Oliver Sims - The ooDialog User Guide

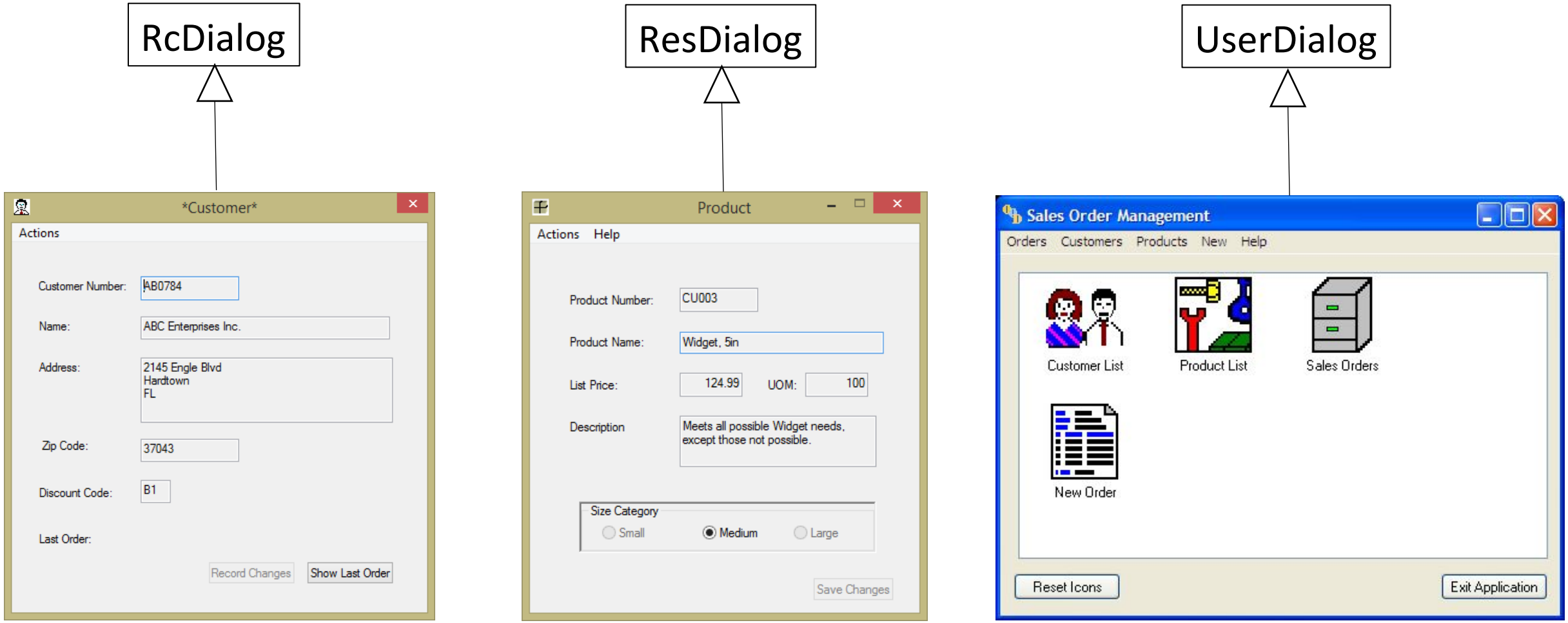
1943-60	Born in Scotland – moved to Wales in 1960
1962-66	Buyer in large department store chain in Australia
1966-69	Met computers at University of Wales – ICL and Algol
1969-93	IBM UK, TSE – CSE. ITSC Austin 1983-85. Met OO through OS2 UI
1993-96	Integrated Object Systems Ltd (IOS) in UK – an IBM Joint 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

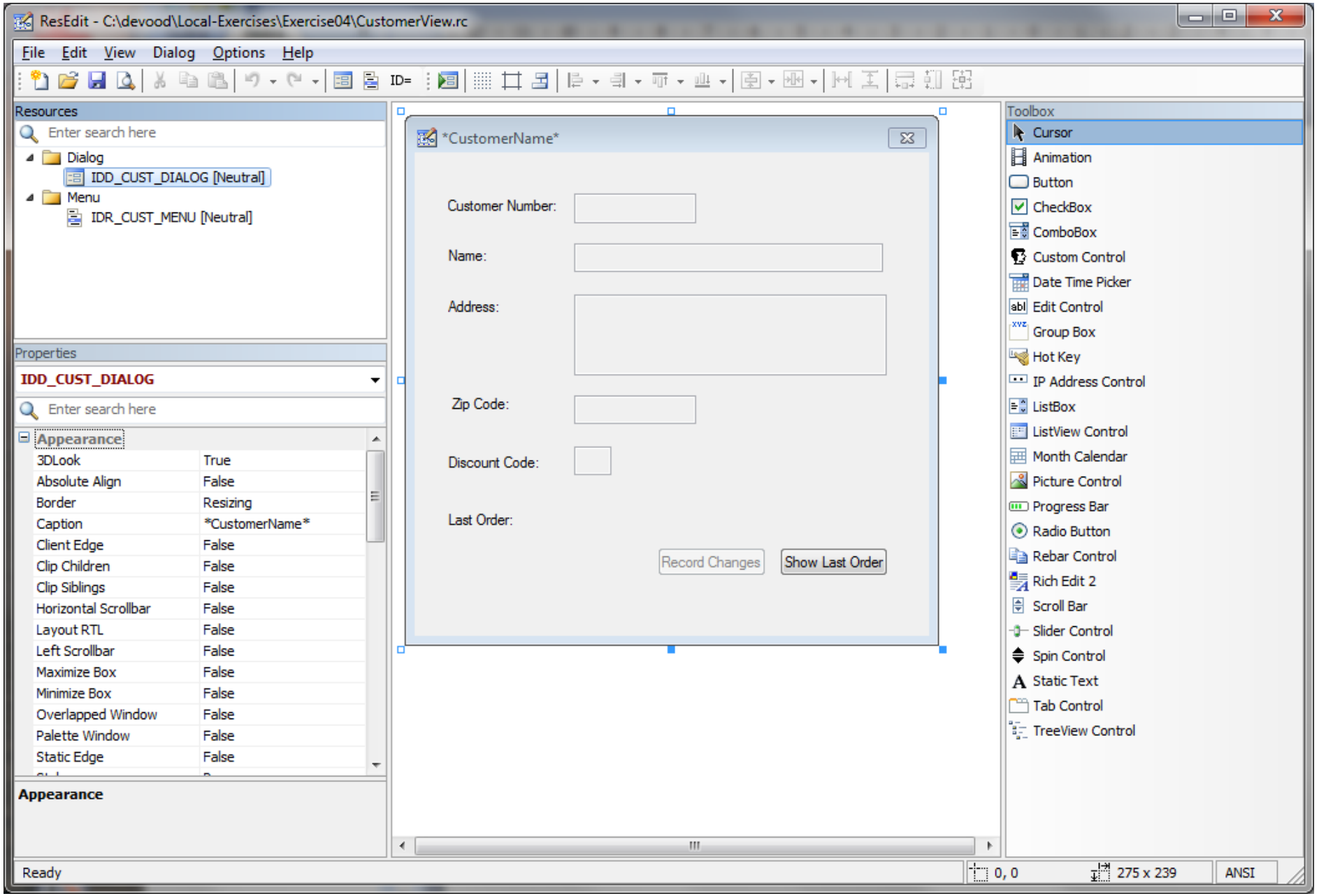
- Objectives:
  - Provide an introduction to ooDialog through sample dialogs in the context of a simple working “Sales Order Management” application
  - Provide an “Infrastructure” that makes things easier for the app developer

