

定制并扩展控件

5

Copyright © 2004 ESRI. All rights reserved.

Developing Applications with ArcGIS Engine

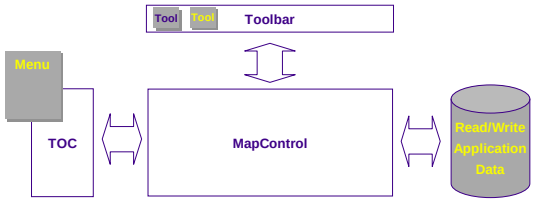
课程 5

- ToolbarControl框架
- 创建定制命令，工具和扩展
- Customize dialog（定制对话框）
- ToolbarMenus（工具栏菜单）

Copyright © 2004 ESRI. All rights reserved.

定制和扩展类型

- 许多客户化定制是可使用的
  - 在运行时准许增加或删除命令
  - 在工具栏上可定制保存且导入
  - 在运行时建立和展示菜单
- 不同的扩展也可以执行
  - 创建定制命令，工具，菜单和工具栏
  - 定制扩展



Copyright © 2004 ESRI. All rights reserved.

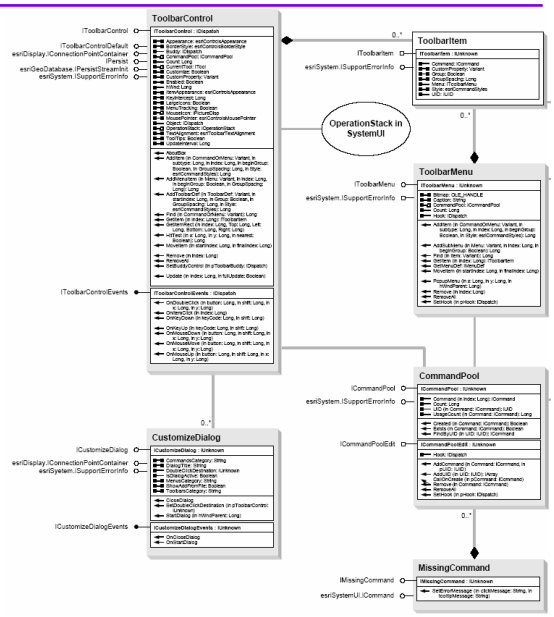
Customizations and extensions

The ToolbarControl allows for a number of customization options. The user of an ArcGIS Engine application can be given the chance to build the user interface (add and remove commands/tools) at runtime through a CustomizeDialog. It is also possible to add and arrange the tools and menus at design time. In this case, the ToolbarControl automatically saves and loads the individual control/tool objects it contains. The developer of the ArcGIS Engine application can also change and manipulate the items of the control programmatically at runtime. The ToolbarControl supports core ESRI ArcGIS tools/commands as well as customized components.

The ToolbarControl always works in conjunction with a ‘buddy control’. The ‘buddy control’ can be a MapControl, PageLayoutControl, ReaderControl, SceneControl, or GlobeControl. The ‘buddy control’ can be set at design time through the ToolbarControl property pages (in the development environments that support property page capability) or programmatically using the SetBuddyControl method. The ToolbarControl itself hosts a panel of commands, tools, tool controls and menus that work with the display of the ‘buddy control’. Imagine a ‘ZoomIn’ tool. When the ToolbarControl is buddied with a MapControl and the ‘ZoomIn’ control is clicked, then the action of the ‘ZoomIn’ functionality takes its information like the map scale and zoom factor from the currently activated buddy (i.e., the MapControl).

The ToolbarControl 框架

- ToolbarControl
  - ToolbarItems
  - CommandPool
- ToolBarMenu
  - ToolbarItems
  - CommandPool
- CustomizeDialog



Copyright © 2004 ESRI. All rights reserved.

The framework of the ToolbarControl

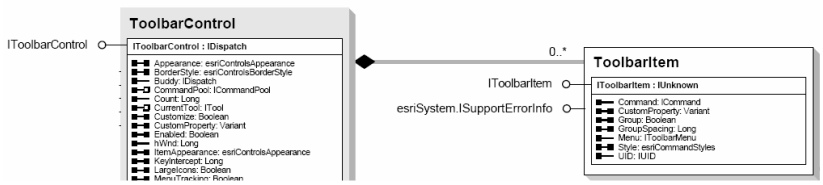
The ToolbarControl and the objects within its library can supply pieces of a framework similar to the ArcGIS Desktop application framework. The developer can use some or all of these framework pieces when building an application with the ToolbarControl. The ToolbarControl is a container object for ToolbarItems. The control itself allows for the arrangement of the commands and tools it contains (visually and with respect to the function each ToolbarItem exposes). The ToolbarControl also represents a bridge for the buddy control; that is, the ToolbarControl itself is assigned to the 'buddy control' and in return, delivers the display and environment for each ToolbarItem that it contains. A ToolbarControl can host Commands, Tools, ToolControls, and ToolbarMenus.

A ToolbarMenu is another type of container of ToolbarItems. Think of a ToolbarMenu as a logical collection of items with similar functionality or an arrangement of items for a specific functionality. A ToolbarMenu can either be hosted on the ToolbarControl or can appear as a popup menu and cannot host Tool and ToolControl objects.

The CustomizeDialog contains a modeless dialog that allows commands, toolbar definitions (toolsets), and menu definitions to be added to a ToolbarControl at runtime, when the IToolbarControl::Customize property is set to true. Because of the modeless property of the customize dialog, the ICustomizeDialog::StartDialog method launches the CustomizeDialog on the screen and returns immediately, rather than waiting until the CustomizeDialog closes. Consequently, when the CoClass goes out of scope, the CustomizeDialog will be closed. It is sensible practice to ensure the CustomizeDialog object remains alive while the container hosting the ToolbarControl is shown, by storing the CustomizeDialog object as a form (or class) level variable.

ToolbarItems

- 执行添加item到工具栏的类
- 用于管理已经存在的item
- 新建项（items） 类型
  - Commands, Tools, ToolControls和ToolbarMenus



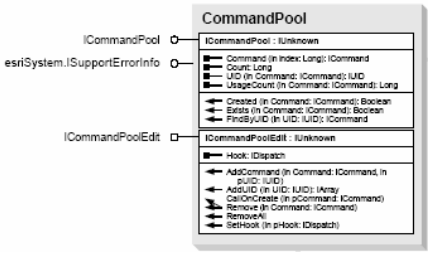
Copyright © 2004 ESRI. All rights reserved.

ToolbarItems

A ToolbarControl creates a ToolbarItem class for each item added. A ToolbarItem in return represents a Command, Tool, ToolControl, or menu item. These items generally appear as buttons (shown as a bitmap, a caption, or both), however, a ToolControl will provide its own window, and will typically be a combo box. The Command and Menu properties of the IToolbarItem interface return the actual command or menu that the ToolbarItem represents.

CommandPool

- 对所有 CommandItems 进行管理
- 仅有两个对象可能直接访问pool
  - ToolbarControl
  - ToolbarMenu
- 提供了一些低层接口项（ items）
  - ICommandPool::Created
  - ICommandPool::Exists
  - ICommandPoolEdit::CallOnCreate
  - ICommandPoolEdit::Hook
- 开发工程师一般都不使用



Copyright © 2004 ESRI. All rights reserved.

