



# Extending ArcGIS Desktop Applications with .NET

*Kevin Deege*  
*Katy Dalton*



# Introductions

- **Who are we?:**
  - Katy Dalton – ArcGIS .NET SDK team
  - Kevin Deege – ArcGIS ArcObjects Instructor
  
- **Who are you?:**
  - .NET developer
  - Some familiarity with ArcObjects

# Schedule

- 75 minute session
  - 60 – 65 minute presentation
  - 10 – 15 minutes Q & A following the presentation
- Cell phones and pagers

***Please!***  
***Turn OFF cell phones***  
***and paging devices***



***Please complete the session survey!***

# Agenda

- **Installation requirements**
- **Developer resources**
- **ArcGIS System architecture**
- **ArcGIS Desktop application framework**
- **Implementing custom components**
  - **Commands, Toolbars, Dockable Windows, Extensions**
  - **Using IDE integration and BaseClasses**

# Desktop Development requirements

## *Software and Licensing*

- Desktop development requires the following:
  - Software
    - ArcGIS Desktop
    - ArcGIS Desktop Developer Kit
    - Visual Studio 2005 / Visual Studio 2008
  - Understand product licensing
    - ArcView, ArcEditor, ArcInfo
    - Extensions
  - Desktop Administrator
    - Check available license

# ESRI Developer Network

*GIS and Mapping Solutions for Developers*



- Entire ArcGIS platform
- Full licenses for development and basic testing
- Annual subscription
- Discounts: technical support, training
- Online resources
- User communities

# Resource Centers

## Online Developer Resources

- SDK Help
- Blog
- Code Gallery
- Forums



ArcObjects Development Blog

# ArcGIS Desktop Development

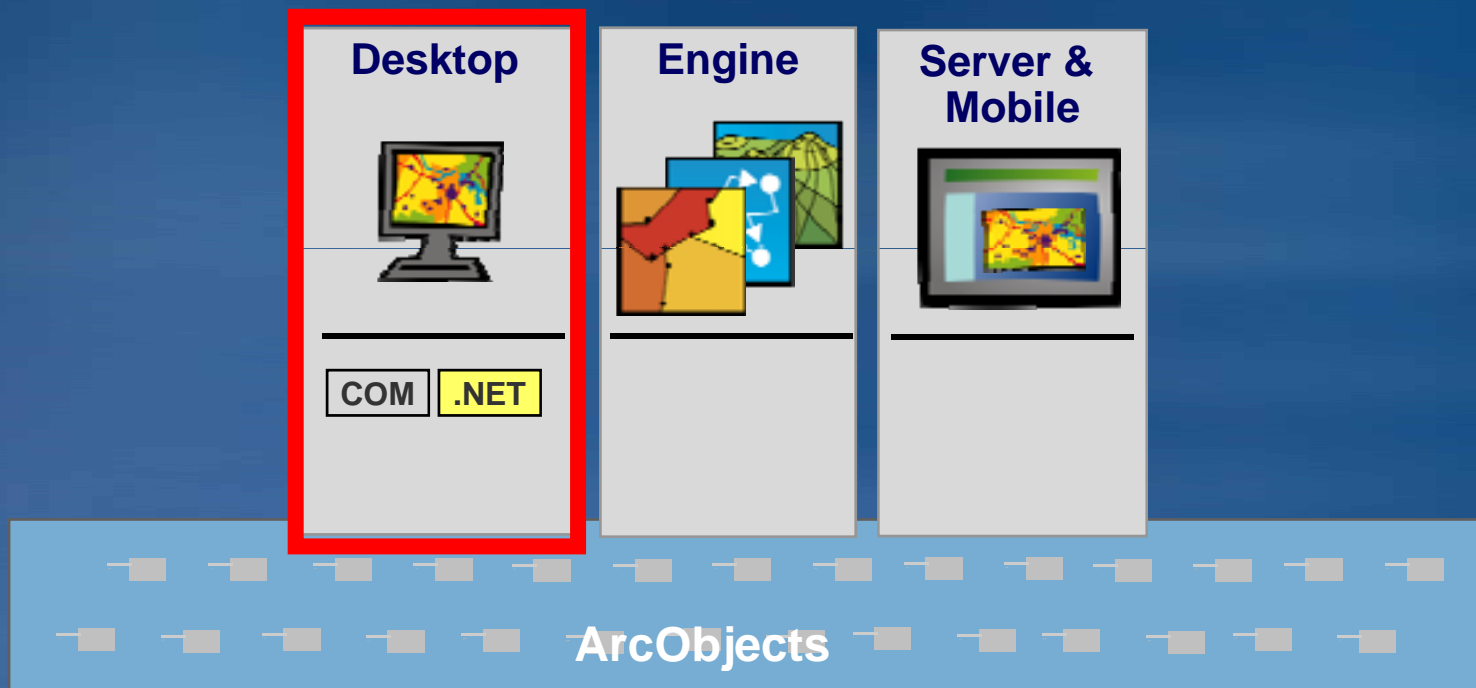
## *Core knowledge*

- ArcGIS architecture
- ArcObjects and the .NET API
- The Application framework



# ArcGIS Desktop

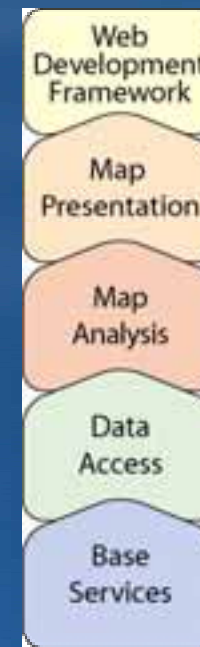
*Built with ArcObjects*



# ArcGIS Architecture

*Products Share ArcObjects*

- Desktop extends the core ArcObjects functionality
  - Applications, Extensions



# ArcObjects Assemblies

- **Shared Assemblies**
  - System
  - Carto
  - Geodatabase
  - ....
- **Desktop Assemblies**
  - e.g. esriArcMapUI, Framework
- **Identify libraries that belong to different products**
- **Understand library dependencies**
- **Developer tools**
  - ArcGIS Developer Help

### ArcObjects libraries shared across the ArcGIS platform

**Summary**  
Many ArcObjects libraries included with ArcGIS Desktop are actually shared across the entire ArcGIS platform. These shared libraries are available through ArcGIS Engine and contain components that are not part of the ArcGIS user interface (UI). These shared libraries are summarized in this document. Understanding the library structure, dependencies, and basic functionality helps you as a developer navigate through the components of ArcGIS.