Note: Requires ooDialog 4.2.4

Main ooDialog Superclasses



Dialog Layout Tool (e.g. ResEdit)

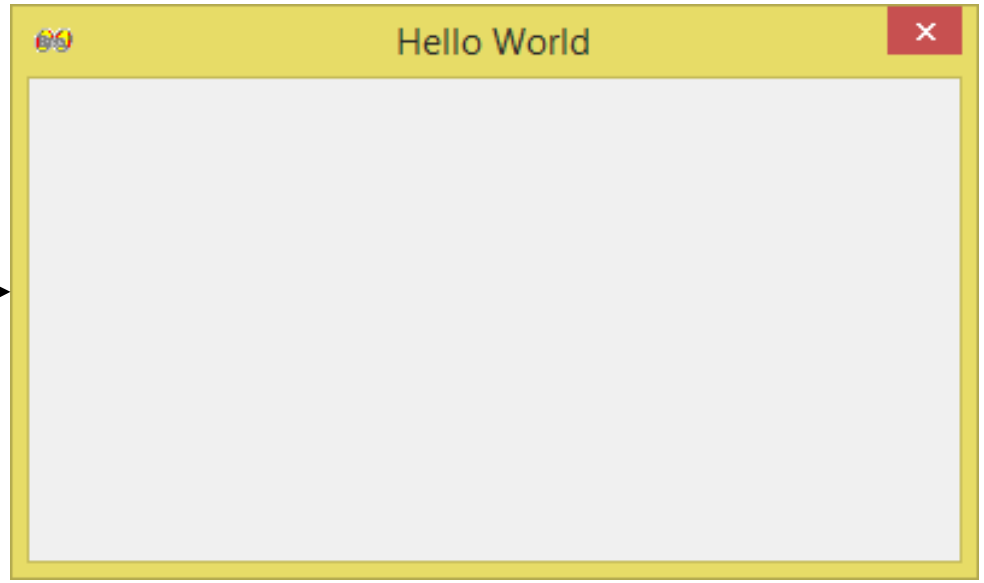


The .rc File:

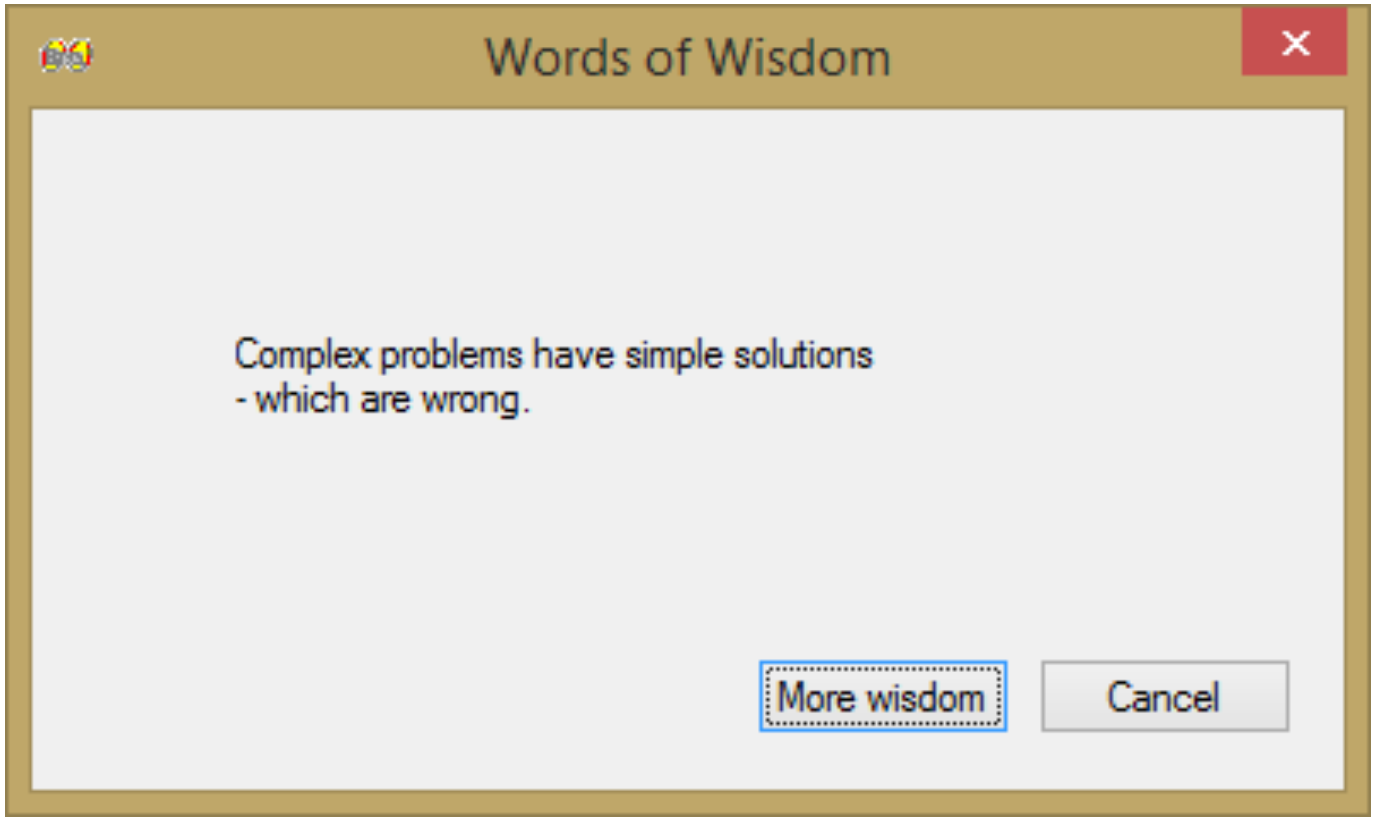
```
#include <windows.h>
#include <commctrl.h>
#include <richedit.h>
#include "CustomerView.h"

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_SYSMENU
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

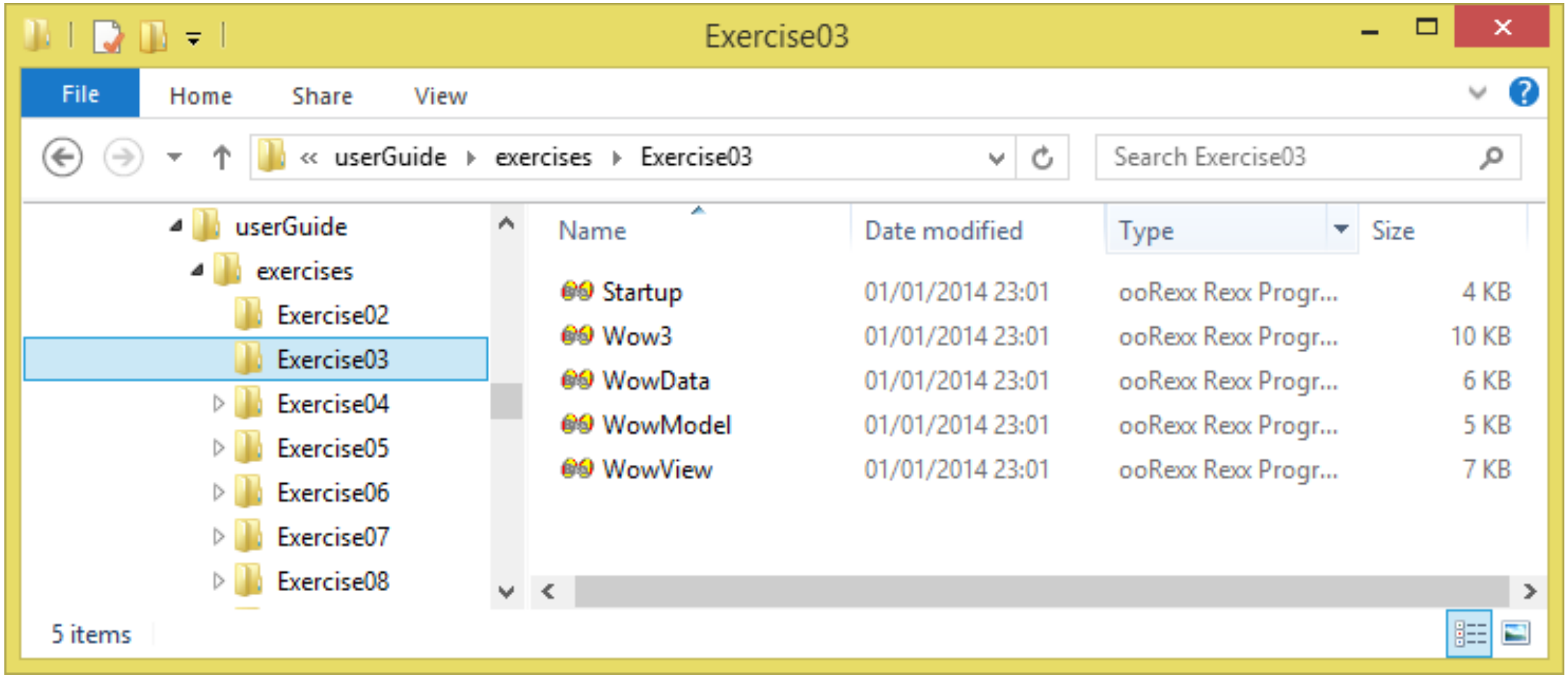
```
dlg = .HelloWorld~new
dlg~execute("SHOWTOP", IDI_DLG_OOREXX)