The CommandPool

Each ToolbarControl and ToolbarMenu has a CommandPool that is used to manage the collection of Command objects that it is using. Normally, as a developer, you will not interact with the CommandPool. When a command is added to the ToolbarControl, either through the property pages of the ToolbarControl or programmatically, the command is automatically added to the CommandPool.



Adding items to the CommandPool

Essentially there are two methods to add an item to a ToolbarControl.

Method 1

Pass the item as an existing or new instance of a Command object or group of commands.

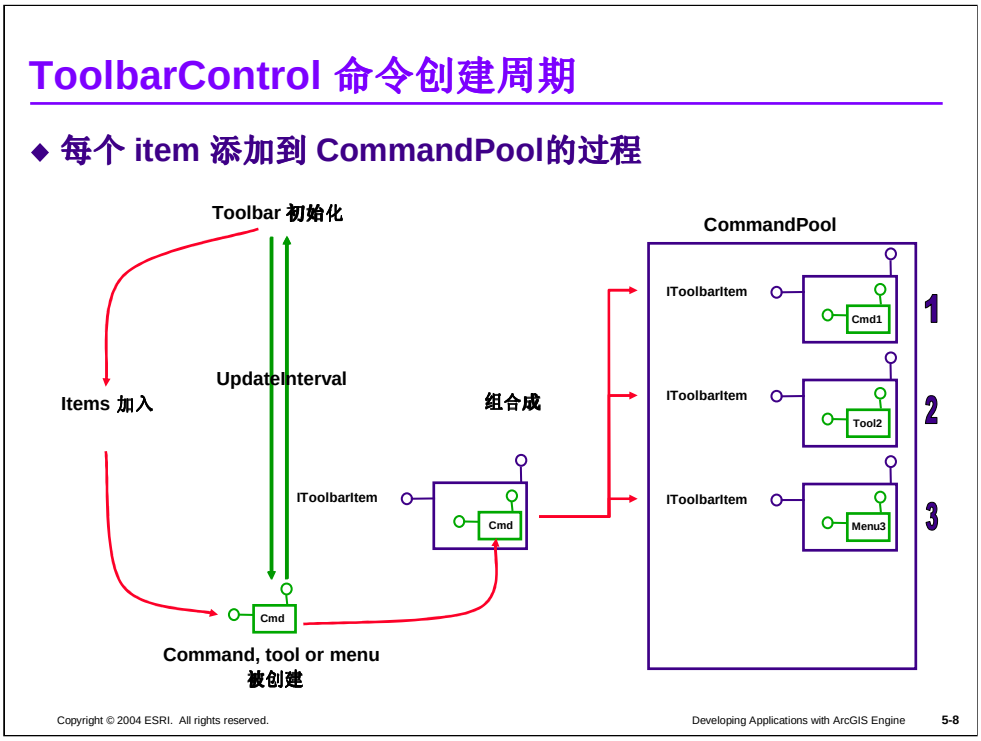
If an existing instance of a Command object is added, there is no unique identifier for the Command, and multiple instances of the same Command can exist in the CommandPool, (i.e., on the same ToolbarControl).

Method 2

Command objects are added to the CommandPool as a UID object that uniquely identifies the Command (using a GUID or a ProgID).

If a UID object is supplied, the CommandPool can identify whether the Command already exists in the CommandPool, and if so can reuse the previous instance of the Command. The CommandPool manages this by tracking whether the OnCreate method of a Command has been called. If the OnCreate method has been called, it will reuse the Command and increment its UsageCount.

If multiple ToolbarControls exist within a single ArcGIS Engine application, it is the responsibility of the developer to synchronize the different CommandPools of each ToolbarControl.

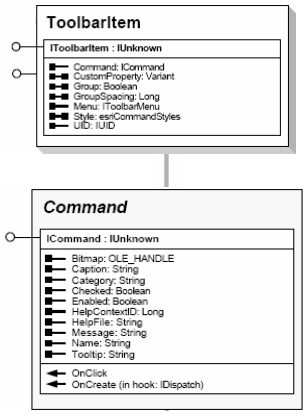


The cycle of adding and refreshing items on the ToolbarControl

The ToolbarControl is initialized first. Then the ICommand::OnCreate method is called shortly after the Command object is hosted on the ToolbarControl. The method is passed a handle or 'hook' to the application that the Command will work with. The implementation of a Command normally tests to see if the 'hook' object is supported (i.e., the Command tests to see that the 'hook' is an object that the Command can work with). If the 'hook' is supported, then a ToolbarItem is created and placed into the pool (i.e., the ToolbarItem is placed on the control). After that, the ToolbarControl updates itself automatically every half second to ensure that the appearance of each ToolbarItem hosted on the ToolbarControl is synchronized with the Enabled, Bitmap, and Caption properties of its underlying command.

ToolbarItems 和 Commands共同起作用

- ToolbarItems 包含 commands
- IToolbarItem
  - Command: 提供访问command
  - Style: 指定 command类型
  - UID: command的唯一识别ID




```
Dim toolbarItem As IToolbarItem
For i = 0 To AxToolbarControl1.Count - 1
    toolbarItem = AxToolbarControl1.GetItem(i)
          Dim command as ICommand
    command = toolbarItem.Command
    MsgBox(command.Name & toolbarItem.Style &
toolbarItem.UID.Value)
Next
```

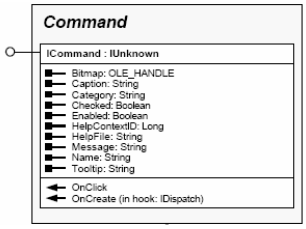
Copyright © 2004 ESRI. All rights reserved.

Working with ToolbarItems and commands

As discussed before, a ToolbarItem is a single command or menu hosted on a ToolbarControl or ToolbarMenu. The IToolbarItem interface has properties to determine the appearance of the item to the end user (e.g., whether the item has a vertical line to its left signifying that it begins a Group and whether the Style of the item displays with a bitmap, a caption, or both). The Command and Menu properties return the actual command or menu that the ToolbarItem represents.

如何运行定制命令（commands）

- 框架提供可以创建定制命令（commands）能力
- 如何无缝添加到ToolbarControl
  - 导入一个 CommandItem
  - 其外表以及行为和其他 items一样
  - 执行任何ArcObjects 代码
- 步骤
  1. 创建一个class
  2. 执行ICommand
  3. 增加类（class） 到 工具栏（toolbar）




```
Public Class ZoomToLayerClass
    Implements ICommand
```

Copyright © 2004 ESRI. All rights reserved.

Custom commands

The framework can be extended with custom commands that should behave just like the built-in ESRI tools and commands given the following prerequisites:

- The class implements the ICommand interface from the esriSystem package/namespace/library.
- The implementation is exposed to the user either through the ToolbarControl or an ToolbarMenu.
- The class is a COM-compliant (in order to appear on the ToolbarControl).



Custom command properties

Setting Properties on the Class determines the appearance of the ToolbarItem. The following are the common ICommand properties.

Name: The Name property of a command is a programmatic identifying name string. By convention, a command's name should include its category name. The format of the Name string should be <Category>_<Caption>. For example, the name of the built-in ArcMap command About ArcMap is 'Help_About'.

Caption: The Caption property of a command is the string that appears when the command is placed on a command bar.

Category: The Category determines where the command will appear in the Commands panel of the Customize dialog.

ToolTip: Use the ToolTip property to set the string that appears in the screen tip when the mouse hovers over the command.

Message: Use the Message property to set the message string that appears in the status bar of the application when the mouse passes over the command. By default ArcGIS Engine application controls do not provide a status bar. However it is a good development custom to provide this message string to custom command implementations.