**In this topic**

<a href="#">About ArcObjects libraries shared across the ArcGIS platform</a>	
<a href="#">System library</a>	<a href="#">GeodatabaseExtensions library</a>
<a href="#">SystemUI library</a>	<a href="#">Carto library</a>

### ArcObjects libraries exclusively in ArcGIS Desktop

**Summary**  
The libraries contained in ArcGIS Desktop are summarized in this document. Understanding the library structures, their dependencies, and basic functionality helps you as a developer to navigate through the components of the ArcGIS Desktop applications.

**About ArcObjects libraries**  
The libraries are discussed in dependency order. The libraries that are common for the entire ArcGIS platform are discussed separately in [ArcObjects libraries shared across the ArcGIS platform](#).

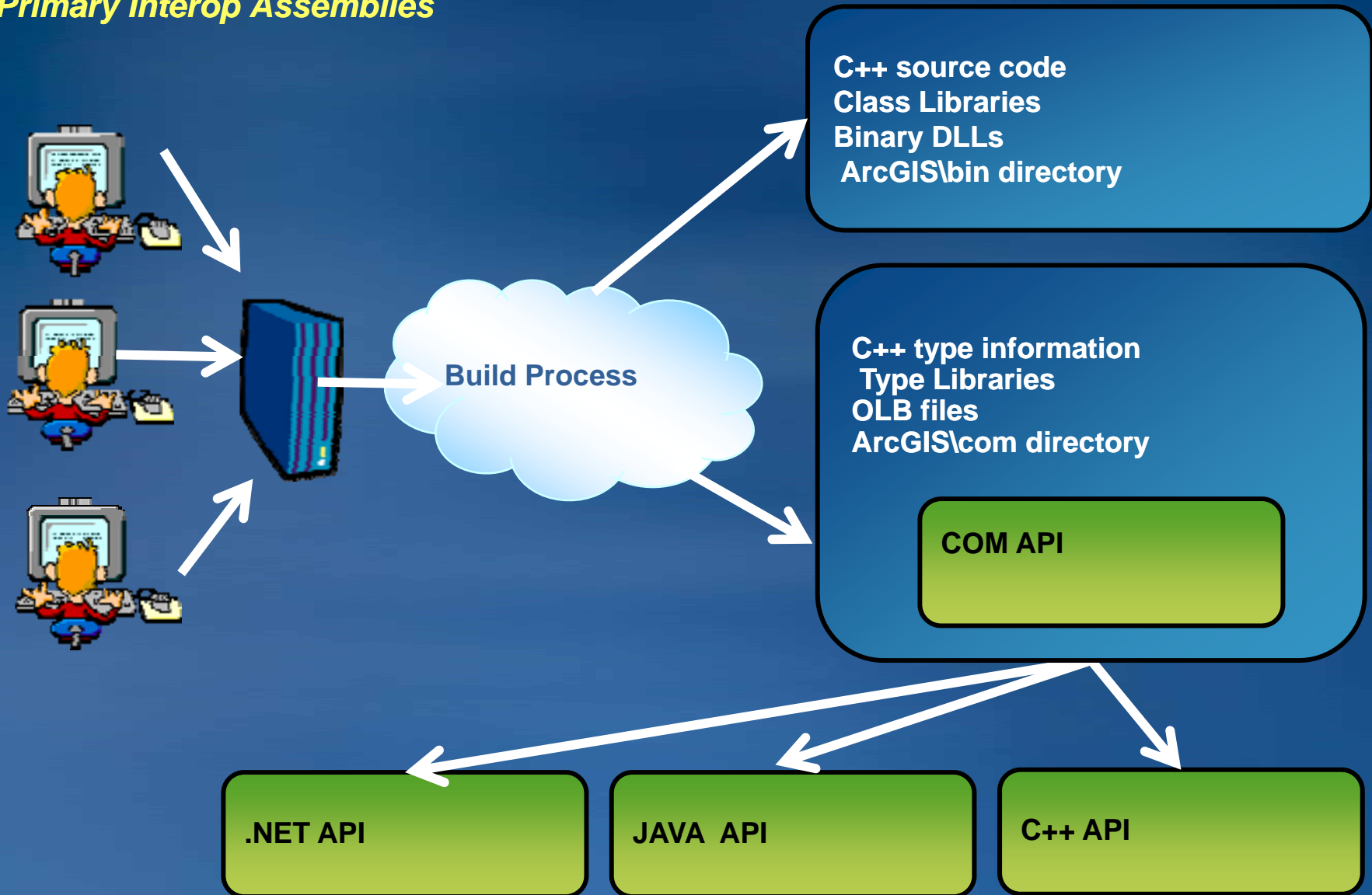
For a comprehensive discussion on each library, see the overview topic for each library in the [ArcObjects library reference](#) section of the help system.

Knowing the library dependency order is important, since it affects the way developers interact with the libraries as they develop software. For example, C++ developers must include the type libraries in the library dependency order to ensure correct compilation. Understanding the dependencies also helps when deploying your developments.

**Framework library**  
The [Framework](#) library provides core components and software interfaces to support user interface (UI) components and ArcGIS applications. A number of the objects in the Framework library are used internally by ArcGIS applications to support their customization environment. There are a number of helper objects in the Framework library that developers can use when creating user interfaces for inclusion in one of the ArcGIS applications—ComPropertySheet, ModelessFrame, and MouseCursor are three examples—along with a set of dialog boxes that support common UI operations in an ArcGIS application; ColorSelector and NumberDialog are two commonly used dialog boxes. The Framework library defines the software interfaces that developers use when creating UIs for extending the ArcGIS system using property pages and dockable windows. The Framework library is not extended by the developer, but by implementing interfaces defined in the library, developers can extend the ArcGIS architecture with UI components.