::REQUIRES "ooDialog.cls"

::CLASS 'HelloWorld' SUBCLASS UserDialog

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



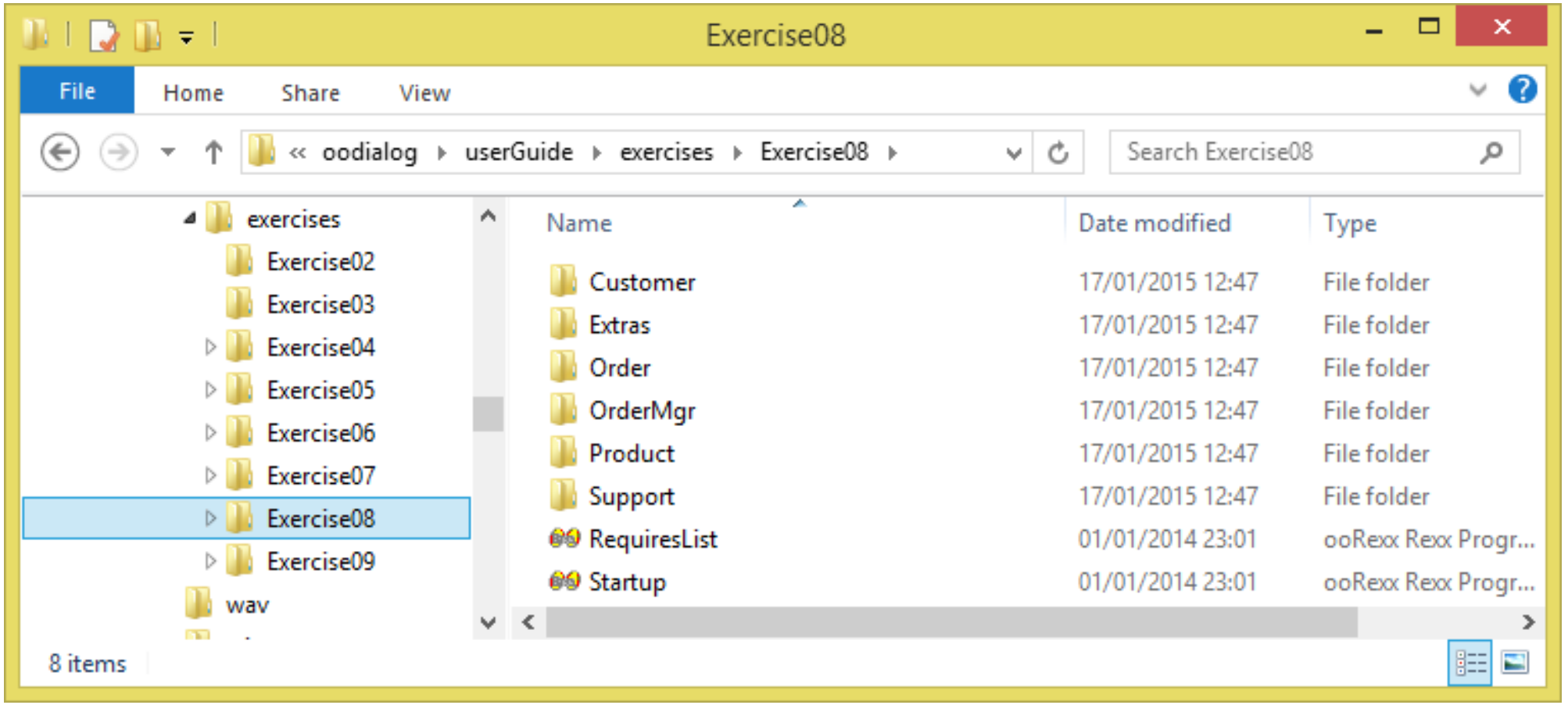