Bitmap: The bitmap that is used as the icon on this command.

HelpContext: The HelpContextID property is used to specify the mapped numeric value of the topic ID of the context-sensitive help topic for your custom command. Specify the name of your help file in the HelpFile property.

HelpFile: The HelpFile property is used to specify the name of the help file that contains the context-sensitive help topic for your custom command.

定制命令 commands: 访问钩子（ hook）对象

- 对于已经创建命令的对象将返回一个钩子（ hook）
- HOOK有许多不同的对象（ objects）
  - ToolbarControl, MapControl, PageLayoutControl, ...
- 当被创建时加入 command
- ICommand
  - OnCreate

```
Private Sub ICommand_OnCreate(ByVal hook As Object)
    If (TypeOf hook Is esriToolbarControl.IToolbarControl) Then
      Dim ToolBarCon As esriToolbarControl.IToolbarControl
      Set ToolBarCon = hook
      If (TypeOf ToolBarCon.Buddy is MapControl) Then
          Dim mapControl as IMapControl
          Set MapControl = ToolBarCon.Buddy
          Set m_map = mapControl.Map
      End If
    ElseIf (TypeOf hook is IMapControl3) then
          Dim mapControl as IMapControl
          Set mapControl = hook
          Set m_map = mapControl.Map
    End If
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

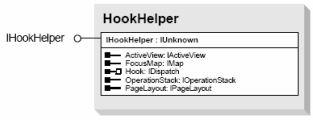
The Hook object

The ICommand::OnCreate method is called shortly after the Command object is hosted on the ToolbarControl. The method is passed a handle or ‘hook’ to the application that the Command will work with. The implementation of a Command normally tests to see if the type of ‘hook’ object is supported (i.e., the Command tests to see that the ‘hook’ is an object that the Command can work with). If the ‘hook’ is not supported, the Command custom commands are expected to disable itself or to provide proper error handling. If the ‘hook’ is supported, the Command should store the ‘hook’ for later use. For example, if an ‘Open Map Document’ Command is to work with the MapControl or PageLayoutControl and they are passed to the OnCreate method as the ‘hook’, the Command will store the ‘hook’ for later use.

If the ToolbarControl is passed to the OnCreate event as the ‘hook’, the Command would normally check the type of ‘buddy control’ being used in conjunction with the ToolbarControl using the Buddy property. For example, if a Command hosted on the ToolbarControl only works with the ReaderControl and the ToolbarControl ‘buddy’ is a MapControl, the ‘Command’ should disable itself.

定制命令（ commands）: HookHelper 类

- 提供了普通方法来访问内在对象（interr



- 简单代码（Simplifies code ）
- IHookHelper
  - ActiveView
  - FocusMap
  - Hook
  - OperationStack
  - PageLayout

```
Private m_hookHelper as IHookHelper
Private m_map as IMap
Private m_pageLayout as IPageLayout

Private Sub ICommand_OnCreate(ByVal hook As Object)

  Set m_hookHelper = New HookHelper
  Set m_hookHelper.hook = hook

  Set m_map = m_hookHelper.FocusMap
  Set m_pageLayout = m_hookHelper.PageLayout

End Sub
```

- Globe 和Scene 都拥有他们自身的钩子（ hook） helper 类

Copyright © 2004 ESRI. All rights reserved.

The HookHelper

The HookHelper is considered a convenience class designed to help developers write custom commands that will work with ArcGIS Controls and ArcGIS applications.

Use the HookHelper object to simply navigate the multiple supported environments like ArcGIS Engine, ArcGIS Desktop applications, and the extension for either engine or desktop. The HookHelper is used to hold the hook and return the IActiveView, PageLayout or FocusMap regardless of whether the hook is a MapControl, PageLayoutControl, ToolbarControl (with a ‘buddy’ MapControl or PageLayoutControl), the ArcMap application or a custom control or application implementing ICommandHook.

HookHelper: ArcMap, MapControl and PageLayoutControl applications.

SceneHookHelper: ArcScene and SceneControl applications.

GlobeHookHelper: ArcGlobe and GlobeControl applications.

定制命令（ commands）：有效性控制

- 运用其属性设置命令（command's ）是否有效
- ICommand
  - Enabled: 激活或不可用命令
  - Checked: 设置按纽的检测状态
- ToolbarControl会经常调用他们的成员
  - UpdateInterval: 定义工具栏的值

```
Private Property Get ICommand_Enabled() As Boolean
   ICommand_Enabled = IIF(TypeOf m_pHookHelper.Hook Is MapControl, True, False)
End Property

Private Property Get ICommand_Checked() As Boolean
   ICommand_Checked = IIF(TypeOf m_pHookHelper.Hook Is MapControl, True, False)
End Property
```

Copyright © 2004 ESRI. All rights reserved.

Controlling command availability with Enable and Checked

When implementing ICommand to create a custom command, add some logic to the Enabled property to specify what state the application should be in for the command to be enabled. For example, enable the tool only in conjunction with a MapControl or only enable the tool if data with a specific spatial reference is loaded in the application.

The Checked property indicates the state of this Command. If a command item appears depressed on a ToolbarControl, the command is checked. If the custom command implements this property only when implementing the ICommand interface, this property can be used in a toggle state. If the custom command implementation implements the ICommand and ITool interface, the item is checked as long as the tool is activated. As long as the item is activated, it is receiving the display and mouse events.

The ToolbarControl will query these properties frequently. The default interval is every half second. This interval can be changed by the developer using IToolbarControl::UpdateInterval.

定制命令: 基类（Base classes）

- 定制命令的基类是可用的、有价值的
- 必要使用一些成员 (比如： OnCreate)
- .NET
  - ESRI.ArcGIS.Utility.BaseClasses.BaseCommand
- Java

```
Public Class ZoomToLayerClass
    Inherits BaseCommand

   Public Sub New()
        MyBase.New()