# ArcObjects – Inside the .NET API

## Primary Interop Assemblies



# ArcObjects – .NET API

## COM Interoperability

- All ArcObjects types are defined in .NET assemblies
- Exposes all classes, interfaces, and constants as managed .NET types
- **Primary Interop Assemblies** were generated by importing the COM type libraries
- Developer Help
  - Programming with .NET

Programming with .NET	
In this section	Description
<a href="#">MSDN .NET developer centers</a>	Provides links to Microsoft C# and Visual Basic (VB) developer centers for detailed .NET help
<a href="#">Interoperating with COM</a>	Discusses working with .NET and the Component Object Model (COM) together
<a href="#">Binary compatibility</a>	Discusses maintaining binary compatibility in .NET
<a href="#">System.__ComObject and casting to strongly typed RCWs</a>	Describes the System.__ComObject type and explains why casting sometimes fails when it appears it should succeed
<a href="#">Releasing COM references</a>	Provides information on the interoperation of COM and .NET including how memory is managed in each

Interop

.NET Assemblies  
(\* .dll)

COM Type Libraries  
(\* .olb)

ArcObjects C++  
(\* .dll)

# ArcObjects .NET API

## *Assemblies and Namespaces*

- All .NET projects must reference the ESRI assemblies
  - Eg., ESRI.ArcGIS.Carto.dll
- Import namespace to access ArcObjects .NET types
  - Eg., ESRI.ArcGIS.Carto

```
'VB.NET
Imports ESRI.ArcGIS.CatalogUI
Imports ESRI.ArcGIS.ArcMapUI
Imports ESRI.ArcGIS.Framework
. . .
// Visual C#
using ESRI.ArcGIS.CatalogUI;
using ESRI.ArcGIS.ArcMapUI;
using ESRI.ArcGIS.Framework;
```

# ArcGIS Desktop Application Framework

## *Integrate customizations*

- All Desktop applications share a similar framework
  - Customization
  - Event Handling
  - Property Persistence
- Customize the user interface at runtime
  - Add/Remove/Reorganize commands, tools, menus, and toolbars
- Customize the applications with VBA
  - Macros, buttons, tools, toolbars, and menus
- **Extend the applications by:**
  - **Creating a custom COM component**
  - **Registering it in a Component Category**

# Extending ArcGIS Desktop Applications

*Create visual and Non-visual components*

## UI Components

- 1) Commands and Tools
- 2) Menus and Toolbars
- 3) Embedded Windows
  - Dockable Windows, Contents Views

## Non-visual Components

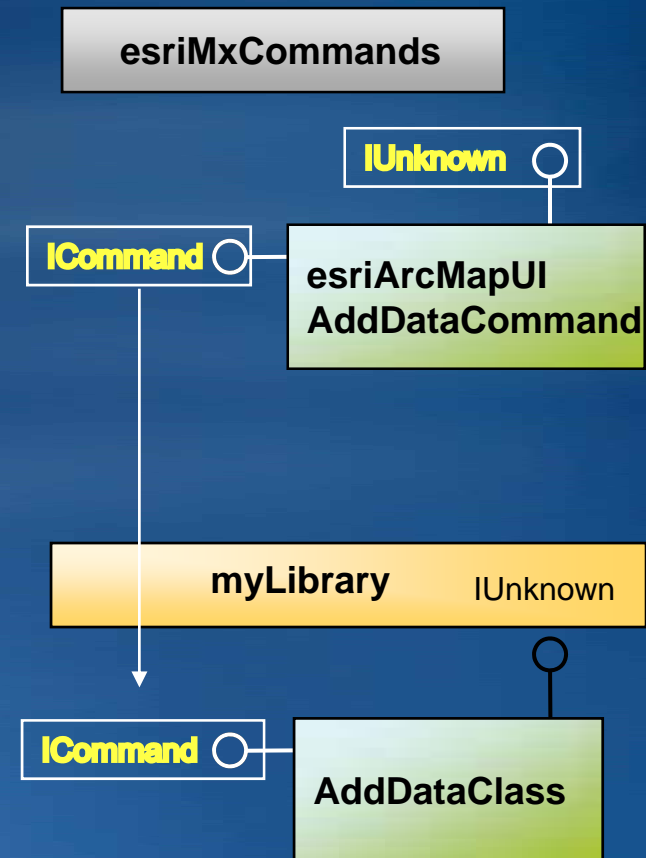
- 1) Application Extension
- 2) Undo/Redo Operations
- 3) Command keyboard shortcuts



# Developing COM components in .NET

*Create and Register custom components*

- **Steps:**
  1. Create a .NET project
  2. Create a COM class
  3. Reference the ArcGIS libraries
  4. Implement an **interface**
  5. Compile
  6. Register in a component category
- **Model after ESRI components**
- **Use IDE integration!**



