatting_FontCase' 'MatchCase16np' ''
        '' ''
←'rib'□wi'*zAddGroupButton' 'Home_Font_Formatting_ClearFormatting'
        'ClearFormatting_16x16' '' '' ''
←'rib'□wi'*zAddGroupLabel' 'Home_Font_Formatting_~Sep1' '' '' '' ''
←'rib'□wi'*zAddGroupColorChooser' 'Home_Font_Formatting_BackColor'
        'FillColorTemplate16np' '' '' ''
←'rib'□wi'*zAddGroupColorChooser' 'Home_Font_Formatting_ForeColor'
        'FontColorTemplate16np' '' '' ''
```

This displays the following groups of buttons in the **Font** panel of the ribbon:

Copyright © 2016 Eric Lescasse

The **zAddGroupColorChooser**[16] method adds a split button to a Group.

Also note that you can add a separator within a Group of buttons using the **zAddGroupLabel** method. For example, you could use:

```
←'rib'⎕wi'*zAddGroupLabel' 'Home_Font_Formatting_~Sep1' '' '' '' ''
```

or:

```
←'rib'⎕wi'*zAddGroupLabel' 'Home_Font_Formatting_ | ' '' '' '' ''
```

Note that in this latter case I am using ⎕av[125] (|) as the separator[17] and added a blank on each part of the separator to adjust the second Group with to be the same as th first Group width (see the next screen shot).

Note:
It is important to note that Groups are the only way to be sure to position buttons next to each other horizontally in a Panel.

Adding a Font Combo Box and a Font Size Combo Box

You sometimes want your users to be able to select a font and a font size using ribbon controls.

You can add a Font Combo Box to a Group using the **zAddGroupFontComboBox** method. Its arguments are:

1. The Combo Box name
2. The image name
3. The tooltip title
4. The tooltip text
5. The tooltip icon
6. The desired Combo Box width in pixels

You can add a Font Size Combo Box to a Group using the **zAddGroupFontSizeComboBox** method. Its arguments are:

1. The Combo Box name
2. The image name
3. The tooltip title

[16] In this version of LC.zObjects.dll, the color chooser buttons do not display any dropdown with colors to choose from. This will be added in a subsequent release.

[17] Do not use the APL modulo character as the separator character, as this is not an ASCII character.

Copyright © 2016 Eric Lescasse

4. The tooltip text
5. The tooltip icon
6. The desired Combo Box width in pixels

Example:

```
←'rib'□wi'*zAddGroup' 'Home_Font_NameSize'
←'rib'□wi'*zAddGroupFontComboBox' 'Home_Font_NameSize_FontNames' '' '' '' ''108
←'rib'□wi'*zAddGroupFontSizeComboBox' 'Home_Font_NameSize_FontSizes' '' '' '' ''40
```

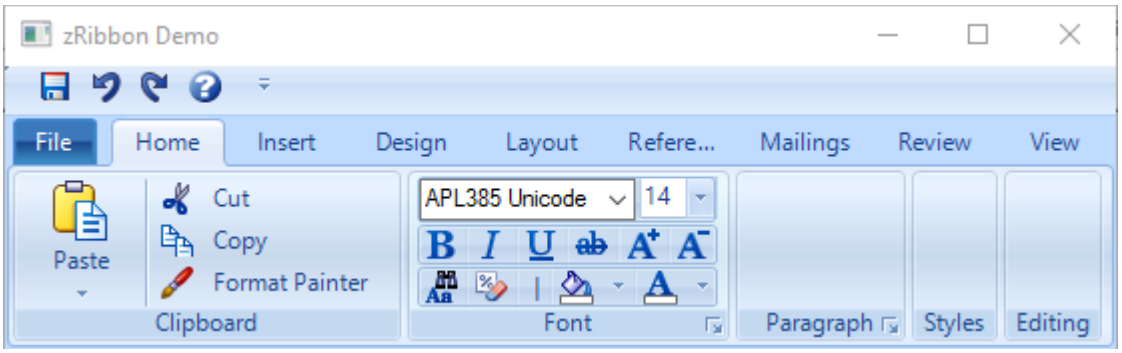
Note:
Almost everything in a **znetRibbon** is indeed a button. So, for example, the menu items in the File menu are buttons, the options items in the Paste split button are buttons, etc.

Similarly, the Font Size combo box is not a real Windows Combo Box and its items are buttons.

However, the Font Combo Box itself needs to contain several hundreds items since there generally are several hundreds fonts installed on a computer and for this reason displaying items as buttons would not be practical.

For this reason, the Font Combo box is a real Windows Combo Box[18].

The ribbon now reads:



Adding a Gallery using the zAddGallery method

Adding a gallery of items to the ribbon is pretty simple, you just need to select the images you want to use in the gallery, but be careful that all the images you choose must have the same zize.

You use the zAddGallery method which accepts the following arguments:

[18] In this version of the znetRibbon, there can be only one real Windows control per ribbon Panel.

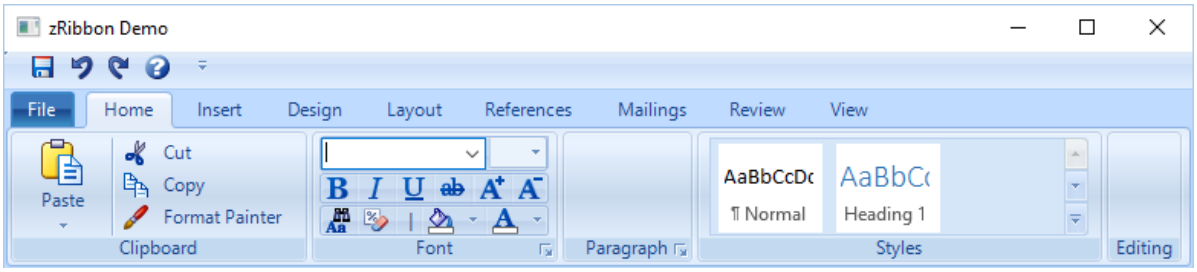
Copyright © 2016 Eric Lescasse

1. The Gallery full name
2. A nested vector of image names

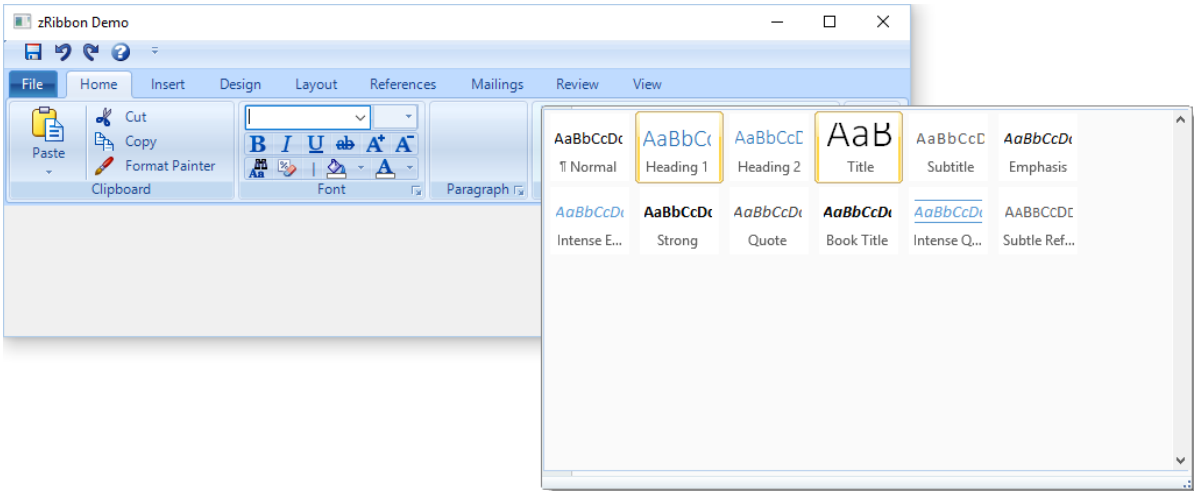
Example:

```
s←'StyleNormal' 'StyleHeading1' 'StyleHeading2' 'StyleTitle' 'StyleSubtitle'
s,←'StyleEmphasis' 'StyleIntenseEmphasis' 'StyleStrong' 'StyleQuote'
s,←'StyleBookTitle' 'StyleIntenseQuote' 'StyleSubtleReference'
←'rib'⎕wi'*zAddGallery' 'Home_Styles_Styles' s
```

The ribbon now looks like this:



If you click on the button, then the gallery is displayed in full with all buttons visible:



You can click on any button in the displayed gallery to select it.

Adding a DropDownButton to a Panel using zAddDropDownButton

We have already seen how we can add a Split Button (the Paste button) to a Panel. A DropDownButton is very similar to a SplitButton except that the button is not split into 2 parts.

Copyright © 2016 Eric Lescasse

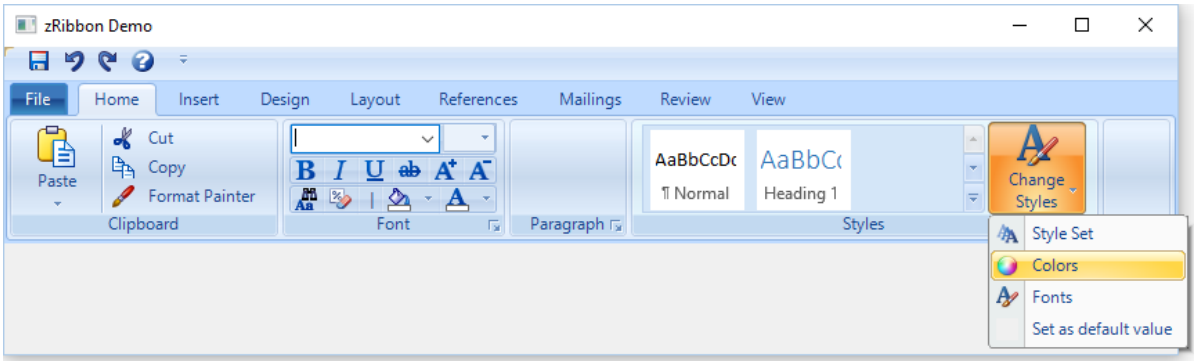
You use the **zAddDropDownButton** and **zAddDropDownButtonItem** methods to add a DropDownButton to a Panel: both methods accept the following arguments:

1. The Button or Button Item name
2. The image name
3. The tooltip title
4. The tooltip text
5. The tooltip icon

Example:

```
←'rib'□wi'*zAddDropDownButton' 'Home_Styles_Change Styles'
        'ChangeFontStyle_32x32' '' '' ''
←'rib'□wi'*zAddDropDownButtonItem' 'Home_Styles_Change Styles_Style Set'
        'styleset16' '' '' ''
←'rib'□wi'*zAddDropDownButtonItem' 'Home_Styles_Change Styles_Colors'
        'Colors_16x16' '' '' ''
←'rib'□wi'*zAddDropDownButtonItem' 'Home_Styles_Change Styles_Fonts'
        'ChangeFontStyle_16x16' '' '' ''
←'rib'□wi'*zAddDropDownButtonItem' 'Home_Styles_Change Styles_Set as default
        value' 'Transparent16' '' '' ''
```

The ribbon now looks like this:



Filling the Paragraph and Editing panels with controls

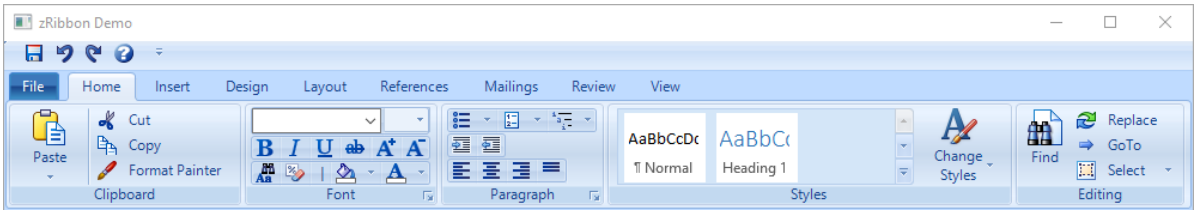
Just to make the ribbon Home Tab complete, let's use some methods we have already reviewed to populate the Paragraph and Editing panels.

```
←'rib'□wi'*zAddGroup' 'Home_Paragraph_Bullets'
←'rib'□wi'*zAddGroupSplitButton' 'Home_Paragraph_Bullets_Bullets'
        'Bullets16np' '' '' ''
←'rib'□wi'*zAddGroupSplitButton' 'Home_Paragraph_Bullets_Numbering'
        'NumberedList16np' '' '' ''
←'rib'□wi'*zAddGroupSplitButton' 'Home_Paragraph_Bullets_MultiList'
        'multilevellist16' '' '' ''
←'rib'□wi'*zAddGroup' 'Home_Paragraph_Indent'
←'rib'□wi'*zAddGroupButton' 'Home_Paragraph_Indent_Indent'
        'IndentIncrease_16x16' '' '' ''
←'rib'□wi'*zAddGroupButton' 'Home_Paragraph_Indent_Unindent'
```

Copyright © 2016 Eric Lescasse

```
          'IndentDecrease_16x16' '' '' ''
←'rib'⍞wi'*zAddGroup' 'Home_Paragraph_Justify'
←'rib'⍞wi'*zAddGroupButton' 'Home_Paragraph_Justify_Left' 'LeftAlignText16np'
          '' '' ''
←'rib'⍞wi'*zAddGroupButton' 'Home_Paragraph_Justify_Center' 'CenterText16np'
          '' '' ''
←'rib'⍞wi'*zAddGroupButton' 'Home_Paragraph_Justify_Right'
          'RightAlignText16np' '' '' ''
←'rib'⍞wi'*zAddGroupButton' 'Home_Paragraph_Justify_Justify' 'JustifyText16np'
          '' '' ''
←'rib'⍞wi'*zAddGroupLabel' 'Home_Paragraph_Justify_~Sep1' '' '' '' ''