            MyBase.m_caption = "ZoomToLayer"
            MyBase.m_category = "DAAE Commands"
            MyBase.m_message = "Zoom To the extent of the layer"
            MyBase.m_name = "DaaeCustomCommands_ZoomToLayer"
            MyBase.m_toolTip = "ZoomToLayer"
    End Sub
```

Copyright © 2004 ESRI. All rights reserved.

Using the base classes

ESRI provides two abstract base classes (BaseCommand and BaseTool) to help you create new custom commands and tools for ArcGIS in .NET. The classes are abstract classes (marked as MustInherit in Visual Basic .NET), which means that although the class may contain some implementation code, it cannot itself be instantiated directly and can only be used by being inherited by another class. Both base classes are defined in the ESRI.ArcGIS.Utility.dll assembly and belong to the ESRI.ArcGIS.Utility.BaseClasses namespace.

Base classes for the .NET languages simplify the creation of custom commands and tools by providing a default implementation for each of the members of ICommand and ITool. Instead of stubbing out each member and providing implementation code, you only have to override the members that your custom command or tool requires. The exception is ICommand::OnCreate; this member must be overridden in your derived class.

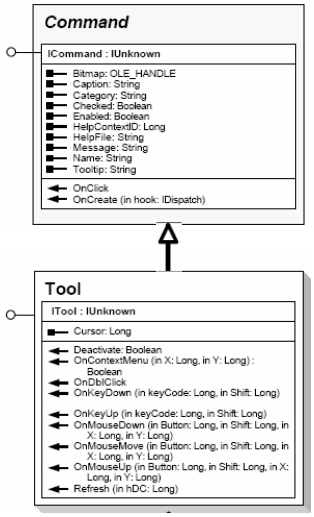
Using these base classes is the recommended way to create commands and tools for ArcGIS applications in .NET languages. You can always create similar COM classes from the original ICommand and Itool; however, using the base class technique is quicker, simpler and less error-prone.

Base classes are not available in Java at 9.0 but they will be in future releases. Currently, Java programmers can write their own version of these base classes.

实现定制工具

- 框架（ framework ）提供了创建定制工具的功能
- Added seamlessly to ToolbarControl
  - 以CommandItems加载
  - 与显示相结合
- 步骤
  1. 创建类（Create a class）
  2. 实现 ICommand 和 ITool
  3. 增加类到toolbar中

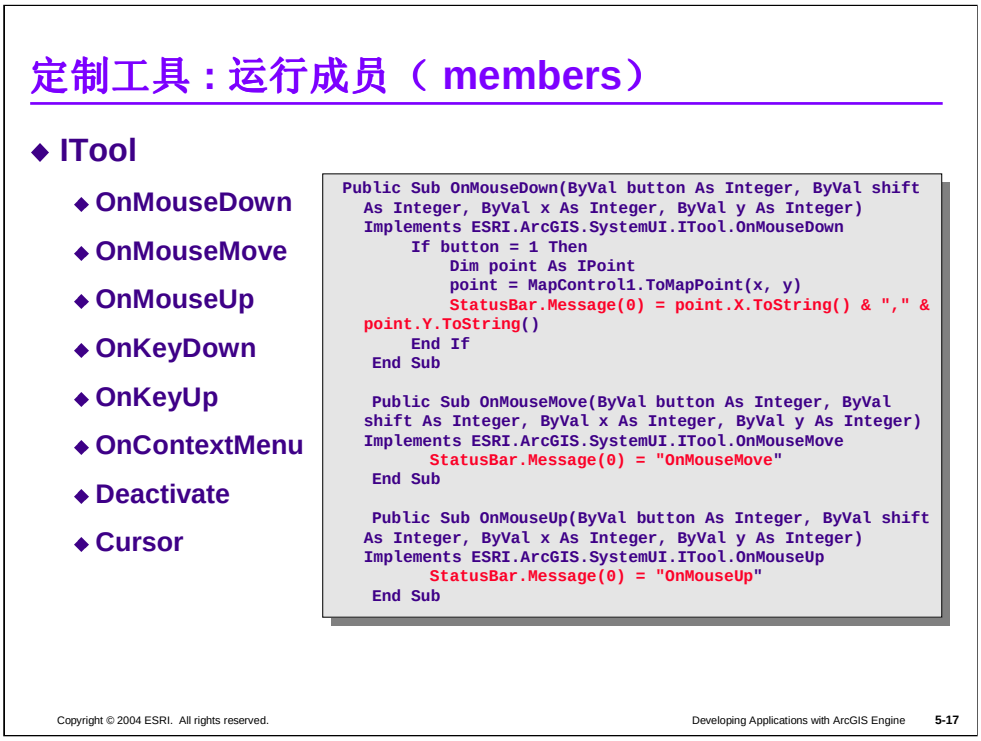
```
Public Class ZoomToLayerClass
    Implements ICommand, ITool
```



Copyright © 2004 ESRI. All rights reserved.

Custom tools

The framework allows for the implementation of custom tools. If a custom command implements the ICommand and ITool interface, the custom command is able to receive the current display and mouse events fired by the application or the control. This command will continue to receive events up to the moment that the tool is deactivated.



The ITool interface

The methods on the ITool interface provide custom functionality for the display and mouse events. Below is a list of ITool members:

OnMouseDown: Occurs when a mouse button is pressed when the tool is active. The arguments passed into the code are the mouse position in screen coordinates, which mouse button was pressed, and if the Shift key was pressed at the same time.

OnMouseMove: Occurs when the mouse is moved when the tool is active. The arguments passed into the code are the mouse position in screen coordinates, which mouse button was pressed, and if the Shift key was pressed at the same time.

OnMouseUp: Occurs when a mouse button is released when the tool is active. The arguments passed into the code are the mouse position in screen coordinates, which mouse button was pressed, and if the Shift key was pressed at the same time.

OnKeyDown: Occurs when a key on the keyboard is pressed when the tool is active. The arguments passed into the code are the key code of the corresponding key on the keyboard, and if the Shift key was pressed at the same time.

OnKeyUp: Occurs when a key on the keyboard is released when the tool is active. The arguments passed into the code are the key code of the corresponding key on the keyboard, and if the Shift key was pressed at the same time.

OnContextMenu: Context menu event occurred at the given x,y location. The position of the mouse pointer is provided in x- and y-coordinates in screen units. The custom implementation is expected to return True if the event was handled.

Deactivate: Causes the tool to no longer be the active tool. When implementing ITool to create a custom tool, add some logic to the Deactivate method to specify whether this tool can be deactivated. If ITool_Deactivate is set to False, the default, you cannot interact with any other tools because this tool cannot be interrupted by another tool.

Cursor: When implementing ITool to create a custom tool, use the Cursor property to set the mouse pointer of the tool.

定制工具 :维护选择图层

- 一些工具需要访问在TOC中的选择的图层
- 当一个图层被选择时候, 在 control中存储图层
  - 比如: IMapControl3::CustomProperty

```
Private Sub AxTOCControl1_OnMouseDown(ByVal sender As Object, ByVal e As
  ESRI.ArcGIS.TOCControl.ITOCControlEvents_OnMouseDownEvent) Handles
  AxTOCControl1.OnMouseDown

      Try
          Dim map As IMap
          Dim layer As ILayer
          Dim other As Object
          Dim item As esriTOCControlItem
          Dim index As Object
          Dim tocControl As ITOCControl

          tocControl = AxTOCControl1.Object
          tocControl.HitTest(e.x, e.y, item, map, layer, other, index)

          If item = esriTOCControlItem.esriTOCControlItemLayer Then
              AxMapControl1.CustomProperty = layer
              statusBar.Text = "Layer: " & layer.Name
...
```

Copyright © 2004 ESRI. All rights reserved.

The IMapControl3::CustomProperty

The IMapControl3 interface provides additional members for tasks related to the MapControl, such as setting a custom property and suppressing data redraw while the MapControl is being resized. The CustomProperty is used to associate any useful data with the MapControl. This is similar to a Tag property, and can be used to store strings, numbers, and objects. The code in the slide uses the CustomProperty to store selected layers from TOCControl to be used for interaction and/or analysis with the current tool.

定制工具:访问选择图层

- 定制工具通过控件（control）来访问图层

```
Public Overrides Sub OnCreate(ByVal hook As Object)
    'Create an IHookHelper object
    m_hookHelper = New HookHelper
    m_hookHelper.Hook = hook
    If TypeOf m_hookHelper.Hook Is MapControl Then
        m_mapControl3 = m_hookHelper.Hook
    ElseIf TypeOf m_hookHelper.Hook Is ToolbarControl Then
        Dim toolbarControl As IToolbarControl
        toolbarControl = m_hookHelper.Hook
        m_mapControl3 = toolbarControl.Buddy
    End If
End Sub

Public Overrides Sub OnClick()
    Dim layer As ILayer
    layer = m_mapControl3.CustomProperty
    If (TypeOf layer Is IGeoDataset) Then
        Dim pGeoDataset As IGeoDataset
        pGeoDataset = layer
        m_hookHelper.ActiveView.Extent = pGeoDataset.Extent
        m_hookHelper.ActiveView.Refresh()
    End If
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

Accessing the CustomProperty

The MapControl associated with the ToolbarControl can be accessed thru the Buddy property (IToobarControl::Buddy). So, every time the OnClick event of the command is fired, the layer selected in the TOCControl is provided by IMapControl::CustomProperty.