Words of Wisdom



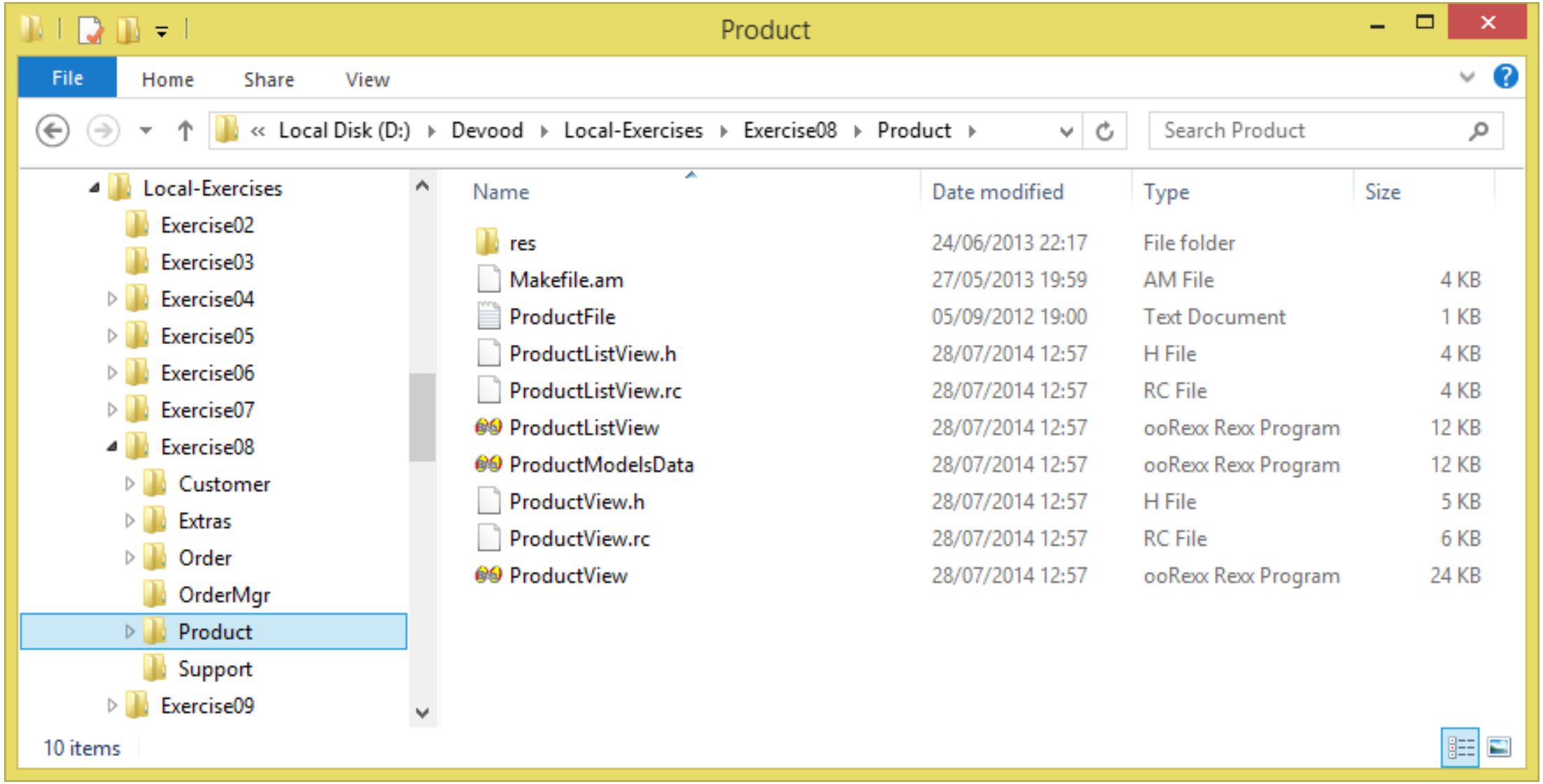
Separation of Concerns



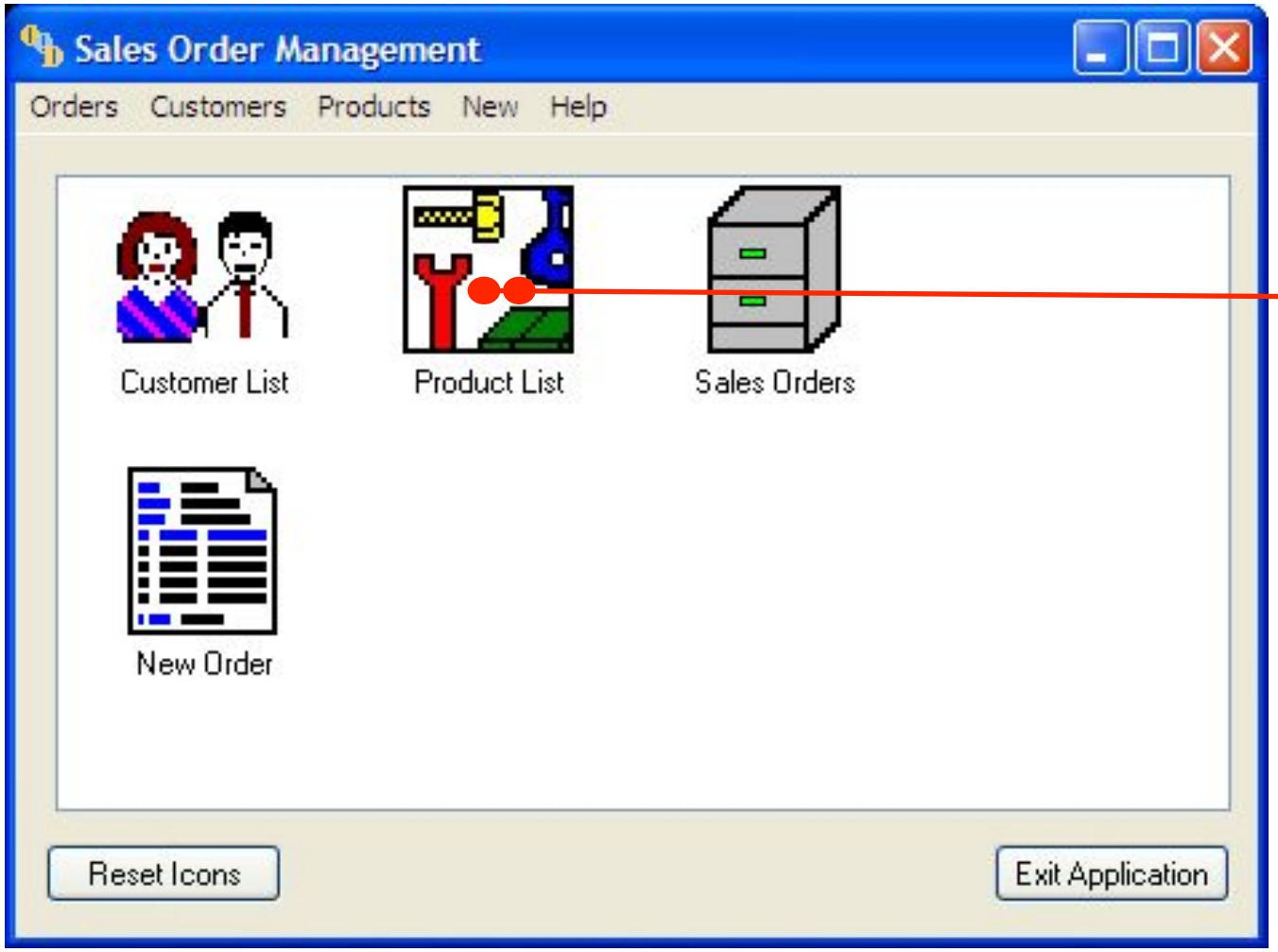
Business Components

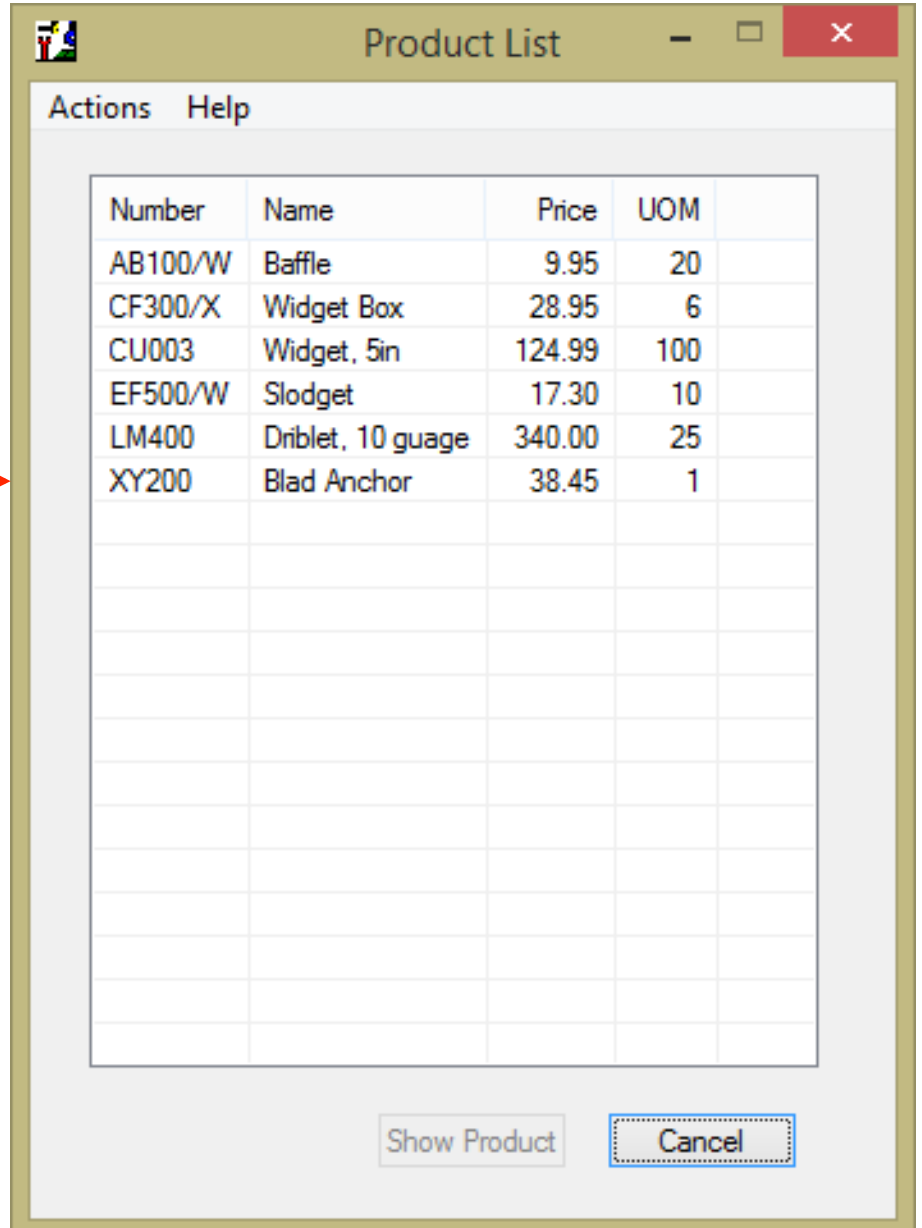