*Help Topic – Working with ArcGIS Components*

# Application Startup cycle

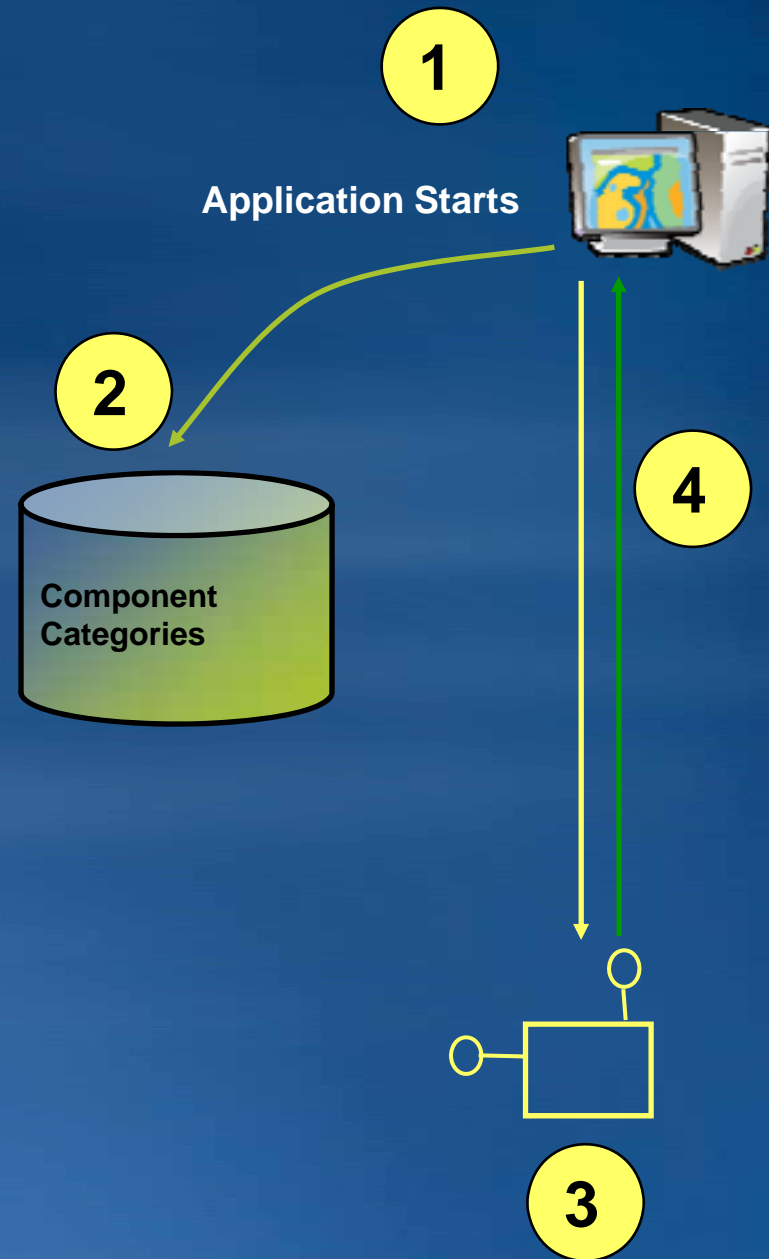
## Component Categories

- Components loaded at runtime using Component Categories

- Mx.... = ArcMap
- GMx ..= ArcGlobe
- Gx.... = ArcCatalog

- Steps:

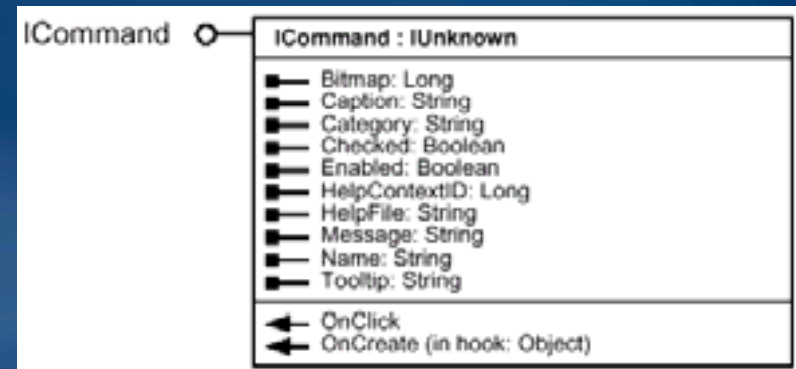
1. Application starts
2. Reads the component categories from the registry
3. Creates all ESRI and custom components
4. Loads the objects



# Creating Commands and Tools

## *Extending Toolbars*

- Model after the Button class
- **Base Class: BaseCommand**
- Interface: ICommand
- Type Library: esriSystemUI
- Component Category
  - ESRI Mx Commands: ArcMap
  - ESRI Gx Commands: ArcCatalog
  - ESRI Sx Commands: ArcScene
  - ESRI GMx Commands: ArcGlobe
- Other related interfaces and objects
  - ICommandItem
  - ITool, IToolControl

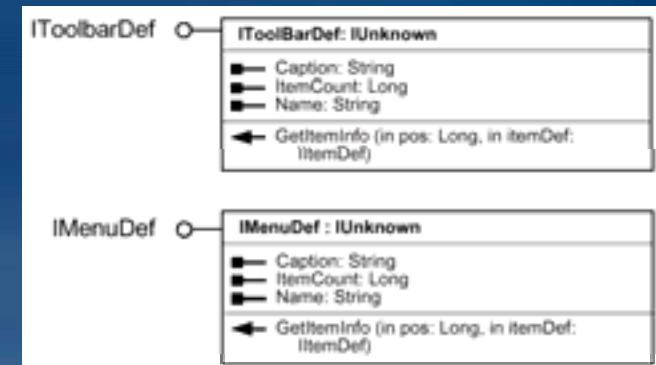


*Walkthrough – Create a command by inheriting from BaseCommand*

# Creating Toolbars

## Implementation Details

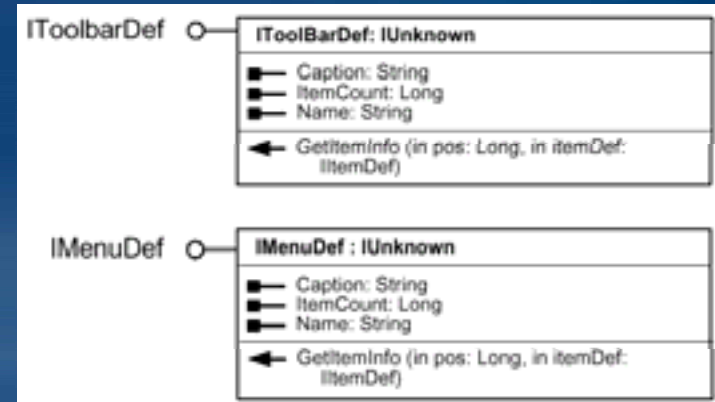
- Model after an ESRI toolbar
- **Base Class: BaseToolbar**
- Interfaces: IToolBarDef
- Type Library: esriSystemUI
- Component Category
  - ESRI Mx CommandBars: ArcMap
  - ESRI GX CommandBars: ArcCatalog
  - ESRI Sx CommandBars: ArcScene
  - ESRI GMx CommandBars: ArcGlobe
- Other related interfaces and objects
  - Buttons, Tools
  - Extensions



