Using OODialog without the Resource Workshop
Introduction

- Object Rexx Windows Gui Manager
Introduction

- Object Rexx Windows Gui Manager
- Came with an IDE Application
Introduction

- Object Rexx Windows Gui Manager
- Came with an IDE Application
- IDE did not come across to ooRexx
Introduction

- Object Rexx Windows Gui Manager
- Came with an IDE Application
- IDE did not come across to ooRexx
- Object Rexx developer since 2001
Introduction

- Object Rexx Windows Gui Manager
- Came with an IDE Application
- IDE did not come across to ooRexx
- Object Rexx developer since 2001
- Designed 80+ dialogs using ooDialog
Introduction

- Object Rexx Windows Gui Manager
- Came with an IDE Application
- IDE did not come across to ooRexx
- Object Rexx developer since 2001
- Designed 80+ dialogs using ooDialog
- No expert on ooDialog code or history
Introduction

OO Dialog

- Object Rexx Windows Gui Manager
- Came with an IDE Application
- IDE did not come across to ooRexx
- Object Rexx developer since 2001
- Designed 80+ dialogs using ooDialog
- No expert on ooDialog code or history
- Stopped using Resource Workshop

Me
The development cycle with The Resource Workshop
Resource Workshop development cycle

- Design Windows GUI Object
Resource Workshop development cycle

- Design Windows GUI Object
- Run OODialog Template Generator
Resource Workshop development cycle

- Design Windows GUI Object
- Run OODialog Template Generator
- Generator creates Code Template
Resource Workshop development cycle

- Design Windows GUI Object
- Run OODialog Template Generator
- Generator creates Code Template
- You amend Code to add functionality
Resource Workshop development cycle

- Design Windows GUI Object
- Run OODialog Template Generator
- Generator creates Code Template
- You amend Code to add functionality
- Now you can't amend the GUI!
Problems with the Resource Workshop way of developing OODialog GUls
Resource Workshop problems

- Hard to amend GUI
Resource Workshop problems

- Hard to amend GUI
- Difficult to debug when goes wrong
Resource Workshop problems

- Hard to amend GUI
- Difficult to debug when goes wrong
- Never get to understand how OODialog classes work
Resource Workshop problems

- Hard to amend GUI
- Difficult to debug when goes wrong
- Never get to understand how OODialog classes work
- It's not there in ooRexx!
Resource Workshop problems

- Hard to amend GUI
- Difficult to debug when goes wrong
- Never get to understand how OODialog classes work
- It's not there in ooRexx!
- **Solution** – Create a Static template
Resource Workshop problems

- Hard to amend GUI
- Difficult to debug when goes wrong
- Never get to understand how OODialog classes work
- It's not there in ooRexx!

**Solution** – Create a Static template
- Template for subclassing UserDialog class covers most bases
What does a UserDialog Subclass look like?
Anatomy of a UserDialog Subclass

- Init method run when you create a new instance of an ooRexx object
Anatomy of a UserDialog Subclass

- Init method run when you create a new instance of an ooRexx object
- `self~Init:super` runs UserDialog Init Method

Could equally well be `forward class (Super) continue`
Anatomy of a UserDialog Subclass

Init

- Init method run when you create a new instance of an ooRexx object
- `self~Init:super` runs UserDialog Init Method
- `Self~Create` or `~CreateCenter` creates the Windows Object
The ooRexx & Windows Objects

- OoRexx Object exists before & after Windows Object
The ooRexx & Windows Objects

- OoRexx Object exists before & after Windows Object
- `self~GetData` & `~SetData` transfer data between them
The ooRexx & Windows Objects

- OoRexx Object exists before & after Windows Object
- `self~GetData` & `~SetData` transfer data between them
- Possible to interact with Windows object Directly

![Image of ooRexx and Windows objects interaction](image.png)
Anatomy of a UserDialog Subclass

- **Init**
  - Init method run when you create a new instance of an ooRexx object
  - `self~Init:super` runs UserDialog Init Method
  - `Self~Create` or `~CreateCenter` creates the Windows Object
  - Establish Connections in init method
OODialog Connections

- Two types of Connections:
Two types of Connections:

- **Data Connection**
OODialog Connections

- **Data Connection**
- **Event Notify Connection**
OODialog Connections

- Two types of Connections:
  - Data Connection
  - Event Notify Connection
- Many are added implicitly
Anatomy of a UserDialog Subclass

