```
Private Sub VBMagic()
  ' Dim a variable to the IUnknown interface on the simple object.
  Dim pUnk As Iunknown

  ' Co Create simpleobject asking for the IUnknown interface.
  Set pUnk = New SimpleObject 'refCount = 1

  ' QI for a useful interface.
  ' Define the interface.
  Dim pMagic As IsimpleObject

  ' Perform the QI operation
  Set pMagic = punk 'refCount = 2

  ' Dim another variable to hold another interface on the object.
  Dim pMagic2 As IanotherInterface

  ' QI for that interface
  Set pMagic2 = pMagic 'refCount = 3

  ' Release the interface pointer.
  Set pMagic2 = Nothing 'refCount = 2

  ' Release the interface.
  Set pMagic = Nothing 'refCount = 1

  ' Now reuse the pUnk variable - what will VB do for this?
  Set pUnk = New SimpleObject 'refCount = 1, then 0, then 1

  ' Let the interface variable go out of scope and let VB tidy up.
End Sub 'refCount = 0
```

查看磁盘上 Visual Basic Magic 示例来查找这段代码。鼓励运行示例并逐步通过代码。该对象也使用一个 ATL C++工程来定义 SimpleObject 及其接口；鼓励仔细研究这段代码以学习 C++ ATL 对象的简单实现。

接口通常有一些实际上是指向其它接口的指针的属性。Visual Basic 允许通过链接这些接口以一种快速（shorthand）方式访问这些属性。例如，假设有一个指向*IFoo*接口的指针，而该接口有一个名为*Gak*的属性，该属性是一个具有*DoSomething()*方法的*IGak*接口。可以选择如何访问*DoSomething*方法。第一种方法是一个慢速（long-handed）方法。

```
Dim pGak as IGak
Set pGak = pFoo 'Assign IGak interface to local variable.
pGak.DoSomething 'Call method on IGak interface.
```

也可以用一行代码链接接口并完成同样的任务。

```
pFoo.Gak.DoSomething 'Call method on IGak interface.
```

当仔细研究示例代码时会看到这两种方法。通常前一种方法用在比较简单的示例中，因为这种方法明确告诉使用的是什么接口。更复杂一些的示例使用快速方法。

将接口链接起来的技术常常被用来获得一个属性值，但不能始终用来设置一个属性值。只有当链中的所有接口是通过引用设置的，才可以使用接口链接来设置属性值。例如，下面的代码会成功执行。

```
Dim pMxDoc As ImxDocument
Set pMxDoc = ThisDocument
pMxDoc.FocusMap.Layers(0).Name = "Foo"
```

上面的这个例子可以正确执行，因为*Map*中的*Layer*和文档中的*Map*都是以引用返回的。下面这行代码就不能正确执行，因为*Extent*外接矩形在活动视图中是以值设置的。

```
pMxDoc.ActiveView.Extent.Width = 32
```

上面这样不能正常运行的原因是VBVM为了获得终端属性而扩展了接口链。因为链中的接口是通过值来处理的，VBVM拥有的是变量自身的副本，而不是被链接的变量。为了设置上面例子中范围外接矩形的*Width属性*，VBVM必须象下面这样编写代码：

```
Dim pActiveView as IActiveView
Set pActiveView = pMxDoc.ActiveView

Dim pEnv as IEnvelope
Set pEnv = pActiveView.Extent ' This is a get by value,

PEnv.Width = 32 ' The VBVM has set its copy of the Extent and not
                ' the copy inside the ActiveView
```

为了使上面代码正常运作，VBVM额外要求下面一行代码。

```
pActiveView.Extent = pEnv ' This is a set by value,
```

访问ArcObjects

现在将看一些涉及ArcObjects的创建实例和查询接口操作的特殊用法。为在Visual Basic 或VBA中使用ArcGIS对象，首先必须引用包含该对象的ESRI库。如果在ArcMap或ArcCatalog中使用VBA，大多数常用的ESRI对象库已经被引用了。在独立的Visual Basic应用程序或组件中，必须手工引用所需的库。

为找出一个 ArcObjects 在哪个库中，查看开发帮助中的对象模型图，或者使用开发工具包工具目录的 LibraryLocator 工具。

下面以识别一个简单的对象及其支持的接口作为开始。在这个例子中，会使用一个*Point*对象和*IPoint*接口。为这个点设置坐标的一种方法就是在*IPoint*接口上调用*PutCoords*方法并传入坐标值。

```
Dim pPt As IPoint
Set pPt = New Point
pPt.PutCoords 100, 100
```

IID 是 Interface Identifier（一个 GUID）的缩写。

这个简单代码块的第一行举例说明了持有该对象支持接口的引用的用法。这一行代码为*IPoint*接口从ESRI对象库中读取*IID*。在接口名之前加上库名会减少模糊性（符合编码指导方针），尤其是如果在相同的工程中还引用了其它的对象库，例如：

```
Dim pPt As esriCore.IPoint
```

这样，如果碰巧在工程中还有另一个被引用的 *IPoint*，关于正在引用哪一个库对象就不会出现任何歧义。

Coclass 是组件对象类的缩写

代码块中的第二行创建了对象或组件对象类的一个实例，然后为赋给*pPt*的*Ipoint*接口执行一个*QI*操作。

对于象*Point*这样普通的组件对象类的名字，可能希望在组件对象类的名字前加上库名，例如：

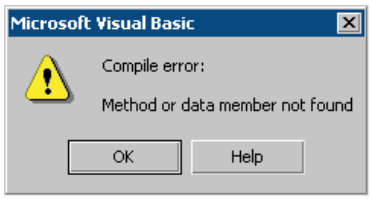
```
Set pPt = New esriCore.Point
```

由于对象的缺省接口是 IUnknown，所以 QI 是必须的。因为 pPt 变量被声明为 IPoin 类型，缺省的 IUnknown 接口是 IPoint 接口的 QI’d。

代码块的最后一行调用*PutCoords*方法。如果一个方法不能在接口上定位，编译时会出现错误。

操作HRESULTs

目前为止已经看见了通过从方法返回的HRESULT来标识操作成功或失败的所有COM方法；没有异常在接口之外引发。也已经知道当遇到错误时，Visual Basic会引发异常。在Visual Basic中，HRESULT*s*从来不从方法调用上返回，而当错误已经发生时Visual Basic会抛出一个异常，这让人更加糊涂。怎么会这样呢？答案就在于Visual Basic 虚拟机。接收HRESULT的是VBVM；如果接收到的不是S_OK，VBVM就抛出异常。如果能从COM错误对象上检索到任何有价值的错误信息，VBVM就用该信息填充Visual Basic Err对象。这样，VBVM控制所有从客户返回的HRESULTs。



当在某个接口上找不到方法或属性时，会弹出这个编译错误消息窗口。

当在Visual Basic中实现接口时，抛出一个HRESULT错误以通知调用者发生了错误是一个很好的编程习惯。通常，当方法还没有实现时会发生这种异常。

```
' Defined in module
Const E_NOTIMPL = &H80004001 'Constant that represents
HRESULT
'Added to any method not implemented
On Error GoTo 0
Err.Raise E_NOTIMPL
```

必须编写代码处理返回的HRESULT不是S_OK的可能情况。发生这种情况时，应调用错误处理程序并处理错误。这可能只是简单地通知用户，或者也可能是自动处理错误并继续执行函数。如何选择取决于环境。下面是一个简单的错误处理程序，可以捕获函数内任何的错误并报告给用户。注意*Err*对象的使用将有关错误的一些描述提供给了用户。

```
Private Sub Test()
  On Error GoTo ErrorHandler
  ' Do something here.
  Exit Sub ' Must exit sub here before error handler
  ErrorHandler:
  Msgbox "Error In Application  -  Description " & Err.Description
End Sub
```

操作属性

有的属性引用ESRI对象库中的特定接口，而其它属性则具有标准数据类型的值，例如字符串、数值表达式和布尔值等。对于接口引用，声明一个接口变量并用*Set*语句将接口引用赋给属性。对于其它值，声明一个数据类型明确的变量或使用Visual Basic的*Variant*数据类型。然后使用简单的赋值语句给变量赋值。

本身是接口的属性可以通过引用或者通过值来设置。通过值来设置的属性不需要Set语句。

```
Dim pEnv As IEnvelope
Set pEnv = pActiveView.Extent 'Get extent property of view.
pEnv.Expand 0.5, 0.5, True 'Shrink envelope.
pActiveView.Extent = pEnv 'Set By Value extent back on IActiveView.
Dim pFeatureLayer as IfeatureLayer
Set pFeatureLayer = New FeatureLayer 'Create New Layer.
Set pFeatureLayer.FeatureClass = pClass 'Set ByRef a class into layer.
```

正如期望的那样，有些属性是只读的，有些属性是只写的，而且还有一些属性是可读写的。所有对象浏览器和ArcObjects Class Help（在ArcGIS 开发帮助系统中可以找到）都提供这方面的信息。如果试图使用一个属性，并且忘记或错用了*Set*关键字，Visual Basic对源代码的编译将失败，返回一个“方法或数据成员没有找到”的错误消息。这个错误看起来很奇怪，因为可能解释为试图给一个只读的属性赋值。产生这个消息的原因是Visual Basic试图在类型库中找到一个映射属性名的方法。在上面的例子中，在类型库中的底层方法调用是*put_Extent*和*putref_FeatureClass*。

操作方法

方法执行某种动作，并且可能返回值或不返回值。在有些情况下，方法返回的值是一个接口；例如，在下面这段代码中，*EditSelection*返回一个枚举型要素接口：

```
Dim pApp As IApplication
Dim pEditor As IEditor
Dim pEnumFeat As IEnumFeature 'Holds the selection
Dim pID As New UID
'Get a handle to the Editor extension
pID = "esriEditor.Editor"
Set pApp = Application
Set pEditor = pApp.FindExtensionByCLSID(pID)
'Get the selection
Set pEnumFeat = pEditor.EditSelection
```

在其他情况下，方法返回一个反映操作成功与否的布尔值，或将数据写入一个参数中；例如，如果发生一个选择，*GxDialog*的*DoModalOpen*

方法返回一个 True 值并将该选择写到一个 *IEnumGxObject* 参数中。注意不要混淆 Visual Basic 从一个方法调用返回值这个概念和所有 COM 方法必须返回一个 HRESULT 这个概念。VBVM 可以读取类型库信息并将 VB 方法调用的返回值设置为 COM 方法的合适参数。

操作事件

事件让用户知道什么时候会发生某件事。可以添加代码以响应一个事件。例如，命令按钮有一个*Click*事件。可以添加代码以便当用户单击这个控件时执行某个动作。也可以添加某些对象产生的事件。VBA和Visual Basic允许用关键字*WithEvents*声明一个变量。*WithEvents*告诉开发环境该对象变量会被用来响应对象事件。这有时候也称作一个“事件汇”。*WithEvents*必须在类模块或窗体中声明。下面是在*Declarations*部分声明变量和暴露对象事件的方法。

```
Private WithEvents m_pViewEvents as Map
```

Visual Basic中每个组建对象类只支持一个出接口（在IDL中标记为缺省出接口）。为克服这个缺陷，实现多个出接口的组件对象类有一个相关联的伪（dummy）组件对象类，允许访问第二个出接口。这些伪组件对象类的名字与它们包含的出接口相同，只是没有前缀I。

```
Private WithEvents m_pMapEvents as MapEvents
```

一旦声明了变量，就可以在Code窗口左上方的Object组合框中查找其名字。然后在Code窗口右上角的Procedure/Events组合框中检查可以添加代码的事件列表。

不是所有事件出接口的方法都需要存根，如同Visual Basic会为任何没有实现的方法存根一样。这一点与入接口不同，它的所有方法就必须存根以进行编译。

在调用方法之前，必须建立事件源和汇之间的连接。可以通过设置表示事件汇到事件源的变量来完成这个任务。

```
Set m_pMapEvents = pMxDoc.FocusMap
```

指向作为参数的有效对象的指针

有些ArcGIS方法需要接口作为它们的参数。在方法调用前或方法调用完成之后，被传递的接口指针可以指向一个实例化的对象。

例如，如果想查找一个多边形(*pPolygon*)的中心点，可以象下面这样编写代码：

```
Dim pArea As IArea
Dim pPt As IPoint
Set pArea = pPolygon ' QI for IArea on pPolygon
Set pPt = pArea.Center
```

开发人员不需要创建 *pPt*，因为 *Center* 方法会创建了一个 *Point* 对象并通过其 *IPoint* 接口传回一个对该对象的引用。只有使用客户端存储的方法才需要在方法调用之前创建对象。

在模块间传递数据

当在模块间传递数据时，最好使用处理特定私有成员变量的存取函数和赋值函数（accessor和mutator functions）。这提供了数据封装，既面向对象编程的一种基本技术。永远不要使用公有变量。

例如，可能已经将一个变量的有效范围设为1-100。如果允许其他开发人员直接访问这个变量，他们可能会为该变量设置非法值。处理这些非法值的唯一的方法就是在使用前检验它们。这在编程上既容易出错，也十分烦琐。声明所有变量为类模块的私有成员变量并为处理这些变量提供存取函数和赋值函数可以解决这个问题。

在下面的例子中，这些属性被添加到类的缺省接口。注意用来给客户抛出错误的技术。

```
Private m_lPercentage As Long
Public Property Get Percentage() As Long
   Percentage = m_lPercentage
End Property
Public Property Let Percentage(ByVal lNewValue As Long)
   If (lNewValue >= 0) And (lNewValue <= 100) Then
      m_lPercentage = lNewValue
   Else
      Err.Raise vbObjectError + 29566, "MyProj.MyObject",
      "Invalid Percentage Value. Valid values (0 -> 100)"
   End If
End Property
```

当编写代码从一个窗体、类或模块将对象引用传递到另一个窗体、类或模块时，例如：

```
Private Property Set PointCoord(ByRef pPt As IPoint)
   Set m_pPoint = pPt
End Property
```

这些代码传递了指向IPoint接口实例的一个指针。这意味着只是传递对接口的引用，而不是接口本身；如果添加*ByVal*关键字（像下面这样），接口通过值来传递。

```
Private Property Let PointCoord(ByVal pPt As IPoint)
   Set m_pPoint = pPt
End Property
```

在这两个例子中，接口指向的对象总是通过引用来传递。为了通过值传递对象，必须创建对象的一个副本并传递这个副本。

使用TypeOf关键字

可以用Visual Basic的*TypeOf*关键字来检查一个对象是否支持某个接口。例如，给定ArcMap内容列表中所选的一个数据项，可以用下面的代码检测该数据项是否为一个*FeatureLayer*：

```
Dim pDoc As IMxDocument
Dim pUnk As IUnknown
Dim pFeatLyr As IGeoFeatureLayer
Set pDoc = ThisDocument
Set pUnk = pDoc.SelectedItem
If TypeOf pUnk Is IGeoFeatureLayer Then ' can we QI for
   IGeoFeatureLayer ?
   Set pFeatLyr = pUnk ' actually QI happens here
   ' Do something with pFeatLyr.
End If
```

使用Is操作符

如果代码要求比较两个接口引用变量，可以使用*Is*操作符来完成比较。一般地，可以在以下情况中使用*Is*操作符：

- 检测是否有一个有效接口。

```
Dim pPt As IPoint
Set pPt = New Point
If (Not pPt Is Nothing) Then 'a valid pointer?
   ... ' do something with pPt

End If
```

- 检查两个变量是否引用同一个实际对象。比如说有两个*IPoint*类型的接口变量*pPt1*和*pPt2*。它们是否指向相同的对象？如果是，则*pPt1* Is *pPt2*。

*Is*关键字使用一个对象的COM身份。下面一个例子说明了*Is*关键字的用法。该例子检验接口的某个方法返回的是对象的副本还是真实对象的引用。

在下面的例子中，地图（*IMap*）的*Extent*属性返回一个副本，而文档（*IMxDocument*）的*ActiveView*属性总返回一个对真实对象的引用。

```
Dim pDoc As IMxDocument
Dim pEnv1 As IEnvelope, pEnv2 as IEnvelope
Dim pActView1 As IActiveView
Dim pActView2 as IActiveView
Set pDoc = ThisDocument
Set pEnv1 = pDoc.ActiveView.Extent
Set pEnv2 = pDoc.ActiveView.Extent
Set pActView1 = pDoc.ActiveView
Set pActView2 = pDoc.ActiveView
' Extent returns a copy,
' so pEnv1 Is pEnv2 returns False
Debug.Print pEnv1 Is pEnv2
' ActiveView returns a reference,
' so pActView1 Is pActView2
Debug.Print pActView1 Is pActView2
```

在集合中迭代

在操作ArcMap和ArcCatalog时，会发现许多情况下要操作集合。可以用一个计数器在这些集合中迭代。计数器是一个提供遍历元素列表方法的接口。计数器接口一般以*IEnum*开始并有两个方法：*Next*和*Reset*。*Next*返回集合中的下一个元素并向前移动内部指针，*Reset*复位内部指针到集合的开始。

计数器可以支持其它方法，但所有计数器中都有这两个方法。

这里是一些VBA代码，在一个地图中选中的要素（*IEnumFeature*）上执行循环操作。要试验这些代码，加载States示例图层到地图中并用Select工具选择多个要素（用鼠标拖出一个矩形框进行选择）。将代码添加到一个VBA宏中，然后执行该宏。每个选中的州的名字会打印在调试窗口中。

```
Dim pDoc As IMxDocument
Dim pEnumFeat As IEnumFeature
Dim pFeat As IFeature
Set pDoc = ThisDocument
Set pEnumFeat = pDoc.FocusMap.FeatureSelection
Set pFeat = pEnumFeat.Next
Do While (Not pFeat Is Nothing)
   Debug.Print pFeat.Value(pFeat.Fields.FindField("state_name"))
   Set pFeat = pEnumFeat.Next
Loop
```

有些集合对象，Visual Basic Collection就是一个，实现了一个名为*_NewEnum*的特殊接口。因为有“_”前缀，这个接口会被隐藏，但是Visual Basic开发人员仍然可以用它来简化集合中的迭代。Visual Basic *For Each*结构使用这个接口，以在一个集合中执行*Reset*和*Next*步骤。

```
Dim pColn as Collection
Set pColn = GetCollection()' Collection returned from some function

Dim thing as Variant ' VB uses methods on _NewEnum to step through
For Each thing in pColn ' an enumerator.
   MsgBox Cstr(thing)
Next
```

本章前一节基本上集中讲述如何在ArcGIS Desktop应用程序内嵌的VBA开发环境下编写代码。这一节重点讲述有关创建可以添加到应用程序中的ActiveX DLLs和使用Visual Basic开发环境编写外部独立应用程序方面的内容。

创建COM组件

大多数开发者使用Visual Basic来创建操作ArcMap或ArcCatalog的COM组件。在本章前面已经知道因为ESRI应用程序是COM客户端—它们的体系结构支持使用遵从COM规程的软件组件—因而可以用包括Visual Basic在内的不同语言来构建组件。然后这些组件就可以轻松地添加到应用程序中。要了解关于打包和部署用Visual Basic创建的COM组件方面的更多信息，请参见本章最后一节。

本节并不想作为一个Visual Basic教程;而是重点介绍使用ArcObjects时应该了解的Visual Basic知识。

在Visual Basic中，可以通过创建一个ActiveX DLL来构建一个可以操作ArcMap或ArcCatalog 的COM组件。这里回顾一些涉及到的基本步骤。注意这些步骤不是什么都包括的。具体的工程可能涉及其它需求。

1. 启动Visual Basic。在New Project对话框中创建一个ActiveX DLL Project。
2. 在Properties窗口中，确认最初的类模块和添加到Project中的任何其它类模块的Instancing属性设置为5—MultiUse。
3. 引用所需的ESRI Object Libraries。
4. 实现需要的接口。在一个类模块中实现一个接口时，该类提供了该接口的类型库中指定的所有公共过程在该类自己上的版本。除了提供接口原型和过程之间的映射外，*Implements*语句使得类接受调用专门接口ID的COM *QueryInterface*。必须包含所有涉及的公共过程。接口或类的实现中如果缺少一个成员就会导致错误。如果没有在所实现的类中的过程中输入代码，会引发适当的错误（*Const E_NOTIMPL* = &H80004001）。那样，如果其他人使用这个类，就会明白一个成员没有被实现。
5. 添加所需的任何其他代码。
6. 确定Project Name和其它属性来识别组件。在Project Properties对话框中，指定的工程名会被作为组件类型库名。可以将工程名和类名结合起来提供唯一的类名（也叫做ProgIDs）。这些名称出现在Component Category Manager中。保存工程。

可以使用ESRI VB Add-In接口实现工具自动执行步骤3和4。

Visual Basic为类、接口和库自动产生必要的GUIDs。设置二进制兼容性强制VB从一个先前编译的DLL复用GUIDs。这样做是必要的，因为ArcMap在文档中储存了命令的GUIDs以备下次加载。

7. 编译DLL。
8. 设置组件的Version Compatibility为二进制。因为代码需要不断进化，所以将组件设置为Binary Compatibility是一个很好的习惯，这样，如果修改了一个组件，会被警告破坏了兼容性。要了解更多其他信息，请参见Visual Basic在线帮助中的“二进制兼容模式”帮助主题。
9. 保存工程。
10. 使组件对应用程序可用。通过在Customize对话框中的Commands选项卡上单击Add from file按钮添加组件到一个文档或模板中。另外，还可以在Component Category Manager中注册一个组件。

实现接口

Visual Basic中接口实现的方式是不同的，取决于它们是入接口还是出接口。Visual Basic将出接口被看作事件源，并通过*WithEvents*关键字来支持。要在Visual Basic中操作出接口*IactiveViewEvents*（*Map*类缺省出接口），使用*WithEvent*关键字并提供适当的函数来处理事件。

```
Private WithEvents ViewEvents As Map

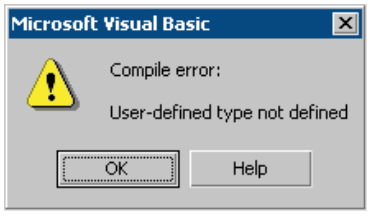
Private Sub ViewEvents_SelectionChanged()
  ' User changed feature selection update my feature list form
  UpdateMyFeatureForm
End Sub
```

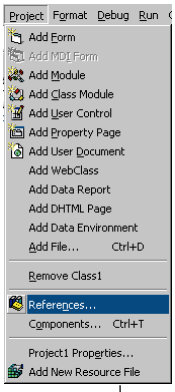
入接口用Implements关键字来支持。然而，与出接口不同，定义在入接口上的所有方法必须存根。这保证对象实例化时形成正确的vTable。不是所有的方法都必须完全编码的，但是函数必须存根。如果实现是空白的，应有适当的返回代码通知用户方法未被实现（参见“操作HRESULTs”一节）。为实现*IExtension*接口，需要类似于下面的代码。注意到所有的方法都被实现。

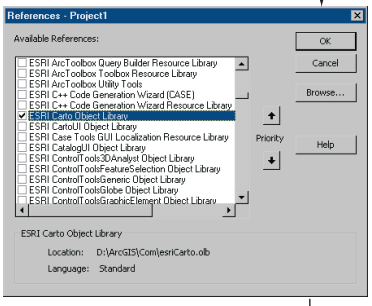
```
Private m_pApp As IApplication
Implements IExtension
Private Property Get IExtension_Name() As String
  IExtension_Name = "Sample Extension"
End Property

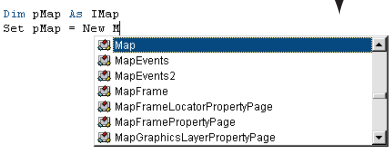
Private Sub IExtension_Startup(ByRef initializationData As Variant)
  Set m_pApp = initializationData
End Sub

Private Sub IExtension_Shutdown()
  Set m_pApp = Nothing
End Sub
```









ESRI Object Library被引用后，包含在其内的所有类型都可通过Visual Basic获得。IntelliSense也会操作对象库的内容。

设置ESRI对象库引用

使用内嵌于应用程序内的VBA开发环境和Visual Basic的主要差异就在于后者要求加载所有合适的对象库，这样所声明的任何对象变量都可以被找到。如果不添加引用，会出现左侧的错误消息。另外全局变量*ThisDocument*和 *Application*对后者用户不可用。

添加对一个对象库的引用

取决于想要代码做的事情，可能会需要添加几个ESRI的核心对象和扩展库。可通过在开发者帮助中查看对象模型图，或使用位于开发工具包工具目录中的LibraryLocator工具确定一个对象属于哪个库。

选择Visual Basic Project菜单中的References来显示References对话框，以便在其中设置所需的引用。

在通过复选名字右侧的复选框设置了一个对象库的引用后，可以在对象浏览器中找到特定的对象及其方法和属性。

如果没有使用任何被引用库中的对象，应该清除该引用的复选框使Visual Basic必须处理的对象引用最少，这样会减少工程编译花费的时间。一定不要移除工程中正被使用的引用。

不能移除“Visual Basic for Applications”和“Visual Basic objects and procedures”引用，因为它们是运行Visual Basic所必须的。

引用文档

每个VBA工程(Normal, Project, TemplateProject)都有一个叫做*ThisDocument*的类，它表示一个文档对象。在VBA中任何地方编写代码都可以用*ThisDocument*来引用文档。而且，如果在*ThisDocument* Code窗口中编写代码，可以直接访问*IDocument*中的所有方法和属性。在Visual Basic中这是不行的。必须先获得一个对*Application*的引用，然后得到一个对文档的引用。添加扩展模块和命令到ArcGIS应用程序中时，提供了一个指向*Application*接口的指针。

```
Implements IExtension
Private m_pApp As IApplication

Private Sub IExtension_Startup(ByRef initializationData As      Variant)
   Set m_pApp = initializationData ' Assign IApplication.
End Sub
Implements ICommand

Private m_pApp As IApplication
Private Sub ICommand_OnCreate(ByVal hook As Object)
   Set m_pApp = hook ' QI for IApplication
End Sub
```

由于对应用程序的引用在一个*IApplication*指针成员变量中，因此文档和所有其它对象可以从类中的任何方法上访问。

```
Dim pDoc as IDocument
Set pDoc = m_pApp.Document
MsgBox pDoc.Name
```

获取对象

上一个例子中，在ArcMap中浏览一个对象是一个直截了当的过程，因为指向Application对象（它是大多数ArcGIS应用程序对象的根对象）的指针通过它的一个接口被传递到对象上。然而，ArcObjects应用程序框架内执行的所有接口不象上面这样。当实现存在于框架中的一个对象时，从这个对象上遍历对象的层次结构是不可能的。这是因为只有少数几个对象支持对其父对象的引用（*IDocument*接口有一个名为*Parent*的引用*IApplication*接口的属性）。为了让开发人员访问应用程序对象，有一个singleton对象提供一个指向正在运行的应用程序对象的指针。下面的代码说明了它的用法。

Singletons是只支持一个例程的对象。这些对象有一个类工厂来确保一个对象被请求时，返回一个已有对象的指针。

```
Dim pAppRef As New AppRef
Dim pApp as IApplication
Set pApp = pAppRef
```

要注意确保这个对象只在ArcMap和ArcCatalog中运行的实现中才使用。例如，在一个自定义要素中使用此功能不是一个好主意，因为这样会限制查看要素类所用的应用程序。

用命令行参数运行ARCMAP

可以从命令行启动ArcMap并给它传递一个参数，这个参数或者是文档(.mxd)的路径名或者是模板(.mxt)的路径名。在前一个这个例子中，ArcMap会打开一个文档；在后一个的例子中，ArcMap会基于指定的模板创建一个新的文档。

也可以传递一个参数并通过给Win32 API的*ShellExecute*函数或Visual Basic的*Shell*函数提供一个变量来创建一个ArcMap实例，如下所示：

在Visual Basic中，不可能确定用来启动程序的命令行。磁盘上有一个例子提供了这种功能。在<ArcGIS Developer Kit install>\samples\COM Techniques\ Command Line中可以找到。

```
Dim ret As Variant
ret = Shell("d:\arcgis\bin\arcmap.exe _
d:\arcgis\bin\templates\LetterPortrait.mxt", vbNormalFocus)
```

缺省情况下，*Shell*异步运行其它的程序。这意味着在*Shell*函数后面的语句执行之前ArcMap可能不会结束执行。

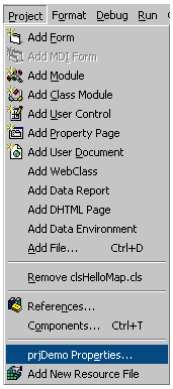
必须调用三个Win32API函数来执行一个程序并等待直到它终止。首先调用*CreateProcessA*函数来加载和执行ArcMap。其次调用*WaitForSingleObject*函数，强制操作系统等待直到ArcMap终止。最后，当用户终止应用程序时，调用*CloseHandle*函数来释放系统池中应用程序的32位标识符。

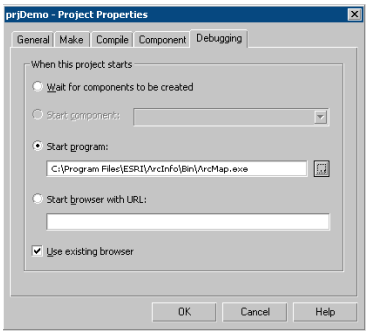
调试VISUAL BASIC代码

Visual Basic有一个集成在开发环境中的调试器。调试Visual Basic代码时，它在很多情况下都是一个有效的工具；然而在有的情况下不可能使用VB调试器。调试器的使用及这些特殊的情况在下面的内容中会详细讨论。

在应用程序中运行代码

可能会使用Visual Basic调试器调试基于ArcObjects的源代码，甚至当ActiveX DLLs为目标服务器时。DLL的宿主应用程序必须设置为Debug应用程序。要完成这个任务，选择合适的应用程序，例如ArcMap.exe，并在Project Properties的Debugging Options中设置它为Start Program。







使用Debug工具条上的命令可以启动ArcMap并加载和调试DLL。可以设置断点以跳过的行，跳入的函数和选中的变量。移动左边的行指示器也可以设置当前执行行。

Visual Basic调试问题

在许多情况下，Visual Basic调试器会正常工作；然而，在使用Visual Basic 6提供的调试器时会出现两个问题。这两个问题存在的原因都是因为Visual Basic实现其调试器的方式问题。

通常在ArcMap中运行一个工具时，DLL被加载到ArcMap地址空间中，并且调用直接进入到DLL中。调试时就不是这样。Visual Basic会修改注册表，以使DLL的类识别符（CLSID）不是指向用户的DLL，而是指向Visual Basic Debug DLL（VB6debug.dll）。Debug DLL必须支持所有被用户类动态实现的接口。当VB Debug DLL加载到ArcMap中时，DLL上的任何方法调用被转寄给Visual Basic，也就是调试的代码要执行的位置。这两个问题就是由于对注册表的修改和跨进程空间方法调用而导致的。第一次遇到这些限制时，可能会感到糊涂，因为对象在调试器外操作或者至少直到遇到错误代码区域时是这样。

因为从ArcMap到自定义工具的方法调用是跨单元的，所以需要接口被调度。这种调度在某些情况下会导致错误。大多数数据类型可以被系统自动调度，但是有一些需要自定义代码因为标准调度法不支持这种数据类型。如果这些类型的数据在自定义工具中被接口使用且没有自定义调度代码，调试器会以“Interface not supported error”失败。

注册表操作也会破坏对组件类目的支持。每当有一个对组件类目的请求时，COM中的类目管理器就不能找到用户的组件，因为不是COM询问用户的DLL是否属于组件类目，而是询问VB调试器DLL是否属于组件类目，而结果显然是不属于。这意味着每当组件类目用来自动加载一个

DLL时，这个DLL不能用Visual Basic调试器进行调试。

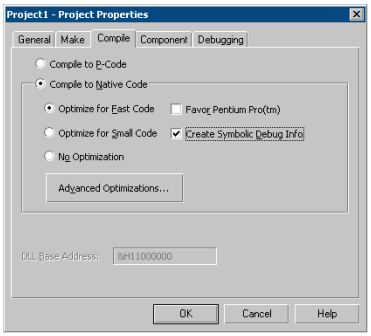
这显然对于许多扩展框架的方法都会导致一些问题。扩展框架最常用的方法是添加一个命令或工具。前面讨论过这种情况下如何使用组件类目。记住组件类目只是用来建立对话框中的命令列表。这意味着如果被调试的命令已经出现在一个工具条上，就可以使用Visual Basic调试器。因此，调试实现了*ICommand*接口的Visual Basic对象的过程是为了确保当ArcMap单独执行时命令被添加到工具条中，并且在保存文档之后，通过调试器加载ArcMap。

有些情况下，如扩展模块和属性页面等，是不可能用Visual Basic调试器的。如果已经有了Visual C++调试器，可以选择下面概括的选项之一。幸运的是，有许多ESRI Visual Basic Add-ins使得快速有效地跟踪错误成为可能。这些插件在ArcGIS 开发帮助中的‘Visual Basic Developer Add-Ins’部分描述过，它们提供包括行和模块细节的错误日志信息。下面给出了一个错误日志的输出例子；注意与行号在一起的调用栈信息。

```
Error Log saved on : 8/28/2000 - 10:39:04 AM
Record Call Stack Sequence - Bottom line is error line.

   chkVisible_MouseUp   C:\Source\MapControl\Commands\frmLayer.frm
Line : 196

   RefreshMap C:\Source\MapControl\Commands\frmLayer.frm Line : 20
Description
Object variable or With block variable not set
```



在Project Properties对话框的Compile选项卡中使用Create Symbolic Debug info选项来创建调试符号信息。

Visual Basic调试器的替代选择

如果Visual Basic调试器和插件不能提供足够的信息，可以使用Visual C++调试器，或者单独使用，或者与C++ ATL包装类一块使用。Visual C++调试器不运行在ArcMap进程以外被调试的对象，这意味着上面的问题都不适合。在“Developer Studio中的调试技巧”一节给出了常用的调试命令。下面的两个技术都需要Visual Basic工程被编译为带有调式符号信息。

Visual C++调试器可以操作这些符号调试信息和源文件。

Visual C++调试器

可以通过加入一个运行中的、拥有被加载的Visual Basic待调试对象的进程并在Visual Basic文件中设置断点而直接使用Visual C++调试器。到达断点行代码后，调试器将停止执行并跳入源文件的正确行。所需的步骤如下所示。

1. 启动一个合适的应用程序，如ArcMap.exe。
2. 启动Microsoft Visual C++。
3. 用菜单选项Build > Start Debug > Attach to process来加入ArcMap进程。
4. 加载合适的Visual Basic Source文件到Visual C++调试器中并设置断点。
5. 在ArcMap中调用方法

在调试器中不能对源代码作任何修改，而且变量也不能被查看，但是可以查看和改变代码执行。这样常常足以确定什么错了，尤其是与逻辑相关的问题。

ATL包装类

使用ATL可以创建一个实现与Visual Basic类接口相同的类。在创建ATL对象时，也就创建了Visual Basic对象。所有方法调用都被传递给Visual Basic对象来执行。通过在合适的C++包装方法中设置断点来调试所包含的对象，并且当代码到达断点时，调试器跳入Visual Basic代码中。要了解有关这个技术的更多信息，请查看ArcGIS 开发帮助系统中开发示例的ATL Debugger示例。

在Visual C++中开发是一个庞大而复杂的题目，因为与其它开发环境比较，Visual C++提供了与底层的Windows APIs 和COM APIs更低级别的交互。

尽管这对于快速程序开发来说可能是一个阻碍，但它是最灵活的方法。大量的设计模式，例如COM聚合和单元素集合，在Visual C++中是可能的，但在Visual Basic 6是不可能的。通过使用标准类库，如Active Template Library，复杂的COM管件代码可能会被隐藏。然而，彻底地了解ATL COM运行的底层仍然十分重要。

这部分文档是基于Microsoft Visual C++ 6编制的，为这种环境下的ArcGIS开发提供了一些指导。随着Visual Studio C++ .NET（也称作VC7）的发布，C++的开发人员有了一些新的东西。VC7可以操作.NET环境，也可以操作ArcGIS .NET API，这只会增加访问底层ArcGIS COM对象的额外操作。因此在VC7中的ArcGIS开发，建议以传统方式工作—也就是说，直接使用ArcGIS COM接口和对象。

VC7中ATL增加了许多功能。本章后面的”ATL与Visual C++.NET”一节讨论了其中一些相关的变化。

随着Visual C# .NET语言的加入，移植Visual C++代码到这种环境中并使用ArcGIS .NET API是值得的。C#与C++的语法不一样，但结果代码通常更加简单和协调。

这部分内容主要服务于两个目的。

1. 熟悉的Visual C++编码风格并调试，开始关于ATL的讨论。
2. 详细介绍在Visual C++中操作ArcObjects编程平台的需求和建议。

操作ATL

这部分不能覆盖使用ATL的开发人员应该有效了解的所有问题，但可作为ATL的一个介绍。ATL帮助实现COM对象并节省了输入，但这不能因此就不需要知道C++和如何开发COM对象。

ATL是实现COM对象的推荐框架。ATL可以与Microsoft Foundation Class Library (MFC)代码结合起来，这样为编写应用程序提供了更多的支持。MFC的可供选择的替代项是Windows Template Library (WTL)，WTL建立在ATL模板方法的基础上，并为windows类和其它支持ATL的应用程序提供了许多包装。WTL可以从Microsoft上下载；编写本书时7.1版本是最新的并且可以与Visual C++ 6和Visual C++ .NET一块使用。

ATL简介

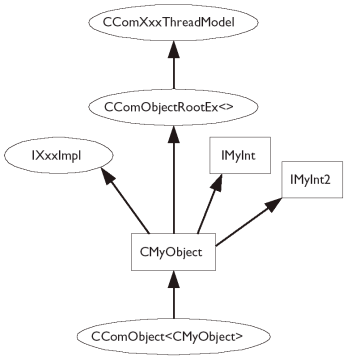
ATL是C++模板类的集合，被设计为小巧、快速并且可扩展，松散地建立在Standard Template Library (STL)的基础上。STL为C++对象提供了普通的模板类，例如向量、堆栈和队列。ATL也提供了扩展Visual

Studio开发环境的向导的集合。这些向导自动生成ATL工程必须的单调的管件代码。这些向导包括但不限于下面列出的项。

- Application—用于初始化ATL C++工程。
- Object—用于创建COM对象。生成C++和IDL代码，和一些运行时支持创建对象的合适的代码。
- Property—用于添加接口的属性。
- Method—用于添加方法到接口上；Property和Method向导都要求知道IDL语法。
- Interface Implementation—用于实现现有接口的存根函数。
- Connection Point Implement—用于实现出站事件接口。

典型情况下通过在Visual Studio Workspace/Class View中右键单击一个工程、类或接口访问这些向导。

ATL为实现COM对象和实现一些通用COM接口，包括*IUnknown*、*IDispatch*和*IClassFactory*提供了基类。也提供一些支持ActiveX控件和它们的容器的类。



ATL层次结构

ATL提供了暴露ATL-based COM对象所需的一些服务，包括注册、服务器生命周期和类对象。

这些模板类创建了包含我们的类的层次结构。这些类继承关系在左侧图中说明。*CComxxxThreadModel*类支持对全局变量、实例和静态数据的线程安全访问。*CComObjectRootEx*为*IUnknown*的方法提供了行为。处于第二层的接口表示类要实现的接口，这种接口有两种：*IXxxImpl*接口包含了一些ATL支持包含实现的接口；另一接口有一些必须在类内完全实现的纯虚函数。*CComObject*类继承了用户的类，这个类与对象实例化和生命周期控件一起提供了*IUnknown*方法的实现。

关于Direct-To-COM更为详细的讨论，参见后面Direct-To-COM聪明类型一节。

ATL和DTC

Direct-To-COM (DTC)与智能类型（在本章后面讨论）一块提供了一些有用的编译器扩展信息，创建ATL-based对象时会用到这些。函数*__declspec*和*__uuidof*就是两个这样的函数，但最有用的是*#import*命令。

COM接口在IDL中定义，然后被Microsoft IDL编译器(MIDL.exe)编译。这样创建了一个类型库和头文件。当编译引用这些接口的软件时，工程自动使用这些文件。这个方法的局限性在于，操作接口时，必需访问IDL文件。作为一个ArcGIS的开发人员，我们只能访问包含在.olb和.ocx文件中的ArcGIS类型库信息。尽管可以从一个类型库中设计一

个头文件，但这是一个单调乏味的过程。*#import*命令自动创建编译器所需的文件。因为该命令是支持DTC的，当使用它导入ArcGIS类型库时，需要传递大量的参数，这样正确地完成导入。关于这个过程更多的信息，见后面“导入ArcGIS类型库”。

使用ATL处理错误

可能只是在一个方法内返回一个E_FAIL 的HRESULT代码表明错误；然而这不会给调用者关于错误本质的任何提示。有许多标准的Windows HRESULTs，例如E_INVALIDARG（一个或多个变量无效）和E_POINTER（无效指针）。这些出错代码在Windows头文件*winerror.h*中列出。不是所有的开发环境都全面地支持HRESULT；Visual Basic客户常常将错误结果看作是“Automation Error - Unspecified Error”。 ATL提供了操纵错误信息对象的简单机制，可以提供出错代码和一个出错信息描述。

创建一个ATL对象时， Object向导有一个支持*ISupportErrorInfo*的选项。如果触发了这个选项，向导结束时，用户的对象会执行接口*ISupportErrorInfo*，并且会添加像下面这样一个方法：

```
STDMETHODIMP MyClass::InterfaceSupportsErrorInfo(REFIID riid)
{
  static const IID* arr[] =
  {
    &IID_IMyClass,
  };

  for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)
  {
    if (InlineIsEqualGUID(*arr[i], riid))
    return S_OK;
  }

  return S_FALSE;
}
```

现在就可以通过调用一个ATL出错函数返回丰富的出错信息。这些函数甚至会操作源文件以保证信息串的国际化过程简单。

```
// Return a simple string
AtlReportError(CLSID_MyClass, _T("No connection to Database."),
IID_IMyClass, E_FAIL);
// Get the Error Text from a resource string
AtlReportError(CLSID_MyClass,    IDS_DBERROR,    IID_IMyClass,
E_FAIL,
_Module.m_hInstResource);
```

虽然Visual C++支持异常机制(try ... catch)，建议不要和COM代码混合使用这些机制。如果异常出现在COM接口外，不能保证用户可以捕获这个异常，并极有可能导致崩溃。

为了从错误的方法提取出错字符，使用Windows函数*GetErrorInfo*。这个函数提取当前线程的最后一个IErrorInfo对象，并清除当前出错状态。

链接ATL代码

ATL的一个主要目的是支持创建小巧快速的对象。因此，当编译和链接源文件时，ATL给了开发人员大量的选择项。必须选择如何链接或动态访问C运行时(CRT)库、注册代码、和各种ATL应用程序功能。如果CRT没有被代码调用，会从链接上移除。如果CRT被调用并且链接器开关*_ATL_MIN_CRT*没有从链接行移出，在创建过程中会产生下面错误：

```
LIBCMT.lib(crt0.obj) : error LNK2001: unresolved external symbol _main
ReleaseMinSize/History.dll : fatal error LNK1120: 1 unresolved externals
Error executing link.exe.
```

当编译一个debug版本时，可能不会有什么问题；然而，当编译一个release版本时，可能会出现一些问题，这些取决于所编写的代码。如果遇到这类错误，要么移出CRT调用，要么修改链接器开关。

如果应用程序代码在运行时被动态加载，必须保证适当的DLL (ATL.DLL)被安装并且在用户系统上注册。安装ArcGIS 9运行时（runtime）会安装ATL.dll。下面的表列出了不同的选择和相关的链接器开关。

	Symbols	CRT	Utilities	Registrar
Debug		是	静态	动态
RelMinDepend	_ATL_MIN_CRT _ATL_STATIC_REGISTRY	否	静态	静态
RelMinSize	_ATL_MIN_CRT _ATL_DLL	否	动态	动态

有一些缺省条件下ANSI和Unicode构建的构建配置。通过ANSI编译的组件会在Windows 9.x上运行；然而考虑到只有unicode操作系统(Windows NT、Windows 2000、和Windows XP)支持ArcGIS，因此这些配置是多余的。在Visual Studio中删除配置，单击Build / Configurations ...，然后删除*Win32 Debug*、*Win32 Release MinSize*、和*Win32 Release MinDependency*。

COM组件注册

ATL工程向导为注册生成了标准Windows入口点。这些代码会在DLL的类型库中注册并为DLL内每个COM对象执行一个注册脚本文件(.rgs)。执行其它注册任务的附加的C++代码可以插入到这些函数中。

```
STDAPI DllRegisterServer(void)
{
   // registers object in .rgs, typelib and all interfaces in typelib
   // TRUE instructs the type library to be registered
   return _Module.RegisterServer(TRUE);
}
STDAPI DllUnregisterServer(void)
{
 return _Module.UnregisterServer(TRUE);
}
```

ATL提供了一种文本文件格式.rgs，当注册和取消注册一个DLL时，ATL的注册器组件解析这个文件。.rgs文件是作为一个自定义资源嵌入到DLL中。可编辑这个文件来添加其他的注册项目、包含的ProgID、ClassID和要放置在注册表中的组件类别入口。语法描述了从注册表中添加和删除的键、值、名称和子键。这种格式总结如下：

```
[NoRemove | ForceRemove | val] Name | [ = s 'Value' | d 'Value' | b
'Value' ]
{
   .. optional subkeys for the registry
}
```