实现定制 的COM 组件

- 框架也可以通过自己写的COM 组件被扩展
- 类型
  - Commands, tools, menus, toolbars, 和扩展
- 一般步骤
  1. 创建一个工程 (VB, VC++, .NET, …)
  2. 定义库的类型并且定义类的名称
  3. 引用适当的ArcGIS Engine 库
  4. 实现用户界面
  5. 编译和注册
  6. 注册成一个组件类型（ component category）

Copyright © 2004 ESRI. All rights reserved.

Creating custom COM components

The steps for creating a custom COM component are:

1. Create a COM project, in the COM-compliant language of choice.

2. Create a COM class within this project.

3. Make certain that the appropriate ArcGIS Engine libraries are referenced. If these libraries (e.g., esriControlCommands) are not referenced, the access to ArcObjects object will not be possible.

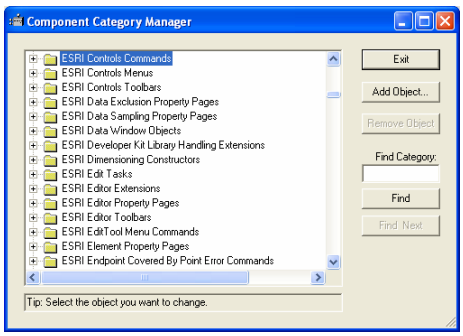
4. Implement an interface by stubbing out all the members, and writing the appropriate implementation for each of these members.

5. After the implementation is written out, compile the component with the operating system. Typically, a DLL binary file is created with an associated GUID.

6. After registering the component with the operating system, this component needs to be registered with the ArcGIS Engine. This is done by placing the already registered component in its respective component category.

存储定制COM components

- 所有定制COM组件都正确的ArcGIS Engine component category中注册
- 是对于CustomizeDialog的类的可用的
  - ESRI Controls Commands
  - ESRI Controls Menus
  - ESRI Controls Toolbars
- 注册的方法
  - Categories.exe
  - ESRI Add-ins
  - Registration scripts (.reg)



Copyright © 2004 ESRI. All rights reserved.

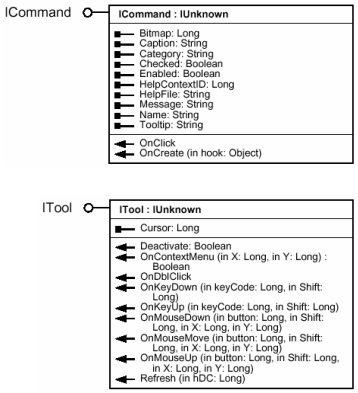
Custom COM component category registration

Component categories are just folders in the registry. These folders organize how the components are filed. ArcGIS Engine controls (the ToolbarControl at 9.0) make use of these component categories by reading them upon loading. Based upon where the components are placed, (assuming that they are placed in the correct category with the correct implementation), the component will be loaded accordingly. Component categories are responsible for both organization and optimization—organization for grouping the components and optimization because the application can quickly grab what it needs and based upon the category, know where and how it needs to be loaded.

The ESRI Component Category Manager is a tool providing read/write access into the component categories. You can also modify them via the regedit utility; however, the developer tool is a bit more user-friendly. Microsoft Visual Studio's OleView tool is also very helpful. It is similar to a Windows Explorer for COM components. It also breaks the components down based upon their category. This utility is strictly for browsing and reading component information.

运行 COM 命令和工具组

- ESRI命令之后建模
- 接口: Icommand和 ITool
- 库类型: esriSystemUI
- 组件分类
  - ESRI Controls Commands
- 另外有关的接口和对象
  - ToolbarItem
  - IToolbarDef 和 IMenuDef
- 其他方面详细资料和本地的类及COM类是一样的



Copyright © 2004 ESRI. All rights reserved.

Implementing COM commands and tools

Similar to native classes, COM-based custom commands and tools are created by implementing ICommand and ITool. The only difference is that COM components need to be registered with the system and under the appropriate component category. The ArcGIS Engine component category used for commands and tools is named “ESRI Controls Commands”.

双重 COM 命令

- 你可以创建 COM 命令运用在 ArcGIS Engine 和 Desktop
- 那么需要运用HookHelper类来连接、取得这个钩子
- 做 Engine开发的机器上不能引用Desktop的类

```
Private Sub ICommand_OnCreate(ByVal hook As Object)
    If TypeOf hook Is esriToolbarControl.IToolbarControl Then
      Dim ToolBarCon As esriToolbarControl.IToolbarControl
      Set ToolBarCon = hook
      If (TypeOf ToolBarCon.Buddy is MapControl) Then
          Dim mapControl as IMapControl
          Set MapControl = ToolBarCon.Buddy
          Set m_map = mapControl.Map
      End If
    ElseIf (TypeOf hook is IMxApplication) then
        Dim application as IApplication
        Set application = hook
        Dim mxDoc as IMxDocument
        Set mxDoc = application.Document
        Set m_map = mxDoc.FocusMap
    End If
End Sub
```

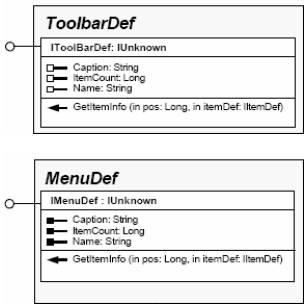
Copyright © 2004 ESRI. All rights reserved.

Dual-Purpose COM commands

As discussed previously, the HookHelper object is most useful when the custom command is to support more than one type of hook objects. So, if the custom command makes use of desktop objects that are only present in an ArcGIS Desktop installation, such as IEditor and IMxApplication, it will not work on a machine where only ArcGIS Engine is installed. One possible solution is to write dual-commands that runs under both ArcGIS Desktop and ArcGIS Engine environments. In this case, the dual-commands logic should test whether the hook object is a type of ArcGIS Control, and if so, provide an alternative implementation for an ArcGIS Engine only installation.

运行COM工具栏和 menus

- ESRI toolbars 和 menus之后建模
- 接口: IToolBarDef, IMenuDef
- 库的种类: esriSystemUI
- 组件分类:
  - ESRI Controls Menus
  - ESRI Controls Toolbars
- 另外有关系的 接口和对象
  - IMenuDef and IToolbarMenu
  - ICommandPool, Icommand, and ITool



Copyright © 2004 ESRI. All rights reserved.

COM toolbars and menus

When creating a custom toolbar, the IToolbarDef interface should be implemented in the class code. IToolBarDef properties allow setting the caption and name of the toolbar as well as specifying what command items exist on the toolbar.

A ToolbarMenu supplies the implementation of a menu item that can host commands and submenus but cannot host tools and ToolControl objects. A ToolbarMenu can either be hosted on the ToolbarControl or can appear as a popup menu. The ToolbarMenu can be built in two ways:

- By passing an object implementing IMenuDef to the IToolbarControl::AddItem or IToolbarControl::AddMenuItem methods.
- By using the properties and methods on the IToolbarMenu interface.

