

Oliver Sims - The ooDialog User Guide

1943-60	Born in Scotland – moved to Wales in 1960
1962-66	Buyer in large department store chain in Australia
1966-69	Met computers at University of Wales – ICL and Algol
1969-93	IBM UK, TSE – CSE. ITSC Austin 1983-85. Met OO through OS2 UI
1993-96	Integrated Object Systems Ltd (IOS) in UK – an IBM Joint Venture to develop OO software – middleware - objects in the large, built with OO or procedural languages (including C, C++, Cobol, RPGII, and classic REXX)
1996-98	Bought out by US company (SSA Inc, which went broke in '98)
1998-99	Bought out by UK company (MSI) – then laid off a year later
1999-00	Genesis Development (great US consultancy based on the East Coast)
2000-01	Bought out by Iona Technologies, but then laid off in the 2001 recession
2001-11	Went independent – Consulting – Sims Associates LLP
2011-	Retired. Yippee!

The ooDialog User Guide

- Objectives:
 - Provide an introduction to ooDialog through sample dialogs in the context of a simple working “Sales Order Management” application
 - Provide an “Infrastructure” that makes things easier for the app developer

Note: Requires ooDialog 4.2.4

Main ooDialog Superclasses

RcDialog



A screenshot of a dialog box titled "*Customer*" with a close button in the top right corner. The dialog contains several input fields and buttons. The fields are: Customer Number (AB0784), Name (ABC Enterprises Inc.), Address (2145 Engle Blvd, Hardtown, FL), Zip Code (37043), Discount Code (B1), and Last Order. At the bottom, there are two buttons: "Record Changes" and "Show Last Order".

ResDialog



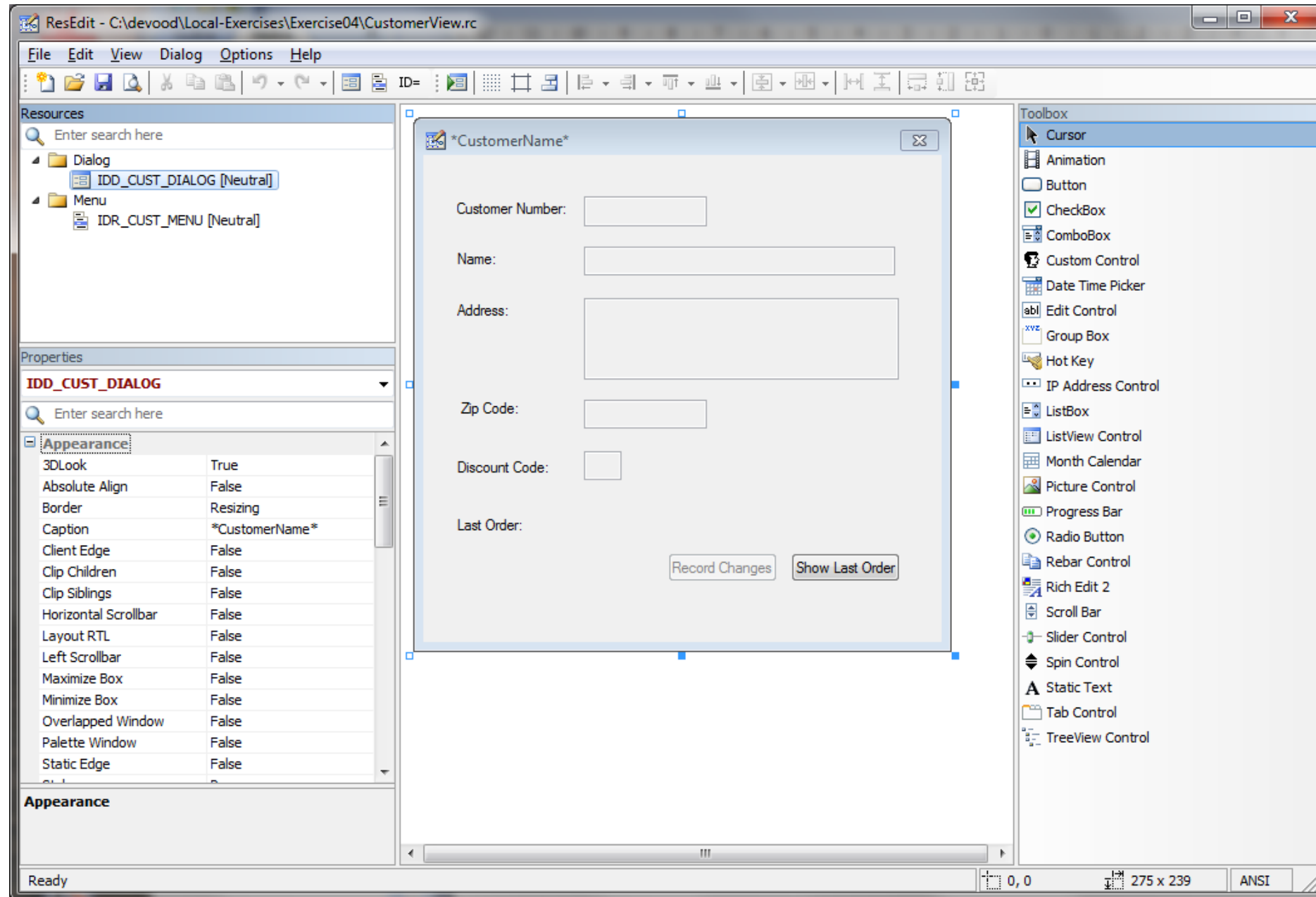
A screenshot of a dialog box titled "Product" with standard window controls in the top right corner. The dialog has a menu bar with "Actions" and "Help". It contains input fields for Product Number (CU003), Product Name (Widget, 5in), List Price (124.99), and UOM (100). There is a text area for Description containing the text "Meets all possible Widget needs, except those not possible." At the bottom, there is a "Size Category" section with radio buttons for Small, Medium (selected), and Large. A "Save Changes" button is located at the bottom right.

UserDialog



A screenshot of a dialog box titled "Sales Order Management" with standard window controls in the top right corner. It has a menu bar with "Orders", "Customers", "Products", "New", and "Help". The main area contains four icons: "Customer List" (two people), "Product List" (tools), "Sales Orders" (cabinet), and "New Order" (document). At the bottom, there are two buttons: "Reset Icons" and "Exit Application".

Dialog Layout Tool (e.g. ResEdit)



The .rc File:

```
#include <windows.h>
#include <commctrl.h>
#include <richedit.h>
#include "CustomerView.h"

LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
IDR_CUST_MENU MENU
{
    POPUP "Actions"
    {
        MENUITEM "New Customer...", IDM_CUST_NEW
        MENUITEM "Update...", IDM_CUST_UPDATE
        MENUITEM "Print...", IDM_CUST_PRINT
        MENUITEM "Last Order", IDM_CUST_LAST_ORDER
    }
}

LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
IDD_CUST_DIALOG DIALOG 0, 0, 275, 239
STYLE DS_3DLOOK | DS_CENTER | DS_SHELLFONT | WS_CAPTION | WS_VISIBLE | WS_POPUP | WS_THICKFRAME | WS_SYSTEMMENU
EXSTYLE WS_EX_WINDOWEDGE
CAPTION "*CustomerName*"
FONT 8, "Microsoft Sans Serif"
{
    LTEXT          "Name:", IDC_CUST_LBL_CUSTNAME, 18, 47, 22, 8, SS_LEFT
    EDITTEXT      IDC_CUST_EDT_CUSTNO, 85, 20, 65, 15, ES_AUTOHSCROLL | ES_READONLY
    LTEXT          "Customer Number:", IDC_CUST_LBL_CUSTNO, 18, 22, 59, 8, SS_LEFT
    EDITTEXT      IDC_CUST_EDT_CUSTNAME, 85, 45, 165, 14, ES_AUTOHSCROLL | ES_READONLY
    LTEXT          "Address:", IDC_CUST_LBL_CUSTADDR, 18, 72, 28, 8, SS_LEFT
    EDITTEXT      IDC_CUST_EDT_CUSTADDR, 85, 70, 167, 40, ES_AUTOHSCROLL | ES_MULTILINE | ES_READONLY
    LTEXT          "Zip Code:", IDC_CUST_LBL_ZIP, 20, 120, 32, 8, SS_LEFT
    EDITTEXT      IDC_CUST_EDT_CUSTZIP, 85, 120, 65, 14, ES_AUTOHSCROLL | ES_READONLY
    LTEXT          "Last Order:", IDC_CUST_LBL_LASTORDER, 18, 177, 36, 8, SS_LEFT
    LTEXT          "    ", IDC_CUST_STC_LASTORDERDETAILS, 88, 177, 145, 8, SS_LEFT
    LTEXT          "Discount Code:", IDC_CUST_LBL_DISCOUNT, 18, 149, 50, 8, SS_LEFT
    EDITTEXT      IDC_CUST_EDT_DISCOUNT, 85, 145, 20, 14, ES_AUTOHSCROLL | ES_READONLY
    LTEXT          "    ", IDC_CUST_STC_ERRORMSG, 18, 215, 8, 8, SS_LEFT
    DEFPUSHBUTTON "Record Changes", IDC_CUST_BTN_RECORDCHANGES, 130, 195, 58, 14, WS_DISABLED
    PUSHBUTTON    "Show Last Order", IDC_CUST_BTN_SHOWLASTORDER, 195, 195, 58, 14
}
}
```

