Control Models

Jim Fawcett
CSE775 – Distributed Objects
Spring 2007

Control Types

- Components
  - Hosted by toolbox and in Form's tray
- Controls
  - Visible interface hosted by toolbox and on Form's surface
- UserControls
  - Composite control
  - Container for controls
  - Control for forms
- Derived Controls
  - Any of the above, with much functionality provided by a base component or controls

What it takes to be a Component

- Class that implements the IComponent interface
  - Can be hosted in containers
  - Reusable,configurable classes
  - Don’t have hosted UI, keyboard, and mouse processing
  - Components may have Tooltip or dialog UI, e.g., OpenFileDialog and SaveFileDialog.
  - Neither of these requires Form real-estate

What it takes to be a Component

- When pulled onto a Form, components:
  - Join the form's System.ComponentModel.Container
  - Are given, in their constructor, a reference to ComponentModel.Container
  - If the component exposes properties and events these are automatically integrated with the Form Designer's property window
  - A component does this by defining .Net class properties and delegates

Standard Components

- BackgroundWorker
- DirectoryEntry
- DirectorySearcher
- ErrorProvider
- EventLog
- FileSystemWatcher
- HelpProvider
- ImageList
- MessageQueue
- PerformanceCounter
- Process
- SerialPort
- ServiceController
- Timer

Building a Custom Component

- You can create a component this way:
  - Use the project wizard to create a Windows Forms Control library
  - Change the base class to:

```
System::ComponentModel::Component
```

  - Add a constructor:

```
componentClass(IContainer^ container)
{
   container->Add(this);
   InitializeComponent();
}
```

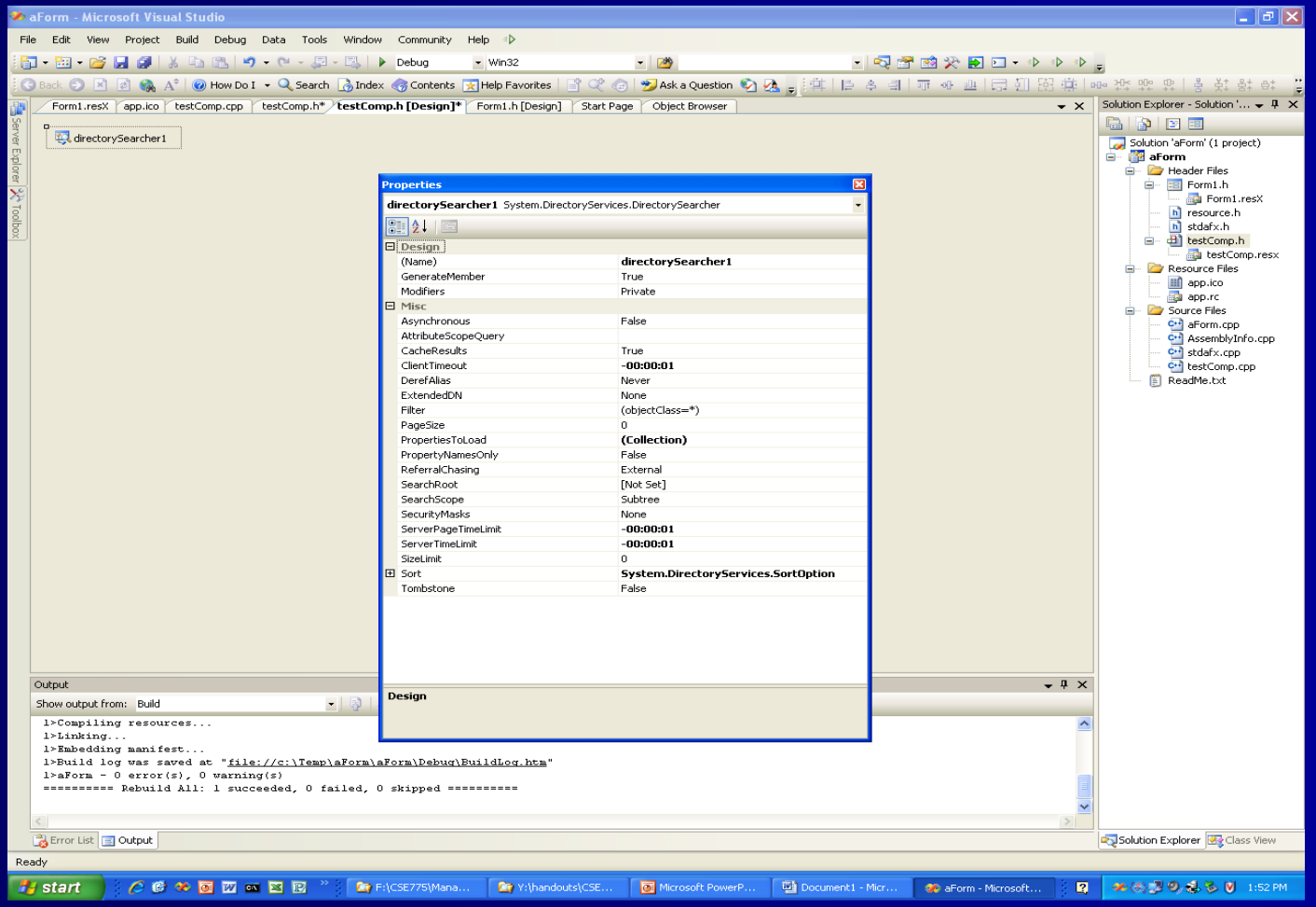
  - Change the contents of the InitializeComponent() method to the line of code:

```
this->components = gcnew
System::ComponentModel::Container();
```

Building a Custom Component

- An even easier way is to:
  - Build a Windows Forms Application
  - Right-click on the Form project and select add new item
  - Add Component Class
  - A blank design view will appear onto which you can, but do not have to, pull other components from the toolbox

Here is the result of pulling on the DirectorySearcher



Adding Components to the Toolbox

- When you build a project that contains either components or controls, they will automatically be added to the toolbox under a category with the name of the project.
- Now, you can just pull them onto a form.
- So a good strategy for building components and controls is to:
  - start with a test form
  - add component and usercontrol items with the new item wizard
  - Build the form project
  - Pull the component or control onto the form for testing.

Adding Properties and Events

- To add events and properties to a component you insert the following:
  - Event

```
delegate void TestEvent(String^);
event TestEvent^ test_event;
```

  - Property:

```
property String^ test_property;
```

- If you've made them public, once you build the project you will find them in the component's property sheet.

Component Candidates in Project #3

- Communication component
  - Hide all the block handling behind an adapter that accepts and returns strings through postMessage and getMessage functions.
- Filehandler component
  - Use the .Net standard components or our FileInfo and Wintools facilities.

Controls

- Controls are very similar to components, but they also:
  - Provide a UI that is painted on the surface of the container
  - Respond to user input in the form of keystrokes and mouse actions

What it takes to be a control

- Derives from System.Windows.Forms.Control
- Control derives from Component, so you inherit all of the features we've already discussed.