# Creating Menus

## Root Level or Context Menu

- Model after an ESRI menu
- **Base Class: BaseMenu**
- Interfaces: IMenuDef
- Type Library: esriSystemUI
- Component Category
  - ESRI Mx CommandBars: ArcMap
  - ESRI GX CommandBars: ArcCatalog
  - ESRI Sx CommandBars: ArcScene
  - ESRI GMx CommandBars: ArcGlobe
- Other related interfaces and objects
  - IRootlevelMenu, IShortcutMenu, IMultitem



# Visual Studio IDE Integration

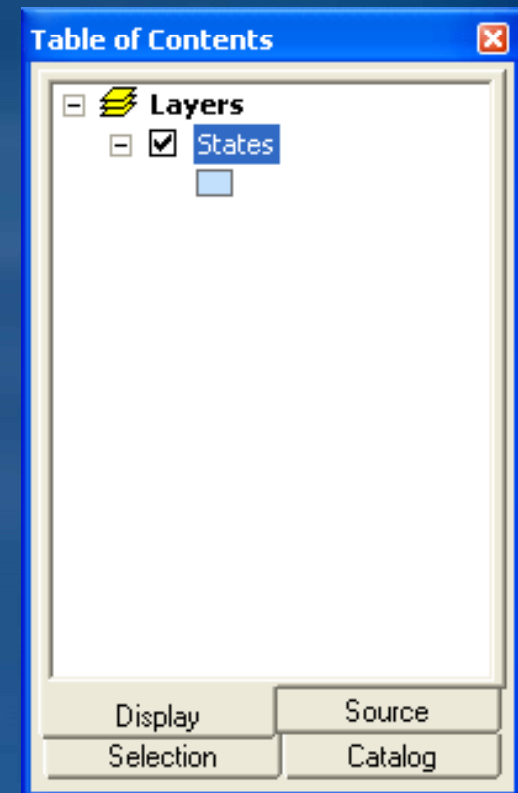
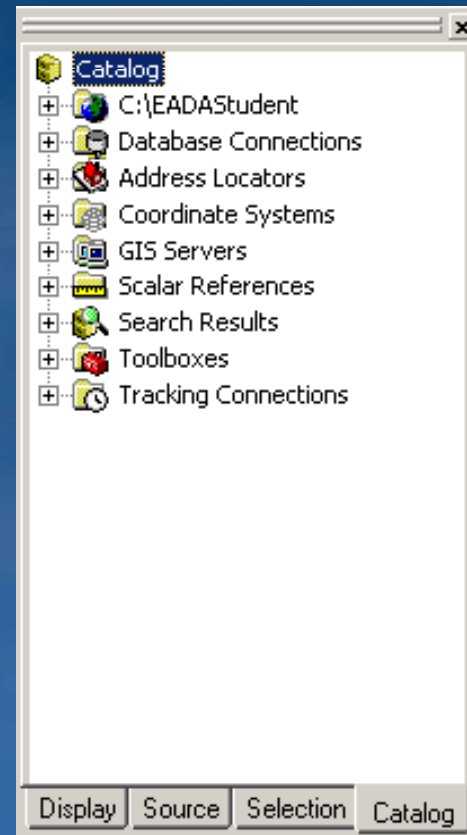
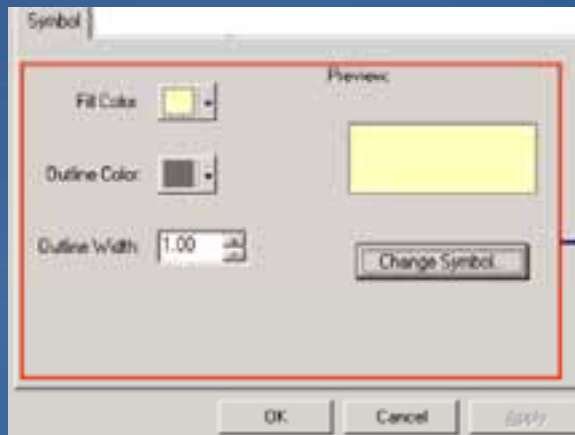
## *Tools and shortcuts*

- **New Project templates**
- **Base classes**
  - BaseCommand, BaseTool, BaseToolbar, BaseMenu
- **Add Class wizard**
- **Code Snippets**
  - Snippet Finder Utility
  - Snippet Editor (9.3.1)
- **Quickly adding Imports/Using statements**

# Custom Windows

*Plug a User Control into the Application*

- Three main types
  - Property pages
  - Tabbed views
  - Dockable windows



# ArcGIS Custom windows

## *Plug in user controls*

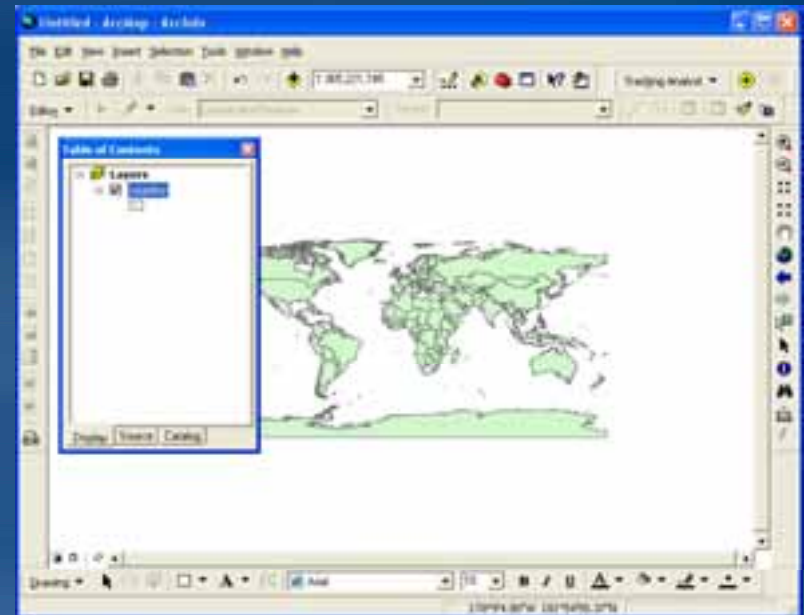
- **Application Framework manages**
  - Parent frame of the window
  - Placement of windows
  - Resizing
  - User events
  - Persisting properties of windows
- **Provides internal classes with default implementation**
  - Dockable window, contents view, property sheets
- **User provides behavior of internal child window**