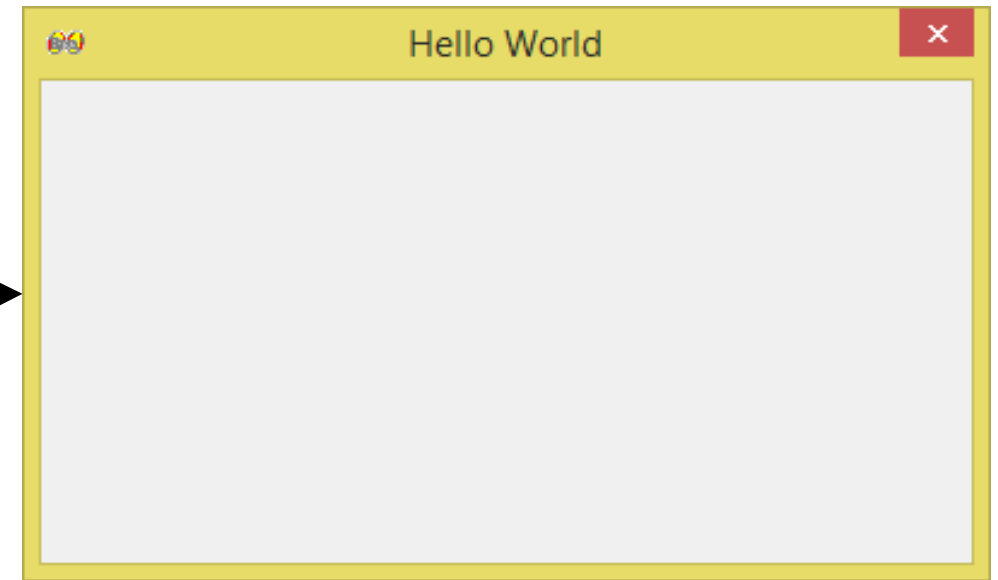
ooDialog Basics

```
dlg = .HelloWorld~new
dlg~execute("SHOWTOP", IDI_DLG_OOREXX)

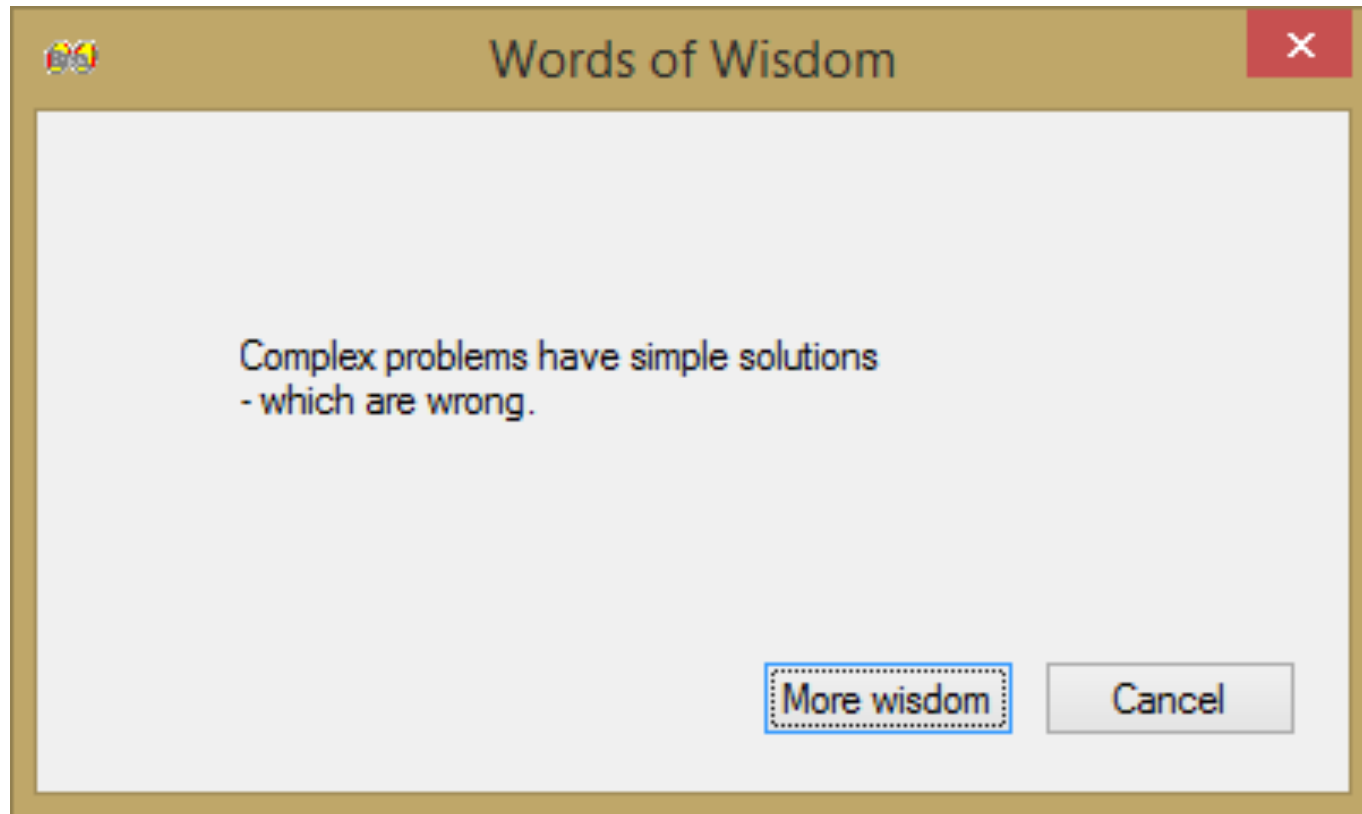
::REQUIRES "ooDialog.cls"

::CLASS 'HelloWorld' SUBCLASS UserDialog

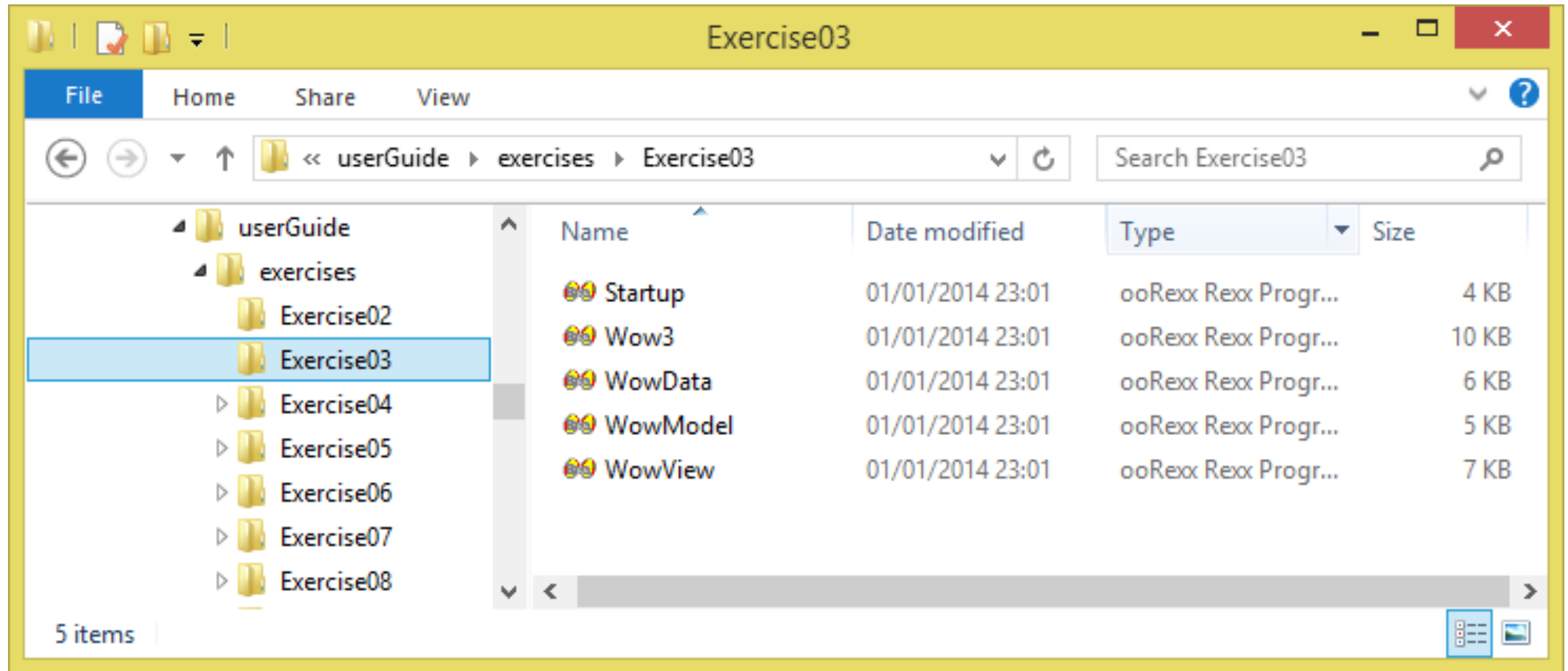
::METHOD init
  forward class (super) continue
  self~create(30, 30, 257, 123, "Hello World", "CENTER")
```