The “Product” Business Component



Opening Another Dialog

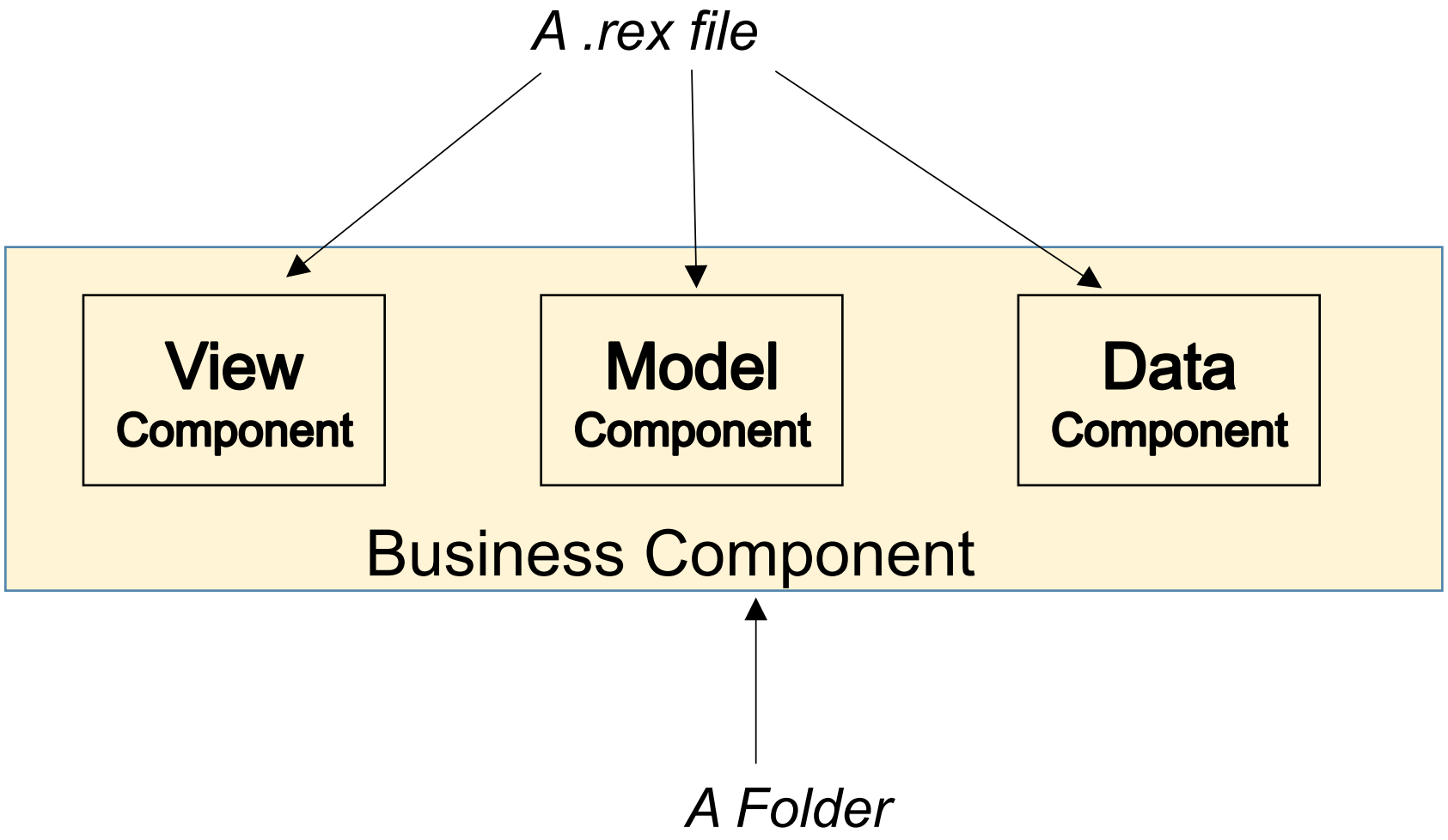




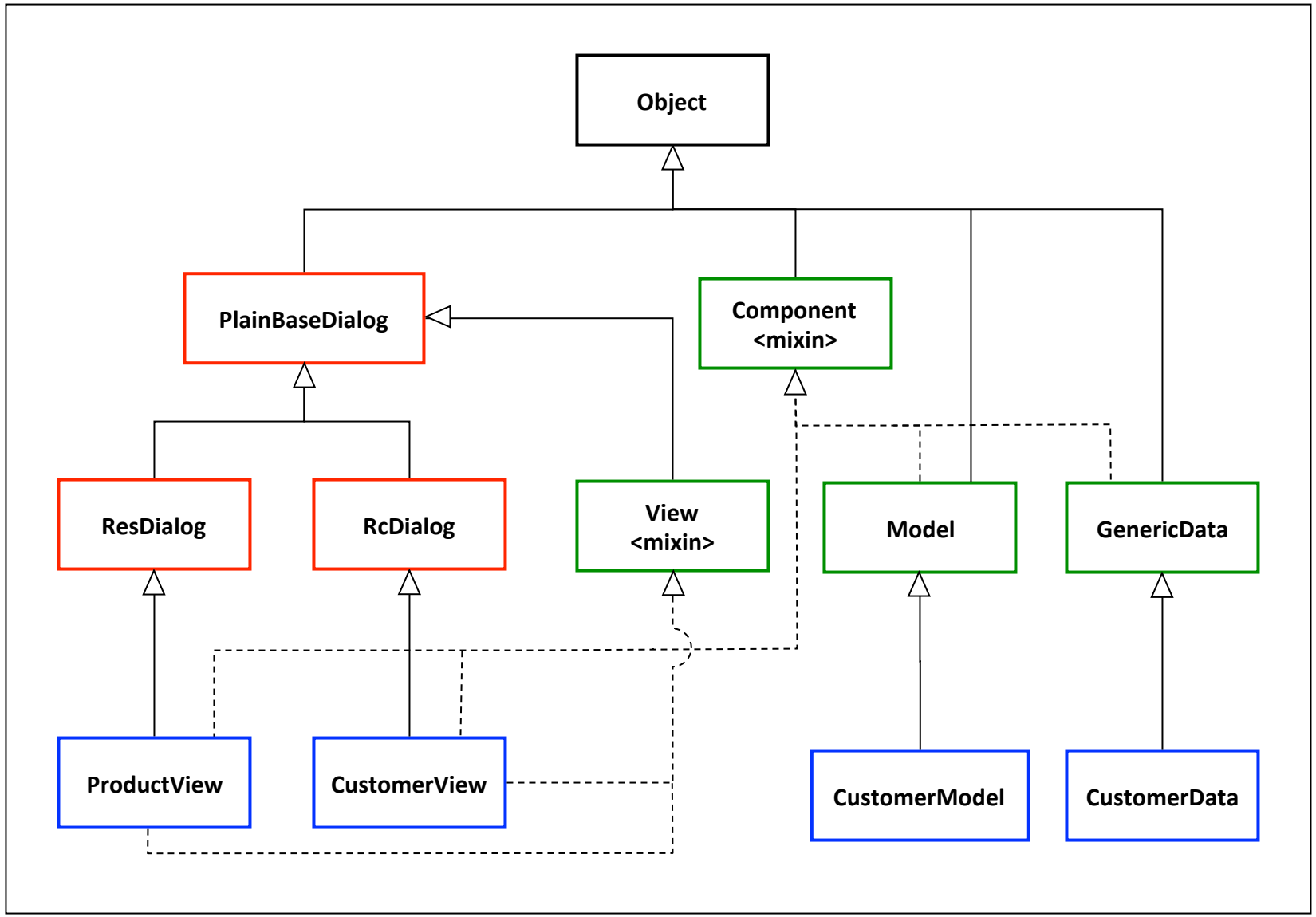
The “Infrastructure”