# Dockable Windows

*Available for all applications*

- Floating or docked window that is associated with the application
- Show with a command
- Examples
  - ESRI TOC window
  - Show any type of data
    - Symbols, charts, and metadata
  - May contain ESRI controls



# Dockable Windows

## *Show with a command*

- Create a custom command to manage the window
  - OnCreate: Find the window, hold on to window reference
  - OnClick: Show or hide the window
  - Provided by the template
- Need to know the CLSID of the dockable window

```
Private Const wGUID as string = {27b18bbd-756a-4921-815d-7a1516e2b762}

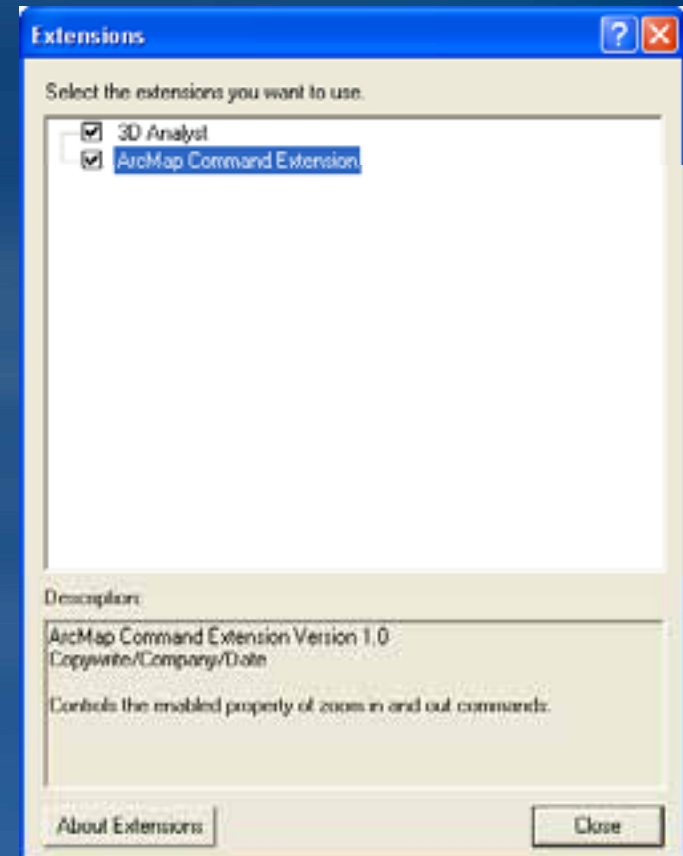
Public Overrides Sub OnCreate(ByVal hook As Object)
    m_app = CType(hook, IApplication)
    m_dockWinManager = CType(m_app, IDockableWindowManager)
    Dim u As New UID
    u.Value = wGUID
    m_dockableWindow = m_dockWinManager.GetDockableWindow(u)
End Sub

Public Overrides Sub OnClick()
    m_dockableWindow.Show(Not m_pDockWin.IsVisible)
End Sub
```

# ArcGIS Desktop Extensions

*Supported by all applications*

- **Centrally accessible**
  - Event Handling
  - Document Persistence
  - Shared Data
  - License Management
- **Visible through Extension Manager**



# Applications Extensions

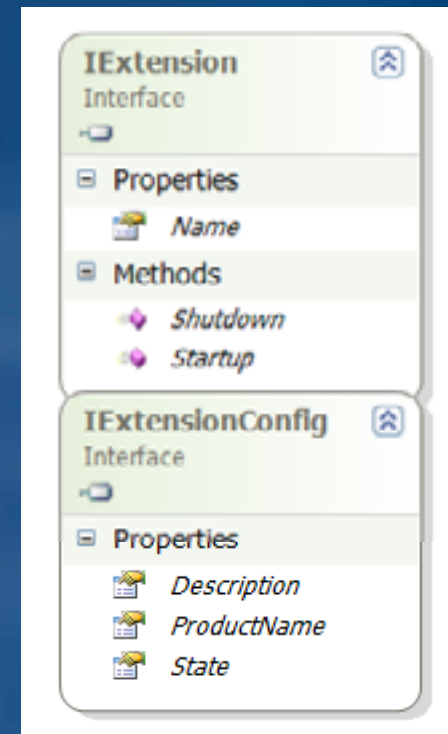
## *Invisible or Visible*

- **Two types of extensions:**
- **Invisible**
  - Automatically enabled by application at runtime (e.g., Editor)
  - Transparent to user that extension has been loaded
- **Visible**
  - Enabled through extension manager window
  - Generally require a license or special criteria to enable
  - Some check out an ESRI license (e.g., 3D Analyst)

# Application Extensions

## Implementation and Category Registration

- Model after any ESRI Extension
- Interface(s):
  - IExtension, IExtensionConfig
- Type Library: esriSystem
- Component Category (load on startup)
  - ESRI Mx Extensions
  - ESRI GX Extensions
  - ESRI Sx Extensions
  - ESRI GMx Extensions
- “JIT” Extensions
  - Load at runtime



# Implementing IExtension

## *Invisible Extensions*

- **Required Implementation:**
  - **Name:** Internal name
  - **Startup:** Passed in the application or other data
  - **Shutdown:** Clean up all references

```
Private m_app as IApplication

Public ReadOnly Property Name() As String Implements IExtension.Name
    Get
        Return "My Custom Extension"
    End Get
End Property

Public Sub Shutdown() Implements IExtension.Shutdown
    m_app = Nothing
End Sub

Public Sub Startup(ByRef initializationData As Object) Implements IExtension.Startup
    If (TypeOf initializationData Is IMxApplication) Then
        m_app = initializationData
    End If
End Sub
```