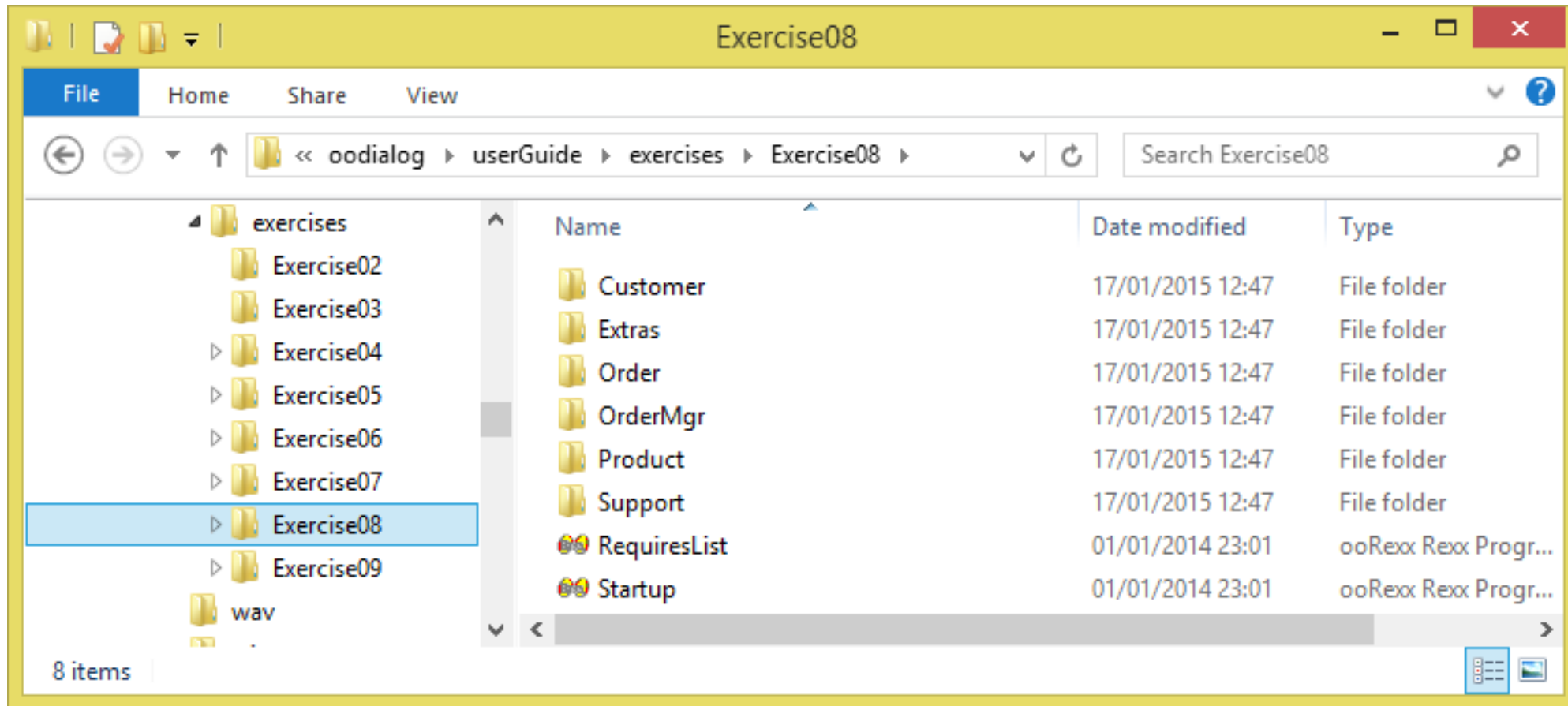
Words of Wisdom



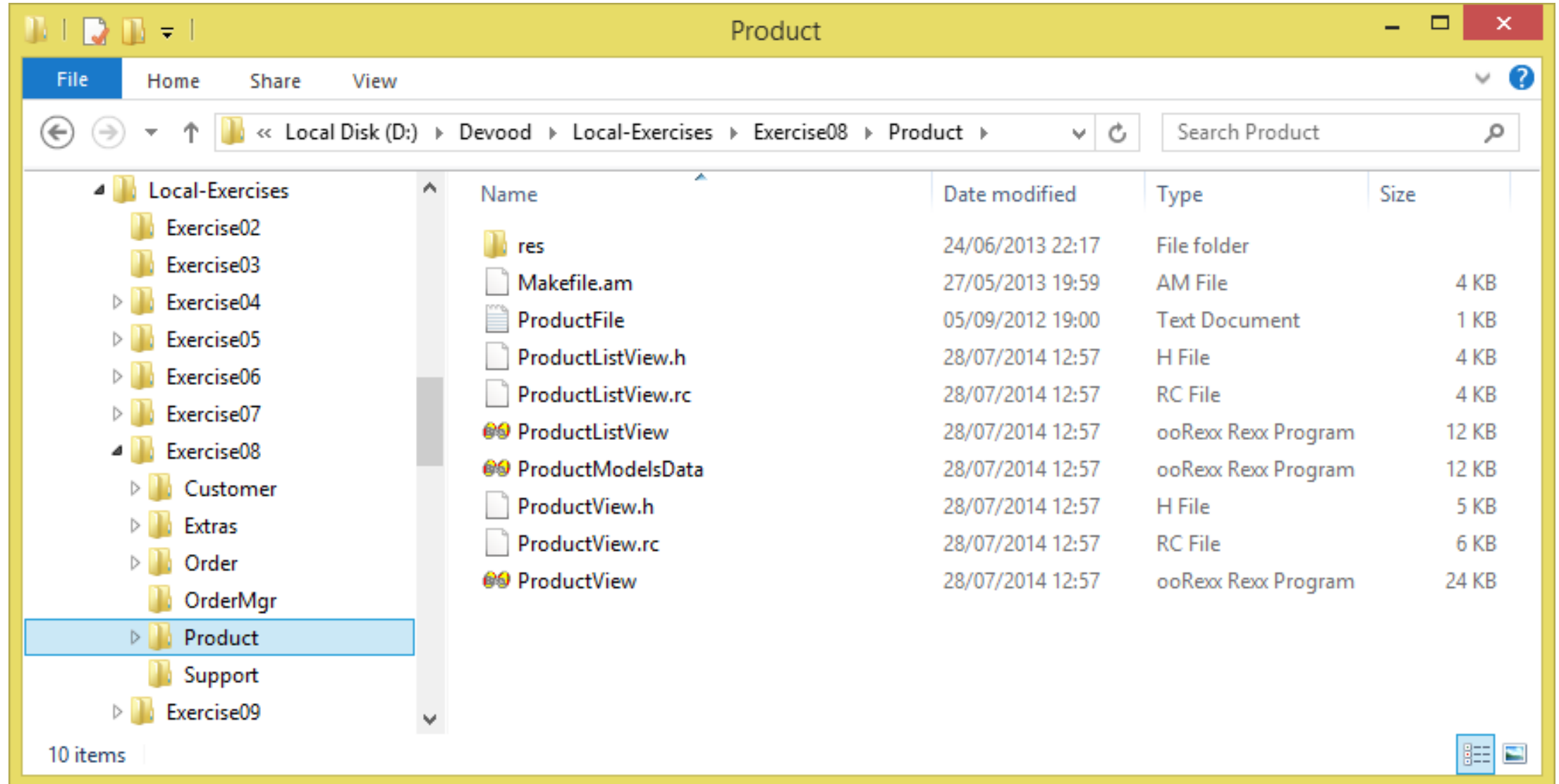
Separation of Concerns