*NoRemove*表示注册表键不应该通过取消注册移除。*ForceRemove*确保在注册新键前移除键和子键。*s*、*d* 和 *b* 值分别表示字符型(包括省略号)，双精度型(32位整型值)，和二进制型注册表值。下面是一个典型的注册脚本。

```
HKCR
{
  SimpleObject.SimpleCOMObject.1 = s 'SimpleCOMObject Class'
  {
    CLSID = s '{2AFFC10E-ECFB-4697-8B3D-0405650B7CFB}'
  }
  SimpleObject.SimpleCOMObject = s 'SimpleCOMObject Class'
  {
    CLSID = s '{2AFFC10E-ECFB-4697-8B3D-0405650B7CFB}'
    CurVer = s 'SimpleObject.SimpleCOMObject.1'
  }
  NoRemove CLSID
  {
ForceRemove {2AFFC10E-ECFB-4697-8B3D-0405650B7CFB} = s
'SimpleCOMObject Class'
  {
      ProgID = s 'SimpleObject.SimpleCOMObject.1'
      VersionIndependentProgID = s 'SimpleObject.SimpleCOMObject'
      InprocServer32 = s '%MODULE%'
      {
            val ThreadingModel = s 'Apartment'
      }
      'TypeLib' = s '{855DD226-5938-489D-986E-149600FEDD63}'
      'Implemented Categories'
      {
            {7DD95801-9882-11CF-9FA9-00AA006C42C4}
      }
    }
  }
}
```

*NoRemove CLSID*保证注册键*CLSID*永远不会被移除。所有的COM对象都使用该子键注册它们的ProgIDs和GUIDs，因此，删除这个子键会导致

严重的注册表崩溃。*InprocServer32*是将一个组件的GUID与DLL文件关联的标准COM机制；ATL会使用%MODULE%变量插入正确的模块名称。GUID下的其它项目指定ProgID、线程模型和类型库来使用这个组件。

有两种方式可以将COM组件对象类注册到一个组件类别中。推荐的方法上面已经说明：将组件类别的GUIDs放置在Implemented Categories键下面，该键依次返回组件对象类的GUID。另一个方法是在一个对象的头文件中使用ATL宏：BEGIN_CATEGORY_MAP, IMPLEMENTED_CATEGORY，和END_CATEGORY_MAP。不过，像在MSDN文章*Q279459 BUG: Component CategoryRegistry Entries Not Removed in ATL Component*中解释的一样，这些宏不能正确地移除注册表项目。头文件中提供了ArcGIS使用的所有组件类别的GUIDs；在\Program Files\ArcGIS\include\CatIDs\ArcCATIDs.h中可以找到这个头文件。

如果一个组件的GUID在开发期间改变或者类型库名称改变，那么保持.rgs内容与这些变动的一致非常重要，否则注册将不正确，并且对象创建失败。

调试ATL代码

除了Visual Studio标准工具外，ATL提供了大量专门调试COM对象的调试选项。这些调试选项的输出显示在Visual C++ Output窗口中。可通过设置符号*_ATL_DEBUG_QI*来调试*QueryInterface*调用，*AddRef*和*Release*调用可使用符号*_ATL_DEBUG_INTERFACES*来调试，当*_ATL_DEBUG_INTERFACES*符号被定义时，就可以在终止时通过监视泄漏接口列表跟踪被泄漏的对象。泄漏接口列表有像下面这样的条目：

```
INTERFACE LEAK: RefCount = 1, MaxRefCount = 3, {Allocation = 10}
```

不过这只能告诉我们一个对象被泄漏了，因为接口指针没有被释放。不过当服务器启动时通过设置*CComModule*的*m_nIndexBreaktA*成员获得接口时，*Allocation*数允许自动暂停。接着调用*DebugBreak()*函数强制代码执行停止在调试器相关的位置。不过只有程序流程相同时才可这样。

```
extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason,
LPVOID /*lpReserved*/)
{
  if (dwReason == DLL_PROCESS_ATTACH)
  {
    _Module.Init(ObjectMap, hInstance, &LIBID_HISTORYLib);
    DisableThreadLibraryCalls(hInstance);
    _Module.m_nIndexBreakAt = 10;
    }
  else if (dwReason == DLL_PROCESS_DETACH)
  {
    _Module.Term();
  }
  return TRUE;
}
```

布尔类型

历史上，ANSI C没有Boolean数据类型，而是用整数值代替，0表示假，非零表示真。不过，布尔数据类型现在成为ANSI C++的一部分。COM APIs是独立于开发语言的并定义了一种不同的布尔类型，VARIANT_BOOL。另外，Win32 API使用不同的布尔类型。在合适的时候使用正确的类型是十分重要的。下面的表概括了它们的用法：

类型	真值	假值	哪里定义	何时使用
bool	true (1)	false (0)	编译器定义	编译器固有类型，使用潜力很大。可提升为整型值。表达式(e.g., i!=0)返回一个布尔类型值。特别使用在类成员变量和局域变量中。
BOOL (int)	TRUE (1)	FALSE (0)	Windows 数据类型（在windef.h中定义）	与windows API函数一块使用，经常作为一个指示成功失败的返回值
VARIANT_BOOL (16 bit short)	VARIANT_TRUE (-1)	VARIANT_FALSE (0)	COM Boolean values wtypes.h)	使用在COM APIs的布尔值。也在VARIANT类型中使用；如果VARIANT类型是VT_BOOL，那么VARIANT赋上一个VARIANT_BOOL值。将布尔类成员变量转换为正确的VARIANT_BOOL值时一定要小心。经常使用条件检验"hook－colon"操作符。例如，如果bRes被定义为一个布尔类型，那么设置一个结果类型：*pVal = bRes ? VARIANT_TRUE :VARIANT_FALSE;

字符类型

虽说字符串是简单的概念，但不幸地是它是C++中成为最复杂和容易混淆的主题。两个主要原因是，在相同的代码中支持变长字符串，并要符合支持ANSI和Unicode字符集的要求，这是C++所缺乏的。因为ArcGIS只在Unicode平台下可用，所以可以简化开发过程来移除ANSI的需求。C++对字符串的规定是一个以0结束的字符数组。当计算较大字符串的长度时，通常没有很好的性能。为支持变长字符串，字符数组在堆栈中是动态分配和释放的，典型的是使用*malloc*和 *free*或*new*和*delete*。因此大量的封装类提供这种支持；在MFC和WTL中定义的CString是应用最广泛的。另外，为使用COM，定义了BSTR类型的用法，并且ATL封装类CComBSTR也是可用的。

考虑到国际字符集，Microsoft Windows从8位ANSI字符表示（8位字符）（在Windows 95、Windows 98、和Windows Me平台上可以见到）到移植到16位Unicode字符(16位无符号短整型)中。Unicode与宽字符（wchar_t）是同义的。OLECHAR是在COM APIs中使用的类型，在Windows上被定义为wchar_t。Windows操作系统例如Windows NT, Windows 2000和Windows XP，本来就支持Unicode字符。为了在ANSI和Unicode平台下编译相同的C++代码，采用编译转换器修改Windows API函数为ANSI版本(SetWindowTextA)或Unicode版本(SetWindowTextW)。此外， 引入字符独立类型（在tchar.h中定义的TCHAR）来表示字符：在一个ANSI

构建上被定义为*char*，而在Unicode构建上为*wchar_t*，一个typedef定义为短整型。要执行标准C字符操作，相同的函数通常有三种不同的定义；例如，对于一个区分大小写的比较，*strcmp*提供了ANSI版本，*wcscmp*提供了Unicode版本，*_tcscmp*提供了TCHAR版本。也有第四种版本，*_mbscmp*，是一个八字节版本的变种，在八字节字符内解释多字符序列（MBCS）。

```
// Initialize some fixed length strings
char* pNameANSI = "Bill"; // 5 bytes (4 characters plus a terminator)
wchar_t* pNameUNICODE = L"Bill"; // 10 bytes (4 16-bit characters plus a 16-bit terminator)
TCHAR* pNameTCHAR = _T("Bill"); // either 5 or 10 depending on compiler settings
```

COM APIs使用BSTR来表示变长字符串；这是一个指向一个OLECHAR字符序列的指针，被定义为Unicode字符串，和wchar_t相同。必须使用SysAllocString和SysFreeString windows函数来分配和释放BSTR。与C字符串不一样，它们可包含嵌入零的字符串，虽然这种情况很少见。BSTR也有一个计数值，它在BSTR指针地址前以四个字节储存。CComBSTR包装经常被用来管理字符串的生命周期。

不要传递一个指向C风格的Unicode字符串(OLECHAR或wchar_t)数组的指针给一个期望得到BSTR的函数。如果类型相同，编译器不会引发错误。然而，接收BSTR的函数不能正确运转或当访问该字符串长度时会崩溃,该字符串会是随机的内存值。

```
ipFoo->put_WindowTitle(L"Hello"); // This is bad!
ipFoo->put_WindowTitle(CComBSTR(L"Hello")); // This correctly initializes and passes a BSTR
```

ATL提供了在ANSI (A)、TCHAR(T)、Unicode (W)、和OLECHAR (OLE)之间转换字符串的转换宏。此外,这些类型会有一个常量修饰词(C)。这些宏以缩写形式显示在括号中，宏与宏之间以“2”隔开。例如，要在OLECHAR（例如，一个输入的BSTR）到常量TCHAR （在一个Windows函数中使用）之间转换，使用OLE2CT转换宏。转换ANSI到Unicode，使用A2W。这些宏要求USES_CONVERSION宏被放置在一个方法的最上部；这样会创建一些转换宏所使用的局部变量。当源字符和目标字符集不同时并且目标类型不是BSTR，宏在调用堆栈中分配目标字符（使用_alloca 运行时函数）。意识到这点是非常重要的，尤其是当在一个循环语句中使用这些宏时；否则，堆栈会越来越大并用完堆栈空间。

检验两个字符串是否不同，不要使用不相等(“!=”)运算符。“==”运算符进行两个字符串内容区分大小写的比较；然而，“!=”比较的是指针值，而不是字符串内容，通常返回假。

```
STDMETHODIMP CFoo::put_WindowTitle(BSTR bstrTitle)
{
   USES_CONVERSION;
   if (::SysStringLen(bstrTitle) == 0)
      return E_INVALIDARG;
   ::SetWindowText(m_hWnd, OLE2CT(bstrTitle));
   return S_OK;
}
```

实现非创建类

非创建类是不能被CoCreateInstance创建的COM对象。而是在一个不同对象的方法调用中创建对象，并且返回一个指向非创建类的接口指针。这种对象在地理数据库模型中会大量见到。例如，*FeatureClass*是不可创建的并只能通过调用一个方法成员来获得；一个例子是*IFeatureWorkspace::OpenFeatureClass*方法。

非创建类的一个好处就是使用没有在COM API中暴露的方法调用来以私有数据初始化。下面是一个返回不可创建对象的简化例子：

```
// Foo is a cocreatable object.
IFooPtr ipFoo;
HRESULT hr = ipFoo.CreateInstance(CLSID_Foo);

// Bar is a noncreatable object,cannot use ipBar.CreateInstance(CLSID_Bar).
IBarPtr ipBar;
// Use a method on Foo to create a new Bar object.
hr = ipFoo->CreateBar(&ipBar);
ipBar->DoSomething();
```

将一个可创建的ATL类转换为非创建类所需要的步骤如下：

1. 添加“noncreatable”到.idl文件的coclass属性。

```
[
   uuid(DCB87952-0716-4873-852B-F56AE8F9BC42),
   noncreatable
]
coclass Bar
{
   [default] interface IUnknown;
   interface IBar;
};
```

2. 改变类厂实现舍弃任何的非创建类的共建例程。通过主要的DLL模块中的ATL对象映射图完成。

```
BEGIN_OBJECT_MAP(ObjectMap)
   OBJECT_ENTRY(CLSID_Foo, CFoo) // Creatable object
   OBJECT_ENTRY_NON_CREATEABLE(CLSID_Bar,  CBar)  //
Noncreatable object
END_OBJECT_MAP()
```

3. 作为可选项，可以移除注册表条目。首先，从资源（在这个例子中是Bar.rgs）中移除对象的注册表脚本。然后改变类定义DECLARE_REGISTRY_RESOURCEID(IDR_BAR)为DECLARE_NO_REGISTRY()。

4. 要在一个方法内创建非创建对象，使用CComObject模板补充CreateInstance的实现。

```
// Get NonCreatable object Bar (implementing IBar) from COM object Foo
STDMETHODIMP CFoo::CreateBar(IBar **pVal)
{
```

```
    if (pVal==0)    return E_POINTER;
    // Smart pointer to noncreatable object Bar
    IBarPtr ipBar = 0;

    // C++ Pointer to Bar, with ATL template to supply CreateInstance
implementation
    CComObject<CBar>* pBar = 0;

    HRESULT hr = CComObject<CBar>::CreateInstance(&pBar);
    if (SUCCEEDED(hr))
    {
      // Increment the ref count from 0 to 1 to protect the object
      // from being released in any initialization code.
      pBar->AddRef();

      // Call C++ methods (not exposed to COM) to initialize the Bar
  object.
      pBar->InitialiseBar(10);
      // QI to IBar and hold a smart pointer reference to the object Bar.
      hr = pBar->QueryInterface(IID_IBar, (void**)&ipBar);

      pBar->Release();
    }
    // Return IBar pointer to the caller.
    *pVal = ipBar.Detach();

    return S_OK;
}
```

Visual C++ .NET中的ATL

Visual C++ 版本6可用于此帮助的绝大部分。然而随着Visual C++.NET的发布，有了大量与ArcGIS ATL开发人员有关的增强和改变。下面概括了一些：

基于属性的编程—这是在VC7中引入的最大的变化。属性被插在被方括号围起的源代码中，例如，[coclass]。属性设计用来简化COM编程和.NET框架通用语言运行时开发。当源文件包含属性时，编译器操作提供者DLLs来插入代码或修改生成的对象文件中的代码。有一些属性用来帮助创建.idl文件、接口、类型库和其它的COM元素。在IDE中，通过向导和Properties窗口来支持属性。ATL向导广泛地使用的属性将ATL样板代码插入类中。因此，VC7中典型的COM组件对象类头文件比VC6包含更少的ATL代码。因为IDL是从属性生成的，所以通常在COM工程没有像以前那样的.idl文件表示，并且.idl文件是在编译时生成的。

构造设置—在VC7中只有两个缺省的构造设置；即基于ANSI Debug和Release的构造。因为ArcGIS只在Unicode平台下可以使用，所以建议

通过修改工程属性进行修改。工程的通用属性页有一个“CharacterSet”选项。将该选项的“Use Multi-Byte Character Set”改为Use Unicode Character Set”。

字符转换宏—字符转换宏（USES_CONVERSION, W2A, W2CT, and so forth）有改进的备选版本。不会在堆栈上分配空间，所以在循环语句中应用，而不会用完堆栈空间。不再需要USES_CONVERSION宏。这些宏现在都作为类来实现，并以“C”开头—例如，CW2A和CW2CT。

安全数组支持—通过CComSafeArray和CcomSafeArrayBound类来获得。

模块级全局变量—模块级全局变量CComModule _module被拆分为许多相关的类，例如，CAtlComModule和CAtlWinModule。为检索源模块例程，使用下面的代码：**_AtlBaseModule.GetResourceInstance();**

字符串支持—通用的变长字符支持通过ATL中的CString获得。这一点定义在头文件atlstr.h和cstring.h中。如果ATL与MFC结合，则缺省为以MFC的CString实现。

文件路径处理—通过定义在atlpath.h的CPath类可获得处理文件路径组件的相关函数集。

ATLServer—这是ATL类的一个新选择，为了编写Web应用程序、XML Web服务和其它的服务器应用程序而设计。

 # import问题—使用*#import*时，需要做一些修改。例如，esriSystem的*#import*需要一个*GetObject*的排斥或重命名，并且esriGeometry的*#import*需要一个排斥或*ISegment*的重命名。

ATL参考

Microsoft Developer Network (MSDN)提供了大量与Visual Studio产品一起安装的文档、文章和例子。Visual Studio 6的ATL参考文档在如下目录中：*MSDN Library - October 2001 / Visual Tools and Languages / Visual Studio 6.0Documentation / Visual C++ Documentation / Reference / Active Template Library*

其它文档可以在MSDN Web站点上获得，如下：

http://www.msdn.microsoft.com。

下面的书可能也是有用的：

Grimes, Richard. *ATL COM Programmer's Reference.* Chicago: Wrox Press Inc., 1988.

Grimes, Richard. *Professional ATL COM Programming.* Chicago: Wrox Press Inc.,1988.

Grimes, Richard, Reilly Stockton, Alex Stockton, and Julian Templeman. *Beginning ATL 3 COM Programming.* Chicago: Wrox Press Inc. 1999.

King, Brad and George Shepherd. *Inside ATL.* Redmond, WA: Microsoft Press, 1999.

Rector, Brent, Chris Sells, and Jim Springfield. *ATL Internals.* Reading, MA: Addison-Wesley, 1999.

智能类型

智能类型是和类型有同样行为的对象。它们是C++的类实现，封装了一种数据类型，可以使用容易操作底层类型且更少出错的函数和操作符来包装。当这些智能类型封装了一个接口指针时，被称作*smart pointers*。智能指针操作*IUnknown*接口以确保正确地管理资源分配和存储单元分配。这些可以通过不同的函数、构造函数和析构函数，以及重载运算符来实现。C++程序设计者可获得大量可用的智能类型。这里要讲到的两个智能类型是Direct-To-COM和Active Template Library。

智能类型使得操作COM接口和数据类型更加容易，因为许多API调用被移至一个类的实现中，然而，这些调用必须很谨慎地使用，并且需清楚地理解它们如何与封装的数据类型交互。

Direct-To-COM智能类型

使用DTC支持的智能类型类被称作Compiler COM Support Classes，有下面这些组成部分：

- *_com_error* — 这个类表示一个COM支持类的异常条件。这个对象封装了HRESULT和*IErrorInf* COM异常对象。
- *_com_ptr_t* — 这个类封装了一个COM接口指针。参见下面介绍的一般用法。
- *_bstr_t* — 这个类封装了BSTR数据类型。这个类的函数和操作符不像ATL CComBSTR智能类型那么丰富，因此，它不经常用到。
- *_variant_t* — 这个类封装了VARIANT数据类型。这个类的函数和运算符不像ATL *CcomVariant* 智能类型那么丰富，因此，它不经常用到。

可以使用宏_COM_SMARTPTR_TYPEDEF为接口定义一个聪明指针，像这样：

```
_COM_SMARTPTR_TYPEDEF(IFoo, __uuidof(IFoo));
```

一旦声明，就声明一个变量为接口类型和并追加*Ptr*到接口后。下面是一些智能指针的通常用法，用户会在大量的C++实例中见到。

```
// Get a CLSID GUID constant.
extern "C" const GUID __declspec(selectany) CLSID_Foo = \
  {0x2f3b470c,0xb01f,0x11d3,{0x83,0x8e,0x00,0x00,0x00,0x00,0x00,0x00}};
```

```
// Declare Smart Pointers for IFoo, IBar, and IGak interfaces.
_COM_SMARTPTR_TYPEDEF(IFoo, __uuidof(IFoo));
_COM_SMARTPTR_TYPEDEF(IBar, __uuidof(IBar));
_COM_SMARTPTR_TYPEDEF(IGak, __uuidof(IGak));
STDMETHODIMP SomeClass::Do()
{
   // Create Instance of Foo class and QueryInterface (QI) for IFoo
   interface.
   IFooPtr ipFoo;
   HRESULT hr = ipFoo.CreateInstance(CLSID_Foo);
   if (FAILED(hr)) return hr;
   // Call method on IFoo to get IBar.
   IBarPtr ipBar;
   hr = ipFoo->get_Bar(&ipBar);
   if (FAILED(hr)) return hr;
   // QI IBar interface for IGak interface.
   IGakPtr ipGak(ipBar);
   // Call method on IGak.
   hr = ipGak->DoSomething();
   if (FAILED(hr)) return hr;
   // Explicitly call Release().
   ipGak = 0;
   ipBar = 0;
   // Let destructor call IFoo's Release.
   return S_OK;
}
```

使用DTC智能指针的一个主要优点就在于它们是*#import*编译器语句为类型库中的所有的接口和组件对象类定义自动生成的。关于这个功能更多的细节，参见后面一节“导入ArcGIS类型库”。

可以在一个DTC智能指针的构造函数内含蓄地创建一个对象—例如：

```
IFooPtr ipFoo(CLSID_Foo)
```

然而，如果在对象创建的过程中有错误发生，会引发一个C++异常—例如，如果包含该对象实现的DLL文件被意外地删除。这种异常通常不被处理并导致崩溃。一个更健壮的方法是避免COM中的异常，显式调用CreateInstance，并能够处理错误的代码，例如：

```
IFooPtr ipFoo;
HRESULT hr = ipFoo.CreateInstance(CLSID_Foo);
if (FAILED(hr))
   return hr; // Return object creation failure code to caller.
```

Active Template Library智能类型

ATL定义了不同的智能类型，如下面列表中所示。可以任意地合并ATL和DTC智能类型到代码中。然而，对于智能指针，通常情况下使用DTC，因为它们可简单地通过导入类型库生成。对于BSTR和VARIANT类型，通常使用CComVariant，CComBSTR的ATL版本。

ATL智能类型包括：

- *CComPtr*—通过包装*IUnknown*接口的*AddRef*和*Release*方法封装了一个COM接口指针
- *CComQIPtr*—封装了一个COM接口并支持*IUnknown*接口的所有三个方法：*QueryInterface*、*AddRef*和*Release*
- *CComBSTR*—封装了BSTR数据类型
- *CComVariant*—封装*VARIANT*了数据类型
- *CRegKey*—为操作Windows注册表条目提供了方法
- *CComDispatchDriver*—提供了获取和设置属性值的方法，并通过对象的*IDispatch*接口调用方法。
- *CSecurityDescriptor*—为建立和操作Discretionary Access Control List (DACL)提供方法。

这部分分析了前四种智能类型和它们的用途。下面是用ATL智能指针编写的例子代码：

```
// Get a CLSID GUID constant.
extern "C" const GUID __declspec(selectany) CLSID_Foo = \
   {0x2f3b470c,0xb01f,0x11d3,{0x83,0x8e,0x00,0x00,0x00,0x00,0x00,0x00,0x
00}};

STDMETHODIMP SomeClass::Do ()
{
   // Create Instance of Foo class and QI for IFoo interface.
   CComPtr<IFoo> ipFoo;
   HRESULT hr = CoCreateInstance(CLSID_Foo, NULL,
CLSCTX_INPROC_SERVER, IID_IFoo, (void **)&ipFoo);
   if (FAILED(hr)) return hr;

   // Call method on IFoo to get IBar.
   CComPtr<IBar> ipBar;
   HRESULT hr = ipFoo->get_Bar(&ipBar);
   if (FAILED(hr)) return hr;
   // IBar interface for IGak interface
   CComQIPtr<IGak> ipGak(ipBar);
   // Call method on IGak.
   hr = ipGak->DoSomething();
   if (FAILED(hr)) return hr;
   // Explicitly call Release().
   ipGak = 0;
```

当使用智能指针比较时，等号运算符（“== ”）实现方式会有所不同。COM规定状态对象识别是通过比较IUnknown指针值实现的。当使用“==”运算符时，DTC智能指针会执行必要的QI和比较。然而，ATL智能指针不会做这些，必须使用ATL IsEqualObject()方法。

```
    ipPar = 0;
    // Let destructor call Foo's Release.
    return S_OK;
}
```

在Visual C++例子中最常见的智能指针是DTC类型。在下面的例子中，说明了BSTR和*VARIANT*数据类型，使用了DTC指针。当操作*CComBSTR*时，使用文本映射“L”来声明常量*OLECHAR*字符串。*CComVariant*从VARIANT数据类型上直接派生，这意味着它的实现不会有重载的情况，这反过来也简化了它的使用。它包含一个丰富的构造和析构函数集合，这使得操作*VARIANT*s更加直接；甚至有从数据流上读写的方法。重用变量之前一定要调用*Clear*方法。

```
ipFoo->put_Name(CComBSTR(L"NewName"));
if FAILED(hr)) return hr;

// Create a VT_I4 variant (signed long).
CComVariant vValue(12);
// Change its data type to a string.
hr = vValue.ChangeType(VT_BSTR);
if (FAILED(hr)) return hr;
```

IDL里调用的一些方法被标识为可选的并接受变量型参数。然而在Visual C++中，仍然要提供这些参数。为了表示一个参数值没有被提供，传递一个变量来指定错误代码或类型DISP_E_PARAMNOTFOUND：

```
CComBSTR documentFilename(L"World.mxd");
CComVariant noPassword;
noPassword.vt = VT_ERROR;
noPassword.scode = DISP_E_PARAMNOTFOUND;
HRESULT hr = ipMapControl->LoadMxFile(documentFilename,
noPassword);
```

当操作*CComBSTR*和CComVariant时，*Detach()*函数释放了来自智能类型的下层数据类型，这样当传递一个结果作为一个方法的[out]参数时它就可以使用了。Detach方法与*CComBSTR*一起使用如下所示：

```
STDMETHODIMP CFoo::get_Name(BSTR* name)
{
  if (name==0) return E_POINTER;
  CComBSTR bsName(L"FooBar");
  *name = bsName.Detach();
}
```

CComVariant(VARIANT_TRUE)会创建一个短整型变量（type VT_I2）而不是所期望的Boolean变量类型（type VT_BOOL）。可以使用CComVariant(true)来创建一个Boolean变量。

CComVariant myVar(ipSmartPointer)会产生一个Boolean变量类型(VT_BOOL)和带有一个对象引用的非变量类型VT_UNKNOWN），象期望的一样。最好传递明确的类型到构造函数，也就是说，该类型本身不是带有重载运算符的智能类型。

```
// Perform QI if IUnknown.
IUnknownPtr ipUnk = ipSmartPointer;
// Ensure IUnknown* constructor of CComVariant is used.
CComVariant myVar2(ipUnk.GetInterfacePtr());
```

智能指针的一个通常应用是使用*Detach()*从一个方法调用返回一个对象。当返回一个接口指针时，COM标准指向方法执行内的[out]参数的增量引用数。当指针不再需要时，调用者负责调用Release。因此，必须小心以避免在一个成员变量上直接调用*Detach()*。一个典型的模式如下所示：

```
STDMETHODIMP CFoo::get_Bar(IBar **pVal)
{
  if (pVal==0) return E_POINTER;
  // Constructing a local smart pointer using another smart pointer
  // results in an AddRef (if pointer is not 0).
  IBarPtr ipBar(m_ipBar);
  // Detach will clear the local smart pointer, and the
  // interface is written into the output parameter.
  *pVal = ipBar.Detach();
  // This can be combined into one line
  // *pVal = IBarPtr(m_ipBar).Detach();
  return S_OK;
}
```

上面的模式与下面的代码有相同的结果。注意在调用AddRef之前需要对一个零指针进行条件测试，在一个零指针上调用AddRef（或任何方法）会导致一个访问异常并且通常会使应用程序崩溃：

```
STDMETHODIMP CFoo::get_Bar(IBar **pVal)
{
  if (pVal==0) return E_POINTER;
  // Copy the interface pointer (no AddRef) into the output parameter.
     *pVal = m_ipBar;
  // Make sure interface pointer is nonzero before calling AddRef.
  if (*pVal)
      *pVal->AddRef();
  return S_OK;
}
```

当使用智能指针从一个方法的[out]参数上接收一个对象时，使用智能指针“&”解除操作符。这样会导致智能指针上前一个接口指针被释放。智能指针然后被赋予新的[out]值。方法的实现会增加对象引用数，当智能指针超出范围时会被释放：

```
{
  IFooPtr ipFoo1, ipFoo2;
  ipFoo1.CreateInstance(CLSID_Foo);
```

```
    ipFoo2.CreateInstance(CLSID_Foo);

    // Initalize ipBar Smart pointer from Foo1.
    IBarPtr ipBar;
    ipFoo1->get_Bar(&ipBar);
    // The "&" dereference will call Release on ipBar.
    // ipBar is then repopulated with a new instance of IBar.
    ipFoo2->get_Bar(&ipBar);
  }
  // ipBar goes out of scope, and the smart pointer destructor calls
Release.
```

命名规则

类型名称

所有的类型名称（*class*、*struct*、*enum*和*typedef*）都以大写字母开头，剩下的部分大小写混合：

```
class Foo : public CObject { . . .};
struct Bar { . . .};
enum ShapeType { . . . };
typedef int* FooInt;
```

函数指针（callbacks）的Typedefs添加Proc到它们的名称后面。

```
typedef void (*FooProgressProc)(int step);
```

枚举数据值都以标识工程的小写字符串开始；对于ArcObjects为esri，每一个字符串出现在单独的行中：

```
typedef enum esriQuuxness
{
  esriQLow,
  esriQMedium,
  esriQHigh
} esriQuuxness;
```

函数名称

命名函数采用下面的规则：

- 对于简单的accessor和mutator函数，采用Get<Property>和Set<Property>：

```
int GetSize();
void SetSize(int size);
```

- 如果用户为结果提供存储，使用Query<Property>：

```
void QuerySize(int& size);
```

这里有一些命名规则的建议。这些帮助可以识别变量的用途和类型，并减少编码错误。这是一个删减了的匈牙利表示法：

[<scope>_]<type><name>

前缀	变量作用域
m	例程类成员
c	静态成员（包括常量）
g	全局静态变量
<empty>	局部变量或结构或公共类成员

<type>

前缀	数据类型
b	Boolean
by	字节或无符号字符
cx/cy	用做大小的短整型
d	双精度型
dw	DWORD双精度字或无符号长整型
f	浮点型
fn	函数
h	句柄
I	Int（整型）
IP	聪明指针
L	长整型
p	指针
s	字符串
sz	ASCIIZ空值终止的字符串
w	WORD无符号整型
x，y	用作坐标的短整型

<name>描述了如何使用变量或变量里包含了什么。<scope>和<type>部分通常应是小写的，并且<name>应该使用混合大小写：

变量名	描述
m_hWnd	HWND手柄
ipEnvelope	指向COM接口的聪明指针
m_pUnkOuter	指向对象的指针
m_pUnkOuter	静态类成员
g_pWindowList	指向对象的全局指针

- 对于状态函数，使用Set<State>和Is<State>或Can<State>：

```
bool IsFileDirty();
void SetFileDirty(bool dirty);
bool CanConnect();
```

- 一个操作的语义从变量类型来看是明显的地方，把类型名留在函数名的外面。

不使用：

```
AddDatabase(Database& db);
```

考虑使用：

```
Add(Database& db);
```

不使用：

```
ConvertFoo2Bar(Foo* foo, Bar* bar);
```

考虑使用：

```
Convert(Foo* foo, Bar* bar)
```

- 如果客户端放弃对象某些数据的所有权，使用Give<Property>。如果对象放弃客户端某些数据的所有权，使用Take<Property>：

```
void GiveGraphic(Graphic* graphic);
Graphic* TakeGraphic(int itemNum);
```

- 当特定的操作处理不同的自变量类型时使用函数重载：

```
void Append(const Cstring& text);
void Append(int number);
```

自变量命名

在函数声明中使用描述性的自变量名称时，自变量的名称应该清楚地表明自变量的用途：

```
bool Send(int messageID, const char* address, const char* message);
```

Developer Studio中的调试技巧

Visual C++提供了特色丰富的调试器。这些提示会帮助我们从调试过程中获得最多的信息。

故障时的后退

当一个函数调用失败时，并且想知道出错原因（通过跳入函数），不必重新启动应用程序。使用Set Next Statement命令重新设置程序光标位置回到出错的语句上（右键单击该语句打开调试弹出式菜单）。然后跳入该函数。

编辑和继续

Visual Studio 6允许在调试期间修改源代码。这些修改可以被重新编译并合并到可执行代码中，并不需要停止调试器。对于可修改的类型，有一些限制，在这种情况下，必须重新启动调试任务。缺省条件下

这个特征是激活的。在工程菜单的Settings命令中可以设置这些：单击C/C++选项卡，然后从Category下拉列表中选择General。在Debug info下拉列表中，单击Program Database for Edit and Continue。

Unicode字符串显示

要设置调试器选项来显示Unicode字符串，单击Tools菜单，单击Options，单击Debug，然后选中Display Unicode Strings复选框。

变量值显示

将光标停留在源代码中一个变量名上查看它的当前值。如果该变量是一个结构，单击该变量，弹出QuickWatch对话框（单击Eyeglasses图标或按下Shift+F9）或将之拖到Watch窗口中。

不停靠窗口

如果Output窗口（或者是任何停靠的窗口）看起来太小，试着通过右键单击该窗口并触发Docking View项目来不停靠它以使它成为一个真窗口。

有条件的断点

当满足某种条件时需要在断点上停留时，采用有条件的断点—例如，当一个for循环语句达到某一特定计数器值时。为达到这一目的，设置正常断点，然后调出Breakpoints窗口(Ctrl+B or Alt+F9)。选择设置的断点，并单击Condition按钮显示一个对话框，在其中设定断点条件。

预先加载DLLs

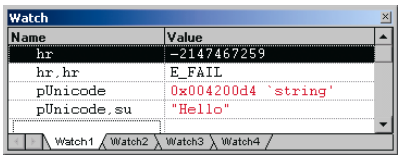
在执行程序之前可以预先加载要调试的DLLs。这样就使得你可以从一开始设置断点而不是一直等到程序执行中DLL被加载之后。操作如下，单击Project，单击Settings，单击Debug，单击Category，然后单击Additional DLLs。然后单击列表区域来添加想要预先加载的任意DLLs。

改变显示格式

可以在QuickWatch对话框中改变变量的显示格式，或在Watch窗口中使用下表中的格式符号。

符号	格式	值	显示
d, i	有正负号十进制整型	0xF000F065	-268373915
u	无正负号十进制整型	0x0065	101
o	无正负号八进制整型	0xF065	0170145
x, X	十六进制整型	61541	61541
l, h	d, l, u, o, x, X的长整型或短整型前缀	00406042, hx	0x0C22
f	有正负号的浮点数	f3./2.	1.500000
e	有正负号的科学计数	3./2.	1.500000e+00
g	e或 f中更短的一个	3./2.	1.5
c	单个字符	0x0012FDE8	"Hello"
su	Unicode字符串		"Hello"
hr	字符串	0	S_OK

使用格式符号，输入变量名后面带有逗号和适合的符号。例如，如果变量有一个0x0065值，并且想要以字符形式看这个值，在Watch窗口的列表中的Name栏输入“var,c”。当按下时Enter时，字符格式值出现：var,c = ‘e’。同样地，假定hr是一个拥有HRESULTs的变量，在Name栏中输入“hr,hr”，以人们可读的格式查看HRESULT。



可以用下表中的格式化符号格式化存储位置的内容。

符号	格式	值
ma	64位ASCII字符	0x0012ffac .4...0...″.OW&.. .....1W&.0.:W..1 ....″..1.JO&.1.2 ..″..1...0y....1
m	16字节的十六进制数，加16 ASCII字符	0x0012ffac B3 34 CB 00 84 30 94 80 FF 22 8A 30 57 26 00 00 .4...0....″.OW&..
mb	16字节的十六进制数，加16 ASCII字符	0x0012ffac B3 34 CB 00 84 30 94 80 FF 22 8A 30 57 26 00 00 .4...0...″.OW&..
mw	8位词	0x0012ffac 34B3 00CB 3084 8094 22FF 308A 2657 0000
md	四位双词	0x0012ffac 00CB34B3 80943084 308A22FF 00002657
mu	2字节字符(Unicode)	0x0012fc60 8478 77f4 ffff ffff 0000 0000 0000 0000

使用存储定位格式化符号，可以输入任何值，或计算一个位置值的表达式。要显示一个字符数组为字符串，在数组名前加一个“&”，如*&yourname*。格式化字符也可遵循一个表达式：

- *rep+1,x*
- *alps[0],mb*
- *xloc,g*
- *count,d*

查看某个地址的值，或一存储点的值，使用*BY*、*WO*或*DW*操作符：

- *BY* 字节指向的内容
- *WO* 返回字指向的内容
- *DW* 返回双字指向的内容

操作符后面跟上变量、存储器或常量。如果*BY*、*WO*或*DW*操作符后面跟上一个变量，那么环境监视包含该变量地址的字节、字或双字。

也可以使用上下文操作符{ }显示任何位置的内容。

使用su格式区分符在Watch窗口或QuickWatch对话框中显示Unicode字符串。使用mu格式区分符在Watch窗口或QuickWatch对话框中用Unicode字符显示数据字节。

键盘快捷键

大量的键盘快捷键会使对Visual Studio Editor的操作更加快捷。下面列出一些比较有用的键盘快捷键。

文本编辑器使用许多Windows应用程序(例如Word)使用的标准快捷键。下面列出了一些详细的源代码编辑快捷键。

快捷键	功能
Alt+F8	正确缩进用环绕线选中的代码
Ctrl+]	查找匹配的大括号
Ctrl+J	显示成员列表
Ctrl+Spacebar	一旦输入的字母数使得编辑者可以识别，完成字。当完成函数和变量名时使用full。
Tab	右缩进一个制表位
Shift+Tab	左缩进一个制表位

下面是在调试器中使用的通用键盘快捷键表。

快捷键	功能
F9	从当前行插入或删除断点。
Ctrl+Shift+F9	删除所有断点。
Ctrl+F9	禁止断点。
Ctrl+Alt+A	显示auto窗口，移动光标到里面。
Ctrl+Alt+C	显示调用栈，移动光标到里面。
Ctrl+Alt+L	显示本地窗口，移动光标到里面。
Ctrl+Alt+A	显示auto窗口，移动光标到里面。
Shift+F5	终止调试任务。
F11	一次执行代码一条语句，跳入函数。
F10	一次执行代码一条语句，跳过函数。
Ctrl+Shift+F5	重新启动调试任务。
Ctrl+F10	从当前语句恢复到选中语句。
F5	运行应用程序。
Ctrl+F5	不使用调试器运行应用程序。
Ctrl+Shift+F10	设置下一条语句。
Ctrl+Break	停止执行。

加载下面的快捷键会大大提高使用Visual Studio开发环境的效率。

快捷键	功能
ESC	关闭一个菜单或对话框，取消正在进行的操作，或在当前文档窗口放置焦点。
CTRL+SHIFT+N	创建新文件。
CTRL+N	创建新工程。
CTRL+F6 或 CTRL+TAB	一次循环一个MDI子窗口。
CTRL+ALT+A	显示auto窗口，将光标移入。
CTRL+ALT+C	显示call stack窗口，将光标移入。
CTRL+ALT+T	显示文档概要窗口，将光标移入。
CTRL+H	显示查询窗口
CTRL+F	显示查询窗口。如果没有当前查询规则，在查询框光标下输入字
CTRL+ALT+I	显示查询窗口并移入光标。如果实在文本编辑窗口中不可用。
CTRL+ALT+L	显示本地窗口，将光标移入。
CTRL+ALT+O	显示输出窗口，将光标移入。
CTRL+ALT+J	显示工程浏览器窗口，将光标移入。
CTRL+ALT+P	显示属性窗口，将光标移入。
CTRL+SHIFT+O	打开文件。
CTRL+P	打印文档的全部或部分内容。
CTRL+SHIFT+S	保存所有的文件、工程或文档。
CTRL+S	选择全部。
CTRL+A	保存当前文档或选中项或所有项。

浏览在线帮助主题

右键单击工具条的空白区域显示所有可用工具的列表。Infoviewer工具条包含上下箭头，通过它们可以按照帮助主题在内容列表中出现的顺序来浏览所有的主题。使用左右箭头可按照访问的顺序浏览帮助主题。

导入ARCGIS类型库

为了引用ArcGIS接口、类型和对象，需要把这些定义导入到Visual C++类型中。*#import*命令自动创建编译器所必要的文件。开发*#import*是为了支持Direct-To-Com。导入ArcGIS类型库时，必须传递大量的参数。

```
#pragma warning(push)
#pragma warning(disable : 4192) /* Ignore warnings for types that are
                                   duplicated in win32 header files.
*/
#pragma warning(disable : 4146) /* Ignore warnings for use of minus on
                                   unsigned types. */
#import "\Program Files\ArcGIS\com\esriSystem.olb"
                                /* Type library to generate C++
    wrappers.*/ \
    raw_interfaces_only,        /* Don't add raw_ to method names. */ \
    raw_native_types,           /* Don't map to DTC smart types. */ \
    no_namespace,               /* Don't wrap with C++ name space. */ \
    named_guids,                /* Named guids and declspecs. */ \
    exclude("OLE_COLOR", "OLE_HANDLE", "VARTYPE")
                                /* Exclude conflicting types. */
#pragma warning(pop)
```

*#import*的主要用途是为接口定义和GUID常量（LIBID、CLSID和IID）创建C++代码并定义智能指针。排除（“OLE_COLOR”、“OLE_HANDLE”、“VARTYPE”）是必须的，因为Windows定义这些为无正负号的长整型，与ArcGIS所定义的长整型相冲突—这一点需要支持将Visual Basic作为ArcObjects的客户端，因为Visual Basic不支持无正负号的数据。这样就不存在排除这些类型的问题。

可以在类型库的头文件（.tlh）中查看通过*#import*生成的代码，该头文件类似于a .h文件的格式。也可以查找类型库实现（.tli）文件，相当于一个a .cpp文件。这些文件可能很大不过只有当类型库改变时才重新生成。

ArcGIS 9中有许多对于不同功能区域的类型库。可以开始导入那些包含所需要定义的类型库。然而，*#import*不会自动包含所有导入类型库需要的其它定义。例如，当导入类型库esriGeometry时，它会包含对定义在esriSystem中的类型的引用，所以esriSystem必须在esriGeometry之前导入。

库依赖关系的完整列表可以在每个库的Overview主题中找到。

选择类型库的最小集有助于减少了编译时间，尽管减少通常并不显著。下面的步骤有助于确定所需类型库的最小数：

1. 编译一次并查看从代码中生成的“missing type definition”错误（例如，没有找到ICommand）。
2. 对需要引用的库放置一个*#import*语句到stdafx.h文件中。使用LibraryLocator应用程序或组件来帮助完成这项任务。
3. 再次编译工程。
4. 编译器会发布在导入的类型库中不能解决的类型错误，这些通常是类型定义，例如WKSPoint或继承到其它接口的接口。如果操作几何对象（例如点），通过导入esriGeometry开始。编译器会发布不同类型的错误信息，例如：c:\temp\sample\debug\ esrigeome-try.tlh(869)：error C2061： syntax error：identifier WKSPoint

   查看WKSPoint定义，会看到它是在esriSystem中定义的。在esriGeometry之前导入esriSystem会解决这些问题。

   下面是一个为操作ActiveX控件的典型导入列表。

```
#pragma warning(push)
#pragma warning(disable : 4192) /* Ignore warnings for types that are
                                   duplicated in win32 header files. */
#pragma warning(disable : 4146) /* Ignore warnings for use of minus on
                                   unsigned types. */
```

```
#import "\Program Files\ArcGIS\com\esriSystem.olb"
raw_interfaces_only,raw_native_types, no_namespace, named_guids,
exclude("OLE_COLOR","OLE_HANDLE", "VARTYPE")
#import "\Program Files\ArcGIS\com\esriSystemUI.olb"
raw_interfaces_only,raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\com\esriGeometry.olb"
raw_interfaces_only,raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\com\esriDisplay.olb"
raw_interfaces_only,raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\com\esriOutput.olb"
raw_interfaces_only,raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\com\esriGeoDatabase.olb"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\com\esriCarto.olb"
raw_interfaces_only,raw_native_types, no_namespace, named_guids

// Some of the Engine controls
#import "\Program Files\ArcGIS\bin\TOCControl.ocx"
raw_interfaces_only,raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\bin\ToolbarControl.ocx"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\bin\MapControl.ocx"
raw_interfaces_only,raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\bin\PageLayoutControl.ocx"
raw_interfaces_only, raw_native_types, no_namespace, named_guids

// additionally for 3D controls
#import "\Program Files\ArcGIS\com\esri3DAnalyst.olb"
raw_interfaces_only,raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\com\esriGlobeCore.olb"
raw_interfaces_only,
raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\bin\SceneControl.ocx"
raw_interfaces_only,raw_native_types, no_namespace, named_guids
#import "\Program Files\ArcGIS\bin\GlobeControl.ocx"
raw_interfaces_only,raw_native_types, no_namespace, named_guids
```

当编写包含其它类型库定义的IDL时，会出现类似的问题。在这种情况下，在库定义后使用importlib。例如，为编写一个ArcMap外部命令会要求用户创建一个实现*ICommand*的COM对象。这个定义在*esriSystemUI*中，并像下面这样导入到IDL中：

```
library WALKTHROUGH1CPPLib
{
   importlib("stdole32.tlb");
   importlib("stdole2.tlb");
   importlib("\Program Files\ArcGIS\com\esriSystemUI.olb");
   coclass ZoomIn
   {
      [default] interface IUnknown;
      interface ICommand;
   }
};
```

对于ATL的一般的论述，参见前面的部分“ATL简介”。

ATL和ACTIVEX控件

这部分包含了如何使用ATL并添加控件到对话框中的内容。尽管ATL重点是提供COM支持，但也提供了一些有用的Windows编程封装类。一个最有用的类是CWindow，它是一个窗口句柄的封装(HWND)。CWindow上的方法名称相当于Win32 API函数。例如：

```
HWND buttonHWnd = GetDlgItem( IDC_BUTTON1 ); // Get window handle of button.
CWindow myButtonWindow( buttonHWnd ); // Attach window handle to CWindow class
myButtonWindow.SetWindowText(_T("Button Title")); // Win32 function to change
                                                  button caption.
```