←'rib'⍞wi'*zAddPanel' 'Home_Editing'0
←'rib'⍞wi'*zAddButton' 'Home_Editing_Find (Ctrl+F)' 'FindDocument32np' '' ''
          ''
←'rib'⍞wi'*zAddButton' 'Home_Editing_>Replace (Ctrl+H)' 'Replace16np' '' '' ''
←'rib'⍞wi'*zAddButton' 'Home_Editing_>GoTo (Ctrl+G)' 'goto16' '' '' ''
←'rib'⍞wi'*zAddSplitButton' 'Home_Editing_>Select' 'Selection16np' '' '' ''
←'rib'⍞wi'*zAddSplitButtonItem' 'Home_Editing_Select_Select Everything'
          'SelectAll2_16x16' '' '' ''
←'rib'⍞wi'*zAddSplitButtonItem' 'Home_Editing_Select_Text' 'select16' '' '' ''
←'rib'⍞wi'*zAddSplitButtonItem' 'Home_Editing_Select_Select text with similar
          format' 'Transparent16' '' '' ''
```

Our ribbon now looks like this:



Adding a real Windows Combo Box to a Panel with zAddHostedComboBox

It is possible to add one (and only one) Windows real Combo Box to a given ribbon Panel.

For this purpose, you use the **zAddHostedComboBox** method which arguments are:

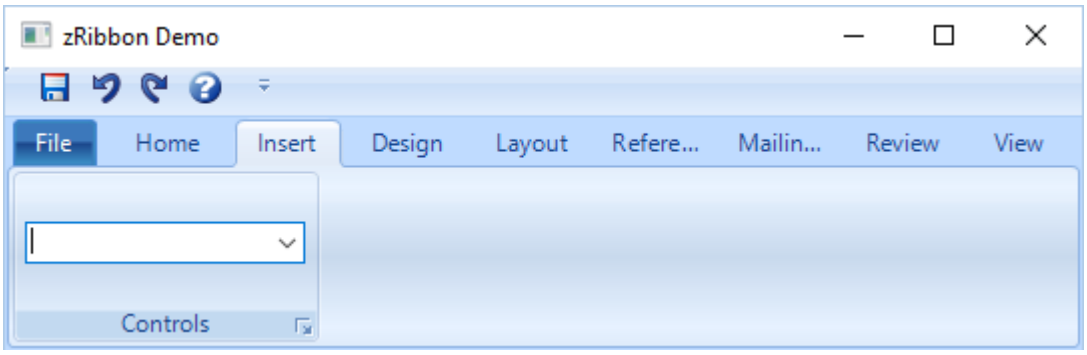
1. The Button or Button Item name
2. The image name
3. The tooltip title
4. The tooltip text
5. The tooltip icon
6. The desired Combo Box width
7. A nested vector of strings (items to be displayed in the Combo Box dropdown list)

Example:

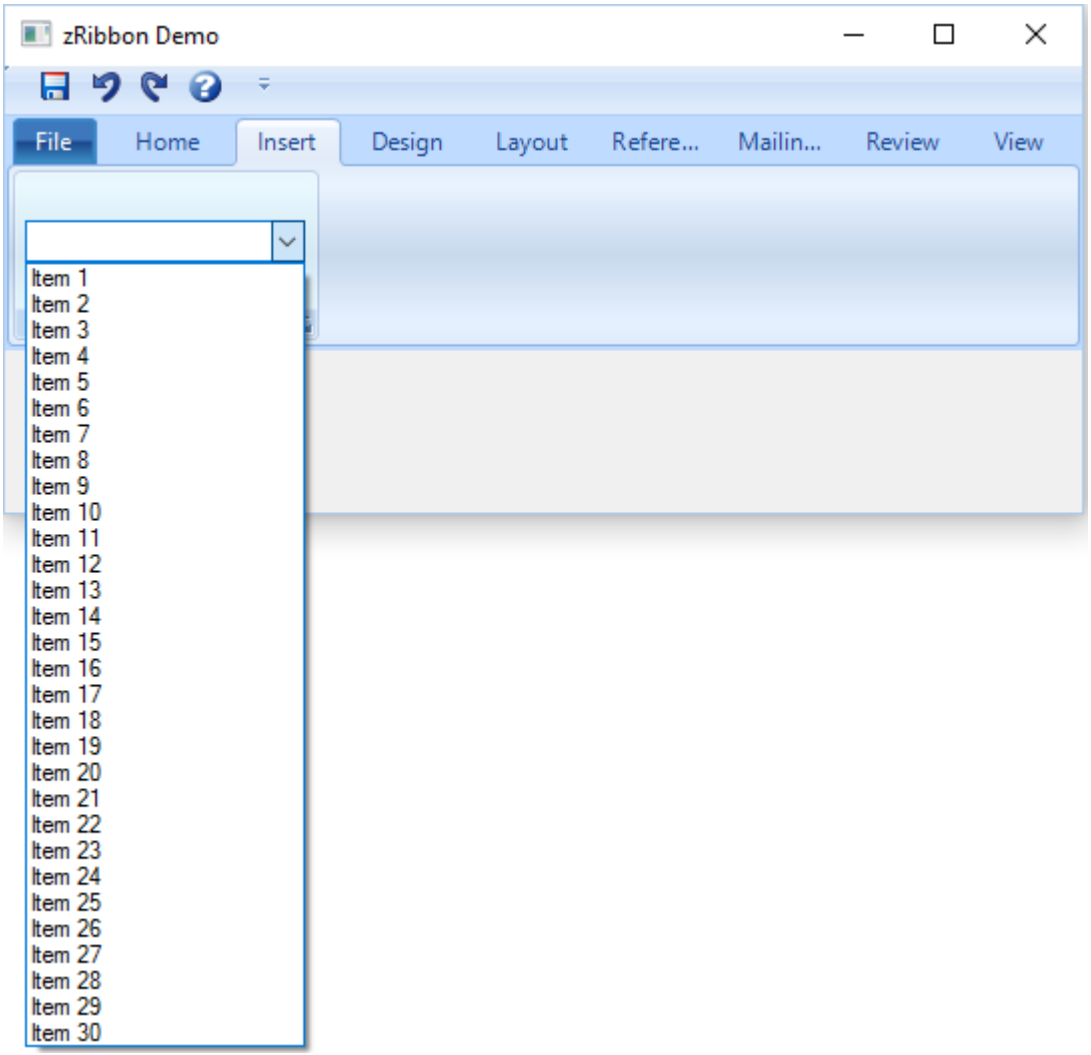
```
←'rib'⍞wi'*zAddHostedComboBox' 'Insert_Controls_MyCombo' '' '' ''
          ''140((⊂'Item '),¨⍳¨⍳30)
```

Copyright © 2016 Eric Lescasse

Here is how it looks:



and with the list being dropped down:



Copyright © 2016 Eric Lescasse

Adding Check Boxes to a Panel using the zAddCheckBox method

You can add one or more Check Boxes to a Panel, using the zAddCheckBox method.

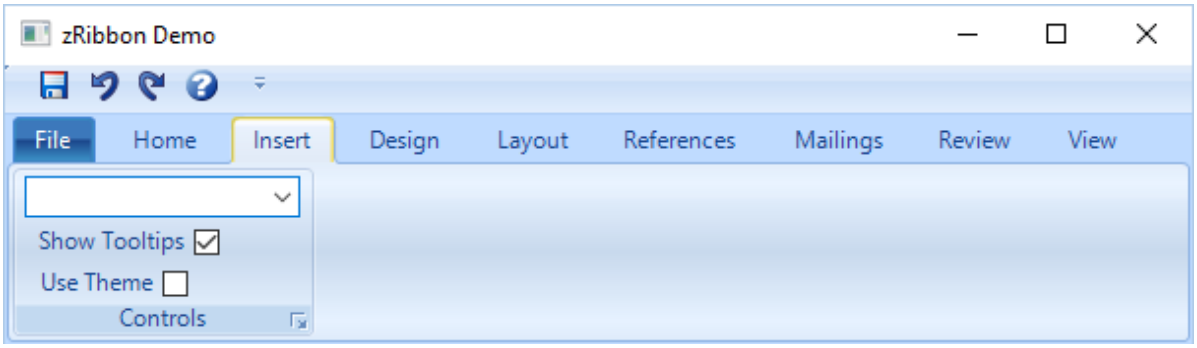
Its arguments are:

1. The Check Box name
2. The image name
3. The tooltip title
4. The tooltip text
5. The tooltip icon
6. A Boolean indicating if the check box should be initially checked (1) or not (0)
7. The check box alignment (one of 'right' or 'left') compared to its caption

Example:

```
←'rib'□wi'*zAddCheckBox' 'Insert_Controls_Show Tooltips' '' '' '' ''1'right'
←'rib'□wi'*zAddCheckBox' 'Insert_Controls_Use Theme' '' '' '' ''0'right'
```

The ribbon now reads:



Adding Radio Buttons to a Panel using the zAddRadioButton method

Adding a Radio Button to a Panel is very similar to adding a Check Box: you should use the **zAddRadioButton** method.

Its arguments are:

1. The Radio Button name
2. The image name
3. The tooltip title
4. The tooltip text
5. The tooltip icon
6. A Boolean indicating if the check box should be initially checked (1) or not (0)
7. The radio button alignment (one of 'right' or 'left') compared to its caption

Copyright © 2016 Eric Lescasse

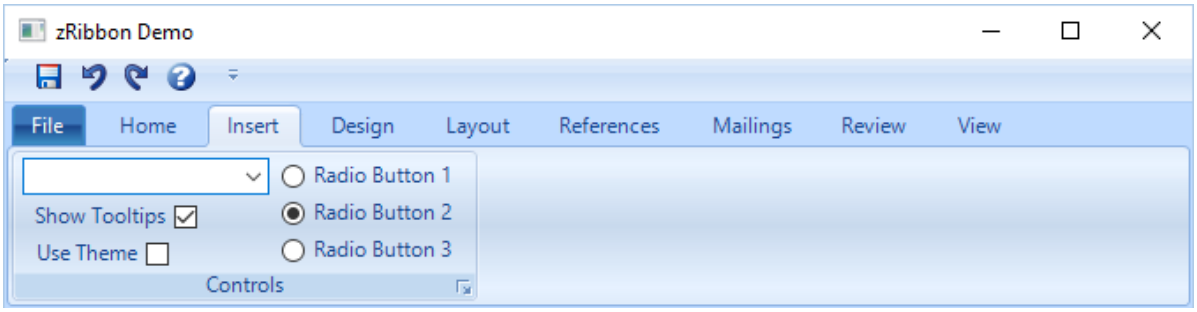
Note:
There may be only one group of radio buttons in any given Panel.

All radio buttons added to a given Panel are considered memvers of the same group of radio buttons, i.e. checking one of them automatically unchecks all other radio buttons in the Panel.

Example:

```
←'rib'⎕wi'*zAddRadioButton' 'Insert_Controls_Radio Button 1' '' '' '' ''0'left'
←'rib'⎕wi'*zAddRadioButton' 'Insert_Controls_Radio Button 2' '' '' '' ''1'left'
←'rib'⎕wi'*zAddRadioButton' 'Insert_Controls_Radio Button 3' '' '' '' ''0'left'
```

The ribbon now looks like this:



That concludes the ribbon layout possibilities in this version of **znetRibbon**.

Handling Events in the znetRibbon

The onXRibbonElementClick event

A beautiful .Net ribbon would be of no value if you could not receive and handle events occurring in the ribbon.

Conforming to the overall idea of simplifying things as mush as possible for the developer, the **znetRibbon** object mainly includes a single event called **RibbonElementClick**.

There's also a RibbonResized event that we will study later on.

You handle it in APL by subscribing to the **onXRibbonElementClick** event.

When the onXRibbonElementClick event fires you receive information about the ribbon element that has been clicked in the ⎕warg system variable.

Copyright © 2016 Eric Lescasse

⎕warg is a nested vector of strings.

⎕warg[1] is always the ribbon element full name
⎕warg[2] (if it exists) may be additional information about the event that occurred

Here is how you could handle the onXRibbonElementClick event in a small **zznetribbon**[19] APL function:

```
    ∇ zznetribbon a;c;s
[1]   ⍝∇ This function demonstrates how to use the zRibbon object
[2]
[3]   :select a
[4]   :case''
[5]       ←'ff'⎕wi'*Create' 'zForm'('*caption' 'zRibbon Demo')⍝('*size'300 500)
[6]       ←'ff'⎕wi'*.rib.Create' 'znetRibbon'('wherelc'0 0 142'>>')('anchor'
'ltr')
[7]
[8]       ←'rib'⎕wi'*zUseTheme' 'blue'
[9]
[10]      ⍝ Build the ribbon UI below
[11]    …
[128]
[129]     ←'rib'⎕wi'AddHandler' '*onXRibbonElementClick' 'zznetribbon"onXRib-
bonElementClick"'0
[130]
[131]     ←'ff'⎕wi('DemoShow'.5 .5 1 1)
[132] :case'onXRibbonElementClick'
[133]     ←'rib'⎕wi'*zRefresh'
[134]     ⎕←c←t⎕warg
[135]     :selectc
[136]     :case'Home_Font'
[137]         :if'ButtonMore'≡2⊃c ⋄ ⎕←⎕ucmd']display c' ⋄ :endif
[138]     :case'Home_Paragraph'
[139]         :if'ButtonMore'≡2⊃c ⋄ ⎕←⎕ucmd']display c' ⋄ :endif
[140]     :caselist'Save' 'File_Save'                    ⋄ ⎕←⎕ucmd']display c'
[141]     :case'Undo'                                    ⋄ ⎕←⎕ucmd']display c'
[142]     :case'Redo'                                    ⋄ ⎕←⎕ucmd']display c'
[143]     :case'Help'                                    ⋄ ⎕←⎕ucmd']display c'
[144]     :case'File_New'                                ⋄ ⎕←⎕ucmd']display c'
[145]     :case'File_Open'                               ⋄ ⎕←⎕ucmd']display c'
[146]     :case'File_Save As'                            ⋄ ⎕←⎕ucmd']display c'
[147]     :case'File_Print'                              ⋄ ⎕←⎕ucmd']display c'
[148]     :case'File_Print Setup'                        ⋄ ⎕←⎕ucmd']display c'
[149]     :case'File_Print Preview'                      ⋄ ⎕←⎕ucmd']display c'
[150]     :case'File_Prepare'                            ⋄ ⎕←⎕ucmd']display c'
[151]     :case'File_Send'                               ⋄ ⎕←⎕ucmd']display c'
[152]     :case'File_Publish'                            ⋄ ⎕←⎕ucmd']display c'
[153]     :case'File_Close'                              ⋄ ⎕←⎕ucmd']display c'
[154]     :case'File_Exit'                               ⋄ ⎕←⎕ucmd']display c'
[155]     :case'File_RecentItem'                         ⋄ ⎕←⎕ucmd']display c'
[156]     :case'Home_Clipboard_Paste'                    ⋄ ⎕←⎕ucmd']display c'
[157]     :case'Home_Clipboard_Cut'                      ⋄ ⎕←⎕ucmd']display c'
[158]     :case'Home_Clipboard_Copy'                     ⋄ ⎕←⎕ucmd']display c'
```

[19] The zznetribbon function itself is not a zObject, just a simple function to use in the workspace to test the znetRibbon zObject.

Copyright © 2016 Eric Lescasse

```
[159]     :case'Home_Clipboard_Format Painter'          ⋄ ⎕←⎕ucmd']display c'
[160]     :case'Home_Font_NameSize_FontCase'            ⋄ ⎕←⎕ucmd']display c'
[161]     :case'Home_Font_NameSize_ClearFormatting'     ⋄ ⎕←⎕ucmd']display c'
[162]     :case'Home_Font_Styles_Bold'                  ⋄ ⎕←⎕ucmd']display c'
[163]     :case'Home_Font_Styles_Italic'                ⋄ ⎕←⎕ucmd']display c'
[164]     :case'Home_Font_Styles_Underline'             ⋄ ⎕←⎕ucmd']display c'
[165]     :case'Home_Font_Styles_Strikeout'             ⋄ ⎕←⎕ucmd']display c'
[166]     :case'Home_Font_Styles_FontLarger'            ⋄ ⎕←⎕ucmd']display c'
[167]     :case'Home_Font_Styles_FontSmaller'           ⋄ ⎕←⎕ucmd']display c'
[168]     :case'Home_Font_Styles_BackColor'             ⋄ ⎕←⎕ucmd']display c'
[169]     :case'Home_Font_Styles_ForeColor'             ⋄ ⎕←⎕ucmd']display c'
[170]     ⍝ Etc.
[171]     :endselect
[172] :endselect
    ∇
```

This function demonstrates how to:

1. create a **Ribbon** object
2. subscribe to the **onXRibbonElementClick** event
3. how to handle the **onXRibbonElementClick** event

> **Note:**
> Note that it is important to call the zRefresh ribbon method in your onXRibbonElementClick event handler as, in some cases, the znetRibbon would not get repainted correctly.

The above sample function just simply displays in the APL Session the information APL received in ⎕warg.

In real life, you would instead call another APL function to handle each click separately, or better, if you want to follow our good object oriented encapsulation rules, you would call a method inside your object instead.

So, in the above function we could replace:

```
:case'File_Open'                                ⋄ ⎕←⎕ucmd']display c'
```

by:

```
:case'File_Open'                                ⋄ ←⎕wi'File_Open'
```

and have a File_Open method defined elsewhere in your object:

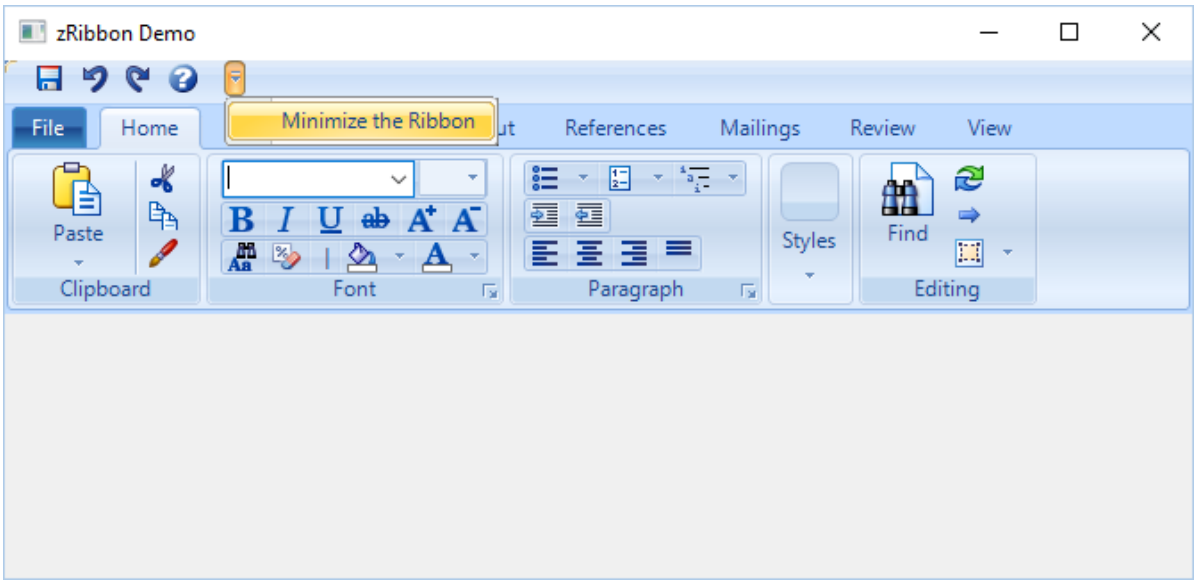
```
    :region----------------------------------------File_Open
    ∆FileOpen:
        ⍝∇ Handles a click on the Open option of the File menu
        ...
        ⍝ your code here to perform a File Open
        ...
        :return
```