Submenus on a ToolbarMenu are implemented as another instance of IToolbarMenu.

Toolbar and Menu components are registered under the categories “ESRI Controls Toolbars” and “ESRI Controls Menus” respectively.

创建定制COM工具栏

```
Implements IToolbarDef

Private Property Get IToolbarDef_Caption() As String
  IToolbarDef_Caption = "Navigation"
End Property

Private Sub IToolbarDef_GetItemInfo(ByVal pos As Long, ByVal itemDef As
   esriSystemUI.IItemDef)
  Select Case pos 'Commands for the menu
    Case 0
      itemDef.ID = "esriControlCommands.ControlsMapZoomInFixedCommand"
    Case 1
      itemDef.ID = "esriControlCommands.ControlsMapZoomOutFixedCommand"
    Case 2
      itemDef.ID = "esriControlCommands.ControlsMapFullExtentCommand"
    Case 3
      itemDef.ID = "MyLibrary.MyCommand"
    End Select
End Sub

Private Property Get IToolbarDef_ItemCount() As Long
  IToolbarDef_ItemCount = 3
End Property

Private Property Get IToolbarDef_Name() As String
  IToolbarDef_Name = "Navigation"
End Property
```

Copyright © 2004 ESRI. All rights reserved.

Custom toolbars

The code in the slide shows how to create a custom toolbar by implementing IToolBarDef. The following are the properties of the IToolbarDef interface:

Caption: The string that appears as the toolbar title when the toolbar is in a floating state.

GetItemInfo: The CLSID for the item on this toolbar at the specified index.

ItemCount: The number of items in this toolbar.

Name: The name of this toolbar.

Note that the fourth command added to the toolbar is a custom command "MyLibrary.MyCommand", while the first three are built-in ArcGIS Engine commands.

创建定制COM菜单

```
Implements IMenuDef

Private Property Get IMenuDef_Caption() As String
  IMenuDef_Caption = "Navigation"
End Property

Private Sub IMenuDef_GetItemInfo(ByVal pos As Long, ByVal itemDef As
   esriSystemUI.IItemDef)
  Select Case pos 'Commands for the menu
    Case 0
      itemDef.ID = "esriControlCommands.ControlsMapZoomInFixedCommand"
    Case 1
      itemDef.ID = "esriControlCommands.ControlsMapZoomOutFixedCommand"
    Case 2
      itemDef.ID = "esriControlCommands.ControlsMapFullExtentCommand"
    Case 3
      itemDef.ID = "MyLibrary.MyCommand"
  End Select
End Sub

Private Property Get IMenuDef_ItemCount() As Long
  IMenuDef_ItemCount = 3
End Property

Private Property Get IMenuDef_Name() As String
  IMenuDef_Name = "Navigation"
End Property
```

Copyright © 2004 ESRI. All rights reserved.

Custom menus

The code in the slide shows how to create a custom toolbar by implementing IMenuDef. The following are the properties of the IMenuDef interface:

Caption: The string that appears as the menu title.

GetItemInfo: The CLSID for the item on this menu at the specified index.

ItemCount: The number of items in this menu.

Name: The name of this menu.

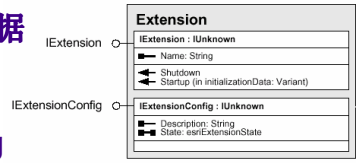
Note that the fourth command added to the menu is a custom command "MyLibrary.MyCommand", while the first three are built-in ArcGIS Engine commands.

If you want your custom menu to appear in the Menus category in the Customize dialog, you also need to implement the IRootLevelMenu interface. IRootLevelMenu is an indicator interface that is used only to indicate to the application that the menu should be treated as a root menu.

If you are creating a context menu, you would implement both IMenuDef and IShortcutMenu. IShortcutMenu is an indicator interface that is used only to indicate to the application that this menu should be treated as a context menu.

(implements)实现 COM 扩展

- 在软件组件之间的扩展之间能共享数据
- ESRI扩展模型
- 接口: IExtension, IExtensionConfig
- 引用库: esriSystem
- 详细资料
  - ExtensionManager 是一个单个、独立
  - 任何一个组件进程Extension 类是可用的
  - 资源容易共享
  - 比如: ZoomExtension




```
Public Class ZoomExtension
   Implements IExtension
   Implements IExtensionConfig
```

Copyright © 2004 ESRI. All rights reserved.

Implementing COM extensions

Extensions in ArcGIS Engine can be modeled after ESRI ArcGIS Desktop Extensions. An extension can provide many different types of functionality such as deliver controls and toolbars, perform event handling, store data commonly shared between controls, perform validation, and much more.

Extensions act as a central point of reference for developers when they are creating commands and tools for use within the applications. Often these commands and tools must share data or access common UI components. An extension is the logical place to store this data and develop the UI components. The main reason for this is that there is only ever one instance of an extension per running application and it is always possible to locate the extension and work with it.

Extension components are created by implementing two interfaces: IExtension and/or IExtensionConfig.

COM 扩展工具: IExtension

- 名称: 查找扩展
- 启动:通过程序初始化 data
- Shutdown: （停止、结束）Terminate

```
Dim m_Form as Form

Private ReadOnly Property IExtension_Name() As String Implements
ESRI.ArcGIS.esriSystem.IExtension.Name
    Get
        Return "Zoom Factor Extension"
    End Get
End Property

Private Sub IExtension_Startup(ByRef initializationData As Object) Implements
ESRI.ArcGIS.esriSystem.IExtension.Startup
      m_Form = initializationData
      m_zoomFactor = 2
      m_extensionState = esriExtensionState.esriESDisabled
End Sub

Private Sub IExtension_Shutdown() Implements
ESRI.ArcGIS.esriSystem.IExtension.Shutdown
    'Not implemented
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

Implementing IExtension

The IExtension interface allows you to set the name of the extension and specifying what action takes place when the extension is started or shutdown.

Name: The name of the extension.

Shutdown: Shuts down the extension.

Startup: Starts up the extension with the given initialization data.

When IExtensionManager::FindExtensionByName is called, the Name property will be used to identify the extension (this will *not* be the name which is displayed in the Extensions dialog box). The Startup method is used to perform some action when the extension gets loaded. This method has a parameter called initializationData, which is a reference to the object with which this extension is registered. The Shutdown method is used to perform some action when the extension gets unloaded.

COM 扩展接口: IExtensionConfig

- IExtensionConfig: 控制控件状态
- 定制界面用于数据进行访问
  - 比如: ZoomFactor

```
Implements IExtensionConfig
…

Private Property Let IExtensionConfig_State(ByVal RHS As
   esriSystem.esriExtensionState)
  m_extensionState = RHS
End Property

Private Property Get IExtensionConfig_State() As esriSystem.esriExtensionState
  IExtensionConfig_State = m_extensionState
End Property

' Custom interface and property to get and set ZoomFactor
Private Property Let IZoomExtension_ZoomFactor(ZoomFactor As Double)
  m_zoomFactor = ZoomFactor
End Property