CWindow是所有窗口句柄的类封装，所以对于通用窗口控件的特定Windows消息，例如按钮、树视或编辑框，一种方法是直接发送窗口消息到窗口上，例如

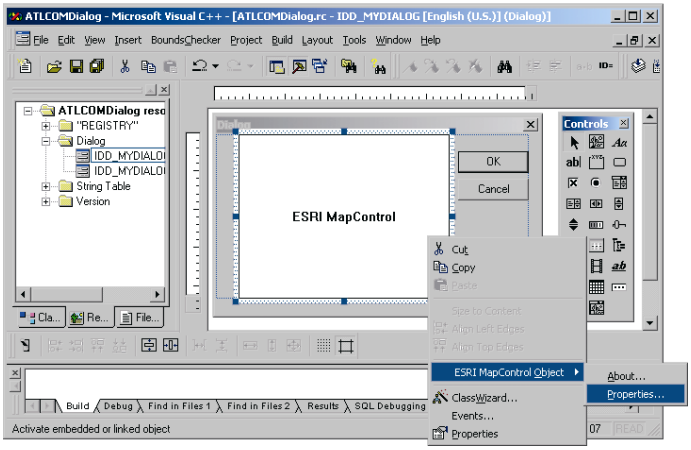
```
// Set button to be checked (pushed in or checkmarked, depending on buttonstyle)
myButtonWindow.SendMessage(BM_SETCHECK, BST_CHECKED);
```

然而，在头文件*atlcontrols.h*中有一些这些标准窗口通用控件的封装类。在MSDN中，作为提供的ATL实例ATLCON的一部分被使用。参见文章“HOWTO: Using Class Wrappers to Access Windows Common Controls in ATL”这个头文件是Windows Template Libraries (WTL)的早期版本，从Microsoft可以下载。

Visual Studio Resource Editor可用来在对话框中设计和放置通用窗口和ActiveX控件。为了创建和操作对话框，通常创建从*CAxDialogImpl*继承而来的C++类。这个类提供信息通道来在一个窗口中创建和管理ActiveX控件。ATL向导可以用来提供大多数的样板代码。下面讨论创建对话框和在一个ATL工程中添加ActiveX控件的步骤。

1. 单击菜单命令Insert/New ATL Object。
2. 单击Miscellaneous类别，然后单击Dialog对象。
3. 一个从CAxDialogImpl继承而来的对话框资源和类会添加到工程中。
4. 右键单击资源窗口中的对话框并单击Insert ActiveX Control。这样会显示一个可用的ActiveX控件的列表。
5. 双击列表中的一个控件，添加该控件到对话框中。

6. 右键单击控件，并单击 Properties 来设置控件设计时间的属性。



确认含有 ActiveX 控件的对话框从 CAxDialogImpl 而不是 CDialogImpl 继承而来。如果这里出了错，对话框的 DoModal 方法只是简单地退出且没有明显的出错原因。

确认使用通用窗口控件的应用程序（如树视）正确地调用 InitCommonControlsEx来加载窗口类。否则，类不能正确运行。

通过COM接口访问对话框上的控件

为获取一个包含在窗体中的控件的句柄，使用*GetDlgControl*，一个从*CAxDialogImpl*继承而来的ATL方法，用来获得一个资源ID并返回底层控件指针：

```
ITOCControlPtr ipTOCControl;
GetDlgControl(IDC_TOCCONTROL1, IID_ITOCControl, (void**)
&ipTOCControl);
ipTOCControl->AboutBox();
```

确认应用程序使用COM对象调用CoInitialize。这样初始化了应用程序中的COM。如果没有这个调用，任何CoCreate调用都会失败。

监听控件事件

添加事件最简单的方法是使用类向导。简单地右键单击控件并选择Events。然后，单击控件的资源ID，然后单击事件(例如，*OnMouseDown*)。然后单击Add Handler。最后，通过添加AtlAdviseSinkMap(this,TRUE)到*OnInitDialog*中来确认对话框开始监听事件。要完成监听事件，为*OnDestroy*添加消息句柄并添加一个AtlAdviseSinkMap(this,FALSE)调用。

关于在ATL中处理事件的详细地论述，参见后面“在ATL中处理COM事件“一节。

在运行时创建控件

CAxWindow类以与其它窗口类似的方式提供创建和包含ActiveX控件的机制。如果控件的父窗口也在运行时创建可能是合乎需要的。

```
AtlAxWinInit();
CAxWindow wnd;
   //m_hWnd is the parent window handle.
   //rect is the size of ActiveX control in client coordinates.
   //IDC_MYCTL is a unique ID to identify the controls window.
RECT rect = {10,10,400,300};
wnd.Create(m_hWnd, rect, _T("esriReaderControl.ReaderControl"),
WS_CHILD|WS_VISIBLE, 0, IDC_MYCTL);
```

设置伙伴控件属性

在对话框上，*ToolbarControl*和*TOCControl*需要与一个“伙伴”控件相联系。该操作通常在一个对话框*OnInitDialog*窗口消息句柄上执行。

```
LRESULT CEngineControlsDlg::OnInitDialog(UINT uMsg, WPARAM
wParam, LPARAM lParam, BOOL& bHandled)
{
   // Get the Control's interfaces into class member variables.
      GetDlgControl(IDC_TOOLBARCONTROL, IID_IToolbarControl,
(void **)
   &m_ipToolbarControl);
     GetDlgControl(IDC_TOCCONTROL, IID_ITOCControl, (void **)
   m_ipTOCControl);
     GetDlgControl(IDC_PAGELAYOUTCONTROL,
   IID_IPageLayoutControl, (void **) &m_ipPageLayoutControl);
   // Connect to the controls.
   AtlAdviseSinkMap(this, TRUE);

   // Set buddy controls.
   m_ipTOCControl->SetBuddyControl(m_ipPageLayoutControl);
   m_ipToolbarControl->SetBuddyControl(m_ipPageLayoutControl);

   return TRUE;
}
```

已知的Visual Studio C++ Resource Editor和ArcGIS ActiveX控件的局限性

属性页上无效的伙伴属性

在Visual Studio C++中不能通过General属性页设置*TOCControl*和*ToolbarControl*的“Buddy”属性。Visual C++不支持控件在设计时查找其它控件。然而这一步可在*OnInitDialog*方法的代码中可以被执行。

ToolbarControl不会调整按钮的高度

在其它的环境(Visual Basic 6, .NET)中，ToolbarControl会自动调整按钮的高度。然而，在Visual Studio C++ 6中可以是任何大小。在MFC和ATL中，ActiveX主机类不允许控件确定自身的大小

当显示上下文敏感的帮助时，设计时属性页消失

在设计时查看控件属性页时，右键单击并单击“What’s This?”会显示帮助提示；然而属性页接着就关闭了。这是Visual Studio浮动窗口与来自于HTML帮助的浮动提示窗口相结合的局限性。单击Help按钮为整个属性页提供相同的文本。

MFC和 ACTIVEX控件

在Visual C++中对于如何使用ArcGIS ActiveX Controls有多种选择。第一个选择就是框架用来包含控件的方法(例如ATL或MFC)。第二个是决定被包含控件的位置(Dialog、MDI应用程序等等)。这部分论述MFC和在对话框中包含控件。

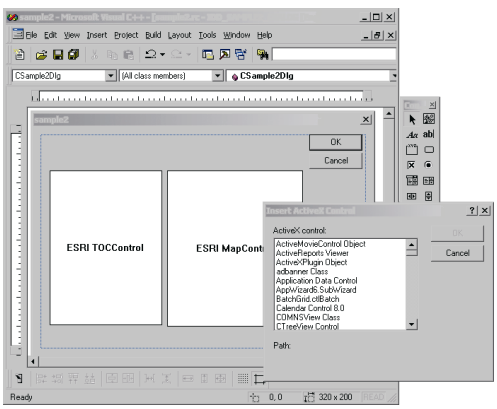
创建一个基于MFC对话框的应用程序

如果在应用程序和组件中没有对话框，这里是创建一个MFC对话框应用程序的步骤。

1. 启动Visual Studio C++ 6，并单击New。
2. 单击Projects选项卡并选择MFC AppWizard (exe)。输入工程名称和位置并单击OK。
3. 在向导的第一步：从radio按钮中，改变应用程序类型为Dialog Based。单击Next。
4. 在向导第二步：缺省工程要素是好的，虽然可以不选择AboutBox以简化应用程序。确认支持ActiveX Controls的选项被选中。单击Next。
5. 向导第三步：该页中缺省设置是很好的。MFC DLL被共享。单击Next。
6. 向导第四步：这展示了向导要生成的内容。单击Finish。

现在应该有一个简单的基于对话框的应用程序了。在资源视图中，会看到“TODO: Place Dialog Controls Here”。可以在对话框中放置按钮，列表框等等。对话框也可以容纳ActiveX控件；有两种方法可以做这个工作，如下面所讨论的。也可以编译和运行这个应用程序。

在Visual Studio C++设计时在对话框上插入ActiveX控件。TOCControl和MapControl被添加到对话框中。然后是ToolbarControl。



在对话框上包含控件并使用IDispatch访问这些控件

1. 右键单击MFC对话框，并单击Insert ActiveX控件。
2. 从列表框中双击一个控件。控件会以缺省的大小出现在对话框上。
3. 控件的大小和位置是必需的。
4. 对每个控件重复步骤1到3。
5. 可以右键单击控件并选择Properties设置控件的设计时属性。
6. 在代码中访问控件，需要IMapControl的ArcGIS接口定义。可以使用stdafx.h文件中的*#import*命令来完成这些操作。关于如何完成这些操作，参见“导入ArcGIS类型库”一节。

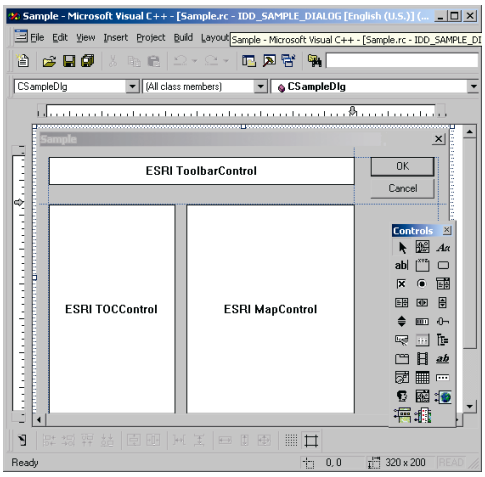
7. MFC提供了包含在对话框中的控件；这会转换Windows消息（例如WM_SIZE）为合适的控件方法调用，。然而，为了能够在控件上调用，必须执行从资源ID到控件接口的不多的几步。下面的代码演示了设置TOCControl的Buddy为MapControl：

```
// Code to set the Buddy property of the TOCControl to be the
//MapControl
// Get a pointer to the PageLayoutControl and TOCControl.
IPageLayoutControlPtr ipPageLayoutControl;
GetDlgControl(IDC_PAGELAYOUTCONTROL1,
                    IID_IPageLayoutControl, (void**)
                    &ipPageLayoutControl);
ITOCControlPtr ipTOCControl;
GetDlgControl(IDC_TOCCONTROL1, IID_ITOCControl, (void**)
&ipTOCControl);

// Get the IDispatch of the PageLayoutControl.
IDispatchPtr ipBuddyDisp = ipPageLayoutControl;

// Set the TOCControls Buddy to the map control.
ipTOCControl->putref_Buddy(ipBuddyDisp);
```

8. 为了从控件上捕获事件，在窗体上双击控件并提供要被调用的方法名。缺省条件下，向导会在事件句柄的开始添加额外的单词“On”。删除这个单词以防止事件句柄名称变成“OnOnMouseDownMapcontrol1”。这个向导然后会自动生成必要的MFC接收映射宏来监听事件。



设计环境展示了TOCControl、MapControl和ToolbarControl已经被添加到工具条和对话框中。

使用IDispatch封装添加控件到MFC对话框中

所有的控件都支持，这是添加控件到工程中的典型的方法：

1. 单击Project，单击Add，然后单击Components Controls。
2. 单击Registered ActiveX Controls。
3. 双击选择一个控件（例如，是ESRI TOCControl），然后单击OK插入一个组件。单击OK生成封装，然后单击插入一个组件。在Visual Studio中这样会为Controls工具条上的控件添加一个图标。
4. 附加的源文件被添加到工程（例如toccontrol.cpp和toccontol.h）。这些文件包含一个封装类（例如，*CTOCControl*）提供访问该控件的方法和属性。这个类会通过*IDispatch*调用机制来调用控件。注意到当调用方法和属性时，*IDispatch*不会导致任何封装参数的性能花费。封装类从包含一个ActiveX控件的MFC CWnd类继承。
5. 重复步骤1到4来添加每个控件到工程的Controls工具条中。

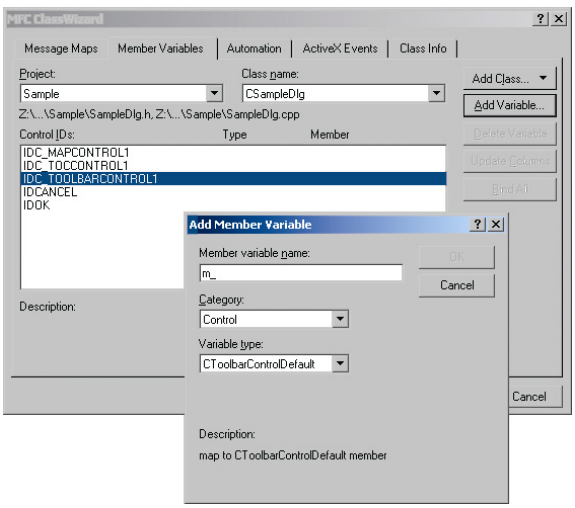
6. 从 Controls 工具条上选择一个控件并拖到对话框中。

7. 右键单击控件并单击Properties。这样允许在控件上设置设计时的属性。注意：在Visual Studio C++中，不能设置*TOCControl*和*ToolbarControl*的Buddy属性。
这个环境不支持在运行时查找其它控件的控件。然而这一步可以在代码中使用*OnInitDialog*方法执行。

```
// Note no addref performed with GetControlUnknown, so no need to
//release this pointer.
LPUNKNOWN pUnk = m_mapcontrol.GetControlUnknown();
LPDISPATCH pDisp =0;
pUnk->QueryInterface(IID_IDispatch, (void **) &pDisp);

// Set TOCControls buddy to be MapControl.
m_toccontrol.SetRefBuddy(pDisp);
pDisp->Release();
```

8. 右键单击控件并选择 Class Wizard 来启动类向导。单击 Member Variables 选项卡，并单击与控件对应的资源 ID 来提供控件成员变量名。对话框类成员变量现在可以用来调用控件的方法和属性。



Visual Studio C++ Class Wizard 为ActiveX控件添加成员变量到对话框中。

9. 为了捕获控件事件，单击类向导的 Message Maps 选项卡，并选择控件的资源 ID。在消息列表中，单击事件捕获它—例如，OnBeginLabelEdit。双击这个事件并为它添加一个句柄到对话框类中。缺省情况下，向导会在事件句柄的开头加上“On”。去掉这个避免事件句柄名称成为 OnOnBeginLabelEditToccontrol1。

不要在封装类上使用GetDispatch方法（从MFC的CcmdTarget继承）；它倾向于为对象实现IDispatch而不是正在调用IDispatch的封装类。相反要获取控件的IDispatch使用m_mapcontrol.GetControlUnknown()和IDispatch上的QueryInterface。参见上面设置Buddy属性的例子。

在ATL中处理COM事件

下面是一些在Visual C++和ATL中讨论COM事件时所使用的术语总结。

入接口—这是COM对象实现一个预定义接口的常用情况。

出接口—这是一个COM对象会在不同时间激发的方法接口。例如，*MapCoClass*会激发一个*IActiveViewEvents*上的事件以响应地图中的改动。

事件源—当某种动作发生时，源COM对象会激发事件到出站接口。例如，*MapCoClass*是一个*IActiveViewEvents*的源，并且当一个新的图层添加到地图中来时会激发*IActiveViewEvents::ItemAdded*事件。源对象可以有任意多个客户或*event sink objects*来监听事件。源对象也可以有多个出接口；例如，*MapCoClass*也可以在*IMapEvents*接口上激发事件。事件源通常情况下使用*[source]*标签在IDL中声明它的出接口。

事件汇—接听事件的COM对象被称作事件的“汇”。汇对象实现出接口；这在类型库不是总被注意到的，因为汇在内部接听事件。事件汇通常情况下使用关联点机制注册它感兴趣的源对象的事件。

关联点—作为事件源的COM对象通常情况下使用关联点机制允许汇与一个源关联。关联点接口是标准的COM接口*IconnectionPointContai-ner*和*IConnectionPoint*。

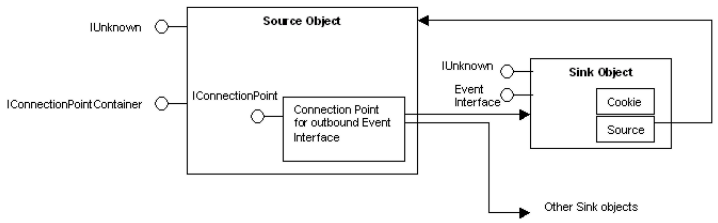
激发事件—当源对象需要通知所有有一个特定的行为的汇时，称之为源激发一个事件。这导致源迭代所有的汇，且在每个汇上调用相同的方法。例如，添加一个图层到地图上时，*Map*组件对象类激发*ItemAdded*事件。所以所有监听Map的出站*IActiveViewEvents*接口的对象都会在它们的ItemAdded方法的实现上被调用。

建议和取消建议事件—开始接收事件，一个汇对象被称作建议一个需要接受事件的源对象。当事件不再需求时，汇会对源取消建议。

ConnectionPoint机制

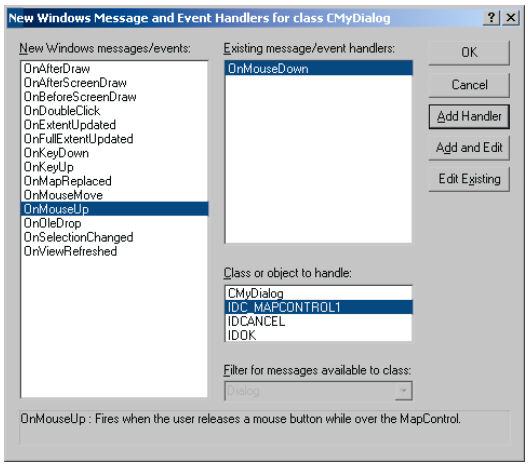
源对象实现IConnectionPointContainer接口，允许汇为特定的出站接口查询一个源。执行下面的步骤开始监听一个事件。ATL用*AtlAdvise*方法实现这个操作。

1. 汇将QI源对象的*IConnectionPointContainer*并调用*FindConnectionPoint*为出站接口提供一个接口ID。为了能够接收事件，汇对象必须实现这个接口。
2. 源可能会实现许多出站接口并会返回一个指向特定关联点对象的指针，该关联点对象实现*IconnectionPoint*来表示一个出接口。



关联源到汇对象的关联点机制

1. 汇调用*IConnectionPoint::Advise*，传递一个指向它自身*IUnknown*实现的指针。源会将这个指针与其它任何的正在侦听事件的汇一同储存。如果调用*Advise*方法成功，汇被给予一个标识符—一个简单的无符号长整型值，称作cookie—在后面不再需要监听事件时返回给源。



Visual Studio C++ Class Wizard。在一个对话框上添加ActiveX控件的事件句柄。

向导中有一个隐含的错误：不添加建议和取消建议代码到对话框中。要修复这个问题，为OnDestroy添加一个消息句柄。然后在OnInitDialog句柄上以TRUE值作为第二个参数调用AtlAdviseSinkMap开始监听事件。在句柄中放置一个相应的对AtlAdviseSinkMap的调用（以FALSE作为第二个参数）。这个在MSDN文章“BUG: ActiveX Control Events Are Not Fired in ATL Dialog(Q190530)”中进一步讨论。

现在完成了关联；可以通过源在任何正在侦听的汇上调用方法。汇通常情况下会持有一个指向源的接口指针，因此当汇完成了监听，可以通过调用*IConnectionPoint ::Unadvise*从源上释放它。这个过程用*AtlUnadvise*实现。

IDispatch事件与纯COM事件

出接口可以是一个纯分发接口。这意味着调用是通过*IDispatch::Invoke*机制实现的，而不是源直接在汇上调用方法。与纯COM调用相比较，*IDispatch*机制有封装参数的性能开销。然而，有一些情况下必须使用*IDispatch*机制。ActiveX控件必须实现它们缺省的出接口作为纯*IDispatch*接口；例如，IMapControlEvents2是一个纯分发接口。第二，Microsoft Visual Basic 6只能作为纯*IDispatch*事件的源。关联点机制对于纯COM机制也是一样的，主要的不同在于事件如何被激发。

ATL提供了一些帮助监听*IDispatch*事件的宏语言；这一点在MSDN的‘Event Handling and ATL’中论述到。有两个模板可用，*IDispEventImpl*和*IDispEventSimpleImpl*,在下面的一节会讨论到。

使用IdispEventImpl侦听事件

ATL模板*IDispEventImpl*会使用类型库来破解*IDispatch*调用并将自变量处理为C++方法调用。当添加一个ActiveX控件到对话框中时，Visual Studio Class向导自动提供这种机制。右键单击Control，单击Events。在Class向导中选择控件的资源ID，选择事件，然后单击Add Handler。下面的代码举例说明了如何通过向导添加代码处理事件，做了一些修改以保证建议和取消建议被执行。

```
#pragma once

#include "resource.h" // Main symbols
#include <atlhost.h>
```

```
////////////////////////////////////////////////////////////////////////////
// CMyDialog
class CMyDialog :
   public CAxDialogImpl<CMyDialog>,
   public IDispEventImpl<IDC_MAPCONTROL1, CMyDialog>
{
public:
   enum { IDD = IDD_MYDIALOG };

BEGIN_MSG_MAP(CMyDialog)
   MESSAGE_HANDLER(WM_INITDIALOG, OnInitDialog)

   // Add a handler to ensure event unadvise occurs.
   MESSAGE_HANDLER(WM_DESTROY, OnDestroy)

   COMMAND_ID_HANDLER(IDOK, OnOK)
   COMMAND_ID_HANDLER(IDCANCEL, OnCancel)
END_MSG_MAP()

LRESULT OnInitDialog(UINT uMsg, WPARAM wParam, LPARAM
lParam, BOOL&bHandled)
{
   // Calls IConnectionPoint::Advise() for each control on the     dialog box
with sink map entry
   AtlAdviseSinkMap(this, TRUE);
   return 1; // Let the system set the focus.
}

LRESULT OnDestroy(UINT uMsg, WPARAM wParam, LPARAM
lParam, BOOL& bHandled)
{
   // Calls IConnectionPoint::Unadvise() for each control on the    dialog
box
   with sink map entry
   AtlAdviseSinkMap(this, FALSE);
   return 0;
}

LRESULT OnOK(WORD wNotifyCode, WORD wID, HWND hWndCtl,
BOOL& bHandled)
{
   EndDialog(wID);
   return 0;
}
LRESULT OnCancel(WORD wNotifyCode, WORD wID, HWND
hWndCtl, BOOL& bHandled)
{
    EndDialog(wID);
    return 0;
}
```

```
// ATL callback from SinkMap entry
VOID __stdcall OnMouseDownMapcontrol1(LONG button, LONG shift,
LONG x,
LONG y, DOUBLE mapX, DOUBLE mapY)
{
      MessageBox(_T("MouseDown!"));
}

BEGIN_SINK_MAP(CMyDialog)
   // Make sure the Event Handlers have __stdcall calling convention.
   // The 0x1 is the Dispatch ID of the OnMouseDown method.
   SINK_ENTRY(IDC_MAPCONTROL1, 0x1,
   OnMouseDownMapcontrol1)
END_SINK_MAP()
};
```

使用IdispEventImpl时下面这些使用事件的问题在MSDN Knowledge Base中说明。对这些问题部分中ATL代码大修改也在MSDN中加以说明；然而通常修改和复制ATL头文件不是让人满意的。在这种情况下，可以使用IDispEventSimpleImpl代替。

BUG: Events Fail in ATL Containers when Enum Used as Event Parameter (Q237771) BUG: IDispEventImpl Event Handlers May Give Strange Values for Parameters (Q241810)

使用IDispEventSimpleImpl侦听事件

从这个模板的名字上可看出，它是*IDispEventImpl*的简装版。不再使用类型库将*IDispatch*变量转换为C++方法调用。虽然这是一个简单的实现过程，现在需要开发人员提供一个指向描述事件参数的格式的结构的指针。这个结构一般放置在.cpp文件中。例如，下面是一个MapControl上描述*OnMouseDown*事件参数的结构。

```
_ATL_FUNC_INFO g_ParamInfo_MapControl_OnMouseDown =
{
   CC_STDCALL,                          // Calling convention
   VT_EMPTY,                            // Return type
   6,                                        // Number of arguments
   {VT_I4, VT_I4, VT_I4, VT_I4, VT_R8, VT_R8} // VariantArgument types
};
```

参阅本附录中前面的“导入ArcGIS类型库”一节了解#import的解释。

头文件从*IDispEventSimpleImpl*继承而来并使用在SINK_MAP中不同的宏，SINK_ENTRY_INFO也需要事件接口ID，*#import*可用来定义这个符号。注意到分派接口通常以DIID为前缀而不是IID。

```
#pragma once

#include "resource.h" // Main symbols
#include <atlhost.h>

// reference to structure defining event parameters
extern _ATL_FUNC_INFO g_ParamInfo_MapControl_OnMouseDown;
//////////////////////////////////////////////////////////////////////////
// CMyDialog2
class CMyDialog2 :
public CAxDialogImpl<CMyDialog2>,
public IDispEventSimpleImpl<IDC_MAPCONTROL1, CMyDialog2,
&DIID_IMapControlEvents2>
{
public:

//Message handler code removed, it is the same as CMyDialog using IDispEventSimple
```

```
BEGIN_SINK_MAP(CMyDialog2)
    // Make sure the Event Handlers have __stdcall calling convention.
    // The 0x1 is the Dispatch ID of the OnMouseDown method.
    SINK_ENTRY_INFO(IDC_MAPCONTROL1,   // ID of event source
    DIID_IMapControlEvents2,                    // interface to listen to
    0x1,                                        // dispatch ID of MouseDown
    OnMapControlMouseDown,                      // method to call when event
    arrives
    &g_ParamInfo_MapControl_OnMouseDown)     // parameter info for method
call

END_SINK_MAP()
};
```

在COM对象上监听多个IDispatch事件接口

如果单个COM对象需要从多个*IDispatch*源上接收事件，这会造成*DispEventAdvise*方法不明确定义的编译问题。这不是出现在对话框中的一个普通问题，像*AtlAdviseSinkMap*会处理所有的连接。可以通过每次*IDispEventSimpleImpl*被继承时，引入不同的typedefs来避免歧义。下面的例子说明了一个称作*CListen*COM对象，该对象是一个从*MapControl*和*PageLayoutControl*分发事件的汇。

```
#pragma once

#include "resource.h" // Main symbols

// This is the parameter information
extern _ATL_FUNC_INFO g_ParamInfo_MapControl_OnMouseDown;
extern _ATL_FUNC_INFO
g_ParamInfo_PageLayoutControl_OnMouseDown;

//
// Define some typedefs of the dispatch template.

//
class CListen; // Forward definition

typedef IDispEventSimpleImpl<0, CListen, &DIID_IMapControlEvents2>
        IDispEventSimpleImpl_MapControl;
typedef IDispEventSimpleImpl<1, CListen,
        &DIID_IPageLayoutControlEvents>
        IDispEventSimpleImpl_PageLayoutControl;

////////////////////////////////////////////////////////////////////////////
// Clisten

class ATL_NO_VTABLE CListen :
```

```
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CListen,&CLSID_Listen>,
    public IDispEventSimpleImpl_MapControl,
    public IDispEventSimpleImpl_PageLayoutControl,
    public IListen
{
public:
    CListen()
    {
    }

DECLARE_REGISTRY_RESOURCEID(IDR_LISTEN)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CListen)
    COM_INTERFACE_ENTRY(IListen)
END_COM_MAP()
// Associated source and dispatchID to a method call
BEGIN_SINK_MAP(CListen)
    SINK_ENTRY_INFO(0,                          // ID of event source
    DIID_IMapControlEvents2,                    // Interface to listen to
    0x1,                                        // Dispatch ID to receive
    OnMapControlMouseDown,                      // Method to call when
event arrives
    &g_ParamInfo_MapControl_OnMouseDown)    // Parameter info for
                                                // method call
    SINK_ENTRY_INFO(1,
        DIID_IPageLayoutControlEvents,
        0x1,
        OnPageLayoutControlMouseDown,
        &g_ParamInfo_PageLayoutControl_OnMouseDown)
END_SINK_MAP()

// IListen
public:
    STDMETHOD(SetControls)(IUnknown* pMapControl, IUnknown*
pPageLayoutControl);
    STDMETHOD(Clear)();
private:
    void __stdcall OnMapControlMouseDown(long button, long shift, long
x, long
y, double mapX, double mapY);
    void __stdcall OnPageLayoutControlMouseDown(long button, long shift,
long
x, long y, double pageX, double pageY);
    IUnknownPtr m_ipUnkMapControl;
    IUnknownPtr m_ipUnkPageLayoutControl;
};
```

CListen的实现包含下面的代码来启动侦听控件；使用typdef避免了*DispEventAdvise*实现的歧异。

```
// Start listening to the MapControl.
IUnknownPtr ipUnk = pMapControl;
HRESULT hr =
IDispEventSimpleImpl_MapControl::DispEventAdvise(ipUnk);
if (SUCCEEDED(hr))
   m_ipUnkMapControl = ipUnk; // Store pointer to MapControl for
   Unadvise
   .
// Start listening to the PageLayoutControl.
ipUnk = pPageLayoutControl;
hr =
IDispEventSimpleImpl_PageLayoutControl::DispEventAdvise(ipUnk);
if (SUCCEEDED(hr))
   m_ipUnkPageLayoutControl = ipUnk; // Store pointer to
   PageLayoutControl
                                          // for Unadvise.
```

CListen的实现包含下面的代码来UnAdvise和停止监听控件。

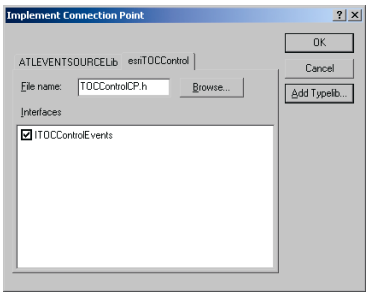
```
// Stop listening to the MapControl.
if (m_ipUnkMapControl!=0)
   IDispEventSimpleImpl_MapControl::DispEventUnadvise(m_ipUnkMap
Control);
m_ipUnkMapControl = 0;

if (m_ipUnkPageLayoutControl!=0)
   IDispEventSimpleImpl_PageLayoutControl::DispEventUnadvise(m_ipU
nkPageLayoutControl);
m_ipUnkPageLayoutControl= 0;
```

创建COM事件源

作为事件源的对象，需要提供一个*IConnectionPointContainer*的实现和一个关于哪个汇侦听哪个*IConnectionPoint*接口的跟踪机制。ATL通过*IConnectionPointContainerImpl*模板提供这些。另外ATL提供了一个向导，为给定的分发事件接口的所有成员生成激发*IDispatch*事件的代码。下面的步骤修改一个ATL COM组件对象类来支持关联点：

1. 首先确保用户的ATL组件对象类已经被编译了至少一次，使向导能找到初始的类型库。
2. 在Class视图中，右键单击COM对象并单击Implement Connection Point。
3. 要么使用工程中的IDL上的事件定义，要么单击Add Type Lib浏览其它定义。
4. 在组件对象类中选中要被实现的出接口。



5. 单击OK会修改用户的ATL类，并在头文件中生成一个名称以CP结尾的代理类，用来激发事件。

如果向导运行失败，使用下面的例子，这个例子说明了作为ITOCControlEvents的源，一个纯分发接口的组件对象类。

```
#pragma once

#include "resource.h"                    // Main symbols
#include "TOCControlCP.h"                // Include generated connection
point class
                                          // for firing events.

//////////////////////////////////////////////////////////////////////////
// CMyEventSource
class ATL_NO_VTABLE CMyEventSource :
   public CComObjectRootEx<CComSingleThreadModel>,
   public CComCoClass<CMyEventSource,&CLSID_MyEventSource>,
   public IMyEventSource,
   public CProxyITOCControlEvents< CMyEventSource >, // Generated
                                                     // ConnectionPoint class
   public IConnectionPointContainerImpl< CMyEventSource > //
   Implementation
                                                  // of Connection point Container
{
public:
   CMyEventSource()
   {
   }

DECLARE_REGISTRY_RESOURCEID(IDR_MYEVENTSOURCE)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CMyEventSource)
   COM_INTERFACE_ENTRY(IMyEventSource)
   COM_INTERFACE_ENTRY(IConnectionPointContainer) // Allow QI
to this
                                                              // interface.
END_COM_MAP()
```

```
// List of available connection points
BEGIN_CONNECTION_POINT_MAP(CMyEventSource)
   CONNECTION_POINT_ENTRY(DIID_ITOCControlEvents)
END_CONNECTION_POINT_MAP()
};
```

关联点类（在上面的类中是*TOCControlEventsCP.h*）包含了激发关联点上所有汇对象事件的代码。

在这个类中对于每个事件开始都有一个“Fire_”方法。每个方法都建立一个变量列表参数来作为一个自变量传递给分发Invoke方法。每个汇被跌代，且指向汇的指针存储在从*IConnectionPointContainerImpl*继承而来的一个m_vec成员变量中。注意到m_vec可能包含了指向零的指针；在激发事件前必须进行检查。

```
template <class T>
class CProxyITOCControlEvents : public IConnectionPointImpl<T,
&DIID_ITOCControlEvents, CComDynamicUnkArray>
{
public:
   VOID Fire_OnMouseDown(LONG button, LONG shift, LONG x,
   LONG y)
   {
      // Package each of the parameters into an IDispatch argument list.
      T* pT = static_cast<T*>(this);
      int nConnectionIndex;
      CComVariant* pvars = new CComVariant[4];
      int nConnections = m_vec.GetSize();

      // Iterate each sink object.
      for (nConnectionIndex = 0; nConnectionIndex < nConnections;
nConnectionIndex++)
      {
         pT->Lock();
         CComPtr<IUnknown> sp = m_vec.GetAt(nConnectionIndex);
         pT->Unlock();
         IDispatch* pDispatch = reinterpret_cast<IDispatch*>(sp.p);

         // Note m_vec can contain 0 entries so it is important to check for
         this.
      if (pDispatch != NULL)
         {
         // Build up the argument list.
         pvars[3] = button;
         pvars[2] = shift;
         pvars[1] = x;
         pvars[0] = y;
         DISPPARAMS disp = { pvars, NULL, 4, 0 };
         // Fire the dispatch method, 0x1 is the DispatchId for MouseDown.

         pDispatch->Invoke(0x1, IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD, &disp, NULL, NULL, NULL);
```

```
            }
        }
        delete[] pvars; // Clean up the parameter list.

    }
    VOID Fire_OnMouseUp(LONG button, LONG shift, LONG x, LONG y)
    {
      // ... Other events
```

从源上激发事件，只是在遇到请求时简单地调用*Fire_OnMouseDown*。对于激发事件到一个纯COM（non IDispatch）接口可采用类似的方法。向导不会产生关联点类，所以必须手工编写；下面的例子说明了一个激发*ITOCBuddyEvents::ActiveViewReplaced*事件的类；*ITOCBuddyEvents*是一个纯COM非IDispatch接口。主要的不同是不需要封装参数。可以直接进行方法调用。

```
template < class T >
class CProxyTOCBuddyEvents : public IConnectionPointImpl< T,
&IID_ITOCBuddyEvents, CComDynamicUnkArray >
{
    // This class based on the ATL-generated connection point class
public:
    void Fire_ActiveViewReplaced(IActiveView* pNewActiveView)
    {
    T* pT = static_cast< T* >(this);
    int nConnectionIndex;
    int nConnections = this->m_vec.GetSize();
    for (nConnectionIndex = 0; nConnectionIndex < nConnections;
    nConnectionIndex++)
      {
          pT->Lock();
          CComPtr< IUnknown >
          sp=this->m_vec.GetAt(nConnectionIndex);
          pT->Unlock();
          ITOCBuddyEvents* pTOCBuddyEvents = reinterpret_cast<
               ITOCBuddyEvents* >(sp.p);
          if (pTOCBuddyEvents)
               pTOCBuddyEvents->ActiveViewReplaced(pNewActiveVie
          w);
      }
    }
};
```

支持事件对象的IDL声明

对象被导出到类型库时，通过使用相对接口名称的*[source]*标签来声明事件接口。例如，激发*ITOCBuddyEvents*的对象声明为

```
[source] interface ITOCBuddyEvents;
```

如果出站接口是分发事件接口，使用*dispinterface*代替*interface*。另外，组件对象类可能会有一个缺省的出站接口；使用*[default]*标签

来指定。缺省接口通过某些设计环境（例如Visual Basic 6）来识别。下面是缺省出站事件接口的声明：

```
[default, source] dispinterface IMyEvents2;
```

事件循环引用问题

汇执行完源上的Advise后，常常会有一个COM循环引用。出现这种情况是因为源有一个指向汇的接口指针来激发事件，这使得汇保持继续活动。类似地，汇对象有一个指回源的指针，因此可以在后面执行Unadvise。这使得源保持继续活动。然而这两个对象永远不会被释放并会导致大量的内存泄露。有多种方法可以处理这个问题：

1. 确保Advise和Unadvise在一个方法或窗口消息中执行，这样保证成对发生，并且独立于一个对象的生命周期。例如，在一个接收窗口消息的组件对象类上，使用Windows消息*OnCreate*(WM_CREATE)和OnDestroy(WM_DESTROY)来执行Advise和Unadvise。

2. 如果ATL对话框类需要侦听事件，一种方法是使对话框为一个私有的COM类，并直接在对话框上实现事件接口。ATL不需要额外的代码就可完成这些。这种方法在下面举例说明。对话框类创建一个*CustomizeDialog*组件对象类，并监听*ICustomizeDialogEvents*。方法*OnInitDialog*和*OnDestroy*用来在*CustomizeDialog*上建议和取消建议。

```
class CEngineControlsDlg :
    public CAxDialogImpl<CEngineControlsDlg>,
    public CComObjectRoot, // Make Dialog Class a COM Object as
    well.
    public ICustomizeDialogEvents // Implement this interface
    directly on this object.

CEngineControlsDlg() : m_dwCustDlgCookie(0) {} // initialize
                                                cookie for
                                                event
                                                listening
// ... Event handlers and other standard dialog code has been
removed ...

BEGIN_COM_MAP(CEngineControlsDlg)
    COM_INTERFACE_ENTRY(ICustomizeDialogEvents) // Make
    sure QI works for
                                                    // this
    event interface.
END_COM_MAP()

    // ICustomizeDialogEvents implementation to receive events on
    this
    // dialog box.
    STDMETHOD(OnStartDialog)();
    STDMETHOD(OnCloseDialog)();

    ICustomizeDialogPtr m_ipCustomizeDialog; // The source of
    events
    DWORD m_dwCustDlgCookie;            // Cookie for
                                        // CustomizeDialogEvents
}
```

对话框需要像非创建COM对象那样来创建，而不能像一个局部变量一样在堆栈上创建。在堆栈上分配对象并允许通过COM引用计数机制释放。

```
// Create dialog class on the heap using ATL CComObject template.
CComObject<CEngineControlsDlg> *myDlg;
CComObject<CEngineControlsDlg>::CreateInstance(&myDlg);

myDlg->AddRef();    // Keep dialog box alive until you're done with it.
myDlg->DoModal(); // Launch the dialog box; when method returns,
   dialog box has
                        //exited.
myDlg->Release();    // Typically, the refcount now goes to 0 and frees
                        // the dialog object.
```

3. 通过汇为使用实现一个中间的COM对象，这种情况有时候被称作监听者或事件帮助对象。该对象通常不包含实现，而只是简单地调用C++方法传送事件到汇对象。监听者通过源增加引用计数，但是汇的引用数是不受影响的。这样就打破了循环，当所有的其它引用被释放时，允许汇的引用数达到0。当汇执行它的析构函数代码时，指示它的监听者执行Unadvise并释放源。

使用C++指针在监听者和汇之间通信的一个备选方案是使用一个弱引用的接口指针。也就是说，监听者包含一个指向汇的COM指针但是不增加汇的引用数。汇负责保证汇对象被释放后这个指针不能再被访问。

什么是.NET框架？

本节“什么是.NET框架”总结了Microsoft对.NET Framework的概述，作为MSDN库的部分可以在线获得。整个文档可以在下面的网址获得：
http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpovrintroductiontonetframeworksdk.asp

.NET框架是一套完整的Windows组件，支持下一代应用程序和XML Web服务的创建和运行。.NET框架是为实现下列目标而设计的：

- 提供一致的面向对象编程环境，而无论对象代码是在本地储存和执行，本地执行但在互联网上分布，还是远程执行。
- 提供一个最小化软件部署和版本冲突的代码执行环境。
- 提供保证安全执行代码，包括未知或半信任的第三方创建的代码执行环境。
- 提供消除脚本化或解释环境带来的性能问题的代码执行环境。
- 使得开发人员编程经验跨越广泛类型的应用程序取得一致性，例如基于Windows的应用程序和基于Web的应用程序。
- 在工业标准之上建立所有通信确保基于.NET框架的代码可以与其它代码集成。

.NET框架有两个主要组件：通用语言运行时和.NET框架类库。通用语言运行时是.NET框架的基础。可以将运行时看作是在执行时管理代码的智能体，提供诸如内存管理、线程管理和远程等核心服务，同时也加强了严格的类型安全和其它形式的代码精度以确保安全性和健壮性。事实上，代码管理的概念是运行时的基本原理。以运行时为目标的代码被称作被管理代码，而不以运行时为目标的代码被称为未被管理的代码。类库，.NET框架的另外一个主要组件，是一个全面的、面向对象的可重用类型的集合，可以用它来开发各种应用程序，包括从传统的命令行或图形用户界面应用程序到基于ASP.NET提供的最新技术上的应用程序，例如Web Forms和XML Web服务。

.NET框架可以作为未被管理的组件的主机，这些组件加载通用语言运行时到它们的进程中并初始化被管理代码的执行，因此创建了一个软件环境既可以使用被管理的要素也可以使用未被管理的要素。.NET框架不仅提供几个运行时主机，也支持第三方运行时主机的开发。

例如，ASP.NET包含运行时为被管理代码提供一个可缩放的、服务器端环境。ASP.NET直接操作运行时激活ASP.NET应用程序和XML Web服务，这两个在后面都要讨论到。

Internet Explorer是一个包含运行时的未被管理应用程序的例子（以MIME类型扩展的形式）。使用Internet Explorer包含运行时，使得用户可以在HTML文档中嵌入被管理的组件或Windows窗体控件。以这种方式包含运行时，使得被管理的移动代码（类似于Microsoft ActiveX

控件）成为可能，并且有了很大的改善，这些只有被管理的代码能提供，例如半信任执行和安全独立文件存储。
下面这些部分更详细地描述了.NET框架的主要组件和特性。

通用语言运行时的特性

通用语言运行时管理内存、线程执行、代码执行、代码安全性校验、编译和其它系统服务。这些特性是运行在通用语言运行时的被管理代码的本质属性。

关于安全性，被管理组件被授予不同等级的信任，这取决于大量的因素，包括这些组件的来源，例如Internet，企业网或本地计算机。这意味着被管理的组件可能或者不可能执行文件访问操作、注册表访问操作或其它敏感的功能，即使在相同的活动应用程序中使用。

运行时加强了代码访问安全性。例如，用户可以信任嵌入在一个Web页的可执行程序可以放映一个屏幕动画或唱一首歌，但不能访问它们的个人数据、文件系统或网络。运行时安全特性这样使得合法的Internet-deployed软件特征格外丰富。

运行时也通过实现一个叫做通用类型系统（CTS）的严格的type-and-code-verification基础结构增强了代码的健壮性。CTS确保了所有被管理代码都是自描述的。不同的Microsoft和地方语言编译器生成符合CTS的被管理代码。这意味着被管理代码能够消耗其它的被管理类型和例程，然而严格地加强了类型保真度和类型安全性。

此外，运行时的被管理环境除去了许多通用软件问题。例如，运行时自动地使用对象设计和管理对象的引用，当它们不再被使用时释放它们。这个自动内存管理解决了两个最常见的应用程序错误：内存泄漏和无效内存引用。

运行时也提高了开发人员的生产效率。例如，程序员可以选择编写应用程序的开发语言，仍可以利用其它的开发人员用其它语言编写的运行时、类库和组件。任何选择以运行时为目标的编译器的卖主可以完成这些工作。以.NET框架为目标的语言编译器使得用某种语言编写的现有代码可以利用.NET框架的特性，这使得对现有应用程序的移植处理变得十分的容易。

虽然运行时是为未来的软件设计的，但它也支持现在的和过去的软件。在被管理的和未被管理的代码之间的互操作使得开发人员可以继续使用必要的COM组件和DLLs。

运行时是为提高性能而设计的。虽然通用语言运行时提供了许多标准运行时服务，被管理代码从来都没有被解释过。一个称为just-in-time

(JIT)编译的特性使得所有被管理的代码在本地机器语言系统上执行。其间内存管理器消除了内存破碎化的可能性并提高了内存引用的局部性，从而进一步提高了性能。

最后，运行时可以被高性能的服务器端应用程序使用，例如Microsoft SQL Server™和Internet Information Services (IIS)。这个结构使得用户可以使用被管理的代码编写自己的事务逻辑，但仍可以享用支持拥有运行时的业界最好企业服务器的出众性能。

.NET框架类库

.NET框架类库是一个与通用语言运行时紧密结合在一起的可重用类型的集合。类库面向对象，提供用户自己的被管理代码可以从中获得功能的类型。这不仅使得.NET框架类型易于使用，也减少了与学习.NET框架的新特性有关的时间。此外，第三方组件可以与NET框架类无缝结合。