```
:endregion
```

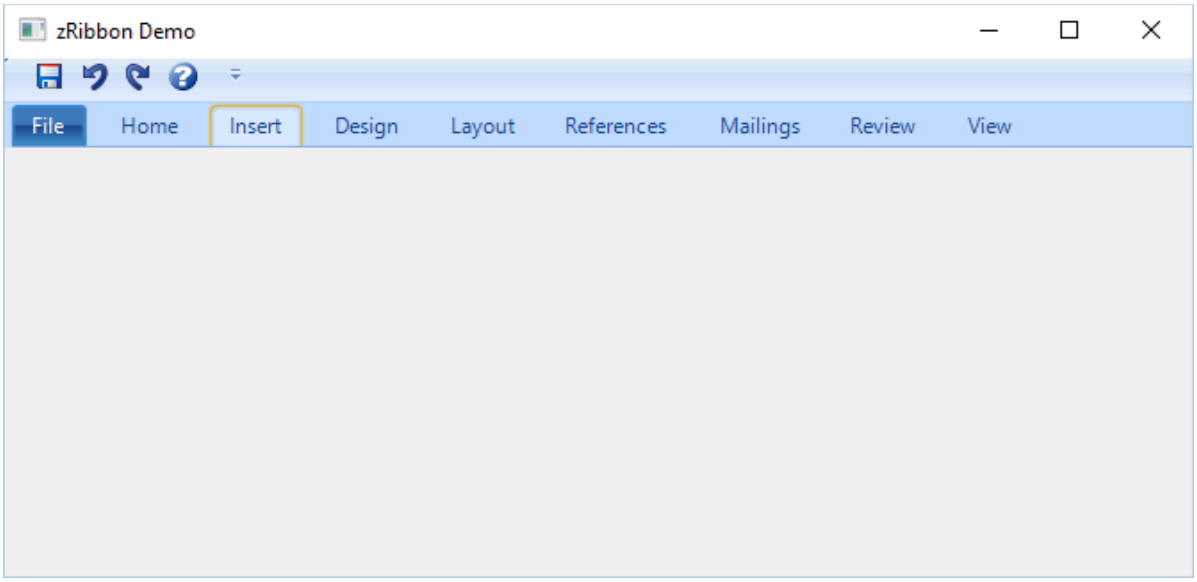
It is as simple as that to handle events in the **znetRibbon** object.

The onXRibbonResized event

You may have noticed that the Quick Access Toolbar includes a menu which you can display by clicking the little  button at the right of the QAT.



If you select the **Minimize the Ribbon** option, then the Ribbon gets minimized and the form now looks like this:



This is interesting as it frees quite some space for your application UI.

Copyright © 2016 Eric Lescasse

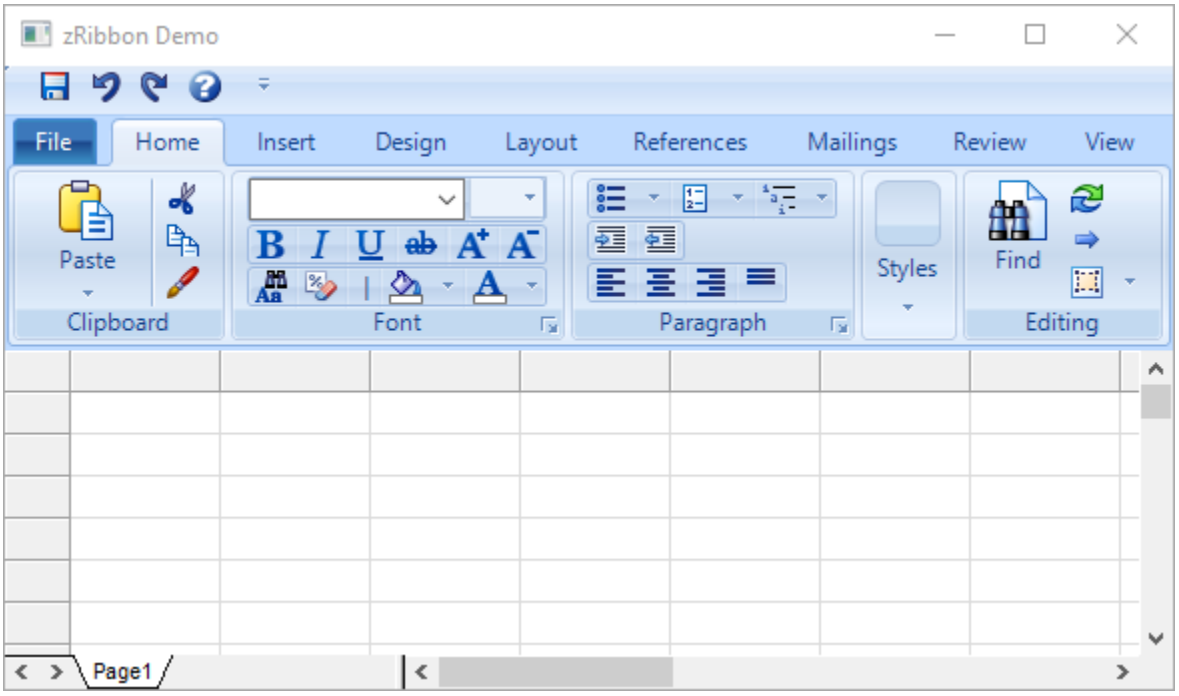
But you need to be able to react to the **Minimize the Ribbo**n action in order for example to move your UI upward and increase its height.

The following function shows the same form with an **APL+Win Grid** below the ribbon and shows how you can handle the **onXRibbonResized** event to change the Grid position and height when the ribbon is minimized or maximized.

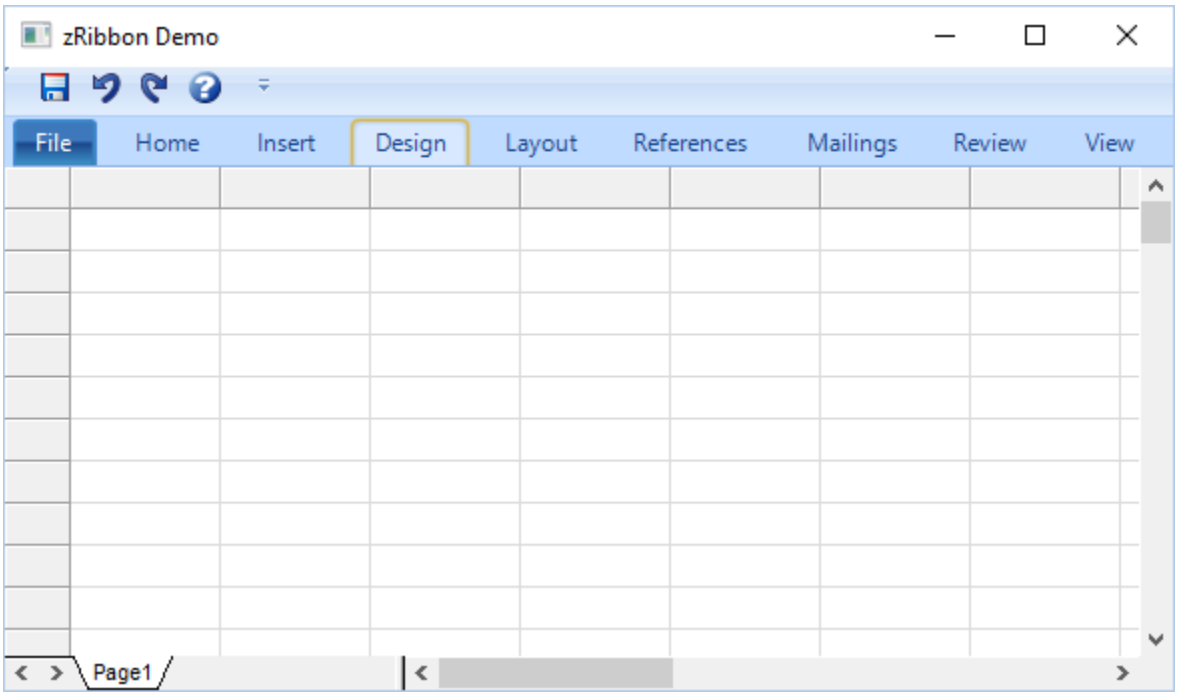
```
    ∇ zznetribbon a;c;s;h;w
[1]   ⍝∇ This function demonstrates how to use the zRibbon object
[2]
[3]   :select a
[4]   :case''
[5]       ←'ff'⎕wi'*Create' 'zForm'('*caption' 'zRibbon Demo')
[6]       ←'ff'⎕wi'*.rib.Create' 'znetRibbon'('wherelc'0 0 142'>>')('anchor' 'ltr')
[7]       ←'ff'⎕wi'*.grid.Create' 'zGrid'('wherelc' '>>'0'>>' '>>')('anchor' 'ltrb')
[8]       →'ff'⎕wi'*.grid.Set'('*xRows'100)('*xCols'20)('*xHeadRows'1)
[9]
[10]      ←'rib'⎕wi'*zUseTheme' 'blue'
[11]
[12]      ⍝ Cretae the ribbon UI below
[13]
…
[130]
[131]     ←'rib'⎕wi'AddHandler' '*onXRibbonElementClick'
                  'zznetribbon"onXRibbonElementClick"'0
[132]     ←'rib'⎕wi'AddHandler' '*onXRibbonResized' 'zznetribbon"onXRibbonResized"'
[133]
[134]     ←'ff'⎕wi('DemoShow'400 600 1 1)
[135] :case'onXRibbonElementClick'
[136]     ⍝ Handle the RibbonElementClick events below
[137]     …
…
[138]
[175] :case'onXRibbonResized'
[176]     (h w)←⎕warg
[177]     ←'grid'⎕wi'wherelc'h 0'>>' '>>'
[178] :endselect
    ∇
```

Using this function, the form looks like this when the ribbon is maximized:

Copyright © 2016 Eric Lescasse



And it looks like this when the ribbon is minimized:



The Grid has automatically been moved up and its height increased so that it still uses the whole form remaining client area.

If your form contains many controls (not just one control like the above form), it is advisable to host all these controls in a container control that can for example be:

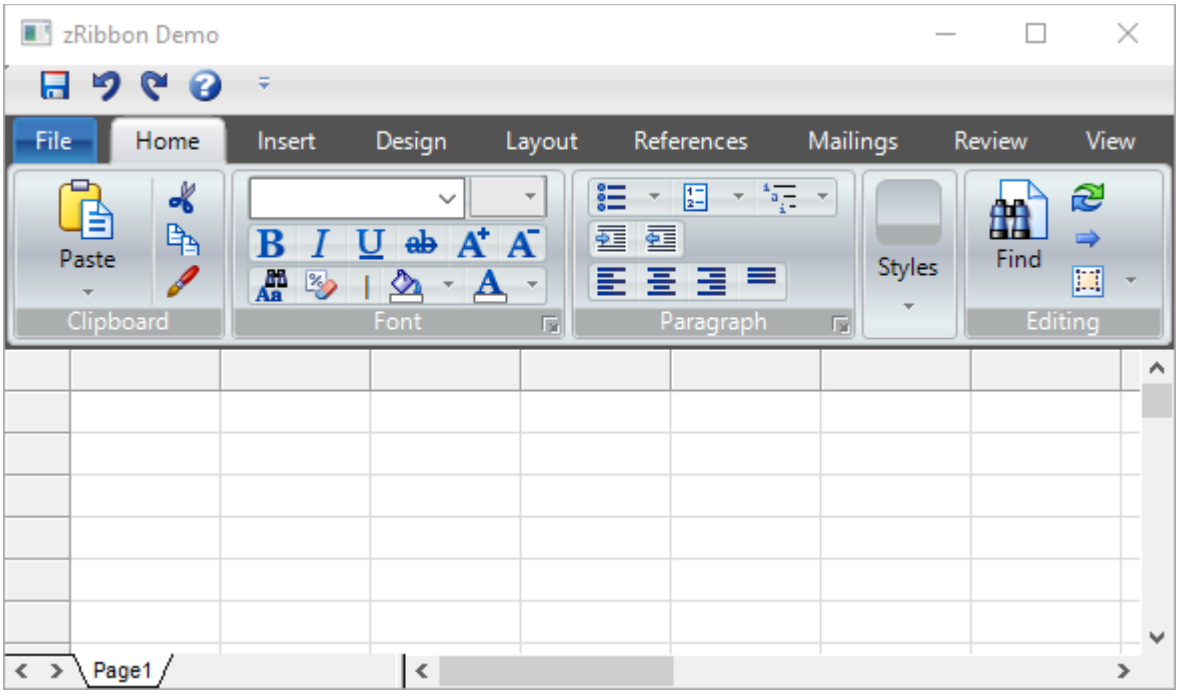
Copyright © 2016 Eric Lescasse

- A Label
- A Frame
- A Selector

This way you only have to reposition and resize a single control (the container) in the **onXRibbonResized** event handler.

The final sample ribbon with a black theme

To conclude this section on the znetRibbon object, here is how the form would like if we use the black them instead of the blue theme:



Conclusion

As you have seen a **znetRibbon** is a fairly complex object, but the **znetRibbon** API is simple to use, though a ittle verbose.

Everything has been done to make it as simple and as less verbose as possible.

Adding a **znetRibbon** to an APL application can give it a more professional and modern look as well as make it much more attractive.

Copyright © 2016 Eric Lescasse

The znetSmtpClient object

Introduction

With the **znetSmtpClient** object, you can easily programmatically send Emails from an APL+Win application.

If you study the .Net Framework, you will find out that sending an Email with attachments involves the use of a number of objects:

- SmtpClient
- MailMessage
- MailAddress
- Attachment

and is over complicated for the task to be done.

The znetSmtpClient object includes a Converter that knows how to converted an APL nested array to all these .Net objects and assemble them in the right manner allowing to send the Email.

How to use znetSmtpClient

Loading the required object

Let's first load the **znetSmtpClient** object from the zObjects.sf UCMD file:

```
]zload znetSmtpClient
```

Creating an instance of the znetSmtpClient object

To send an Email, you first need to create an instance of the **znetSmtpClient** object:

```
←'mail'⎕wi'*Create' 'znetSmtpClient'
```

Specifying the Smtp server to use

The next step, which is essential is to specify the Smtp server to use, with the **zHost** property:

```
←'mail'⎕wi'*zHost' 'smtp.orange.fr'[20]
```

If you have an Email account set up on your computer (which basically everyone has), this is all what is required and you should not have to enter your Email account credentials as they are already registered on your computer.

[20] Be sure to replace **smtp.orange.fr** by your own Smtp server. If your mail client is Outlook, you can find it by opening Outlook, selecting File/Account Settings then clicking on your account and clicking Change. The dialog that gets displayed contains your Smtp server name across the **Outgoing mail server (SMTP)** label. If you deliver your application to clients you should put the Smtp Server name in a .INI or .XML configuration file and have your application read it from there.

Copyright © 2016 Eric Lescasse

Specifying the mail to be sent

With the **znetSmtpClient** object all you have to do is to define a nested vector of property/value pairs corresponding to the Email you want to send.

The properties to use are:

- **Sender**
- **From**
- **To**
- CC
- Bcc
- **Subject**
- **Body**
- IsBodyHtml
- Priority
- Attachments

The properties in **bold** must be set, the other ones are optional. Note that the property names are case sensitive.

Here is an example:

```
mailmsg←''
mailmsg,←⊂'Sender' 'eric@lescasse.com'
mailmsg,←⊂'From' 'eric@lescasse.com'
mailmsg,←⊂'To'('eric@lescasse.com' 'info@lescasse.com')
mailmsg,←⊂'CC' 'eric@lescasse.com' 'Eric Lescasse'
mailmsg,←⊂'Bcc' 'info@lescasse.com'
mailmsg,←⊂'Subject' 'Test with a MailMessage using :case 2'
mailmsg,←⊂'Body' '<h1>Title</h1><p>This is a <b>Test</b></p><ul><li>using
         znetSmtpClient</li><li>using a MailMessage</li></ul>'
mailmsg,←⊂'IsBodyHtml'1
mailmsg,←⊂'Priority'2        ⍝ 2=MailPriority.High
mailmsg,←⊂'Attachments'('c:\temp\test.w3' 'c:\temp\test.xml')
```

Sending the Email

Just call the **zSend_2** method passing it the property/value pairs nested vector to send the Email:

```
←'mail'□wi'*zSend_2'mailmsg
```

Sample function showing how to send an Email

The following sample function show how to send an Email:

Copyright © 2016 Eric Lescasse

```
    ∇ zznetsmtpclient a;body;from;mailmsg;subject;to
[1]   ⍝∇ This function demonstrates how to use the znetSmtpClient object
[2]
[3]   :select a
[4]   :case 1
[5]       ⍝ Sending a simple Email
[6]       from←'eric@lescasse.com'
[7]       to←'eric@lescasse.com'
[8]       subject←'Test sending Email using znetSmtpClient :case 1'
[9]       body←'This is a simple test!'
[10]      ←'mail'⎕wi'*Create' 'znetSmtpClient'
[11]      ←'mail'⎕wi'*zHost' 'smtp.orange.fr'       ⍝ be sure to use your own
                  SMTP server here
[12]      ←'mail'⎕wi'*zSend'from to subject body
[13]  :case 2
[14]      ⍝ Sending an Email with more options and attachments
[15]      ←'mail'⎕wi'*Create' 'znetSmtpClient'
[16]      ←'mail'⎕wi'*zHost' 'smtp.orange.fr'       ⍝ be sure to use your own
                  SMTP server here
[17]      mailmsg←''                    ⍝ note: properties are case sensitive
[18]      mailmsg,←⊂'Sender' 'eric@lescasse.com'
[19]      mailmsg,←⊂'From' 'eric@lescasse.com'
[20]      mailmsg,←⊂'To'('eric@lescasse.com' 'info@lescasse.com')
[21]      mailmsg,←⊂'CC' 'eric@lescasse.com' 'Eric Lescasse'
[22]      mailmsg,←⊂'Bcc' 'info@lescasse.com'
[23]      mailmsg,←⊂'Subject' 'Test sending Email using znetSmtpClient :case 2'
[24]      mailmsg,←⊂'Body' '<h1>Title</h1><p>This is a <b>Test</b></p><ul><li>us
                  ing znetSmtpClient</li><li>using a MailMessage</li></ul>'
[25]      mailmsg,←⊂'IsBodyHtml'1
[26]      mailmsg,←⊂'Priority'2         ⍝ 0=MailPriority.Normal, 1=.Low, 2=.High
[27]      mailmsg,←⊂'Attachments'('c:\temp\test.w3' 'c:\temp\test.xml')  ⍝ note:
                  if more than 1, embed list in parens
[28]      ←'mail'⎕wi'*zSend_2'mailmsg
[29]  :endselect
    ∇
```

Copyright © 2016 Eric Lescasse

The znetWebClient object

Introduction

The znetWebClient object is a non-visual object that allows you to:

- Download Web pages content
- Download files with progress indication (asynchronously or not)
- Upload files with progress indication (asynchronously or not)
- Post data to a given Web site

and more.

How to use znetWebClient

Loading the znetWebClient object

Let's load both the znetWebClient object and the zznetwebclient demo.