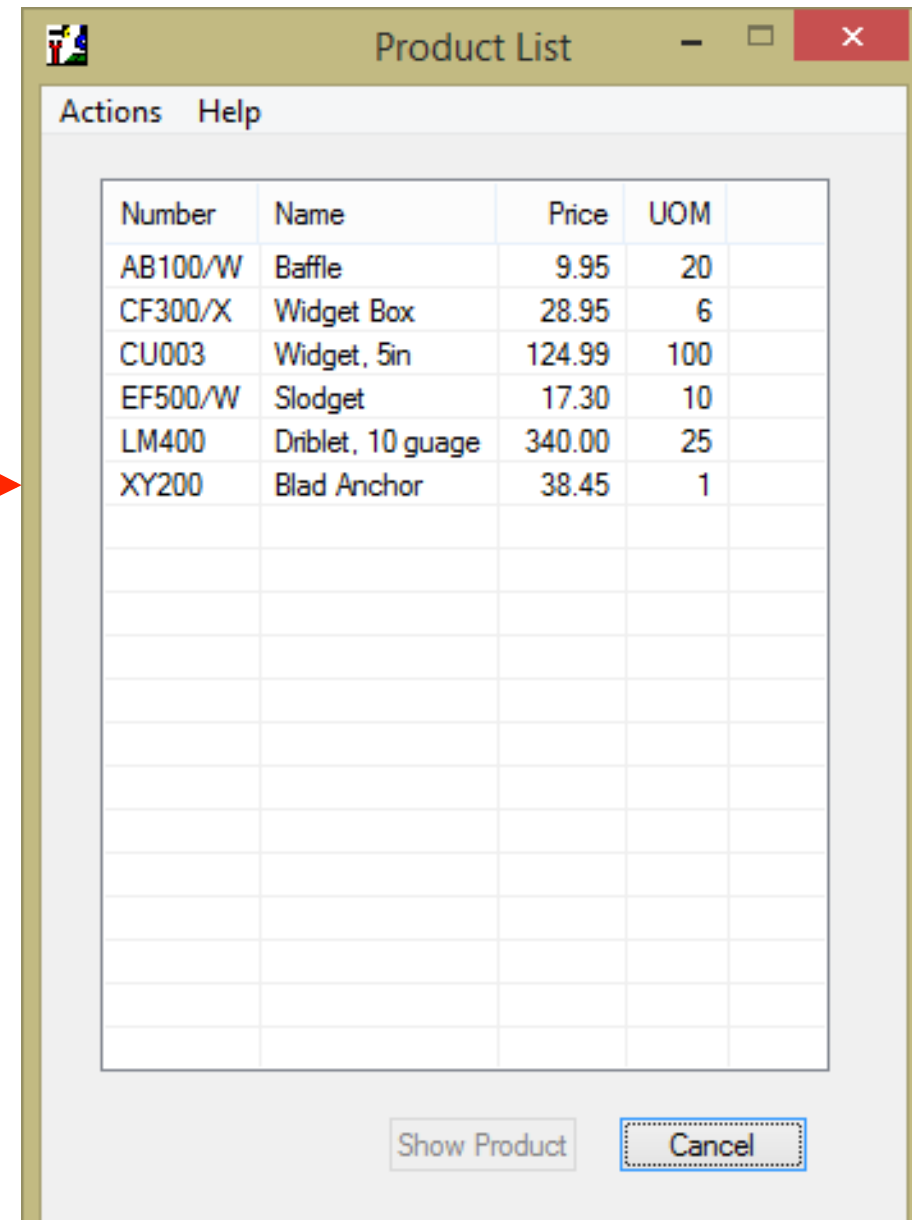
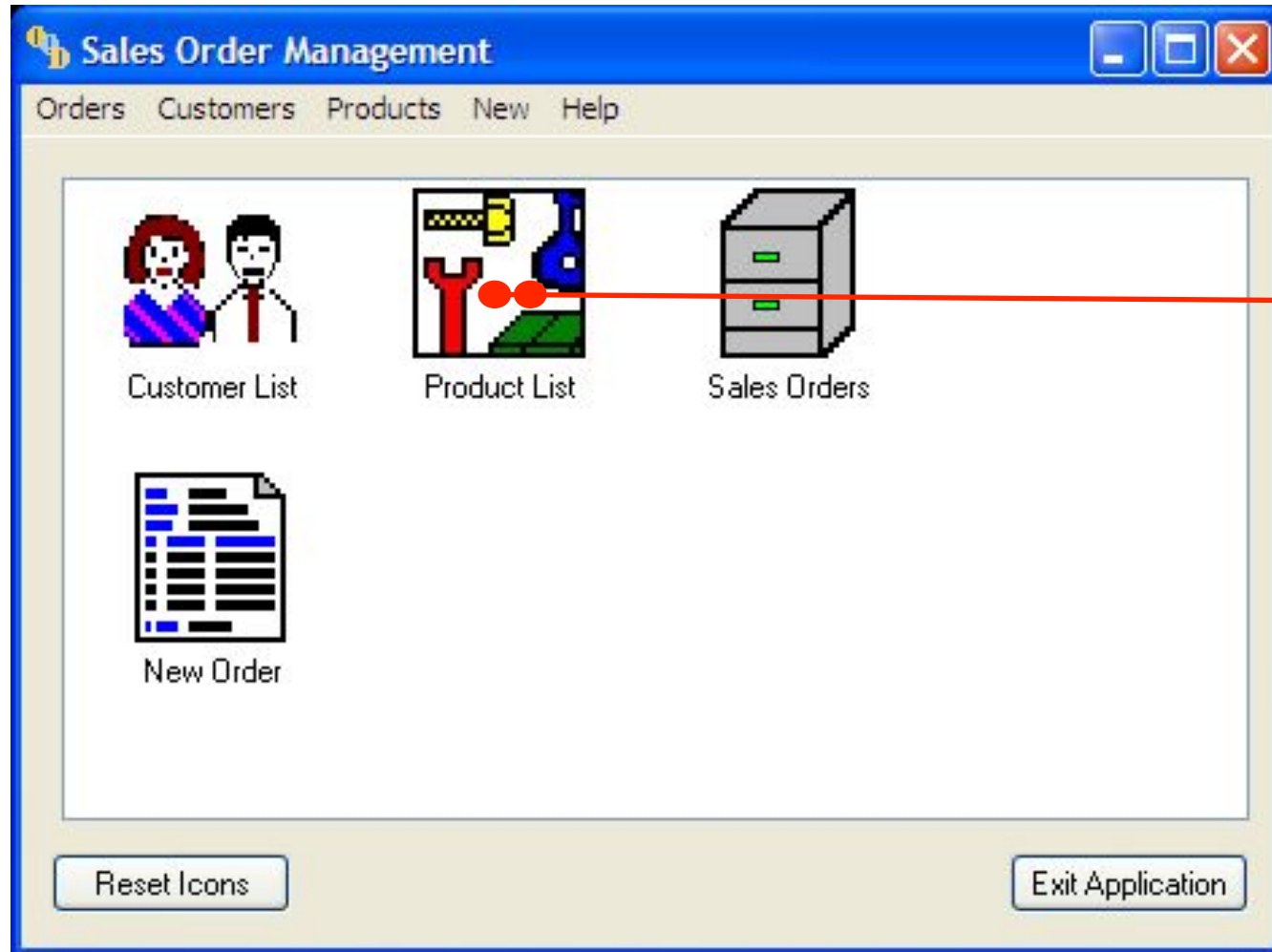
Business Components