- 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`

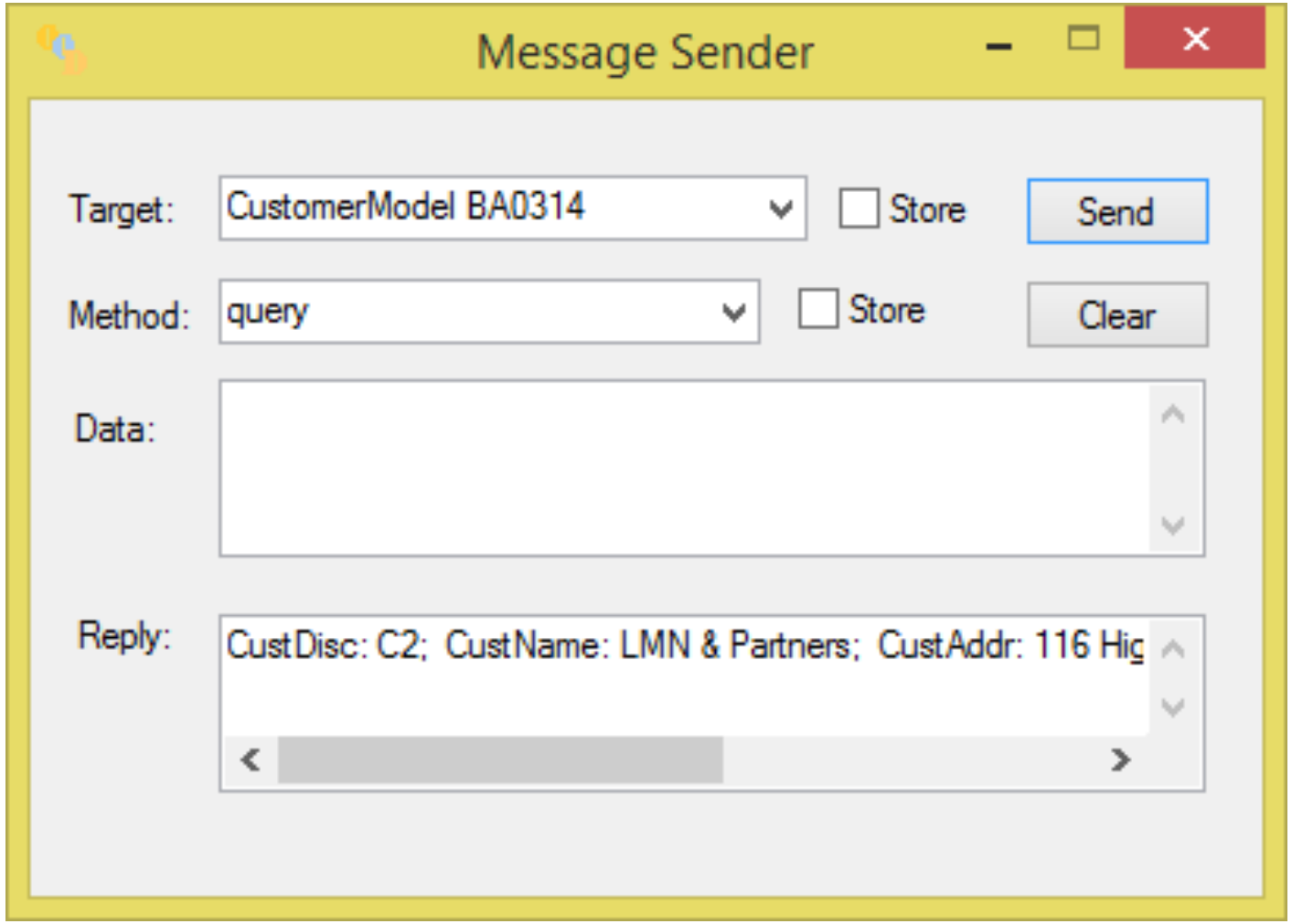
Dialog Setup – The Model-View Framework

- 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):