```
]zload znetWebClient zznetwebclient
```

Creating an instance of znetWebClient

```
←'wb'□wi'*Create' 'znetWebClient'
```

The znetWebClient API

```
Main Properties
---------------
*zBaseAddress  *zCredentials  *zIsBusy  *zUseDefaultCredentials

Main Methods
------------
*zCancelAsync            *zOpenReadAsync      *zUploadFile_2
*zDoc                    *zOpenReadAsync_2    *zUploadFileAsync
*zDownloadData           *zOpenWriteAsync     *zUploadFileAsync_2
*zDownloadDataAsync      *zOpenWriteAsync_2   *zUploadFileAsync_3
*zDownloadDataAsync_2    *zOpenWriteAsync_3   *zUploadString
*zDownloadFile           *zUploadData         *zUploadString_2
*zDownloadFileAsync      *zUploadData_2       *zUploadStringAsync
*zDownloadFileAsync_2    *zUploadDataAsync    *zUploadStringAsync_2
*zDownloadString         *zUploadDataAsync_2  *zUploadStringAsync_3
*zDownloadStringAsync    *zUploadDataAsync_3
*zDownloadStringAsync_2  *zUploadFile

Main Events
-----------
*onXDownloadDataCompleted    *onXOpenReadCompleted    *onXUploadProgressChanged
*onXDownloadFileCompleted    *onXOpenWriteCompleted   *onXUploadStringCompleted
*onXDownloadProgressChanged  *onXUploadDataCompleted  *onXUploadValuesCompleted
*onXDownloadStringCompleted  *onXUploadFileCompleted
```

Copyright © 2016 Eric Lescasse

As you can see, most of the behavior of a **znetWebClient** object concerns uploading and downloading data from Internet.

But you can also do this with the **znetFtpWebRequest** object that we have already seen.

The main differences with the **znetFtpWebRequest** object are:

- **znetWebClient** can perform asynchronous operations
- **znetWebClient** has a whole lot of events you can handle in your application

How to use znetWebClient

Download a Web page content

The first think you can do with **znetWebClient** is to download the content of a Web page: just use the **zDownloadString** method:

```
      aaa←'wc'⍁wi'*zDownloadString' 'http://www.google.com'

      ⍴aaa
53951

      2000↑aaa
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="fr"
      ><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
      <meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.pn
      g" itemprop="image"><title>Google</title><script>(function(){window.google
      ={kEI:'BkrHVrG6J4GrsgHqxbnAAw',kEXPI:'3700251,3700388,4028790,4029815,4031
      109,4032677,4033307,4036509,4036527,4038012,4039268,4042552,4042784,404279
      2,4043492,4044606,4045841,4046304,4047780,4048346,4048596,4048882,4049501,
      4049549,4049557,4050886,4050912,4051242,4051558,4051596,4052304,4052781,40
      54284,4054551,4055380,4055554,4055776,4056039,4056162,4056589,4057135,4057
      170,4057586,4057836,4057920,4058002,4058117,4058277,4058316,4058328,405833
      7,4058343,4058384,4058624,4058636,4059318,4059328,4059362,4059438,4059505,
      4059513,4059636,4059790,4059860,4059976,4060657,4060681,4061298,8300272,83
      00310,8502094,8502315,8502349,8502451,8502691,8502985,8503012,8503039,8503
      132,8503156,8503303,8503306,8503404,8503585,8503611,10200084,10201647',aut
      huser:0,kscs:'c9c918f0_24'};google.kHL='fr';})();(function(){google.lc=[];
      google.li=0;google.getEI=function(a){for(var b;a&&(!a.getAttribute||!(b=a.
      getAttribute("eid")));)a=a.parentNode;return b||google.kEI};google.getLEI=
      function(a){for(var b=null;a&&(!a.getAttribute||!(b=a.getAttribute("leid")
      ));)a=a.parentNode;return b};google.https=function(){return"https:"==windo
      w.location.protocol};google.ml=function(){return null};google.wl=function(
      a,b){try{google.ml(Error(a),!1,b)}catch(d){}};google.time=function(){retur
      n(new Date).getTime()};google.log=function(a,b,d,e,g){a=google.logUrl(a,b,
      d,e,g);if(""!=a){b=new Image;var c=google.lc,f=google.li;c[f]=b;b.onerror=
      b.onload=b.onabort=function(){delete c[f]};window.google&&window.google.ve
      l&&window.google.vel.lu&&window.google.vel.lu(a);b.src=a;google.li=f+1}};g
      oogle.logUrl=function(a,b,d,e,g){var c="",f=google.ls||"";if(!d&&-1==b.sea
      rch("&ei=")){var h=google.getEI(e),c="&ei="+h;-1==b.search("&lei=")&&(
```

Uploading a file to an FTP Site[21]

You may use the zUploadFile or zUploadFileAsync series of methods to upload a file to aWeb site or FTP site.

You must provide your credentials to do so. If you don't you'll get the following kind of error message:

```
      ←'wc'⎕wi'*zUploadFile' 'ftp://ftp.albahari.com/log.txt' 'c:\temp\log.txt'
⎕WI ERROR: System exception -2146233079 (0x80131509) The remote server returned an
error: (530) Not logged in.
      ←'wc'⎕wi'*zUploadFile' 'ftp://ftp.albahari.com/log.txt' 'c:\temp\log.txt'
           ^
```

So let's provide our credentials using the **zCredentials** property as follows:

```
      ←'wc'⎕wi'*zCredentials'(('UserName' 'nutshell')('Password' 'oreilly'))
```

You can now upload the file ok:

```
      ←'wc'⎕wi'*zUploadFile' 'ftp://ftp.albahari.com/log.txt' 'c:\temp\log.txt'
```

But note that the upload is done synchronously, i.e. your application waits for the download to finish. Itf the file you upload is very large, this could be a problem.

In most occasions, it is better to upload files asynchronously using the **zUploadFileAsync** method.

Downloading a file from an FTP site

Downloading a file from an FTP site is very similar. Providing that you have set the zCredentials property ok, all you have to do is:

```
      ←'wc'⎕wi'*zDownloadFile' 'ftp://ftp.albahari.com/log.txt' 'c:\temp\log.txt'
```

Again, this operation is synchronous and blocks the APL application until it is finished.

Uploading a file to an FTP Site with a progress bar

The following function demonstrates how to upload a file to an FTP site showing a progress bar.

[21] As mentioned earlier for the znetFtpWebRequest object examples, the ftp.albahari.com FTP site answers very slowly to the first request in an APL Session and sometimes also afterwards. Please be patient and wait fot the operation to get done.

Copyright © 2016 Eric Lescasse

```
    ∇ zznetwebclient1 a;c;d;e;f;g;m;p;⎕io
[1]   ⍝∇ Demonstrates how to use the znetWebClient object
[2]   ⍝∇ to upload a file to an FTP site with a progress bar
[3]
[4]   ⎕io←1
[5]   ⍝ File to upload
[6]   f←'c:\temp\log.txt'
[7]
[8]   :select a
[9]   :case''
[10]
[11]      ⍝ Create some UI to display the upload progress
[12]      ←'ff'⎕wi'*Create' 'zForm'('DemoShow'200 400 1 1)
[13]      ←'ff'⎕wi'*caption' 'Uploading a file'
[14]      ←'ff'⎕wi'*.ll.Create' 'zLabel'('wherelc'0 0 13'>')
[15]      ←'ll'⎕wi'*caption'('Uploading file: ',f)
[16]      ←'ff'⎕wi'*.pp.Create' 'zProgress'('wherelc' '>' '=' '=' '=')('anchor' 'lrt')
[17]      ←'ff'⎕wi'*.wa.Create' 'zLabel'('wherelc' '>' '=' '=' '=')('*caption' '')
[18]      ←'ff'⎕wi'*.wb.Create' 'zLabel'('wherelc' '>' '=' '=' '=')('*caption' '')
[19]
[20]      ⍝ Create znetWebClient object
[21]      ←'wc'⎕wi'*Create' 'znetWebClient'
[22]      ⍝ Provide credentials
[23]      ←'wc'⎕wi'*zCredentials' 'nutshell' 'oreilly'
[24]      ⍝ Events
[25]      ←'wc'⎕wi'*onXUploadProgressChanged' 'zznetwebclient1"onXUploadProgressChanged"'
[26]      ←'wc'⎕wi'*onXUploadFileCompleted' 'zznetwebclient1"onXUploadFileCompleted"'
[27]      ⍝ Upload the file
[28]      ←'wc'⎕wi'*zUploadFileAsync' 'ftp://ftp.albahari.com/log.txt' f
[29]
[30]  :case'onXUploadProgressChanged'
[31]      ⎕warg[5]←⎕warg[5]×2*100≠|5⊃⎕warg
[32]      (c d e g p)←5↑⎕warg
[33]      m←zzDEB(,'CI16'⎕fmt e),' out of ',(,'CI16'⎕fmt g)
[34]      m,←,' bytes uploaded (',(⍕p),'%)'
[35]      ←'ff'⎕wi'*.pp.value'(5⊃⎕warg)
[36]      ←'wa'⎕wi'*caption'm
[37]
[38]  :case'onXUploadFileCompleted'
[39]      ←'ll'⎕wi'*caption' 'The file has uploaded successfully!'
[40]
[41]  :endselect
    ∇
```

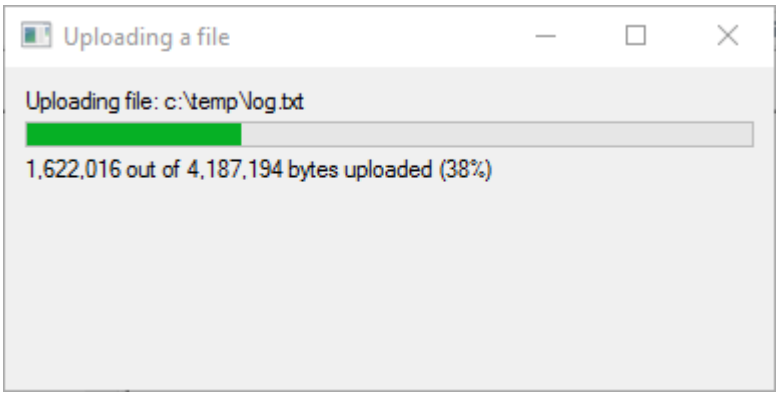
When the **onXUploadProgressPercentage** event fires, ⎕warg[contains the following values:

⎕warg[1] = the number of bytes received
⎕warg[2] = the total number of bytes to receive
⎕warg[3] = the number of bytes sent
⎕warg[4] = the total number of bytes to send
⎕warg[5] = the progress percentage (int)

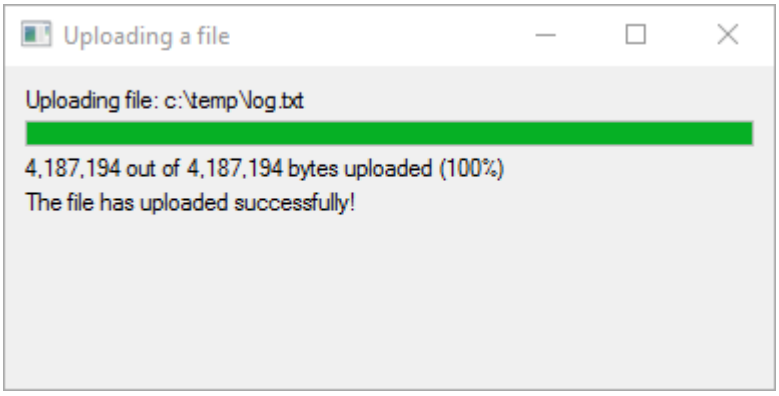
Note that the **onXUploadProgressChanged** .Net event works a little strangely and provides a progress percentage which is half what it should be, except for the value 100!

Copyright © 2016 Eric Lescasse

Hence line 31 which forces the progress percentage (contained in ⎕warg[5]) to be correct.



At the end of the upload we see:



Downloading a file from an FTP site with a progress bar

The principle is the same as for uploading a file to an FTP site with a progress bar, but the handing of the **onXDownloadProgressChanged** event is a little more complex.

The following sample function does the job:

```
    ∇ zznetwebclient2 a;c;d;e;f;m;p;s;⎕io
[1]   ⍝∇ Demonstrates how to use the znetWebClient object
[2]   ⍝∇ to download a file to an FTP site with a progress bar
[3]
[4]   ⎕io←1
[5]   ⍝ File to upload
[6]   f←'c:\temp\log.txt'
[7]
[8]   :select a
[9]   :case''
[10]
[11]      ⍝ Create some UI to display the upload progress
[12]      ←'ff'⎕wi'*Create' 'zForm'('DemoShow'200 400 1 1)
[13]      ←'ff'⎕wi'*caption' 'Downloading a file'
```