The “Product” Business Component



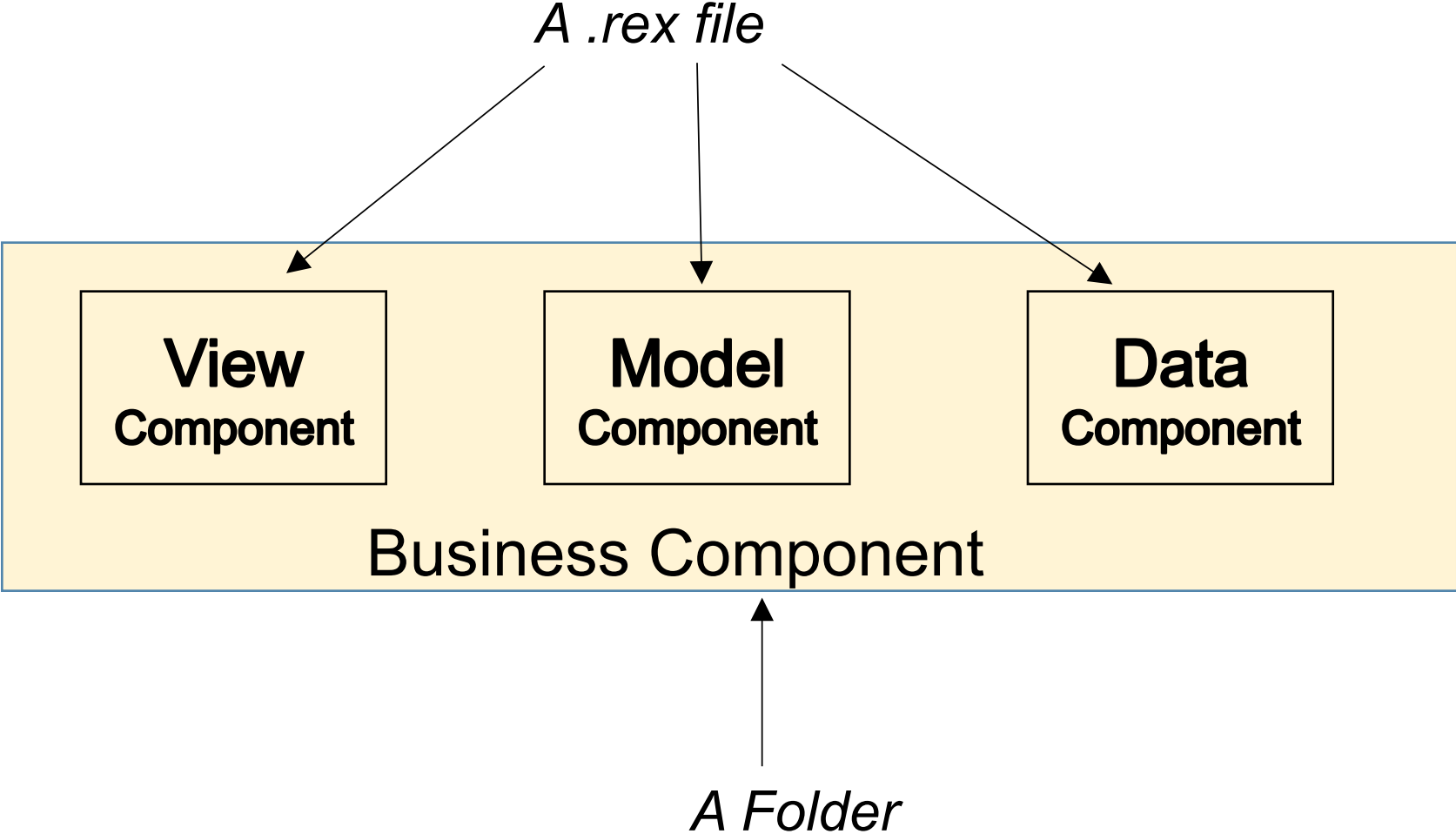
Opening Another Dialog



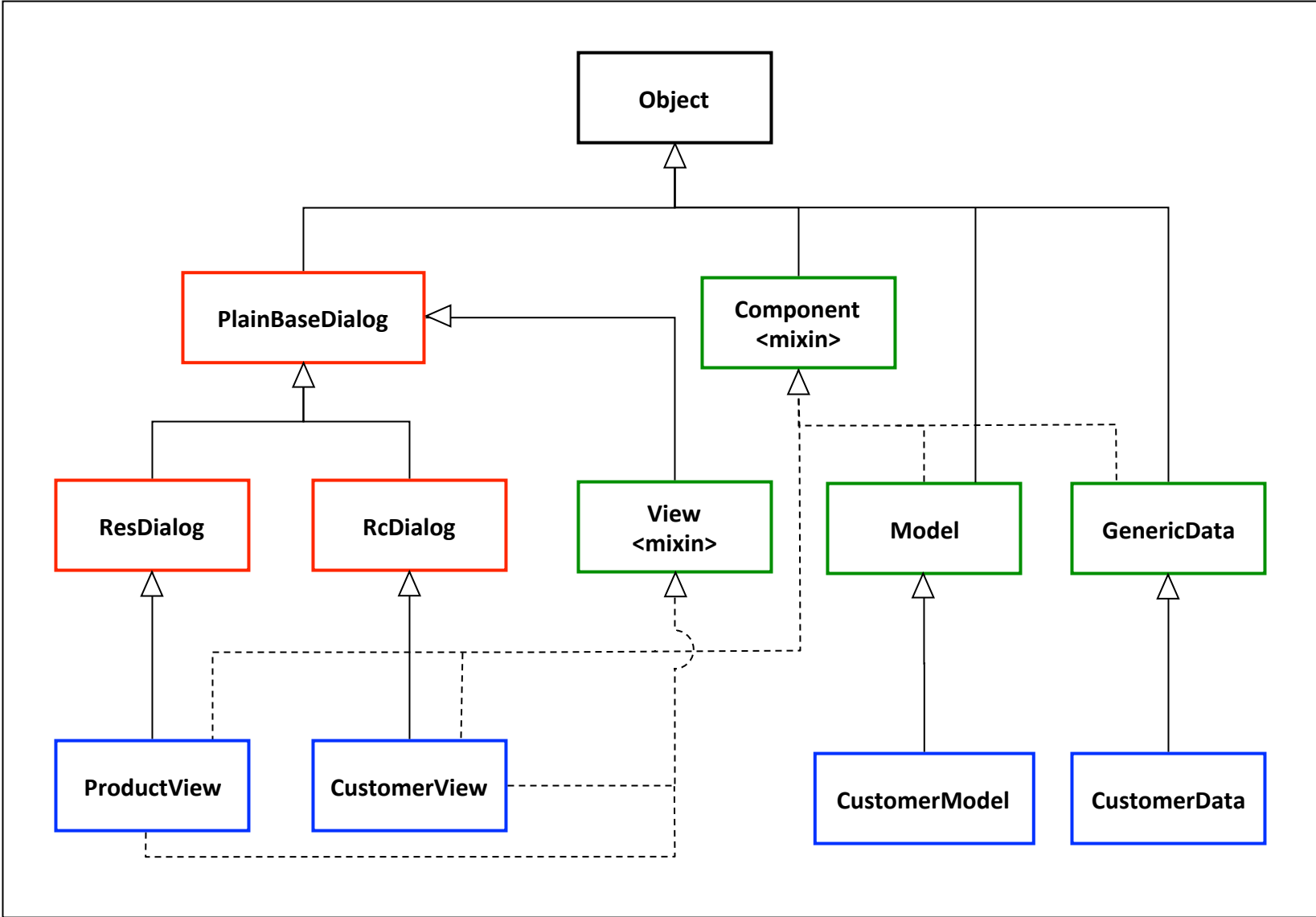
The “Infrastructure”

- The “Component” approach
- Class Hierarchy
- Dialog setup - the “Model-View Framework”
- Class Identity and the “Object Manager”
- Debug Tool - The Message Sender
- Drag/Drop (aka Direct Manipulation)
- Event Management

The Component Approach (e.g. "Customer")



Class Hierarchy



```
Example - ::CLASS CustomerView SUBCLASS RcDialog PUBLIC INHERIT View Component
```