例如，.NET框架集合实现了一套接口，可以使用这些接口开发自己的集合类。自己的集合类会与.NET框架中的类无缝混合。

如同用户对面向对象类库所期望的一样，.NET框架使得用户可以完成大量的常用编程任务，包括字符串管理、数据收集、数据库连接和文件访问。除了这些常用的任务外，类库包括了支持大量专门化开发场景的类型。例如，可以使用.NET框架开发下面类型的应用程序和服务：

- 控制台应用程序
- Windows GUI应用程序（Windows Forms）
- ASP.NET应用程序
- XML Web服务
- Windows服务

例如，Windows Forms类是一个可重用类型的完整集合，它们极大地简化了Windows GUI程序的开发。如果编写一个ASP.NET Web Form应用程序，可以使用Windows Forms类。

客户端应用程序开发

在基于Windows的程序开发中，客户端应用程序最接近于传统风格的应用程序。这些程序为在桌面上显示窗口或窗体使得用户可以执行一项任务的应用程序类型。客户端应用程序包括那些文字处理和电子表格数据，也包括顾客商业应用程序，例如数据记录和报告工具。客户端程序通常使用窗口、菜单、按钮和其它的GUI元素，并且有可能访问本地资源，例如文件系统和外部设备如打印机。

另外一种客户端应用程序是传统的ActiveX控件（现在被管理的Windows Forms控件代替），作为一个Web页被配置在Internet上。这个应用程序很像其它的客户端应用程序：本地执行，可访问本地资源，包括图形元素。

在过去，开发人员使用C或C++结合Microsoft Foundation Classes或使用快速应用程序开发(RAD)环境（例如Microsoft Visual Basic）来创建这样的应用程序。.NET框架将这些现有产品的各个方面合并成为一个单一的一致的开发环境，大大简化了客户端程序的开发。

包含在.NET框架中的Windows Forms类设计用于GUI开发。可以简单地创建具有必要的灵活性的命令窗口、按钮、菜单、工具条和其它的屏幕元素，以适应多变的商业需求。

例如，.NET框架提供了调整与窗体有关的视觉属性的简单属性。在某些情况下底层的操作系统不支持直接修改这些属性，并且在这些情况下.NET框架自动重建窗体，这是.NET框架结合开发人员接口使得编码更简单和更一致的许多方法中的一个。

与ActiveX控件不同，Windows Forms控件可以半信任地访问一个用户的计算机。这意味着二进制或本地执行代码可以访问用户的系统上的某些资源，例如GUI元素和有限制的文件访问，而不能访问或危及其它资源的安全。因为代码访问安全性，许多过去需要在用户系统上安装的应用程序现在可以通过Web安全地配置。用户的应用程序配置时像展开一个Web页一样，实现一个本机应用程序的特征。

服务器应用程序开发

服务器端应用程序在被管理的世界中通过运行时主机实现。未被管理的应用程序拥有通用语言运行时，这允许用户的客户端管理代码控制服务器行为。这个模型提供给用户通用语言运行时的所有特征和获得主机服务器的性能与可伸缩性时的类库。

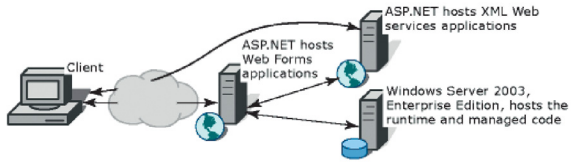
服务器端管理的代码

ASP.NET是开发人员使用.NET框架开发基于Web应用程序的主机环境。然而，ASP.NET不仅是一个运行时主机；它是一个使用被管理代码开发Web站点和互联网分布式对象的完整结构。Web Forms和XML Web服务都使用IIS和ASP.NET作为应用程序的发布机制，两者都有一个在.NET 框架中支持类的集合。

XML Web服务，基于Web技术中的一个重要进展，是分布式的、服务器端的应用程序组件，它类似于通用Web站点。然而与基于Web的应用程

序不同，XML Web服务组件没有UI并且也不是以浏览器（例如Internet Explorer和Netscape Navigator）为目标的，。相反，XML Web服务包含可重用的软件组件，设计为可被其它的应用程序（如传统的客户端应用程序、基于Web的应用程序，或其它的XML Web服务）使用。结果，XML Web服务技术是快速移植应用程序开发和配置到高度分布的Internet环境的技术。

如果已经使用过ASP技术的早期版本，很快就会注意到ASP.NET和Web Forms所提供的改进。例如，可以在支持.NET框架的任何语言下开发Web Forms页。此外，代码也不再需要与HTTP文本（虽然如果你愿意，也可继续这样做）共享相同的文件。Web Forms页在本地机器语言上执行，因为和其它的被管理应用程序一样，可以充分地利用运行时。与此相对，未被管理的ASP页常常是脚本化的的和解释性的。ASP.NET也更加快速，更多功能，并且比未被管理的ASP页更容易开发，因为像其它的被管理应用程序一样，它们与运行时交互。.NET框架也提供了一个类和工具的集合，这些类和工具有助于开发和使用XML Web服务应用程序。XML Web服务建立在一些标准之上，如SOAP，一个远程过程调用协议；XML，一个可扩展的数据格式；和WSDL，Web Services Description Language。NET框架建立在这些标准上提高与非Microsoft解决方案互操作能力。



本图说明了一个基础网络模式，使用运行在不同服务器环境下的被管理的代码。服务器（如IIS和SQL Server）可以执行标准的操作，然而用户应用程序的逻辑执行被管理的代码。

例如，包括.NET框架SDK的Web Services Description Language工具可以查询一个发布在Web上的XML Web服务，解析它的WSDL描述并产生C#或Visual Basic源代码，这样，应用程序可使用这些代码成为服务的一个XML Web客户端。这些源代码可以从类库的类创建类，这个类库使用SOAP和XML解析操纵所有底层的通信。虽然可以使用类库直接使用XML .NET框架 Web服务，Web Services Description Language工具和包含在SDK中的其它工具促进了使用.NET框架开发的工作。

如果开发和发布了自己的XML Web服务，.NET框架提供了符合有底层通信标准的一套类，这些标准有SOAP、WSDL和 XML等。使用这些类使得用户能集中在自己服务的逻辑性上，而不关心分布式软件开发所要求的通信基础结构。

最后，像在被管理的环境中的Web Forms，用户的XML Web服务会使用可伸缩的IIS通信以本地机语言的速度运行。

与COM互操作

运行在.NET框架控件上的代码称作被管理的代码；相反，运行在.NET框架之外的代码称作未被管理的代码。COM是一个未被管理代码的例子。.NET框架通过一个被称作COM Interop的技术与COM交互。

为了使用COM Interop，CLR需要所有COM类型的元数据。这意味着COM类型定义通常储存在需要转换为元数据的类型库中。使用Type Library Importer（tlbimp.exe）应用程序，用.NET框架SDK装载，很容易完成这些任务。这个应用程序生成了包含一个类型库中所有COM定义元数据的interop集合。一旦这些元数据可用，.NET客户就可以无缝地创建COM类型的例程并调用该例程方法，就像本地.NET例程一样。

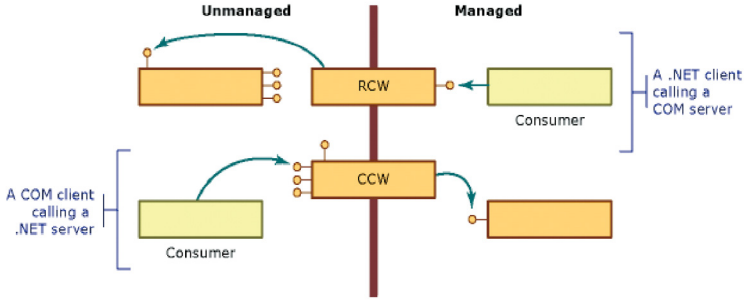
主要互操作（interop）集合

主要互操作集合（PIAs）是正式的，供应商提供的，为与底层的COM类型交互而定义的.NET类型。主要互操作集合被COM库发行者强硬地命名的，以保证唯一性。

ESRI为实现COM的所有类型库提供了主要互操作集合。如果检测到.NET框架的1.1版本，ArcGIS .NET开发人员应该只使用这些在安装过程中安装在Global Assembly Cache （GAC）上的主要互操作集合。ESRI仅支持那些装载ArcGIS的主要互操作集合。可以通过公共键（8FC3CC631E44AD86）识别一个有效的ESRI集合。

COM包装

.NET运行时提供了包装类，使得被管理的和未被管理的客户认为它们正在各自的环境中与对象通信。当被管理的用户在COM对象上调用一个方法时，运行时创建了一个运行时可调用包装（RCW）来操纵两个环境之间的列集。同样.NET运行时为相反的情况创建COM可调用包装，COM客户与.NET组件通信。上面的图示说明了这个过程。



向COM提供.NET组件

创建COM客户端要使用的.NET组件时，要遵循下面的原则以确保互操作性。

- 避免使用参数化的构造函数。
- 避免使用静态方法。

- 用被管理的代码定义事件源接口。
- 在用户定义的异常中包括HRESULTs
- 为需要的类型提供GUIDs
- 期望继承差异。

要了解更多的信息，请查看MSDN帮助集中的“Interoperating with Unmanaged Code”。

性能考虑

COM Interop增加了添加一个新的开销层到应用程序，但是在COM和.NET之间互操作的总体开销很小，所以这个开销往往是察觉不到的。然而，创建包装和在不同环境之间列集的开销确实会累加起来；如果怀疑COM Interop是应用程序性能的瓶颈，试着创建一个COM worker类，将所有的COM调用封装在一个可被管理代码调用的函数中。这样通过限制两个环境之间的调度改善了性能。

COM到.NET的类型转换

通常来说，类型库导入者导入与最初在COM中名字相同的类型。所有导入的类型都添加到一个遵循下面命名规则的命名空间中：ESRI.ArcGIS加上库名。例如Geometry库的命名空间是ESRI.ArcGIS.Geometry。所有类型通过它们的完整的命名空间和类型名识别。

类、接口和成员

所有的组件对象类被转换成为被管理的类；被管理的类名与起初的类名加上‘Class’相同。例如，Point组件对象类的名称是PointClass。

所有类也有一个和组件对象类相同名称的接口，该接口对应于组件对象类缺省接口。例如，PointClass有一个Point接口。类型库导入者添加这个接口，这样客户端可以注册为事件汇。

.NET类也有一个.NET支持但COM不支持的类成员。该类实现的每个接口的每个成员被作为一个类成员添加进来。从一个类移植的任何属性和方法都可直接从类访问而不必强制转换到一个特殊的接口。因为接口成员命名不是唯一的，所以命名冲突要通过接口名加前缀和给每个冲突成员加下划线解决。当成员命名冲突时，列在组件对象类中的第一个接口不会改变。

C#中有以引用传递或多个参数的属性不支持规则的属性语法。在这种情况下，有必要使用accessor方法。下面摘录的代码展示了一个实例。

```
        ILayer layer = mapControl.get_Layer(0);
    MessageBox.Show(layer.Name);
```

事件

类型库导入者创建了几个类型，使得被管理的应用程序接听由COM类激

发的事件。第一个类型是delegate，以事件接口加上下划线加上事件名再加上EventHandler命名。例如，在*IActiveViewEvents*接口上定义的*SelectionChanged*事件有下面这样定义的delegate：*IActiveViewEvents_SelectionChangedEventHandler*。导入者也使用起初接口名加后缀‘_Event’来创建一个事件接口。例如，*IActiveViewEvents*生成*IActiveViewEvents_Event*。使用事件接口设立事件汇。

非OLE自适应类型

非OLE自适应的COM类型通常不能在.NET下操作。ArcGIS包含了几个非自适应的方法并且这些方法不能在.NET中使用。然而在大多数情况下，追加的接口被添加进来，使得不和规定的成员被自适应地改写。例如，当通过一个点数组定义一个外包矩形时，不能使用*IEnvelope::DefineFromPoints*，相反，必须使用*IEnvelopeGEN::DefineFromPoints*。

[VB.NET]

```
Dim pointArray(1) As IPoint
pointArray(0) = New PointClass
pointArray(1) = New PointClass
pointArray(0).PutCoords(0, 0)
pointArray(1).PutCoords(100, 100)

Dim env As IEnvelope
Dim envGEN As IEnvelopeGEN
env = New EnvelopeClass
envGEN = New EnvelopeClass

'Won't compile
'env.DefineFromPoints(2, pointArray)

'Doesn't work
env.DefineFromPoints(2, pointArray(0))
'Works
envGEN.DefineFromPoints(pointArray)
```

[C#]

```
IPoint[] pointArray = new IPoint[2];
pointArray[0] = new PointClass();
pointArray[1] = new PointClass();
pointArray[0].PutCoords(0,0);
pointArray[1].PutCoords(100,100);

IEnvelope env = new EnvelopeClass();
IEnvelopeGEN envGEN = new EnvelopeClass();

// Won't compile
env.DefineFromPoints(3, ref pointArray);
```

```
// Doesn't work
env.DefineFromPoints(3, ref pointArray[0]);

// Works
envGEN.DefineFromPoints(ref pointArray);
```

.NET编程技巧和要考虑的问题

这部分包含了一些编程提示和技巧，提供给.NET的开发人员。

接口间强制转换

.NET使用强制转换在同一个类上从一个接口跳跃至另一个接口。在COM中叫做QueryInterface。VB .NET和C#的强制转换不同。

VB .NET

有两种类型的强制转换，隐式的和显式的。隐式的强制转换不要求附加语义，然而显式的强制转换需要强制转换操作符。

```
geometry = point 'Implicit cast
geometry = CType(point, IGeometry) 'Explicit cast
```

在接口之间强制转换时，使用隐式的强制转换是完全可接受的，因为在数字类型间强制转换时不会有数据丢失。然而，强制转换失败时，掷出异常（System.InvalidCastException）；为避免处理不必要的异常，如果对象实现两种接口时，最好预先进行检验。推荐的技巧是使用*TypeOf*关键字，这是一个比较从句，检验是否一个对象源于或实现一个特定类型，例如接口。下面的例子执行了一个从IPoint到IGeometry的隐式转换，只有在运行时确定Point类实现*IGeometry*。

```
Dim point As New PointClass
Dim geometry As IGeometry
If (TypeOf point Is IGeometry) Then
        geometry = point
End If
```

如果更喜欢在使用Option Strict On语句限定隐式的转换，使用CType函数进行显式的强制转换。下面的例子添加了一个显式的强制转换到上面的代码例子中。

```
Dim point As New PointClass
Dim geometry As IGeometry
If (TypeOf point Is IGeometry) Then
        geometry = CType(point, IGeometry)
End If
```

C#

在C#中，在接口之间强制转换的最好办法就是使用as操作符。使用as操作符是一个比直接强制转换分配更好的编码策略，因为它在转换失败时产生一个空值而不是引发一个异常。

下面代码第一行是一个直接强制转换。如果完全确定被讨论的对象实现两种接口，这样是可接受的；如果对象不实现，用户试图得到句柄

的接口，.NET抛出异常。更安全的模型是使用as操作符，如果对象不能返回期望接口的引用，返回一个空值。

```
IGeometry geometry = point; // Straight cast
IGeometry geometry = point as IGeometry; // As operator
```

下面的例子展示了如何操作返回空接口句柄的可能性。

```
IPoint point = new PointClass();
IGeometry geometry = point;
IGeometry geometry = point as IGeometry;
if (geometry != null)
{
    Console.WriteLine(geometry.GeometryType.ToString());
}
```

二进制兼容性

大多数现在的ArcGIS Visual Basic 6开发人员都熟悉二进制兼容性的概念。Visual Basic中这个编译标记确保组件每一次被编译时保持相同的GUID。当不设置该标记时，每一次工程被编译时，为每个类产生一个新的GUID。这样有一个负面效应，必须在合适的组件类目中重新注册组件。

为防止在.NET中出现同样的问题，可以使用GUIDAttribute类手工为类指定一个GUID。明确指定一个GUID保证它永远不会改变。如果不指定一个GUID，当用户第一次导出组件到COM中时，类型库导出者会自动产生一个，并且虽然导出者的意思是在后面的导出中都使用相同的GUID，但不能保证如此。

下面例子展示了GUID属性被应用到一个类中。

[VB.NET]

```
<GuidAttribute("9ED54F84-A89D-4fcd-A854-44251E925F09")>
Public Class SampleClass
'
End Class
```

[C#]

```
[GuidAttribute("9ED54F84-A89D-4fcd-A854-44251E925F09")]
Public class SampleClass
{
//
}
```

事件

事件是对象发送的一个通知某个行为发生的消息。行为的发生可以被用户的交互引发，如单击鼠标或被一些其它的程序逻辑触发。引发（triggers）事件的对象叫做事件发送者。捕获事件并响应事件的对象叫做事件接收者。

在事件通信中，事件发送者类不知道哪个对象或方法会接收（处理）

它引发的事件。需要在事件源和接收者之间有一个中间件（或类似指针的机制）。.NET框架定义了一种专门的类型(<Delegate>)提供函数指针的功能。Delegate是一个持有某个方法的引用的类。与其它的类不同，delegate类有一个签名，并且它只持有与签名匹配的方法的引用。这样，delegate相当于一个类型安全函数指针或一个回调函数。

为了在一个应用程序中使用一个事件，必须提供一个事件句柄（事件处理方法），响应事件执行程序逻辑并注册事件源的事件句柄。事件句柄必须有一个和事件delegate一样的签名。此过程被称作事件配线。

下面摘录的ArcObjects代码展示了一个自定义ArcMap命令对*Map*对象选中项改变的事件的配线。为了简单，该事件在*OnClick*事件中配线。

[VB.NET]

```
    'Can't use WithEvents because the outbound interface is not the
    'default interface

    'IActiveViewEvents is the sink event interface.
    'SelectionChanged is the name of the event.
    'IActiveViewEvents_SelectionChangedEventHandler is the delegate
    name.

    'Declare the delegate.
    Private SelectionChanged As
    IActiveViewEvents_SelectionChangedEventHandler

    Private m_mxDoc As IMxDocument

    Public Overloads Overrides Sub OnCreate(ByVal hook As Object)
        Dim app As IApplication
        app = hook
        m_mxDoc = app.Document
    End Sub
   Public Overrides Sub OnClick()
        Dim map As Map
        map = m_mxDoc.FocusMap

        'Create an instance of the delegate, add it to SelectionChanged event.
        SelectionChanged = New
    IActiveViewEvents_SelectionChangedEventHandler(AddressOf
    OnSelectionChanged)
        AddHandler map.SelectionChanged, SelectionChanged

End Sub

    'Event handler
    Private Sub OnSelectionChanged()
       MessageBox.Show("Selection Changed")
End Sub
```

```
[C#]
    // IActiveViewEvents is the sink event interface.
    // SelectionChanged is the name of the event.
    // IActiveViewEvents_SelectionChangedEventHandler is the delegate
    name.
    IActiveViewEvents_SelectionChangedEventHandler
    m_selectionChanged;
    private ESRI.ArcGIS.ArcMapUI.IMxDocument m_mxDoc;

    public override void OnCreate(object hook)
    {
        IApplication app = hook as IApplication;
        m_mxDoc = app.Document as IMxDocument;
    }

    public override void OnClick()
    {
        IMap map = m_mxDoc.FocusMap;
        // Create a delegate instance and add it to SelectionChanged event.
        m_selectionChanged = new
        IActiveViewEvents_SelectionChangedEventHandler(SelectionChang
        ed);
        ((IActiveViewEvents_Event)map).SelectionChanged +=
        m_selectionChanged;
    }
    // Event handler
    private void SelectionChanged()
    {
        MessageBox.Show("Selection changed");
    }
```

错误处理

在Visual Studio .NET中出错处理的构造被称作结构化的异常处理。这个构造可能对于Visual Basic用户比较新，但对于C++或Java用户比较熟悉。

结构化异常处理的实现很简单，并且同样的概念也适用于VB .NET 或C#。VB .NET通过常见的*On Error GoTo*语句和*Err*对象提供了非结构化的异常处理，也允许向下兼容，尽管该模型在这部分不会讨论到。

异常

在Visual Studio .NET中异常是用来处理出错状态的。它提供了出错状态的信息。

异常是一个继承System.Exception基础类的类的实例。.NET框架提供了许多种不同类型的异常类，并且可以创建用户自己的异常类。每种类型通过允许更多地访问关于所发生的特殊类型的出错信息，扩展了System.Exception类的基本功能。

当.NET框架遇到错误情况时，创建并引发一个Exception实例。可以通过使用Try、Catch、Finally构造来处理异常。

Try、Catch、Finally

这个构造允许用户在代码中捕获抛出的错误。下面是这个概念的一个例子。试图旋转一个外包矩形，抛出了一个错误。

[VB.NET]

```
Dim env As IEnvelope = New EnvelopeClass()
env.PutCoords(0D, 0D, 10D, 10D)
Dim trans As ITransform2D = env
trans.Rotate(env.LowerLeft, 1D)
Catch ex As System.Exception
    MessageBox.Show("Error: " + ex.Message)

    ' Perform any tidy up code.
End Try
```

[C#]

```
{
    IEnvelope env = new EnvelopeClass();
    env.PutCoords(0D, 0D, 10D, 10D);
    ITransform2D trans = (ITransform2D) env;
    trans.Rotate(env.LowerLeft, 1D);
}
catch (System.Exception ex)
{
    MessageBox.Show("Error: " + ex.Message);
}

{
    // Perform any tidy up code.
}
```

在可能出错的代码周围设置一个Try块。如果应用程序在Try块里抛出了一个错误，执行点会切换到第一个Catch块中。

Catch块处理一个抛出的错误。当引发错误的Type与Catch块所指定的错误Type匹配时，应用程序执行Catch块。用户可以有不止一种的Catch块处理不同种类的错误。下面所示代码首先检验是否抛出的异常是一个*DivideByZeroException*。

[VB.NET]

```
...
Catch divEx As DivideByZeroException
    ' Perform divide by zero error handling.
Catch ex As System.Exception
    ' Perform general error handling.
...
```

[C#]

```
...
catch (DivideByZeroException divEx)
```

```
{
    // Perform divide by zero error handling.
  }
  catch (System.Exception ex)
  {
  // Perform general error handling.
  }
...
```

如果有不止一个 Catch块，注意越特殊的异常，Types，应该在常见的System.Exception之前，这样会始终处在类型检验之后。

如果错误抛出，应用程序始终会执行Finally块，或者在Try块完成后，或者在Catch块之后。因此，Finally块通常应该包含必须被执行的代码，例如整理资源，例如文件句柄，或数据库关联。

如果没有任何的整理代码，不需要包含一个Finally块。

无异常处理的代码

如果没有包含在Try块中的一行代码抛出错误，.NET运行时在调用函数中搜索Catch块，在调用堆栈一直向上直到发现一个Catch块。

如果在整个调用堆栈中没有Catch块，那么确切的结果就取决于被执行的代码的位置和.NET运行时的构造。因此，建议一个程序的所有进入点至少包括一个Try、 Catch、Finally构造。

COM组件错误

结构化异常处理模型和COM使用的HRESULT模型不同。C++开发人员容易在HRESULT中忽略一个错误状态；在Visual Basic 6中，在HRESULT中的错误状态给*Err*对象赋值并引发一个错误。

.NET运行时对COM组件的出错处理在某种程度上类似于在VB6中处理COM错误的方式。如果.NET程序在COM组件（通过COM interop服务）上调用一个函数并以HRESULT返回一个错误情形，这个HRESULT用来为*COMException*类的例程赋值。这个错误由.NET运行时抛出错误，可以用通常的方法处理，如通过使用Try、Catch、Finally块。

因此，建议将所有在COM组件中将所有可能引发错误的代码装入一个Try块中，并有相应的Catch块来捕获*COMException*。下面是改写的第一个例子，从COM组件上检查错误。

[VB.NET]

```
    Dim env As IEnvelope = New EnvelopeClass()
    env.PutCoords(0D, 0D, 10D, 10D)
    Dim trans As ITransform2D = env
    trans.Rotate(env.LowerLeft, 1D)
```

```
  Catch COMex As COMException
    If (COMex.ErrorCode = -2147220984) Then
       MessageBox.Show("You cannot rotate an Envelope")

       MessageBox.Show _
       ("Error " + COMex.ErrorCode.ToString() + ": " + COMex.Message)
    End If
Catch ex As System.Exception
   MessageBox.Show("Error: " + ex.Message)
...
[C#]
{
    IEnvelope env = new EnvelopeClass();
    env.PutCoords(0D, 0D, 10D, 10D);
    ITransform2D trans = (ITransform2D) env;
    trans.Rotate(env.LowerLeft, 1D);
}
catch (COMException COMex)
{
    if (COMex.ErrorCode == -2147220984)
    MessageBox.Show("You cannot rotate an Envelope");
    MessageBox.Show ("Error " + COMex.ErrorCode.ToString() + ": " +
    COMex.Message);
}
catch (System.Exception ex)
{
   MessageBox.Show("Error: " + ex.Message);
}
...
```

*COMException类*属于System.Runtime.InteropServices命名空间。通过*ErrorCode*属性提供了对初始HRESULT的访问，可以通过测试这个值来查找出哪一种错误情形发生。

抛出错误和异常的等级

如果正在为一个用户接口编写代码，用户可能想要试图在代码中改正错误状态并再次尝试调用。或者可能想要报告错误给用户，让他们决定应该采取什么行动；这里用户可以利用Exception类的Message属性来识别问题。

然而，如果正在编写一个仅从其它代码可以调用的函数，希望这样处理这些错误，创建一个专门的错误状态并将这个错误传送给调用者。可以使用Throw关键字完成这个操作。

为了使现存的错误到达调用者函数，像下面这样，使用Throw关键字编写出错处理程序。

```
[VB.NET]
Catch ex As System.Exception
```

```
...

[C#]
catch (System.Exception ex)
{
   throw;
}
...
```

如果希望传送一个不同的或更特殊的错误给调用者，应该创建一个Exception类的新例程，正确地赋值，并将这个异常返回给调用者。下面的例子使用*ApplicationException*构造函数设置Message属性。

```
[VB.NET]
Catch ex As System.Exception
   Throw New ApplicationException _
      ("You had an error in your application")
...
[C#]
catch (System.Exception ex)
{
   throw new ApplicationException("You had an error in your application");
}
...
```

然而如果这样做，起初的异常丢失。为了使全部的错误信息被传送，Exception 类包括 InnerException 属性。这个属性在新的异常被抛出之前，应该被设置为与被捕获的异常相等。这样创建了一个错误等级。下面这个例子展示了使用 ApplicationException 构造函数来设置 InnerException 和 Message 属性。

```
[VB.NET]
Catch ex As System.Exception
   Dim appEx As System.ApplicationException = _
      New ApplicationException("You had an error in your application",
      ex)
      Throw appEx
...
[C#]
catch (System.Exception ex)
{
   System.ApplicationException appEx =
   new ApplicationException("You had an error in your application",
   ex);throw appEx;
}
...
```

这样，最终处理出错情形的函数可以访问关于错误情形和其前后关系原因的所有信息。

如果抛出错误，应用程序在控制被返回到调用函数之前会执行当前函数的 Finally 子句。

操作资源

直接使用字符串和嵌入图形（没有本地化）

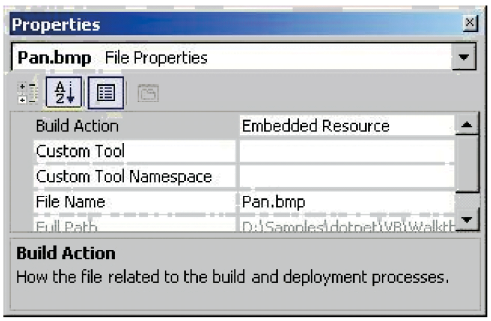
如果当前定制不支持本地化，并且也不打算在以后支持本地化，可以直接使用字符串和图形而不需要资源文件。例如可以在代码中直接指定和使用字符串：

```
[VB.NET]
Me.TextBox1.Text = "My String"

[C#]
this.textBox1.Text = "My String";
```

图象文件（BMPs、JPEGs、PNGs等）按照下面的步骤直接嵌入到部件中：

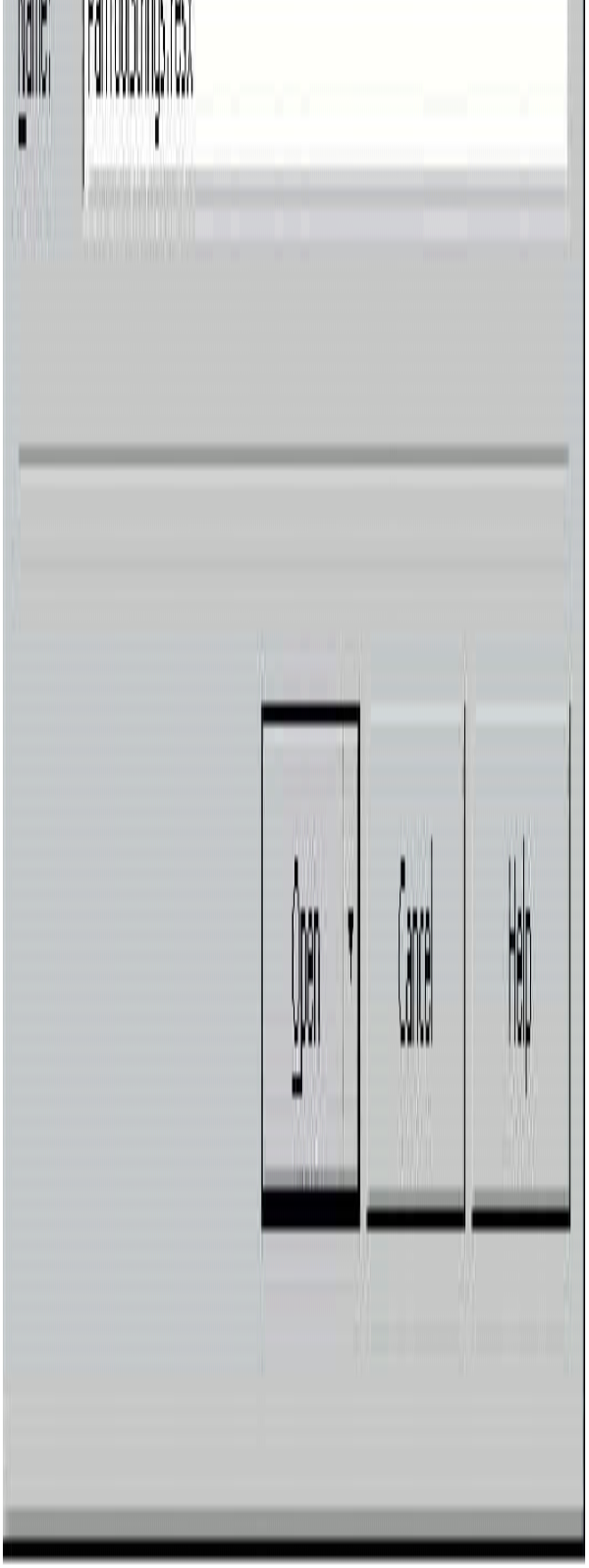
1. 在 Solution Explorer 中右键单击 Project，单击 Add，然后单击 Add Existing Item。
2. 在 Add Existing Item 对话框中，浏览图象文件并单击 Open。
3. 在 Solution Explorer 中，选择刚添加的图形文件，然后按下 F4 显示属性。
4. 设置 Build Action 属性到 Embedded Resource。



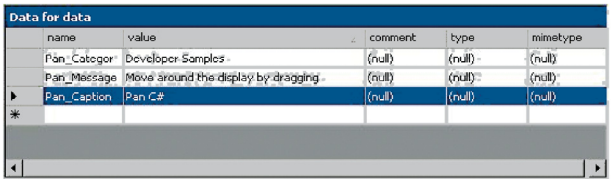
现在可以在代码中引用图象。例如，下面的代码从部件中第一个嵌入的资源创建一个位图对象。

```
[VB.NET]
Dim res() As String =
GetType(Form1).Assembly.GetManifestResourceNames()
If (res.GetLength(0) > 0)
    Dim bmp As System.Drawing.Bitmap = New
    System.Drawing.Bitmap( _
       GetType(Form1).Assembly.GetManifestResourceStream(res(0)))
    ...

[C#]
string[] res = GetType().Assembly.GetManifestResourceNames();
if (res.GetLength(0) > 0)
{
```



3. 打开Visual Studio中的新.resx文件，并添加名字—value pairs for the culture—应用程序中特定的字符串。



4. 当编译工程时，.resx 文件会编译成为嵌在主部件的.resources模块。

在Microsoft .NET Framework文档中可找到有助于操作资源的一列工具，其网址为http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptutorials/html/appendix_b__resource_tools.asp。

在Microsoft .NET Framework文档中可找到关于ResEditor例子的附加信息。其网址为http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptutorials/html/resource_editor__reseditor_.asp

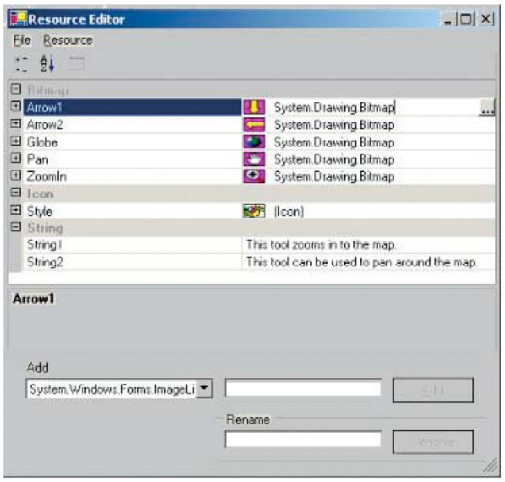
ResEditor例子是由Microsoft作为源代码提供的。如果想要用这个工具创建资源文件，必须先创建实例。关于创建SDK实例的信息可以在Visual Studio .NET安装目录的SDK子目录下找到。

为图象资源创建资源文件

在.NET中添加图象、图标或光标到一个resources文件中的过程比创建一个仅包含字符串值的文件要复杂，因为当前在Visual Studio .NET IDE可获得的工具只能用于添加字符串资源。

然而，使用Visual Studio .NET Framework SDK可获得大量例子工程，这些例子会有助于操作资源文件。一个这样的例子就是Resource Editor （ResEditor）。

ResEditor实例可用来添加图形、图标、图象列表框和字符串到一个资源文件。这个工具不能用来添加光标资源。文件可以被保存为.resx或者.resource文件。



通过编程创建资源文件

可以通过使用*ResXResourceWriter*类（.NET框架的一部分）编程创建包含资源的XML.resx文件。可以通过使用*ResourceWriter*类（也是.NET框架的一部分）编程创建二进制.resources文件。这些类允许更灵活地添加所需的资源种类。

如果要添加不能通过.NET Framework SDK实例和工具操作的资源（例如光标），这些类非常有用。这两个类基本用法是类似的：首先，创建一个新的资源写字板类指定文件名，然后使用*AddResource*方法逐个地地添加资源。

下面的代码示范了如何使用类创建一个新的.resx 文件并添加一个位图和光标到文件中。

```
[VB.NET]
Dim img As System.Drawing.Image = CType(New
System.Drawing.Bitmap("ABitmap.bmp"), System.Drawing.Image)
Dim cur As New System.Windows.Forms.Cursor("Pencil.cur")

Dim rsxw As New System.Resources.ResXResourceWriter("en-AU.resx")
rsxw.AddResource("MyBmp_jpg", img)
rsxw.AddResource("Mycursor_cur", cur)
rsxw.Close()
[C#]
System.Drawing.Image img = (System.Drawing.Bitmap) new
System.Drawing.Bitmap("ABitmap.bmp");
System.Windows.Forms.Cursor cur = new
System.Windows.Forms.Cursor("Pencil.cur");

System.Resources.ResXResourceWriter rsxw = new
System.Resources.ResXResourceWriter("en-GB.resx");
rsxw.AddResource("MyBmp_jpg", img);
rsxw.AddResource("Mycursor_cur", cur);
rsxw.Close();
```

PanTool开发人员实例（Samples\Map Analysis\Tools）包含一个脚本—MakeResources—展示了怎样使用*ResXResourceWriter*类，将位图、光标文件和字符串写入一个.resx文件。该实例也展示了如何使用*ResXResourceReader*类读取一个.resx文件。这个例子包括一个含有一个位图、两个光标和三个字符串的文件。

在http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptutorials/html/ResX_Generator__RESXGEN_.asp上可找到关于ResXGen的更多信息。

编译.resx文件到.resources文件中

基于XML的.resx文件可以使用指南中Visual Studio IDE或ResX Generator (ResXGen)实例被编译为二进制.resources文件。

- 当工程创建之后，Visual Studio工程中的任何一个.resx文件都会被编译成为一个.resources模块。关于如何使用多个资源文件本地化，见下面“使用本地化资源”部分。
- 可以使用.NET Framework SDK命令resgen,独立于创建过程转换一个.resx文件为.resources文件，例如：

  resgen PanToolCS.resx PanToolCS.resources

使用本地化资源

下面这部分说明如何为用户定制本地化资源。

如何使用本地化资源

在.NET中，一个特定的语言和国家/地区联合在一起被叫做*culture*。例如，通过字符串“en-US”表示英语的美国方言，通过使用“fr-CH”表示法语的瑞士方言。

如果想要工程支持不同的文化（语言和方言），应该为每种文化构建一个包含特定文化字符串和图象的独立.resources文件。

创建使用资源的.NET工程时，在主部件中嵌入了缺省的.resources文件。特定文化的.resources文件编译为satellite部件（使用命名规则<Main Assembly Name>.resources.dll）并被放置在主构建目录的子目录下。这个子目录以它们所包含的satellite部件的文化命名。例如，Swiss - French资源被包含在fr-CH子目录中。

Pan Tool开发人员实例的Visual Basic .NET和C#风味举例说明了如何为German语言环境本地化资源。该实例可以在Developer Samples\ArcMap\Commands and Tools\Pan Tool目录中找到。严格地说，该实例仅需要本地化的字符串，但是图象也被转变为“de”文化，作为例子用。

一个应用程序运行时，自动地使用包含在适当文化的satellite部件中的资源。适当的文化由Windows设置决定。如果适当文化的satellite部件没有被找到，会使用缺省的资源（嵌入到主部件中）代替。

下面这部分给出了关于创建用户自己的.resx和.resources文件更多的信息。

在工程中嵌入一个缺省的.Resources文件

Pan Tool实例中提供了一个叫做buildResources.bat的批处理文件，用来创建缺省的.resources文件和特定文化的satellite部件。

1. 在Solution Explorer中右键单击Project name，单击Add，然后单击Add Existing Item浏览至用户的.resx或.resources文件。
2. 在Solution Explorer中，选择要添加的文件并单击F4显示属性。
3. 设置Embedded Resource的Build Action属性。

这样如果对于应用程序运行的文化没有资源DLL，确保用户的应用程序始终有一套资源可用。

为工程所支持的文化创建.Resources.dll文件

1. 首先在工程中有一个缺省的.resx或.resources文件。
2. 获得.resx或.resources文件并为每个要支持的文化创建一个独立的本地化文件。
   - 每个文件都应该包含使用相同Names的资源；文件中的每个资源Value都应该包含本地值。

- 本地化的资源文件应该根据它们的文化命名，例如，<BaseName>.<Culture>.resx 或 <BaseName>.<Culture>.resources。

3. 添加新的资源文件到工程中，确保每个资源文件都有设置到Embedded Resource的Build Action。
4. 创建工程。
   编译器和链接器会为每一个文化创建独立的satellite部件。satellite部件被放置包含的主部件目录的子目录下。子目录按照文化命名，允许.NET运行时为应用程序运行的文化查找合适资源。主要的（缺省的）资源文件内嵌在主部件中。

部件版本化和重定向

使用经过强硬命名的特定版本部件建立起来的应用程序，运行时要求相同的部件。例如，如果使用ESRI.ArcGIS.System 9.0.452版本创建一个应用程序，不能在安装了新版本ESRI.ArcGIS.System（例如9.0.0.692）的系统上运行这个应用程序。如果安装了更新版本的ArcGIS，可能会出现这种情况；然而使用配置文件,用户可以重定向一个应用程序来使用一个部件的新版本。
应用程序部件重定向可以有两种选择：

- 应用程序配置文件
- 机器配置文件

应用程序配置文件

应用程序配置文件包含有针对于一个应用程序的设置。该文件包含了通用语言运行时读取的配置设置（例如部件绑定原则或远程对象）和应用程序可读取的设置。

应用程序配置文件的名称和位置取决于应用程序的主机，可以是下面的一种：

- Execuable-hosted 应用程序—可执行主机拥有的应用程序配置文件在应与应用程序相同的目录下。配置文件的名称是应用程序名称加上一个扩展名。例如，一个叫做 myApp.exe 的应用程序可能与一个叫做 myApp.exe.config 的配置文件相关联。

- ASP.NET-hosted应用程序—ASP.NET配置文件在ASP.NET应用程序中被称作Web.Config.Configuration文件，继承了配置文件在URL路径中的设置。例如，给定URL www.esri.com/aaa/bbb，www.esri.com/aaa是Web应用程序，与应用程序想关联的配置文件位于www.esri.com/aaa.。在子目录/bbb中的ASP.NET页使用在应用程序级别的配置文件设置和在/bbb中的配置文件设置。

- Internet Explorer-hosted应用程序—如果Internet Explorer上的应用程序有一个配置文件，这个文件的位置使用下面的语句在一个<link>标签中指定：

  <link rel="ConfigurationFileName" href="location">

  在这个标签中，location是一个配置文件的URL。这样设置了应用程序的基础。配置文件必须位于与应用程序相同的Web站点。

机器配置文件

机器配置文件，Machine.config,，包含应用于整个计算机的设置。这个文件位于%runtime install path%\Config目录。Machine.config包含机器级别的部件绑定，内置远程通道和的配置环境和ASP.NET。

配置系统首先在机器配置文件中查找<appSettings>元素和其它开发人员可能会定义的配置部分。然后查看应用程序配置文件。为了使机器配置文件容易管理，最好在应用程序配置文件中进行设置。然而在机器配置文件中设置可以使得系统更加可维护。例如，有一个客户和服务器应用程序都要使用的第三方组件，在一个位置为该组件设置是比较容易的。在这种情况子下，机器配置文件是设置的合适位置，因此在不同的文件中不会有相同的设置。

使用XCOPY复制应用程序不会复制机器配置文件的设置。

下面设置文件展示了如何绑定到一个部件上并使之适应新的版本。

```
<configuration>
   <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
         <dependentAssembly>
            <assemblyIdentity name="ESRI.ArcGIS.System"
publicKeyToken=”8fc3cc631e44ad86"
                                    culture="neutral" />
  <!— Assembly versions can be redirected in application,
publisher policy, or machine configuration files. —>
                        <bindingRedirect oldVersion="9.0.0.452"
newVersion="9.0.0.692"/>
            </dependentAssembly>
         </assemblyBinding>
      </runtime>
   </configuration>
```

使用.NET开发ARCGIS

使用.NET，可以定制ArcGIS应用程序，创建使用ESRI类型的独立应用程序和扩展ESRI的类型。例如，可以为ArcMap创建一个自定义工具，创建一个使用MapControl的独立应用程序或创建一个自定义层。这部分讨论了关于使用ArcGIS和.NET和开发的几个关键问题。

用COM注册.NET组件

使用自定义.NET组件扩展ArcGIS应用程序需要在COM注册表中注册组件并导出.NET部件到一个类型库(TLB)中。当开发一个组件时，有两种方法可以执行这个任务：使用装载.NET Framework SDK的RegAsm应用程序或有一个针对COM Interop的Register编译器标签的Visual Studio.NET。

下面的例子展示了一个使用COM注册的EditTools部件。/tlb参数表明应该生成一个类型库，/codebase选项表明部件的路径应该包含在注册表设置中。当使用.NET组件扩展ArcGIS应用程序时，这两个参数都需要。

```
regasm EditTools.dll /tlb:EditTools.tlb /codebase
```

如果为COM Interop编译器标签设置了Register，Visual Studio.NET会自动执行相同的操作；这是在一个开发机器上执行注册的最简单方法。要检查工程的设置，在Project菜单中单击Project Properties，然后在Configuration Properties下查看Build属性。最后一项，Register for COM Interop，应该设置为True。

在COM组件类目内注册.NET类

许多ArcGIS的扩展性依赖于COM组件类目。事实上，大多数自定义的ArcGIS组件必须在适合语境的组件类目内和主机应用程序要使用到功能的函数内注册。例如，所有的ArcMap命令和工具必须在*ESRI Mx Commands*组件类目内注册。在一个特定的类目内注册一个.NET组件可以有不同的方法，但是在注册之前，必须使用COM注册.NET组件。更加详细的内容参见上面“使用COM注册.NET组件”的部分。

自定义对话框

自定义的.NET ArcGIS命令和工具可以通过Customize对话框中的Add From File按钮快速地添加到工具条中。在这种情况下，只需要简单地浏览一下TLB并打开它。ArcGIS框架会自动地将从类型库中选中的类添加到合适的组件类目中。

类目实用工具

另外一个选项是使用Component Categories Manager（Categories.exe）。在这种情况下，可以在应用程序中选择组件类目，浏览类型库，并选择合适的类。

COM注册函数

最后的也是推荐的解决方案是添加代码到.NET类中，这样每当使用COM注册组件的时候，会在特定的组件类目中自动注册他们。.NET Framework包含两个属性类（*ComRegisterFunctionAttribute*和

ComUnregisterFunctionAttribute），它们允许用户指定方法，不论什么时候注册组件或取消注册组件时都可以调用。两个方法都传递给当前被注册类的CLSID。使用这个消息，可以在方法内编写代码来安排适合的注册录入和删除。在组件类目内注册一个组件也要求用户知道组件类目的唯一ID（CATID）。