```
[14]      ←'ff'⎕wi'*.ll.Create' 'zLabel'('wherelc'0 0 13'>')
[15]      ←'ll'⎕wi'*caption'('Downloading file: ',f)
[16]      ←'ff'⎕wi'*.pp.Create' 'zProgress'('wherelc' '>' '=' '=' '=')('anchor' 'lrt')
[17]      ←'ff'⎕wi'*.wa.Create' 'zLabel'('wherelc' '>' '=' '=' '=')('*caption' '')
[18]      ←'ff'⎕wi'*.wb.Create' 'zLabel'('wherelc' '>' '=' '=' '=')('*caption' '')
[19]
[20]      ⍝ Create a znetFtpWebRequest object
[21]      ←'req'⎕wi'*Create' 'znetFtpWebRequest'
[22]      ←'req'⎕wi'*zCreate' 'ftp://ftp.albahari.com' 'nutshell' 'oreilly'
[23]      s←'req'⎕wi'*zGetFileSize'((⍵^\⍵'\'≠f)/f)      ⍝ get size of file to download
[24]      ←'ff'⎕wi'*ΔΔfilesize's
[25]      ←'req'⎕wi'*Close'
[26]      ←'req'⎕wi'*Delete'
[27]
[28]      ⍝ Create znetWebClient object
[29]      ←'wc'⎕wi'*Create' 'znetWebClient'
[30]      ⍝ Provide credentials
[31]      ←'wc'⎕wi'*zCredentials' 'nutshell' 'oreilly'
[32]      ⍝ Events
[33]      ←'wc'⎕wi'*onXDownloadProgressChanged' 'zznetwebclient2
                  "onXDownloadProgressChanged"'
[34]      ←'wc'⎕wi'*onXDownloadFileCompleted' 'zznetwebclient2"onXDownloadFileCompleted"'
[35]      ⍝ Upload the file
[36]      ←'wc'⎕wi'*zDownloadFileAsync' 'ftp://ftp.albahari.com/log.txt' f
[37]
[38]    :case'onXDownloadProgressChanged'
[39]      s←'ff'⎕wi'*ΔΔfilesize'
[40]      (c d p e)←⎕warg
[41]      p←⌊.5+100×c÷s                                ⍝ progress %
[42]      m←zzDEB(,'CI16'⎕fmt c),' out of ',(,'CI16'⎕fmt s)
[43]      m,←,' bytes downloaded (',(⍕p),'%)'
[44]      ←'ff'⎕wi'*.pp.value'p
[45]      ←'wa'⎕wi'*caption'm
[46]
[47]    :case'onXDownloadFileCompleted'
[48]      ←'ll'⎕wi'*caption' 'The file has downloaded successfully!'
[49]      ←'wc'⎕wi'*zDispose'
[50]
[51]    :endselect
    ∇
```

The problem here is that, when the onXDownloadProgressChanged event fires, ⎕warg contains the following information:

⎕warg[1] = bytes downloaded so far
⎕warg[2] = ¯1 (instead of the total number of bytes to download)
⎕warg[3] = 0 (instead of the download progress percentage
⎕warg[4] = (not used)

Hence we have to calculate the progress percentage ourselves.

First, we use a **znetFtpWebRequest** object to get the exact size of the file to download and save it in a **ΔΔfilesize** user defined variable in the **ff** form.

Copyright © 2016 Eric Lescasse

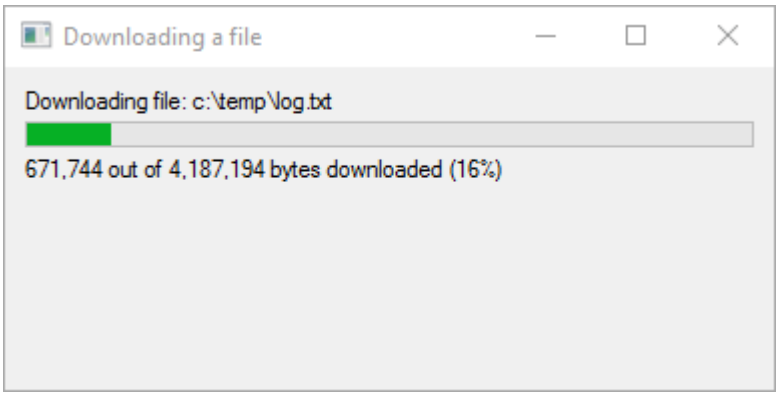
Then in the **onXDownloadProgressChanged** event handler, we retrieve the file size and compute the progress percentage (p).

Note that it is important to call **zDispose** on the **znetWebClient** object when the download has completed to free memory used by this object.

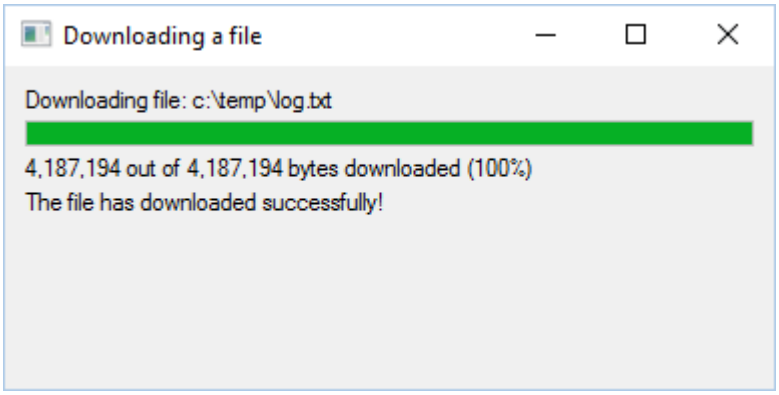
Let's run the **zznetwebclient2** function:

```
zznetwebclient2''
```

Here is what we see:



and when the file has finished downloading:



Copyright © 2016 Eric Lescasse

The znetWebBrowser object

Introduction

The znetWebBrowser object is an Internet Explorer control that you can include in your APL+Win forms.

The znetWebBrowser API

```
Main Properties
---------------
*zActiveXInstance       *zIsOffline
*zAllowNavigation       *zIsWebBrowserContextMenuEnabled
*zAllowWebBrowserDrop   *zObjectForScripting
*zCanGoBack             *zReadyState
*zCanGoForward          *zScriptErrorsSuppressed
*zDocumentText          *zScrollBarsEnabled
*zDocumentTitle         *zStatusText
*zDocumentType          *zUrl
*zEncryptionLevel       *zVersion
*zIsBusy                *zWebBrowserShortcutsEnabled

Main Methods
------------
*zDoc         *zGoSearch     *zNavigate_4           *zShowPrintPreviewDialog
*zGoBack      *zNavigate     *zPrint                *zShowPropertiesDialog
*zGoForward   *zNavigate_2   *zShowPageSetupDialog  *zShowSaveAsDialog
*zGoHome      *zNavigate_3   *zShowPrintDialog      *zStop

Main Events
-----------
*onXCanGoBackChanged       *onXEncryptionLevelChanged  *onXNewWindow
*onXCanGoForwardChanged    *onXFileDownload            *onXProgressChanged
*onXDocumentCompleted      *onXNavigated               *onXStatusTextChanged
*onXDocumentTitleChanged   *onXNavigating
```

How to use znetWebBrowser

Loading the znetWebBrowser object

Load the following functions:

```
      ]zload znetWebBrowser zznetwebbrowser
```

Creating an instance of the znetWebBrowser object

The znetWebBrowser object is a control so it needs to be a childe of an APL or C# Form or-container.

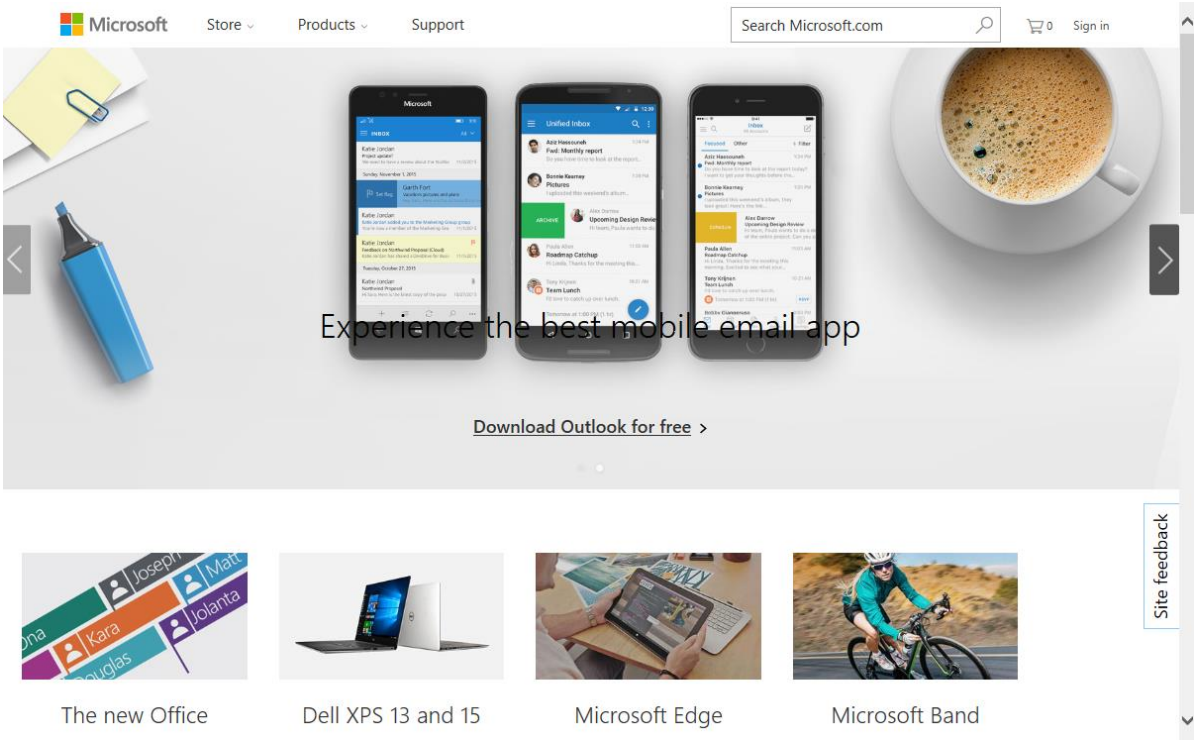
The following function demonstrates basic use of the znetWebBrowser object:

Copyright © 2016 Eric Lescasse

```
    ∇ zznetwebbrowser a
[1]   ⍝∇ Sample function demonstrating how to use the znetWebBrowser object
[2]
[3]   :select a
[4]   :case''
[5]
[6]       ←'ff'□wi'*Create' 'zForm'('size'.5 .5)('*caption' 'znetWebBrowser Demo')
[7]       ←'ff'□wi'*.wb.Create' 'znetWebBrowser'('wherelc'0 0'>>' '>>')
              ('anchor' 'lrtb')
[8]       ←'ff'□wi'*.wb.zBorderStyle'0
[9]       ←'ff'□wi'*.wb.zNavigate' 'http://www.microsoft.com'
[10]      ←'ff'□wi'CenterScreen'
[11]      ←'ff'□wi'Show'
[12]  :endselect
[13]
    ∇
```

Let's try it:

```
      zznetwebbrowser''
```



Using the **zNavigate** method is enough to load a Web page in the browser.

Preventing script errors from being displayed

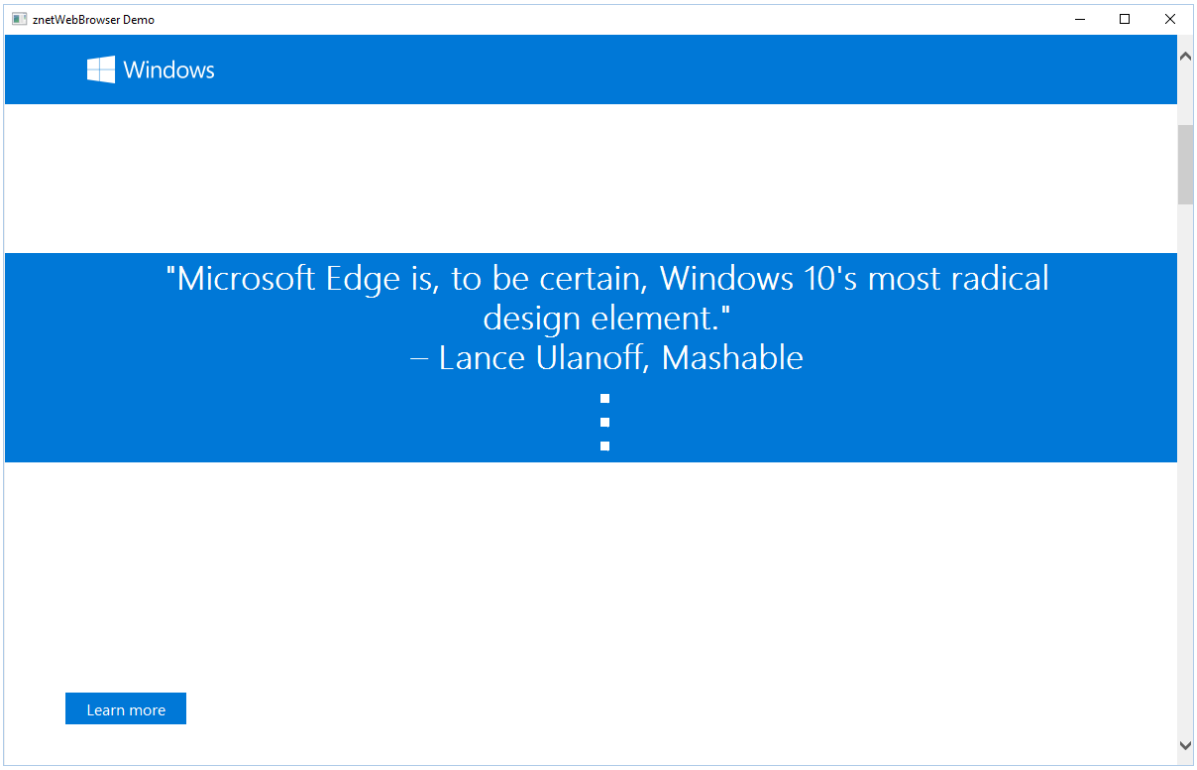
However, if you click on the Microsoft Edge image, you'll get the following error being displayed:

Copyright © 2016 Eric Lescasse

This is not a default of the **znetWebBrowser** object but rather a defect in the Microsoft Edge Web page!

If you click **Yes**, another error will be displayed. If you click **Yes** again, still another error will be displayed. You'll have to click Yes 4 or 5 times to finally be able to load the page. This is of course annoying.

You can prevent such errors from being displayed by setting the **zScriptErrorsSuppressed** property to **true**: `'wb'wi'*zScriptErrorsSuppressed'1`



Copyright © 2016 Eric Lescasse

The onXDocumentCompleted event

The second most important thing in the **znetWebBrowser** object is the **onXDocumentCompleted** event.

Assume you'd like to retrieve the content of the Web page you are loading in the znetWebBrowser object. You can do that using the **zDocumentText** property.

You might first think about adding a call to **zDocumentText** juad after the call to **zNavigate** in the zznetwebbrowser program.

Unfortunately, that would be forgetting that a method like zNavigate is asynchronous and that it takes time to go fetch a Web page on Internet.

Instead you want to handle the **onXDocumentCompleted** event and only then make your call to **zDocumentText**.

The following function demonstrates this:

```
    ∇ zznetwebbrowser1 a;m
[1]   ⍝∇ Sample function demonstrating how to use the znetWebBrowser object
[2]
[3]   :select a
[4]   :case''
[5]       ←'ff'⎕wi'*Create' 'zForm'('size'.5 .5)('*caption' 'znetWebBrowser Demo')
[6]       ←'ff'⎕wi'*.wbr.Create' 'znetWebBrowser'('wherelc'0 0'>>' '>>')('anchor' 'lrtb')
[7]       ←'wbr'⎕wi'*zScriptErrorsSuppressed'1
[8]       ←'wbr'⎕wi'*zBorderStyle'0
[9]       ←'wbr'⎕wi'*onXDocumentCompleted' 'zznetwebbrowser1"onXDocumentCompleted"'
[10]      ←'wbr'⎕wi'*zNavigate' 'http://www.microsoft.com'
[11]
[12]      m←'The document has not had time to load yet!'
[13]      m,←⎕tcnl,' zReadyState= ','wbr'⎕wi'*zReadyState'
[14]      m,←⎕tcnl,' zIsBusy= ','wbr'⎕wi'*zIsBusy'
[15]      m,←⎕tcnl,' zStatusText= ','wbr'⎕wi'*zStatusText'
[16]      ⎕←m
[17]      ⎕←⍴'wbr'⎕wi'*zDocumentText'
[18]
[19]      ←'ff'⎕wi'CenterScreen'
[20]      ←'ff'⎕wi'Show'
[21]  :case'onXDocumentCompleted'
[22]      m←'The document has been loaded!'
[23]      m,←⎕tcnl,'⎕warg= ',⍕⎕warg
[24]      m,←⎕tcnl,' zReadyState= ','wbr'⎕wi'*zReadyState'
[25]      m,←⎕tcnl,' zIsBusy= ','wbr'⎕wi'*zIsBusy'
[26]      m,←⎕tcnl,' zStatusText= ','wbr'⎕wi'*zStatusText'
[27]      ⎕←m
[28]      ⎕←⍴⎕wi'*zDocumentText'
[29]
[30]  :endselect
[31]
    ∇
```

If you run this function, you'll get the following output:

```
      zznetwebbrowser1''
      zznetwebbrowser1''
The document has not had time to load yet!
 zReadyState=  0
 zIsBusy=  0
 zStatusText=
0
The document has been loaded!
⎕warg=  http://www.microsoft.com/en-us
 zReadyState=  4
 zIsBusy=  0
 zStatusText= Done
89614
```

Programmatically navigating

You can use **zNavigate** to navigate to various Web pages, for example when a user clicks on various buttons in your User interface.

You can then navigate back, forward or home among the pages that have already been visited, by using the **zGoBack**, **zGoForward**, **zGoHome** methods.

Programmatically filling the znetWebBrowser document

You can use the znetWebBrowser object together with the zHtml object to dynamically create HTML Web pages and display them in your APL+Win forms.

This is very practical for displaying reports that includes text, tables of numeric results, images, etc.

Please see the zHtml object to find out how to programmatically create perfect HTML pages.

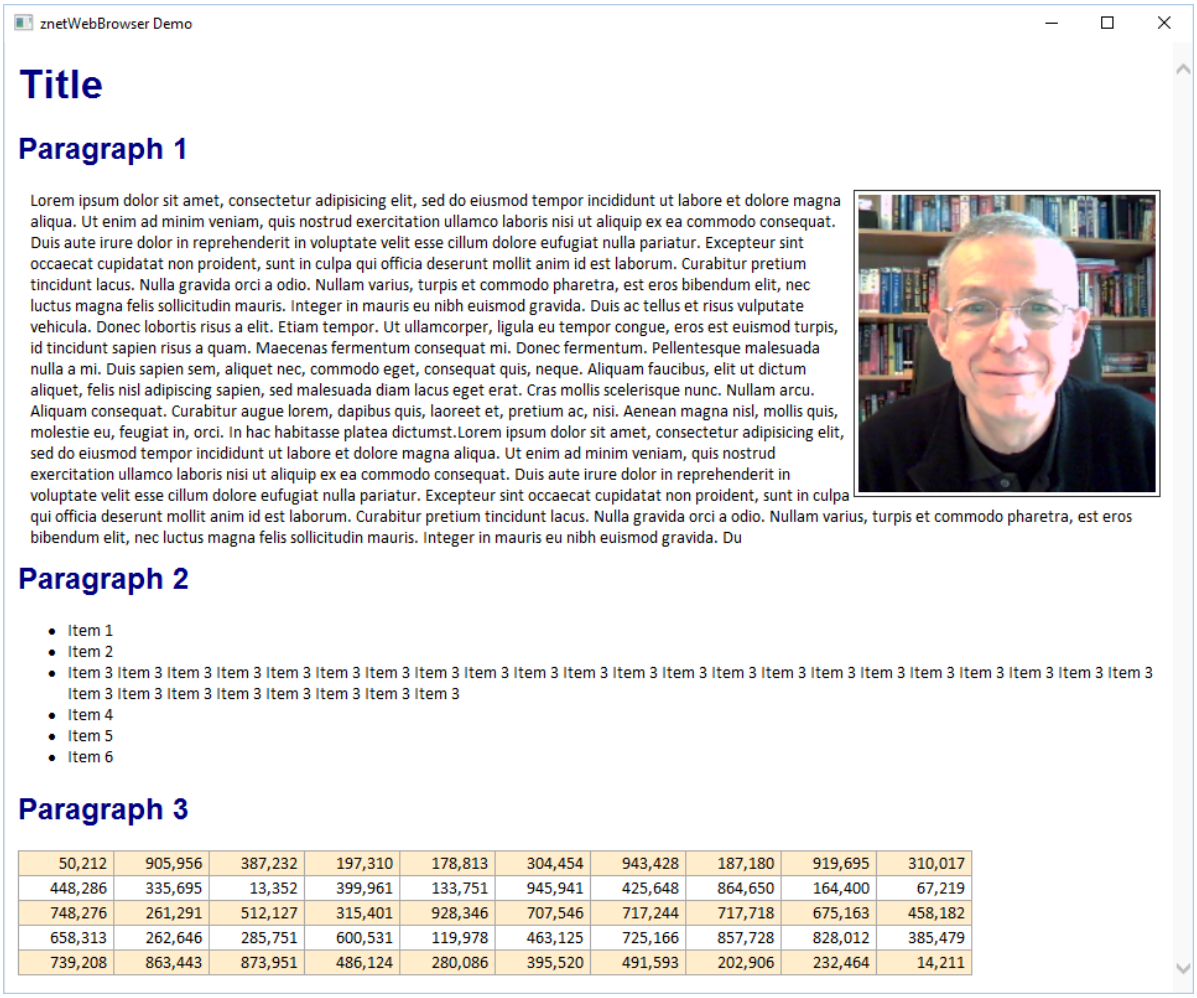
The following sample function:

```
    ∇ zznetwebbrowser2 a
[1]   A∇ Sample function demonstrating how to use the znetWebBrowser object
[2]
[3]   :select a
[4]   :case''
[5]       ←'ff'⎕wi'*Create' 'zForm'('size'.7 .5)('*caption' 'znetWebBrowser Demo')
[6]       ←'ff'⎕wi'*.wbr.Create' 'znetWebBrowser'('wherelc'0 0'>>' '>>')('anchor' 'lrtb')
[7]       ←'wbr'⎕wi'*zScriptErrorsSuppressed'1
[8]       ←'wbr'⎕wi'*zBorderStyle'0
[9]       ←'wbr'⎕wi'*zDocumentText'zzhtml2
[10]      ←'ff'⎕wi'CenterScreen'
[11]      ←'ff'⎕wi'Show'
[12]  :endselect
[13]
    ∇
```

uses the zzhtml2 demo function to programmatically create a Web page and then displays it in a znetWebBrowser object in a zForm.

Here is what gets displayed:

```
zznetwebbrowser2 ''
```



The znetXmlDocument object

Introduction

The **znetXmlDocument** object allows you to query an XML file and extract any data you are interested in from this XML file.

To do so, it uses an XML query language called **XPath**[22].

How to use znetXmlDocument

Loading the znetXmlDocument object

Let's first load the **znetXmlDocument** object from the **zObjects.sf** file:

```
      ]zload znetXmlDocument
```

The sample XML file

To explain **znetXmlDocument**, we will use an XML file that you can create by running the **zznetxmlwriter** APL sample function.

This function creates an XML file called **c:\temp\test.xml**, so first ensure you have a **c:\temp** folder and that you don't have an important **test.xml** file in this folder as it will be overwritten by running **zznetxmlwriter**.

```
      ]zload zznetxmlwriter
      zznetxmlwriter
<?xml version="1.0" encoding="utf-8"?>
<employees>
  <employee id="1" age="45">
    <!--Should be promoted-->
    <firstname>David</firstname>
    <lastname>Smith</lastname>
    <salary>10000</salary>
    <children>
      <child name="Elisa" age="10" />
      <child name="Maria" age="12" />
    </children>
  </employee>
  <employee id="3" age="52">
    <firstname>Mark</firstname>
    <lastname>Drinkwater</lastname>
    <salary>30000</salary>
    <children>
      <child name="John" age="3" />
      <child name="Elisa" age="10" />
      <child name="Maria" age="12" />
```

[22] You can learn more about XPath at: http://www.w3schools.com/xml/xml_xpath.asp

Copyright © 2016 Eric Lescasse

```
    </children>
  </employee>
  <employee id="4" age="24">
    <!--Recommendation for being fired!-->
    <firstname>Norah</firstname>
    <lastname>Miller</lastname>
    <salary>20000</salary>
  </employee>
  <employee id="12" age="33">
    <firstname>Cecil</firstname>
    <lastname>Walker</lastname>
    <salary>120000</salary>
    <children>
      <child name="Samuel" age="18" />
    </children>
  </employee>
</employees>
```

Creating an instance of znetXmlWriter and loading the XML document

The first 2 steps are to create an instance of **znetXmlWriter** and then to load the XML document using the **zLoad** method:

```
      ←'xml'□wi'*Create' 'znetXmlDocument'
      ←'xml'□wi'*zLoad' 'c:\temp\test.xml'
```

We are no ready to extract data from the XML document.

Extracting data from an XML document with zSelectNodes and zSelectSingleNode

We must use the **zSelectSingleNode** method to extract data from a single node or the **zSelectNodes** method to extract data from multiple nodes.

Let's first show a couple of examples and then explain how to use XPath a little better:

```
      'xml'□wi'*zSelectNodes' '//employee/@age'
 age Attribute 1 45 45 age="45"
 age Attribute 1 52 52 age="52"
 age Attribute 1 24 24 age="24"
 age Attribute 1 33 33 age="33"

      ]display 'xml'□wi'*zSelectNodes' '//employee/@age'
.→4 6---------------------------------.
↓.→3-..→9-------.   .→2..→2..→8------.|
||age||Attribute| 1 |45||45||age="45"||
|'---''---------'   '--''--''--------'|
|.→3-..→9-------.   .→2..→2..→8------.|
||age||Attribute| 1 |52||52||age="52"||
|'---''---------'   '--''--''--------'|
|.→3-..→9-------.   .→2..→2..→8------.|
||age||Attribute| 1 |24||24||age="24"||
|'---''---------'   '--''--''--------'|
|.→3-..→9-------.   .→2..→2..→8------.|
||age||Attribute| 1 |33||33||age="33"||
```

```
|'---''---------'   '--''--''--------'|
'ϵ-------------------------------------'

      'xml'⍁wi'*zSelectSingleNode' '/employees/employee[3]/lastname'
 lastname Element 1 Miller Miller <lastname>Miller</lastname>

      ]display 'xml'⍁wi'*zSelectSingleNode' '/employees/employee[3]/lastname'
.→6-------------------------------------------------------------------.
|.→8------..→7-----.   .→6----..→6----..→27-----------------------.|
||lastname||Element| 1 |Miller||Miller||<lastname>Miller</lastname>||
|'--------''-------'   '------''------''-------------------------'|
'ϵ-------------------------------------------------------------------'
```

The result of the **zSelectNodes** and **zSelectSingleNode** method is always an Nx6 nested matrix, as follows:

[;1]= node name (string)
[;2]= node type (string)
[;3]= has children (bool)
[;4]= inner text (string)
[;5]= inner xml (string)
[;6]= outer xml (string)

In general, the data you extract can be found in the 4th column:

```
      ⊃('xml'⍁wi'*zSelectNodes' '//employee')[;4]
DavidSmith10000
MarkDrinkwater30000
NorahMiller20000
CecilWalker120000
```

Note however that the extracted data are stuck together.

So we would be better to extract data with 5 successive instructions as follows:

```
      ('xml'⍁wi'*zSelectNodes' '//employee/@id')[;4]
 1 3 4 12
      ('xml'⍁wi'*zSelectNodes' '//employee/@age')[;4]
 45 52 24 33
      ('xml'⍁wi'*zSelectNodes' '//employee/lastname')[;4]
 Smith Drinkwater Miller Walker
      ('xml'⍁wi'*zSelectNodes' '//employee/firstname')[;4]
 David Mark Norah Cecil
      ('xml'⍁wi'*zSelectNodes' '//employee/salary')[;4]
 10000 30000 20000 120000
```

Note:
You should use **zSelectNodes** when your query may return more than one node.

You should use **zSelectSingleNode** if you are sure your query will return only one node.

Copyright © 2016 Eric Lescasse

Extracting data with the SelectData method

Extracting data with multiple calls to **zSelectNodes** is a bit cumbersome and too verbose for the developer, so I created a much easier to use **SelectData** method in **znetXmlDocument**.

Here is how it can be used:

```
      'xml'⎕wi'SelectData' '//employee/' '@id @age lastname firstname salary'
 1  45 Smith      David 10000
 3  52 Drinkwater Mark  30000
 4  24 Miller     Norah 20000
 12 33 Walker     Cecil 120000

      ]display 'xml'⎕wi'SelectData' '//employee/' '@id @age lastname firstname
salary'
.→4 5-------------------------------.
↓.→. .→2..→5---.     .→5---..→5---. |
||1| |45||Smith|     |David||10000| |
|'-' '--''-----'     '-----''-----' |
|.→. .→2..→10------..→4--. .→5---. |
||3| |52||Drinkwater||Mark| |30000| |
|'-' '--''----------''----' '-----' |
|.→. .→2..→6----.     .→5---..→5---. |
||4| |24||Miller|     |Norah||20000| |
|'-' '--''------'     '-----''-----' |
|.→2..→2..→6----.     .→5---..→6----.|
||12||33||Walker|     |Cecil||120000||
|'--''--''------'     '-----''------'|
'∈---------------------------------'
```

This method uses the APL ⌷ indexing primitive to extract the 4th columns from each element of a nested vector of matrices:

```
:region-----------------------------------------------SelectData
∆SelectData:
    ⍝∇ Extracts data from an XML file
    ⍝∇ Syntax:  nm←'obj'⎕wi'SelectData'pathradical entities
    ⍝∇ pathradical:  the XPath radical to append to entities
    ⍝∇ entities:  a string with a mix of attributes names prefixed by @ and element names
    ⍝∇ nm:  the nested matrix result
    ⍝∇ Example:
    ⍝∇       'xml'⎕wi'SelectData' '//employee/' '@id @age lastname firstname salary'
    ⍝∇ 1  45 Smith      David 10000
    ⍝∇ 3  52 Drinkwater Mark  30000
    ⍝∇ 4  24 Miller     Norah 20000
    ⍝∇ 12 33 Walker     Cecil 120000
    (d c)←1↓⎕warg
    c←(c≠' ')⊂c
    ⎕wres←⍉⊃4⌷[2]¨⎕wi¨(⊂⊂'*zSelectNodes'),¨⊂¨(⊂d),¨c
    :return
    :endregion
```

Learning more about XPath

The XPath language uses the following characters:

/	select from the root node (example: employees)
//	select elements matching the selection no matter where they are
.	select the current node
..	select the parent node
@	select attributes
*	select all
|	allows to use multiple selection (careful: | means AND)

More XPath examples

Here are a number of other XPath examples you can try, and their meanings:

Note:
Always remember that XML is a case sensitive language, so your XPath queries are also case sensitive!

With **zSelectNodes:**

XPath Query	Meaning:
employees	extract the employees node (there's only one) as a 1x6 nested matrix
//employee	extract all employee nodes no matter where they are
employees//employee	extract all employee nodes which are descendants of employees no matter where they are
//@age	extract all age attributes no matter where they are
//employee/@age	extract all employees ages
//child/@age	extract employees children ages
//child[@name=''Maria'']/../..	extract the employees that have a child named Maria
/employees/employee[position()<3]	extract the first 2 employees
/employees/employee[last-name=''Smith'']/*	extract all child nodes of Smith

Copyright © 2016 Eric Lescasse

With **zSelectSingleNode**:

XPath Query	Meaning:
employees	extract the employees node
/employees/employee[1]	extract the first employee
/ /employee[last()]	extract the last employee
/ /employee[last()-1]	extract the next to last employee
/ /employee[@id=3]	extract the first employee
/ /employee[lastname='Drinkwater']	extract employee who's lastname is Drink-water
/ /employee[lastname=''Smith'']/firstname	extract Smith's first name
/ /employee[lastname=''Smith'']/@age	extract Smith's age
/ /employee[lastname=''Smith'']/children	extract Smith children

You can of course use all these queries with the SelectData method:

```
      'xml'□wi'SelectData' '//employee[lastname=''Drinkwater'']/' '@id @age
salary'
 3 52 30000

      'xml'□wi'SelectData' '/employees/employee[last()-1]/' 'firstname lastname'
 Norah Miller

      'xml'□wi'SelectData' '//child[@name=''Maria'']/../../' 'firstname lastname'
 David Smith
 Mark  Drinkwater

     'xml'□wi'SelectData' '//employee[lastname=''DrinkWater'']/children/child/'
'@name @age'
```

Oops! We do not get any result while we know this employee has 3 children!

Can you find the error in our query? (answer next page)

Copyright © 2016 Eric Lescasse

You must remember everything is case sensitive with XML.

The error we made is to have wrongly spelled DrinkWater: it should have been Drinkwater (as it is in the XML document)!

```
     'xml'□wi'SelectData' '//employee[lastname=''Drinkwater'']/children/child/'
'@name @age'
 John  3
 Elisa 10
 Maria 12
```

Copyright © 2016 Eric Lescasse

The znetXmlWriter object

Introduction

In your APL applications, you sometimes need to be able to create an XML file based on some APL data.

The .Net Framework contains a whole bunch of XML classes and several ways to create XML files.

The znetXmlWriter object allows you to create an XML document using the .Net XmlWriter approach.

How to use znetXmlWriter

Loading the object

As usual the first step is to bring in the object from the zObjects.sf file:

```
      ]zload znetXmlWriter
```

Example

In the following sections, we will show how to create an XML file based on the following APL nested array variable:

```
employees←4⍴⊂''
employees[1]←⊂1"David" "Smith"10000 45(('Elisa'10)('Maria'12))'Should be promoted'
employees[2]←⊂3"Mark" "Drinkwater"30000 52(('John'3)('Elisa'10)('Maria'12))''
employees[3]←⊂4"Norah" "Miller"20000 24'' 'Recommendation for being fired!'
employees[4]←⊂12"Cecil" "Walker"120000 33('Samuel'18)''
```

Let's have a look at this **employees** variable structure:

```
      ]display employees
.→4-------------------------------------------------------------------------------
|.→7-----------------------------------------------------------------------------..
||   .→5---..→5---.           .→2------------------------..→18---------------.||
|| 1 |David||Smith| 10000  45 |.→2---------..→2---------.||Should be promoted|||
||   '-----''-----'           ||.→5---.    ||.→5---.    ||'------------------'||
||                            |||Elisa| 10 |||Maria| 12 ||                    ||
||                            ||'-----'    ||'-----'    ||                    ||
||                            |'⍝----------''⍝----------'|                    ||
||                            '⍝-------------------------'                    ||
|'⍝-----------------------------------------------------------------------------''
'⍝-------------------------------------------------------------------------------
```

Copyright © 2016 Eric Lescasse

```
--------------------------------------------------------------------------------
→7------------------------------------------------------------------------..→7--
   .→4--..→10-------.           .→3-----------------------------------..⊖.||   .
 3 |Mark||Drinkwater| 30000  52 |.→2-------..→2---------..→2---------.|| ||| 4 |
   '----''----------'           ||.→4--.   ||.→5---.    ||.→5---.    ||'-'||   '
                                |||John| 3 |||Elisa| 10 |||Maria| 12 ||   |'∊---
                                ||'----'   ||'-----'    ||'-----'    ||   |
                                |'∊--------''∊----------''∊----------'|   |
                                '∊------------------------------------'   |
∊-------------------------------------------------------------------------'
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-------------------------------------------------------------..→7---------------
→5---..→6----.           .⊖..→31----------------------------.||    .→5---..→6---
Norah||Miller| 20000  24 | ||Recommendation for being fired!||| 12 |Cecil||Walke
-----''------'           '-'''-------------------------------'||    '-----''-----
-------------------------------------------------------------'|
                                                              |
                                                              '∊----------------


--------------------------------------------------------------------------------

--------------------------------.
-------------------------------.|
-.            .→2----------..⊖.||
r| 120000  33 |.→6----.    || |||
-'            ||Samuel| 18 |'-'||
              |'------'    |   ||
              '∊-----------'   ||
-------------------------------'|
                                |
                                |
--------------------------------'
```

Creating an instance of znetXmlWriter and initializing things

You must first create an instance of **znetXmlWriter** object:

```
←'xx'⎕wi'*Create' 'znetXmlWriter'
```

You should then define any settings you want to use with the zSettings property:

```
←'xx'⎕wi'*zSettings' 'indent'
```

The next step is to create the XML output file:

```
←'xx'⎕wi'*zCreate'  'c:\temp\test.xml'
```

Note that you most often want to use the **indent** settings, so that XML lines be indented in your output file. Other possible settings are:

```
      'xx'⎕wi'?Settings'
Syntax: {settings←}'obj'⎕wi'*zSettings'{settings}
```

```
Gets or sets the settings to be used by the XmlWriter object
settings:  a comma separated string of settings
Supported settings are:
        CloseOutput
        DoNotEscapeUriAttributes
        Indent
        OmitXmlDeclaration
        WriteEndDocumentOnClose
```

Note:
The **zSettings** property must be set BEFORE using **zCreate** to create the XML file!

Creating the root XML node

You must then create a root XML node, using the **zWriteStartElement** method:

```
←'xx'□wi'*zWriteStartElement' 'employees'
←'xx'□wi'*zWriteEndElement'
```

Note:
It is compulsory to create a root XML node with **zWriteStartElement**, otherwise **znetXmlWriter** will fail creating the XML document ok.

Also please carefully note that there should be one call to **zWriteEndElement** for each **zWriteStartElement** call.

Creating child XML nodes

The next step is to write a loop using the **zWriteStartElement** method to create one child XML node for each employee.