- Where we add controls to our Dialog
Anatomy of a UserDialog Subclass

- Where we add controls to our Dialog
- Called Automatically by ~Create
Anatomy of a UserDialog Subclass

- Where we add controls to our Dialog
- Called Automatically by ~Create
- Use Add... Methods
Anatomy of Add... Methods

- self~AddControl(id, x, y, cx, cy, text, Options, MsgToRaise, Attribute)
- Not all parameters required for each type of control
Anatomy of Add... Methods

- `self~AddControl(id, x, y, cx, cy, text, Options, MsgToRaise, Attribute)`
- What sort of a widget ie:
  - Button
  - ListControl
  - ComboBox
  - Tree Control
  - EntryLine
  - Static Text
  - Frame etc...
Anatomy of Add... Methods

- `self~AddControl(id, x, y, cx, cy, text, Options, MsgToRaise, Attribute)`
- Unique Identifier for this control
- Nos 1 – 9 reserved
  - 1 for OK Button
  - 2 for Cancel Button
  - 9 for Help Button
- IDs may be symbolic

Init
DefineDialog
| AddMethods

Jon Wolfers - International Rexx Symposium - Austin - April 2006
Anatomy of Add... Methods

- `self~AddControl(id, x, y, cx, cy, text, Options, MsgToRaise, Attribute)

- Coordinates of control relative to dialog
- Measured in 'Dialog Units'
- Dialog Units depend on screen resolution & Active fonts
- A runtime conversion to pixels may be made using UserDialog attributes FactorX & FactorY
Anatomy of Add... Methods

- `self~AddControl(id, x, y, cx, cy, text, Options, MsgToRaise, Attribute)`
- Control Width & Height in Dialog Units
Anatomy of Add... Methods

- self~AddControl(id, x, y, cx, cy, text, Options, MsgToRaise, Attribute)

- Title associated with Control where appropriate
  - Button Text
  - Initial value of an entry line
  - Static Text
  - Title of a Radio Button/Checkbox...
Anatomy of Add... Methods

- `self~AddControl(id, x, y, cx, cy, text, Options, MsgToRaise, Attribute)`

- Controls style or behaviour
  - Make Pushbutton Default
  - Size of Icons in a list
  - etc..
Anatomy of Add... Methods

- self~AddControl(id, x, y, cx, cy, text, Options, MsgToRaise, Attribute)
- Creates a Notify-Connection for Buttons
Anatomy of Add... Methods

- self~AddControl(id,x,y,cx,cy,text,Options,MsgToRaise,Attribute)
- Initiates connection to (& creates) an attribute for Control Title or Selected element
Anatomy of a UserDialog Subclass

- Where we add controls to our Dialog
- Called Automatically by ~Create
- Use Add... Methods
Anatomy of a UserDialog Subclass

- Where we add controls to our Dialog
- Called Automatically by ~Create
- Use Add... Methods
- Realisation of a Dialog is facilitated Using My DlgArea Class
Anatomy of a UserDialog Subclass

- Run After windows object Created but before it is populated or displayed
Anatomy of a UserDialog Subclass

- Run After windows object Created but before it is populated or displayed
- Start with `self~InitDialog:Super`
Anatomy of a UserDialog Subclass

- Run After windows object Created but before it is populated or displayed
- Start with `self~InitDialog:Super`
- Populate Combo Boxes and lists, add columns to reports
Anatomy of a UserDialog Subclass

- Run After windows object Created but before it is populated or displayed
- Start with `self~InitDialog:Super`
- Populate Combo Boxes and lists, add columns to reports
- A SetData is run after this method
Anatomy of a UserDialog Subclass

- 'Run' method runs after SetData (Not Asynchronous Dialogs)
Anatomy of a UserDialog Subclass

- 'Run' method runs after SetData (Not Asynchronous Dialogs)
- Dialog is shown but not yet active

Init
DefineDialog
InitDialog
Run
Anatomy of a UserDialog Subclass

- 'Run' method runs after SetData (Not Asynchronous Dialogs)
- Dialog is shown but not yet active
- Must contain `self~Run:Super`
Anatomy of a UserDialog Subclass