# Application Extensions

## *Handle Document Events*

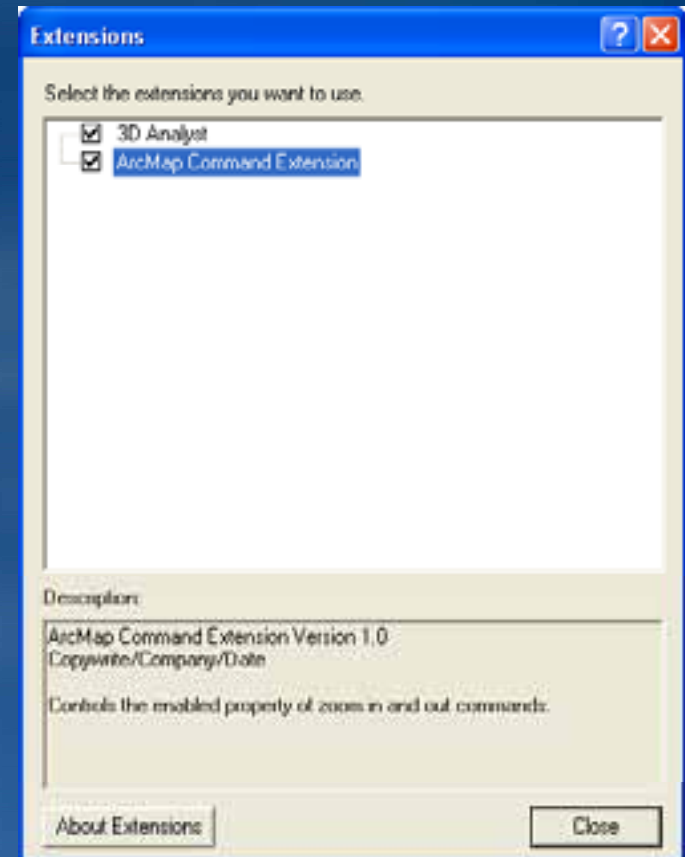
- Use Startup to sink document events
  - NewDocument, OpenDocument, CloseDocument
  - ActiveViewChanged, MapsChanged, OnContextMenu

```
Private documentEvents As MxDocument
Private dNewDocument As IDocumentEvents_NewDocumentEventHandler
Public Sub Startup(ByRef initializationData As Object) Implements IExtension.Startup
    If (TypeOf initializationData Is IMxApplication) Then
        m_app = initializationData
        documentEvents = CType(m_app.Document, MxDocument)
        SinkEvents()
    End If
End Sub
Private Sub SinkEvents()
    dNewDocument = New IDocumentEvents_NewDocumentEventHandler(AddressOf OnNewDoc)
    AddHandler CType(documentEvents, IDocumentEvents_Event).NewDocument,
dNewDocument
End Sub
Public Sub OnNewDoc()
    ' Do some sort of check
    If (Not ValidUser) Then DisableSomeCommands
End Sub
```

# Application Extensions- IExtensionConfig

## *Visible Extensions*

- Adds extension to the extension manager window
- Name appears in the list
- Looks like all other ESRI extensions
- Description appears at the bottom
  - Extension name and version
  - Copyright and company name
  - Extension purpose

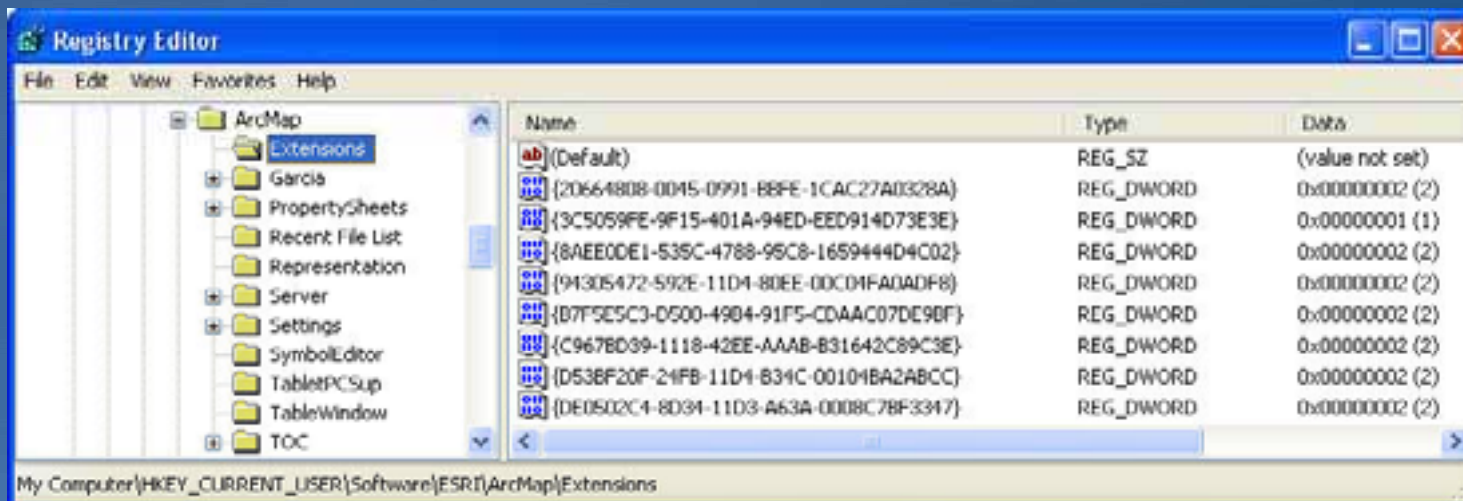




# ExtensionConfig

## Extension State

- Three different states are possible
  - esriESEnabled: License available and is checked
  - esriESDisabled: Is not checked
  - esriESUnavailable: License is not available
- Extension state is persisted in the registry
  - HKEY\_CURRENT\_USER\Software\ESRI\ArcMap\Extensions