Private Property Get IZoomExtension_ZoomFactor() As Double
    IZoomExtension_ZoomFactor = m_zoomFactor
End Property
```

Copyright © 2004 ESRI. All rights reserved.

Implementing IExtensionConfig

To list a custom extension in a dialog box, the IExtensionConfig interface needs to be implemented. The IExtensionConfig interface is used to provide the Extension dialog with the name of the extension, a description of the extension, and the state of the extension.

Description: Detailed description of the extension.

ProductName: Name of the extension.

State: The state of the extension.

The esriExtensionState enumeration is used to specify whether the extension is enabled, disabled, or unavailable. The state of the extensions is user-based. When an extension is installed, its default state is unchecked (esriESDisabled) and the user must knowingly check the extension on in the Extensions dialog box.

With a custom extension, the programmer has full control over what happens when the extension is turned on or off. However, it is a good idea to follow the same design as the ArcGIS extensions.



Accessing extensions

The ExtensionManager is a singleton designed to load and enable existing extensions. The ExtensionManager CoClass supports the following Interface:

IExtensionManager: Provides access to members that query extensions.

IExtensionManagerAdmin: Provides access to members that give life to the extensions.

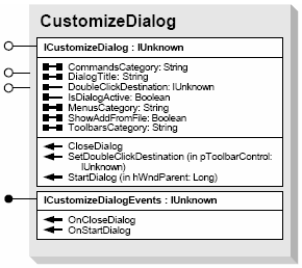
IJIExtensionManager : Provides access to the Just In Time Extension Manager.

ISupporterrorInfo: Indicates whether a specific interface can return Automation error objects.

To load an extension, the CLSID of this extension and the InitializationData (Variant) are passed to the method IExtensionManagerAdmin::AddExtension. In the code above, ZoomFactorExtension.ZoomExtension is passed as the CLSID, and Form1 is passed as the InitializationData.

关于 CustomizeDialog

- 用户可创建非模态的对话框:
  - 在运行时可增加并删除命令在 toolbarcontrol 中
  - 浏览ESRI已经存在commands, tools, 菜单, 和工具栏
  - 浏览定制的commands, tools, menus, 和 toolbars
  - 拖放选项（ items）
  - 右鍵删除
- 需要激活程序



Copyright © 2004 ESRI. All rights reserved.

The CustomizeDialog

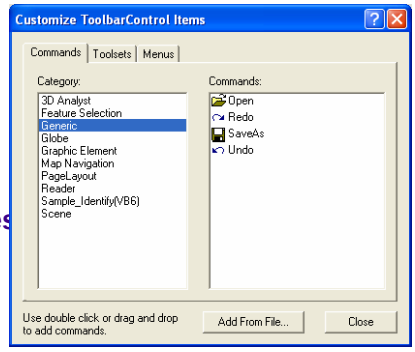
When the ToolbarControl is in customize mode the developer can programmatically launch the modeless CustomizeDialog. The CustomizeDialog lists all of the Controls Commands together with any custom commands, toolsets, and menus. It does this by reading entries from the ‘ESRI Controls Commands’, ESRI Controls Toolbars’ and ‘ESRI Controls Menus’ component categories. If required, the developer can change the CustomizeDialog to use alternative component categories.

The end user can add these commands, toolsets, and menus to the ToolbarControl either by dragging and dropping them onto the ToolbarControl or double clicking on them. Within the Java environment, only the double-click will work to place the command on the toolbar and the right-click on the individual item to remove it.

The CustomizeDialog is modeless to allow the user to interact with the ToolbarControl. When the CustomizeDialog is launched with the StartDialog method, the method call returns immediately while the CustomizeDialog remains open on the screen. In order to keep a reference to the CustomizeDialog while it is open, it is sensible practice to store a class level variable to the CustomizeDialog and to listen to its ICustomizeDialogEvents.

从组件种类导入选项（ items）

- 所有子项（items）必须正确注册在 category
- Tabs map to 组件种类（ category）名称
  - Commands = ESRI Controls Command
  - Toolsets = ESRI Controls Toolbars
  - Menus = ESRI Controls Menus
- 通过文件增加
  - 浏览 COM DLLs
  - 自动注册 DLL
  - 更新组件种类 component categories



Copyright © 2004 ESRI. All rights reserved.

Loading items from the component categories

The component categories that are searched by default are:

- ESRI Controls Command
- ESRI Controls Toolbars
- ESRI Controls Menus

Custom component category is accessible through ICustomizeDialog::ToolbarsCategory. Optionally, the user can browse for additional custom commands/tools through the “Add From File…” button. It is expected that the DLL is registered with the OS. In this case, the tool will automatically be added to the existing component categories and be ready to use within the framework.

显示定制对话框（ customize dialog）

- 创建对话框
- 接收事件（Sink events）
- 设置状态（Set state）

```
Private WithEvents pCustDialogEvents As CustomizeDialog
Private pCustDialog As ICustomizeDialog

Private Sub ToolbarControl1_OnMouseDown(ByVal button As
   Long, ByVal shift As Long, ByVal x As Long, ByVal y As
   Long)
   If (button = vbRightButton) Then
      Set pCustDialog = New CustomizeDialog
      pCustDialog.SetDoubleClickDestination ToolbarControl1
      pCustDialog.DialogTitle = "Customize"
      pCustDialog.StartDialog ToolbarControl1.hWnd
      Set pCustDialogEvents = pCustDialog
   End If
End Sub

Private Sub pCustDialogEvents_OnCloseDialog()
  ToolbarControl1.Customize = False
End Sub

Private Sub pCustDialogEvents_OnStartDialog()
  ToolbarControl1.Customize = True
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

Showing the customize dialog

The code above shows how a new instance of the CoClass CustomizeDialog can be created and started when the user right-clicks on the ToolBarControl.

The events ICustomizeDialogEvents::OnStartDialog and ICustomizeDialogEvents::OnCloseDialog are then used to toggle the Customize property of the ToolbarControl.

Setting the IToolbarControl::Customize property to True allows the following:

- Use the left mouse button to drag the selected item to a new position on the ToolbarControl.
- Use the left mouse button to drag and drop the item off the ToolbarControl to remove it.
- Use the right mouse button to select an item and display the Customize menu. Then use the Customize menu to either delete the item or set its Style, Group, or GroupSpacing properties.

保存 toolbar 项

- 当程序关闭时它们都可以把所有在toolbar的项（items）都写入到文件中
- 为用户保存当前状态
- 对于不同的 CommandItem, store the:
  - Order
  - Style
  - UID

```
Private Sub WriteTextFile(ByVal WriteFileName As String)
        Dim objStreamWriter As StreamWriter
        Dim i As Integer
        Dim toolbarItem As IToolbarItem
        objStreamWriter = New StreamWriter(WriteFileName)
        For i = 0 To AxToolbarControl1.Count - 1
            toolbarItem = AxToolbarControl1.GetItem(i)
            If Not toolbarItem.UID Is Nothing Then
                objStreamWriter.WriteLine(toolbarItem.Style)
                objStreamWriter.WriteLine(toolbarItem.UID.Value)
            End If
        Next
        objStreamWriter.Close()
    End Sub
```

Copyright © 2004 ESRI. All rights reserved.

Saving the toolbar items

As discussed previously, ToolbarItems can be added and removed from the toolbar at runtime of the ArcGIS Engine application. It is the responsibility of the developer to provide the option of persisting the user changes at runtime (as opposed to defining the toolbar items at design time). One way to persist the currently selected ToolbarItems is to loop through all of them before the application closes and store the following information:

- Position of the ToolbarItem
- Style of the ToolbarItem (Bitmap, caption, or both)
- The UID of the ToolbarItem to uniquely identify the tool/command/menu

读取 toolbar 项（items）

- 当程序启动时自动读取所有项（items）从外部文件中
- 对于不同的 CommandItem, 重建他们:
  - Order
  - Style
  - UID

```
Private Sub ReadTextFile(ByVal ReadFileName As String)
        Dim objStreamReader As StreamReader
        Dim strLine As String
        Dim strStyle As String
        objStreamReader = New StreamReader(ReadFileName)
        strStyle = objStreamReader.ReadLine
        Do While Not strStyle Is Nothing
            If strStyle <> "" Then ' Read each item and apply the style
                strLine = objStreamReader.ReadLine
      AxToolbarControl1.AddItem(strLine, , -1, False, , CInt(strStyle))
            End If
            strStyle = objStreamReader.ReadLine
        Loop
        objStreamReader.Close()
    End Sub
```

Copyright © 2004 ESRI. All rights reserved.