下面摘录的代码展示了一个自定义ArcMap命令，当它所在的.NET部件使用COM注册时，该命令自动在*MxCommands*组件类目中注册自己。

```
public sealed class AngleAngleTool: BaseTool
{
   [ComRegisterFunction()]
   static void Reg(String regKey)
   {
      Microsoft.Win32.Registry.ClassesRoot.CreateSubKey(regKey.
Substring(18)+ "\\Implemented Categories\\" +
"{B56A7C42-83D4-11D2-A2E9-
080009B6F22B}");
      }
   [ComUnregisterFunction()]
   static void Unreg(String regKey)
   {
   Microsoft.Win32.Registry.ClassesRoot.DeleteSubKey(regKey.Substrin
   g(18)+
"\\Implemented Categories\\"
+"{B56A7C42-83D4-11D2-A2E9-080009B6F22B}");
   }
```

为简化此过程，ESRI为ArcGIS展示的每个组件类目的类提供了静态函数来注册组件和取消注册组件。每个类都知道组件类目GUID所表示的组件类目，所以注册自定义组件大大地简化了。关于使用这些类更多的细节，见下面“操作ESRI .NET组件类目类”部分。

使用ESRI.ArcGIS.Utility部件简化编码过程

一部分ArcGIS Developer Kit包括了大量的.NET应用程序类，这些程序利用一些.NET功能（包括对象继承和静态函数），方便了.NET开发。

操作ESRI .NET基类

ESRI提供了两个抽象基类（BaseCommand and BaseTool），有助于为ArcGIS创建新的自定义命令和工具。这些类是抽象类（在Visual Basic .NET中标记为MustInherit），这意味着虽然这些类包含一些实现的代码，它自身不能被直接被实例化并只能通过被另外的类继承来使用。这两个基础类都是在ESRI.ArcGIS.Utility部件中定义的，并且都属于ESRI.ArcGIS.Utility.BaseClasses命名空间。

这些基类为每个ICommand和ITool每个成员提供了缺省的实现，这样简化了创建自定义命令和工具的过程。不用除去每个成员并提供实现代码，只需重载自定义命令或工具所需要的成员。异常是ICommand::OnCreate；该成员在初始的类中必须被重载。

在.NET语言中为ArcGIS应用程序创建命令和工具，推荐使用这些基类。可以从第一法则创建类似的COM类；然而用户很快就会发现基类技术是一个更快速、更简单、更少出错的创建命令和工具的方法。

句法

两种基础类都有重载的构造函数，使得用户可以快速地通过构造函数参数设置命令和工具的许多属性，例如Name和Category。

重载的*BaseCommand*构造函数有下面的签名：

```
[VB.NET]
Public Sub New( _
   ByVal bitmap As System.Drawing.Bitmap _
   ByVal caption As String _
   ByVal category As String _
   ByVal helpContextId As Integer _
   ByVal helpFile As String _
   ByVal message As String _
   ByVal name As String _
   ByVal tooltip As String)
[C#]
  public BaseCommand(
      System.Drawing.Bitmap bitmap,
      string caption,
      string category,
      int helpContextId,
      string helpFile,
      string message,
      string name,
      string toolTip,
);
```

重载的BaseTool构造函数有下面的签名

```
[VB.NET]
Public Sub New( _
ByVal bitmap As System.Drawing.Bitmap _
   ByVal caption As String _
   ByVal category As String _
   ByVal cursor As System.Windows.Forms.Cursor _
   ByVal helpContextId As Integer _
   ByVal helpFile As String _
   ByVal message As String _
   ByVal name As String _
   ByVal tooltip As String _
)

[C#]
public BaseTool(
   System.Drawing.Bitmap bitmap,
   string caption,
   string category,
```

```
    System.Windows.Forms.Cursor cursor,
    int helpContextId,
    string helpFile,
    string message,
    string name,
    string toolTip,
);
```

继承基类

当编写一个新类时，使用这些参数化的构造函数，例如，下面显示了称作PanTool的新类继承了BaseTool类。

```
[VB.NET]
Public Sub New()
    MyBase.New( Nothing, "Pan", "My Custom Tools", _
        System.Windows.Forms.Cursors.Cross, 0, "", "Pans the map.",
"PanTool", "Pan")
End Sub

[C#]
public PanTool() : base ( null,"Pan", "My Custom Tools",
   System.Windows.Forms.Cursors.Cross, 0, "","Pans the map.",
"PanTool",
"Pan")
{
...
}
```

直接设置基类成员

作为使用参数化构造函数的备选方案，可以直接设置基类的成员。

基础类暴露他们内部成员变量给继承类，每个属性一个，这样可以在起初的类上直接访问它们。例如，代替使用构造函数来设置Caption或重载Caption函数，可以在设置在基类中声明的m_caption类成员变量。

```
[VB.NET]
Public Sub New()
    MyBase.New()
    MyBase..m_bitmap = New
System.Drawing.Bitmap([GetType]().Assembly.GetManifestResourceStream("Name
space.Pan.bmp"))
    MyBase..m_cursor = System.Windows.Forms.Cursors.Cross
    MyBase..m_category = "My Custom Tools"
    MyBase..m_caption = "Pan"
    MyBase..m_message = "Pans the map."
    MyBase..m_name = "PanTool"
    MyBase..m_toolTip = "Pan"
End Sub
```

```
[C#]
public PanTool()
{
   base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream("Names
pace.Pan.bmp"));
    base.m_cursor = System.Windows.Forms.Cursors.Cross;
    base.m_category = "My Custom Tools";
    base.m_caption = "Pan";
    base.m_message = "Pans the map.";
    base.m_name = "PanTool";
    base.m_toolTip = "Pan";
}
```

重载成员

当创建继承一个基类的自定义命令和工具时，可能不仅需要重载几个成员。当重载类中的一个成员时，会执行用户提供给该成员实现的代码，而不会执行从基类继承而来的缺省实现代码。例如，在*BaseCommand*中*OnClick*方法没有任何的实现代码，同样的在缺省情况下*OnClick*不会做任何事。这样对于一个工具来说是合适的，但是可能对于一个命令来说不适合。

要重载任何成员，可以在Solution Explorer Window中右键单击基类成员，单击Add，然后单击Override将该成员作为重载存根。注意如果右键单击底层的接口（*ICommand*或*ITool*）成员而不是基础类成员，重载的成员不会包含在重载关键字中，并且方法也会是代替为有阴影的。

```
[VB.NET]
Public Overrides Sub OnClick()
      ' Your OnClick
End Sub

[C#]
public override void OnClick()
{
   // Your OnClick
}
```

重载基类的成员，在Code Window Wizard条上右侧的下拉列表中单击Overrides，然后从左侧下拉列表中选择想要重载的成员。这样会存根成员为重载的。

缺省状态下基类会做什么

下面的表展示了具有明显基类实现的基类成员和该实现的描述。当基类行为与自定义不一致时，重载这些成员。例如，在缺省状态下Enabled设置为TRUE；如果要自定义命令只有当符合一套特定的准则时才激活，必须在初始类中重载这个属性。

Member	Description
ICommand::Bitmap	The given bitmap is made transparent based on the pixel value at position 1,1. The bitmap is null until set by the derived class.
ICommand::Category	If null, sets the category "Misc."
ICommand::Checked	Set to False.
ICommand::Enabled	Set to True.
ITool::OnContextMenu	Set to False.
ITool::Deactivate	Set to True.

操作ESRI.NET组件类目类

为了帮助在COM组件类目中注册.NET组件，ESRI提供了命名空间ESRI.ArcGIS.Utility.CATIDs，它包含表示每个ArcGIS组件类目的类。每个类都知道自己的CATID并暴露给静态方法（Register and Unregister）来添加和移除组件。注册组件像使用合适的属性添加COM注册方法和传递接受到的CLSID至适合的静态方法一样容易。

下面的例子展示了一个自定义Pan工具在ESRI Mx Commands组件类目中注册自己。注意到在这个例子中使用MxCommands.Register和MxCommands.Unregister 代替 Microsoft.Win32.Registry.ClassesRoot.CreateSubKey和Microsoft.Win32.Registry.ClassesRoot.DeleteSubKey。

[VB.NET]

```
Public NotInheritable Class PanTool
    Inherits BaseTool

    <ComRegisterFunction()> _
    Public Shared Sub Reg(ByVal regKey As [String])
        MxCommands.Register(regKey)
    End

    <ComUnregisterFunction()> _
    Public Shared Sub Unreg(ByVal regKey As [String])
        MxCommands.Unregister(regKey)
    End Sub
```

[C#]

```
public sealed class PanTool : BaseTool
{
    [ComRegisterFunction()]
    static void Reg(string regKey)
    {
        MxCommands.Register(regKey);

}
```

```
    [ComUnregisterFunction()]
    static void Unreg(string regKey)
    {
        MxCommands.Unregister(regKey);
    }
}
```

扩展服务器

当使用.NET创建在GIS服务器上使用的COM对象时，需要遵循一些指导方针以确保可以在一个服务器环境下使用该对象并且在那样的环境下会很好地执行。下面指导方针专门应用于创建在服务器内运行的COM对象。

- 必须显式创建一个COM类实现的接口。与Visual Basic 6不一样，.NET不会为当在一个服务器环境下创建对象时要使用的COM类创建一个隐式的接口。
- 应该使用Automation列集器为用户的COM类列集。可通过添加*AutomationProxyAttribute*到带有真值的类中来指定这些。
- COM类应该生成一个双重的类接口。可以通过添加*ClassInterfaceAttribut*到带有ClassInterfaceType.AutoDual值的类中来指定。
- 为确保用户的COM对象在服务器上很好地运行，该对象必须从*ServicedComponent*继承，该组件位于System.EnterpriseServices部件中。由于.NET Framework的当前COM interop的实现，这是必须的。

更多的细节和用.NET编写的自定义Server COM对象的例子，参见*ArcGIS Server Administrator and Developer Guide*第四章‘Developing ArcGIS Server applications’。

释放COM引用

ArcGIS Engine和ArcGIS Desktop应用程序

当一个独立的应用程序试图关闭时，意外的崩溃可能会发生。例如，一个加载了地图文档的带有MapControl的应用程序在退出时会崩溃。崩溃的原因是COM对象挂起的时间比预期的要长。为避免崩溃，所有的COM引用在程序关闭之前必须卸载。为了帮助卸载COM引用，添加一个专门的静态Shutdown函数到ESRI.ArcGIS.Utility部件中。下面摘录的代码展示了正在使用的函数。

[VB.NET]

```
Private Sub Form1_Closing(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
```

```
        ESRI.ArcGIS.Utility.COMSupport.AOUninitialize.Shutdown()
End Sub

 [C#]
private void Form1_Closing(object sender, CancelEventArgs e)
{
        ESRI.ArcGIS.Utility.COMSupport.AOUninitialize.Shutdown();
}
```

在独立的应用程序中，使用*AOUninitialize.Shutdown*函数可以处理大多数关闭问题，但仍然会遇到要求显式释放COM对象的问题；该情况下调用*System.Runtime.InteropServices.Marshal.ReleaseComObject()*来减少引用数，允许应用程序正常地终止。StyleGallery就是一个这样的对象，下面的例子表明了如何处理这个类的引用。

```
[VB.NET]
    Dim styleGallery As IStyleGallery
    styleGallery = New StyleGalleryClass
    MessageBox.Show(styleGallery.ClassCount)
    Marshal.ReleaseComObject(styleGallery)
[C#]
    IStyleGallery sg = new StyleGalleryClass() as IStyleGallery;
    MessageBox.Show(sg.ClassCount.ToString());
    Marshal.ReleaseComObject(sg);
```

在ArcGIS Server中操作光标

某些可以在服务器环境下创建的对象可以锁定或使用仅在对象的析构函数上释放的资源。例如，一个地理数据库临时表可能在一个基于文件的要素类上获得一个共享模式锁，或在一个表上，地理数据库建立在该表上或占有一个SDE流。

虽然共享模式锁在适当的位置，其它的应用程序可以继续查询或更新表中的记录，但是不能删除要素类或修改模式。如果是基于文件的数据源，例如shape文件，更新临时表在该文件上获得一个唯一的写保护锁，这样会防止其它的应用程序访问文件，进行读写操作。锁的效果就是所有的数据不能被其它的应用程序使用，直到临时表对象上所有的引用被释放。

如果是SDE数据源，临时表拥有一个SDE流，并且如果应用程序有多个用户，每个用户获得或拥有一个SED流，最后会消耗完最大允许流。SDE流超过最大数的效果是其它用户打不开自己的临时表来查询数据库。

由于上面的这些原因，确保引用任何应用程序打开的临时表被及时释放，这是很重要的。在.NET中，对临时表（或任何其它的COM对象）的引用直到垃圾集合被删除才被释放。在一个Web应用程序或一个服务多个当前场景和请求的Web服务，依赖于垃圾集合来释放对对象的引用会导致临时表和其它的资源不能被及时地释放。

为确保当COM对象超出范围时被释放，*WebControls*部件包含了一个称为*WebObject*帮助者对象。使用*ManageLifetime*方法添加COM对象到对象集上，当*WebObject*被处理掉时，该对象集会显式释放。必须在一个*Using*块内使用*WebObject*。当限定在一个使用块中使用*WebObject*时，任何使用*ManageLifetime*方法添加到中的*WebObject*对象会在使用块的末尾被显式释放。

下面的实例演示了这种编码模式：

[VB.NET]

```
  Private Sub doSomething_Click(ByVal sender As System.Object, ByVal
e AsSystem.EventArgs) Handles doSomething.Click
      Dim webobj As WebObject = New WebObject
      Dim ctx As IServerContext = Nothing
      Try
         Dim serverConn As ServerConnection = New
         ServerConnection("doug", True)
         Dim som As IServerObjectManager =
         serverConn.ServerObjectManager

         ctx = som.CreateServerContext("Yellowstone", "MapServer")
         Dim mapsrv As IMapServer = ctx.ServerObject
         Dim mapo As IMapServerObjects = mapsrv
         Dim map As IMap = mapo.Map(mapsrv.DefaultMapName)

         Dim flayer As IFeatureLayer = map.Layer(0)
         Dim fClass As IFeatureClass = flayer.FeatureClass

         Dim fcursor As IFeatureCursor = fClass.Search(Nothing, True)
            webobj.ManageLifetime(fcursor)

            Dim f As IFeature = fcursor.NextFeature()
            Do Until f Is Nothing
               ' Do something with the feature.
               f = fcursor.NextFeature()
         Loop

        Finally
         ctx.ReleaseContext()
         webobj.Dispose()
            End Try
         End Sub
```

[C#]

```
   private void doSomthing_Click(object sender, System.EventArgs e)
    {
    using (WebObject webobj = new WebObject())
    {
       ServerConnection serverConn = new ServerConnection("doug",true);
       IServerObjectManager som = serverConn.ServerObjectManager;
```

```
        IServerContext ctx =
som.CreateServerContext("Yellowstone","MapServer");
        IMapServer mapsrv = ctx.ServerObject as IMapServer;
        IMapServerObjects mapo = mapsrv as IMapServerObjects;
        IMap map = mapo.get_Map(mapsrv.DefaultMapName);

        IFeatureLayer flayer = map.get_Layer(0) as IFeatureLayer;
        IFeatureClass fclass = flayer.FeatureClass;

        IFeatureCursor fcursor = fclass.Search(null, true);
        webobj.ManageLifetime(fcursor);

        IFeature f = null;
        while ((f = fcursor.NextFeature()) != null)
        {
                // Do something with the feature.
        }

        ctx.ReleaseContext();
    }
}
```

WebMap、*WebGeocode*和*WebPageLayout*对象也有一个*ManageLifetime*方法。例如，如果正在使用一个*WebMap*方法，并且限定在一个使用块中，可以在使用块的末尾添加*ManageLifetime*来显式释放对象。

部署.NET ArcGIS定制

所有ArcGIS Engine和Desktop定制需要在所有的客户机上安装ArcGIS。ArcGIS安装必须包括ESRI 主要的interop部件，安装程序在全局部件缓存中安装它。例如，部署一个只需要一个ArcGIS Engine的license的独立GIS应用程序，要求在所有的目标机器上安装ArcGIS Engine。

独立应用程序

部署独立应用程序到ArcGIS Engine或Desktop客户端上涉及复制可执行程序到客户机器上。复制可执行程序可能只是简单的使用xcopy或涉及更多一点，如创建一个自定义安装或安装程序。注意到除ArcGIS主要interop部件和.NET Framework部件之外，依赖的所有文件必须也被封装和部署。

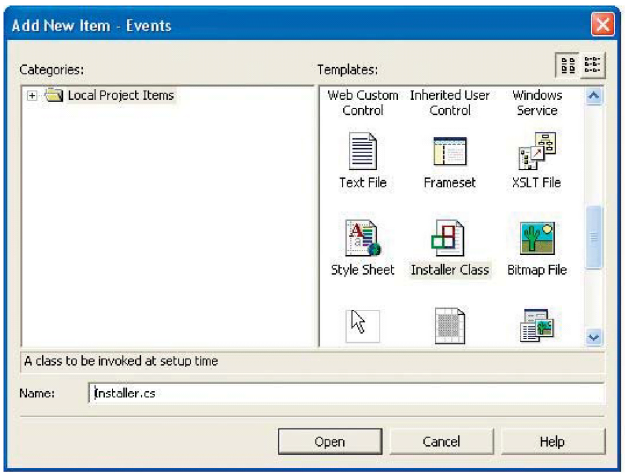
ArcGIS组件

扩展ArcGIS应用程序的组件的部署比独立的应用程序的部署更难处理，因为在特定的组件类目内使用COM注册。像前面讨论过的，实现*COMRegisterFunction*和*COMUnregisterFunctions*通过提供自己的类目注册方便了部署，但是，只有当组件被注册时才会这样发生。

用COM注册组件有两个技巧。一个选项是运行装载.NET Framework SDK.的注册部件的应用程序(RegAsm.exe)。通常这不是一个可行的解决方案，因为客户机不一定有这个应用程序，并且不容易自动化。第二个也是推荐的方法是添加自动注册步骤到一个自定义安装或安装程序中。创建一个配置和注册组件的自定义安装程序的关键是*System.Runtime.InteropServices.RegistrationServices*类。这个类有*RegisterAssembly*和*UnregisterAssembly*两个成员，这两个成员使用COM注册和取消注册被管理类。这些是RegAsm应用程序使用的相同函数。在一个用户自定义安装类内与安装程序一块使用这些函数是一个完整的解决方案。

下面的基本步骤概括了创建一个可部署解决方案的创建。注意：这些步骤假定用户已经开始了一个解决方案，该方案包含一个至少有一个激活类的工程。

1. 在Visual Studio .NET中，添加一个新的Installer Class并相应地命名。



重载在Installer基类中实现的Install和Uninstall函数并使用RegistrationServices类的*RegisterAssembly*和*UnregisterAssembly*方法注册组件。确保使用SetCodeBase标记；这表明部件的代码基键应该在注册表中设置。

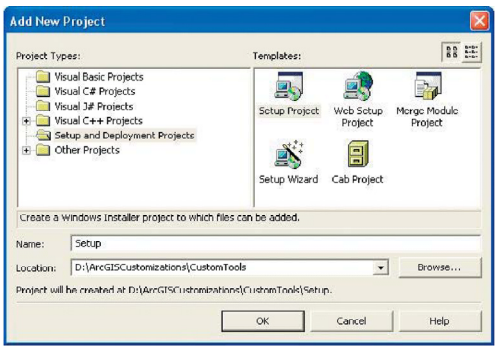
```
[VB.NET]
Public Overrides Sub Install(ByVal stateSaver As
System.Collections.IDictionary)
    MyBase.Install(stateSaver)
    Dim regsrv As New RegistrationServices
    regsrv.RegisterAssembly(MyBase.GetType().Assembly,
AssemblyRegistrationFlags.SetCodeBase)
End Sub
```

```
Public Overrides Sub Uninstall(ByVal savedState As
System.Collections.IDictionary)
    MyBase.Uninstall(savedState)
    Dim regsrv As New RegistrationServices
    regsrv.UnregisterAssembly(MyBase.GetType().Assembly)
End Sub
End Class

[C#]
public override void Install(IDictionary stateSaver)
{
    base.Install (stateSaver);
    RegistrationServices regSrv = new RegistrationServices();
    regSrv.RegisterAssembly(base.GetType().Assembly,
AssemblyRegistrationFlags.SetCodeBase);
}

public override void Uninstall(IDictionary savedState)
{
   base.Uninstall (savedState);
   RegistrationServices regSrv = new RegistrationServices();
    regSrv.UnregisterAssembly(base.GetType().Assembly);
    }
```

2. 添加安装程序到用户的解决方案中。



a. 在Solution Explorer中，右键单击新工程，并单击Add > Project Output。选择要部署的工程并选择Primary output。

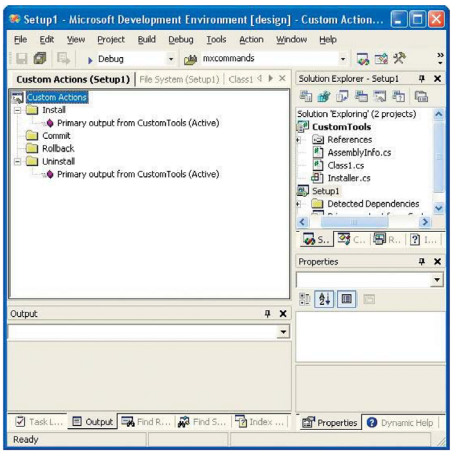
b. 从检测到的生成的依赖关系列表中，移出所有到ESRI主要interop部件（例如，ESRI.ArcGIS.System）和stdole.dll的引用。一般在列表中留下的项目是用户的TLB和从<AssemblyName><Version>的Primary输出，这代表了正在编译的DLL或EXE。

c. 最后的步骤包含将在一个新安装类中部署的自定义安装步骤与安装工程相联合。要做到这一点，在Solution Explorer

中，右键单击安装工程并单击View Custom Actions。

d. 在结果视图中，右键单击Install文件夹并单击Add Custom Action。双击Application文件夹，然后双击Primary output from the <AssemblyName><Version>项。这个步骤将前边创建的自定义安装函数与自定义安装动作相联系。

e. 为安装程序的卸载重复最后一步。



3. 最后，重建整个解决方案生成安装程序可执行文件。在目标机器运行可执行文件安装组件，并用COM注册这些组件。然后*COMRegesiterFunction*程序会在适合的组件类目中注册组件。

ArcGIS服务器的部署

使用内置的Visual Studio.NET工具来部署在一个开发服务器上开发的Web应用程序以产生产品服务器。

1. 在Solution Explorer中，单击工程。
2. 单击Project菜单，然后单击Copy Project。
3. 在Copy Project对话框中，指定部署位置。
4. 单击OK。

除了复制工程，必须复制和注册任何包含自定义COM对象的相关DLLs到用户的Web服务器和所有GIS服务器的服务器对象容器机器。

ArcGIS 9.0 版本的 Java API 是一种专门为 Java 开发者设计的能与 ArcObjects 互操作的程序接口。Java 技术由 Sun MicroSystems 开发，既是一种平台，又是一种面向对象的程序设计语言，它共有三个版本并由两个部分组成：

三个版本为：

- Java 2 平台，标准版（J2SE）
- Java2 平台，企业版（J2EE）
- Java2 平台，精简版（J2ME）

两个组成部分为：

- Java 虚拟机（JVM）—Java 运行时和客户/服务器编译器。
- Java 应用程序接口（API）—核心、集成和用户接口工具包套件。

Java 语言非常重要，因为它基于一个开放的标准。这种编程语言的所有实现都必须遵守 JVM 提供的标准。这使得应用程序可以在任何安装了 JVM 的硬件平台上运行。

平台配置

本节将描述使用 Java API 编程所需的所有必要配置，包括类路径（classpath）和环境设置。

Java 开发工具包

为了用 Java API 进行 ArcObjects 开发，必须安装 Java2 平台标准软件开发工具包（J2SDK）。所有 J2SDK 工具都位于安装目录中。可以显式从该目录中调用这些工具，或者把该目录添加到 PATH 环境变量中。将该目录添加到 PATH 环境变量中包括以下两个步骤：

设置 JAVA_HOME 变量并不是绝对必需的；然而，有的 java IDEs 和 java 工具需要设置该变量。

1. 创建一个新的名为 JAVA_HOME 的环境变量。

   JAVA_HOME=[path to JDK install directory]

   例如：

   JAVA_HOME=c:\j2sdk

2. 编辑 PATH 变量，以包括 JAVA_HOME 的 bin 目录。

   PATH=..;%JAVA_HOME%\bin

添加 PATH 变量允许从系统的任何目录中运行可执行文件（javac，java，javadoc 等等）。

为了编译基于服务器的应用程序，如 servlets 和 EJBs，也需要安装 Java2 企业版开发工具包并将其包括在类路径中。Java 应用程序服务器一般会提供这个工具包，或者从 Sun MicroSystems 提供的参考实现中获得。

ArcGIS Engine

ArcGIS Engine 开发工具包使用标准的 Java 本地接口（JNI，Java Native Interface）访问 ArcObjects 核心组件。当编译和运行应用程序时，需要在开发者的路径中包含一些本地动态库。必须保证包含正确的路径以将互操作引入本地 ArcObject 中。

本地*.dll 位于以下路径中：

..\ArcGIS\bin

ESRI 推荐设置一个 ARCENGINEHOME 环境变量并将 bin 目录添加到 PATH 变量中。尽管这不是使用 ArcGIS Engine 开发工具包所必须的，但所有开发范例都使用这个变量以保证类路径（classpath）设置是正确的。

设置 ARCENGINEHOME 变量：

ARCENGINEHOME=[path to ArcGIS install directory]

例如：

ARCENGINEHOME=c:\ArcGIS

编辑 PATH 变量使系统可以使用 ArcGIS Engine Runtime 随带的本地资源库。编辑 PATH 目录以包括 ARCENGINEHOM 的 jre\bin 目录。

PATH=..%ARCENGINEHOME%\java\jre\bin

类路径（Classpath）

Java API 为 ArcObjects 和本地运行时库提供了 Java Archive（JAR）文件（*.jar）。这些 JAR 文件位于磁盘上的以下目录 %ARCENGINEHOME%\java

所有用 ArcGIS 开发工具包构建的 Java 应用程序在各自的应用程序类路径中必须引用下列的 JARs：

•arcobjects.jar—在一个完整的文档中包含所有非 UI 类。

•jintegra.jar—包含处理与COM互操作的运行时库中的所有类。

另外，根据需要，应将单个的 arcgis_xxx.jar 文件加入到类路径中。例如，支持在 ArcObjects 中包含 Java 可视化 beans 的应用程序需要将 arcgis_visualbeans.jar 文件添加到类路径中。

JRE

ArcGIS Engine 和 Server 开发工具包包括一个 Java 运行时环境（JRE，Java Runtime Environment）版本。只要上面描述的必要设置在本地运行，就可以运行任何 ArcGIS Java 应用程序。用户可以注意到 bin 目录中的必要*.dlls 文件和库扩展目录中的必要*.jar 文件。启动这个运行时环境需要做的全部就是确保 bin 被添加到了 PATH 环境变量中。

PATH=..\ArcGIS\java\jre\bin

Java 编程技术

本节讲述 Java 编程语言的一些基本概念。这里假定开发者掌握了一般的编程概念，但对于 Java 却是一个新手。

Java 虚拟机的特性

Java 虚拟机（JVM）规程为执行代码提供了一个平台无关、抽象的计算器。JVM 不知道 Java 语言的任何方面，但理解一种特定的二进制格式，即包含字节码形式指令的类文件。Java 虚拟机规程提供了一个既可编译又可以解释程序的环境。编译器创建一个 java 文件，产生一系列字节码并把它们存贮在一个.class 文件中，而 Java 解释器执行存贮在.class 文件中的字节码。

JVM 的每个实现都与特定平台相关，因为 JVM 实际上要与操作系统进行交互。JVM 处理诸如内存分配、垃圾回收和安全监测等事务。

Java 本地接口（Java Native Interface）

即使 Java 程序设计要在多个平台上运行，也有可能标准的 Java 类库不支持某个特定应用程序所需的平台相关特性，或者某个 Java 程序需要实现更低层次的程序并具有调用它的 Java 程序。JNI 是一个由 Java 语言提供的标准的跨平台编程接口。它允许编写能操作其它编程语言，如 C 或 C++等，编写的应用程序和类库的 java 程序。这是一种用于在 ArcGIS 中沟通本地 ArcObject 和 Java API 的技术。

为了为 ArcObject 的本地使用而初始化 Java 环境，每个 ArcGIS Engine Java 应用程序必须调用 EngineInitializer 类上的静态 initializeEngine()方法。这应该是要进行的第一个调用，甚至在 AoInitialize 之前。

```
public static void main(String[] args){
/* always initialize ArcGIS Engine for native usage */
EngineInitializer.initializeEngine();
...
}
```

要了解 initializeEngine 方法如何作为第一次调用来使用，请参考 ArcGIS Developer Help 中的 Java 开发示例。

用 Java 进行 ArcGIS 开发

本节的目标读者是使用 Java SDK for ArcGIS Engine 的开发者。SDK 提供了与 ArcObjects 的互操作，允许开发者象访问 Java 对象一样访问 ArcObjects。Java API 不受限于任何特定的 Java 虚拟机或平台，并使用标准的 Java 本地接口访问 ArcObjects。Java API 通过 Java 类和接口提示了 ArcObjects 的全部功能，允许 Java 开发者编写一次代码就可以到处运行，并且也受益于 ArcObjects 组件重复。Java API 提供了产生于 ArcObjects 组件类型库（TLBs）中的“代理（Proxy）”类，允许与底层所有组件进行互操作。这些代理类提供了与 ArcObjects 的属性、方法和事件对等的 Java 对等物。

接口

本地 ArcObjects 使用基于接口的编程模式。接口概念对 ArcObjects 很重要，并强调以下四点：

1. 接口不是类。
2. 接口不是对象。
3. 接口是强类型相关的。
4. 接口是不可改变的。

ArcObjects 接口是抽象的，即没有与接口相关的实现。对象使用类型继承；与接口相关的代码来自类的实现。

该模型共享了 Java 接口模型的一些特性，并引入 Java 接口模型以强化单一继承模型。在 Java 语言中，接口是一种方法的规程，对象声明了该方法的实现。一个 Java 接口并不包含例程变量或实现代码。

Java API 为每个 ArcObjects 接口提供两个对象：一个相应接口和一个接口代理类。接口名以 ArcObjects 样式命名，并以字母 I 为前缀。接口代理类则在类名前加上 proxy。下面是这种映射的一个例子：

```
interface IArea : IUnknown public interface IArea{
        public class IAreaProxy implements IArea{
```

代理类在 Java API 内部使用以提供各个接口的实现。由于代理类没有实现代码，应用程序开发者从来不应使用这些类的缺省构造器。ArcObjects 要求开发者通过接口访问对象。Java 语言并不使用这个模式；因此，面向 ArcObjects 的 Java API 提供了两种访问对象的方法：通过接口或通过类访问。

```
/* use the class implementing IPoint */
IPoint iPoint = new Point();
/* access object through class */
Point cPoint = new Point();
```

但不能通过缺省的接口代理类访问对象：

```
IPointProxy proxyPoint = new IPointProxy(); // incorrect usage
```

在后面的几节中将更深层次地讨论这个问题。

ArcObjects 接口是不可改变的，因此从来没有版本更新。接口一旦定义和公布就从不改变。当接口需要增加新方法时，API 通过在接口名称后增加版本号来定义一个新的接口，如下表所示：

```
interface IGeometry : IUnknown   public interface IGeometry{
interface IGeometry2 : IGeometry   public interface IGeometry2 extends
                                        IGeometry{
interface IGeometry3 : IGeometry2 public interface IGeometry3 extends
                                        IGeometry2{
                        public interface IGeometry4 extends
                                        IGeometry3{
```

类

在 ArcObjects 模型中，类提供了所定义接口的实现。ArcObjects 提供三个类型的类：抽象类、类、组件对象类。可以通过 ArcGIS 开发中帮助中提供的对象模型图来区分这些类。在使用这三种“类”之前熟悉它们是极为重要的。

在 ArcObjects 中，一个抽象类不能用来创建新的对象，而在 Java API 中没有抽象类。抽象类在 ArcObjects 中是通过类型继承的子类例程的规程。抽象类列举了可以通过子类来执行的接口类型，而没有提供对那些接口的执行。对于 ArcObjects 中的每个抽象类，都有子类提供实现。

在 ArcObjects 中“类”不能公开创建；但是，“类”对象可以作为另一个类的属性创建或由来自另一个类的对象实例化。在 Java API 中，没有为 ArcObjects 类定义通常用来创建类的缺省构造器。

```
/* the constructor for FeatureClass() is unsupported*/
FeatureClass fc = new FeatureClass();
```

下面的例子解释了在进入打开一个要素类的过程时的这种行为。

```
IWorkspaceFactory wf = new ShapefileWorkspaceFactory();
IFeatureWorkspace fw = new
IFeatureWorkspaceProxy(wf.openFromFile(" \path \to \data", 0));
/* create a Feature Class from FeatureWorkspace */
IFeatureClass fc = fw.openFeatureClass("featureclass name");
```

在 ArcObjects 中，组件对象类是可以公开创建的类。这意味着可以仅仅通过声明一个新的对象来创建一个自己的对象，如下所示：

```
/* create an Envelope from the Envelope CoClass */
Envelope env = new Envelope();
```

结构

结构定义一个新的由成员元素构成的数据类型。Java 没有作为复杂数据类型的结构。Java 语言通过类来提供这种功能；可以用合适的例程变量来声明一个类。对于 ArcObjects 中的每种结构，都有一个代表性类，该类具有能匹配结构成员的公开声明的例程变量，下面给出了结构成员的轮廓。

```
struct WKSPointZ                public class _WKSPointZ ... {
   double X                     public double x;
   double Y                     public double y;
   double Z                      public double z;
        }
```

可以象使用 Java 中的任何其它类一样使用这些类：

```
_WKSPointZ pt = new _WKSPointZ();
pt.x = 2.23;
pt.y = −23.14;
pt.z = 4.85;

System.out.println(pt.x + " " + pt.y + " " + pt.z);
```

枚举

Java 没有枚举类型。为了在 Java 中模仿枚举，必须创建一个拥有常量的类或接口。对于本地 ArcObjects 中的每个枚举，都有一个 Java 接口，该接口具有公开声明的代表枚举值的静态整数。

```
enum esri3DAxis
    esriXAxis = 0
    esriYAxis = 1
    esriZAxis = 2
        }
```

```
public interface esri3DAxis {
public static final int esriXAxis = 0;
public static final int esriYAxis = 1;
public static final int esriZAxis = 2;
```

现在可以使用下面的符号表示引用 esriXAxis 常量：

```
esri3DAxis.esriXAxis;
```

Variants

Variant 数据类型包含一系列子类型。通过 Variants，所有类型可以被包含在一个单个类型的 Variant 中。Java 编程语言中的每个事物都是一个对象。如果需要的话，甚至原始数据类型也可以封装在对象中。Java 中的每个类都对 java.lang.Object 进行了扩展；因此，Java API 中的任何对象类型都可以传递给 ArcObjects 中以 Variant 为参数的方法。

以“variant”对象为参数调用方法

对于以 variants 为参数的方法，任何对象类型都能传递给它们，因为所有对象都源自 java.lang.Object。由于这被视为一种“扩展转换（windening cast）”，不需要显式地转换为“对象”。如果在需要 variants 时想将原始数据类型作为参数传递给方法，可以使用相应的原始封装（wrapper）类。

使用返回 variants 的方法

当使用由方法返回的 variant 对象时，要显式将那些对象“向下转换（downcast）”为相应的封装（wrapper）类。例如，如果希望得到一个 String，则向下转换为 java.lang.String；如果希望得到一个 Short，则向下转换为 Short 的封装对象类，即 java.lang.Short，如下列代码所示。

```
ICursor spCursor = spTable.ITable_search(spQueryFilter, false);
/*Iterate over the rows*/
IRow spRow = spCursor.nextRow();
while (spRow != null) {
    Short ID = (Short) (spRow.getValue(1));
    String name = (String) (spRow.getValue(2));
    Short baseID = (Short) (spRow.getValue(3));

System.out.println("ID="+ ID +"\t name="+ name +"\tbaseID="+
baseID);
/* Move to the next row.*/
spRow = spCursor.nextRow();
}
```

IrowBuffer是Irow的一个超接口，并定义了getValue(int)方法，如下所示：

```
public Object getValue(int index)
        throws IOException,
               AutomationException
The value of the field with the specified index.
Parameters:
index – The index (in)
Returns:
return value. A Variant
```

返回值为一个对象，javadoc 指定其为“variant”。因此，取决于其在被查询 geodatabase 中的类型，该值可转换为 String 或 Short。

转换（Casting）

ArcObjects 遵循基于接口的编程样式。许多方法使用接口类型作为参数并以接口为返回值。当某个方法的返回值为接口类型时，该方法返回一个实现该接口的对象。当某个方法以接口类型为参数时，该方法可以纳入任何实现该接口的对象。这种编程样式的优点是同一方法可以操作许多不同的对象类型，只要它们都实现了该接口。

例如，IFeature.getShape()方法返回一个实现 Igeometry 的对象。返回的对象可能为下列实现了 Igeometry 类中的一种：BezierCurve, CircularArc, EllipticArc, Envelope,GeometryBag, Line, MultiPatch, Multipoint, Path, Point, Polygon, Polyline, Ray,Ring, Sphere, TriangleFan, Triangles, TriangleStrip。

转换（casting）可用于类型之间的转换。Java API 开发者可以使用以下三种转换方法：

1. 接口到具体类的转换（casting）
2. 接口之间的转换（casting）
3. 接口向下转换（downcasting）

因为 JVM 中不存在对象引用，ArcObjects 中的方法返回的对象其行为不同于隐式定义的对象行为，理解这一点很重要。

如果有一个方法 doSomeProcessingOnPolygon(Polygon p)，只能操作 Polygon 对象，而又想将 IFeature.getShape()获取的结果对象传递给该方法，则需要一种将来自 Igeometry 的对象类型转换到 Polygon 的方法。在 Java 中，可以使用一个类的转换操作来完成：

```
/* incorrect usage: will give ClassCastException */
Polygon poly = (Polygon)geom;
```

然而，对 ArcObjects Java API 使用相同的代码将会抛出一个异常 ClassCastException。抛出该异常的原因是“geom”对象引用实际上是对本地 ArcObjects 组件的一个引用。Java 和本地 ArcObjects 组件间的互操作的结果使得这个对象引用到 Polygon 对象的转换逻辑驻留在 Polygon 对象构造器中，而不是在 JVM 中。

Java API 中每个类都有一个以单个对象为参数的构造器。该构造器可以通过引用 ArcObjects 组件来创建相应的对象。因此，当使用 Java API 时为获得某个类转换的对等物，要使用转换目标类的“对象构造器”。

```
Polygon poly = new Polygon(geom);
```

下面的代码展示了用于将 geom 对象“转换”到一个 Polygon 的对象构造器：

```
IFeature feature = featureClass.getFeature(i);
IGeometry geom = feature.getShape();
   if (geom.getGeometryType() ==
   esriGeometryType.esriGeometryPolygon){
   /*Note: "Polygon p = (Polygon) geom;" will give
   ClassCastException*/
   Polygon poly = new Polygon(geom);
   doSomeProcessingOnPolygon(poly);
}
```

这样构造的 Polygon 对象将实现 Polygon 类实现了的所有接口。因此，可以调用属于 Poly 对象上任何实现了的接口的方法。

可以用单独对象构造器编写所有代码，但有的时候可能把要实现一个特定接口的某个对象转换为该对象实现的另一个接口而不是某个类类型更好。

前面的例子中，假设想使用 doSomeProcessingOnPolygon(Polygon p)方法在 Polygon 对象或其它对象上实现 IArea 的对象，如 Envelope 和 Ring。可以编写一般的 doSomeProcessingOnPolygon(Polygon p)方法，以操作实现了 IArea 接口的所有对象的。由于 Polygon, Envelope 和 Ring 对象都实现了 IArea 接口，可以将这些对象传入该一般方法，从而防止为每个对象类型编写其他方法，诸如 doSomeProcessingOnEnvelope(Envelope env) 和 doSomeProcessingOnRing(Ring ring)等。为完成这个任务，需要将 IGeometry 类型转换到 IArea 类型。在 Java 中，这通常用接口之间的转换来完成。

```
/* incorrect usage: will give ClassCastException */
IArea area = (IArea) geom ;
```

然而，与上面类转换中的所说的原因一样，这种转换会以异常 ClassCastException 而告失败。为了能转换为 ArcObjects 接口，需要使用前一节中讨论的接口代理类。在 Java API 中，通过使用转换目标接口的 InterfaceProxy 可以实现与接口之间转换等价的功能。

```
IArea area = new IAreaProxy(geom);
```

下列代码显示了 InterfaceProxy 类在将 geom 对象到 IArea 的接口间转换的使用：

```
IFeature feature = featureClass.getFeature(i);
IGeometry geom = feature.getShape();
/*Note: "IArea area = (IArea) geom;" will give ClassCastException*/
IArea area = new IAreaProxy(geom);
```

```
doSomeProcessingOnArea(area);
```

象上述代码中那样使用 IareaProxy 类允许通过其 IArea 接口访问该对象，这样该对象就能传递给以类型 IArea 为参量的方法。因此，在这个特定的例子中，一个方法可以处理三个不同的对象类型。然而，只有属于 IArea 接口的方法才对 IArea 对象有效。为调用对象的其它方法，需要用对象构造器将类转换为合适的对象类型，或使用 InterfaceProxy 类获得对其它接口的一个引用。

Instanceof

Java 中的 Instanceof 算子使开发者可以决定其第一个运算对象是其第二运算对象的例程。

```
operand1 instanceof operand2
```

当类型背后的逻辑存在于 Java 中时，可以在 ArcObjects—Java 中使用 Instanceof 算子。当类型存在于 ArcObjects 中，而决定一个对象是否为某个特定类型的例程的逻辑驻留在该对象类型的构造器而不是 JVM 中时，不能使用 Instanceof 算子。

```
Point point = new Point();
point.putCoords(0, 0);
point.putCoords(10, 10);
point.putCoords(20, 20);

if(point instanceof IGeometry){
System.out.println(" point is a IGeometry");
geom = point;
}
if(point instanceof IClone){
System.out.println(" point is a IClone");
}
```

由于 Point 的类型信息存在于 Java 中，上述代码可以起作用。在构造一个 Point 对象时，也为每个实现接口构造了一个代理类。这允许在这些类型的任何一个上使用 Instanceof 算子。开发者可能已经访问了实现 IGeometry 或 Iclone 接口的 Point 上的任何方法。

这也可以向后兼容：

```
if(geom instanceof Polyline){
    System.out.println(" geom is a Polyline");
}
    else if(geom instanceof Point){
    System.out.println(" geom is a Point");
pnt = (IPoint)geom; // allowable cast as the type is held in JVM
}
```

因为直接将 geom 对象转换到了 Point 对象，geom 对象就是类型 Point，而且可以用 Instanceof 算子来检测这个信息。然而，由于该类型信息在进行上面的检测前就已经知道了，因此并不是非常有用。有用的应该是将上述逻辑应用到返回超接口的对象的方法上。

设想返回一个 IWorkspace 的 IWorkspaceFactory.openFromFile()方法。由于返回的对象是一个实现了 IWorkspace 的 Java 对象，无法检测返回的对象是否为已知的实现 IWorkspace 类中的任何一个。在这种情况下，为了检测类型信息，应调用一个被期望的返回对象上的一个方法。如果该方法没有抛出异常，则就是那种类型。发生这种情况是因为这个对象的逻辑在运行时声明并存在底层 ArcObjects 组件中。

```
RasterWorkspaceFactory rasterWkspFactory = new
RasterWorkspaceFactory();
IWorkspace wksp = rasterWkspFactory.openFromFile( aPath,
0 );

if(wksp instanceof RasterWorkspace){
   /*code does not execute as logic is in ArcObjects*/
   System.out.println(" wksp is a RasterWorkspace");
   rasWksp = (RasterWorkspace)wksp;
}
else{
   try{
      rasWksp = (RasterWorkspace)wksp;
      rasWksp.openRasterDataset( aRaster );

}catch(Exception e){
      /*code executes if wksp is not a RasterWorkspace*/
      System.out.println(" wksp is not a RasterWorkspace");
   }
}
```

获取参数的方法

ArcObjects 提供了许多返回多个值的方法。Java API 要求传送单个元素数组的参数到这种方法中。最基本的，要传入想返回的对象的单个元素数组，而 ArcObjects 是那些返回值数组的第一个元素。当从方法调用返回时，数组的第一个元素包含了在方法调用过程中已设定的值。本节中将使用的这样的一个方法就是 IARMap 接口的 toMapPoint 方法。查看一下这种方法的 javadoc：

```
public void toMapPoint(int x,
                       int y,
                       double[] xCoord,
                       double[] yCoord)
    throws IOException,
                AutomationException

Converts a point in device coordinates (typically pixels) to
coordinates in map units.
Converts the x and y screen coordinates supplied in pixels to x and
y map coordinates. The returned map coordinates will be in
MapUnits.

Parameters:
x – The x (in)
```

```
y – The y (in)
xCoord – The xCoord (in/out: use single element array)
yCoord – The yCoord (in/out: use single element array)
```

注意参数 xCoord 和 yCoord 标记为“in/out: use single element array“。为使用这个方法，前两个参数是以像素为单位的 x 和 y 坐标。接下来的两个参数真正用于获得从方法调用中返回的值。可以传入一维单元素的双精度数组：

```
double [] dXcoord = {0.0};
double [] dYcoord = {0.0};
```

当方法调用完成时，可以查询 dXcoord[0] 和 dYcoord[0]的值。这些值会被该方法修改并将实际指代以地图单位表示的 x 和 y 坐标。一个该方法调用的实际例子就是当鼠标经过控件时用当前地图坐标更新状态条。

```
public void updateStatusBar(IARControlEventsOnMouseMoveEvent
params)throws IOException {
   /*create single dimensional array of doubles
   *the values of the first element of the arrays will be filled in
   *by the arMap.toMapPoint(...) method
   */
   double [] dXcoord = {0.0};
   double [] dYcoord = {0.0};
   int screenX = params.esri_getX();
   int screenY = params.esri_getY();
   IARMap arMap =
   arControl.getARPageLayout().getFocusARMap();
   arMap.toMapPoint(screenX, screenY, dXcoord, dYcoord);
   /*set the statusLabel*/
   statusLabel.setText( “Map x,y: “ + dXcoord[0]+”,
    “+dYcoord[0]);
}
```

Java API 不允许开发者采用超类类型填充数组，即使已经转换到了一个超类类型。考虑一下下面的 Java 例子：

```
Integer[] integers = { new Integer(0), new Integer(1), new Integer(2)};
Object[] integersAsObjects = (Object[])integers;
integersAsObjects[0] = new Object();
```

上面这样是不允许的，并会抛出一个异常 ArrayStoreException。考虑一下下面的 ArcObjects 例子：

```
Polyline[] polyline = {new Polyline()};
tin.interpolateShape( breakline, polyline, null );
Polyline firstPolyLine = polyline[0];
```

与前一个例子一样，上面这样做是不允许的，并会抛出相同的异常 ArrayStoreException。请阅读 Isurface 的 interpolateShape()方法并分析这里会发生什么结果。

```
public void interpolateShape(IGeometry pShape,
            IGeometry[] ppOutShape,
```

```
            Object pStepSize)
    throws IOException,
            AutomationException
```

Parameters:

pShape – A reference to a com.esri.arcgis.geometry.IGeometry (in)
ppOutShape – A reference to a com.esri.arcgis.geometry.IGeometry
(out: use single element array)
pStepSize – A Variant (in, optional, pass null if not required)

Throws:

IOException – If there are communications problems.
AutomationException – If the remote server throws an exception.