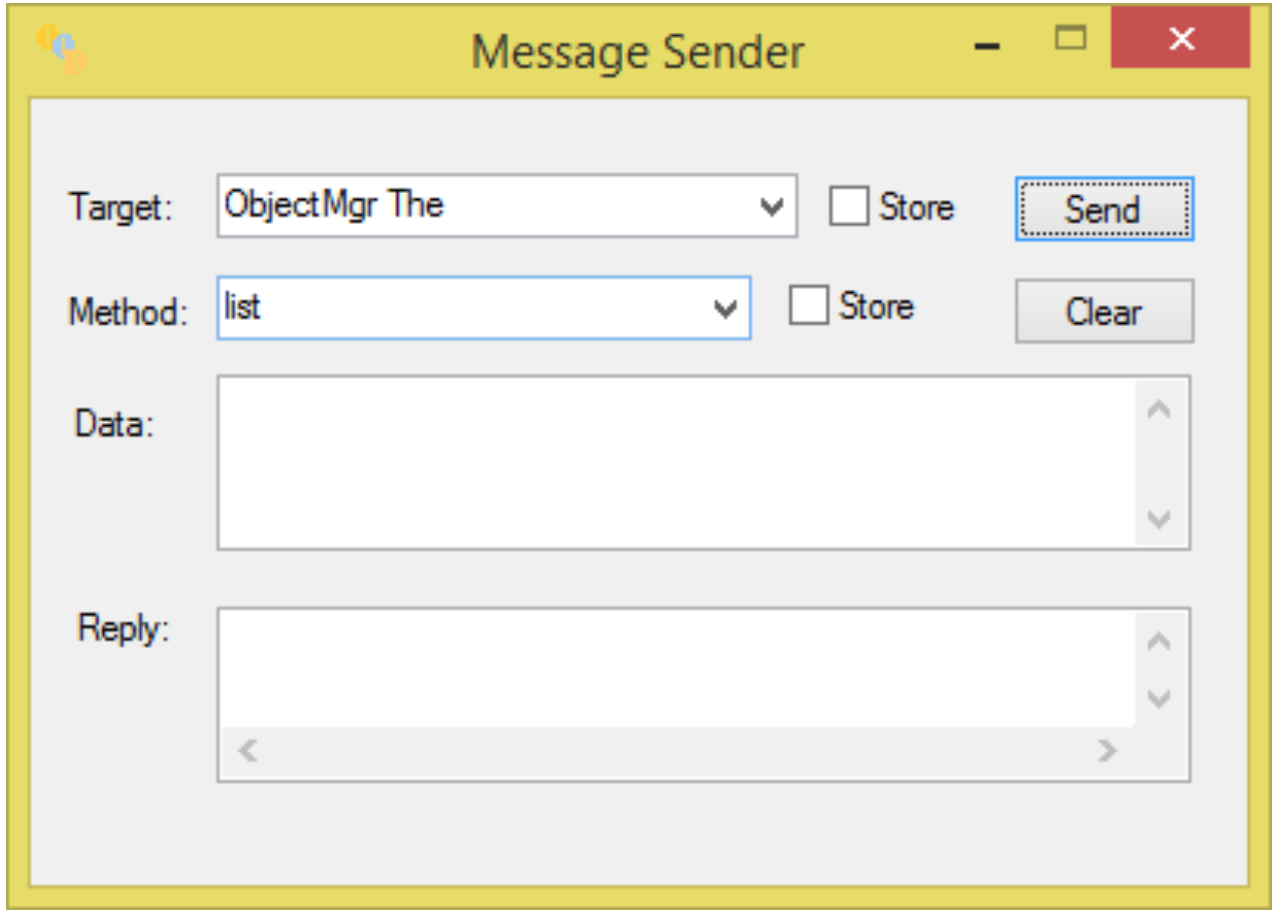
```
::METHOD showCustomer UNGUARDED
  expose lvCustomers
  info = .Directory~new
  -- identify which Customer has been double-clicked (custNum is in info~text)
  if lvCustomers~getItemInfo(item, info) then do
    objectMgr = .local~my.ObjectMgr
    objectMgr~showModel("CustomerModel", info~text, rootDlg)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