```
:for employee :in employees
    ←'xx'□wi'*zWriteStartElement' 'employee'
    ←'xx'□wi'*zWriteEndElement'
:endfor
```

Properly closing the znetXmlWriter object

Once you have written all nodes to the XML file and are done, you must properly close the **znetXmlWriter** object instance with the **zClose** method:

```
←'xx'□wi'*zClose'
```

Copyright © 2016 Eric Lescasse

Looking at the XML file we created so far

We can use the znetFile object to easily read the entire content of the XML file we just created:

```
      ←'io'⎕wi'*Create' 'znetFile'
     ⎕tclf⍨'io'⎕wi'*zReadAllText'file
<?xml version="1.0" encoding="utf-8"?>
<employees>
  <employee />
  <employee />
  <employee />
  <employee />
</employees>
```

Not very exciting yet, but we'll improve things in the next paragraphs.

So, assuming employees is a global variable in the workspace, our program should look like this so far:

```
    ∇ r←zznetxmlwriter1;employee;file
[1]   ⍝∇ Demonstrates use of the znetXmlWriter object
[2]
[3]   file←'c:\temp\test.xml'
[4]
[5]   ←'xx'⎕wi'*Create' 'znetXmlWriter'
[6]   ←'xx'⎕wi'*zSettings' 'indent'
[7]   ←'xx'⎕wi'*zCreate'file
[8]   ←'xx'⎕wi'*zWriteStartElement' 'employees'
[9]   :for employee :in employees
[10]      ←'xx'⎕wi'*zWriteStartElement' 'employee'
[11]      ←'xx'⎕wi'*zWriteEndElement'
[12]  :endfor
[13]  ←'xx'⎕wi'*zWriteEndElement'
[14]  ←'xx'⎕wi'*zClose'
[15]
[16]  ←'io'⎕wi'*Create' 'znetFile'
[17]  r←⎕tclf⍨'io'⎕wi'*zReadAllText'file
    ∇
```

Securing things with :try :catchall :finally

If you made any error in your program which consequently would fail, if you corrected the error and tried to rerun it, you would get the following error:

```
      zznetxmlwriter1
⎕WI ERROR: mscorlib exception -2147024864 (0x80070020) The process cannot access
the file 'c:\temp\test.xml' because it is being used by another process.
zznetxmlwriter1[6] ←'xx'⎕wi'*zCreate'file
                         ^
```

The reason is that the XML file you are creating is left in an **in use** state until you use zClose to close the znetXmlWriter instance.

Therefore, to protect yourself against this potential problem it is important to secure things using a :try :catchall :endtry structure.

Our function becomes:

```
    ∇ r←zznetxmlwriter1;employee;file
[1]   ⍝∇ Demonstrates use of the znetXmlWriter object
[2]
[3]   file←'c:\temp\test.xml'
[4]
[5]   ←'xx'⎕wi'*Create' 'znetXmlWriter'
[6]   ←'xx'⎕wi'*zSettings' 'indent'
[7]   :try
[8]       ←'xx'⎕wi'*zCreate'file
[9]       ←'xx'⎕wi'*zWriteStartElement' 'employees'
[10]      :for employee :in employees
[11]          ←'xx'⎕wi'*zWriteStartElement' 'employee'
[12]          ←'xx'⎕wi'*zWriteEndElement'
[13]      :endfor
[14]      ←'xx'⎕wi'*zWriteEndElement'
[15]  :catchall
[16]      ⎕error⎕dm
[17]  :finally
[18]      ←'xx'⎕wi'*zClose'
[19]  :endtry
[20]
[21]  ←'io'⎕wi'*Create' 'znetFile'
[22]  r←⎕tclf~⍨'io'⎕wi'*zReadAllText'file
    ∇
```

This is a very good example of the use of a **:finally** clause.

The **:finally** clause will run in all cases, whether an error occurs or whether the program runs fine. Hence, you are assured the **znetXmlWriter** instance is closed when your program ends.

Writing attributes with zWriteAttributeString

OK: we now have a sound program structure for creating an XML file with **znetXmlWriter**.

We would like to display the employee **id** and **age** as attributes of the **employee** nodes.

So let's improve things by using the **zWriteAttributeString** method.

Our function becomes:

```
    ∇ r←zznetxmlwriter2;employee;file
[1]   ⍝∇ Demonstrates use of the znetXmlWriter object
[2]
[3]   file←'c:\temp\test.xml'
[4]
[5]   ←'xx'⎕wi'*Create' 'znetXmlWriter'
```

Copyright © 2016 Eric Lescasse

```
[6]   ←'xx'⎕wi'*zSettings' 'indent'
[7]   :try
[8]       ←'xx'⎕wi'*zCreate'file
[9]       ←'xx'⎕wi'*zWriteStartElement' 'employees'
[10]      :for employee :in employees
[11]          ←'xx'⎕wi'*zWriteStartElement' 'employee'
[12]          ←'xx'⎕wi'*zWriteAttributeString' 'id'(⍕1⊃employee)
[13]          ←'xx'⎕wi'*zWriteAttributeString' 'age'(⍕5⊃employee)
[14]          ←'xx'⎕wi'*zWriteEndElement'
[15]      :endfor
[16]      ←'xx'⎕wi'*zWriteEndElement'
[17]  :catchall
[18]      ⎕error⎕dm
[19]  :finally
[20]      ←'xx'⎕wi'*zClose'
[21]  :endtry
[22]
[23]  ←'io'⎕wi'*Create' 'znetFile'
[24]  r←⎕tclf~⍨'io'⎕wi'*zReadAllText'file
    ∇
```

Let's run it:

```
      zznetxmlwriter2
<?xml version="1.0" encoding="utf-8"?>
<employees>
  <employee id="1" age="45" />
  <employee id="3" age="52" />
  <employee id="4" age="24" />
  <employee id="12" age="33" />
</employees>
```

Creating child elements with zWriteElementString

Now we'd like to create employee child nodes for each of the following pieces of data: first-name, lastname and salary.

We can do that using the **zWriteElementString** as follows:

```
    ∇ r←zznetxmlwriter3;employee;file
[1]   ⍝∇ Demonstrates use of the znetXmlWriter object
[2]
[3]   file←'c:\temp\test.xml'
[4]
[5]   ←'xx'⎕wi'*Create' 'znetXmlWriter'
[6]   ←'xx'⎕wi'*zSettings' 'indent'
[7]   :try
[8]       ←'xx'⎕wi'*zCreate'file
[9]       ←'xx'⎕wi'*zWriteStartElement' 'employees'
[10]      :for employee :in employees
[11]          ←'xx'⎕wi'*zWriteStartElement' 'employee'
[12]          ←'xx'⎕wi'*zWriteAttributeString' 'id'(⍕1⊃employee)
[13]          ←'xx'⎕wi'*zWriteAttributeString' 'age'(⍕5⊃employee)
[14]          ←'xx'⎕wi'*zWriteElementString' 'firstname'(2⊃employee)
[15]          ←'xx'⎕wi'*zWriteElementString' 'lastname'(3⊃employee)
[16]          ←'xx'⎕wi'*zWriteElementString' 'salary'(⍕4⊃employee)
```

```
[17]          ←'xx'⎕wi'*zWriteEndElement'
[18]      :endfor
[19]      ←'xx'⎕wi'*zWriteEndElement'
[20]  :catchall
[21]      ⎕error⎕dm
[22]  :finally
[23]      ←'xx'⎕wi'*zClose'
[24]  :endtry
[25]
[26]  ←'io'⎕wi'*Create' 'znetFile'
[27]  r←⎕tclf~⍨'io'⎕wi'*zReadAllText'file
    ∇
```

Let's run it:

```
      zznetxmlwriter3
<?xml version="1.0" encoding="utf-8"?>
<employees>
  <employee id="1" age="45">
    <firstname>David</firstname>
    <lastname>Smith</lastname>
    <salary>10000</salary>
  </employee>
  <employee id="3" age="52">
    <firstname>Mark</firstname>
    <lastname>Drinkwater</lastname>
    <salary>30000</salary>
  </employee>
  <employee id="4" age="24">
    <firstname>Norah</firstname>
    <lastname>Miller</lastname>
    <salary>20000</salary>
  </employee>
  <employee id="12" age="33">
    <firstname>Cecil</firstname>
    <lastname>Walker</lastname>
    <salary>120000</salary>
  </employee>
</employees>
```

Our XML file starts to look good!

Adding comments with zWriteComment

If you look back at the **employees** global variable[23], you'll see that the last element of each employee nested array is a comment about the employee.

We can display these comments as comments in the XML file, using the **zWriteComment** method, as follows:

[23] I would never use a global variable: I just do it here to avoid creating employees in each instance of the sample functions displayed in this document

```
    ∇ r←zznetxmlwriter4;comment;employee;file;⎕io
[1]   ⍝∇ Demonstrates use of the znetXmlWriter object
[2]
[3]   ⎕io←1
[4]   file←'c:\temp\test.xml'
[5]
[6]   ←'xx'⎕wi'*Create' 'znetXmlWriter'
[7]   ←'xx'⎕wi'*zSettings' 'indent'
[8]   :try
[9]       ←'xx'⎕wi'*zCreate'file
[10]      ←'xx'⎕wi'*zWriteStartElement' 'employees'
[11]      :for employee :in employees
[12]          ←'xx'⎕wi'*zWriteStartElement' 'employee'
[13]          ←'xx'⎕wi'*zWriteAttributeString' 'id'(⍕1⊃employee)
[14]          ←'xx'⎕wi'*zWriteAttributeString' 'age'(⍕5⊃employee)
[15]          :if~0∈⍴comment←7⊃employee
[16]              ←'xx'⎕wi'*zWriteComment'comment
[17]          :endif
[18]          ←'xx'⎕wi'*zWriteElementString' 'firstname'(2⊃employee)
[19]          ←'xx'⎕wi'*zWriteElementString' 'lastname'(3⊃employee)
[20]          ←'xx'⎕wi'*zWriteElementString' 'salary'(⍕4⊃employee)
[21]          ←'xx'⎕wi'*zWriteEndElement'
[22]      :endfor
[23]      ←'xx'⎕wi'*zWriteEndElement'
[24]  :catchall
[25]      ⎕error⎕dm
[26]  :finally
[27]      ←'xx'⎕wi'*zClose'
[28]  :endtry
[29]
[30]  ←'io'⎕wi'*Create' 'znetFile'
[31]  r←⎕tclf~⍨'io'⎕wi'*zReadAllText'file
    ∇
```

If we run it, we get:

```
<?xml version="1.0" encoding="utf-8"?>
<employees>
  <employee id="1" age="45">
    <!--Should be promoted-->
    <firstname>David</firstname>
    <lastname>Smith</lastname>
    <salary>10000</salary>
  </employee>
  <employee id="3" age="52">
    <firstname>Mark</firstname>
    <lastname>Drinkwater</lastname>
    <salary>30000</salary>
  </employee>
  <employee id="4" age="24">
    <!--Recommendation for being fired!-->
    <firstname>Norah</firstname>
    <lastname>Miller</lastname>
    <salary>20000</salary>
  </employee>
  <employee id="12" age="33">
    <firstname>Cecil</firstname>
    <lastname>Walker</lastname>
```

Copyright © 2016 Eric Lescasse

```
    <salary>120000</salary>
  </employee>
</employees>
```

Creating new elements, children of the employee element, with zWriteStartElement

Some employees have children and it seems natural to display these children as children nodes of the employee nodes.

We can do that using the same zWriteStartElement method that we used to create the employee nodes.

We just need to remember calling one zWriteEndElement for each zWriteStartElement we called.

Our final function now becomes:

```
    ∇ r←zznetxmlwriter5;child;children;comment;employee;file;⎕io
[1]   ⍝∇ Demonstrates use of the znetXmlWriter object
[2]
[3]   ⎕io←1
[4]   file←'c:\temp\test.xml'
[5]
[6]   ←'xx'⎕wi'*Create' 'znetXmlWriter'
[7]   ←'xx'⎕wi'*zSettings' 'indent'
[8]   :try
[9]       ←'xx'⎕wi'*zCreate'file
[10]      ←'xx'⎕wi'*zWriteStartElement' 'employees'
[11]      :for employee :in employees
[12]          ←'xx'⎕wi'*zWriteStartElement' 'employee'
[13]          ←'xx'⎕wi'*zWriteAttributeString' 'id'(⍕1⊃employee)
[14]          ←'xx'⎕wi'*zWriteAttributeString' 'age'(⍕5⊃employee)
[15]          :if~0∈⍴comment←7⊃employee
[16]              ←'xx'⎕wi'*zWriteComment'comment
[17]          :endif
[18]          ←'xx'⎕wi'*zWriteElementString' 'firstname'(2⊃employee)
[19]          ←'xx'⎕wi'*zWriteElementString' 'lastname'(3⊃employee)
[20]          ←'xx'⎕wi'*zWriteElementString' 'salary'(⍕4⊃employee)
[21]          :if~0∈⍴∈children←6⊃employee
[22]              :if 2=≡children ⋄ children←,⊂children ⋄ :endif
[23]              ←'xx'⎕wi'*zWriteStartElement' 'children'
[24]              :for child :in children
[25]                  ←'xx'⎕wi'*zWriteStartElement' 'child'
[26]                  ←'xx'⎕wi'*zWriteAttributeString' 'name'(1⊃child)
[27]                  ←'xx'⎕wi'*zWriteAttributeString' 'age'(⍕2⊃child)
[28]                  ←'xx'⎕wi'*zWriteEndElement'
[29]              :endfor
[30]              ←'xx'⎕wi'*zWriteEndElement'
[31]          :endif
[32]          ←'xx'⎕wi'*zWriteEndElement'
[33]      :endfor
[34]      ←'xx'⎕wi'*zWriteEndElement'
[35]  :catchall
[36]      ⎕error⎕dm
```

Copyright © 2016 Eric Lescasse

```
[37]  :finally
[38]      ←'xx'⎕wi'*zClose'
[39]  :endtry
[40]
[41]  ←'io'⎕wi'*Create' 'znetFile'
[42]  r←⎕tclf~⍨'io'⎕wi'*zReadAllText'file
      ∇
```

And its output:

```
<?xml version="1.0" encoding="utf-8"?>
<employees>
  <employee id="1" age="45">
    <!--Should be promoted-->
    <firstname>David</firstname>
    <lastname>Smith</lastname>
    <salary>10000</salary>
    <children>
      <child name="Elisa" age="10" />
      <child name="Maria" age="12" />
    </children>
  </employee>
  <employee id="3" age="52">
    <firstname>Mark</firstname>
    <lastname>Drinkwater</lastname>
    <salary>30000</salary>
    <children>
      <child name="John" age="3" />
      <child name="Elisa" age="10" />
      <child name="Maria" age="12" />
    </children>
  </employee>
  <employee id="4" age="24">
    <!--Recommendation for being fired!-->
    <firstname>Norah</firstname>
    <lastname>Miller</lastname>
    <salary>20000</salary>
  </employee>
  <employee id="12" age="33">
    <firstname>Cecil</firstname>
    <lastname>Walker</lastname>
    <salary>120000</salary>
    <children>
      <child name="Samuel" age="18" />
    </children>
  </employee>
</employees>
```

We now have created a complete and well structured XML file.

Conclusion

Creating XML files with APL is not that difficult per se, but creating well structured valid XML files is more difficult. Using **znetXmlWriter** is simple and ensures that the XML files you create are valid.

Copyright © 2016 Eric Lescasse

Extending zObjects

Introduction

At this stage, you should have a pretty good understanding of zObjects and how they can help you develop APL+Win applications more easily.

However, it is time to clearly explain how you can extend zObjects yourself.

You can extend zObjects by:

- Creating your own objects that inherit from zObjects
- Creating your own objects that inherit from standard APL+Win objects

Let's study the various possibilities.

General rules about creating your own objects

Rule # 1

The first rule is that you should always use the zObject **Derive** method to create a new object of your own. See examples later in this chapter.

Rule # 2

The second rule is that you should choose a letter prefix for your own objects (maybe u or x for example), but never use the **z** prefix: this prefix is reserved for the **zObjects** product.

This is very important since you may want to install updates of the zObjects product in the future and you want to avoid any potential conflict with your object names and the zObjects object names.

Rule # 3

You should never write any APL function of your own starting with a **z**.

Rule # 4

You should never make any change at all to an existing starting with a **z**. Provided you abide to rule # 3, all functions starting with a **z** should be considered "untouchable".

This is again very important for the same reason as the one exposed for Rule # 2.

Creating your own objects that inherit from zObjects

Reasons for inheriting from existing zObjects

As you have seen in this whole book, **zObjects** are more powerful than raw APL+Win objects.

Copyright © 2016 Eric Lescasse

There are 2 main reasons for inheriting from existing zObjects:

- The first one is the following: if you decide to use **zObjects** to develop your application, everything in your application should be an object: at least, all the forms and all the controls you are using. This is necessary for the **wherelc** and **anchor** property to work properly[24].

  Moreover, all forms in your application should inherit from **zForm** (or **zMDIForm**), for the same reason.

  So you generally start writing a new object oriented application by creating a form object of your own, deriving from **zForm** as follows:

```
)clear
]zload zzInit zObject zAPLFormTemplate
zzInit
'zz'⎕wi'Derive' 'uMyForm' 'zForm' 'zAPLFormTemplate'
```

  By doing so, you are inheriting from all the properties, methods and events contained in **zForm**, but can start developing your application User Interface in your own **uMyForm** object without touching **zForm**.

- The second reason is that you may want to:

  - Add properties, methods or events to existing **zObjects**
  - Override existing **zObjects** properties, methods or events

  Why would you want to do that?

  It is simple: you may consider that one of the **zObjects** you want to use (for example: **zEdit**) is not complete enough for your needs and that you would like to add a couple of properties, methods or events to **zEdit**.

  However, **rule # 4** states that you should never make any change to a **z** object (like **zEdit**).

  Instead, you just need to create a new **uEdit** object of your own that inherits from **zEdit**.

  Example:

[24] The reason this is essential is that the **wherelc** and **anchor** property work **recursively** throughout the whole hierarchy of object in your application. If one of the container objects in your application is not a **zObject**, for example if you use a **Selector** instead of a **zSelector**, or a **Frame** instead of a **zFrame**, you are basically breaking the wherelc or anchor recursion and things will stop working as you expected.

Copyright © 2016 Eric Lescasse

```
]zload zAPLControlTemplate
'zz'⎕wi'Derive' 'uEdit' 'zEdit' 'zAPLControlTemplate'
```

Alternatively, you may not like the way a zObject property, method or event has been designed in one of the zObjects (say **zEdit**) and would like to use a different one with a different behavior.

All you need to do is, again, to create an object (say **uEdit**) that inherits from **zEdit** and to develop your own property, method or event having the same name as the **zEdit** property, method or event you disagree with.

zObjects inheritance works in such a way that it is the first occurrence of a property or method with a given name, in the object hierarchy, which is used.

To take an example: the **zEdit** object has a **font** property and you don't like the way it works.

Just create a **uEdit** object that inherits from **zEdit** and add a **font** property (using the same name is essential here) to your **uEdit** object.

Now, when you create an instance of **uEdit** and use its **font** property, it is the **font** property defined in **uEdit** that will be used and the one defined in **zEdit** will be ignored.

Anothter case where you may want to create your own object inheriting from a zObject is if you find a better and faster algorithm for some zObjects property or method. Again, you would create your own object inheriting from a zObject and develop your own better property or method, overriding the one already defined in the zObject.

Note:
This is really nice as it allows you to change the **zObjects** behavior without changing **zObjects** at all!

You really are "driving" yourself and are not hostages of the **zObjects** product you're using!

Example creating your own object that inherits from zObjects

Let's create a **zForm** with a **zFrame**:

```
←'ff'⎕wi'*Create' 'zForm'('DemoShow'200 300 1 1)('*color'255)
←'ff'⎕wi'*.fr.Create' 'zFrame'('wherelc'0 0'>' '>')('anchor' 'lrtb')
```

Copyright © 2016 Eric Lescasse

Note that I have set the form background color to red so that we can clearly see the zFrame border.

Suppose you would prefer a **zFrame** object:

- That does not display a caption
- That has a frame starting 6 pixels higher

so that it looks like the following when you create it:

You know you could achieve the above by running the additional 2 instructions:

```
      ←'fr'⎕wi'*caption' ''
      ←'fr'⎕wi'wherelc' '∘' '∘' '∘' '∘' ¯6 0 6 0
```

but you are tired of having to always run these 2 instructions when you create a **zFrame**.

So, just create a **uFrame** object inheriting from **zFrame**:

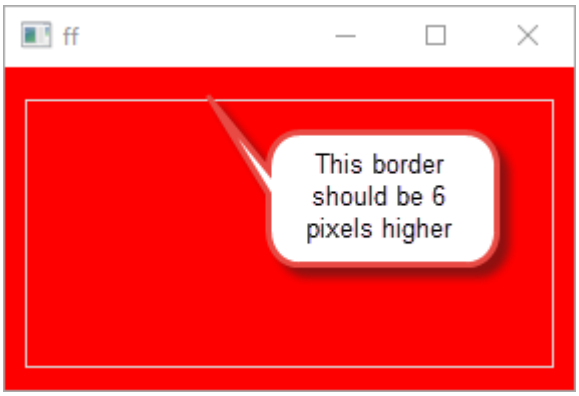
```
      'zz'⎕wi'Derive' 'uFrame' 'zFrame' 'zAPLControlTemplate'
Class <uFrame> successfully created and registered!
```

Your first idea might be to copy/paste the above 2 instructions to the **uFrame** constructor so that it reads:

```
:region----------------------------------------------Constructor
∆New:
    ⍝∇ Create a new instance of uFrame
    ⍝∇ Example:
    ⍝∇       'zt'⎕wi'*Create' 'uFrame'
    ←a⎕wi'*Create' 'zFrame'('*scale'5)
    ←a⎕wi'*onAction'(∈'uFrame' 'zFrame' 'zObject',¨⊂'"Action"',⎕tcnl)
    ←a⎕wi'*ΔΔanchor'1 2
    ←a⎕wi'clipsiblings'1
    ←zzAddLink a
    ←a⎕wi'*caption' ''
    ←a⎕wi'wherelc' '⍜' '⍜' '⍜' '⍜' ¯6 0 6 0
    :return
    :endregion
```

However, if you do so, here is what you get:

```
←'ff'⎕wi'*Create' 'zForm'('DemoShow'200 300 1 1)('*color'255)
←'ff'⎕wi'*.fr.Create' 'uFrame'('wherelc'⍬ ⍬'>' '>')('anchor' 'lrtb')
```



Unfortunately, the **uFrame** does not start 6 pixels higher, as you would expect.

This is because the **uFrame** constructor executes at the time the **Create** method runs and therefore before the **wherelc** property is run. As a consequence, when the **wherelc** property executes in:

```
←'ff'⎕wi'*.fr.Create' 'uFrame'('wherelc'⍬ ⍬'>' '>')('anchor' 'lrtb')
```

it overrides whatever **wherelc** you've already run in the **uFrame** constructor.

We must find a way to adjust **wherelc** by running ('fr'⎕wi'wherelc' '⍜' '⍜' '⍜' '⍜' ¯6 0 6 0) after the ('wherelc'⍬ ⍬'>' '>') instruction runs, not before.

There's one way to do that.

We can define a **wherelc** property in **uFrame** as follows:

Copyright © 2016 Eric Lescasse

```
:region-----------------------------------------whereIc
∆whereIc:
    :if 1=⍴⍞warg
        ⍞wres←⍞wi'base' 'zObject' 'whereIc'
    :else
        ←⍞wi'base' 'zObject',⍞warg
        ←⍞wi'base' 'zObject' 'whereIc' '∘' '∘' '∘' '∘' ¯6 0 6 0
    :endif
    :return
    :endregion
```

This property requires explanations:

- Normally the **whereIc** property is only defined in **zObject**: it is an inherited property
- So when, you call the **whereIc** property, if the system does not find one in **uFrame**, it will try to find one in its parent, then grand-parent, then… until finding one and executes the first one it finds in the object hierarchy
- As **whereIc** is normally only defined in **zObject**, it normally is the **zObject whereIc** property which is run: this is where the whole **whereIc** logic lies
- If you create a **whereIc** property in **uFrame**, it will be this **whereIc** property that will be run instead
- But, in our case, when it runs, we want to call the **whereIc** property defined in **zObject** and run it a second time to adjust the uFrame position and size

Note:
This is very advanced and a little complex.

However, the important thing to remember here, is that you can run a property defined in a parent object of your object hierarchy by using the **base** property: its arguments must be:

• the name of the class in which you want to run the property or method
• the name of the property or method and its arguments

Example:

```
←⍞wi'base' 'zObject' 'whereIc' '∘' '∘' '∘' '∘' ¯6 0 6 0
```

In C#, one would write:

```
base.whereIc(a,b,c,d,e,f,g,h);
```

Note that an alternative coding (not as object oriented) would be to write things as follows:

Copyright © 2016 Eric Lescasse

```
⎕warg←'wherelc' '⍥' '⍥' '⍥' '⍥' ¯6 0 6 0
zObject'Action'
```

Reminder: '⍥' means: leave this element unchanged
Reminder: the last 4 elements in wherelc are adjustments to the top, left, height and width

So 'wherelc' '⍥' '⍥' '⍥' '⍥' ¯6 0 6 0 means: leave the object top, left, height and width unchanged, then retrieve 6 pixels from the top position and add 6 pixels to the object height.

Now, when you run:

```
←'ff'⎕wi'*.fr.Create' 'uFrame'('wherelc'0 0'>' '>')('anchor' 'lrtb')
```

it is the **uFrame wherelc** property which is run and you get the desired result.

Let's check it:

```
←'ff'⎕wi'*Create' 'zForm'('DemoShow'200 300 1 1)('*color'255)
←'ff'⎕wi'*.fr.Create' 'uFrame'('wherelc'0 0'>' '>')('anchor' 'lrtb')
```

And you now have an object of yours (**uFrame**) that inherits from **zFrame**, i.e. supports **wherelc**, **anchor**, etc. but behaves as you prefer it to behave.

Example overriding properties in an existing zObject

The **zList zObject** has a **value** property which returns a 2-element nested vector: the selected index and the selected text.

Let's assume you do not like that and would prefer the **value** property returning only the selected text.

Copyright © 2016 Eric Lescasse

You do that by creating a **uList** object that derives from **zList**:

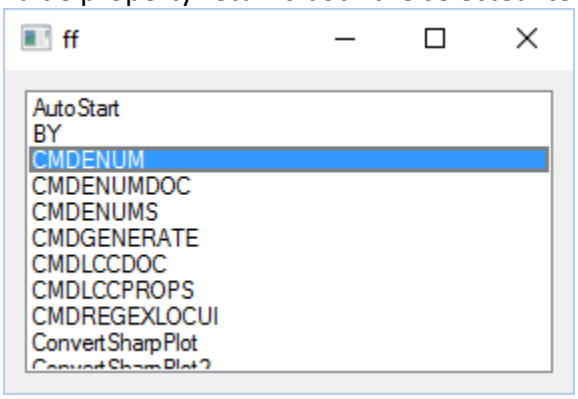
```
      'zz'⎕wi'Derive' 'uList' 'zList' 'zAPLControlTemplate'
Class <uList> successfully created and registered!
```

All you need to do then is to add a **value** property in **uList,** that only returns the selected text.

First, let's use the **zList** object:

```
      ←'ff'⎕wi'*Create' 'zForm'('DemoShow'200 300 1 1)
      ←'ff'⎕wi'*.ls.Create' 'zList'('wherelc'0 0'>' '>')('anchor' 'lrtb')
      ←'ls'⎕wi'*list'(⎕cn⎕nl 3)
      ←'ls'⎕wi'value'3
      'ls'⎕wi'value'
 3 CMDENUM
```

We can set the value property by specifying the origin-1 index of the item to select and the value property returns both the selected item index and text.

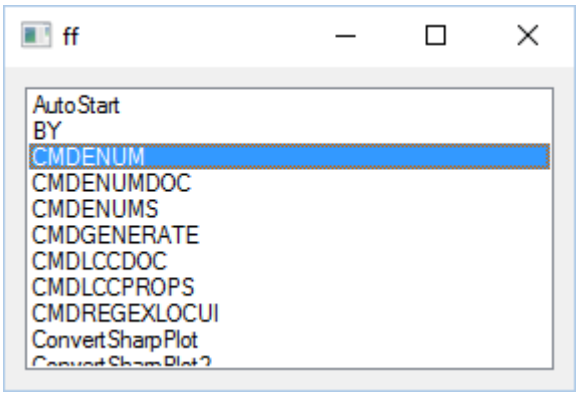


Now let's use the **uList** object instead:

```
      ←'ff'⎕wi'*Create' 'zForm'('DemoShow'200 300 1 1)
      ←'ff'⎕wi'*.ls.Create' 'uList'('wherelc'0 0'>' '>')('anchor' 'lrtb')
      ←'ls'⎕wi'*list'(⎕cn⎕nl 3)
      ←'ls'⎕wi'value' 'CMDENUM'
      'ls'⎕wi'value'
CMDENUM
```

The user interface is the exact same UI as before:

Copyright © 2016 Eric Lescasse



but this time, the **value** property must be set by specifying the text of the item to select and it returns the text of the selected item as well.

You have created a **uList** object that inherits from **zList**, i.e. an object that can use **wherelc**, **anchor** and any property or method defined in **zList**, but which behaves differently as far as the **value** property is concerned.

Note:
Reminder: a **zList** or **uList** object is based on an APL+Win **List** object and an APL+Win **List** object also has a **value** property.

To call the **value** property of the underlying APL+Win object which is at the top of the hierarchy, you must prefix it with a *.

Example:

```
      ←'ff'□wi'*Create' 'zForm'('DemoShow'200 300 1 1)
      ←'ff'□wi'*.ls.Create' 'uList'('wherelc'0 0'>' '>')('anchor' 'lrtb')
      ←'ls'□wi'*list'(□cn□nl 3)
      ←'ls'□wi'value' 'CMDENUM'
      'ls'□wi'value'
CMDENUM

but:

      'ls'□wi'*value'
3
```

Example overriding events in an existing zObject

Copyright © 2016 Eric Lescasse

Object Oriented Development with zObjects

Introduction

This chapter teaches mostly everything that needs to be known if you want to develop your APL+Win applications with your own objects using the zObjects technology.

The various parts of an Object

Overal structure of an object

The top code

The Constructor

Passing data to a Constructor
Grouping events subscriptions at the end of the Constructor

The help and class properties

Regions

Properties

Methods

Events

The bottom code

Creating a new Object

The various types of objects

The Derive method

How objects are registered

z-objects vs non-z objects
in zzInit
in zzCustomClasses

zzAddLink and links

Copyright © 2016 Eric Lescasse

Creating a Property

Code structure for a Property

User Defined Properties

The Property argument (□warg)

The property result (□wres)

Property documentation

Creating a Method

Code structure for a Method

The Method arguments (□warg)

The Method results (□wres)

Methods documentation

Events

Subscribing to Events

Developing Event Handlers

The AddHandler method

Events documentation

Object Orientation in zObjects

Encapsulation

Inheritance

Where is inheritance defined?
How inheritance works
The zObject object
    Useful zObject Properties and Methods
The * prefix
Did you intend to use *XXXXXX in YYYYYY?

Copyright © 2016 Eric Lescasse

Polymorphism

Multicast Event Handlers

Overriding properties and methods

Calling properties or methods in the base class

Missing Features

Finalizers
Partial Types
Virtual Function Members
Abstract Classes and Abstract Members
Sealed Classes
Access Modifiers
Interfaces
Enums
Delegates
Anonymous Methods

Getting Help in Depth

Reviewing important zObject Properties and Methods
Allprops, props, allsyntax, hevents, hmethods, hproperties
Inheritance, info, mainprops, isnetclass, mainsyntax, methods,
cetclasses, classinheritance, properties, objects, zobjects,
Doc, ExtractCode, GetHelp, GetParents

The zz functions

zzInit
zzReset
The zz sample functions
The zz utility functions

Copyright © 2016 Eric Lescasse

zObjects User Commands

]enum

]enumdoc

]enums

]lccdoc

]lccprops

Copyright © 2016 Eric Lescasse

Index

Copyright © 2016 Eric Lescasse

Copyright © 2016 Eric Lescasse

Appendix 1

Copyright © 2016 Eric Lescasse

Notes

Copyright © 2016 Eric Lescasse

References

Glossary

Copyright © 2016 Eric Lescasse