IGeometry 是 IPolyline 的一个超接口，且 Polyline 类实现了这两个接口。在第一个尝试中试着传送一个单元素 Polyline 数组到需要一个 in/out IGeometry 参数的方法中。这会抛出一个异常 ArrayStoreException，因为 ArcObjects 试图用 IGeometry 对象填充一个 IPolyline 数组，试图将一个超类类型放入一个子类数组中。下面概括了使用这个方法的正确方式：

```
/*Set up the array and call the method*/
IGeometry[] geoArray = {new Polyline()};
tin.interpolateShape( breakline, geoArray, null );
/* "Cast" the first array element as a Polyline – this is
* the equivalent of calling QueryInterface on IGeometry
*/
IPolyline firstPolyLine = new IPolylineProxy(geoArray[0]);
```

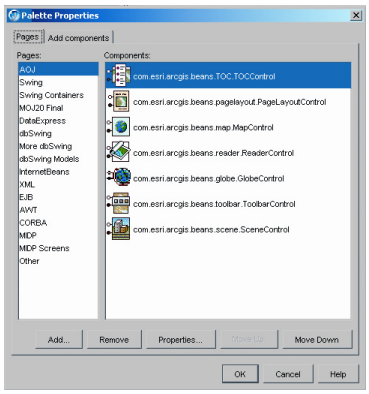
非OLE自适应类型（Non-OLE Automation Compliant Types）

不是 OLE 自适应类型的 ArcObjects 类型不能在 Java API 中起作用。ArcObjects 包含少数几个这种方法，不幸的是，这种方法不能在 Java API 中使用。但是，在大多数情况下，已经添加了追加接口，这些追加接口有一些方法作为自动化兼容而重写。这些新接口前面都添加了字母“GEN”，表明它们是普通针对所有支持的 API 的。

```
IPoint[] points = new Point[2];
points[0] = new Point();
points[1] = new Point();

points[0].putCoords(0, 0);
points[1].putCoords(10, 10);

IEnvelope env = new Envelope();
IEnvelopeGEN envGEN = new Envelope();
/*not automation compatible – throws exception*/
env.defineFromPoints(2, points[0]);
/*automation compatible*/
envGEN.defineFromPoints(points);
```



使用可视化 beans

Java API 提供了一套可重用组件，是设计用于提供图形功能的预建软件代码块。可视化 beans 开发者只需编写代码以将它们作为“伙伴”应用到 ArcGIS Engine 应用程序中。Beans 的使用在 Java 和 ArcGIS 提供的 ActiveX 控件之间搭建了一座桥梁。这些可视化组件是重量级 AWT 组件，并与 JavaBeans™组件架构相一致，允许其在 JavaBeans 兼容的 IDEs 中用作拖放组件来设计 Java 用户界面。

重量级组件与轻量级组件的混合

Swing 体系的主要目标之一就是要基于现有的 AWT 体系。这允许开发者在同一个应用程序中混合使用 Swing 和 AWT 两种组件。当使用具有 Swing 组件的 ArcObject JavaBeans 时，在混合使用重量级组件与轻量级组件时应该十分仔细。这方面的指导方针可以参考网址 http://java.sun.com/products/jfc/tsc/articles/mixing/上的“混合重量级和轻量级组件”一文。

如果使用 Swing 组件，在选项可用的地方，使用类似于下面的代码使轻量级组件失效：

```
jComboBox.setLightWeightPopupEnabled(false);
jPopupMenu.setLightWeightPopupEnabled(false);
```

侦听事件

所有的 ArcObject JavaBeans 都可以触发事件。例如，ARControl bean 可以触发下列事件：

```
void onAction(IARControlEventsOnActionEvent theEvent)
void onAfterScreenDraw(IARControlEventsOnAfterScreen
DrawEvent theEvent)
void onBeforeScreenDraw(IARControlEventsOnBeforeScreen
DrawEvent theEvent)
void onCurrentViewChanged(IARControlEventsOnCurrentView
ChangedEventtheEvent)
void onDocumentLoaded(IARControlEventsOnDocumentLoa
dedEvent theEvent)
void onDocumentUnloaded(IARControlEventsOnDocument
UnloadedEventtheEvent)
void onDoubleClick(IARControlEventsOnDoubleClickEvent
theEvent)
void onFocusARMapChanged(IARControlEventsOnFocusARMap
ChangedEvent
theEvent)
void onKeyDown(IARControlEventsOnKeyDownEvent theEvent)
void onKeyUp(IARControlEventsOnKeyUpEvent theEvent)
void onMouseDown(IARControlEventsOnMouseDownEvent
theEvent)
void onMouseMove(IARControlEventsOnMouseMoveEvent
theEvent)
void onMouseUp(IARControlEventsOnMouseUpEvent theEvent)
```

为了给事件添加和删除侦听器，beans 提供 addXYZEventListener 和 removeXYZEventListener 方法。为了便于创建侦听器对象，提供了 Adapter 类。

```
public void addIARControlEventsListener(IARControlEvents
    theListener)throws IOException

public void removeIARControlEventsListener(IARControlEvents
theListener)
    throws IOException
```

下列代码显示了使用具有 IARControlEventsAdapter 的匿名内部类为 onDocumentLoaded 和 onDocumentUnloaded 事件给 arControl 对象添加事件侦听器。

```
arControl = new ARControl();
...
/*wire up the events for arControl*/
arControl.addIARControlEventsListener(new
IARControlEventsAdapter(){
    public void onDocumentLoaded(IARControlEventsOnDocument
    LoadedEvent evt)
    throws IOException{
        /*set the statusbar text to point to the currently loaded
        document*/
        statusLabel.setText(" Document filename: "+
arControl.getDocumentFilename());
        /*Determine whether permission to toggle TOC visibility*/
        if (arControl.hasDocumentPermission(
esriARDocumentPermissions.esriARDocumentPermissionsViewTO
C))
        {
        tocVisibilityCheckBox.setEnabled(true);
        tocVisibilityCheckBox.setSelected(arControl.isTOCVisible());
        } else {
        JOptionPane.showMessageDialog((Component)arg0.getSource
(),"You do not have permission toggle TOC visibility");
        }
}

public void onDocumentUnloaded(IARControlEventsOnDocument
UnloadedEventevt)
    throws IOException{
        /*set the statusbar text to empty string*/
        statusLabel.setText("");
    }
});
```

值得注意的是，由 beans 触发的事件是自定义事件，事件侦听器作为 Java API 的一部分提供给它们。从 java.awt.event 包中加载侦听器（如 MouseListener）是没有用的，因为 Java beans 不触发那些事件。相反，可以使用由相应的事件侦听器提供的类似事件，诸如 onMouseDown，onMouseUp 和 onMouseMove 等事件，而在 ARControl 示例中则是 IARControlEvents。

C++是面向对象的编程语言，是 20 世纪 80 年代中期从 C 发展而来。C++语言具有许多为其提供无与伦比的表现力的特性，如面向对象的继承、操作符重载、虚函数、模板和一个称为“标准模板库（STL， Standard Template Library)”的有用和常用必要函数库。C++语言已经被国际标准化组织（ISO）和几个有影响的国家标准组织标准化。

开发人员可能会考虑使用 ArcGIS C++ API，与其它 API 相比，使用 C++ API 的原因如下：

- 执行速度——C++代码执行速度明显比对应的 Java、VB 和 C＃代码快。
- 跨平台兼容——VB 和 C#当前主要用于 Windows 平台。C++和 Java 本身就可以跨更多平台。
- 已有经验——如果组织机构中的开发人员已经具备大量使用该语言的经验，那么 C++就是一个合理的选择。

本节打算用于两个目的：

1. 让用户熟悉一般的 C++编程风格和调试。
2. 提供 ArcGIS C++ API 入门介绍，详细阐述具体的使用需求，并就使用 ArcObjects 编程平台提供一些建议。

C++开发技术

命名习惯

类型名

所有类型名称（class、struct、enum 和 typedef）都以一个大写字母开头，而名称其它部分则为大小写混合：

```
class Foo : public CObject {...};
struct Bar {...};
enum ShapeType {...};
typedef int* FooInt;
```

函数指针（回调）的 Typrdef 以 Proc 作为名称的结尾。

```
typedef void (*FooProgressProc)(int step);
```

枚举值都以一个标识该工程的小写字符串开始；在 ArcObjects 中为 esri,每个字符串都是单独一行：

```
typedef enum esriQuuxness
{
esriQLow,
esriQMedium,
esriQHigh
} esriQuuxness;
```

这里是命名习惯的一些建议。这有助于识别变量的用法和类型以减少编程错误。这是有删节的匈牙利命名规则：

[<scope>_]<type><name>

前缀	变量范围
m	例程类成员
c	静态类成员（包括常量）
g	全局静态变量
无	局部变量或结构或公共类成员

<type>

前缀	数据类型
b	布尔型
by	字节或无符号字符
cx/cy	短整型，用作大小
d	双精度
dw	DWORD，双精度字或无符号长整型
f	浮点型
fn	函数
h	句柄
i	整型
ip	聪明指针
l	长整型
p	指针
s	字符串
sz	ASCIIZ 无结束符字符串
w	WORD 无符号整型
x,y	短整型，用作坐标

<name>描述变量怎样使用或它包含的内容。<scope>和<type>部分应总是小写，而<name>应使用大小写混合。

变量名	描述
m_hWnd	HWND 句柄
ipEnvelope	COM 接口聪明指针
m_pUnkOuter	对象指针
c_isLoaded	静态类成员
g_pWindowList	全局对象指针

函数名

用以下习惯为函数命名：

对于简单取值和赋值函数，使用 Get<Property> 和 Set<Property>：

```
int GetSize();
void SetSize(int size);
```

如果客户端提供对结果的存储，使用 Query<Property>：

```
void QuerySize(int& size);
```

对于状态函数，使用 Set<State> 和 Is<State> 或 Can<State>：

```
bool IsFileDirty();
void SetFileDirty(bool dirty);
bool CanConnect();
```

在操作语义明显来自变量类型的地方，函数名中不包含类型名。

不用：

```
AddDatabase(Database& db);
```

考虑用：

```
Add(Database& db);
```

不用：

```
ConvertFoo2Bar(Foo* foo, Bar* bar);
```

考虑用：

```
Convert(Foo* foo, Bar* bar)
```

如果客户端将某些数据的所有权交给对象，使用 Give<Property>。如果对象将某些数据的所有权交给客户端，使用 Take<Property>：

```
void GiveGraphic(Graphic* graphic);
Graphic* TakeGraphic(int itemNum);
```

当某个特定的操作使用不同的参量类型时，使用函数重载：

```
void Append(const CString& text);
void Append(int number);
```

参数名

在函数声明中使用描述性参数名。参量名应该很清楚地指示参数的用途：`bool Send(int messageID, const char* address, const char* message);`

智能类型

智能类型是行为像普通类型的对象。它们是封装了一种数据类型的 C++ 类的实现，用操作符和函数将其包含。这些操作符和函数使得操作底层类型更容易且更少出错。当这些智能类型封装一个接口指针时，它们就称为聪明指针。智能指针操作 IUnknown 接口以保证正确地管理资源的分配和存储单元分配。通过不同的函数、构造和析构方法和重载操作来实现这些功能。

智能类型使得操作 COM 接口和数据类型的任务更简单，因为许多 API 的调用可以移植到一个类的实现中；可是使用它们必须谨慎，必须清楚了解它们与封装的数据类型进行互操作的方法。

C++ API 支持的智能类型包括：

- _com_ptr_t —这个类封装一个创建智能指针的 COM 接口指针。
- CComBSTR－封装 BSTR 数据类型的类。
- CComVariant－封装 VARIANT 数据类型的类。

可以使用宏_COM_SMARTPTR_TYPEDEF 来为一个接口定义一个智能指针，像下面一样：

```
_COM_SMARTPTR_TYPEDEF(IFoo, __uuidof(IFoo));
```

编译器可以如下代码所示对此进行扩展：

```
typedef _com_ptr_t< _com_IIID<IFoo, __uuidof(IFoo)> > IFooPtr;
```

一旦声明后，声明一个变量为接口类型并在接口末端追加 Ptr 是很简单的一件事。下面是在许多 C++例子中比较常见的该类智能指针的通常用法。

```
// Get a CLSID GUID constant
extern “C” const GUID __declspec(selectany) CLSID_Foo = \
{0x2f3b470c,0xb01f,0x11d3,{0x83,0x8e,0x00,0x00,0x00,0x00,0x00,0x00}};

// Declare Smart Pointers for IFoo, IBar and IGak interfaces
_COM_SMARTPTR_TYPEDEF(IFoo, __uuidof(IFoo));
_COM_SMARTPTR_TYPEDEF(IBar, __uuidof(IBar));
_COM_SMARTPTR_TYPEDEF(IGak, __uuidof(IGak));

HRESULT SomeClass::Do()
{
   // Create Instance of Foo class and QueryInterface (QI) for IFoo
   interface
   IFooPtr ipFoo;
   HRESULT hr = ipFoo.CreateInstance(CLSID_Foo);
   if (FAILED(hr)) return hr;

   // Call method on IFoo to get IBar
   IBarPtr ipBar;
   hr = ipFoo->get_Bar(&ipBar);
   if (FAILED(hr)) return hr;

   // QI IBar interface for IGak interface
   IGakPtr ipGak(ipBar);

   // Call method on IGak
   hr = ipGak->DoSomething();
   if (FAILED(hr)) return hr;
```

```
    // Explicitly call Release()
    ipGak = 0;
    ipBar = 0;

    // Let destructor call IFoo's Release
    return S_OK;
}
```

*如果以前使用过智能指针，可能看到了智能类型在使用比较操作符号（“==”）时的不同之处。COM 规定的状态对象标识通过比较 IUnknown*指针值来进行。当使用“==”时，智能指针将执行必要的 QI 和比较。*

使用 CComBSTR 时，使用文字映射 L“”来声明 OLECHAR 字符串常量。在命令行显示一个 CComBSTR，使用 wccerr。需要包含输入输出流来使用 wcerr。

```
CComBSTR bsName(L"Matt");
std::wcerr << L"The name is " << (BSTR) bsName << std::endl;
```

CComVariant 直接源自于 VARIANT 数据类型，意味着它的实现没有重载，这样反过来简化它的使用。它包含一个构造函数和函数的丰富集合，使得操作 VARIANT 简单直接；甚至包含一些读写流的方法。重用变量以前确认调用 Clear 方法。

omVariant 将创建一个短整型（VT_I2 类型）变量而不是期望的布尔型（VT_BOOL 类型）变量。可以使用 CcomVariant（true）来创建一个布尔变量。

```
ipFoo->put_Name(CComBSTR(L"NewName"));
if FAILED(hr)) return hr;

// Create a VT_I4 variant (signed long)
CComVariant vValue(12);

// Change its data type to a string
hr = vValue.ChangeType(VT_BSTR);
if (FAILED(hr)) return hr;
```

在 IDL 中的一些方法调用被标记为可选，并占用一个变量参数。然而在 VC++中，这些参数仍必须提供。为了表示一个参数值不被支持，会传递一个变量来指示错误代码或类型 DISP_E_PARAMNOTFOUND：

CComVariant myVar(ipSmartPointer)将得到一个布尔型（VT_BOOL）变量而不是期望带有对象引用（VT_UNKNOWN）的变量。最好是传递明确类型给构造函数，如输入哪些自身不是带有重载强制转换操作符的聪明类型的类型。

```
CComBSTR documentFilename(L"World.mxd");

CComVariant noPassword;
noPassword.vt = VT_ERROR;
noPassword.scode = DISP_E_PARAMNOTFOUND;
HRESULT hr = ipMapControl->LoadMxFile(documentFilename,
noPassword);
```

不过，如果有一个希望传递到变量的值，使用智能类型，CComVariant。

```
int val = 1;
CComVariant smartVal(val);
ipRowBuffer->put_Value(2, smartVal);
```

```
// Perform QI it IUnknown
IUnknownPtr ipUnk =
    ipSmartPointer;
// Ensure we use IUnknown*
    constructor
    of CComVariant
CComVariant
myVar2(ipUnk.GetInterfacePtr());
```

使用 CComBSTR 和 CComVariant 时，Detach（）函数从智能类型中释放底层数据类型，并在将结果作为方法的[out]参数传递时可以使用这个函数。下面就是和 CComBSTR 一起使用 Detach 函数的例子：

```
HRESULT CFoo::get_Name(BSTR* name)
{
   if (name==0) return E_POINTER;
   CComBSTR bsName(L”FooBar”);
   *name = bsName.Detach();
}
```

使用智能指针的常用练习就是使用 Detach（）来从一个方法调用返回一个对象。当返回一个接口指针时，COM 标准是增加在方法实现中的[out]参数的引用数。当不再需要指针时，调用 Release 是调用者的责任。因此，必须注意避免在成员变量上直接调用 Detach()函数，下面就是一个典型的模式：

```
HRESULT CFoo::get_Bar(IBar **pVal)
{
   if (pVal==0) return E_POINTER;

   // Constructing a local smart pointer using another smart pointer
   // results in an AddRef (if pointer is not 0).
   IBarPtr ipBar(m_ipBar);

   // Detach will clear the local smart pointer and the
   // interface is written into the output parameter.
   *pVal = ipBar.Detach();

   // This can be combined into one line
   // *pVal = IBarPtr(m_ipBar).Detach();

   return S_OK;
}
```

上面的模式与下面的代码有相同的结果，注意在 AddRef 可被调用前，需要 0 指针的条件测试，在一个 0 指针上调用 AddRef（或其它的方法）将导致一个访问违例异常并且通常使应用程序崩溃：

```
HRESULT CFoo::get_Bar(IBar **pVal)
{
   if (pVal==0) return E_POINTER;

   // copy the interface pointer (no AddRef) into the output parameter
   *pVal = m_ipBar;

   // Make sure interface pointer is non 0 before calling AddRef
   if (*pVal)
   *pVal->AddRef();

   return S_OK;
}
```

当使用一个智能指针时从方法上的[out]参数获取一个对象时，使用智能指针的“&”解除引用操作符。这将引起在智能指针上的上一个接口指针被释放。智能指针就被载入新的[out]值。这个方法的实现将已经增加了引用对象数。当智能指针越界时，这个值将被释放。

```
{
   IFooPtr ipFoo1, ipFoo2;
   ipFoo1.CreateInstance(CLSID_Foo);
   ipFoo2.CreateInstance(CLSID_Foo);

   // Initalise ipBar Smart pointer from Foo1
   IBarPtr ipBar;
   ipFoo1->get_Bar(&ipBar);

   // The  “&”  de-reference will call Release on ipBar
   // ipBar is then repopulate with a new instance of IBar
   ipFoo2->get_Bar(&ipBar);
}

// ipBar goes out of scope and the smart pointer destructor calls Release
```

调试器

Visual C++提供了特色丰富的调试器。这些提示会帮助我们从调试任务中获得最多的信息。

失败后的后退

当一个函数调用失败时，并且想知道出错原因（通过跳入函数），不必重新启动应用程序。使用 Set Next Statement 命令重新设置程序光标位置回到出错的语句上（右键单击该语句打开调试弹出式菜单）。然后跳入该函数。

编辑和继续

Visual Studio 6 允许在调试期间修改源代码。这些修改可以被重新编译并合并到可执行代码中，并不需要停止调试器。对于可修改的类型，有一些限制；在这种限制情况下，必须重新启动调试任务。缺省条件下激活这个特征；在工程菜单的 Settings 命令中可以设置这些。单击 C/C++选项卡，然后从 Category 下拉列表中选择 General。在 Debug info 下拉列表中，单击 Program Database for Edit and Continue。

Unicode 字符串显示

要设置调试器选项来显示 Unicode 字符串，单击 Tools 菜单，单击 Options，单击 Debug，然后选中 Display Unicode Strings 复选框。

变量值显示

将光标停留在源代码中一个变量名上查看它的当前值。如果该变量是一个结构，单击该变量，弹出 QuickWatch 对话框（单击 Eyeglasses 图标或按下 Shift+F9）或将之拖到 Watch 窗口中。

不停靠窗口

如果 Output 窗口（对这种情形，或者是任何停靠的窗口）看起来太小，试着通过右键单击该窗口并触发 Docking View 项目来不停靠它以使它成为一个真窗口。

条件断点

当满足某种条件时需要在断点上停留时，采用有条件的断点—例如，当一个 for 循环语句达到某一特定计数器值时。为达到这一目的，设置正常断点，然后调出 Breakpoints 窗口(Ctrl+B or Alt+F9)。选择设置的断点，并单击 Condition 按钮显示一个对话框，在其中设定断点条件。

预先加载 DLLs

在执行程序之前可以预先加载要调试的 DLLs。这样就使得你可以从一开始设置断点而不是一直等到程序执行中 DLL 被加载之后。操作如下，单击 Project，单击 Settings，单击 Debug，单击 Category，然后单击 Additional DLLs。然后单击列表区域来添加想要预先加载的任意 DLLs。

改变显示格式

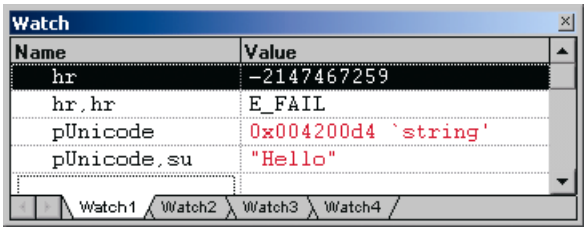
可以在 QuickWatch 对话框中改变变量的显示格式，或在 Watch 窗口中使用下表中的格式符号。

符号	格式	值	显示
d,i	有符号十进制整型	0xF000F065	-268373915
u	无符号十进制整型	0x0065	101
o	无符号八进制整型	0xF065	0170145
x,X	十六进制整型	61541	0x0000F065
l,h	d,l,u,o,x,X的长或短前缀	00406042,hx	0x0C22
f	有符号浮点型	3./2.	1.500000
e	有符号科学记数法	3./2.	1.500000e+00
g	短的 e 或 f	3./2.	1.5
c	单字符	0x0065	‘e’
s	字符串	0x0012FDE8	“Hello”
su	Unicode 字符串		“Hello”
hr	字符串	0	S_OK

使用格式符号，输入变量名后面带有逗号和适合的符号。例如，如果变量有一个 0x0065 值，并且想要以字符形式看这个值，在 Watch 窗口的列表中的 Name 栏输入“var,c”。

可以只对结构、数组、指针和对象使用格式化符号来作为未扩展的变量，如果扩展变量，指定的格式影响所有成员。不能对单个成员使用格式化符号。

当按下时 Enter 时，字符格式值出现：var,c = ‘e’。同样地，假定 hr 是一个拥有 HRESULTs 的变量，在 Name 栏中输入“hr,hr”，以人们可读的格式查看 HRESULT。



可以用下表中的格式化符号格式化存储位置的内容。

符号	格式	值
ma	64 位 ASCII 字符	0x0012ffac .4...0...".0W&.. .....1W&.0.:W..1 ...."..1.JO&.1.2 .."..1...0y....1
m	16 字节十六进制，后面接着 16 个 ASCII 字符	0x0012ffac B3 34 CB 00 84 30 94 80 FF 22 8A 30 57 26 00 00 .4...0...."..0W&..
mb	16 字节十六进制，后面接着 16 个 ASCII 字符	0x0012ffac B3 34 CB 00 84 30 94 80 FF 22 8A 30 57 26 00 00 .4...0...".0W&..
mw	8 个字	0x0012ffac 34B3 00CB 3084 8094 22FF 308A 2657 0000
md	4 个双精度字	0x0012ffac 00CB34B3 80943084 308A22FF 00002657
mu	2 字节字符（Unicode）	0x0012fc60 8478 77f4 ffff ffff 0000 0000 0000 0000

使用存储定位格式化符号，可以输入任何值，或计算一个位置值的表达式。要显示一个字符数组为字符串，在数组名前加一个“&”，&yourname。格式化字符也可遵循一个表达式：

- rep+1,x
- alps[0],mb
- xloc,g
- count,d

查看某个地址的值，或一存储点的值，使用 BY、WO 或 DW 操作符：

- BY字节指向的内容
- WO返回字指向的内容
- DW返回双字指向的内容

操作符后面跟上变量、存储器或常量。如果 BY、WO 或 DW 操作符后面跟上一个变量，那么环境监视包含该变量地址的字节、字或双字。也可以使用上下文操作符{ }显示任何位置的内容。

使用 su 格式区分符在 Watch 窗口或 QuickWatch 对话框中显示 Unicode 字符串。使用 mu 格式区分符在 Watch 窗口或 QuickWatch 对话框中用 Unicode 字符显示数据字节。

键盘快捷键

大量的键盘快捷键会使对 Visual Studio Editor 的操作更加快捷。下面列出一些比较有用的键盘快捷键。

文本编辑器使用许多 Windows 应用程序（例如 Word）使用的标准快捷键。下面列出了一些详细的源代码编辑快捷键。

快捷键	动作
Alt+F8	基于包围线正确缩进所选代码
Ctrl+]	查找匹配的大括号（brace）
Ctrl+J	显示成员列表
Ctrl+Spacebar	一旦输入的几个字母能让编辑器认识就完成单词。在完成函数和变量名时很有用。
Tab	将所选代码向右缩进一个制表符
Shift+Tab	将所选代码向左缩进一个制表符

下面是在调试器中使用的通用键盘快捷键表。

快捷键	动作
F9	在当前行添加断点或从当前行删除断点
Ctrl+Shift+F9	删除所有断点
Ctrl+F9	使断点无效
Ctrl+Alt+A	显示 auto 窗口并将光标移入该窗口
Ctrl+Alt+C	显示调用栈窗口并将光标移入该窗口
Ctrl+Alt+L	显示局部窗口并将光标移入该窗口
Ctrl+Alt+A	显示 auto 窗口并将光标移入该窗口
Shift+F5	结束调试会话
F11	每次执行代码的一条语句，步入函数
F10	每次执行代码的一条语句，跳过函数
Ctrl+Shift+F5	重新启动一个调试会话
Ctrl+F10	从当前语句继续执行到所选语句
F5	运行应用程序
Ctrl+F5	脱离调试器运行应用程序
Ctrl+Shift+F10	设置下一条语句
Ctrl+Break	停止执行

加载下面的快捷键会大大提高使用 Visual Studio 开发环境的生产率。

快捷键	动作
ESC	关闭一个菜单或对话框，取消进程中的一个操作，或将焦点放在当前文档窗口
CTRL+SHIFT+N	创建一个新的文件
CTRL+N	创建一个新的工程
CTRL+F6 或 CTRL+TAB	在 MDI 子窗口中循环，每次一个窗口
CTRL+ALT+A	显示 auto 窗口并将光标移入该窗口
CTRL+ALT+C	显示调用栈窗口并将光标移入该窗口
CTRL+ALT+T	显示文档轮廓窗口并将光标移入该窗口
CTRL+H	显示查找窗口
CTRL+F	显示查找窗口。如果当前没有查找标准，将光标下的单词放入查找框中
CTRL+ALT+I	显示立即窗口并将光标移入该窗口。如果在文本编辑器窗口中，则此快捷键不能用
CTRL+ALT+L	显示局部窗口并将光标移入该窗口
CTRL+ALT+O	显示输出窗口并将光标移入该窗口
CTRL+ALT+J	显示工程浏览器窗口并将光标移入该窗口
CTRL+ALT+P	显示属性窗口并将光标移入该窗口
CTRL+SHIFT+O	打开一个文件
CTRL+O	打开一个工程
CTRL+P	打印所有或部分文档
CTRL+SHIFT+S	保存所有文件、工程或文档
CTRL+S	选择所有
CTRL+A	保存当前文档或所选项

使用在线帮助主题

右键单击工具条的空白区域显示所有可用工具的列表。Infoviewer 工具条包含上下箭头，通过它们可以按照帮助主题在内容列表中出现的顺序来浏览所有的主题。使用左右箭头可按照访问的顺序浏览帮助主题。

使用 C++的 ArcGIS 开发

将要出现的章节打算给使用 C++的开发人员介绍使用 ArcGIS Engine 开发。ArcGIS EngineDeveloper Kit 让开发人员访问 ArcObjects，并被设计为满足 C++开发工程的需求，这些工程需要 ArcObjects 而不需要 ArcGIS Desktop 应用程序。

理解 COM 和 ArcObjects

ArcGIS 的 C++ API 让 C++开发人员访问用于构建 ArcGIS 家庭产品的 ArcObjects 组件，这些组件自身就采用 COM 框架用 C++编写。不必为使用 C++ API 编写任何的 COM 代码。为了使用 ArcObjects，对 COM 对象如何工作有个基本理解是必要的。如果你还不太熟悉 ArcObjects 框架,推荐阅读这章最初的两节“微软组件对象模型”和“使用 ArcObjects 开发”。

获取帮助

除了 Start Menu -> Programs 快捷方式外，ArcGIS Developer Help 的 C++版本也能从 ArcGIS Engine 安装目录中获取到。通过浏览至 <ArcGIS install directory>\DeveloperKit\Help\COM 打开该目录并双击 ArcGISDevHelpVC.chm。

关于在这种 API 中用到的对象的帮助（接口、类等）， 参阅与 ArcGIS Engine 一起安装的 ArcGIS Developer Help for C++部分。如果接受的是缺省安装选项，可以从 Start Menu ->Programs -> ArcGIS -> Developer Help -> C++ Help 中访问该帮助。

支持的编译器

ArcGIS C++ API 支持 Visual Studio 6.0 和 Visual Studio.NET 2003 的编译器。

开发环境

作为 ArcGIS C++ API 开发人员，可以选择任意开发环境。不过，如果希望使用集成开发环境(IDE),Visual Studio6.0 和 Visual Studio.NET 2003（7.1）都是推荐的选择。如果你不想用 IDE，可以在文本编辑器中编写自己的代码，并从命令行编译它。如果选择这个选项，推荐使用 Windows 命令提示的 nmake 应用程序。对于开发环境的选择完全取决于个人偏好和工具的可用性。下面几节集中在这三种选择上。

也可以在一个开发环境下的混合和匹配代码，并在另一个开发环境下编译。例如，你可以在 Visual Studio 中编写代码，但通过利用命令行工具的脚本来编译和构建。

在 Visual Studio 6.0 中使用 C++进行 ArcGIS 开发

建立自己的应用程序

在 Visual Srudio 6.0 中开始创建 ArcGIS Engine 应用程序，启动 Microsoft Visual C++并使用 Win32 Console Application 向导创建一个空的工程。选择 File>New，选择 Win32 Console Application，输入工程名称和选择工程存放位置。

ArcSDK.h 和 ArcGIS Engine olb 文件缺省情况下分别被安装到 <ArcGIS install directory>\include\CPPAPI 和 <ArcGIS install directory>\com 文件夹

接下来，设置必须的工程选项。在 Porject Menu>Setting 对话框的 C/C++选项卡上，从 Category 组合框中选择“Preprocessor”。在 Additional Include Directories 文本框中键入 ArcSDK.h 路径。另外，输入 ArcGIS Engine olb 文件的位置。不要忘了用分号隔开两个路径。然后，进入 Preprocessor Definition 文本框并输入“ESRI_WINDOWS”来定义 ESRI_WINDOWS 符号。与目录相似，符号也需要用逗号隔开。

推荐使用/GX 和/NOLOGO 编译器标识。/GX 能进行同步异常处理；/NOLOGO 防止显示编译器启动标语和通知的编译器信息。通过打开工程 Property Pages 来激活/GX，移动至 C/C++ ->C++ language，并选中‘Enable exception handling’复选框。通过从菜单条 Project->Setting ->C/C++ tab ->Customize category，选中‘Suppress Starup Banner and Information Message’选项来设置/NOLOGO。

最后，到 Project > Add a Project > New 中给工程添加一些文件；这些最终包含用户的代码。选择想添加的文件类型并给它一个名称。向工程中添加应用程序需要的所有文件。现在可以准备编写代码了。不要忘了包含 ArcSDK.h 头文件。

编译工程

要在 Visual Studio 6.0 中编译 ArcGIS Engine 应用程序，按 F7 或选择 Build->Build YourApplicationName.exe。

运行应用程序

在 Visual Studio 6.0 中运行一个 ArcGIS Engine 命令行应用程序之前，需要设置变量。变量通过定制工程的设置加载到程序中；到 Project Menu ->Setting ->Debug 选项卡，将任意变量添加到 Program argument 文本框。

一旦添加了变量，通过选择 Build->Execute YourApplication Name.exe，或按 Ctrl +F5 来运行应用程序。要在调试模式下运行应用程序，选择 Build ->Strat Debug->Go,或按 F5。

ArcSDK.h 和 ArcGIS Engine olb 文件缺省情况下分别被安装到 <ArcGIS install directory>\include\CPPAPI 和 <ArcGIS install directory>\com 文件夹。

在 Visual Studio 6.0.NET 中使用 C++进行 ArcGIS 开发

建立应用程序

在.NET中使用C++ API编程的最简单方法是使用C/C++ Console Application向导。这个向导与Win32 控Console Project and Console Application(.NET)向导不同。要访问C/C++ Console Application向导，你必须从http://www.msdnaa.net/Resources/display.aspx?ResID=1911 安装Academic Tools for VS. NET。一旦你安装了这个向导，选择New > Project ,单击Visual C++ Project文件夹，并选择C/C++ Console Application。

现在你准备进行下一步的工程选择。首先，添加一些附加的包含目录。通过选择 Project Menu ->Properties，单击 C/C++文件夹，单击 General。在 Additional Include Directories 文本框中，输入 ArcSDK.h 文件的路径，另外输入 ArcGIS Engine olb 文件的位置。记住，用分号隔开路径。也可以单击省略号来添加新的目录。

下一步，单击 Preprocessor，在 Preprocessor Definitions 文本框中输入“ESRI_WINDOWS”来定义 ESRI_WINDOWS 符号。你也可以单击省略号来定义符号。

现在准备编写自己的代码了，不要忘记在开始包含 ArcSDK.h 头文件。

编译应用程序

在 Visual Studio .NET 2003 中选择 Build->Build Solution 来编译 ArcGIS Engine 应用程序。

运行程序

在 Visual Studio .NET 2003 中运行一个 ArcGIS Engine 命令行应用程序之前，需要设置变量。变量通过定制工程的设置加载到程序中；

到 Project Menu ->Properties ->Debugging 项，将任意变量添加到‘Command Argument’。确认在配置组合框中选中正在使用的配置。

最后，选择 Debug->Start Without Debugging 或按 Ctrl+F5 来运行应用程序。如果你想在调试模式下来运行应用程序，选择 Debug->Start 或按 F5。

使用 NMAKE 和窗口命令提示进行 ArcGIS 开发

从命令提示中，可以选择支持的编译器；首先要做的是选择其中一个并准备使用它。Visual Studio 命令行构建工具缺省是不可用的，所以需要使用一个提供的批处理文件（vcvars32.bat）来为命令行编译和运行配置 Visual Studio 编译器。

通过命令行访问 Visual Studio 6.0 编译器

Visual Studio 命令行构建工具缺省是不可用的。不过提供一个批处理文件（称作 vcvars32.bat）就可以使它们可用。vcvar32.bat 文件在每次打开一个新的命令行提示时必须运行。可以选择创建自己的批处理文件来运行 vcvars32.bat，并打开为开发准备的命令提示。每一步过程描述如下。

- 从命令行运行 vcvars32.bat
  1. 打开一个命令提示窗口，进入包含 vcvars32.bat 的文件夹。
  2. 输入 vcvars32.bat 来运行该批处理文件。
  3. 要进行开发，进入包含代码的文件夹并开始。Visual Studio 命令行构建工具将在用户的命令提示中可用。
  4. 要运行程序。运行带有任意必须参数的可执行文件。

vcvars32.bat 文件缺省位置是 Program Files\Microsoft Visual Studio\VC98\Bin(Visual Studio 6.0).

- 为用户创建一个批处理文件来运行 vcvars32.bat
  1. 指向想存储批处理文件的目录。
  2. 右键单击该目录，选择 New->Text Document.
  3. 修改文件名为以.bat 结尾（例如 cmdpromptdevel.bat），单击 Yea 来确定名称修改。
  4. 右键单击 File 并选择 Edit。
  5. 找到 vcvars.bat，右键单击该文件选择 Edit。
  6. 复制 vcvars.bat 文件中所有的文本到上面创建并打开的批处理文件中，关闭 vcvars.bat。
  7. 将下面一行加到你的批处理文件中。

     %SystemRoot%\system32\cmd.exe

     这行将打开一个命令提示。
  8. 当希望依赖于命令行开发，双击批处理文件。一个命令行提示将随着已经安装的需要的环境一起打开。你可以为批处理文件创建一个快捷方式，并将它加到开始菜单或工具条上。

vcvars32.bat 文件的缺省位置是 Program Files\Microsoft Visual Studio\VC98\Bin(Visual Studio 6.0).