- 'Run' method runs after SetData (Not Asynchronous Dialogs)
- Dialog is shown but not yet active
- Must contain `self~Run: Super`
- Possible uses:
  - Disable/Enable Buttons
  - Set Initial Values for checkboxes
  - Long initialisation with dialog visible
  - Set Control Colours etc...
Anatomy of a UserDialog Subclass

- One may subclass OK and/or Cancel methods
Anatomy of a UserDialog Subclass

- One may subclass OK and/or Cancel methods
- `self~finished` regulates whether dialog closes
Anatomy of a UserDialog Subclass

- One may subclass OK and/or Cancel methods
- `self~finished` regulates whether dialog closes
- One must call super class
Anatomy of a UserDialog Subclass

- Called by OK Method
Anatomy of a UserDialog Subclass

- Called by OK Method
- Return
  - 0 to prevent dialog closing
  - 1 to allow dialog to close
Anatomy of a UserDialog Subclass

- Your class should be headed

  ::class MyDialog SubClass UserDialog
Anatomy of a UserDialog Subclass

- Your class should be headed
  ::class MyDialog SubClass UserDialog

- You may need to inherit from
  - AdvancedControls
  - MessageExtensions or
  - VirtualKeyCodes
    (requires winSystm.cls)
Anatomy of a UserDialog Subclass

- Your class should be headed
  \[::\text{class MyDialog SubClass UserDialog}\]

- You may need to inherit from
  - AdvancedControls
  - MessageExtensions or
  - VirtualKeyCodes
    (requires winSystm.cls)

- Template gives guidance on inheritance.
Anatomy of a UserDialog Subclass

- $\text{Dlg} = \text{.MyDialog}\text{~new}$

The Class

Init
DefineDialog
InitDialog
Run
OK/Cancel
Validate

The Class Directive

Calling your dialog
Anatomy of a UserDialog Subclass

- \texttt{Dlg = .MyDialog~new}
- Can pass a stem variable to Init

Calling your dialog
Anatomy of a UserDialog Subclass

- `Dlg = .MyDialog~new`
- Can pass a stem variable to Init
- Now we can access attributes i.e.
  - `dlg~UserName='Jon'`
Anatomy of a UserDialog Subclass

- **Dlg = .MyDialog~new**
- Can pass a stem variable to Init
- Now we can access attributes i.e.
  - `dlg~UserName='Jon'`
- `dlg~execute('showtop')`
  (could be ExecuteAsync)
Anatomy of a UserDialog Subclass

- **Dlg = .MyDialog~new**
- Can pass a stem variable to Init
- Now we can access attributes i.e.
  - `dlg~UserName='Jon'
- `dlg~execute('showtop')` (could be ExecuteAsync)
- Now the Dialog runs till OK or Cancel
Anatomy of a UserDialog Subclass

- Dlg = .MyDialog~new
- Can pass a stem variable to Init
- Now we can access attributes i.e.
  - dlg~UserName='Jon'
- dlg~execute('showtop')
  (could be ExecuteAsync)
- Now the Dialog runs till OK or Cancel
- dlg~deInstall

Init
DefineDialog
InitDialog
Run
OK/Cancel
Validate
The Class Directive

Calling your dialog

Jon Wolfers - International Rexx Symposium - Austin - April 2006
Anatomy of a UserDialog Subclass

- Dlg = .MyDialog~new
- Can pass a stem variable to Init
- Now we can access attributes i.e.
  - dlg~UserName='Jon'
- dlg~execute('showtop')
  (could be ExecuteAsync)
- Now the Dialog runs till OK or Cancel
- dlg~deInstall
- OoRexxx object still available!
Anatomy of a UserDialog Subclass

That's all there is to subclassing the OODialog UserDialog class!
Using The Template

A live demo!
Using The Template

- What we want to achieve
Using The Template

DlgAreaU class – coterminous with the dialog
Using The Template

DlgAreaU

DlgArea e

90%

80%

DlgArea created within inner margin of DlgAreaU
Using The Template

DlgAreaU
DlgArea e
DlgArea b

Template
Using The Template

DlgAreaU
DlgArea e
DlgArea b
DlgArea s

Template
The end?

Scripts of interest:

Minimum code to run a dialog
Context Menu (Sort of)
SubClassing GetData & SetData
Resizable Dialog