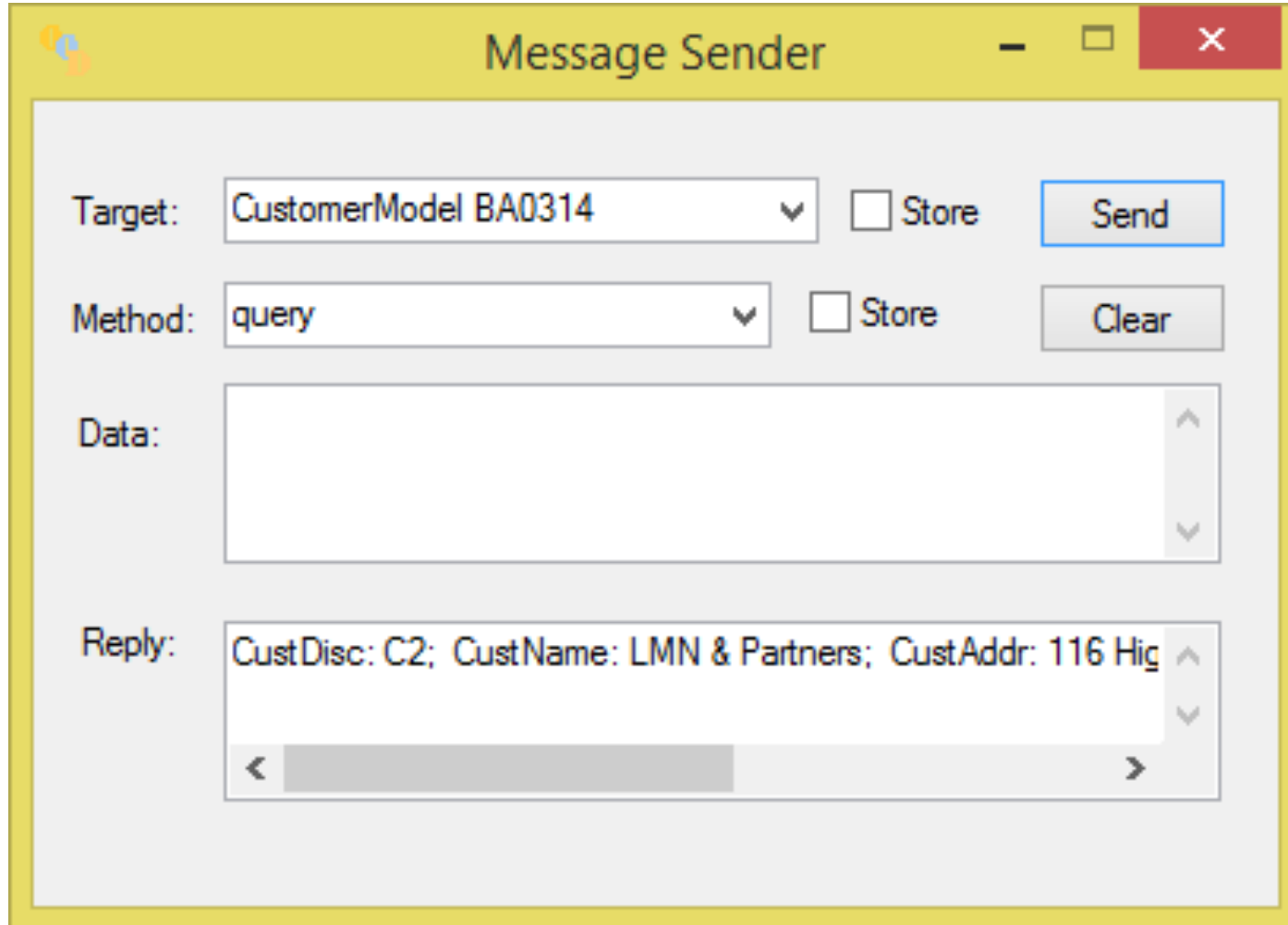
Dialog Setup – The Model-View Framework

- User double-clicks on a Customer with in a Customer List dialog
 - Is the Customer View dialog already instantiated? If so, minimised or just hidden?
 - If not instantiated, is the Customer Model instantiated?
 - If not, is the Customer Data instantiated?
 - Instantiate the required Model and/or Data components in the right order (data first)
 - When it's confirmed that both Data and Model are active, then instantiate the View dialog, which then ...
 - asks the Model for its data so that said data can be displayed on the Customer dialog or view.
- This all handled by a Framework class: - the “Object Manager” (ObjectMgr.rex)
- Example: Surfacing a Customer from a Customer List (excluding error code):

```
::METHOD showCustomer UNGUARDED
  expose lvCustomers
  info = .Directory~new
  -- identify which Customer has been double-clicked (custNum is in info~text)
  if lvCustomers~getItemInfo(item, info) then do
    objectMgr = .local~my.ObjectMgr
    objectMgr~showModel("CustomerModel", info~text, rootDlg)
  end
  else /* error message */
```

- All logic implied by ~showModel is in the “Object Manager” framework class.

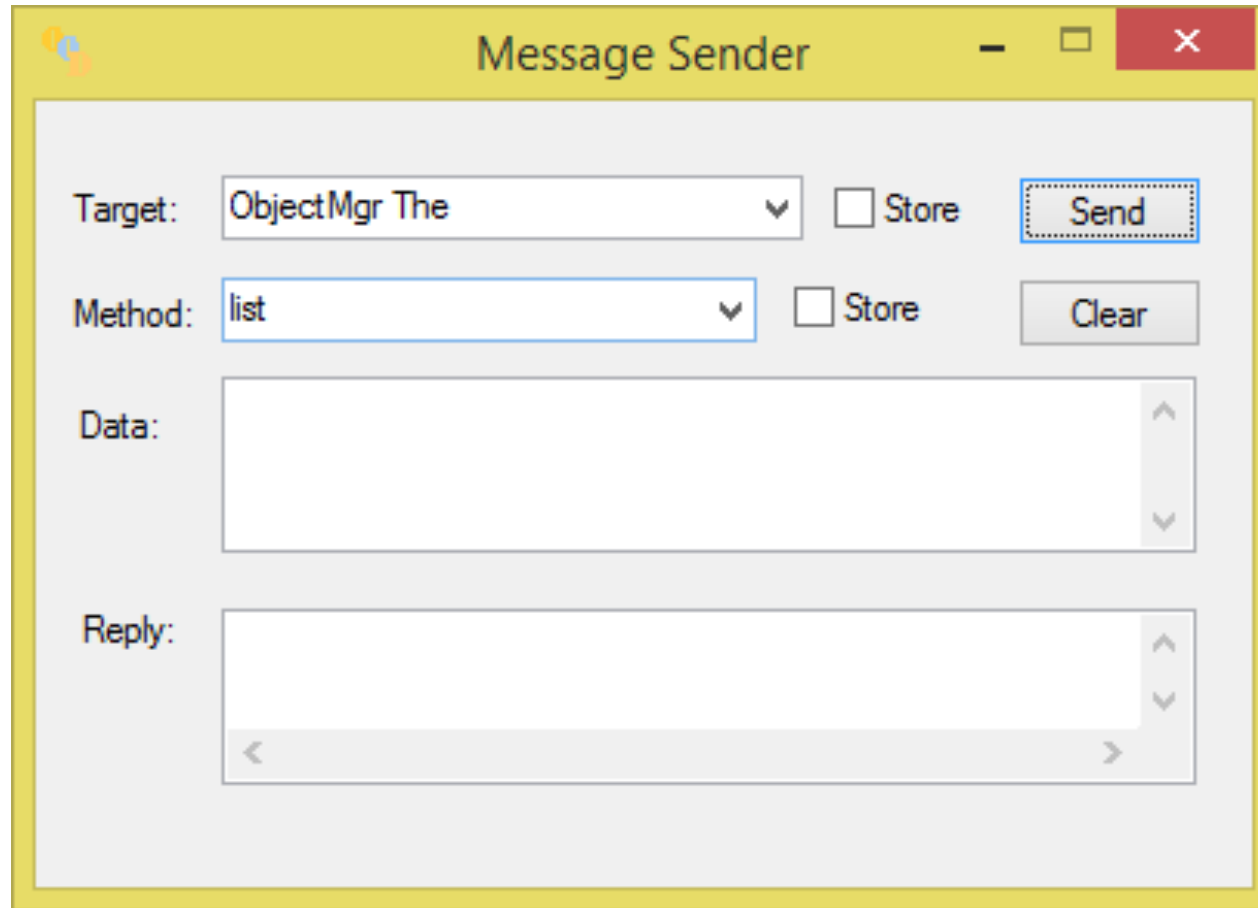
The Message Sender - Querying a Customer Instance



The screenshot shows a window titled "Message Sender" with a yellow title bar. The window contains the following elements:

- Target:** A dropdown menu with "CustomerModel BA0314" selected. To its right is an unchecked checkbox labeled "Store" and a blue "Send" button.
- Method:** A dropdown menu with "query" selected. To its right is an unchecked checkbox labeled "Store" and a grey "Clear" button.
- Data:** An empty text area with a vertical scrollbar on the right side.
- Reply:** A text area containing the text "CustDisc: C2; CustName: LMN & Partners; CustAddr: 116 Hig". It has a vertical scrollbar on the right and a horizontal scrollbar at the bottom.