9. 要进行开发。进入包含你代码的文件夹并开始。Visual Studio 命令行构建工具将在用户的命令提示中可用。
10. 要运行程序。运行带有任意必须参数的可执行文件。

通过命令行访问 Visual Studio.NET 2003 编译器

Visual Studio 命令行构建工具缺省是不可用的。不过 Visual Studio.NET 2003 包含的一个命令提示使该工具可用。要打开命令提示并访问这些工具，进入 Start Menu－>All programs->Visual Studio.NET 2003->Visual Studio .NET Tools-> Visual Studio.NET 2003 Command Prompt。

如果已被打开，提示将自动运行一个批处理文件 vcvars32.bat，使得构建工具可用。vcvars32.bat 文件的缺省地址是：Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin

建立应用程序

打开自己最喜欢的文本编辑器开始写你的代码。使用一个 Makefile 来设置下面内容，包括目录和编译器选项。

1. 使用/I 编译器选项来添加 Program Files\ArcGIS\include \CPPAPI and Program Files\ArcGIS\Com 作为附加的包含目录。
2. 使用/D编译选项定义ESRI_WINDOWS符号，用来指导编译器读取Windows支持的来自ArcSDK.h中的头文件。
3. 使用/GX编译标识激活同步异常处理。
4. 使用/UOLOGO编译标识防止显示编译启动标题和通知的编译器消息。

如果想的话，可以利用提供给 ArcGIS Engine 的 Makefile.Windows 例子。这个例子文件的细节参看下节。

如果没有安装到缺省路径，找到并使用自己的安装地址，并把它加到 \include\CPPAPI and \Com 文件夹作为包含目录。

定制 MakeFile 例子

为了方便起见，ArcGIS Engine 中包括一个名为 Makefile.Windows 的 Makefile 例子供用户使用。下面的步骤突出了文件的特定区域，为了该文件在开发过程中使用必须定制这些区域。修改是基于一个应用程序的，这个应用程序在一个单独的文件中编写，my_application..cpp，它占用一个单独的文件。

1. 在整个 makefile 中，更新程序名称（当前为‘basic_sample’）来反映用户的应用程序名称。

```
# Set up the program name
PROGRAM = my_application.exe
…
# Program name updates — source and object file lists
CPPSOURCES = my_application.cpp
CPPOBJECTS = my_application.obj
…
# Program name updates –- dependencies list
my_application.obj: my_application.cpp my_application.h
```

Makefile.Windows 模板可以在 ArcGIS Developer Help under Development Environments>C++>Makefile 下找到。

这里用来描述 Makefile 代码的注释文本已经从该文件的实际注释中修改，以反映执行的步骤。

2. 上面步骤 2 到步骤中概述的编译器选项已经设置好了，但需要完成第一步以便准备应用程序中使用的模板。

```
…
# Setting up the include directories
INCLUDEDIRS = \
/I  “C:\Program Files\ArcGIS\include\CPPAPI”  \
/I  “C:\Program Files\ArcGIS\Com”
…
# Setting up the compiler options
CPPFLAGS = /DESRI_WINDOWS $(INCLUDEDIRS) /nologo /GX
```

尽管编译器选项已经在模板中设置好了，这里包含这行以展示内置 CPPFLAGS 宏的使用。

3. 为应用程序提供依赖关系列表。

```
…
# Program name updates  –- dependencies list
my_application.obj: my_application.cpp my_application.h
```

这行在第一步中也出现过，以展示程序名称的更新。

准备好 Makefile 后，就可以开始写代码了。不要忘记以包含 ArcSDK.h 开始编写代码。

编译应用程序

一旦 Makefile.Windows 准备编译应用程序，就可以通过输入‘nmake /f Makefile.Windows’从命令行进行编译。

运行应用程序

在运行一个 ArcGIS Engine 命令行应用程序之前，必须设置命令行参数。更新 Makefile 以包括每个输入参数变量和一个运行目标。

这些修改的一个实例如下所示：

```
# Setting up the program argument
INPUT = C:\Data\inputfile
…

# Setting up a run target
run:
$(PROGRAM) $(INPUT)
```

一旦 Makefile.Windows 准备在应用程序中使用，就可以输入‘nmake /f Makefile.Windows run’从命令行运行应用程序。

ArcObjects C++实践

下面是用 ArcGIS C++ API 进行编程的一些建议。

推荐的函数用法

C++ API 为一些函数提供了其自己的实现：

- AoInitialize——在 COM 编程中要使用 AoInitialize 函数的地方使用此函数。

  extern "C" HRESULT AoInitialize(LPVOID pvReserved);

  这个函数初始化 ArcGIS Engine 和 COM。初始化必须在使用除 IAoInitialize 接口外的任何 ArcObjects 之前完成，其中 IAoInitialize 接口处理应用程序的许可问题。

- AoUninitialize——在 COM 编程中使用 AoUninitialize 函数的地方使用此函数。

  extern "C" void AoUninitialize(void);

  这个函数取消 ArcGIS Engine 和 COM 的初始化。

- AoExit——在非 ArcObjects 代码中使用 Exit 的地方使用此函数，就像在 main()中使用 return 一样。

  extern "C" VOID AoExit (int number);

  在应用程序退出之前，必须调用 AoExit。这使得要求正确清理各种 ArcGIS Engine 和 COM 元素的操作系统变得更轻便。

为简单起见，给出的代码块并不总是检查 HRESULT，但开发人员应该总要这么做。

下面的例子说明了上面讨论的 3 个函数在应用程序中应如何使用：

```
int main (int argc, char* argv[])
{
  // Initialize ArcGIS Engine and COM
  ::AoInitialize(NULL);

  // ArcGIS Engine licensing
  {
    IAoInitialize ipInit(CLSID_AoInitialize);
    esriLicenseStatus status;
    ipInit->Initialize(esriLicenseProductCodeEngine, &status);

    // ArcObjects Code here

    ipInit->Shutdown();
  }

  // Uninitialize ArcGIS Engine and COM
  ::AoUninitialize();

  // Exit the application
  AoExit(0);
}
```

注意 IAoInitialize 是限定范围的，因而在 AoUninitialize 被调用前会超出范围。

- AoCreateObject——在COM编程中使用CoCreateInstance函数的地方使用此函数。

```
extern "C" HRESULT AoCreateObject(REFCLSID rclsid,
                                  LPUNKNOWN pUnkOuter,
                                  DWORD dwClsContext,
                                  REFIID riid,
                                  LPVOID *ppv);
```

在使用智能指针的地方不需要这个函数。但是，可以使用这函数来创建一个没有智能指针的对象实例，如下面的代码所示：

```
// Create a Workspace Factory without using smart pointers
IWorkspaceFactory *pWorkspaceFactory;
hr = ::AoCreateInstance(CLSID_ShapefileWorkspaceFactory, 0,
                        CLSCTX_INPROC_SERVER,
                        IID_IWorkspaceFactory,
                        (void **)&pWorkspaceFactory);
```

- AoAllocBSTR 取代 SysAllocString

```
extern "C" BSTR AoAllocBSTR(const OLECHAR *sz);
```

- AoFreeBSTR 取代 SysFreeString

```
extern "C" void AoFreeBSTR(BSTR bstr);
```

在使用智能类型 CComBSTR 时，上面两个函数都不需要。但是，可以用它们创建和释放 BSTR，如下面的例子所示：

```
// display the feature type as "simple" or "other"
BSTR bsFeatureType;
esriFeatureType featType;
pFeatureClass->get_FeatureType(&featType);
switch (featType)
{
   case esriFTSimple :
   bsFeatureType = ::AoAllocBSTR(L"simple");
   break;
   default:
   bsFeatureType = ::AoAllocBSTR(L"other");
}
std::wcerr << L"Feature Type : " << (BSTR) bsFeatureType << std::endl;
::AoFreeBSTR(bsFeatureType);
```

从 VARIANT 类型获取一个要素字段值

一个要素字段值作为 VARIANT 类型被返回，要求作一些处理以获得实际值。下面的例子循环一个要素的所有字段并打印出每个字段的要素值。这里显示的代码只处理了某些字段类型(例如，2 字节整型、4 字节整型和 BSTR 字符串)；也可以根据应用程序的需要选择处理其他类型。

```
// ipFeature is of type IFeaturePtr, and we assume it has already
// been declared and instantiated above
IFieldsPtr ipFields;
```

```
hr = ipFeature->get_Fields(&ipFields);
long fieldCount;
hr = ipFields->get_FieldCount(&fieldCount);
IFieldPtr ipField;
CComVariant fieldValue;

for (long i=0; i<fieldCount; i++)
{
   hr = ipFields->get_Field(i, &ipField);
   hr = ipFeature->get_Value(i, &fieldValue);

   // Get field’s value based on its type
   switch (fieldValue.vt)
   {
   case VT_I2:
      std::cerr << fieldValue.iVal << std::endl;
      break;

   case VT_I4:
      std::cerr << fieldValue.lVal << std::endl;
      break;

   case VT_R4:
      std::cerr << fieldValue.fltVal << std::endl;
      break;

   case VT_R8:
      std::cerr << fieldValue.dblVal << std::endl;
      break;

   case VT_BSTR:
      std::wcerr << fieldValue.bstrVal << std::endl;
      break;

   default:
      std::wcerr <<  “Field type not supported.\n”;
      break;
   }
}
```

在声明智能指针后协同创建一个具有智能指针引用的对象

在声明一个智能指针的同时，通常协同创建了一个类：

```
IFooPtr ipFoo(CLSID_Foo);
```

但是，在某些情况下，可能需要首先声明智能指针，然后再协同创建类。这可以通过下面的方式来完成。注意‘dot’成员选择操作符的使用，与之相对的是通常用于智能指针类型的箭头成员选择操作符：

```
IFooPtr ipFoo;
// more code would be here
ipFoo.CreateInstance(CLSID_Foo);
```

内嵌接口查询（QI）

可以使用内置的智能类型来QI已有其一个接口的组件对象类支持的其它接口：

```
// ipTin is of type ITinPtr and is an interface to an instance of
// the Tin coclass
// The Tin coclass supports both the ITin and ITinSurface interfaces
// GetVolume is a method on the ITinSurface interface
((ITinSurfacePtr) ipTin)->GetVolume(…);
```

函数签名中的裸指针（Raw pointers）

不在函数签名中拥有智能指针，可以考虑使用一个裸指针以节省在函数之上调用一个智能指针构造器/销毁器的开销。仍可以将智能指针对象传递给函数，因为每个智能指针都有一个能返回底层裸指针的重载指针操作符。下面的例子对这进行了说明：

```
HRESULT DoRasterOp(IRaster* pRaster); // Function dec: raw pointer
IRasterPtr ipRaster;

HRESULT hr = DoRasterOp(ipRaster); // Pass in smart pointer
```

从函数中返回ArcObjects

这个技巧建立在前一个技巧之上。在本案例中，裸指针用在函数声明中，并为返回的对象使用一个间接的双精度。这允许改变所传递指针的指向。然后，用希望返回的值初始化一个智能指针对象，并将它分配给已传入的指针。

```
HRESULT GetTinWorkspace(char* path, ITinWorkspace**
ppTinWorkspace)
{
    HRESULT hr = S_OK;
    IWorkspaceFactoryPtr
    ipWorkspaceFactory(CLSID_TinWorkspaceFactory);
    IWorkspacePtr ipWork;
    hr = ipWorkspaceFactory->OpenFromFile(CComBSTR(path), 0,
    &ipWork);
    if (FAILED(hr) || ipWork == 0)
        return E_FAIL;
    // Initialize ipTinWorkspace with ipWork
    ITinWorkspacePtr ipTinWorkspace(ipWork);
    *ppTinWorkspace = ipTinWorkspace;
    // AddRef() if the assignment worked
    if (*ppTinWorkspace)
        (*ppTinWorkspace)->AddRef();
    return hr;
}
```

注意对AddRef的调用。这要求保证合适的管理资源。

IUID

要了解有关 GUID 的更多信息，请参阅本章中的“微软组件对象模型”一节。

ArcObjects 中有一些方法以 IUID 对象作为参数。一个 IUID 对象是一个全局唯一标识符对象。它可以是一个 GUID，如下面例子所示，或者是一个 ProgID。

```
IUIDPtr ipUID(CLSID_UID);
IEnumLayerPtr ipEnumLayer;

// use IGeoFeatureLayer’s GUID
hr = ipUID->put_Value(CComVariant(L”{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}”));
hr = ipMap->get_Layers(ipUID, VARIANT_TRUE, &ipEnumLayer);
```

复制 instanceof（Java）或 TypeOf（Visual Basic）功能

一个接口指针可以指向多个组件对象类中的一个是很常见的。可以通过尝试用 if/else 逻辑查询（QI）其它接口来获得有关组件对象类的更多信息。例如，RasterDataset 和 FeatureDataset 组件对象类都实现了 Idataset 接口。如果将 IDataset 作为函数参数来传递，可以像下面那样来决定 IDataset 引用哪个组件对象：

```
void Foo (IDataset *pDataset)
{
   IFeatureDatasetPtr ipFeatureDataset(ipDataset);
   if (ipFeatureDataset != 0)
   {
      // use IFeatureDataset methods
   }
   else
   {
      IRasterDataset2Ptr ipRasterDataset(ipDataset);
      if (ipRasterDataset!= 0)
      {
         // use IRasterDataset2 methods
      }
   }
}
```

ESRI System 接口

ArcGIS Engine 中的 System 库（用 ArcSDK.h 来包括）包含许多可以简化编程的接口。它包含为其它 ArcGIS 类库提供服务的组件。

脱离 STL 的接口中的对象组

ArcObjects 包含一些用于管理对象组的接口。尽管这里不会讨论所有这些接口，但给出的例子说明了这些接口可以极大简化 C++中 COM 对象的使用。标准模板库（STL）存在的类型与这些接口中的一些很相似；但这些接口通常比它们的 STL 对应物更简单，并已经准备好与 COM 对象一起使用。要了解下面任何接口方面更详细的信息，请参阅 ArcGIS C++开发帮助。

IArray

IArray 接口提供访问控制简单对象数组的成员。有许多相关联的接口，诸如 IDoubleArray 和 IVariantArray 等。下面的代码块展示了怎样将一个 FeatureCursor 中要素的几何图形添加到一个数组中。

```
// ipFeatureLayer is of type IFeatureLayerPtr and ipQueryFilter is of
// type IQueryFilterPtr. Both have already been declared and
// instantiated.
IArrayPtr ipCacheArray (CLSID_Array);
IFeatureCursorPtr ipFeatureCursor;
ipFeatureLayer->Search(ipQueryFilter, VARIANT_FALSE,
&ipFeatureCursor);

IFeaturePtr ipFeature;
while (ipFeatureCursor->NextFeature(&ipFeature) == S_OK)
{
   IGeometryPtr ipGeom;
   ipFeature->get_ShapeCopy(&ipGeom);
   ipCacheArray->Add((IUnknownPtr) ipGeom);
}
```

ISet

ISet 接口提供访问控制唯一对象简单集合的成员。例如，下面的代码块循环地图图层并试图将没有编辑过的所有要素类工作空间添加到一个集合中。

```
// ipMap is of type IMapPtr and was previously
// declared and instantiated
ISetPtr ipSet(CLSID_Set);
ILayerPtr ipLayer;
IFeatureLayerPtr ipFeatLayer;
IFeatureClassPtr ipFeatClass;
IDatasetPtr ipDataset;
IWorkspacePtr ipWorkspace;
IWorkspaceEditPtr ipWorkspaceEdit;
long layerCount;
hr = ipMap->get_LayerCount(&layerCount);
for (long i=0; i<layerCount; i++)
{
   hr = ipMap->get_Layer(i, &ipLayer);
   ipFeatLayer = ipLayer;
   // layer might not be a feature layer
   if (ipFeatLayer == 0 || FAILED(hr)) continue;
   hr = ipFeatLayer->get_FeatureClass(&ipFeatClass);
   // layer could reference bogus data
   if (ipFeatClass == 0 || FAILED(hr)) continue;
   ipDataset = ipFeatClass;
   hr = ipDataset->get_Workspace (&ipWorkspace);
   ipWorkspaceEdit = ipWorkspace;
```

```
        // some data are not editable
        if (ipWorkspaceEdit == 0 || FAILED(hr)) continue;
        VARIANT_BOOL beingEdited;
        hr = ipWorkspaceEdit->IsBeingEdited(&beingEdited);
        if (!beingEdited)
        {
        // only adds unique workspaces
        hr = ipSet->Add(ipWorkspace);
        }
    }
```

复制对象

可能注意到这个例子和情景没有遵循这里概述的 Good Error Handling 惯例。这样做只是为了增加代码的可读性，因为错误检测不是这段代码的重点。

IClone 接口在比较和复制对象方面非常有用，可以节省时间和计算资源。许多组件对象类支持 IClone 接口。请参阅 ArcGIS 开发帮助中的 IClone 文档以了解更详细的信息。下面的代码块复制了一个 Point 对象：

```
// ipMouseClickPoint is of type IPointPtr and was previously declared //
and instantiated.
IClonePtr ipClone (ipMouseClickPoint);
IClonePtr ipCloned;
ipClone->Clone(&ipCloned);
```

错误处理

COM 方法返回一个 HRESULT 来指示方法调用的成功或失败，这在本章前面的‘用 ArcObjects 进行开发’一节中已经讨论过。在用 C++ API 进行编程时，应该检查所有 COM 对象调用返回的 HRESULT。

下面是几个可能返回的常见 HRESULT：

- S_OK 表示成功
- E_FAIL 表示失败
- E_NOTIMPL 表示某个方法没有实现

有一些宏可以用来测试返回的 HRESULT。

- bool FAILED(HRESULT)

  例如，如果打开一个想在其中处理数据的工作空间失败，将不能使用这些数据。此时，应退出应用程序以避免随后的崩溃。

```
// Open the workspace
IWorkspaceFactoryPtr
ipWorkspaceFactory(CLSID_RasterWorkspaceFactory);
IWorkspacePtr ipWorkspace;
HRESULT hr = ipWorkspaceFactory->OpenFromFile(inPath, 0,
&ipWorkspace);
if (FAILED(hr) || ipWorkspace == 0)
{
   std::cerr <<  “Could not open the workspace.”  << std::endl;
   return E_FAIL;
}
```

- bool SUCCEEDED(HRESULT)

例如，如果要创建一个新的栅格数据集，必须首先知道没有同名的数据集存在。为了查找是否存在同名的数据集，试着打开它。如果成功，就知道不能用该名字创建一个新数据集。

```
// Check for existence of a dataset with the desired output name.
// If such exists, we can't create a new one with the name.
IRasterDatasetPtr ipExistsCheck;
hr = ipRastWork->OpenRasterDataset(outFile, &ipExistsCheck);
if (SUCCEEDED(hr))
{
   std::cerr << "A dataset with the output name already exists!" <<
std::endl;
   return E_FAIL;
}
```

解决故障

- 不能打开包含文件“ArcSDK.h”——如果例子因为不能打开ArcSDK.h 而不能编译的话，确保提供了正确的包含路径：ArcGIS\include\CPPAPI。如果路径正确并且仍不能运行，确保文件在该目录中。
- 不能打开类库文件“esriSysten.olb”——如果例子因为不能打开ESRI OLB 文件而不能编译的话，确保提供了正确的包含路径：ArcGIS\Com。如果路径正确并且仍不能运行，确保文件在该目录中。
- 如果例子或自己的代码在编译但不能运行，确保已经注册了ArcGIS Engine。使用 SoftwareAuthorization 工具，可以在 Start > All Programs > ArcGIS 中找到该工具，并检查当前配置。如果配置没有列出“standardengine”，按照提示运行该工具。
- 错误：请定义 ESRI_WINDOWS 或 ESRI_UNIX——需要通知编译器要使用哪个系列的头文件。用户忘了定义 ESRI_WINDOWS 符号。

缺陷

当使用 C++ API 时，只支持 Windows C++命令行应用程序。但是，GUI 应用程序可以用 COM API（包括 Visual C++）来建立，而且一些 ActiveX 控件向独立的 GUI 应用程序提供了 GIS 功能。要了解有关这方面的细节，请参阅本章前面的“Visual C++”一节。

5 许可与部署

ArcGIS Engine 应用程序的开发离不开最终应用程序的部署问题。应用程序的部署涉及到三个独立的过程：应用程序许可的初始化、ArcGIS Engine 运行时软件的安装及其使用授权。本章将详细阐述这些过程，并分析开发人员在分发其 ArcGIS Engine 应用程序之前必须做的部署决策。

ArcGIS 许可选项

ArcGIS Engine 开发人员要做的最重要决定之一就是确定预期应用程序的功能需求及满足这些需求所需的最小化许可级别。在大多数情况下，开发人员不能有效地开发需要 ArcGIS Engine 运行时 3D、空间和地理数据库选项的应用程序，正是因为开发人员没有正确地评估客户或组织的真正需要并做出相应的计划。

本章的这一部分详细阐述各种可能的许可选项。

独立可执行许可的初始化

ArcGIS Engine 开发人员要做的最重要决定之一就是确定预期应用程序的功能需求及满足这些需求所需的最小化许可级别。在大多数情况下，开发人员不能有效地开发需要 ArcGIS Engine 运行时 3D、空间和地理数据库选项的应用程序，正是因为开发人员没有正确地评估客户或组织的真正需要并做出相应的计划。

第六章“开发情景”中的各个示例应用程序也说明了许可初始化过程。

本章这节详细阐述用多个层次的许可初始化应用程序的各种可能选项。首先详细讨论这些选项，然后提供许多许可初始化情况的示例。

许可初始化

用 ArcObjects 开发的每个独立应用程序必须用一个合适的许可进行初始化，以保证在安装了该应用程序的任何机器上成功地运行应用程序。许可初始化必须由应用程序在启动和访问任何 ArcObjects 前执行。许可初始化失败会导致应用程序错误。

虽然每个独立的应用程序必须初始化，但下列实例说明了不是独立应用程序因而不需要进行初始化的情况：

应用程序为一个 DLL，它将集成到另一个能执行许可配置的应用程序中。

应用程序是 ArcMap 或第三方应用程序的扩展。扩展程序负责许可管理。

初始化应用程序时要考虑两类许可：产品许可和扩展许可（如果应用程序使用 ArcGIS 的任何扩展模块的话）。每种许可都是 Engine 单用户（Engine Single Use）、Desktop 多用户（Desktop Concurrent Use）和 Desktop 单用户（Desktop Single Use）许可中的一种。

- Engine 单用户（Engine Single Use）—提供访问 ArcGIS Engine 或具有地理数据库编辑功能的 Engine 许可。
- Desktop 多用户（Desktop Concurrent Use）—FlexLM 技术用于提供对 ArcGIS Desktop 产品 ArcView，ArcEditor 和 ArcInfo 及其扩展进行同步访问的许可。多台机器可以访问该许可；许可被存储在一个许可管理器（license manager）上，使用许可时对许可进行核对。
- Desktop 单用户（Desktop Single Use）—提供访问单用户 ArcView，ArcEditor 和 ArcInfo 的许可。与 Engine 单用户（Engine Single Use）相似，每个许可只能由安装了该许可的机器访问。尽管与 Desktop 多用户（Desktop Concurrent Use）许可有很显著的差异，它们实际上使用的是相同的技术。这意味着 ArcGIS Engine 开发人员无法区分开 Desktop 单用户许可和多用户许可，因此应该将它们看作相同的许可。

不使用 ArcGIS 扩展的应用程序开发要考虑的问题

- 一旦应用程序用某个许可进行了初始化，就不能再进行重新初始化了；应用程序的许可初始化针对的是其整个生命期。在用许可初始化应用程序时必须考虑以下问题：

应用程序运行所需的产品许可类型。例如，企业地理数据库编辑应用程序用 ArcGIS Engine 许可或 ArcView 许可不能运行，而用 ArcGIS Engine 的地理数据库编辑许可、ArcEditor 许可或 ArcInfo 许可则可以运行。

- 应用程序可获得的产品许可类型。例如，用 ArcGIS Engine 许可可以运行的应用程序也可以用 ArcView、ArcEditor 和 ArcInfo 许可运行。然而，用户可能并不想为这种应用程序购买 ArcInfo 许可。

在 ArcGIS Engine 应用程序上使用 ArcView 许可可以让用户访问标准 Engine 许可能获得的所有功能。同样地，在具有地理数据库编辑功能的 ArcGIS Engine 应用程序上使用 ArcEditor 许可可以让用户访问具有地理编辑选项的 Engine 许可能获得的所有功能。

使用 ArcGIS 扩展的应用程序需要考虑的其他问题

当用某个特定的产品许可初始化应用程序时，就建立了一个到许可服务器（license server）上的连接。随后对扩展模块的所有检出和检入调用都针对这个许可服务器。这样，用户就不能组合来自不同许可服务器上的许可或组合单用户 Engine 许可。

- 如果应用程序用 Desktop 多用户许可进行初始化，则该应用程序以后只能访问那个 Desktop 多用户许可服务器及其扩展许可。
- 如果应用程序用 Desktop 单用户许可进行初始化，则该应用程序以后只能访问该 Desktop 单用户许可服务器及其扩展许可。
- 如果应用程序用 Engine 单用户许可进行初始化，则该应用程序以后只能访问 Engine 单用户扩展许可。

在执行初始化之前可以查询许可服务器（Desktop 多用户许可或 Desktop 单用户许可）和 Engine 单用户许可，以查看是否可以获得用户所需的许可。如果用户需要的所有许可都可以用 Engine 单用户许可获得，则建议使用 Engine 单用户许可，而不是 Desktop 多用户许可和 Desktop 单用户许可。这意味着开发人员不会限制任何其他用户获得 Desktop 多用户许可。

用 Engine 单用户许可可以访问下列扩展模块：

对于使用 ArcGIS 控件开发的应用程序，表 1 描述了每个控件运行时所需的许可。

- 3D Analyst
- Spatial Anslyst
- StreetMap

	ArcGIS Engine 单用户许可	带有 Engine 单用户 3D 分析扩展模块的 ArcGIS Engine 单用户许可	ArcView, ArcEditor, ArcInfo 单用户或桌面多用户许可	带有 3D 分析扩展模块的 ArcView, ArcEditor, ArcInfo 单用户或桌面多用户许可	Arc Reader
ArcReaderControl	√				√
MapControl	√		√		
PageLayoutControl	√		√		
ReaderControl	√		√		
ToolbarControl	√		√		
TOCControl	√		√		
GlobeControl		√		√	
SceneControl		√		√	

对于使用 ArcGIS 控件开发的应用程序，表 2 描述了每个控件设计时所需的许可。

Engine 单用户设计选项（ Single Use Designer Option）

使用 ArcObjects 开发应用程序时需要 Engine Single Use Designer 扩展。这个许可与其它扩展的工作方式的不同之处在于：

- 该扩展只在设计和开发应用程序时需要，在运行时从来不需要。
- 该扩展不需要人为检出，它会自动检出。

	ArcGIS Engine 单用户许可	带有 Engine 单用户设计扩展模块的 Engine 单用户许可	带有 Engine 单用户设计和 3D 分析扩展模块的 Engine 单用户许可	ArcView, ArcEditor, ArcInfo 单用户或桌面多用户许可	带有 Publisher 扩展模块的 ArcView, ArcEditor, ArcInfo 单用户或桌面多用户许可
ArcReaderControl					√
MapControl	√			√	
PageLayoutControl	√			√	
ReaderControl		√			
ToolbarControl		√			
TOCControl		√	√		
GlobeControl			√		
SceneControl		√		√	

用许可初始化应用程序

用许可初始化应用程序必须按如下顺序执行：

1. 检查产品许可的可用性。
2. 检查扩展许可的可用性（如果需要的话）。
3. 用产品许可初始化应用程序。
4. 如果需要的话，执行扩展模块的检出与检入。
5. 关闭应用程序。

检查产品许可的可用性

所选的产品许可决定了应用程序可以访问的功能。一旦产品许可被初始化，在应用程序的生命期内就不能改变。

- 如果用户需要的产品没有得到许可，可以选择用更高级的产品许可初始化应用程序。
- 如果没有可用的合适产品许可，应用程序应该将这个问题告知用户，让用户解决许可问题，或者退出应用程序。

检查扩展许可的可用性

如果应用程序设计要使用扩展功能，在初始化应用程序前，要检查扩展许可的可用性。扩展许可的可用性检查必须与应用程序最终用来初始化的产品许可的检查一起完成，因为并不是每个产品许可都可以获得各个扩展许可。

- 如果应用程序成功运行所需的扩展不可用，应用程序应将这个问题通知给用户并退出应用程序。
- 如果扩展功能并不是应用程序运行所必需的，而且该扩展许可不可用，则应用程序应使依赖于该扩展的功能对用户无效。

初始化应用程序

一旦已经确定了可用的合适产品和扩展许可，应用程序就可以用该产品许可进行初始化。一旦进行了初始化，就不能对该应用程序重新进行初始化。

扩展模块的检入与检出

当应用程序需要扩展功能时，可以检出扩展模块，一旦应用程序使用完扩展功能就可以将其检入，或者在应用程序初始化后直接检出扩展模块并在关闭前检入。扩展模块检入和检出的方式取决于应用程序初始化所使用的产品许可类型。

- 如果应用程序用 Engine 单用户许可进行初始化，则应用程序使用的任何扩展模块也使用 Engine 单用户许可。这样任何扩展模块可以在应用程序初始化后直接检出而在关闭前检入。
- 如果应用程序用许可服务器进行初始化，而且应用程序成功运行需要扩展模块，则扩展模块应该在应用程序初始化后直接检出并在关闭前检入。

- 如果应用程序用许可服务器进行初始化，而且扩展功能并不是应用程序运行所必需的，则扩展模块可以在应用程序初始化后直接检出，或者当需要扩展功能时检出。当扩展模块检入时，应该使扩展功能无效。

关闭

在关闭应用程序前，必须关闭 AoInitialize 对象。这样可以保证使用过的任何 ESRI 类库被卸载。

许可初始化失败

如果产品或扩展模块检出失败，许可状态会指出失败的原因。许可检出失败的原因有：

- 产品没有得到许可。
- 许可不可用，因为该许可正被使用（只适合 Desktop 多用户许可）。
- 由于系统管理问题导致的不可测的许可失败。
- 许可已被初始化。用产品许可进行的应用程序初始化对其整个生命期有效。可以检查应用程序用何种产品许可进行了初始化。例如，如果包含企业地理数据库编辑功能的应用程序用具有地理数据库编辑功能的 Engine 单用户许可或 ArcEditor 或 ArcInfo 许可，则可以使用该编辑功能。然而，如果应用程序用 Engine 单用户许可或 ArcView 许可，则必须使该编辑功能无效。

示例 1

应用程序需要最小化的 ArcGIS Engine 许可。如果 ArcGIS Engine 许可不可用，应用程序可以用 ArcView 或 ArcEditor 许可，但不会用 ArcInfo 许可（ArcInfo 能提供用户需要的所有功能，但用户不想为这样简单的应用程序购买 ArcInfo 许可）。成功运行应用程序还需要 3D 分析和空间分析扩展功能，因此在应用程序的生命期中需要检出这两个扩展模块。用许可初始化该应用程序需要以下步骤。

能否用 ArcGIS Engine 产品许可初始化应用程序？

1. 检查 ArcGIS Engine 产品许可是否可用。
2. 检查用 ArcGIS Engine 产品许可是否可以获得 3D 分析扩展许可？
3. 检查用 ArcGIS Engine 产品许可是否可以获得空间分析扩展许可？

如果这些许可不可用，则应用程序不能用 ArcGIS Engine 许可初始化。那么是否可以用 ArcView 产品许可初始化应用程序呢？

如果这些许可都可用，则初始化应用程序。

4. 通过初始化应用程序将 ArcGIS Engine 产品许可检出。

如果该许可检出失败，那么是否可以用 ArcView 产品许可初始化应用程序呢？

如果能检出该许可，则检出扩展许可。

5. 检出 3D 分析扩展模块。

6. 检出空间分析扩展模块。

如果这两个扩展许可中的一个检出失败，则应用程序不能运行。

如果这两个扩展许可被检出，应用程序已经成功地配置好了许可。

能否用 ArcView 产品许可初始化应用程序？

1. 检查 ArcView 产品许可是否可用。

2. 检查用 ArcView 产品许可是否可以获得 3D 分析扩展许可？

3. 检查用 ArcView 产品许可是否可以获得空间分析扩展许可？

如果这些许可不可用，则应用程序不能用 ArcView 许可初始化。那么是否可以用 ArcEditor 产品许可初始化应用程序呢？

如果这些许可都可用，则初始化应用程序。

4. 通过初始化应用程序将 ArcView 产品许可检出。

如果该许可检出失败，那么是否可以用 ArcEditor 产品许可初始化应用程序呢？

如果能检出该许可，则检出扩展许可。

5. 检出 3D 分析扩展模块。

6. 检出空间分析扩展模块。

如果这两个扩展许可中的一个检出失败，则应用程序不能运行。在这里检出 Desktop 多用户许可可能会失败，因为该许可在检查其可用性后可能已被另一个应用程序检出。

如果这两个扩展许可被检出，应用程序已经成功地配置好了许可。

能否用 ArcEditor 产品许可初始化应用程序？

1. 检查 ArcEditor 产品许可是否可用。

2. 检查用 ArcEditor 产品许可是否可以获得 3D 分析扩展许可？

3. 检查用 ArcEditor 产品许可是否可以获得控件分析扩展许可？

如果这些许可不可用，则应用程序不能用 ArcEditor 许可初始化，且应用程序不能运行。

如果这些许可都可用，则初始化应用程序。

4. 通过初始化应用程序将 ArcEditor 产品许可检出。

如果该许可检出失败，则应用程序不能运行。

如果能检出该许可，则检出扩展许可。

5. 检出 3D 分析扩展模块。

6. 检出空间分析扩展模块。

如果这两个扩展许可中的一个检出失败，则应用程序不能运行。在这里检出 Desktop 多用户许可可能会失败，因为该许可在检查其可用性后可能已被另一个应用程序检出。

如果这两个扩展许可被检出，应用程序已经成功地配置好了许可。

示例 2

应用程序为一个企业地理数据库编辑应用程序，因此需要具有地理数据库编辑许可的最小化 ArcGIS Engine。如果具有地理数据库编辑许可的 ArcGIS Engine 不可用，应用程序可以用 ArcEditor 或 ArcInfo 许可。成功运行应用程序需要空间分析扩展功能，因此在应用程序的生命期内要检出该扩展模块。用许可初始化该应用程序需要以下步骤。

能否用具有地理数据库编辑功能的 ArcGIS Engine 产品许可初始化应用程序？

1. 检查具有地理数据库编辑功能的 ArcGIS Engine 产品许可是否可用。
2. 检查用具有地理数据库编辑功能的 ArcGIS Engine 产品许可是否可以获得空间分析扩展许可。

如果这两个许可中的任何一个不可用，则不能用具有地理数据库编辑功能的 ArcGIS Engine 产品许可初始化应用程序。那么能否用 ArcEditor 产品许可初始化应用程序呢？

如果这两个许可都可用，则初始化该应用程序。

3. 通过初始化应用程序检出具有地理数据库编辑功能的 ArcGIS Engine 产品许可。

如果该许可检出失败，那么能否用 ArcEditor 产品许可初始化应用程序呢？

如果能检出该许可，则将扩展许可检出。

4. 检出空间分析扩展许可。

如果该扩展许可检出失败，应用程序不能运行。在这里检出 Desktop 多用户许可可能会失败，因为该许可在检查其可用性后可能已被另一个应用程序检出。

如果这两个扩展许可被检出，应用程序已经成功地配置好了许可。

能否用 ArcEditor 产品许可初始化应用程序？

1. 检查 ArcEditor 产品许可是否可用。

2. 检查 ArcEditor 产品许可是否可以获得空间分析扩展许可。

如果这两个许可中的任何一个不可用，则不能用 ArcEditor 产品许可初始化应用程序。那么能否用 ArcInfo 产品许可初始化应用程序呢？

如果这两个许可都可用，则初始化该应用程序。

3. 通过初始化应用程序检出 ArcEditor 产品许可。

如果该许可检出失败，那么能否用 ArcInfo 产品许可初始化应用程序呢？

如果能检出该许可，则将扩展许可检出。

4. 检出空间分析扩展许可。

如果该扩展许可检出失败，应用程序不能运行。在这里检出 Desktop 多用户许可可能会失败，因为该许可在检查其可用性后可能已被另一个应用程序检出。

如果这两个扩展许可被检出，应用程序已经成功地配置好了许可。

能否用 ArcInfo 产品许可初始化应用程序？

1. 检查 ArcInfo 产品许可是否可用。

2. 检查 ArcInfo 产品许可是否可以获得空间分析扩展许可。

如果这两个许可中的任何一个不可用，则不能用 ArcInfo 产品许可初始化应用程序，而且应用程序不能运行。

如果这两个许可都可用，则初始化该应用程序。

3. 通过初始化应用程序检出 ArcInfo 产品许可。

如果该许可检出失败，则应用程序不能运行。

如果能检出该许可，则将扩展许可检出。

4. 检出空间分析扩展许可。

如果该扩展许可检出失败，应用程序不能运行。在这里检出 Desktop 多用户许可可能会失败，因为该许可在检查其可用性后可能已被另一个应用程序检出。

如果这两个扩展许可被检出，应用程序已经成功地配置好了许可。

示例 3

应用程序需要最小化的 ArcGIS Engine 许可。如果 ArcGIS Engine 许可不可用，应用程序可以用 ArcView，ArcEditor 或 ArcInfo 许可运行。应用程序需要 3D 分析扩展功能，但并不是应用程序成功运行所必需的，因此需要时将检出该扩展模块。

能否用 ArcGIS Engine 产品许可初始化应用程序？

1. 检查 ArcGIS Engine 产品许可是否可用。

2. 检查用 ArcGIS Engine 产品许可是否可以获得 3D 分析扩展许可。

如果这两个许可中的任何一个不可用，则不能用 ArcGIS Engine 产品许可初始化应用程序。那么能否用 ArcView 产品许可初始化应用程序呢？

如果这两个许可都可用，则初始化该应用程序。

3. 通过初始化应用程序检出 ArcGIS Engine 产品许可。

如果该许可检出失败，则应用程序不能运行。那么能否用 ArcView 产品许可初始化应用程序呢？

如果能检出该许可，应用程序已经成功地配置好了许可，但应该使 3D 分析扩展模块功能无效。

能否用 ArcView 产品许可初始化应用程序？

1. 检查 ArcView 产品许可是否可用。

2. 检查用 ArcView 产品许可是否可以获得 3D 分析扩展许可。

如果这两个许可中的任何一个不可用，则不能用 ArcView 产品许可初始化应用程序。那么能否用 ArcEditor 产品许可初始化应用程序呢？

如果这两个许可都可用，则初始化该应用程序。

3．通过初始化应用程序检出 ArcView 产品许可。

如果该许可检出失败，则应用程序不能运行。那么能否用 ArcEditor 产品许可初始化应用程序呢？

如果能检出该许可，应用程序已经成功地配置好了许可，但应该使 3D 分析扩展模块功能无效。

能否用 ArcEditor 产品许可初始化应用程序？

1. 检查 ArcEditor 产品许可是否可用。

2. 检查用 ArcEditor 产品许可是否可以获得 3D 分析扩展许可。

如果这两个许可中的任何一个不可用，则不能用 ArcEditor 产品许可初始化应用程序。那么能否用 ArcInfo 产品许可初始化应用程序呢？

如果这两个许可都可用，则初始化该应用程序。

3． 通过初始化应用程序检出 ArcEditor 产品许可。

如果该许可检出失败，则应用程序不能运行。那么能否用 ArcInfo 产品许可初始化应用程序呢？

如果能检出该许可，应用程序已经成功地配置好了许可，但应该使 3D 分析扩展模块功能无效。

能否用 ArcInfo 产品许可初始化应用程序？

1． 检查 ArcInfo 产品许可是否可用。

2． 检查用 ArcInfo 产品许可是否可以获得 3D 分析扩展许可。

如果这两个许可中的任何一个不可用，则不能用 ArcInfo 产品许可初始化应用程序。

如果这两个许可都可用，则初始化该应用程序。

3． 通过初始化应用程序检出 ArcInfo 产品许可。

如果该许可检出失败，则应用程序不能运行。

如果能检出该许可，应用程序已经成功地配置好了许可，但应该使 3D 分析扩展模块功能无效。

使用 3D 分析功能

1． 检查 3D 分析扩展许可是否已经被检出。

如果该许可已被检出，则应用程序可以使用 3D 分析功能。

如果该许可还没有检出该许可，则检出该许可。

2． 检出 3D 分析扩展许可。

如果检出该许可失败，则应用程序不能使用 3D 分析功能。在这里检出 Desktop 多用户许可可能会失败，因为该许可在检查其可用性后可能已被另一个应用程序检出。

如果该许可被检出，则应用程序可以使用 3D 分析功能。

部署 ArcGIS Engine 运行时

Microsoft Windows 操作系统上可以对 ArcGIS Engine 运行时安装程序进行部署。

下面首先简要介绍用于创建安装程序的技术，然后探讨 Windows 上 ArcGIS Engine 运行时安装程序的各种部署方法。

什么是 ArcGIS Engine 运行时安装程序？

使用 Microsoft Windows Installer 技术创建 ArcGIS Engine 运行时安装程序。这个技术使用一个打包文件(.msi)和一个客户端 Installer 服务（msiexec.exe)。Windows Installer 是运行于用户操作系统上的一个服务。这个服务使操作系统能够管理安装程序并使用包含在打包文件内的信息安装软件。

Msiexec.exe 程序是 Windows Installer 的一个组成部分。Msiexec.exe 用动态链接库 Msi.dll 读取打包文件（.msi)，应用转换程序（.mst)，并集成命令行选项。

有关 Windows Installer 的更多信息可以在 Windows Installer 软件开发工具包（SDK）中找到。网址为：

http://www.microsoft.com/msdownload/platformSDK/sdkupdate/

部署 ArcGIS Engine 运行时

ArcGIS Engine 运行时开发的应用程序需要将 ArcGIS Engine 运行时安装到终端用户的机器上。为确保 ArcGIS Engine 运行时安装到用户机器上，请按以下步骤操作：

用户直接运行 ArcGIS Engine 运行时安装程序

1. 用户可以从 ArcGIS Engine 运行时 CD 上启动安装程序。
2. 在安装前，用户应先检查 ArcGIS Engine 运行时是否已经安装在他们的机器上，方法为：
   I. 运行“控制面板”下的“添加/删除程序”。
   II. 检查 ArcGIS Engine 运行时是否在列在程序列表中。如果在列表中，则已经在这台机器上安装了 ArcGIS Engine 运行时。单击“支持信息”核实产品版本号是否为 9.0。
3. 如果在机器上安装了 ArcGIS Engine 运行时，用户可以继续安装开发人员开发的应用程序。
4. 如果启动了 ArcGIS Engine 运行时安装程序，且已经在这台机器上安装了 ArcGIS Engine 运行时，则安装将作为维护安装来执行。
5. 在继续安装前，用户必须检查这台机器上是否有低于 9.0 版本的 ArcGIS 产品，包括 ArcIMS、ArcGIS Desktop 、ArcGIS Workstation 和 ArcReader。检查这台机器上是否有这些产品的方法为：
   I. 运行“控制面板”下的“添加/删除程序”。

II. 检查程序列表以查看是否在这台机器上安装了任何 ArcGIS 产品。如果在列表中有 ArcGIS 产品，单击“支持信息”以确定产品的版本号。如果版本低于 9.0，在安装 ArcGIS Engine 运行时 9.0 之前应删除这些产品。

6. 如果 ArcGIS Engine 运行时没有在这台机器上安装，则在安装开发的应用程序之前，用户必须启动 ArcGIS Engine 运行时的 Setup.exe。

7. 如果应用程序需要 Java 或.NET 安装特性，用户必须检查是否安装了这些特性。 Java 和.NET 是可选的安装特性，在 ArcGIS Engine 运行时的典型安装中并没有安装这两个特性。要检查是否安装了这些特性，其方法为：

   I. 运行“控制面板”下的“添加/删除程序”。

   II. 检查 ArcGIS Engine 运行时是否列在程序列表中。如果在列表中，则已经在这台机器上安装了 ArcGIS Engine 运行时。

   III. 在 ArcGIS Engine 运行时特性上单击“更改”按钮，然后选择“修改”。“Select Feature”对话框显示了 ArcGIS Engine 运行时安装的目前状态。硬盘图标表明在机器上安装了该特性，红色的叉表明机器上没有安装该特性，但可以进行安装。

   IV. 要安装某个特性，用户必须选择该特性，右击并选择“安装在本地硬盘上”。

在应用程序的安装程序中包含 ArcGIS Engine 运行时安装程序

1. 开发人员可以用许多方法将 ArcGIS Engine 运行时安装程序包含在其开发的应用程序的安装程序中。部署 ArcGIS Engine 运行时的方法将在本章后面讨论。

2. 在应用程序的安装程序启动 ArcGIS Engine 运行时安装程序之前，开发人员要通过执行系统检查来确定是否已经在用户机器上安装了 ArcGIS Engine 运行时。执行系统检查的方法将在本章的后面讨论。

3. 如果机器上没有安装 ArcGIS Engine 运行时，开发人员通过使用下面讨论的部署选项中的一种来安装 ArcGIS Engine 运行时。

4. 如果机器上已经安装了 ArcGIS Engine 运行时，就不用再安装 ArcGIS Engine 运行时，可以继续安装应用程序。

5. 如果应用程序需要 Java 或.NET 安装特性，开发人员必须检查是否安装了这些特性（本章后面讨论）。Java 和.NET 是可选的安装特性，在 ArcGIS Engine 运行时的典型安装中并没有安装这两个特性。如果没有安装这些特性，而应用程序需要其中一个或两者都需要，开发人员必需安装这些所需的特性。

ArcGIS Engine 运行时安装特性

一个.msi 安装程序由若干特性组成。特性是用户可以安装的组件群。

ArcGIS Engine 运行时安装程序由下列安装特性组成：

特性	描述性特性名	描述
ArcEngine	ArcGIS Engine	ArcGIS Engine
JavaRuntime	ArcGIS Engine Java Runtime	Java Archives
DotNetRuntime	ArcGIS Engine .NET Runtime	.Net Assemblies

ArcGIS Engine 运行时的系统需求

Windows NT SP6a，Windows 2000，Windows XP 专业版和 Windows 2003 Server 操作系统都支持 ArcGIS Engine 运行时。

要了解有关ArcGIS Engine运行时的系统需求方面的其他信息或更新信息，请访问http://support.esri.com。

ArcGIS Engine 运行时安装位置

ArcGIS 9.0,ArcGIS Engine 运行时,ArcGIS Engine 开发工具包,ArcGIS Desktop，ArcReader 独立程序和 ArcGIS Server 将安装在相同的安装目录中。首次安装的 ArcGIS 9.0 产品将决定所有后续 ArcGIS 9.0 产品的安装位置。

注意：如果已经安装了 ArcIMS ArcMap Server 9.0，ArcGIS 9.0 产品将默认安装到 ArcIMS 安装目录中。

例如，如果 ArcGIS Desktop 安装到 C:\Desktop 目录中，ArcGIS 9.0 的安装位置将是 C:\Desktop\ArcGIS。如果接着安装 ArcGIS Engine 运行时，用户将没有机会浏览以选择安装位置。ArcGIS Desktop 的安装已经预先确定了所有 ArcGIS 9.0 产品的安装位置。因此，在本例中，ArcGIS Engine 运行时也将安装到 C:\Desktop\ArcGIS 目录中。

如果在安装 ArcGIS 9.0 产品时没有足够的磁盘空间，用户将需要卸载所有的 ArcGIS 9.0 产品（如前所列），然后重新将这些产品安装到有足够磁盘空间的目录中。ArcGIS 9.0 产品（不包括 ArcSDE，ArcIMS 和 ArcInfo Workstation）不能安装到其他目录中。

部署方法

ArcGIS Engine 运行时有两种推荐的部署方法。

1. 用户直接从 CD 中安装 ArcGIS Engine 运行时安装程序。ArcGIS Engine 运行时安装程序可以用 CD 进行重新分发。可以复制 ArcGIS Engine 运行时 CD 映像内容并创建其他 CD，或者联系 ESRI 以获得其他 ArcGIS Engine 运行时 CD。

从 CD 中直接运行安装程序的其他要求：

- 用户需要检查是否已经在其机器上安装了 ArcGIS Engine 运行时。如果已经安装，用户可以继续安装开发人员开发的应用程序。如果启动了 ArcGIS Engine 运行时安装程序，而且 ArcGIS Engine 运行时已经安装在机器上，则安装将作为维护安装来执行。

- ArcGIS Engine 运行时 9.0 不能在有低于 9.0 版本的 ArcGIS 产品的机器上安装。在安装 ArcGIS Engine 运行时 9.0 之前必须将低于 9.0 版本的 ArcGIS 产品卸载。
- 如果应用程序所需的任何 ArcGIS Engine 运行时特性没有安装，用户应安装这些特性。如果所需的 ArcGIS Engine 运行时特性已经在机器上安装，则不必运行 ArcGIS Engine 运行时安装程序。

要获得其他ArcGIS Engine运行时CD，请与www.esri.com上的“ESRI客户服务”联系，或者美国用户可以拨打 888-377-4574，或者与当地的ESRI分部联系。

2. 将 ArcGIS Engine 运行时安装程序集成到应用程序的安装程序中

可以通过运行使用 Windows Installer 命令行参数的安装程序来安装 ArcGIS Engine。

ArcGIS Engine 运行时可以使用下列选项集成到应用程序的安装程序中：

a. 在基于 msi 的安装程序的结尾

b. 在批处理文件内

c. 在脚本安装程序内

本章后面提供这些选项的示例。

集成 ArcGIS Engine 运行时安装程序的其他要求

如果应用程序所需的任何 ArcGIS Engine 运行时特性没有安装，开发人员制作的安装程序应添加这些特性。如果所需的 ArcGIS Engine 运行时特性已经在机器上安装，就不用运行 ArcGIS Engine 运行时的安装程序。

在安装 ArcGIS Engine 运行时的时候，会为它创建一个注册表项，其位置为：

HKEY_LOCAL_MACHINE\Software\ESRI\ArcGIS Engine Runtime

这个注册表键的 RealVersion 值是 9.0。

可以用这个键检查用户机器上是否安装了 ArcGIS Engine 运行时。

1. 为 ArcGIS Engine 运行时选项的安装特性执行系统检查

如果应用程序需要安装 ArcGIS Engine 运行时.NET 或 Java 特性，必须检查用户系统以查看是否有合适的.NET 或 Java 安装特性及 ArcGIS Engine 运行时（参见“为 ArcGIS Engine 运行时执行系统检查”）。

下面的注册表项将决定 ArcGIS Engine 运行时特性的安装状态：
[HKEY_CLASSES_ROOT\Installer\Features\7A1A3A9178A2BC74EB114EA6B5DB1C1B]
“Registry”=””
“ArcEngine”=””
“DotNetRuntime”=”ArcEngine” 或者 “?ArcEngine”
“JavaRuntime”=”ArcEngine” 或者 “?ArcEngine”

确定是否安装了.NET 特性的方法为：

.NET ArcGIS Engine 安装特性要求在机器上安装.NET Framework1.1。下面的注册表键可用于确定是否安装了.NET ArcGIS Engine 特性：
HKEY_CLASSES_ROOT\Installer\Features\7A1A3A9178A2BC74EB114EA6B5DB1C1B

DotNetRuntime 的值代表.NET ArcGIS Engine 安装特性。

如果这个注册表键下 DotNetRuntime 的值没有显示，则表明在运行 ArcGIS Engine 运行时安装程序时还没有在机器上安装.NET Framework1.1。

如果没有安装 DotNetRuntime 特性，则
“DotNetRuntime”=“?ArcEngine”

如果安装了 DotNetRuntime 特性，则
“DotNetRuntime”=“ArcEngine”

确定是否安装了 Java 特性的方法为：

下面的注册表键可用于确定是否安装了.Java ArcGIS Engine 特性：
HKEY_CLASSES_ROOT\Installer\Features\7A1A3A9178A2BC74EB114EA6B5DB1C1B

JavaRuntime 的值代表. Java ArcGIS Engine 安装特性。
如果没有安装 JavaRuntime 特性，则
“JavaRuntime”=“?ArcEngine”
如果安装了 JavaRuntime 特性，则
“JavaRuntime”=“ArcEngine”