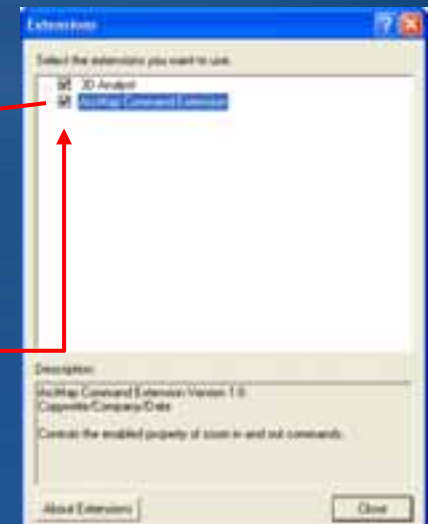
# ArcGIS Desktop Extensions

## Managing Extension State

- Synchronize checkbox state with the extension
- Store value in a member variable in the extension
- Get and set value at various times
  - User interacts with checkbox to start and shut down extension

```
Private m_iExtensionState As esriExtensionState

Public Property State() As ESRI.ArcGIS.esriSystem.esriExtensionState
Implements ESRI.ArcGIS.esriSystem.IExtensionConfig.State
    Get
        'Read the extension state
        Return m_iExtensionState
    End Get
    Set(ByVal value As ESRI.ArcGIS.esriSystem.esriExtensionState)
        'Write the extension state
        m_iExtensionState = value
    End Set
End Property
```



# ArcGIS Desktop Extensions

## Enable Commands and Tools

- Extension is checked: Enable command
- Extension is not checked: Disable command

```
' Extension class
Public Property State() As ESRI.ArcGIS.esriSystem.esriExtensionState
Implements ESRI.ArcGIS.esriSystem.IExtensionConfig.State
    Get
        'Read the extension state
        Return m_iExtensionState
    End Get
    Set(ByVal value As ESRI.ArcGIS.esriSystem.esriExtensionState)
        'Write the extension state
        m_iExtensionState = value
    End Set
End Property
```

```
' Command class
Public Overrides ReadOnly Property Enabled() As Boolean
    Get
        If (m_extensionConfig.State = esriExtensionState.esriESEnabled) Then
            Return True
        Else
            Return False
        End If
    End Get
End Property
```



# ArcGIS Desktop Extensions

## *Find Extensions from the Application*

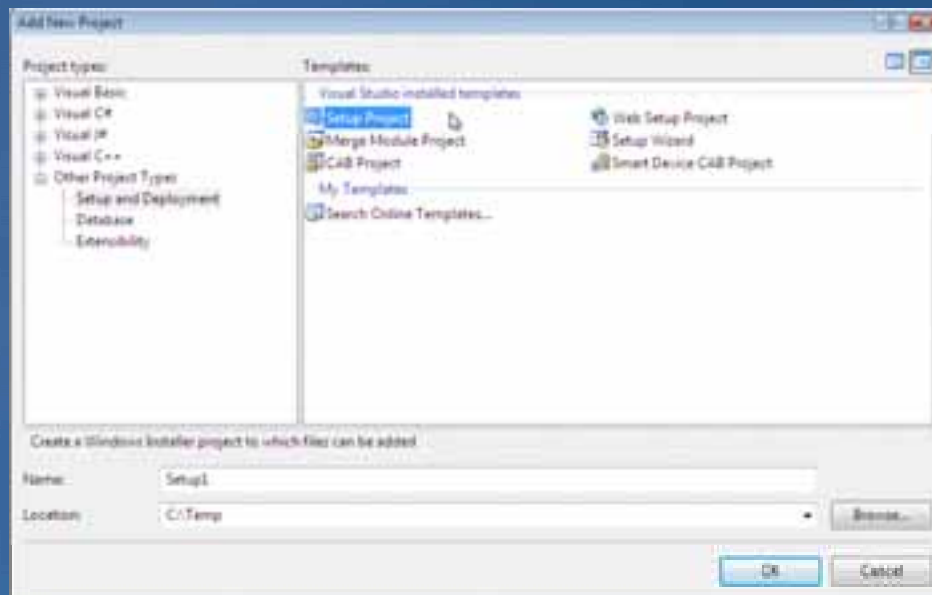
- All objects can find and reference extensions
- You can store common data in your own properties

```
'Command class
Private m_extensionConfig As IExtensionConfig
Public Overrides Sub OnCreate(ByVal hook As Object)
    If Not (hook Is Nothing) Then
        If TypeOf (hook) Is IMxApplication Then
            m_application = CType(hook, IMxApplication)
            Dim uid As New UID
            uid.Value = "myLib.customExtension"
            Dim extension As IExtension
            extension = m_application.FindExtensionByCLSID(uid) 'Find extension
            m_extensionConfig = extension 'QueryInterface
        End If
    End If
End Sub
```

# Deploying customizations

## *Visual Studio Setup project*

- Add to solution containing project to deploy
- Define what will be installed
  - Project output
  - Dependent assemblies
  - Supporting files (data, images, fonts, etc.)



*Help Topic – How to create a custom install program*

# Summary

- **Today we covered**
  - Developer installation requirements
  - Developer resources
  - ArcGIS System architecture
  - ArcGIS Desktop application framework
  - Implementing custom components

*Still have questions?*

# Additional Resources

*Questions, answers and information...*

---

- **Tech Talk**

- *Outside this room right now!*

- **Meet the Team**

- *Support Island*

- **Other sessions**

- *Migrating VBA/ VB6 Applications to .NET (1:00 3/24)*

- *Deploying ArcGIS Desktop and Engine Applications in .NET (10:15AM 3/26)*

- **ESRI Resource Centers**

- PPTs, code and video



[resources.esri.com](http://resources.esri.com)

- **Social Networking**



[www.twitter.com/  
ESRIDevSummit](http://www.twitter.com/ESRIDevSummit)



[tinyurl.com/  
ESRIDevSummitFB](http://tinyurl.com/ESRIDevSummitFB)

# Want to Learn More?

*ESRI Training and Education Resources*

- **Instructor-Led Training**

- **Extending ArcGIS Desktop Applications**

- **Introduction to Programming ArcObjects Using the Microsoft® .NET Framework**

- **Developing Applications with ArcGIS Engine Using the Microsoft .NET Framework**

*<http://www.esri.com/training>*