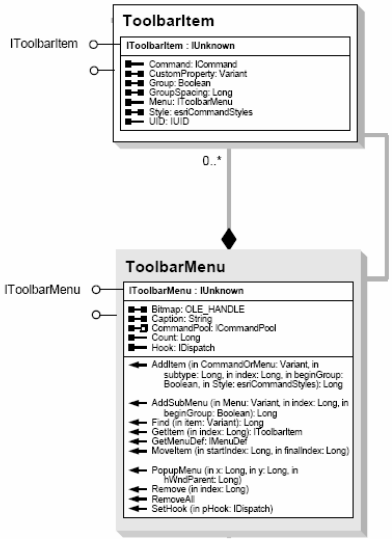
Reading the toolbar items

In order to recreate the state of the application's ToolbarControl, the file that was previously generated is read at the startup of the application. When the visual components are loaded, the items of the ToolbarControl are rehydrated by looping through the following saved settings:

- Position of the toolbaritem
- Style of the ToolbarItem (Bitmap, caption, or both)
- The UID of the ToolbarItem to uniquely identify the tool/command/menu

与 ToolbarMenus共同起作用

- 运用ToolbarMenu to host commands and menus
- 二种方法创建
  - 加入菜单到 ToolbarControl
  - 手动移动命令
- 多样种类
  - Pop-up menu
  - Toolbar menu
  - Submenu



Copyright © 2004 ESRI. All rights reserved.

Working with ToolbarMenus

A ToolbarMenu item presents a vertical list of single-click command items (ITool and IToolControl implementation are not supported). The user must select one of the command items on the ToolbarMenu, or click outside of the ToolbarMenu to make it disappear. The ToolbarMenu itself can be either hosted on the ToolbarControl, hosted on another ToolbarMenu as a submenu, or it can appear as a 'popup menu' and used for a right-click context menu.

As stated before, a ToolbarMenu can be built in two ways:

- An object implementing IMenuDef can be passed to the IToolbarControl::AddItem or IToolbarControl::AddMenuItem method which creates a ToolbarMenu object and populates it from the menu definition.
- The properties and methods on the IToolbarMenu interface can be used to build a menu.

在MapControl 中显示ToolbarMenus

- 新的 menus 要求一个钩子（hook）传递进来
- Hook 是存在ToolbarControl 或任何其他 control

```
Private m_pToolbarMenu As IToolbarMenu

Private Sub Form_Load()

  'Create a MenuDef object
  Dim pMenuDef As esriSystemUI.IMenuDef
  Set pMenuDef = New NavigationMenu
  Set m_pToolbarMenu = New ToolbarMenu
  m_pToolbarMenu.AddItem pMenuDef
  m_pToolbarMenu.SetHook ToolbarControl1.Object

End Sub

Private Sub MapControl1_OnMouseDown(ByVal button As Long, ByVal shift As Long, ByVal x
   As Long, ByVal y As Long, ByVal mapX As Double, ByVal mapY As Double)
  If button = vbRightButton Then
    'Popup the menu
    m_pToolbarMenu.PopupMenu x, y, MapControl1.hWnd
  End If
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

Showing ToolbarMenus in the MapControl

The hook is a shortcut to the ICommandPoolEdit::Hook property. If the ToolbarMenu is an item on the ToolbarControl, the hook is shared and initialized by the ToolbarControl. If the ToolbarMenu is a stand-alone popup menu, the IToolbarMenu::SetHook method is used to set the object that is passed as a hook to the OnCreate event of each item's command. The hooked object can be the ToolbarControl itself or any of the other controls. The code above sets the ToolbarControl as the hook for the ToolbarMenu and passes the MapControl handle to IToolbarMenu::PopupMenu in the OnMouseDown event.

在TOCControl中显示 ToolbarMenus

- 方法相似

```
Private Sub TOCControl1_OnMouseDown(ByVal button As Long, ByVal shift As
   Long, ByVal x As Long, ByVal y As Long)
  If (button = vbRightButton) Then
    Dim pMap As IMap
    Dim pLayer As ILayer
    Dim pOther As IUnknown
    Dim pItem As esriTOCControlItem
    Dim pIndex As Variant

   TOCControl1.HitTest x, y, pItem, pMap, pLayer, pOther, pIndex

    If pItem = esriTOCControlItemLayer Then
      Dim pToolbarMenu As IToolbarMenu
        Set pToolbarMenu = New ToolbarMenu
        pToolbarMenu.AddItem New ControlsMapViewMenu
        pToolbarMenu.SetHook MapControl1.Object
        pToolbarMenu.PopupMenu x, y, TOCControl1.hWnd
    End If
  End If
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

Showing ToolbarMenus in the TOCControl

The code above shows similar functionality to the previous slide, with the main difference of passing in TOCControl handle ToolBarMenu::PopupMenu method.

增加子菜单submenus到存在的ToolbarMenus

- 找到已经存在的或者创建一个新的ToolbarMenu
- 增加一个子菜单到ToolbarMenu

```
Private Sub cmdAddSubMenu_Click()

  'Create a MenuDef object
  Dim pMenuDef As esriSystemUI.IMenuDef
  Set pMenuDef = New ToolbarSubMenu

  'Get the menu, which is the 6th item on the toolbar (indexing from 0)
  Dim pToolbarItem As IToolbarItem
  Set pToolbarItem = ToolbarControl1.GetItem(5)
  Dim pToolbarMenu As IToolbarMenu
  Set pToolbarMenu = pToolbarItem.Menu

  'Add the sub-menu as the third item on the Navigation menu, making it
  'start a new group
  pToolbarMenu.AddSubMenu pMenuDef, 2, True

  cmdAddSubMenu.Enabled = False
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

Adding submenus to existing ToolbarMenus

Submenus are built-in or newly created menus added as an item to an existing menu in the application. Below are the steps followed in the code above to add a submenu in the 3rd position of the ToolBarMenu located at the 6th position on the ToolbarControl.

1. Get a specific ToolbarItem from the ToolbarControl (Should be a ToolbarMenu).
2. QI for the IToolBarMenu interface.
3. Add a new submenu to the currently selected item (6th item) of the ToolbarControl to create a new submenu on the third item entry starting a new group.

Exercise 5 overview

- Enable the CustomizeDialog
- Save and load items to a file
- Create a custom command to zoom to a layer
- Use the CustomProperty member of the controls
- Create a toolbar menu
- Work with a COM command

Copyright © 2004 ESRI. All rights reserved.

Lesson 5 review

- Explain how the CustomizeDialog works?
- How can you extend the Engine framework?
- What is the HookHelper used for?
- Explain the ToolbarItem load cycle.
- What are the advantages and disadvantages between implementing native and COM commands?
- What class do you use to show menus and sub-menus?

Copyright © 2004 ESRI. All rights reserved.