如果发现所需的安装特性没有安装，则只需要在脚本中使用 ADDLOCAL Windows Installer 命令就可以安装。例如，ADDLOCAL=DotNetRuntime 或 ADDLOCAL=JavaRuntime。

如果选择只安装 DotNetRuntime 或 JavaRuntime 特性，而且 ArcEngine 特性（主要的 ArcGIS Engine 安装特性）没有安装，则安装程序在安装过程中会包括 ArcEngine 特性。

如果在机器上没有检测到.NET Framework1.1，则 DotNetRuntime 特性会被隐藏并且不会被安装。

在将 ArcGIS Engine 运行时安装程序集成到应用程序安装程序中时，需要考虑以下需求。

2. ArcGIS Engine 运行时安装程序使用 Windows Installer 2.0

- ArcGIS Engine 运行时安装程序使用 Windows Installer 2.0。在使用命令行参数运行 Engine 运行时安装程序之前，必须先在目标机器上安装并运行 Windows Installer 2.0 版本。检查机器上 Windows Installer 版本的方法为：

  i. 定位到 system32 文件夹中的 msiexec.exe。

  ii. 右击 msiexec.exe 并选择“属性”。

  iii. 在“属性”对话框中，选择“版本”选项卡检查该文件的版本。

- 如果将 ArcGIS Engine 运行时安装程序包含在一个不是基于 msi 的安装程序中，终端用户将需要在他们的机器上安装 Windows Installer 2.0。ArcGIS Engine 运行时安装程序使用 Windows Installer 技术。
- 如果在基于 msi 的安装程序的结尾启动 Engine 运行时安装程序，必须用 Windows Installer 2.0 或更高版本创建自己的 msi 安装程序，以兼容 ArcGIS Engine 运行时安装程序。
- ArcGIS Engine 运行时 msi 不能嵌套在另一个 msi 中。每个产品，包括 ArcGIS Engine 运行时，都必须单独列在“添加/删除程序”列表中。
- Windows Installer 2.0 可以从 <ArcGIS Engine 运行时 CD>\Support\MSI\instmsiw.exe 中获得。

3. 为 ArcGIS Engine 运行时执行系统检查

如果要将 ArcGIS Engine 运行时安装程序包含在应用程序的安装程序中，必须在用户机器上执行系统检查，以检测是否已经安装了 ArcGIS Engine 运行时。如果启动了 ArcGIS Engine 运行时安装程序，而且已经在机器上安装了 ArcGIS Engine 运行时，则安装将作为维护安装来执行。

示例脚本

下面是一个在批处理文件中实现的命令行参数示例，可用来安装 ArcGIS Engine 运行时.NET 特性：

```
REM ##############################
REM Set variables
SET MSI_PATH=\\CDROM\Setup.msi
REM ##############################
REM Launch MSI Silently - NO UI
msiexec.exe /i  “%MSI_PATH%”  /qn ADDLOCAL=DotNetRuntime
```

注释：在上例中，如果在机器上没有检测到.NET Framework 1.1，则不会安装 DotNetRuntime 特性。

下面是一个在批处理文件中实现的命令行参数示例，可用来安装 ArcGIS Engine 运行时 Java 特性：

```
REM ############################
REM Set variables
SET MSI_PATH=\\CDROM\Setup.msi

REM ############################

REM Launch MSI Silently - NO UI
msiexec.exe /i  “%MSI_PATH%”  /qn ADDLOCAL=JavaRuntime
```

部署选项示例

下面是 ArcGIS Engine 运行时部署选项的一些示例。

在基于 MSI 安装程序的结尾部署 ArcGIS Engine 运行时

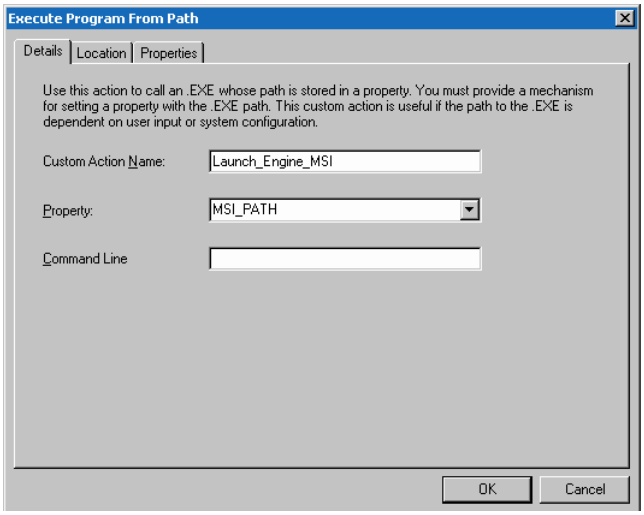
下面的示例使用 Wise for Windows Installer MSI Authoring 程序在另一个基于 MSI 的安装程序的结尾启动 ArcGIS Engine 运行时安装程序。在单击“结束（Finish）”按钮后启动 ArcGIS Engine 运行时安装程序。下面的示例说明了“结束（Finish）”按钮背后的自定义动作。

这个示例假设 ArcGIS Engine 运行时安装程序驻留在与应用程序的安装程序相同的目录中。在这种情况下，Setup.exe 驻留在一个名为“ArcEngine”的文件夹中。启动应用程序媒介上“ArcEngine”文件夹中的 ArcGIS Engine 运行时 Setup.exe 的方法为：

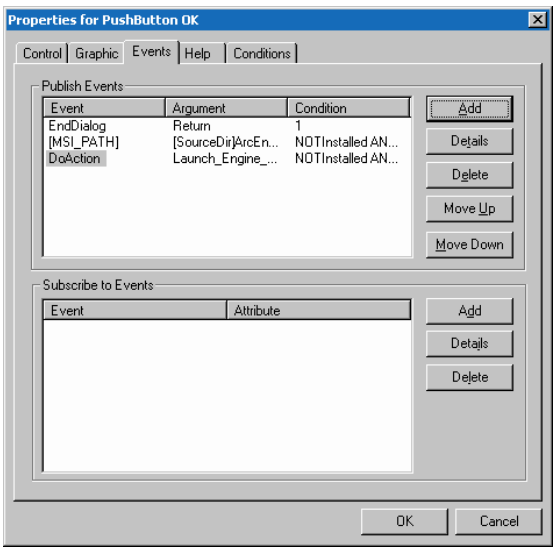
1. 创建 MSI_PATH 和 ArcEngineExists 属性，并在“属性”表中将它们的初始值设置为 1。

2. 在安装程序的开始处执行 ArcGIS Engine 运行时系统检查。系统检查应搜索下列注册表键：HKEY_LOCAL_MACHINE\Software\ESRI\ArcGIS Engine Runtime，而且如果返回的 RealVersion 值为 9.0，则将 ArcEngineExists 属性设置为“True”。

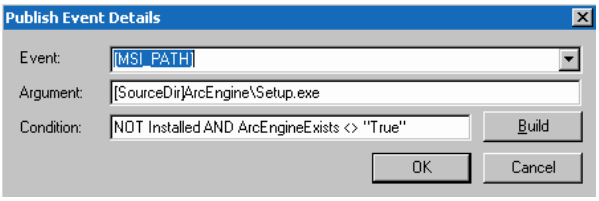
3. 使用“Execute Program from Path”自定义动作类型创建一个名为“Lauch_Engine_MSI”的自定义动作。将这个自定义动作中的“属性”设置为 MSI_PATH。在“属性”字段中指定的路径就是程序从中执行的路径，在这个例子中是 MSI_PATH。启动 setup.exe 时不需要命令行。



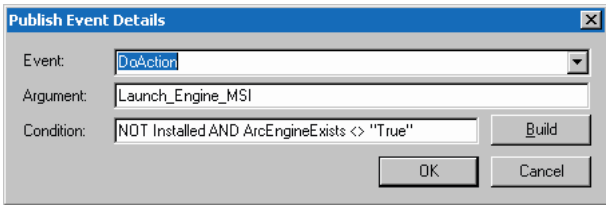
4. 在应用程序安装程序的“退出（Exit）”对话框中，为“结束（Finish）”按钮控件添加两个动作。



第一个动作将 MSI_PATH 属性设置为[SourceDir]ArcEngine\Setup.exe。其值随 ArcGIS Engine 运行时安装程序在媒介上的位置不同而变化。在这个例子中，Setup.exe 位于 CD 上的 ArcEngine 文件夹中。



第二个动作调用前面创建的 Launch_Engine_MSI。



只有在 ArcEngineExists 属性不等于 True 的情况下才能执行这两个自定义动作。

在脚本安装程序中部署 ArcGIS Engine 运行时

脚本安装程序可以用命令行参数来安装 ArcGIS Engine 运行时（下面的示例使用 Wise Install Master Setup Authoring 软件）：

注释：在下面的示例中，MSI_PATH 会随 ArcGIS Engine 运行时安装程序在媒介上的位置不同而变化。在这个例子中，Setup.msi 位于 CD 上的 ArcEngine 文件夹中。

```
Rem Set variable
Set Variable MSI_PATH to \\CDROM\ArcEngine\Setup.msi
Rem Launch ArcGIS Runtime setup program silently – No UI
Execute %SYS32%\msiexec.exe /i %MSI_PATH% /qn (Wait)
Rem Launch ArcGIS Runtime setup program silently – No UI except for
a modal
dialog box displayed at the end
Execute %SYS32%\msiexec.exe /i %MSI_PATH% /qn+ (Wait)
```

在批处理文件中部署 ArcGIS Engine 运行时

下面是在一个批处理文件中实现命令行参数的示例，可用来安装 ArcGIS Engine 运行时。

注释：在下面的示例中，MSI_PATH 会随 ArcGIS Engine 运行时安装程序在媒介上的位置不同而变化。在这个例子中，Setup.msi 位于 CD 上的 ArcEngine 文件夹中。

```
REM ############################
REM Set variables
SET MSI_PATH=\\CDROM\ArcEngine\Setup.msi
REM ############################
REM Launch MSI Silently - NO UI
msiexec.exe /i  “%MSI_PATH%”  /qn
REM Launch MSI Silently - Reduced UI
msiexec.exe /i  “%MSI_PATH%”  /qb
REM Launch MSI Silently - No UI except for a modal dialog box
displayed at
the end.
msiexec.exe /i  “%MSI_PATH%”  /qn+
```

从 ArcGIS Engine 运行时 CD 上启动安装程序

ArcGIS Engine 运行时安装程序可以从 CD 上用 setup.exe 手工启动。

1. 复制 ArcGIS Engine 运行时 CD 映像内容并创建其他 CD，或者将 ArcGIS Engine 运行时放在应用程序的安装程序 CD 中。
2. 通知用户下列问题：

   a.检查系统以确定是否在机器上已经安装了ArcGIS Engine运行时9.0。

   b.检查系统以确定机器上是否安装有低于9.0版本的ArcGIS产品。如果有，必须在安装 ArcGIS Engine 运行时 9.0 之前将它们卸载。

   c.如果已经安装了 ArcGIS Engine 运行时 9.0，则继续运行开发应用程序的安装程序。

   d.如果没有安装 ArcGIS Engine 运行时 9.0，从 ArcGIS Engine 运行时 CD 映像位置上启动 Setup.exe。一旦完成了 ArcGIS Engine 运行时的安装，就启动开发应用程序的安装程序。

如何使用安装程序的命令行参数

ArcGIS Engine 运行时安装程序可以用.msi 文件和客户端 Installer 服务（msiexec.exe）的命令行参数进行安装。下表列举了一些可用的 msiexec.exe 命令行参数。

（来源：http://msdn.microsoft.com/library/default.asp? url=/library/en-us/msi/setup/command_line_options.asp）

选项	参数	描述
/l	程序包|产品代码	安装或配置某个产品
/a	程序包	管理员安装选项。在网络上安装产品。
/x	程序包|产品代码	卸载某个产品。
/q	n|b|r|f	设置用户界面级别 q 无用户界面 qn 无用户界面 qb 基本用户界面。用 qb!隐藏“取消”按钮。 qr 简化用户界面，在安装的结尾不显示模态对话框。 qf 完整用户界面，而且在安装的结尾显示任何 FatalError，UserExit 或 Exit 模态对话框。 qn+ 无用户界面，在结尾显示一个模态对话框。 qb+ 基本用户界面，在结尾显示一个模态对话框。如果用户取消安装则不显示模态对话框。用 qb+!或 qb!+ 隐藏“取消”按钮。 qb- 基本用户界面，在结尾不显示模态对话框。请注意不支持/qb+-级别的用户界面。用 qb+!或 qb!+ 隐藏“取消”按钮。 *注意!选项在 Windows Installer 2.0 版本中可用，而且只能用在基本用户界面选项上，对完整用户界面选项无效。*
/?或/h		显示 Windows Installer 的版权信息。

要了解Windows Installer命令行参数方面的更多信息，请访问 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/command_line_options.asp。

示例：

用下面的命令执行无用户界面的典型安装并将程序安装到非默认位置上：

Msiexec.exe /i <setup location>\setup.msi /qn InstallDir=C:\Mysetup

用下面的命令执行基本用户界面（进展条）的典型安装并将程序安装到非默认位置上：

Msiexec.exe /i <setup location>/setup.msi /qb InstallDir=C:\Mysetup

用下面的命令执行无用户界面的完全安装并将程序安装到默认位置上：

Msiexec.exe /i <setup location>\setup.msi /qn ADDLOCAL=All

用下面的命令执行无用户界面的自定义安装并将程序安装到默认位置上：

Msiexec.exe /i <setup location>\setup.msi /qn
ADDLOCAL=<feature1>,<feature2>,....

需要安装的特性用 ADDLOCAL 参数指定。ArcGIS Engine 具有以下安装特性：

特性	描述性特性名	描述
ArcEngine	ArcGIS Engine	ArcGIS Engine
JavaRuntime	ArcGIS Engine Java Runtime	Java Assemblies
DotNetRuntime	ArcGIS Engine .Net Runtime	.Net Assemblies

用下面的命令执行无用户界面、包含 ArcGIS Engine 和.NET 安装特性的自定义安装：

Msiexec.exe /i <setup location>\setup.msi /qn
ADDLOCAL=ArcEngine,DotNetRuntime

部署方针

- 在卸载开发人员开发的应用程序的过程中，一定不要卸载 ArcGIS Engine 运行时。
- 只有在用户知道没有第三方应用程序使用 ArcGIS Engine 运行时的情况下，开发人员才必须推荐用户手工卸载 ArcGIS Engine 运行时。

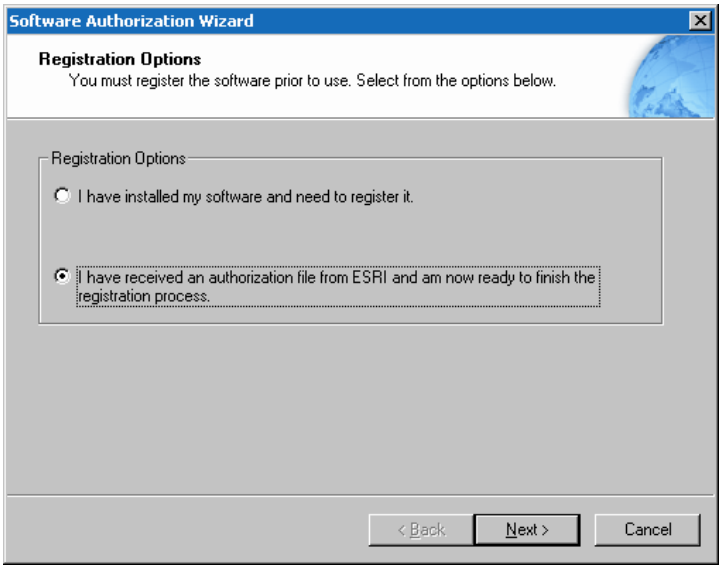
  注释：应该用控制面板卸载 ArcGIS Engine 运行时，而不是删除磁盘上的文件。
- ArcGIS Engine 运行时不能包括在制作好的 MSI 文件中（嵌套 MSI 安装）。
- 如果机器上已经安装了 ArcGIS Engine 运行时，则应用程序的安装程序不用启动 ArcGIS Engine 运行时的安装程序。
- 不能重新分发单个的 Engine 运行时文件；本章讨论的部署方法是部署 ArcGIS Engine 运行时文件的唯一方法。

为 ArcGIS Engine 应用程序授权

开发和部署 ArcGIS Engine 应用程序的最后一步就是确保所有的客户机器具有正确的许可配置以支持 ArcGIS Engine 应用程序。本节详细阐述终端用户和开发人员在客户系统上授权 ArcGIS Engine 运行时组件的各种方法。

必须将许可初始化内建到应用程序中。要了解更多信息，请参阅前面的“独立可执行许可的初始化”一节。

软件授权就是解除对底层 ArcGIS Engine 运行时软件组件的锁定的过程。开发人员在安装和设置 ArcGIS Engine 开发工具包时要进行软件授权。一旦安装好了该软件，就会打开一个软件授权向导。该向导要求用户定位注册产品时获得的授权文件（.ecp）。只有在读取并接受授权文件后，用户才可以设计和运行用 ArcGIS Engine 组件开发的应用程序。尽管有许多不同的 “授权”方法，但所有要部署的应用程序都必须以相似的方式授权。



安装 ArcGIS Engine 开发工具包后，就会打开软件授权向导。但是，ArcGIS Engine 运行时的安装程序不会触发软件授权向导自动启动。

如本章前面所述，用户建立和部署的每个应用程序都必须首先用合适的许可进行初始化。应用程序初始化所需的“合适”许可存储在客户机器或网络上任何可用的软件授权文件或关键码文件中。如果应用程序试图用一个不包含在授权文件中的许可初始化，或者如果所需许可的所有例程都被关闭，则应用程序不能运行。

开发人员必须预先考虑用户如何获得和访问运行应用程序所需的授权文件或关键码文件这个问题。开发人员开发的应用程序的客户可以分为以下三类：

- 具有许可的 ArcGIS Desktop 用户，可以访问应用程序使用的许可特性。
- 可以直接从 ESRI 获取 ArcGIS Engine 运行时软件和/或其授权的用户。
- 不与 ESRI 直接联系，而是从开发人员开发的应用程序中获得打包的 ArcGIS Engine 运行时软件和授权的用户。

下面讨论这三类用户的软件授权过程。

ArcGIS Desktop 用户

如果客户为具有许可的 ArcGIS Desktop 用户，开发人员及其客户安装和运行其建立的应用程序要经过以下步骤：

1. 开发人员检查并确认应用程序的许可需求—是 ArcView，ArcEditor 还是 ArcInfo，单用户还是多用户许可，是否需要任何扩展模块。
2. 终端用户确认自己拥有应用程序所需的 ArcGIS Desktop 授权或关键码文件。

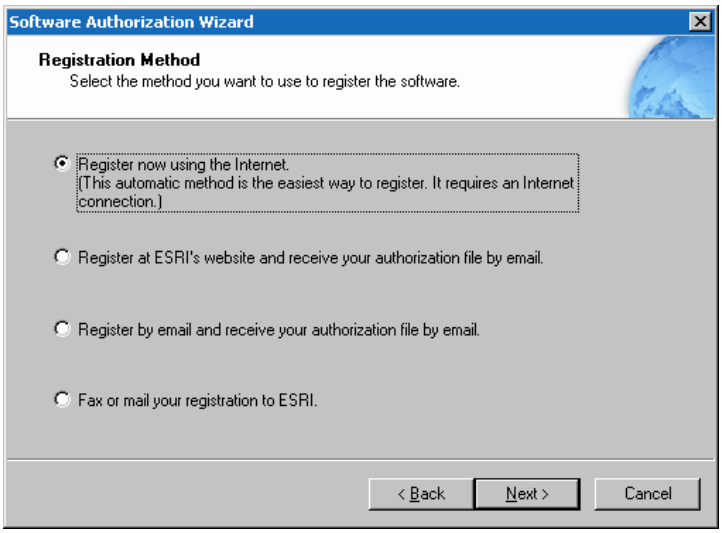
如果终端用户没有预先注册和接收授权文件，软件授权向导可以指导他们完成这个任务。

3. 安装自定义 ArcGIS Engine 应用程序。
4. 应用程序启动后就会立即初始化并从客户端已有的授权文件或关键码文件中检出一个可用的许可。

从 ESRI 获得 ArcGIS 运行时许可的终端用户

第二类终端用户亲自购买和/或授权 ArcGIS Engine 运行时软件。开发人员及其客户安装和运行其建立的应用程序要经过以下步骤：

1. 开发人员检查并确认应用程序所需的许可。
2. 终端用户购买 ArcGIS Engine 运行时和任何需要的选项（3D，地理数据库，空间分析，StreetMap 等）。
3. 终端用户到 ESRI（http://www.service.esri.com）注册 ArcGIS Engine 产品及所需的选项。



4. 终端用户从 ESRI 接收一个授权文件（.ecp）并将其保存到自己的机器上。
5. 终端用户安装 ArcGIS Engine 运行时软件。
6. 一旦完成安装，终端用户就可以定位到\ArcGIS\bin 文件夹并运行其中的 SoftwareAuthorization.exe 文件。
7. 当软件授权向导要求输入授权文件时，终端用户定位授权文件的位置。
8. 开发人员或其客户安装自定义 ArcGIS Engine 应用程序。
9. 应用程序启动后就会立即初始化并从客户端的授权文件中检出一个可用的许可。

虽然在这里记录了授权文件在应用程序中的重新分发，但这种重新分发的使用是有限制的。

- *如果应用程序只在组织内部使用，则可以用这种方式重新分发。但是，重新分发的数目不能超过用户购买的许可个数。*
- *如果应用程序要买给第三方使用，使用“重新分发”的授权文件将违反“标准 ESRI 软件使用许可协议（standard ESRI Software Master License Agreement）”，必须签订一个单独的协议。联系 ESRI 商业伙伴或国际销售商以了解有关这种许可的信息。*

要了解其他有关使用和部署 ArcGIS Engine 应用程序方面的信息，请参阅“ESRI 软件使用许可协议”的脚注 12 和 23。

从应用程序的安装程序中运行 SoftwareAuthorization 工具可以使用以下参量：

/LIF <filename> /S

/S 参量可以触发工具使其不显示用户界面。

参阅开发帮助以了解更多有关 IauthorizeLicense 接口方面的信息。

不与 ESRI 直接打交道的终端用户

应用程序调用包含在安装程序或应用程序中的 SoftwareAuthorization.exe 或 IauthorizeLicense 对象，以解锁 ArcGIS Engine 的功能。这要求开发人员将授权关键码“硬编码”到应用程序中。这种方法的优点就是软件可以无声的授权，而不用提示用户任何有关注册方面的信息。在这种情况下，开发人员及其客户安装和运行其建立的应用程序要经过以下步骤：

1. 开发人员检查并确认应用程序所需的许可。
2. 开发人员购买所需的可重新分发的 ArcGIS Engine 运行时产品及任何所需的选项（3D，地理数据库，空间分析，StreetMap 等）。
3. 开发人员到 ESRI（http://www.service.esri.com）注册 ArcGIS Engine 产品及所需的选项。
4. 开发人员接收一个可重新分发的授权文件（.ecp）并将其特性添加到应用程序的编码中。
5. 终端用户安装自定义建立的 ArcGIS Engine 应用程序。这样将：
   a. 安装 ArcGIS Engine 运行时软件；
   b. 自动运行\ArcGIS\bin\目录下的 SoftwareAuthorization.exe 或使用 IauthorizeLicense 对象。
6. 应用程序启动后就会立即初始化并从客户端的授权文件中检出一个可用的许可。

6 开发情景

本书已介绍了一些编程概念和模式以及一些新的APIs。本章将通过一些应用程序开发情景来应用这些概念。每个情景都使用ArcGIS Engine中可用的工具和APIs建立和部署了一个应用程序。每个情景都是ArcGIS Engine开发工具包中的一个ArcGIS开发示例，其完整代码可以从该工具包中获得。

本章的开发情景包括：

●用ActiveX建立应用程序 ● 用Visual JavaBeans建立应用程序 ● 用Windows控件建立应用程序 ● 建立命令行Java应用程序 ● 建立命令行C++应用程序

用 ActiveX 建立应用程序

本情景针对那些要用 ActiveX 建立和部署应用程序的开发人员，描述了使用 ArcGIS 控件建立和部署应用程序的过程。

用户可以不按照本情景进行，而可以从示例安装路径获得完整的应用程序。示例作为 ArcGIS 开发示例的组成部分而安装。

可以在 ArcGIS 安装目录下的\DeveloperKit\Samples\Developer_Guide_Scenarios\ArcGIS_Engine\Building_an_ArcGIS_Control_Application\Map_Viewer 中找到本示例。

ArcGIS 开发示例不包含在 ArcGIS Engine 开发工具包的“典型”安装中。如果没有安装这些开发示例，可以重新运行开发工具包安装向导，选择“自定义”或“修改”并选择软件开发工具包下的示例特性。

工程描述

本情景的目标就是向开发人员展示并使其熟悉在 COM API 中使用 ArcGIS 控件开发和部署 GIS 应用程序所需的步骤。本情景在 Microsoft Visual Basic 6.0 开发环境中使用 MapControl、PageLayoutControl、TOCControl 和 ToolbarControl 等 ActiveX 控件。使用 C++、Java 和.NET 的开发人员可以参考本章后面的其他情景：“建立命令行 C++应用程序”、“用 Visual JavaBeans 建立应用程序”、“建立命令行 Java 应用程序”和“用 Windows 控件建立应用程序”。

本情景展示了创建用于浏览用 ArcGIS Desktop 应用程序 ArcMap 预制作的地图文档的 GIS 应用程序所需的步骤。本情景涵盖下列技术：

- 在 Microsoft Visual Basic 6.0 中装载和嵌入 ArcGIS 控件。
- 将预制作的地图文档装载到 MapControl 和 PageLayoutControl 中。
- 设置 ToolbarControl 和 TOCControl 伙伴控件。
- 处理窗体调整大小事件。
- 向 ToolbarControl 添加控件命令和工具。
- 创建快捷菜单。
- 管理 TOCControl 中的标签编辑。
- 在 MapControl 上绘制几何形状。
- 创建自定义工具以操作 MapControl、PageLayoutControl 和 ToolbarControl。
- 定制 ToolbarControl。
- 将应用程序部署到 Windows 操作系统上。

概念

ActiveX 是 Microsoft 组件对象模型（COM）对象的另一个术语。所有的 ArcObjects 都基于 COM 之上，而且 ArcGIS 控件都是 COM 对象。

本情景用 Microsoft Visual Basic 6.0 开发环境实现，并使用作为 ActiveX 组件的 ArcGIS 控件。ActiveX 指的是一套能使以不同语言编写的软件组件在网络环境中协同工作的技术。每个 ActiveX ArcGIS 控件都有事件、属性和方法，一旦控件被嵌入到象 VB 窗体等 ActiveX 容器中就可以访问其事件、属性和方法。这些控件中的对象和功能可以与其它的 ESRI ArcObjects 及自定义控件结合以创建定制的终端用户应用程序。

本情景可以用任何其他完全支持ActiveX的COM开发环境编写，包括Microsoft Visual C++、 Borland Delphi、Sybase PowerBuilder和Microsoft Visual Basic for Applications (VBA)。虽然Visual Basic不能提供诸如Visual C++这样的开发环境的全部功能，但由于其受更广大用户的青睐，本情景选择Visual Basic作为示例代码的编写语言。不管使用何种开发环境，使用ArcGIS控件进行应用程序开发的成功与否取决于开发人员所掌握的编程环境和ArcObjects的技能。

本情景使用 MapControl、PageLayoutControl、TOCControl 和 ToolbarControl 以提供应用程序的用户界面。开发人员同时使用 ArcGIS 控件和其它 ArcObjects 及控件命令来创建 GIS 浏览应用程序。

设计

本情景的设计目的在于：首先强调 ArcGIS 控件之间的相互作用，其次向开发人员展示各个 ArcGIS 控件对象模型的部分内容。

每个 ActiveX ArcGIS 控件都有一系列属性页，一旦控件被嵌入到 ActiveX 容器中就可以访问这些属性页。这些属性页提供了控件属性和方法选择的快捷方式，并可以使开发人员不用写任何代码就建立应用程序。本情景不使用属性页，而是通过程序编写建立应用程序。要了解有关属性页方面的更多信息，请参考“ArcGIS 开发帮助”。

需求

要成功地按照本情景进行开发，需要下列条件（部署需求将在后面的“部署”一节中阐述）：

- 安装了 ArcGIS Engine 开发工具包，且具有能用于开发的许可文件。
- 安装了 Microsoft Visual Basic 6.0 开发环境并有合适的许可。
- 熟悉 Microsoft Windows 操作系统并具有使用 Microsoft Visual Basic 6.0 的经验。虽然本情景提供了一些有关如何在 Microsoft Visual Basic 6.0 中使用 ArcGIS 控件方面的信息，但并不能替代该开发环境的培训。
- 尽管不需要有其他 ESRI 软件的经验，但具有 ArcObjects 经验并对诸如 ArcMap 和 ArcCatalog 等 ArcGIS 应用程序具有基本的理解是有益的。
- 可以访问本情景的示例数据和代码，其位置为安装目录下的：

\DeveloperKit\Samples\Developer_Guide_Scenarios\ArcGIS_Engine\Building_an_ArcGIS_Control_Application\Map_Viewer

ArcGIS 开发示例不包含在 ArcGIS Engine 开发工具包的“典型”安装中。如果没有安装这些开发示例，可以重新运行开发工具包安装向导，选择“自定义”或“修改”并选择软件开发工具包下的示例特性。

本情景所使用的控件和类库包括：

- MapControl
- PageLayoutControl
- Carto Object Library
- Display Object Library
- Geometry Object Library
- TOCControl
- ToolbarControl
- System Object Library
- SystemUI Object Library

在 Visual Basic 中，这些控件和类库的名称都带有前缀“esri”。

实现

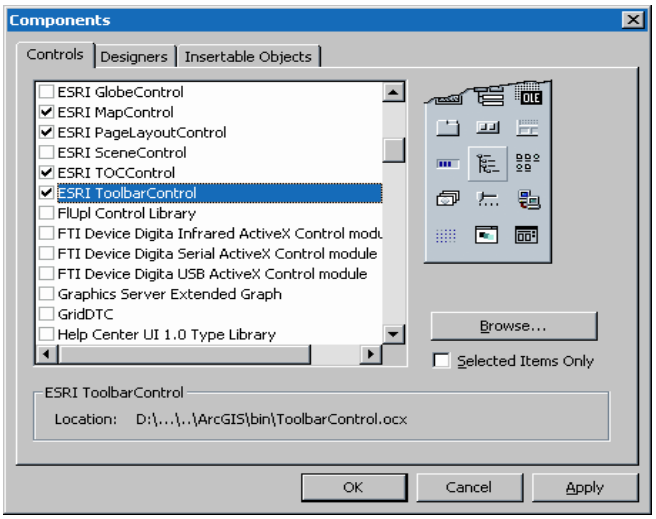
下面的实现过程提供了成功完成本情景所需的所有代码。本情景没有提供在 Visual Basic 6.0 中开发应用程序的逐步指导，因为我们假设用户已经具有 Visual Basic 6.0 开发环境的使用经验。

装载 ArcGIS 控件

在开始编写应用程序前，应用程序要使用的 ArcGIS 控件和其他 ArcGIS Engine 类库参考应装载到开发环境中。

1. 启动 Visual Basic 并从“New project”对话框中创建一个新的“标准 EXE”工程。
2. 单击“Project”菜单并选择“Components”。
3. 在“Components”对话框中，选上“ESRI MapControl”、“ESRI PageLayoutControl”、“ESRI TOCControl”和“ESRI ToolbarControl”。单击 OK 按钮。

可以用“ESRI 自动引用” Visual Basic 插件来快速选择和引用在 Visual Basic 6.0 中经常使用的 ArcGIS 控件和其他 ArcGIS Engine 类库。选择“Add-Ins”菜单中的“Add-In 管理器”并复选装载行为复选框来装载该插件。然后从“Add-Ins”菜单中选择“ESRI 自动引用”以显示该插件。



现在这些控件就会显现在 Visual Basic 的工具箱中。

 ESRIMapControl

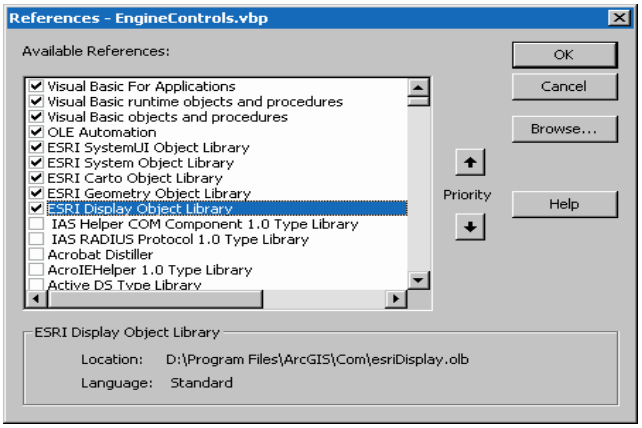
 ESRIPageLayoutControl

 ESRITOCControl

 ESRIToolbarControl

4. 再次单击“Project”菜单并选择“References”。

5. 在“References”对话框中，选上“ESRI Carto Object Library”、“ESRI Display Object Library”、“ESRI Geometry Object Library”、“ESRI System Object Library”和“ESRI SystemUI Object Library”。单击 OK 按钮。



将 ArcGIS 控件嵌入到容器中

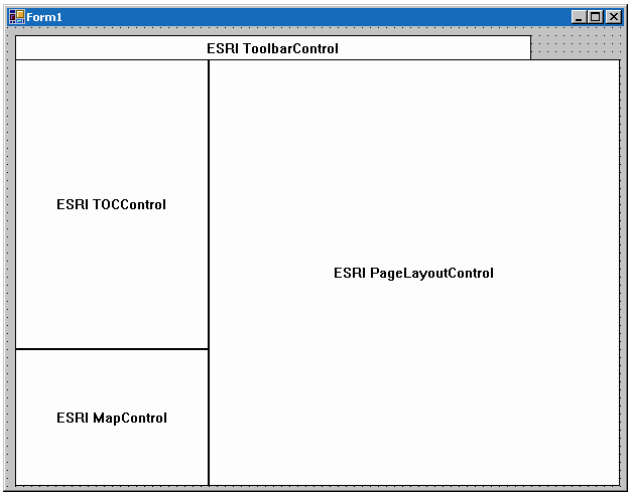
在访问控件的属性、方法和事件之前，要将各个控件嵌入到 ActiveX 容器中。一旦控件被嵌入到窗体中，就可以塑造应用程序的用户界面。

1. 打开 Visual Basic 窗体。

2. 双击 Visual Basic 工具箱中的 MapControl 图标以将 MapControl 添加到窗体上。

3. 重复步骤 2 以将 PageLayoutControl、TOCControl 和 ToolbarControl 添加到窗体中。

4. 如下图所示改变窗体上的各个控件的大小和位置。



将地图文档装载到 PageLayoutControl 和 MapControl 中

单独的数据层或用 ArcGIS Desktop 应用程序 ArcMap 预制作的地图文档可以装载到 MapControl 和 PageLayoutControl 中。可以装载 ESRI 提供的示例地图文档或用户自己的地图文档。可以使用添加一个对话框以浏览选择地图文档。

1. 双击窗体，显示代码窗口。
2. 选择“Form_Load”事件并输入下列代码（如果使用用户自己的地图文档，则用该地图文档的名称替换文件名）。

```
Private Sub Form_Load()

'Check and load a pre-authored map document into the
PageLayoutControl
using relative paths.
Dim sFileName As String
sFileName =
"..\..\..\..\..\..\Data\ArcGIS_Engine_Developer_Guide\Gulf
of St. Lawrence.mxd"
If PageLayoutControl1.CheckMxFile(sFileName) Then
PageLayoutControl1.LoadMxFile sFileName
End If

End Sub
```

3. 选择“PageLayoutControl_OnPageLayoutReplaced”事件并输入下列代码，将同一个地图文档装载到MapControl中。当文档装载到PageLayoutControl中时会触发OnPageLayoutReplaced事件。

```
Private Sub PageLayoutControl1_OnPageLayoutReplaced(ByVal
newPageLayoutAs Variant)

'Load the same pre-authored map document into the MapControl.
MapControl1.LoadMxFile PageLayoutControl1.DocumentFilename
'Set the extent of the MapControl to the full extent of the data.
MapControl1.Extent = MapControl1.FullExtent

End Sub
```

设置 TOCControl 和 ToolbarControl 的伙伴控件

由于这个应用程序的缘故，TOCControl 和 ToolbarControl 将与 PageLayoutControl 而不是 MapControl 协同工作。为此必须将 PageLayoutControl 设置为伙伴控件。TOCControl 使用伙伴控件的活动视图（ActiveView）以存放其地图、图层和符号体系，而 ToolbarControl 上的所有命令、工具或菜单项均会与伙伴控件的显示交互。

1. 双击窗体，显示代码窗口。

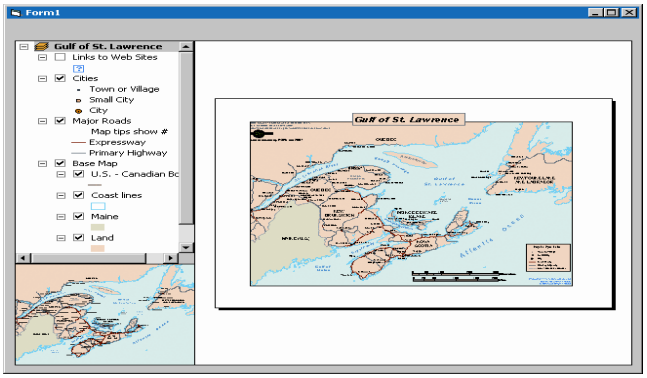
2. 选择“Form_Load”事件并在装载文档后输入下列代码：

```
Private Sub Form_Load()

'Check and load a pre-authored map document into the
PageLayoutControl
using relative paths.
Dim sFileName As String
sFileName =
"..\..\..\..\..\..\Data\ArcGIS_Engine_Developer_Guide\Gulf
of St. Lawrence.mxd"
If PageLayoutControl1.CheckMxFile(sFileName) Then
PageLayoutControl1.LoadMxFile sFileName
End If
'Set buddy controls.
TOCControl1.SetBuddyControl PageLayoutControl1
ToolbarControl1.SetBuddyControl PageLayoutControl1

End Sub
```

3. 运行应用程序。地图文档已经被装载到 PageLayoutControl 中，而 TOCControl 列出了该地图文档中的数据层。使用通过选择和取消选择 TOCControl 的可见性复选框来打开和关闭图层的可见性。默认情况下，地图文档中的活动地图被装载到 MapControl 中。此时 ToolbarControl 是空的，因为还没有向其添加命令。试着改变窗体的大小，会注意到控件的大小并没有改变。



处理窗体调整大小事件

在运行时调整窗体大小并不会使 PageLayoutControl 和 MapControl 自动调整大小。为了调整控件大小以便它们总是填充满窗体范围，开发人员必须对 Form_Resize 事件作出响应。如果 PageLayoutControl 或 MapControl 包含大量数据，在 Form_Resize 过程中重绘这些数据的代价很高。为提高性能，开发人员可以禁止数据重绘，直到调整大小完成。在调整大小过程中，将绘制一个被拉伸的位图。

1. 双击窗体，显示代码窗口。
2. 选择“Form_Resize”事件并输入下列代码：

```
Private Sub Form_Resize()

Dim dWidth As Double, dheight As Double, dMargin As Double

'Set the margin size.
dMargin = TOCControl1.Left

'Resize the PageLayoutControl.
dheight = Form1.ScaleHeight - PageLayoutControl1.Top - dMargin
If dheight > 0 Then PageLayoutControl1.Height = dheight
dWidth = Form1.ScaleWidth - TOCControl1.Width - (dMargin * 2)
If dWidth > 0 Then PageLayoutControl1.Width = dWidth

'Resize the MapControl.
dheight = Form1.ScaleHeight - MapControl1.Top - dMargin
If dheight > 0 Then MapControl1.Height = dheight

End Sub
```

3. 选择“Form_Load”事件并在该程序的末尾添加下列代码：

```
Private Sub Form_Load()

'Set buddy controls…

'Suppress drawing while resizing
MapControl1.SuppressResizeDrawing False, Form1.hWnd
PageLayoutControl1.SuppressResizeDrawing False, Form1.hWnd

End Sub
```

4. 运行应用程序并试着调整窗体大小。

向 ToolbarControl 添加命令

ArcGIS Engine 带有 120 多个直接操作 MapControl、PageLayoutControl 和 ToolbarControl 的命令和工具。这些命令和工具为用户提供了许多经常使用的地图导航、图形管理和要素选择等 GIS 功能。现在开发人员将向应用程序添加部分这些命令和工具。

1. 双击窗体，显示代码窗口。
2. 选择“Form_Load”事件并在装载文档代码前输入下列代码：

```
Private Sub Form_Load()

Dim sProgID As String

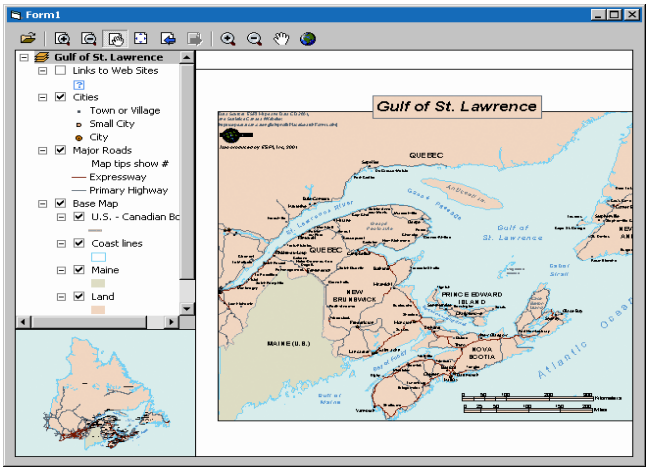
'Add generic commands.
```

```
sProgID = "esriControlTools.ControlsOpenDocCommand"
ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
'Add PageLayout navigation commands.
sProgID = "esriControlTools.ControlsPageZoomInTool"
ToolbarControl1.AddItem sProgID, , , True, , esriCommandStyleIconOnly
sProgID = "esriControlTools.ControlsPageZoomOutTool"
ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
sProgID = "esriControlTools.ControlsPagePanTool"
ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
sProgID = "esriControlTools.ControlsPageZoomWholePageCommand"
ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
sProgID = "esriControlTools.ControlsPageZoomPageToLast
ExtentBackCommand"
ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
sProgID = "esriControlTools.ControlsPageZoomPageToLastExten
tForwardCommand"
ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
'Add Map naviagtion commands.
sProgID = "esriControlTools.ControlsMapZoomInTool"
ToolbarControl1.AddItem sProgID, , , True, , esriCommandStyleIconOnly
sProgID = "esriControlTools.ControlsMapZoomOutTool"
ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
sProgID = "esriControlTools.ControlsMapPanTool"
ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
sProgID = "esriControlTools.ControlsMapFullExtentCommand"
ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly

'Load a pre-authored…

End Sub
```

3. 运行应用程序。ToolbarControl 现在包含 ArcGIS Engine 命令和工具，用户可以用这些命令和工具导航装载到 PageLayoutControl 中的地图文档。用“页面布局”命令集导航页面布局，用“地图”命令集导航数据框中的数据。用“open document”命令浏览和装载其他地图文档。



为 PageLayoutControl 创建快捷菜单

与前一个步骤中向 ToolbarControl 添加控件命令以便操作伙伴控件一样，开发人员也可以为控件命令集创建快捷菜单。下面将给应用程序添加一个操作 PageLayoutControl 的快捷菜单。每当在 PageLayoutControl 显示区域上单击鼠标右键时都会显示快捷菜单。

1. 在窗体的一般声明区添加下列代码：

```
Option Explicit

Private m_pToolbarMenu As IToolbarMenu 'The popup menu
```

2. 在 Form_Load 事件中向 ToolbarControl 添加命令代码段之后和装载文档代码之前添加下列代码：

```
Private Sub Form_Load()

'Add Map naviagtion commands…

'Create a new ToolbarMenu.
Set m_pToolbarMenu = New ToolbarMenu
'Share the ToolbarControl’s command pool.
Set m_pToolbarMenu.CommandPool = ToolbarControl1.CommandPool
'Add commands to the ToolbarMenu.
sProgID = "esriControlTools.ControlsPageZoomInFixedCommand"
m_pToolbarMenu.AddItem sProgID, , , False,
esriCommandStyleIconAndText
sProgID = "esriControlTools.ControlsPageZoomOutFixedCommand"
m_pToolbarMenu.AddItem sProgID, , , False,
esriCommandStyleIconAndText
sProgID = "esriControlTools.ControlsPageZoomWholePageCommand"
m_pToolbarMenu.AddItem sProgID, , , False,
esriCommandStyleIconAndText
sProgID = "esriControlTools.ControlsPageZoomPageToLastExtent
BackCommand"
m_pToolbarMenu.AddItemsProgID, , ,True,
esriCommandStyleIconAndText
sProgID = "esriControlTools.ControlsPageZoomPageToLastExtent
ForwardCommand"
m_pToolbarMenu.AddItem sProgID, , , False,
esriCommandStyleIconAndText
'Set the hook to the PageLayoutControl.
m_pToolbarMenu.SetHook PageLayoutControl1

'Load a pre-authored…

End Sub
```

3. 给PageLayoutControl1_OnMouseDown事件添加下列代码：

```
Private Sub PageLayoutControl1_OnMouseDown(ByVal button As Long,
ByValshift As Long, ByVal x As Long, ByVal y As Long, ByVal pageX
AsDouble, ByVal pageY As Double)

'Popup the ToolbarMenu
If button = vbRightButton Then
m_pToolbarMenu.PopupMenu x, y, PageLayoutControl1.hWnd
End If

End Sub
```