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

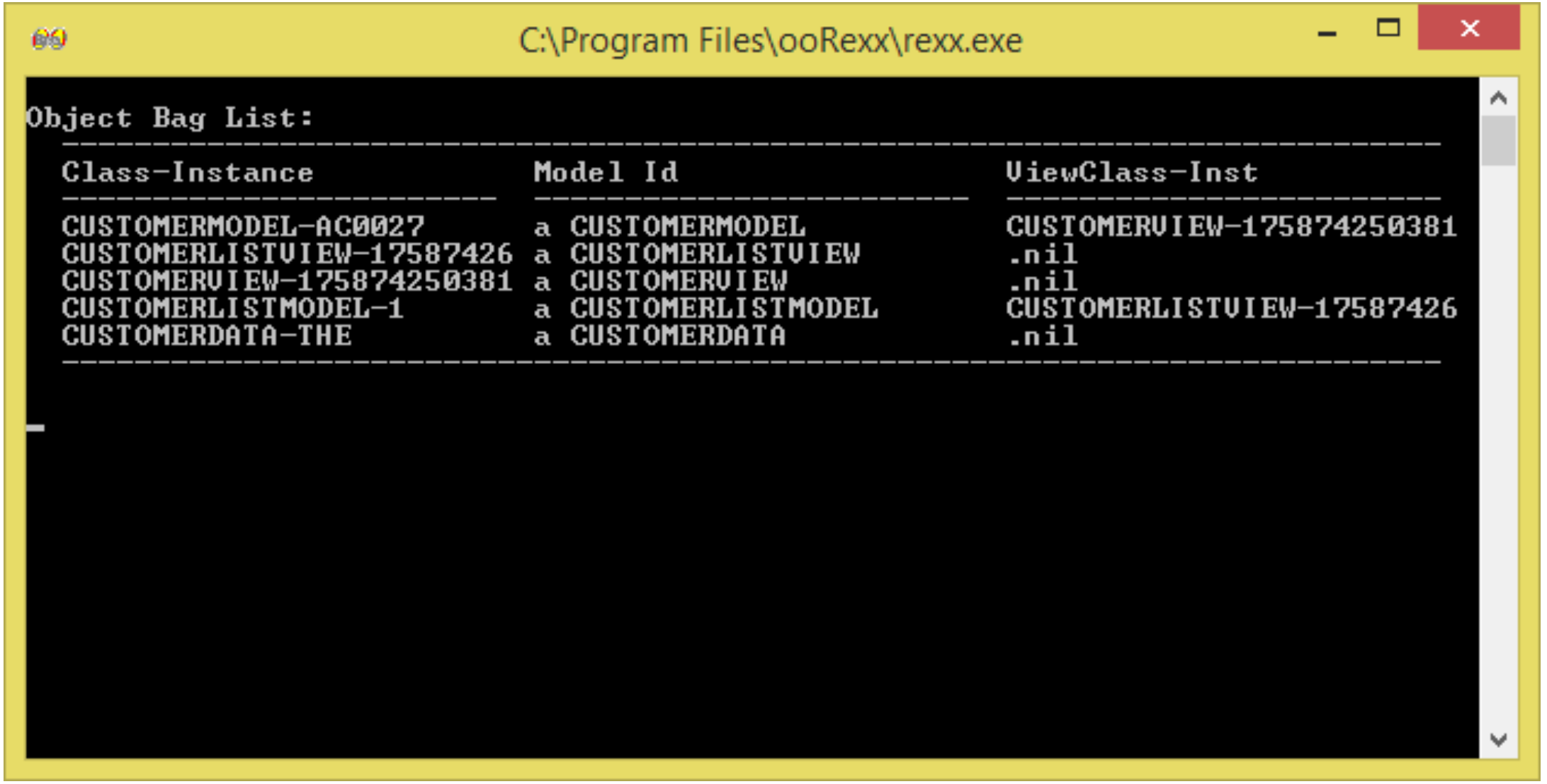
The Message Sender - Querying a Customer Instance



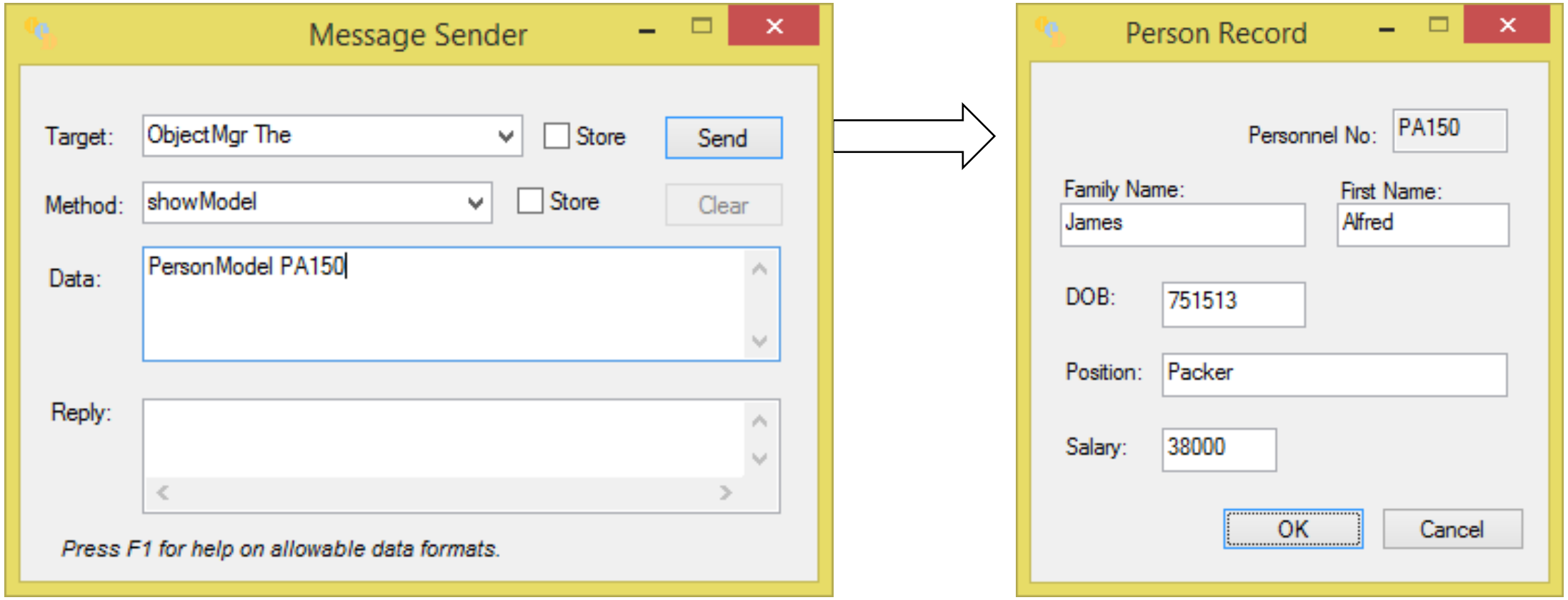
Debugging - The Message Sender – What components exist?



List of Component Names



Debugging - The Message Sender



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.