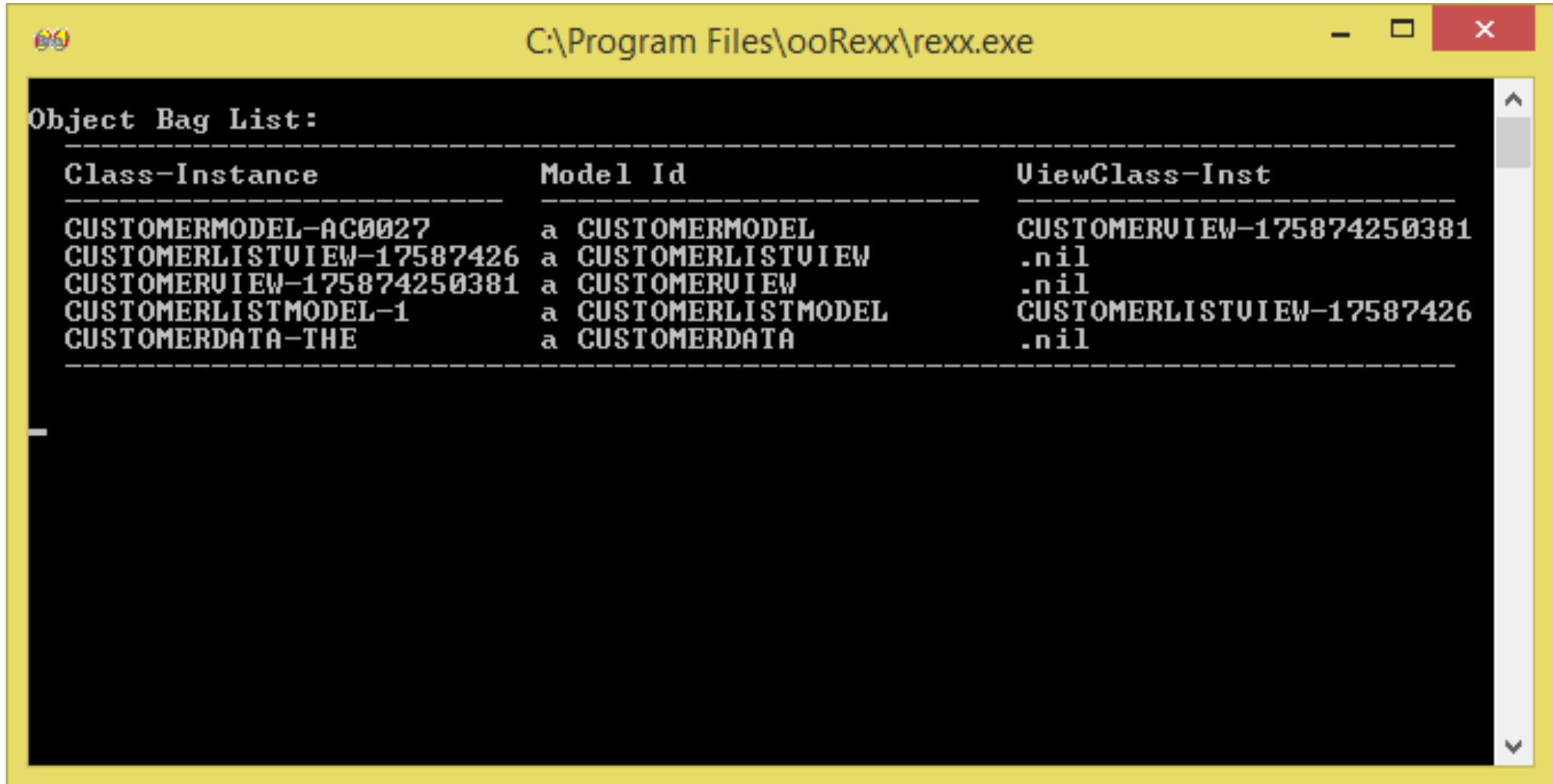
Debugging - The Message Sender – What components exist?



The image shows a screenshot of a software application window titled "Message Sender". The window has a yellow title bar with standard Windows window controls (minimize, maximize, close). The main content area is light gray and contains the following components:

- Target:** A dropdown menu with "ObjectMgr The" selected, followed by an unchecked checkbox labeled "Store" and a "Send" button.
- Method:** A dropdown menu with "list" selected, followed by an unchecked checkbox labeled "Store" and a "Clear" button.
- Data:** A large, empty text area with a vertical scrollbar on the right side.
- Reply:** A large, empty text area with a vertical scrollbar on the right side and a horizontal scrollbar at the bottom.

List of Component Names



The screenshot shows a Windows command prompt window with a yellow title bar. The title bar text is "C:\Program Files\ooRexx\rexx.exe". The window content is a black terminal with white text. The text starts with "Object Bag List:" followed by a dashed line. Below the dashed line is a table with three columns: "Class-Instance", "Model Id", and "ViewClass-Inst". The table contains five rows of data. The first row is "CUSTOMERMODEL-AC0027", "a CUSTOMERMODEL", and "CUSTOMERVIEW-175874250381". The second row is "CUSTOMERLISTVIEW-17587426", "a CUSTOMERLISTVIEW", and ".nil". The third row is "CUSTOMERVIEW-175874250381", "a CUSTOMERVIEW", and ".nil". The fourth row is "CUSTOMERLISTMODEL-1", "a CUSTOMERLISTMODEL", and "CUSTOMERLISTVIEW-17587426". The fifth row is "CUSTOMERDATA-THE", "a CUSTOMERDATA", and ".nil". The table is followed by another dashed line.

```
Object Bag List:
-----
Class-Instance          Model Id                ViewClass-Inst
-----
CUSTOMERMODEL-AC0027   a CUSTOMERMODEL        CUSTOMERVIEW-175874250381
CUSTOMERLISTVIEW-17587426 a CUSTOMERLISTVIEW     .nil
CUSTOMERVIEW-175874250381 a CUSTOMERVIEW         .nil
CUSTOMERLISTMODEL-1   a CUSTOMERLISTMODEL    CUSTOMERLISTVIEW-17587426
CUSTOMERDATA-THE      a CUSTOMERDATA         .nil
-----
```

Debugging - The Message Sender

Message Sender

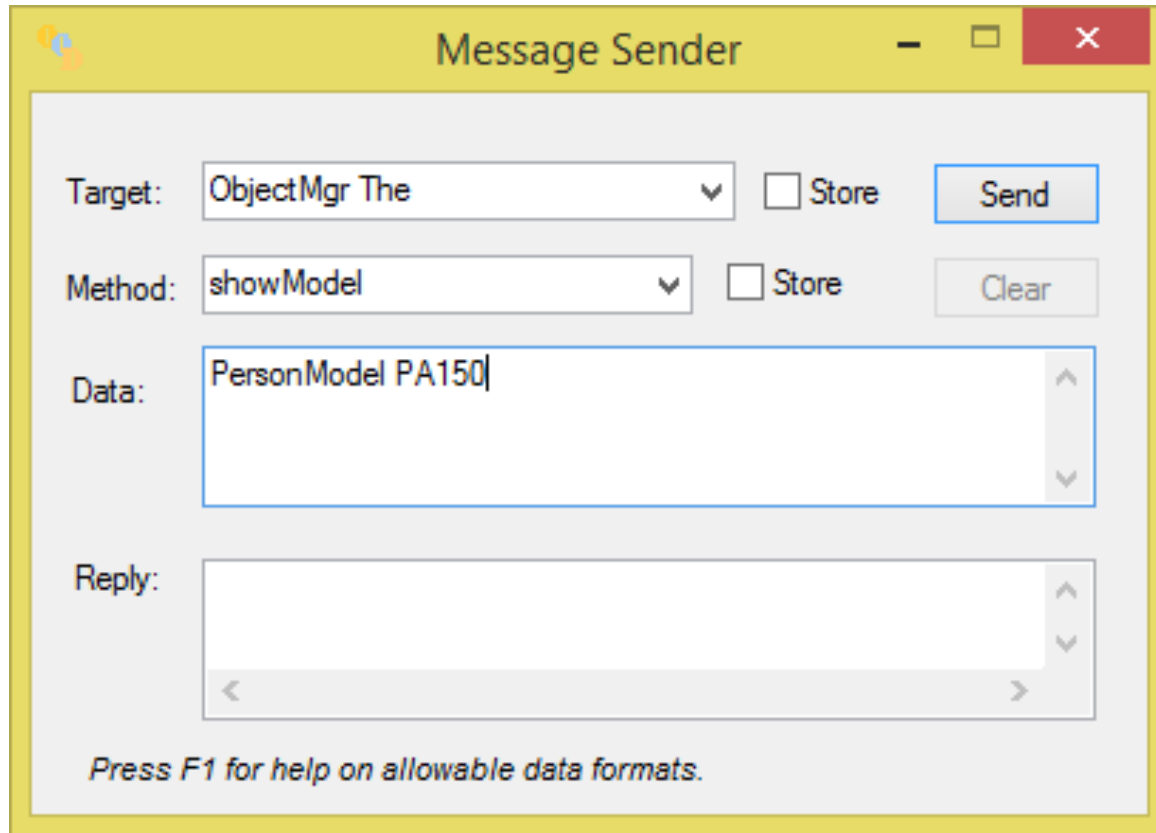
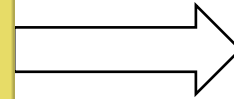
Target: ObjectMgr The Store

Method: showModel Store

Data: PersonModel PA150

Reply:

Press F1 for help on allowable data formats.

A screenshot of a 'Message Sender' dialog box. It has a yellow title bar with standard window controls. The main area is light gray. It contains four rows of controls: 'Target' with a dropdown menu showing 'ObjectMgr The', a 'Store' checkbox, and a 'Send' button; 'Method' with a dropdown menu showing 'showModel', a 'Store' checkbox, and a 'Clear' button; 'Data' with a text area containing 'PersonModel PA150' and vertical scroll arrows; and 'Reply' with an empty text area and vertical scroll arrows. At the bottom, there is a small italicized text prompt: 'Press F1 for help on allowable data formats.'

Person Record

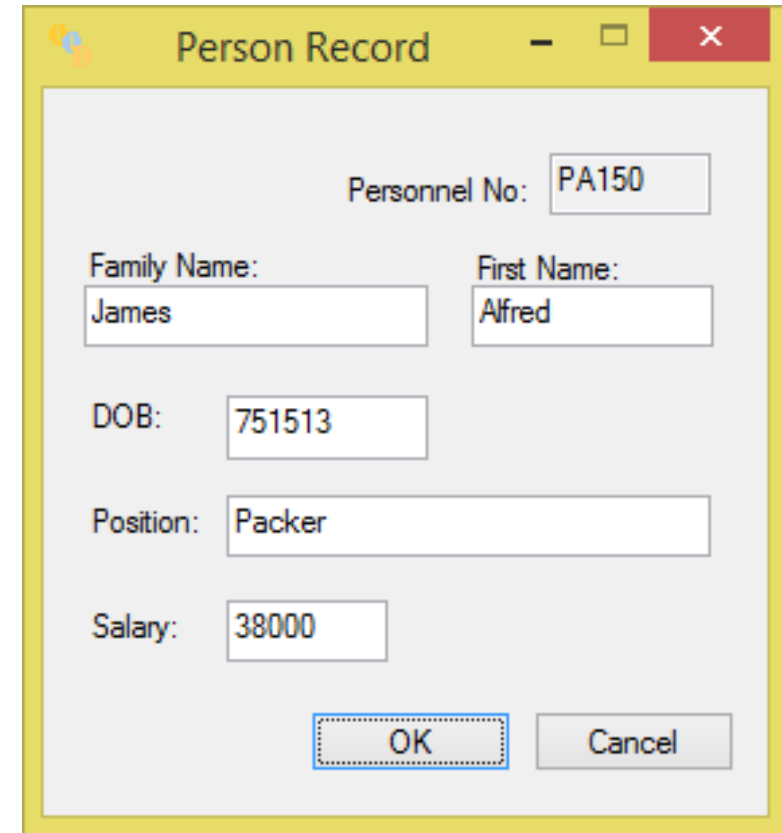
Personnel No: PA150

Family Name: James First Name: Alfred

DOB: 751513

Position: Packer

Salary: 38000

A screenshot of a 'Person Record' dialog box. It has a yellow title bar with standard window controls. The main area is light gray. It contains several rows of controls: 'Personnel No:' with a text field containing 'PA150'; 'Family Name:' with a text field containing 'James' and 'First Name:' with a text field containing 'Alfred'; 'DOB:' with a text field containing '751513'; 'Position:' with a text field containing 'Packer'; 'Salary:' with a text field containing '38000'; and at the bottom, 'OK' and 'Cancel' buttons. The 'OK' button is highlighted with a blue dashed border.

Drag/Drop (aka “Direct Manipulation”)

- Implemented in ooDialog through the **.Mouse** class
- Handled in by the **View** superclass and the **DragManager**
- Phase 1: Setup – in its **initDialog** method, a View tells its superclass that it can be picked up (it’s a “source”) or dropped on (it’s a “target”)
- Phase 2: Drag – a target either will or won’t accept a drop
- Phase 3: Drop - If drop accepted, the target receives a **dmDrop** message with the id of the source component.
- Phase 4 – The Target interacts with the Source (e.g. asks for data)

Drag/Drop code – Setup

Phase 1: Setup -
I'm a drag source

```
::CLASS CustomerView SUBCLASS RcDialog PUBLIC INHERIT View Component
::METHOD initDialog
...
r = self~dmSetAsSource:super("Customer\bmp\Customer.cur")
...
```

Phase 1: Setup -
I'm a drag target

```
::CLASS OrderFormView SUBCLASS RcDialog PUBLIC INHERIT View
Component
::METHOD initDialog
...
self~dmSetAsTarget:super()
...
```

Drag/Drop Operation – e.g. drag Customer to Order Form and Drop

```
::CLASS OrderFormModel SUBCLASS Model PUBLIC
...
::METHOD dmQueryDrop CLASS PUBLIC
  use arg sourceClassName
  if sourceClassName = "CUSTOMERMODEL" then return .true
  if sourceClassName = "PRODUCTMODEL" then return .true
  else return .false
```

Drag-over:
If you're a Product
or Customer, then
OK to drop

```
::CLASS OrderFormView SUBCLASS RcDialog PUBLIC INHERIT View Component
...
::METHOD dmDrop PUBLIC
  use strict arg sourceModel, sourceDlg
  parse var sourceModel . modelName
  ...
```

I've been dropped
on!

Event Management Framework – Example: App Closing

1. A dialog (such as the Order Form) decides that it's interested in the event “AppClosing”:

```
::METHOD activate
```

```
  self~registerInterest("appClosing", self)
```

2. This caught by the Component superclass and forwarded to the EventMgr program.

3. When the Sales Order Manager dialog (the “application”) is closed, its cancel method is:

```
::METHOD cancel
```

```
  self~triggerEvent("appClosing")
```

```
  forward class (super)
```

```
  -- Closes the whole app.
```

4. The triggerEvent method (in the Component superclass) forwards the message to the EventManager, which sends a notify(“appClosing”) message to all interested parties.

5. The OrderFormView component catches the notify event and tidies up the control dialogs:

```
::METHOD notify PUBLIC
```

```
  use strict arg event
```

```
  if event = "appClosing" then do
```

```
    self~closeControlDialogs
```

```
    controlDialogsClosed = .true
```

```
  end
```

To Do

- Write updates to disk (currently read-only)
- Use of ooSQL instead of flat text files
- Drag/drop within a single dialog
- MDI – that is, whole app in single “master” window – (if possible)
- Separate component “class” name from ooRexx class names – use configuration to assigning “external” name to ooRexx class names
- Move all “business” logic from View components to Model components
- Any other suggestions?

The End.